AppSense

You are the technology

AppSense

Server Configuration Utility

Scripting Guide

Management Server 8.6 and Personalization Server 8.5

Contents

Introduction

This guide assists AppSense administrators to automate the configuration of AppSense product servers (such as AppSense Management Server and Personalization Server) using Windows PowerShell.

AppSense product servers are shipped with SCU (Server Configuration Utility) cmdlets – a set of Windows PowerShell cmdlets that allow administrators to easily automate tasks such as:

- Initial configuration of a server
- Creating databases or exporting database scripts
- Starting or stopping a server
- Checking the operation of a server
- Enabling or disable logging

Note

When following examples in this document, enter only the part after the Windows PowerShell prompt. For example, if the example shows:

```
PS C:\> $credential = Get-Credential
```

Enter the following in the Windows PowerShell console:

```
$credential = Get-Credential
```

Limitations

The following limitations apply to the SCU cmdlets:

- Simultaneous use of the Server Configuration tools included with AppSense product servers and the SCU cmdlets is not supported.
- Logging cannot be enabled or disabled for specific services or website directories. To configure logging for specific components only use the Server Configuration tool included with the AppSense product server.

Prerequisites

Installing Windows PowerShell

SCU cmdlets require Windows PowerShell 3.0 to be installed before use. To install Windows PowerShell:

- Windows Server 2012 or 2012 R2 – Windows PowerShell is installed as standard on Windows Server 2012
- Windows Server 2008 SP2 or 2008 R2 SP1 – Download Windows Management Framework 3.0 from the Microsoft Download Center:

  http://www.microsoft.com/en-gb/download/details.aspx?id=34595

Note

For more information refer to “Installing Windows PowerShell” on Microsoft TechNet:
http://technet.microsoft.com/en-us/library/hh847837.aspx

Starting Windows PowerShell

Use either of the following methods to launch a Windows PowerShell console:

- Click **Start**, type PowerShell. Right click Windows PowerShell and select **Run as administrator**
- On the taskbar, right click Windows PowerShell and select Run as administrator

Note

For more information refer to “Starting Windows PowerShell on Windows Server” on Microsoft TechNet:
http://technet.microsoft.com/en-us/library/hh847814.aspx

Checking the Version of Windows PowerShell

The $PSVersionTable variable should be used to confirm that either PSVersion is 3.0 or later, or PSCompatibleVersions contains 3.0.

```
PS C:\> $PSVersionTable

Name                           Value
----                           -----
PSVersion                      3.0
WSManStackVersion              3.0
SerializationVersion           1.1.0.1
CLRVersion                     4.0.30319.18047
BuildVersion                   6.2.9200.16481
PSCompatibleVersions           {1.0, 2.0, 3.0}
PSRemotingProtocolVersion      2.2
```

Getting Help

Help Available in Windows PowerShell

Help for all SCU cmdlets are available in Windows PowerShell using the Get-Help cmdlet:

```
PS C:\> Get-Help Get-ApsInstance

NAME
    Get-ApsInstance

SYNOPSIS
    Gets the product instances that are installed.
...
```

Examples and further details can be obtained using the -full parameter:

```
PS C:\> Get-Help Get-ApsInstance -Full
```

Listing AppSense Modules and Cmdlets

To list modules:

```
PS C:\> Get-Module -Name AppSense* | Format-List

Name              : AppSense.ServerConfiguration.PowerShell
Path              : C:\Program Files\AppSense\Environment Manager\Personalizatio
                    n Server\bin\AppSense.ServerConfiguration.PowerShell.dll
Description       :
ModuleType        : Binary
Version           : 13.0.0.0
NestedModules     : {}
ExportedFunctions :
ExportedCmdlets   : {Get-ApsDatabaseVersion, Disable-ApsLogging,
                    Export-ApsDatabaseScript, Export-ApsServerImage...}

...
```

To list cmdlets:

```
PS C:\> Get-Command -Module AppSense*

CommandType     Name                                               ModuleName
-----------     ----                                               ----------
Cmdlet          Disable-ApsLogging                                 AppSense....
Cmdlet          Enable-ApsLogging                                  AppSense....
Cmdlet          Export-ApsDatabaseScript                           AppSense....
Cmdlet          Export-ApsServerImage                              AppSense....
Cmdlet          Get-ApsCurrentInstance                             AppSense....
Cmdlet          Get-ApsDatabaseScript                              AppSense....
Cmdlet          Get-ApsDatabaseVersion                             AppSense....
Cmdlet          Get-ApsInstance                                    AppSenseI...
Cmdlet          Get-ApsPrerequisite                                AppSense....
Cmdlet          Get-ApsVariance                                    AppSense....
Cmdlet          Import-ApsInstanceModule                           AppSenseI...
Cmdlet          Initialize-ApsDatabase                             AppSense....
Cmdlet          Initialize-ApsServer                               AppSense....
Cmdlet          Install-ApsPrerequisite                            AppSense....
Cmdlet          Repair-ApsVariance                                 AppSense....
Cmdlet          Reset-ApsServer                                    AppSense....
Cmdlet          Set-ApsServerDatabase                              AppSense....
Cmdlet          Start-ApsServer                                    AppSense....
Cmdlet          Stop-ApsServer                                     AppSense....
```

Further Resources

For more information about Windows PowerShell refer to the following resources on Microsoft TechNet:

- “Getting Started with Windows PowerShell”:

  http://technet.microsoft.com/en-us/library/hh857337.aspx

- “Windows PowerShell Core About Topics”:

  http://technet.microsoft.com/en-us/library/hh847856.aspx

Product Instances

Multiple AppSense product servers can be installed simultaneously on a machine – for example a machine can run up to 17 instances of the AppSense Management Server and up to 17 instances of the AppSense Personalization Server. Each different product server is referred to as an instance.

To allow administrators to manage different instances on the same machine a set of cmdlets known as AppSenseInstances is installed by AppSense product servers. You can use AppSenseInstances to:

- List installed product instances
- Load the appropriate set of SCU cmdlets for a product instance

Windows PowerShell automatically loads the AppSenseInstances cmdlets for you when launched – you don't need to load these cmdlets manually.

Note

You can only load the SCU cmdlets for one product instance into each Windows PowerShell session. If you need to work with multiple instances in the same script refer to Scripting Multiple/Remote Servers.

Listing available Product Instances

Use the Get-ApsInstance cmdlet to list available product instances. For example:

```
PS C:\> Get-ApsInstance

Name                : DEFAULT
InstanceId          : f321d748-8600-4cd1-beb9-8f1fc6dd0b85
ProductCode         : 54613b3a-5325-4fc9-8b4e-0b36f4a2863d
OriginalUpgradeCode : f321d748-8600-4cd1-beb9-8f1fc6dd0b85
OriginalProductCode : 54613b3a-5325-4fc9-8b4e-0b36f4a2863d
Version             : 8.5.650.0
InstallPath         : C:\Program Files\AppSense\Environment Manager\Personalizat
                      ion Server\bin\
CmdletPath          : C:\Program Files\AppSense\Environment Manager\Personalizat
                      Ion Server\bin\AppSense.ServerConfiguration.PowerShell.dll
                      ;C:\Program Files\AppSense\Environment Manager\Personaliza
                      tion Server\API\Modules\EMPImportExport
ProductName         : AppSense Personalization Server 8.5
IsDefault           : True
Status              : Unknown
DisplayName         : AppSense Personalization Server 8.5 (DEFAULT)
```

You can also specify filter parameters including:

- -InstanceId –The GUID that uniquely identifies this product instance on the machine. For example, f321d748-8600-4cd1-beb9-8f1fc6dd0b85 would identify the above product instance.
- -Name –The full name associated with a product instance on the machine. This parameter is case insensitive. The default instance is always named "DEFAULT".
- -ProductName – The part of the product name associated with a product instance on the machine. This parameter is case sensitive.
- -IsDefault – Returns details on the default instance.

Loading SCU cmdlets for a Product Instance

Before using the SCU cmdlets, you must specify the instance you want to manage using the Import-ApsInstanceModule cmdlets. For example:

```
PS C:\> Import-ApsInstanceModule -ProductName "Personalization Server" -IsDefault
```

You can specify:

- No parameters – loads the SCU cmdlets for the only product instance installed on the machine as long as there is only a single product instance installed
- -InstanceId – loads the SCU cmdlets for the specified product instance
- Any combination of -Name, -ProductName loads the SCU cmdlets for the matched product instance as long as there is only a single product instance installed that matches the specified parameters.

The following example loads the cmdlets for a Personalization Server named ‘Instance1’:

```
PS C:\> Import-ApsInstanceModule -ProductName "Personalization Server" –Name Instance1
```

Viewing the currently loaded Product Instance

After loading cmdlets for a product instance you can use the Get-ApsCurrentInstance cmdlet to see which instance is loaded. For example:

```
PS C:\> Get-ApsCurrentInstance

Name                : Instance1
InstanceId          : a06aa7f2-973b-7b75-b6aa-eae277dab3ab
ProductCode         : fa206597-d904-4d07-bb97-7f801efa985c
OriginalUpgradeCode : f321d748-8600-4cd1-beb9-8f1fc6dd0b85
OriginalProductCode : 54613b3a-5325-4fc9-8b4e-0b36f4a2863d
Version             : 8.5.650.0
InstallPath         : C:\Program Files\AppSense\Environment Manager\Personalizat
                      ion Server\bin\
CmdletPath          : C:\Program Files\AppSense\Environment Manager\Personalizat
                      Ion Server\bin\AppSense.ServerConfiguration.PowerShell.dll
                      ;C:\Program Files\AppSense\Environment Manager\Personaliza
                      tion Server\API\Modules\EMPImportExport
ProductName         : AppSense Personalization Server 8.5
IsDefault           : False
Status              : Unconfigured
DisplayName         : AppSense Personalization Server 8.5 (Instance1)
```

Server Installation

Completing installation of an AppSense product server involves:

- Installing any missing prerequisites
- Creating a database
- Initializing the product server and connecting to the database

Installing Prerequisites

Each AppSense product server specifies its own list of prerequisites. This list normally includes:

- IIS
- ASP.NET

To view missing prerequisites use the Get-ApsPrerequisite cmdlet. For example, to list missing prerequisites:

```
PS C:\> Get-ApsPrerequisite
```

Missing prerequisites can be installed using the Install-ApsPrerequisite cmdlet. For example, to install all missing prerequisites:

```
PS C:\> Install-ApsPrerequisite -All
```

You can also specify a subset/individual prerequisite using the Where-Object cmdlet. For example, to install the BITS prerequisite only:

```
PS C:\> $prerequisite = Get-ApsPrerequisite | Where-Object { $_.Name -like
"*BITS*" }
PS C:\> Install-ApsPrerequisite -Prerequisite $prerequisite
```

Setting up a new Server and Database

During the creation of product databases and configuration of product servers various credentials are required:

- -ConfigurerCredential – specifies the credentials that will be used to connect to the SQL Server during database creation and schema modification. This can be a dbo account or another user with appropriate privileges. This parameter defaults to the current logged in user
- -ServiceCredential – specifies the credentials that will be used by the product server to connect to the database during normal operation. This parameter is required during database creation to allow the database roles for the specified credentials to be created/mapped

Use Get-Credential to create a credential object for configurer/service credentials:

```
PS C:\> $credential = Get-Credential

cmdlet Get-Credential at command pipeline position 1
Supply values for the following parameters:
Credential
```

Use the Initialize-ApsDatabase cmdlet to create a product database:

```
PS C:\> Initialize-ApsDatabase -DatabaseServer "SQLSERVER\Instance" -DatabaseNam
e AppSensePersonalizationServer -ServiceCredential $credential -Verbose
VERBOSE: Starting Initialize-ApsDatabase
VERBOSE: DatabaseServer: SQLSERVER\Instance
VERBOSE: DatabaseName: AppSensePersonalizationServer
VERBOSE: Initialize-ApsDatabase: Creating database
VERBOSE: Initialize-ApsDatabase: Connecting to database
'AppSensePersonalizationServer'
VERBOSE: Initialize-ApsDatabase: Applying latest database schema
VERBOSE: Initialize-ApsDatabase: Applying default configuration to database
'AppSensePersonalizationServer'
VERBOSE: Completed Initialize-ApsDatabase
```

In addition to the parameters above, Initialize-ApsDatabase can also be supplied with the following parameters:

- ConfigurerCredentials – the account used to perform modifications to the database schema

Once a database has been created, then the server can be created using the Initialize-ApsServer cmdlet:

```
PS C:\> Initialize-ApsServer -DatabaseServer "SQLSERVER\Instance" -DatabaseName
AppSensePersonalizationServer -ServiceCredential $credential -Verbose
VERBOSE: Starting Initialize-ApsServer
VERBOSE: WebsiteName: Default Web Site
VERBOSE: WebsiteAuthentication: Windows
VERBOSE: DatabaseServer: SQLSERVER\Instance
VERBOSE: DatabaseName: AppSensePersonalizationServer
VERBOSE: ConnectionString:
VERBOSE: Initialize-ApsServer: Loading recommendations
VERBOSE: Initialize-ApsServer: Creating application pools
VERBOSE: Initialize-ApsServer: Applying Application Pool Defaults
VERBOSE: Initialize-ApsServer: Creating and initializing IIS directories
VERBOSE: Initialize-ApsServer: Applying web directory defaults
VERBOSE: Initialize-ApsServer: Registering and starting windows services
VERBOSE: Initialize-ApsServer: Applying Service Defaults
VERBOSE: Initialize-ApsServer: Setup complete
VERBOSE: Completed Initialize-ApsServer
```

In addition to the parameters specified above, Initialize-ApsServer can be supplied with the following cmdlets:

- WebSiteName – the name of the website this server will be placed on. The website must already exist.
- WebsiteAuthentication – the authentication of the website
- InstallModes

Using Install Modes to install a partial server

Install Modes allows part of a server to be installed. This functionality is utilised by the Personalization Server to offer:

- Personalization Server and Browser Interface
- Personalization Server only
- Browser Interface only

To determine the available install modes,

```
PS C:\> Get-ApsInstallModes
InstallMode Name                                        Description
----------- ----                                        -----------
ALL         Personalization Server and Browser Interface Configures both the…
EMP         Personalization Server only                 Configures only the…
EMBI        Browser Interface only                      Configures only the…
```

To setup the Personalization Server only, “InstallMode=EMP” would be supplied as an argument to Initialize-ApsServer.

Upgrading an existing Server and Database

To upgrade a product instance:

- Install the product server update – install the appropriate MSI or MSP file from the product server release, available from myAppSense.com
- Upgrade the database – Use Initialize-ApsDatabase with the same parameters to upgrade the database to the version needed for the installed product server

Use Get-Credential to create a credential object for configure or service credentials:

```
PS C:\> $credential = Get-Credential

cmdlet Get-Credential at command pipeline position 1
Supply values for the following parameters:
Credential
```

Upgrade the database, passing in the same parameters used to initially create the database

```
PS C:\> Initialize-ApsDatabase -DatabaseServer "SQLSERVER\Instance" -DatabaseNam
e AppSensePersonalizationServer -ServiceCredential $credential -Verbose
VERBOSE: Starting Initialize-ApsDatabase
VERBOSE: DatabaseServer: SQLSERVER\Instance
VERBOSE: DatabaseName: AppSensePersonalizationServer
VERBOSE: Initialize-ApsDatabase: Creating database
VERBOSE: Initialize-ApsDatabase: Connecting to database
'AppSensePersonalizationServer'
VERBOSE: Initialize-ApsDatabase: Applying latest database schema
VERBOSE: Initialize-ApsDatabase: Applying default configuration to database
'AppSensePersonalizationServer'
VERBOSE: Completed Initialize-ApsDatabase
```

Cloning an existing Server

The SCU cmdlets provide the ability to export and import server configuration settings. This allows the entire configuration of a product server to cloned between machines. The configuration is stored as an XML file and information within the file is encrypted using a password when the configuration is exported. This password must be specified when configuring a server using the configuration file.

To export configuration settings, use the Export-ApsServerImage cmdlet:

```
Export-ApsServerImage -Path “PersonalizationServer.xml" -Password "password"
```

To configure a server using a configuration file use the Initialize-ApsServer cmdlet with the -Path and -Password parameters:

```
Initialize-ApsServer -Path “PersonalizationServer.xml" -Password "password"
```

Database Management

Viewing Product Database Scripts

Use the Get-ApsDatabaseScript cmdlet to display the database scripts for a product:

```
PS C:\> Get-ApsDatabaseScript

Name                Description                 ScriptType AppliesTo
----                -----------                 ---------- ---------
Create Database     Database creation        CreateDatabase 8.5
Get Version         Version retrieva...  GetDatabaseVersion 8.5
IsDbEmpty           Checks If Databa...     IsDatabaseEmpty 8.5
Create Login        Creates a Login ...        CreateLogins 8.5
Purge Events        Purges all event...           BatchJobs 8.5
Create Schema       Create Schema              CreateSchema 8.5
Upgrade Schema 7... Upgrade Schema 7...       UpgradeSchema 7.0
Upgrade Schema 7... Upgrade Schema 7...       UpgradeSchema 7.1
Upgrade Schema 7... Upgrade Schema 7...       UpgradeSchema 7.2
Upgrade Schema 8... Upgrade Schema 8...       UpgradeSchema 8.0
Upgrade Schema 8... Upgrade Schema 8...       UpgradeSchema 8.1
Upgrade Schema 8... Upgrade Schema 8...       UpgradeSchema 8.2
Upgrade Schema 8... Upgrade Schema 8...       UpgradeSchema 8.3
Upgrade Schema 8... Upgrade Schema 8...       UpgradeSchema 8.4
```

Exporting Database Creation Scripts

Use the Export-ApsDatabaseScript cmdlet to export database scripts

```
PS C:\> Export-ApsDatabaseScript -ScriptType "CreateDatabase, CreateSchema,
CreateLogins" -Path C:\Scripts
PS C:\> Get-ChildItem C:\Scripts


    Directory: C:\Scripts


Mode                LastWriteTime     Length Name
----                -------------     ------ ----
-a---        17/07/2013     17:36       2281 Create_Database.sql
-a---        17/07/2013     17:36       4501 Create_Login.sql
-a---        17/07/2013     17:36     349344 Create_Schema.sql
```

Exporting Database Upgrade Scripts

Use the Get-ApsDatabaseVersion to get the version of a database:

```
PS C:\> $version = Get-ApsDatabaseVersion
```

Use the version from Get-ApsDatabaseVersion to retrieve the appropriate upgrade script:

```
PS C:\> Export-ApsDatabaseScript –ScriptType UpgradeSchema –AppliesTo $version |
Out-File "UpgradeSchema.sql"
```

Connecting a Server to a Database

Use the Set-ApsServerDatabase cmdlet to quickly switch a server to use a different database. You must have previously used Initialize-ApsServer to ensure the server has completed initial configuration. For example:

```
PS C:\> Set-ApsServerDatabase -DatabaseServer "SQLSERVER\Instance" -DatabaseName
 "PersonalizationServerNew" -ServiceCredential $credential -Verbose
```

Server Management

Resetting a Server to an out of box state

The Reset-ApsServer cmdlet can be used to reset a server to an out of box state. During a reset:

- All services and application pools are stopped/removed
- All web directories are removed
- The product configuration is deleted
- The server stops running

For example:

```
PS C:\> Reset-ApsServer
```

Starting or Stopping a Server

The Start-ApsServer and Stop-ApsServer cmdlets can be used to start or stop a product server. Starting or stopping a product server starts/stops all services and application pools associated with the product server. Product consoles may display an error message to users if they attempt to connect or perform any actions whilst a server is stopped.

To stop a server use the Stop-ApsServer cmdlet. For example:

```
PS C:\> Stop-ApsServer
```

To start a server use the Start-ApsServer cmdlet. For example:

```
PS C:\> Start-ApsServer
```

The current state of a server can be determined using the Get-ApsCurrentInstance cmdlet. For example:

```
PS C:\> Get-ApsCurrentInstance
```

Review the status field to determine if a server is started, stopped or unconfigured.

Viewing and Fixing Variances

If a server is encountering problems use the Get-ApsVariance cmdlet to highlight known issues:

```
PS C:\> Get-ApsVariance
```

Many variances can be fixed automatically. Use the Repair-ApsVariance cmdlet to attempt to repair variances:

```
PS C:\> Repair-ApsVariance -All
```

You can also limit the variances that will be repaired using the Where-Object cmdlet. For example:

```
PS C:\> $variance = Get-ApsVariance | Where-Object { $_.Name -eq "" }
PS C:\> Repair-ApsVariance -Variance $variance
```

Enabling or Disabling Logging

Logging can be enabled/disabled using the Enable-ApsLogging and Disable-ApsLogging cmdlets. Enabling logging creates log files in product server installation directory:

```
PS C:\> Enable-ApsLogging
```

Once an issue requiring logging has been resolved, use the Disable-ApsLogging cmdlets to disable logging:

```
PS C:\> Disable-ApsLogging
```

Scripting Multiple/Remote Servers

Enabling Remoting

Windows PowerShell remoting allows scripts to communicate with and run Windows PowerShell cmdlets on remote machines. No configuration is required to enable a computer to send remote commands. However, to receive remote commands, Windows PowerShell remoting must be enabled on the machine.

Windows PowerShell remoting is enabled on Windows Server 2012 by default. On all other systems, use the Enable-PSRemoting cmdlet to enable it. You can also use the Enable-PSRemoting cmdlet to re-enable remoting on Windows Server 21012 if it is disabled.

To enable remoting, open a Windows PowerShell console as an administrator and enter:

```
PS C:\> Enable-PSRemoting
```

Note

For more information refer to “Enable-PSRemoting” on Microsoft TechNet:

http://technet.microsoft.com/en-us/library/hh849694.aspx

Enabling CredSSP

CredSSP allows credentials to be passed to secondary machine on the network. This allows the SCU cmdlets to:

- Connect to SQL server databases using the current user
- Read server images (used for cloning servers) from a network share

Whilst CredSSP is not required, if it is not used you must specify the –ConfigurerCredential parameter wherever it is used.

Enabling CredSSP requires changes on both server and client machines:

- On a server machine, to allow CredSSP connections, use:

  `Enable-WSManCredSSP Server`
  Select Yes when prompted to allow CredSSP connections

- On client machines, to allow outbound CredSSP connections to specified servers, use:

  `Enable-WSManCredSSP Client -DelegateComputer @("server1.domain", "server2.domain")`
  Select Yes when prompted to allow CredSSP connections

Note

For more information refer to “Enable-WSManCredSSP” on Microsoft TechNet:

http://technet.microsoft.com/en-us/library/hh849872.aspx.

Working with PowerShell Sessions

Windows PowerShell allows for the concept of sessions, each of which may load in a separate copy of the SCU cmdlets. You can use Invoke-Command to create a temporary session and run cmdlets, or create a session that can be re-used using New-PSSession.

Note

For more information refer to “about_PSSessions” on Microsoft Technet:

http://technet.microsoft.com/en-us/library/hh847839.aspx.

Script Examples

Complete Server Configuration

```
$ProductName = "Personalization Server"
$ServiceUsername = "DOMAIN\AppSensePersonalizationServerService"
$ServicePassword = "password"
$DatabaseServer = "SQLSERVER\Instance"
$DatabaseName = "AppSensePersonalizationServer"

# Load cmdlets for appropriate product
Import-Module AppSenseInstances
Import-ApsInstanceModule -ProductName $ProductName -IsDefault

# Create service credential object
$SecurePassword = ConvertTo-SecureString -AsPlainText $ServicePassword -Force
$ServiceCredential = New-Object System.Management.Automation.PSCredential
($ServiceUsername, $SecurePassword)

# Install/fix all prerequisites
Get-ApsPrerequisite | Install-ApsPrerequisite

# Create database
Initialize-ApsDatabase -DatabaseServer $DatabaseServer -DatabaseName
$DatabaseName -ServiceCredential $ServiceCredential

# Configure server
Initialize-ApsServer -DatabaseServer $DatabaseServer -DatabaseName $DatabaseName
-ServiceCredential $ServiceCredential
```

Clone Multiple Servers from the Current Server

```
$Credential = Get-Credential
$ProductName = "Personalization Server"
$Path = "\\server\share\ServerImage.xml"
$Password = "password"
$ComputerName = @("server1.domain", "server2.domain")

Import-ApsInstanceModule -ProductName $ProductName -IsDefault
Export-ApsServerImage -Path $Path -Password $Password

Invoke-Command -ComputerName $ComputerName -ScriptBlock {
    Param($ProductName, $Path, $Password)
    Import-ApsInstanceModule -ProductName $ProductName -IsDefault
    Initialize-ApsServer -Path $Path -Password $Password
} -ArgumentList ($ProductName, $Path, $Password) -Authentication Credssp -
Credential $Credential
```

Return Variances from Multiple Servers

```
$Credential = Get-Credential
$ProductName = "Personalization Server"
$ComputerName = @("server1.domain", "server2.domain")

Invoke-Command -ComputerName $ComputerName -ScriptBlock {
    Param($ProductName, $Path, $Password)
    Import-ApsInstanceModule -ProductName $ProductName -IsDefault
    Get-ApsVariance
} -ArgumentList ($ProductName, $Path, $Password) -Authentication Credssp -
Credential $Credential
```