Technical Report

NetApp AltaVault

AltaVault CLI Scripting Guide

Mike Braden, NetApp
June 2016 | TR-4521

Abstract

System administrators frequently are required to automate workflows to reduce the time required to perform repetitive tasks. This guide offers an overview of configuring an AltaVault appliance for scripting of workflows. Examples are included to illustrate several types of scripts used to perform automation tasks on an AltaVault appliance. This guide also includes some recommendations, tips and a brief troubleshooting section.

TABLE OF CONTENTS

LIST OF FIGURES

1 Overview

The NetApp® AltaVault™ cloud-integrated storage appliance provides the ability to use object storage to manage large amounts of data, either for backup workloads or for cold storage of infrequently accessed data. To efficiently and securely use low-cost cloud object storage services, the AltaVault appliance significantly reduces the amount of data that is generated from backup applications. Those storage services can be:

- In a public cloud provided by a hyperscaler (Amazon Web Services, Microsoft Azure, Google)
- With a local service provider
- In a private cloud storage solution such as NetApp StorageGRID® Webscale

Scripting command execution on an AltaVault system allows automation of a workflow. The best way to start is to manually perform a workflow to determine the commands that are required. Important aspects to determine are the command syntax, the order of execution, and the responses from the commands. Command responses can be successful responses, error messages, or unexpected responses. Understanding each of these aspects is important for implementing a script that offers predictable behavior.

Common use cases for scripting workflows include:

- Customized monitoring of an appliance
- Integration into monitoring applications
- Automation of workflows for operational staff

1.1 Other Considerations

When appliance monitoring is a primary factor for creating scripts, you should first determine whether an existing feature, such as SNMP, could provide the information that is required. If all or most of the required information is available with an existing feature, it is most likely not worth the investment to develop scripts for this purpose.

1.2 Script Language Selection

You have many choices of script languages and tools. Extensive information about creating scripts is available online, and NetApp recommends that you research the choices to determine what is the best fit for your requirements.

Things to consider are:

- Portability
- Maintenance
- Complexity

This guide describes the process for creating a script, along with details about how the AltaVault CLI functions. Examples are included that use Linux Bash shell and Python. Both offer portability and ease of use when creating simple scripts. Python provides a more robust and feature-rich language when more complex scripts are needed.

The following sections describe some options for languages for scripting. Selecting the one to use depends on several factors. If you have existing investment in a scripting language, it might be the best candidate for your solution as long as it provides Secure Shell (SSH) support that works with the AltaVault CLI. If there is no existing solution, then choosing one of the following languages will depend on the platform that runs your scripts and the time investment to set up the environment.

For example, if you have Linux as the scripting host and you require a simple task to be implemented, then a good choice is to use Bash. Bash is built in and doesn't require external libraries to perform SSH tasks. For Windows platforms, Bash would not be as good a choice because it would require Cygwin and would not provide a typical Windows command line.

Overall, Python makes a good choice for more complex scripting needs. It also is cross-platform, allowing use on both Linux and Windows script hosts. Python has the ability to perform more complex manipulation of the output from AltaVault commands and will perform faster because it does not need to create a new SSH session with each command.

Bash—Linux/UNIX Shell

Bash is the most common shell that is used in current Linux distributions. When you write scripts that run on a local machine or that run remotely by using SSH, a shell script can be a good choice for a task that is not complex.

For anyone with experience, executing shell commands remotely with SSH against a Linux shell on a remote server is an easy task. It's easy if you use the built-in feature to specify a command to run after the session is established. Unfortunately, because the AltaVault CLI is not a shell, it is a little more complex to implement. To execute the command structure for the AltaVault CLI from an SSH session, you must use the `heredoc` functionality of the shell. You cannot use the command option for SSH to execute commands.

Following are example errors when executing a remote command:

```
[user@svr1 ~]$ ssh admin@ava1 show info
NetApp AltaVault
NetApp ssh: ssh remote command is not allowed.
[user@svr1 ~]$ ssh admin@ava1 "show info"
NetApp AltaVault
NetApp ssh: ssh remote command is not allowed.
[user@svr1 ~]$
```

Expect

Expect is an extension of Tcl and provides a solution for scripting in a query-response format against a remote CLI. Expect provides a good method to overcome the limitation of executing remote commands. It also provides a more robust script than a regular Bash shell script. This guide does not provide information about creating Expect scripts. To learn more, go to https://en.wikipedia.org/wiki/Expect.

Perl

Perl has been an excellent choice for writing complex scripts. Perl is included in most Linux distributions and is also available for Windows by using ActiveState Perl. To learn more, go to https://en.wikipedia.org/wiki/Perl.

Python

Given its broad support, large number of libraries, and widely available information, Python is a good choice for more complex automations or for integration into existing frameworks. Python is frequently used for scripting in a large number of automation projects. To learn more, go to https://www.python.org/.

The main requirement for using Python to automate AltaVault workflows is to choose an SSH library to execute the remote commands. Paramiko is a popular SSH library for Python. Installation of Paramiko requires the Python package installer, pip. You can find information about pip and Paramiko at:

- http://www.paramiko.org/
- https://pip.pypa.io/en/stable/

Python also has a library called Pexpect to provide Expect-like functionality; however, this guide does not cover the use of Pexpect..

PowerShell

Windows scripting with PowerShell requires an SSH client library. This guide does not describe how to use PowerShell; however, depending on the library features, the script examples should have a similar structure when using PowerShell. For more information, see:

- OpenSSH for Windows
  https://blogs.msdn.microsoft.com/powershell/2015/10/19/openssh-for-windows-update/
- SSH from PowerShell using the SSH.NET library
  http://www.powershelladmin.com/wiki/SSH_from_PowerShell_using_the_SSH.NET_library

2 Environment Configuration

This section discusses what is required to set up the environment for the host that will execute scripts against the NetApp AltaVault appliance. The primary purpose of this section is to describe the configuration to use key-based authentication for Secure Shell (SSH) access so that you don't have to store passwords in the scripts. Having passwords stored in scripts requires maintaining the passwords, potentially in many files. It is also less desirable because it could be a security risk if the scripts are stored where others can view them.

Key-based authentication uses public-private key pairs for the authentication process. It is much simpler to maintain: When credentials must be changed, only a single key file gets updated on the scripting host, and changing the scripts is not required.

Note: Update the SSH client software to confirm that it is a recent distribution, which should include any recent changes for security.

Note: Do not use the root account to run scripts for AltaVault. For the best level of security, use a normal OS user account.

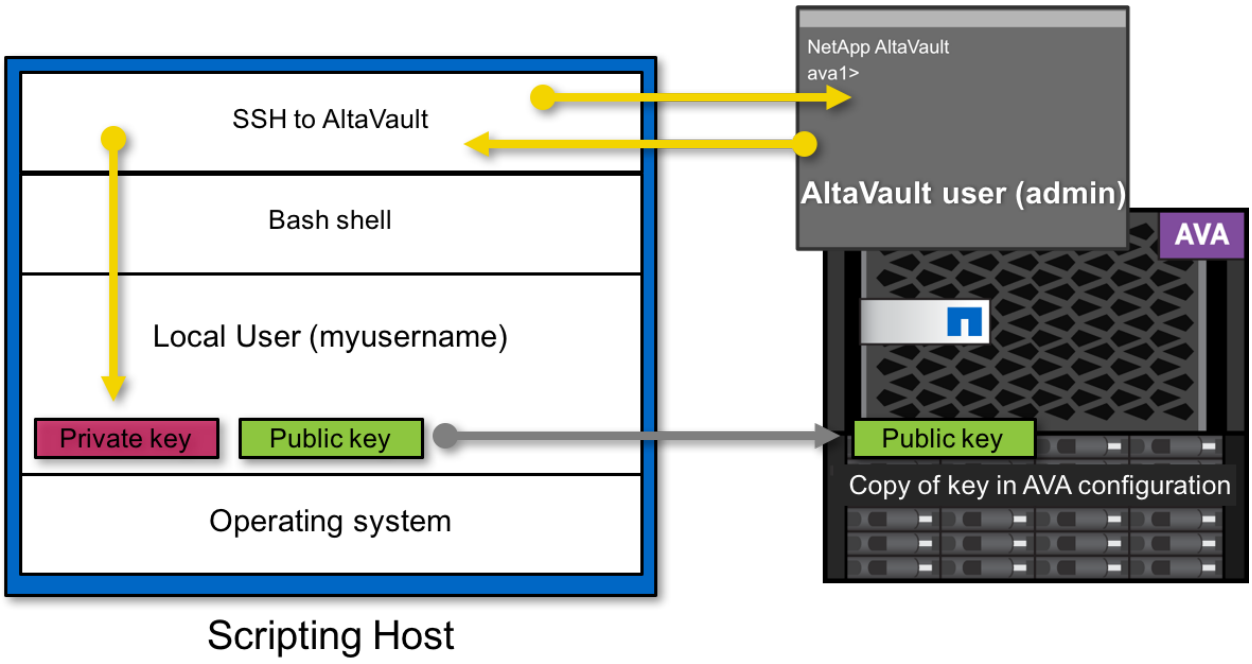
2.1 Script Server Configuration Overview

This document refers to the system that will be running scripts as the *script server*. This system could be any system. The examples in this guide use a Linux system as the host that runs the scripts.

For the scripts to run on the server, a generic user account is employed. It could be a local account on the server or a user from a directory service. It is important to understand that the user who is executing the scripts is the local OS user who logs in to the server where the scripts will execute. It is not the same user account that is used to log in to AltaVault.

Conceptually, we can view it as shown in Figure 1) SSH conceptual view. The public key is generated on the server that will run the scripts, while logged in as the user who will execute the scripts. The public key is saved in the AltaVault configuration. When SSH connects to AltaVault, it uses the user's private key to encode a message. AltaVault uses the public key to decode the message, which confirms that the message came from the user's account. The reverse occurs, and then a secure channel is negotiated.

Figure 1) SSH conceptual view.



2.2 AltaVault Configuration Overview

In the following sections, a public-private key is created and is saved on the AltaVault appliance or appliances that will be the target of the scripts that are created. The public-private key was chosen because it can be used for the authentication process. Using the public-private key for authentication without a passphrase means that the AltaVault user's password is not needed during the login process. Therefore, the scripts can run without user intervention, and the password does not need to be saved and updated in each of the scripts.

There are some differences in the configuration commands depending on the version of AltaVault software that is installed. In 4.1, the SSH server changed with the removal of some encryption types. In 4.2, the command structure changed with SSH client commands being removed, along with a slightly different syntax for setting the public key.

In all versions, the concept is the same. An SSH utility is used to generate a public-private key pair without a passphrase. It is important not to use a passphrase unless there is a plan to store that passphrase within the scripts that will execute. The public version of the key is entered in a configuration

command on the AltaVault appliance. This public key is the key that is generated for the OS user account that the scripts will run as.

The following sections describe one way to generate the public-private key pair for several different operating systems.

2.3 Host Configuration—Linux

The example system used for Linux for this guide has CentOS 7 installed. No specific Linux distribution or version is needed for scripting. The version shown here is to offer a complete description of the system that is used for this guide.

```
[user@svr2 ~]$ cat /etc/redhat-release
CentOS Linux release 7.2.1511 (Core)
```

Use the `ssh-keygen` utility to create the public-private key pair that will be used to execute commands on the AltaVault remotely by using SSH.

```
[user@svr2 ~]$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/user/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/user/.ssh/id_rsa.
Your public key has been saved in /home/user/.ssh/id_rsa.pub.
The key fingerprint is:
53:d5:4b:95:8f:ad:10:c8:11:2c:b1:6d:5d:ba:ce:dd user@svr2
The key's randomart image is:
+--[ RSA 2048]----+
|        .+o+....o|
|        .o+o.oo. |
|        ..+ oo +.|
|         o  ..o o|
|        S   .. . |
|         . o ... |
|            o . E|
|                 |
|                 |
+-----------------+
[user@svr2 ~]$
```

The public key that is generated is used for the configuration command in AltaVault. If you use multiple servers or users, each user and host should have the public key configured on each AltaVault appliance that will have scripts run against it.

The public key is stored in the SSH configuration directory, which is in the user's home directory.

```
[user@svr2 ~]$ ls ~/.ssh/
id_rsa  id_rsa.pub  known_hosts
```

The file that contains the public key is `~/.ssh/id_rsa.pub`. This is the public key that is entered in the configuration on the AltaVault appliance. **Do not share the private key file, which is stored in the `id_rsa` file.**

```
[user@svr2 ~]$ cat ~/.ssh/id_rsa.pub
ssh-rsa
AAAAB3NzaC1yc2EAAADAQABAAABQDT7MwJBHtJwZjcjOq3b/lC+jU7O/wg5dskcJrBU+B2PuptC67Lnv2kt18hMNE5ShY5Ku4
Zdxd9SDnqkV8EFWOCS3jQfdfDegJvC49UDnzxA6IBedTnrSWWtKBuUh0QqzWPUGcp7dVqsbXn user@svr2
[user@svr2 ~]$
```

2.4 Host Configuration—MacOS

MacOS configuration is similar to Linux. The public-private key pair is generated by using the `ssh-keygen` utility. It should be executed as the local OS user who will be executing the scripts.

```
mbraden-mbp01:~ mbraden$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/mbraden/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/mbraden/.ssh/id_rsa.
Your public key has been saved in /Users/mbraden/.ssh/id_rsa.pub.
The key fingerprint is:
ed:e6:79:6f:80:45:22:06:a1:85:31:a1:a7:bc:25:66 mbraden@mbraden-mbp01
The key's randomart image is:
+--[ RSA 2048]----+
|    +++o         |
|   ..+  o . .    |
|  . o  . . o     |
| . o     .  .    |
|  E .   S .o     |
| o +     .. .    |
|  .       o  .   |
|         o .. .  |
|          o. o.  |
+-----------------+
mbraden-mbp01:~ mbraden$
```

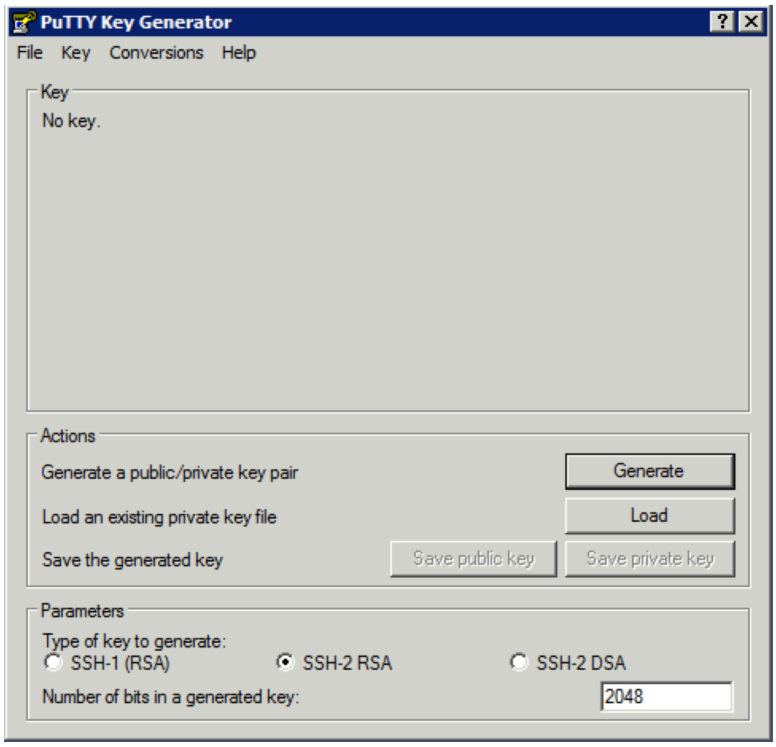
Similar to Linux, the SSH public key is stored in the SSH configuration directory in the user's home directory. The file that contains the public key is `~/.ssh/id_rsa.pub`. This is the public key that is entered in the configuration on the AltaVault appliance. **Do not share the private key file, which is stored in the `id_rsa` file.**

2.5 Host Configuration—Windows

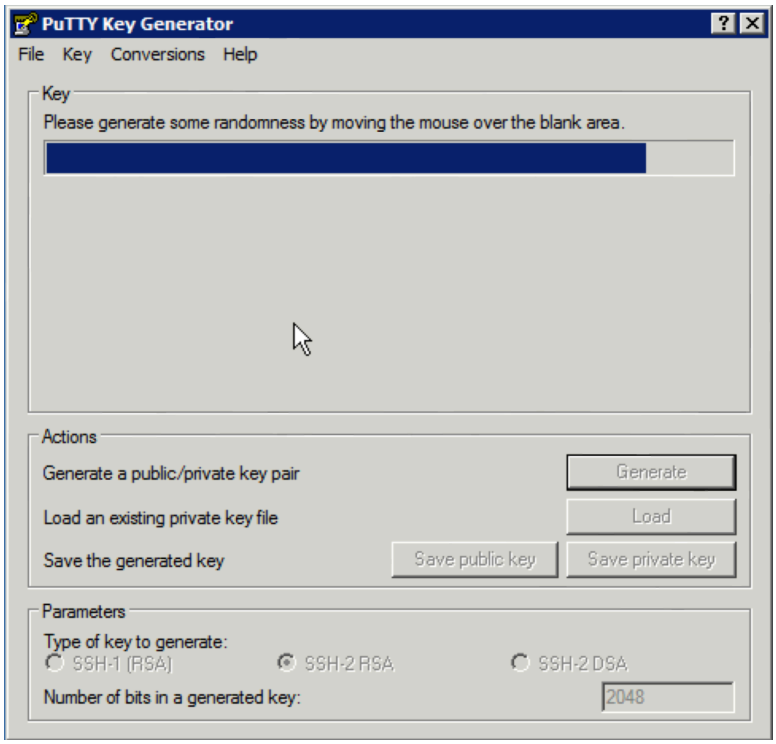
For Windows systems that use PuTTY, there is a utility called PuTTYgen that is used to create the key pair. Launch the utility from the Start > Programs menu or from the location that PuTTY was extracted to. Use the defaults for SSH-2 RSA.

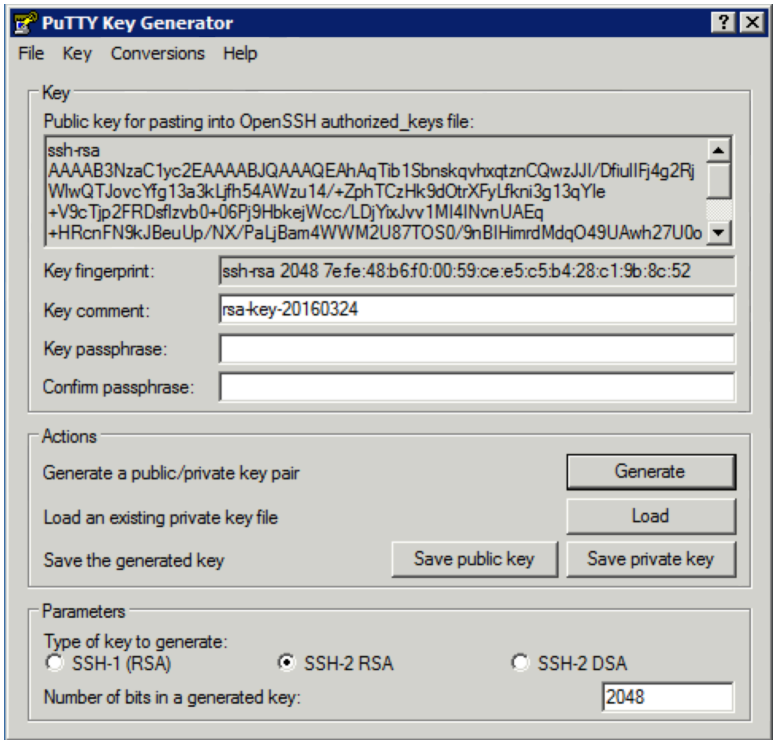
These steps allow a user to log in with PuTTY without entering the password. For a script that uses SSH, it might be possible to use the PuTTY command-line utility called Plink. See the PuTTY help information for more details.

1. Click the Generate button



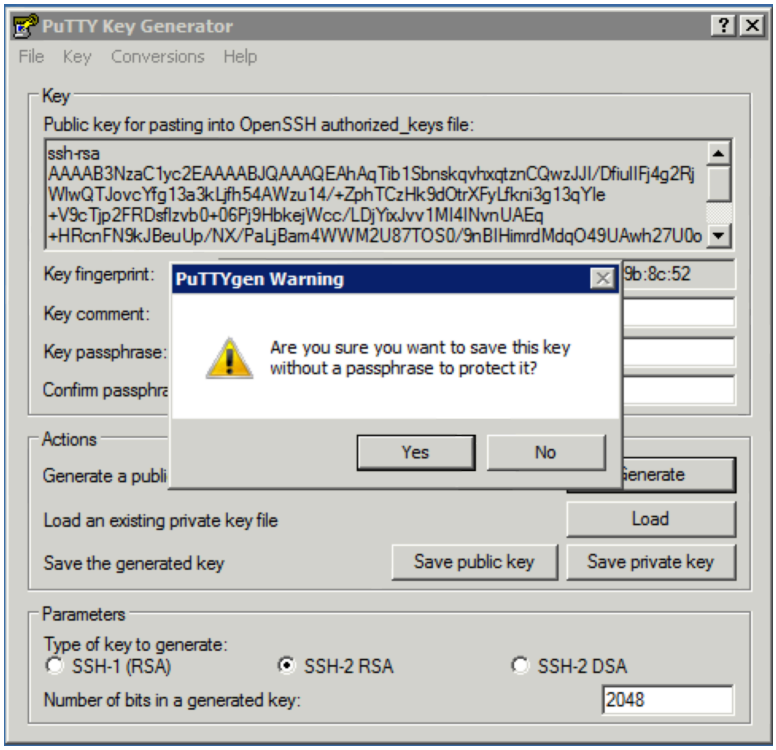
A prompt asks you to move the mouse to generate entropy with a progress bar to indicate the status.





2. Save the private key.
3. Save the public key.
4. (Optional) Use the Conversions menu to export the public key in OpenSSH format.

Note: Do not use a passphrase when you save the key.



2.6 AltaVault Configuration

The AltaVault configuration process is performed for the AltaVault appliance to accept the key that was generated as described in the previous sections. The public key is stored on the AltaVault appliance and is associated with a specific AltaVault user account.

1. Access the AltaVault CLI by using SSH with a password login. You can also use the serial console for a physical appliance or use the hypervisor console for a virtual appliance.
2. Enter the enable mode:
   ```
   ava02 > enable
   ```
3. Enter the configuration mode:
   ```
   ava02 # configure
   ```
4. For AltaVault OS 4.0 through 4.1, use the following command with your public key inside the quotation marks:
   ```
   ava02 (config) # ssh client user admin authorized-key public_key ' '
   ```
   For AltaVault OS 4.2, use the following command with your public key inside the quotation marks:
   ```
   ava02 (config) # ssh server user admin authorized-key ' '
   ```
   Note: You should enter the key within a surrounding pair of single quotation marks.
5. Save the configuration:
   ```
   ava02 (config) # write memory
   ```

Note: Remember to export the updated AltaVault configuration. To allow disaster recovery of the appliance, you should perform this step any time that the configuration is changed.

2.7 Managing AltaVault Authentication Keys

You must use the CLI to manage SSH authentication keys. Administration includes setting up new keys, removing old keys, and viewing the existing keys. Other SSH administration tasks include enabling or disabling specific ciphers and hash methods for message authentication codes (MACs) or keyed-hash message authentication codes (HMACs).

Syntax for AltaVault software 4.0 through 4.1:

```
[no] ssh client user <user> authorized-key key sshv2 <public key>
```

Syntax for AltaVault software 4.2:

```
[no] ssh server user <user> authorized-key <public key>
```

The command is entered in the configuration mode, as the following examples illustrate.

Example—AltaVault software 4.0 through 4.1:

```
ava02 (config) # ssh client user admin authorized-key key sshv2 'public_key'
```

Example—AltaVault software 4.2:

```
ava03 (config) # ssh server user admin authorized-key 'public_key'
```

Configuration Mode Help Output

The following is console output for the options and the syntax that are available for the `ssh client` command in configuration mode, as shown by using the AltaVault built-in help feature. In 4.1, this command is used to configure keys.

Note: The `ssh client` command is available in AltaVault OS up through the 4.1 release. The commands changed with AltaVault OS 4.2.

```
ava02 > en
ava02 # config t
```



```
ava02 (config) # ssh ?
client           Configure SSH client
server           Configure SSH server
ava02 (config) # ssh client ?
generate         Generate SSH client keys
user             Configure an SSH user
ava02 (config) # ssh client user ?
<user name>
ava02 (config) # ssh client user admin ?
authorized-key   Configure authorized-key for the specified SSH user
ava02 (config) # ssh client user admin authorized-key ?
key              Configure RSA authorized-key for the specified SSH user
ava02 (config) #
```

The following example is the console output for the options and the syntax that are available for the `ssh server` configuration mode commands by using the AltaVault built-in help feature for 4.1 and earlier.

```
ava02 (config) # ssh server ?
allowed-ciphers  Configure SSH server allowed ciphers
allowed-macs     Configure SSH server allowed MACs
enable           Enable SSH access to this system
listen           Configure SSH server interface access restrictions
max-auth-tries   Set the maximum number of authentication tries
passwd-auth      Enable SSH password authentication to this system
port             Set a port for SSH access
v2-only          Configure SSH server to accept only v2 connections
ava02 (config) # ssh server
```

The following example is the console output for the options and the syntax that are available for the `ssh server` configuration mode commands by using the AltaVault built-in help feature for 4.2.

```
ava03 (config) # ssh ?
server           Configure SSH server
ava03 (config) # ssh server ?
allowed-ciphers  Configure SSH server allowed ciphers
allowed-macs     Configure SSH server allowed MACs
enable           Enable SSH access to this system
listen           Configure SSH server interface access restrictions
max-auth-tries   Set the maximum number of authentication tries
passwd-auth      Enable SSH password authentication to this system
port             Set a port for SSH access
user             Configure an SSH user
v2-only          Configure SSH server to accept only v2 connections

ava03 (config) # ssh server user ?
<user name>
admin
ava03 (config) # ssh server user admin ?
authorized-key   Configure authorized-key for the specified SSH user
ava03 (config) # ssh server user admin authorized-key ?
<public key, e.g. "ssh-rsa ... user@server.com">

ava03 (config) #
```

Showing Existing Key Configuration

For the AltaVault OS through version 4.1, it is possible to use the `ssh client` command to list the keys that are configured.

Syntax for AltaVault software 4.0 through 4.1:

`show ssh client`

The following example is the console output for listing the keys in 4.1.

```
ava02 (config) # show ssh client
User Identities:
  User admin:
```



```
    Public key:
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQC+nAXWIPRigmssPO1OtBEsSVWrykOnlN7vJStotS8QZmH54ySwzxBoLIwX45+EXemJL
oPmbxWi1bLDU53kk0mCmVmAvV/jCgc/v1U5TELbPmvYzHMT8PYkJGBnAWQR5hevUAEnSuQ78o8fikfFXT admin

SSH authorized keys:
  User admin:
    Key 1:
AAAAB3NzaC1yc2EAAAADAQABAAABAQDfFC8e/IytIrA/S7JVx2wpyOtYQpNwjcatr1/VprYP4GuMYiCey25Qs0ne3p9C5/cua
5ZtKCSNlqpNkUJxCfXrDcQWtzdkqyaaxBDdoeAOfFayfISFL+mNtpxJ3T5wyQzAC32ZokZ/C679IXu2PP
    Key 2:
AAAAB3NzaC1yc2EAAAADAQABAAABAQDfFC8e/IytIrA/S7JVx2wpyOtYQpNwjcatr1/VprYP4GuMYiCey25Qs0ne3p9C5/cua
5ZtKCSNlqpNkUJxCfXrDcQWtzdkqyaaxBDdoeAOfFayfISFL+mNtpxJ3T5wyQzAC32ZokZ/C679IXu2PP mbraden
ava02 (config) #
```

Syntax for AltaVault software 4.2:

`show running-config`

The following example shows output from 4.2 `running-config` to view SSH keys that have been configured.

```
ava03 (config) # show running-config
##
## Network management configuration
##
   username "admin" password 7 …
   clock timezone America North United_States Eastern
   snmp-server engine-ID "0x8000430b80327b23c656"
   ssh server allowed-macs "hmac-sha1"
   ssh server user admin authorized-key "ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDT7MwJBHtJwZjcjOq3b/lC+jU7O/wg5dskcJrBU+B2PuptC67egJvC49UDnzxA6IBed
TnrSWWtKBuUh0QqzWPUGcp7dVqsbXn user@svr1"
   ssh server user admin authorized-key "ssh-rsa
TXXAB3NzaC1yc2EAAAADAQABAAABAQDT7MwJBHtJwZjcjOq3b/lC+jU7O/wg5dskcJrBU+B2PuptC67egJvC49UDnzxA6IBed
TnrSWWtKBuUh0QqzWPUGcp7dVqsbXn mike@svr1"
##
```

Removing Existing Keys

For the AltaVault OS through version 4.1, you can use the `ssh client` command to remove a key that was configured. The command uses the `no` prefix to negate the existing configuration command. The command takes the ID number of the key that was configured as shown when listing the keys.

Syntax for AltaVault software 4.0 through 4.1:

`no ssh client user admin authorized-key key sshv2 1`

The following example is the console output for help when removing a key in 4.1.

```
ava02 (config) # no ssh client user admin authorized-key key sshv2 ?
<public key>
1
2
3
ava02 (config) # no ssh client user admin authorized-key key sshv2
```

Syntax for AltaVault software 4.2:

`no ssh server user admin authorized-key 2`

The following example is the console output for help when removing a key in 4.2. In 4.2, there is no command to show the index number of the key that is needed for the `remove` command. Use `show running-config` to view SSH keys that have been configured. The first key is `index 1`, and it continues sequentially until the last key.

```
ava03 (config) # no ssh server user admin authorized-key ?
```



```
<public key>
1
2
ava03 (config) # no ssh server user admin authorized-key 2
ava03 (config) #
```

2.8 Configuring a Linux Host for Python Scripts

Normally, configuring a Linux host for Python is not very complex. Most distributions have Python installed by default. The more complex task is preparing a system to run scripts against an AltaVault CLI, which is not based on a Linux shell. Therefore, in addition to supporting SSH connections, Python must also be configured to perform tasks in an interactive type of SSH session.

Python does not include a native method to create an SSH session to a remote system. To use Python scripts on an AltaVault appliance, an SSH library is required. One such library is called *Paramiko*. At the date of this publication it has recent active development, so it's likely that it is currently maintained and well documented; therefore, it is a good choice for performing remote tasks with SSH. Paramiko also has the advantage of being written in Python, with the exception of the low-level cryptography.

Paramiko is installed by using the standard Python package management utilities. The examples used in this guide were performed on CentOS. For both CentOS and Red Hat Enterprise Linux, it is possible to install the Python installation management packages, pip and wheel, by using the Fedora Extra Packages for Enterprise Linux (EPEL). For systems that are configured to use Red Hat RHN, there is an optional repository to use EPEL, which is described on the EPEL website. For CentOS, it's possible to install EPEL from the CentOS Extras repository, which is enabled by default.

For more information about EPEL, go to https://fedoraproject.org/wiki/EPEL.

To install Paramiko on a CentOS 7 system, use the following as root (or use sudo):

```
yum install epel-release
yum install python-pip
yum install python-wheel
yum install python-devel
yum group install "Development Tools"
pip install paramiko
```

The following shows installation of Paramiko on CentOS.

```
[root@svr2 ~]# pip install paramiko
You are using pip version 7.1.0, however version 8.1.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
Collecting paramiko
  Using cached paramiko-1.16.0-py2.py3-none-any.whl
Requirement already satisfied (use --upgrade to upgrade): ecdsa>=0.11 in /usr/lib/python2.7/site-
packages (from paramiko)
Collecting pycrypto!=2.4,>=2.1 (from paramiko)
  Using cached pycrypto-2.6.1.tar.gz
Building wheels for collected packages: pycrypto
  Running setup.py bdist_wheel for pycrypto
  Stored in directory:
/root/.cache/pip/wheels/96/b0/e6/03e439d41cb2592b5c4c9c77873761d6cbd417b332076680cd
Successfully built pycrypto
Installing collected packages: pycrypto, paramiko
Successfully installed paramiko-1.16.0 pycrypto-2.6.1
[root@svr2 ~]#
```

After the installation of each of these packages has been completed, it is possible to use the example Python scripts from this guide or to author new scripts for AltaVault.

3 Introduction to the AltaVault CLI

3.1 AltaVault CLI Overview

The NetApp AltaVault CLI uses a network-device-style interface, in which it operates in three modes: user, enable, and configuration. Each mode provides a higher level of privilege with more commands than in the lower privilege mode. The modes are described in the AltaVault CLI Reference Guide (see section 3.2).

3.2 AltaVault CLI Reference

The AltaVault Command-Line Reference Guide is in the AltaVault section within product documentation on the NetApp Support site. Make sure that the version of AltaVault software that is running on your appliance matches the version of the documentation.

The AltaVault CLI provides a built-in help feature that lists available commands in each mode. The help feature is interactive; you can build the command by stepping through the options by using a question mark, `?`.

3.3 Determining a Workflow to Script

The first step is to decide what task you want to accomplish. For the example in this guide, we execute a prepopulation job by using a script.

Example workflow:

- The operator decides which file is needed for the restore.
- The operator executes the script from a Linux host, giving the name of the file as a parameter.

3.4 Determining How to Script a Workflow

After determining a workflow to automate by using a script, the first step is to manually perform that workflow by using the CLI. This step confirms that it is possible to perform the task and validates the commands and the specific syntax that are needed.

For this task, the best method is to use the same host that will be running the scripts and SSH to the AltaVault appliance and enter the commands on the CLI.

Following is an example of SSH from a Linux host to the AltaVault appliance that was configured for key-based authentication to test commands and syntax.

```
[roousert@svr1 ~]# ssh admin@ava02
NetApp AltaVault
Last login: Thu Mar 24 21:02:51 2016 from 10.10.10.11
ava02 > en
ava02 # show interfaces primary ?
<cr>             Display detailed of running state for this interface
arp              Display static and dynamic ARP entries for this interface
brief            Display brief running state for this interface
configured       Display configuration for this interface
dhcp-lease       Display DHCP lease for this interface
ava02 # show interfaces primary brief
Interface primary state
   Up:                 yes
   Interface type:     ethernet
   IP address:         10.10.10.10
   Netmask:            255.255.255.0
   IPv6 link-local address: fe80::250:56ff:fe00:0000/64
   Speed:              10000Mb/s
   Duplex:             full
   MTU:                1500
   HW address:         00:50:56:00:00:00
```

```
   Link:               yes
   Counters cleared date:  2016/02/05 16:44:08

ava02 #
```

When determining the list of commands to use, it is important to note the exact syntax and sequence of commands to accomplish the task up to the point at which output needs to be processed. For Bash shell scripts, this step is critical. For Bash, each command sequence is sent, then the response is received so that it can be manipulated to obtain the part of the output that is required for the next steps in a workflow.

For Bash scripts, the typical flow is to enter the commands to get to the console mode that is required (enable or configure). Then determine the correct syntax for the command that you want to perform. All of this effort is combined and is a small block of the command string that will be executed with a single SSH session.

When performing commands that have output greater than a typical console screen, the AltaVault CLI pages through the output. One very useful feature is to disable the paging function before you execute commands. This step confirms that when a long list of output is sent, there is no pause in the output. A pause in the output could make the script appear to hang.

The command syntax to disable paging is:

```
enable
```

```
no cli session paging enable
```

This step can be performed in enable mode, so there is no need to change to configuration mode. This command disables paging for the duration of the SSH session.

An alternate command to disable paging is:

```
terminal length 0
```

However, this command will disable paging until the terminal length is configured for the appropriate terminal size.

3.5 Executing Commands from a Bash Shell

Executing commands on the AltaVault appliance is accomplished by using the `heredoc` feature of Bash. It is a way to send a list of commands to the AltaVault CLI over a secure SSH connection and receive a response that is the output of the commands. This feature provides a method to send all the commands needed for a single workflow in one SSH session. It is also required because all the commands for an AltaVault CLI script must be input after the login is completed.

This approach is different from scripting commands for a Linux server. When scripting against a Linux server, it is possible to send a shell command to the remote Linux server by using the command feature of SSH. Because the AltaVault CLI is not a Linux shell, it's not possible to send commands in this manner.

Following is an example of executing commands from the shell by using the `heredoc` method with Bash. For this example, a text file is created with the following content:

```
ssh admin@ava02 << EOF
en
show info
EOF
```

Then all the lines are selected, copied, and pasted into the Bash shell on the Linux scripting host as follows:

```
mbraden-mbp01:mbraden$ ssh admin@ava02 << EOF
> en
```



```
> show info
> EOF
Pseudo-terminal will not be allocated because stdin is not a terminal.
NetApp AltaVault
ava02 > en
ava02 # show info
Current User:      admin

Status:            Critical
Config:            initial
Appliance Up Time: 26m 5s
Service Up Time:   Not Running
Number of CPUs:    2
CPU load averages: 0.08 / 0.02 / 0.01
Temperature (C):   0

Serial:            9194768144
Model:             AVA-v2
Revision:          A
Version:           4.1
mbraden-mbp01:mbraden$
```

4 Building a Script Workflow

4.1 Overview of a Script to Prepopulate a File

In this example, the goal is to have a script that can be executed by a Linux database administrator to perform a prepopulation of a file by specifying a path and a file name. The share path is a static variable in the script for this example so that it restricts the database administrator to the share for the database backup that the administrator manages.

Because the example is for an NFS export, the script is simplified by setting a fixed share path. Other methods can be used to make this step work and to check the existence of the file if the Linux host that runs the script is also the host with the NetApp AltaVault export mounted. In that case, it would be possible to list the files in the share to see whether the one that is requested exists. Just be aware that in Linux, it is likely that the mount path is not the same as the export path. Most likely, the base of the AltaVault export is in a directory structure, and it's possible that the export path does not match the AltaVault path setting for the export.

Now that we understand the workflow, the next task is to understand how the prepopulate feature works from the CLI. The AltaVault CLI Guide, available on the NetApp Support site, is the best place to start. Make sure that the version of the guide matches the version of the AltaVault OS that you are running.

The CLI guide has a section about the prepopulation command.

Figure 2) AltaVault CLI guide prepopulate command.

Example

```
CLI (config) # datastore integrity check start
```

datastore prepop

Retrieves data from the cloud and populates the AltaVault with it locally so that the AltaVault has a local copy of the target data.

Syntax

datastore prepop {num-days <number of days> [pattern <pattern>] | pattern <pattern> [num-days <number of days>]} start-date end-date [force] [static-files create-cifs | remove-cifs] [bue-header] [bue-footer]

Parameters

num-days <number of days>	Specify the number of days (from the current date) up to which the AltaVault should go back and start prepopulation. This command filters the data retrieved by the number of days last modified.
pattern <pattern>	Filters the data retrieved by the pattern you specify. When using the **datastore prepop pattern** command, you must use the escape character (\) to handle filenames with the following special characters: \ ^ $ ( ) { } [ ] + ? * Use a backslash (\) before the special character in the filename. You can specify multiple filenames using a pipe symbol (|). Do not use an escape character before the |. For detailed example, see the Usage section.
start-date	Specify the date to start populating data.
end-date	Specify the date to stop populating data.
static-files create-cifs	Creates a CIFS share for storing static files. Static files can be used to serve content for user files which are offline. This is relevant only when using AWS Glacier cloud storage. User files are considered to be offline when some part of the data backing those files is present only in Glacier storage and not on the appliance.
static-files remove-cifs	Removes cifs share used for storing static files.
bue-header	Prepopulate the backup file' headers (without including all of the actual backup file data) for backup operations to succeed. BUE is BackUpExec that is a backup application, which stores special information (like a catalog) in all of its .bkf file header and footer. When you store files in Amazon Glacier and they go offline (the AltaVault evicts cached data), then new backup operations fail because they need to read these headers and footers to succeed.
bue-footer	Prepopulate the backup file' footers (without including all of the actual backup file data) for backup operations to succeed. BUE is BackUpExec that is a backup application, which stores special information (like a catalog) in all of its .bkf file header and footer. When you store files in Amazon Glacier and they go offline (the AltaVault evicts cached data), then new backup operations fail because they need to read these headers and footers to succeed.

Usage

After a disaster, you can perform data recovery. During this process, you must use the **datastore prepop** command to warm the data before you try to restore your backup data using your backup application.

- To prepop a file named a|b in a share named cifs, type the following command:

```
CLI (config) # datastore prepop pattern /cifs/a\|b
```

- To prepopulate a file named a|b in the share named nfs, type the following command:

```
CLI (config) # datastore prepop pattern /nfs/a\|b
```


After reviewing the CLI guide, we can determine the following about performing a prepopulate task:

- The command is executed in configuration mode.
- The script must log in, change to enable mode, then change to configuration mode to execute the command.

The command syntax is:

```
datastore prepop pattern filepath_and_name
```

4.2 Verifying Syntax by Using the AltaVault CLI

The best method for verifying syntax is to use the CLI help command `?`. Enter the start of the command followed by `?` to get the options. Start with the first part of the command and check each additional part of the syntax until the whole command is complete.

```
ava1 # datastore ?
file-status      Get the availability status (Online | Offline) of the file
integrity        Datastore integrity check utility
prepop           Prepopulate files to the appliance
ava1 # datastore prepop ?
end-date         Prepop files modified on or before this date (yyyy-mm-dd)
num-days         Filter by number of last modified days
pattern          Filter by pattern
start-date       Prepop files modified on or after this date (yyyy-mm-dd)
ava1 #
```

Manually Performing the Task to Verify Workflow

```
ava1 # datastore prepop pattern en*
Prepopulating the datastore may take some time.
Are you sure you want to start the prepopulation process? [y/n] y
Prepopulating all files matching en*
ava1 #
```

When using the CLI, we can see that the command can be issued in enable mode and that it's not required to access configuration mode. That saves us one step. It is also asking for a confirmation of the command (Y or N) before it continues. So, our script needs to handle not just sending the request, but also answering the Y/N prompt.

In this test case, we used a dummy path, a file that does not exist (`en*`). It seems to accept the pattern and not check that it's valid. It would be good to have the script confirm that the file exists first.

4.3 Testing the Script Format for the Command

In this example, we created a list of commands to perform the prepopulation in a text file and we pasted it into the shell on the Linux script host to see how it executes. This task tests the whole process so that we can see what happens by using SSH. This text is pasted into a shell on the Linux server, not the AltaVault CLI.

```
ssh admin@ava1 << EOF
datastore prepop pattern /mfsshare/mdir/*
y
EOF

[user@svr2 ~]$ ssh admin@ava1 << EOF
> datastore prepop pattern en*
> y
> EOF
Pseudo-terminal will not be allocated because stdin is not a terminal.
```

```
NetApp AltaVault
ava1 > datastore prepop pattern en*
Prepopulating the datastore may take some time.
Are you sure you want to start the prepopulation process? [y/n] Prepopulating all files matching
en*
[user@svr2 ~]$
```

Now we address the additional part of confirming that the file exists. Reviewing the CLI guide shows a command to check a file – `datastore file-status`. We check the AltaVault CLI for the syntax and check the operation of the command by using both a file that exists and one that does not exist. That way, we see the positive and negative output.

```
ava1 # datastore file-status /share1
Online
ava1 # datastore file-status /share1/test2/myfile.bin
% Internal error (code 1003)
ava1 # datastore file-status /share1/test2/myfile1.bin
Online
ava1 #
```

By using this command, we can verify that a file exists. The expected output is `Online` for a file that exists.

Now all the parts of the script workflow have been worked out. The next task is to combine them into a complete script that can be executed on the Linux script host by a user.

4.4 Finalizing the Script Workflow

The script is entered on the server by using a Linux text file editor. It is a good idea to provide a help feature when the script requires command-line arguments. To make the script easy to change in the future, it starts with the required settings that are most likely to change:

- AltaVault host name
- AltaVault user
- Export path

For this example, the script file name is `ava_restore`. The file must have permissions set appropriately to execute the file. In Linux, it's best to set them with the following command:

```
chmod +xr,-w ava_restore
```

The final script has some added functionality. When executed with `?`, it prints the usage information, and an option was added to allow listing of jobs.

When prepopulating a file, the script checks to see that the file exists. Next the script executes the prepopulation commands. Finally, it prints the job listing so that it is possible to see the job number.

The following is the final workflow that has been identified and tested manually with CLI commands. It has also been tested from the shell by cutting and pasting `heredoc` shell commands.

- Check the arguments provided:
  - If `-h` or `-?` was entered, print the usage information and exit.
  - If `-l` was entered:
    - Execute commands to show prepopulation jobs.
    - List the output.
    - Exit.
  - If `-f` was entered:
    - Check the status of the file:
      - If the file does not exist, exit.
    - Perform the prepopulation of the file.

- Execute commands to show prepopulation jobs.
- List the output.
- Exit.

5 Bash Script Example

The following section describes the script that was created in Bash by using the workflow and command syntax that were created in the previous section.

After finalizing the script workflow, the next step is to create a script by using the commands from the workflow.

5.1 Script Recommendations

- When you use paths for Linux commands, use the absolute path. This step confirms that the commands that you expect to run will be the actual commands that are used. Relative paths are a bad practice and can lead to unexpected behavior when executed by a different user or in a different manner, such as a Cron job.
- For any OS commands, such as `grep`, use a variable with the full path to the command. This approach makes changing the path or the command simpler by keeping it in one place. It also prevents a command from being executed from a different path.
- For variables that will commonly be modified, place them at the start of the script, in a section to group them.
- Add a help option so that a user can easily obtain the correct syntax or use of the script.

5.2 Example Output from a Prepopulation Script

The following examples show the help output for the sample script.

```
[user@svr1 ~]$ ./ava_restore -h
usage: ava_restore [-h|?] [-l] [-f prepop_file_path]
```

One of the options provided is a method to simply list the status of jobs that have been run. The main purpose is to show the status if a long-running job is in progress. To show the status, execute the script with `-l`.

```
[user@svr1 ~]$ ./ava_restore -l

Prepopulation jobs
------------------

Job: Job 10
     Status:             Completed
     Start Time:         2016/03/16 01:19:57
     Complete Time:      2016/03/16 01:19:58
     Files in job:
          /share1/test2/myfile2.txt

[user@svr1 ~]$
```

The primary workflow for the script is to prepopulate a file. Following is an example of this operation being performed.

```
[user@svr1 ~]$ ./ava_restore -f test2/myfile1.bin

Checking file...
file: /share1/test2/myfile1.bin
File exists
Prepopulating the datastore may take some time.
 Prepopulating all files matching /share1/test2/myfile1.bin
```

```
Prepopulation jobs
------------------

Job: Job 10
     Status:             Completed
     Start Time:         2016/03/16 01:19:57
     Complete Time:      2016/03/16 01:19:58
     Files in job:
          /share1/test2/myfile2.txt

Job: Job 11
     Status:             Downloading (4%)
     Start Time:         2016/03/16 01:22:55
     Files in job:
          /share1/test2/myfile1.bin

[user@svr1 ~]$
```

And the operation is performed again, this time using the `-l` option to check the status of the job.

```
[user@svr1 ~]$ ./ava_restore -l

Prepopulation jobs
------------------

Job: Job 10
     Status:             Completed
     Start Time:         2016/03/16 01:19:57
     Complete Time:      2016/03/16 01:19:58
     Files in job:
          /share1/test2/myfile2.txt

Job: Job 11
     Status:             Completed
     Start Time:         2016/03/16 01:22:55
     Complete Time:      2016/03/16 01:22:57
     Files in job:
          /share1/test2/myfile1.bin

[user@svr1 ~]$
```

5.3 Sample Prepopulation Script

```
#!/usr/bin/bash
# Mike Braden
# NetApp
# AltaVault 4.1
# 2016-03-15
#
# Change this to the hostname of your altavault used for ssh
AVA_HOST="your_ava"
# Set this to the user that will be logging into the AVA
AVA_USER="admin"
# Set this to the Export Path setting in the AVA GUI
AVA_SHARE_PATH="/share1"

# Adjust these to match the local OS
SSH_CMD="/usr/bin/ssh -q"
GREP_CMD="/usr/bin/grep"
CUT_CMD="/usr/bin/cut"

# Do not change
OPTIND=1
```

```
FILE_STAT="Not found"

# help usage
function show_help
{
   echo "usage: ava_restore [-h|?] [-l] [-f prepop_file_path]"
}

function list_jobs
{
   #
   # list jobs
   #
   echo -e "\nPrepopulation jobs"
   echo -e "------------------\n"

   $SSH_CMD $AVA_USER@$AVA_HOST << EOF | $GREP_CMD -v $AVA_HOST
en
show datastore prepop jobs files
EOF

}

function file_status
{
echo -e "\nChecking file..."
echo "file: $AVA_SHARE_PATH/$PRE_FILE"

   FILE_STAT=$($SSH_CMD $AVA_USER@$AVA_HOST << EOF | $GREP_CMD -v $AVA_HOST
en
datastore file-status $AVA_SHARE_PATH/$PRE_FILE
EOF
)

   #echo "Status is $FILE_STAT"
   if [ "$FILE_STAT" = "Online" ]; then
      echo "File exists"
   else
      echo "Error: File does not exist"
      exit 1
   fi
}

#
# process input
#
while getopts ":f:hl" opt; do
   case "$opt" in
   h|\?)
      show_help
      exit 0
      ;;
   f)
      PRE_FILE=$OPTARG
      #echo "file is: $PRE_FILE"
      ;;
   l)
```



```
      list_jobs
      exit 0
   ;;
   esac
done


#
# check if file exists
#
file_status

#
# start prepopulate job
#
$SSH_CMD $AVA_USER@$AVA_HOST << EOF | $GREP_CMD -v $AVA_HOST | $CUT_CMD -d] -
f2
datastore prepop pattern $AVA_SHARE_PATH/$PRE_FILE
y
EOF

#
# list jobs
#
list_jobs
```


6 Python Script Example

6.1 Python Example Script Overview

Python allows both interactive entry and executing a script from a file. Interactive mode is a good way to test whether a particular workflow, syntax, or set of commands will operate as expected. One important difference is that when using a script, the time between sending and receiving text can cause problems. When you create scripts based on this guide, NetApp recommends that you insert delays to allow the commands that are sent to the NetApp AltaVault appliance to complete before reading the output or continuing to send more commands.

The example here uses the Paramiko library as the SSH client by using the configuration steps that were shown in section 2, Environment Configuration.

One advantage of using Python and Paramiko is that an SSH session is opened once at the start of the script. For most tasks, the session can be kept open while commands are issued, and responses are received and processed similar to an interactive user session. This capability can reduce the time that it takes to execute scripts because a new SSH session doesn't have to be started with each command, unlike for the previous Bash shell script example.

6.2 Python Example Script Output

The following is an example of the output that the script generates.

```
[user@svr2 ~]$ ./ava_example.py

Logging in to ava1
------------------------------
Last login: Tue Jun  7 13:22:44 2016 from 10.10.10.10


Show Info
------------------------------
show info
ava1 > show info
Current User:      admin

Status:            Healthy
Config:            initial
Appliance Up Time: 5d 20h 42m 15s
Service Up Time:   5d 20h 41m 42s
Number of CPUs:    4
CPU load averages: 0.19 / 0.07 / 0.02
Temperature (C):   0

Serial:            12345678
Model:             AVA-v2
Revision:          A
Version:           4.2
ava1 >

Show Stats Data
------------------------------
show stats data
Storage Optimization
Expanded Data: 5.40 GB
Deduplicated Data: 4.90 GB
Deduplication factor: 1.10

Replicated Data
Cloud Synchronized Until: 2016/06/01 16:44:34
Time to complete replication: Data replication complete
Replication bytes pending: 0.00 B

Disk Storage Allocation
```

```
Used: 5.27 GB
Free: 152.86 GB
Total: 158.53 GB

Inode usage
Used inodes: 1552
Total inodes: 9732096

Cloud Storage allocation
Used: 4.89 GB
Total: 11.00 TB
ava1 >
[user@svr2 ~]$
```

6.3 Python Example Script

```
#!/usr/bin/python
#
# Mike Braden
# NetApp
# AltaVault 4.2
# 2016-06-07

import paramiko
import traceback
import time

# ------------ Change to match your environment - Begin
#
# Change this to the hostname of your altavault used for ssh
ava_host = "ava1"

# Set this to the user that will be logging into the AVA
avauser = "admin"

# Host key file location
host_keyfile = '/home/user/.ssh/known_hosts'

# User key file location
user_keyfile = '/home/user/.ssh/id_rsa'

# ------------ Change to match your environment - End

try:
    client = paramiko.SSHClient()
    client.load_system_host_keys()
    client.load_host_keys(host_keyfile)
#    client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    client.set_missing_host_key_policy(paramiko.WarningPolicy())

    client.connect(ava_host,username=avauser,key_filename=user_keyfile)

    channel = client.invoke_shell()
    output = channel.recv(1000)
    print "\nLogging in to " + ava_host
    print "-" * 30
    print output
    time.sleep(1)

    channel.send("show info\n")
    time.sleep(1)
    output = channel.recv(1000)
    print "\nShow Info"
    print "-" * 30
    print output

    channel.send("show stats data\n")
    time.sleep(1)
    output = channel.recv(1000)
```



```
    print "\nShow Stats Data"
    print "-" * 30
    print output

    channel.close()
    client.close()

except Exception as e:
    print('** Exception: %s: %s' % (e,__class__, e))
    traceback.print_exec()
    try:
        client.close()
    except:
        pass
    sys.exit(1)
```

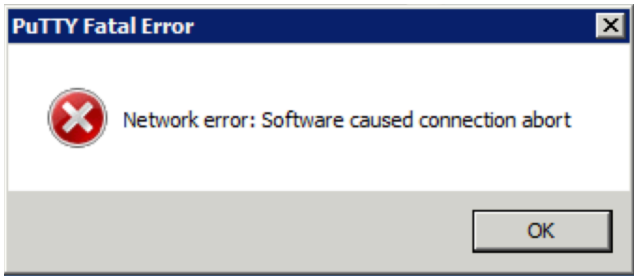
7 Troubleshooting

7.1 Connection Refused or Other Connection Errors

If you receive a connection error message, make sure that you are using the latest version of the SSH software.

For Windows, this version should be at least PuTTY 0.67. Older versions have known issues in connecting to NetApp AltaVault because of recent security changes to mitigate potential vulnerabilities.

Figure 3) PuTTY connection error.



7.2 Yum Group Installation Error “No packages in any requested group”

When you install a group, there could be errors when you try to execute a group installation.

```
[root@svr1 ~]# yum group install development
Loaded plugins: fastestmirror, langpacks
There is no installed groups file.
Maybe run: yum groups mark convert (see man yum)
Loading mirror speeds from cached hostfile
 * base: mirror.us.leaseweb.net
 * epel: mirror.symnds.com
 * extras: ftp.linux.ncsu.edu
 * updates: centos.mirrors.tds.net
Warning: Group development does not have any packages to install.
Maybe run: yum groups mark install (see man yum)
No packages in any requested group available to install or update
```

You can resolve this issue in several ways, such as by using an alternate syntax for installation. The easiest solution is to use the `yum` command to convert the old style groups to objects.

```
yum groups mark convert
```

```
 [root@svr1 ~]# yum groups mark convert
Loaded plugins: fastestmirror, langpacks
There is no installed groups file.
Maybe run: yum groups mark convert (see man yum)
Loading mirror speeds from cached hostfile
 * base: mirror.us.leaseweb.net
 * epel: mirror.cogentco.com
 * extras: ftp.linux.ncsu.edu
 * updates: centos.mirrors.tds.net
Converted old style groups to objects.
```

7.3 Command Does Not Return Output in Bash Script

Some commands that collect data can take longer than usual to run. It is likely that they will not complete before the SSH session is terminated. The best way to determine if this is an issue is to run the command

using the AltaVault CLI interactively. If the command produces output and the same command does not product output when executed from a script, then it is likely experiencing this issue. It is possible to work around the issue using the following example:

```
/usr/bin/ssh -tt admin@ava42nbu03 << EOF
enable
no cli session paging enable
show log
exit
EOF
```

This example of running the show log command uses the "`-tt`" option of SSH to force an interactive terminal session. This session will not close until an exit command is executed. Make sure you use `exit` or your script will appear to hang after receiving output from the command.

Version History

Following is the change history for this document.

Version	Date	Document Version History
Version 1.0	June 2016	Initial version of this document.

Refer to the Interoperability Matrix Tool (IMT) on the NetApp Support site to validate that the exact product and feature versions described in this document are supported for your specific environment. The NetApp IMT defines the product components and versions that can be used to construct configurations that are supported by NetApp. Specific results depend on each customer's installation in accordance with published specifications.

Copyright Information

Trademark Information