

Addons Scripting Guidelines

Last updated by | Eugene Tolmachev | Apr 20, 2021 at 3:55 PM EDT

Overview

These guidelines layout responsibilities for Microsoft and 3rd party PowerShell script developers.

Responsibilities of AVS




AVS Scripting environment is expecting to run scripts targeted for vCenter via PowerCLI from VMware.

Administration Level logins

The 3rd Party script will not have access to administrator password. Prior to executing a 3rd Party script, AVS will establish administrator level login sessions with vCenter. This will allow any API within vCenter to be accessed. There will be two logins established.

The first login will be done with PowerCLI's [Connect-VIServer](#)  cmdlet.

The second login will be done with VMware's [Connect-SsoAdminServer](#) .

When publishing the package, please ensure [VMware.vSphere.SsoAdmin](#)  and [VMware.vSphere.SsoAdmin](#)  are listed [dependencies](#)  of the package.

Environment

AVS will expose some standard runtime options via PowerShell variables. See below table for current list.

Var	Description
VC_ADDRESS	IP Address of VCenter

The script shall assume the directory it is executed in is temporary. The runtime environment allocated will be torn down after script execution.

Script Termination

AVS will terminate the script if it runs beyond the established AVS scripting timeout period. Timeout will be defaulted to 30 minutes unless one is not provided the script author.

Script Review

AVS will review the scripts and attempt to run them. Where necessary it is expected that the script author will provide support to AVS during this process.

Responsibilities of 3rd Party

Script authors of 3rd party solutions will need to follow some guidelines. These guidelines are expected to help scripts be more robust and supportable. The guidelines are also to help avoid negative impact to the AVS customer's Private Cloud.

Never login with cloudadmin through script

Any Module published to PowerShell Gallery should not attempt to login to vCenter with the AVS provided cloudadmin or any other role. Scripts that use Connect-VIServer or Connect-SsoAdminServer will not be allowed to run against an AVS Private Cloud.

Never elevate privileges for cloudadmin

Any Module published to PowerShell Gallery should not attempt to elevate privileges for the AVS provided cloudadmin role. Scripts that attempt to do this will not be allowed to run against an AVS Private Cloud. Elevating privileges for cloudadmin could have unintended consequences by giving elevated access to anyone using cloudadmin. The script is already logged in with administrator privileges and does not require elevating cloudadmin role.

Never use cloudadmin as the user for any installed software


If necessary, use the installation script to create a separate vCenter user and role to give it. Recommendation is to use the cloudadmin role as a base by duplicating it, then add necessary elevated privileges to the new role. The privileges required for the new role will need to be reviewed by Microsoft.

Top-level functionality should be exposed as functions with [CmdletBinding](#) taking all the inputs as the named parameters.

It is important that the commandlets do not use dynamic or conditional parameters. Commandlets we'll be statically analyzed to determine the parameters and all the parameters will be presented for the user to enter and provided to the commandlet.

Use `PSCredential` and `SecureString` if taking credentials or secrets as inputs. The functions and parameters must have user-friendly description.

Scripts should be packaged as a module and published as a nuget package.

Private AVS package repository will be used to install the modules. For the purpose of testing and review consider publishing to [PowerShell Gallery](#) , alternatively the package can be made available to us privately.

Lifecycle cmdlets should include preflight-install, install, preflight-uninstall, and diagnostics

Cmdlet	Description
preflight-install	Customer should be able to run this script prior to installation. It should report on any current state that the install script will depend on. If there are no errors it should be safe to install.
install	This script should call the pre-flight script and only continue if there are no errors in pre-flight. The script should be able to skip install steps already completed. Possibly from a previous install attempt.
preflight-uninstall	It should report on current state that the uninstall script will be working on.
uninstall	This script should be able to skip uninstall steps that were already completed.
diagnostics	This customer should be able to run this to get the most verbose state of the system. The intent is to have a tool to aid troubleshooting if install and/or uninstall do not run successfully

Scripts should be able to check if a step was already done and skip to next step

Things can go wrong and an initial installation attempt may partially complete before failure. In the interest of reducing operational support, the script should be smart enough to know what state the previous attempt is in and be able to either redo the steps leading up to it or skip past them if it can.

Uninstall script should be able to recover from any partially installed state

In the interest of reducing operational support, the script should be smart enough to know what state an installation is in. It may be installed successfully, failed on install, or failed on uninstall. For any scenario the script should be smart enough to either redo the steps it already did or skip past them if they are not needed.

Script should not dump secrets to output

Any user credentials created for the 3rd party software installation, should be kept secret.

Scripts should be published to Powershell Gallery for ease consumption and dependency management

<https://www.powershellgallery.com> 