Getting Started with E-Series PTK

**Architecture**

The E-Series PowerShell Toolkit (PTK) has a distributed architecture consisting of two components. One, of course, is the PTK itself while the second is the SANtricity Web Services Proxy (WSP). The PTK can be installed on each Windows laptop or server that needs to monitor or manage the E-Series storage system. At least one installation of WSP should be installed in the lab on a server that can be accessed on the network by PTK clients. The WSP also needs to be able to connect to the storage systems also by way of the network. At a high level, this is classic client/server architecture. The following diagram illustrates a sample topology.



**Web Services Proxy Requirements**

Please refer to the WSP installation documents for details requirements. The WSP itself is lightweight and robust in nature. While it can manage hundreds of devices while installed on a RasberryPI, we recommend a standard bare metal or guest server to run the WSP.

**E-Series PowerShell Toolkit Requirements**

The PTKis simply a PowerShell module that installs on the target Windows computer. It can be installed on either workstation or server grade operating systems.

**Credentials and Storage System Access**

Access to the storage system is governed by the WSP. The WSP provides 2 roles to access the storage system for which it servers as proxy. The roles are passive and active. The passive role can be thought of as read-only and is designed for the user or script that monitors storage system for health and performance. The active role can be thought of as having read-write access to the storage systems and is designed for the user or script that manages storage systems. Managing storage system involves partitioning, removal and mapping of storage objects. The passive role cannot alter storage systems.

Now that we have covered the basic security infrastructure of the WSP, how does this map to the PTK? Similar to the OnTAP PTK, the E-Series counterpart is based on a storage system specific credential with a standard PowerShell credential wrapper. There are principally 2 methods to obtain a credential. The first, more primitive but flexible method, involves running the Get-NeCredential. With this method you will need to supply a reference to the PowerShell credential, WSP url, WSP user, and WSP password. The second method is a function provided with the TK module called Get-NeProxyCredential. This method still requires all of the WSP parameters but will make a PowerShell credential. In both cases, the credential is persisted to a cache located on the computer where the command or function was executed. For all subsequent commands, a credential does not need to be created. The idea is to reduce typing for interactive sessions. It can be thought of as logging in to the WSP.

**Building a Script**

Script development should be an iterative process. When building a script (and particularly your first few scripts) the following process is recommended.

1. Become an expert with E-Series credentials. Use New-NeProxyCredential and leverage PowerShell "[splatting](https://technet.microsoft.com/en-us/library/jj672955.aspx)" to avoid repetitive code in scripts and typing in interactive sessions. The enhanced sample scripts demonstrate the splatting technique.
2. Use the WSP documentation to interactively run commands with it prior to executing a cmdlet. The WSP documentation is found at <https://yourServer:8443/devmgr/docs>. You will find all of the endpoints for the SANtricity REST API here. The PTK leverages the endpoints in the "storage-systems" grouping.
3. Get parameter values working in an interactive session prior to incorporating them with a script. The E-Series PTK provides a core set of cmdlets that are tunable with cmdlet parameters. This is contrasted by OnTAP cmdlets that are more singular in purpose. Thus there are many more OnTAP cmdlets
4. Use the debugger in the PowerShell ISE. The PTK delivers a robust set of objects that model the E-Series storage system objects. The debugger provides the ideal environment to interrupt script execution to inspect fine grain details of the objects used for cmdlet execution.

**Build a Library**

Once you get the hang of working with the PTK and it's architecture, it is only natural to incorporate repetitive tasks into a function library that can be sourced by other scripts and developers. Reuse!