NETAPP INC.

Continuous Integration with NetApp Technology

Chhaya Gunawat

8/26/2015

Author: Chhaya Gunawat
Reviewer: KumaraSwamy Namburu

This document highlights the various workflows in developer builds.

Table of Contents

Prerequisites and setup

Setting up Cluster

Adding nodes (Refer InfrastructureConfiguration.doc)

Setting up round robin DNS:

Once the cluster is been setup and all the nodes added to the cluster, we recommend setting up DNS load balancing with round robin

- A DNS load-balancing zone is a DNS server inside the cluster that dynamically evaluates the load on all LIFs and returns an appropriately loaded LIF. DNS load balancing works by assigning a weight (metric), which in turn is based on the load.

- It is recommended to use round-robin for load balancing multimode interface groups, use the round robin option for load balancing a single connection's traffic across multiple links to increase single connection throughput.

For additional information refer the ONTAP administration Guide

Cluster Management Interface:

- The cluster management interface is used for managing the cluster. It provides one IP address to manage the cluster and will fail over to another node, if necessary.
- This is the preferred IP address for managing the cluster, but can also manage the cluster by logging in to the node management IP address of a node in the cluster.
- Since the cluster management interface must be able to fail over, the port role for the interface must be "data" and typically the best choice for an IP address is one on the data network.
- Alternatively, can assign an IP address on the data network to the cluster management interface

For additional information/help on setting up the management interface: refer cluster management interface

e.g Creation of Cluster management interface will look like :

```
Enter the cluster management interface port [e0c]:
Enter the cluster management interface IP address: 192.0.2.60
Enter the cluster management interface netmask: 255.255.255.192
Enter the cluster management interface default gateway: 192.0.2.1

A cluster management interface on port e0c with IP address 192.0.2.60 has been created. You
can use this address to connect to and manage the cluster.
```

Handling Multisite

a. Daemons can run across multiple sites.
b. Out of the multiple sites, one sites serves as master site and all other sites follow the master site.
c. On Master site there will be a file global_change_list which records the latest change submitted on the perforce code line and builds the change, once global change list is updated with the latest change other sites also picks up the change and start the build process at the latest change.
d. Switching of the master site is also possible, in case of the master site is down or having network issues, the site representing as master site can be switched with other site

Aggregate creations

Aggregates are classified into two groups:

- Daemon Aggregates: These aggregates are the ones for dedicated storage of the daemon workspaces, initial snapshots for the builds.

- User Aggregates: To store the user-created workspaces.

NOTE: The division of the user/aggregates is purely logical

Junction paths for daemons/volumes

Volume junctions are a way to join individual volumes together into a single, logical namespace to enable data access to NAS clients.

When NAS clients access data by traversing a junction, the junction appears to be an ordinary directory. A junction is formed when a volume is mounted to a mount point below the root and is used to create a file-system tree. The top of a file-system tree is always the root volume, which is represented by a slash (/). A junction leads from a directory in one volume to the root directory of another volume.

- A volume that was not mounted during volume creation can be mounted post-creation.
- New volumes can be added to the namespace at any time by mounting them to a junction point.
- Junction points can be created directly below a parent volume junction, or they can be created on a directory within a volume.

For example, a path to a volume junction for a volume named "vol3" might be /vol1/vol2/vol3, or it might be /vol1/dir2/vol3, or even /dir1/dir2/vol3.

For more information, see the Clustered Data ONTAP File Access Management Guide for CIFS *or the* Clustered Data ONTAP File Access Management Guide for NFS*.*

(Refer InfrastructureConfiguration.doc for details)

In case of daemons the junction path will look like below

Top level Junction path could be as follows-
/SITE/CLUSTER/DAEMON/CODELINES/WS
/SITE/CLUSTER/USER/USERNAMES/WS

2. Sur (setuid program) – for filer related access for developers.
   Control which filer commands that need to be executed by developers

3. ‘Build’ user ID will need filer/admin access.

4. Maintain same user ID permissions to ensure volumes will be owned by same user who owns the p4 client.

What is a Daemon?

Build Daemon is a job that’s kicks off a build at every pre-defined intervals. It can be scheduled to work as cron job or as Jenkins job.
Once the build daemon starts working it checks for new change committed in perforce codeline and starts the build process. The change number is first picked up by Master site and all other sites following the master site also starts the build process

A Build daemon is essentially a p4 client for each codeline that is setup at the time of the codeline creation.
The process involves syncing this client to pick up latest changes on the codeline, and run the needed builds.

Once the build is complete, a snapshot of the client is preserved. By using these snapshots as their client base instead of creating individual workspaces, developers are able to get populated, prebuilt workspaces in matter of minutes.

Some maintenance tasks to manage daemons include –purging old/non busy snapshots, managing number of clones of each snapshot

To Create a Daemon

First get information on where to store the new daemon, i.e the build servers and storage cluster information

Setup And Process Input

a. Validate name of codeline passed; validate vserver and aggregate inputs as well.

b. The *RunDaemon* process wakes up, checks for new submits and runs the required steps if there is a new change.

    i. This set of steps can be custom per branch, or be a default. Check of any custom build script provided (for *RunDaemon*)

c. Depending on where the daemon is located, pick appropriate cluster, or use user provided cluster.
d. Create the daemon’s root directory.

Create the Daemon Volume

- **Create Volume**
  CMD> ssh < /dev/null -x *<clustername>* volume create -vserver *<vs name>* *<daemon_name>* *<aggr name>* -user *<build-id>* -group *<groupname>* -size 3000g -unix-permissions 755 -snapshot-policy none -space-guarantee none 2> /dev/null

- **Mount Volume**
  CMD> ssh < /dev/null -x *<clustername>* volume mount -vserver *<vs name>* testtt_testdevvv *<Junction path to the client location of daemon>* 2> /dev/null

- **Set volume attributes**

  CMD> ssh < /dev/null -x *<clustername>* "set diag;volume modify -vserver *<vs name>* *testtt_testdevvv* -percent-snapshot-space 40" 2> /dev/null

  CMD> ssh < /dev/null -x *<clustername>* "set diag;volume modify -vserver *<vs name>* *testtt_testdevvv* -atime-update false" 2> /dev/null

- **Verify that attributes are set**

  CMD> ssh < /dev/null -x *<clustername>* "set diag;volume show -vserver *<vs name>* *testtt_testdevvv*" 2> /dev/null

Setup the Daemon Client

Create a p4 client in the client directory and mark it read only.
If a change number was provided, sync to that, else sync to tip of codeline

Setup the Daemon Directory

Set up a file that has the relevant cluster info – like filer name, vserver name, volume name, and aggregate.
This should be at the top of client root. Create the snapshot

ssh < /dev/null -x *<clustername>* volume snap create -vserver *<vs name> testtt_testdevvv <change number>* 2> /dev/null

Run Daemon process

The run daemon process essentially wakes up at fixed intervals to see if any new changes are made on the code-line.
If there are new changes, the daemon client is updated to pick up the latest changes and the needed build steps (determined at time of daemon creation) are run, and then a snapshot is preserved.
The user workspaces can then be created as clones of these snapshots, providing developers with prebuilt workspaces.

- This scripts assumes that the daemon is in good state

- If ‘nobuild’ and running are not set, it will run.

- Processes the opts

- Sync to the latest change

- This script expects the underlying build script Daemon Build to return a 0 upon successful build and a 1 upon failure.
  With a consolidated error information in a Errors file in the daemon base directory (the directory that will be snapshot)

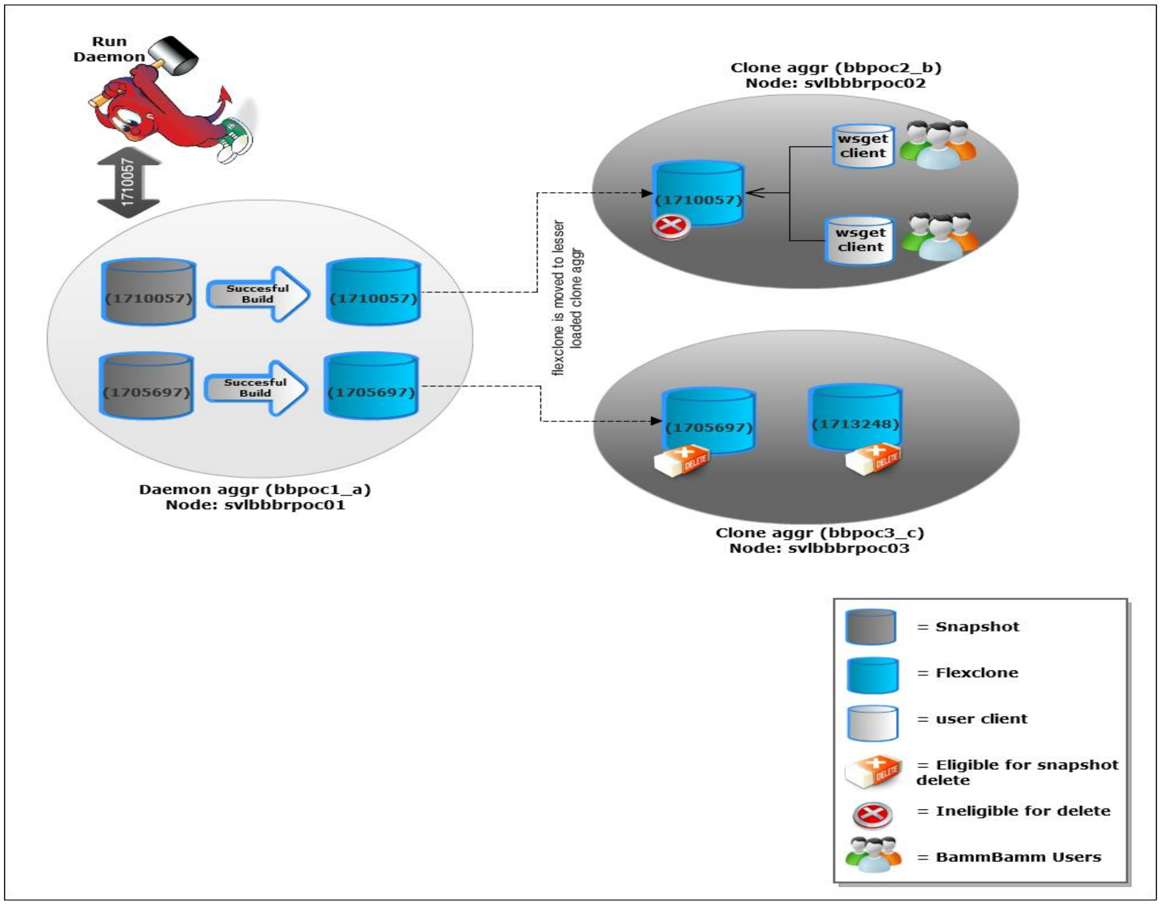
Run Daemon for busy code-lines Vs. Stable code lines

Busy Code-lines

The code-lines which has approximately more then 50 commits per can be considered as Busy Code lines. For busy code-lines

- Daemon create snapshot after every build, then a clone is created of snapshot and then clone is moved to the least loaded aggregate at that time instance.
- Developer workspaces are created of the clones moved to the least loaded aggregate this helps in load balancing and distributing the workload across the various nodes in the cluster.
- This also provides space efficiency

Attach screen shot for reference.



F

For Stable code lines:

The code-lines which has less number of commits, i.e not more then 10-15 commits per day on perforce considered as stable code-line .for Stable code line

- Snapshots created of build and will reside is the same aggregate where daemon exist, thus snapshot will exist in the same daemon aggregate.
- Developer creating the workspace of the snapshot of the latest build will also reside in the same aggregate as daemon and snapshot.
- No overhead of moving the snapshot across aggregates.
- In case of stable code Purging old/unused workspaces would be enough to maintain the storage cluster.

NOTE: To opt for stable code-line or busy code-line can defined in bb.Rundaemon as a configuration option

Setup And Process Input

Verify input parameters are valid. (Daemon name) to get the path to the daemon client.

A daemon could be setup to just sync the daemon client, or in addition-run a build-Set the value based on daemon settings stored.

If a clean build flag is set, 'clean' the daemon client. I.e. delete all prebuilt objects

1. Verify that the Daemon can run

- There is an option at cluster level to stop daemon activities. Check to ensure we are ok to proceed.
- There is also a 'no build' option for the daemon itself. If set, nothing to do.
- If there is a build still in progress, it needs to complete, so nothing to do.
- Make sure that there is a build script to run.
- Make sure that p4 is up and running.

Verify if the Daemon Should Run

Is a 'force build' flag set – then we need to run the build OR
Are there new changes since the last build? Check if client's sync point is older than latest change on code-line

Sync the new Changes into client

If there are new changes, sync them into daemon client.
After sync, ensure that there are no open files in client… error out, otherwise

Run Build Steps

- If the daemon client is 'sync only', skip this step, else use the build script for the daemon, and run it.
- If build is successful, create new snapshot.
- If build fails, also make a snapshot. But also get a list of people who submitted files since the last build and send failure notification to them

Marking daemon builds quality bench marks

To maintain the builds with the quality metrics we have testing flags that can tag with build numbers for quality parameters these are

- CIT OK (Continuous integration testing)
- BUILD_OK

CIT Continuous integration testing:

CIT provides the following business benefits:

- Developer productivity - by automatically reverting changes that break CITs, developers do not get blocked by regressions appearing in their code that are caused by code they depend on (which is maintained by other groups).

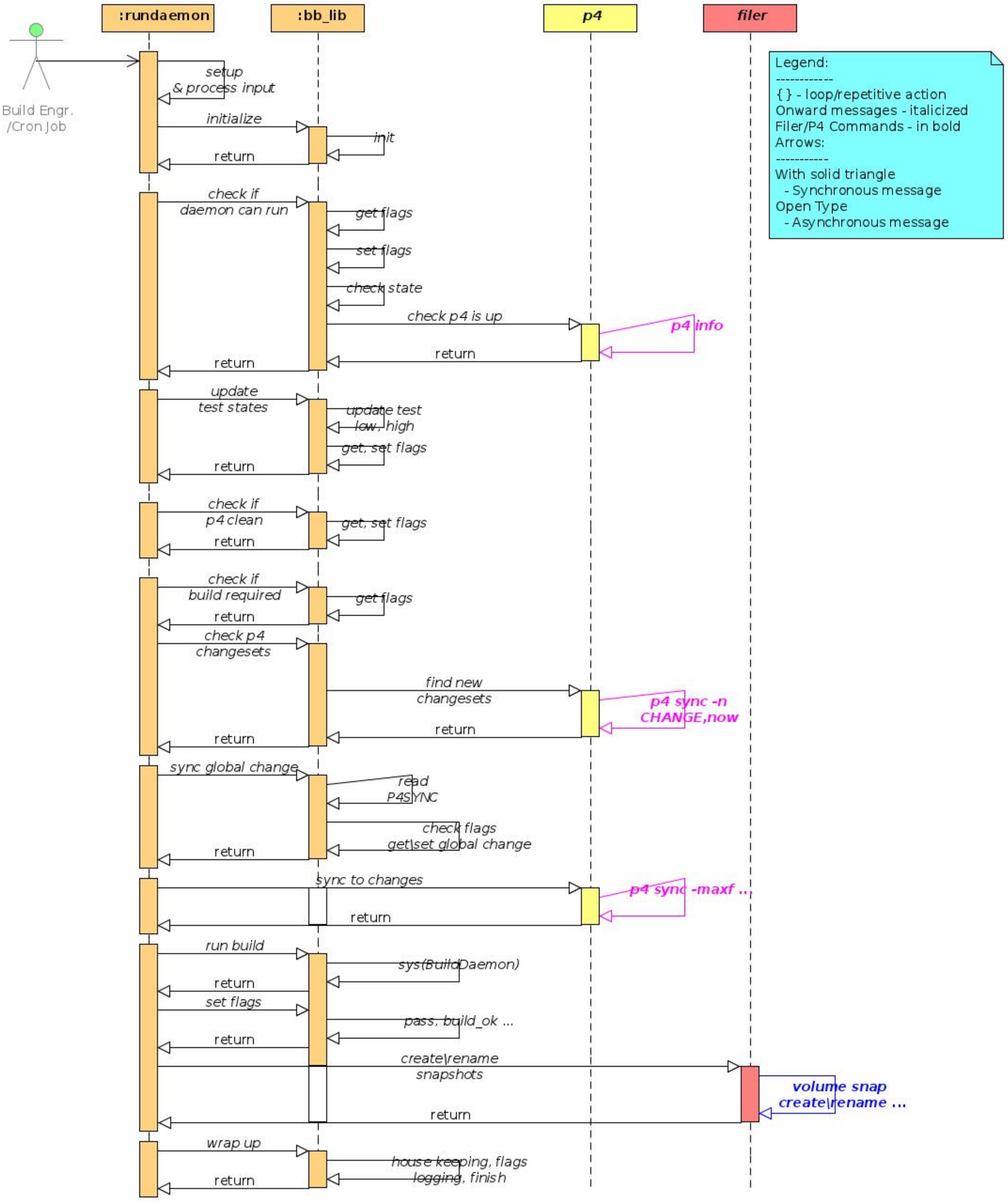
- Guaranteed level of code stability at a revision near the head of the code line - this revision is marked as "cit-ok". Groups other than development (such as QA) can use this revision of software with the knowledge that it is likely to function without any major blocking issues

- Process improvement - by keeping track of which changes are automatically reverted metrics can be gathered and fed back to improve and guide our development processes.

- When the build is marked with CIT, nightly build workflow picks up the change and build and later after successful nightly build generation build can be released for testing

BUILD_OK:

- Once the developer build completes snapshot will be created , for busy code line the volume move from daemon aggregate to user aggregate will start after the successful volume move the build will be marked as build_ok.
- For stable code-line once the developer build is complete and snapshot is created the build will be marked as build_ok


:rundaemon
:bb_lib
p4
filer
Build Engr. /Cron Job
Legend:

{ } - loop/repetitive action
Onward messages - italicized
Filer/P4 Commands - in bold
Arrows:

With solid triangle
- Synchronous message
Open Type
- Asynchronous message
setup & process input
initialize
init
return
check if daemon can run
get flags
set flags
check state
check p4 is up
p4 info
return
return
update test states
update test low, high
get, set flags
return
check if p4 clean
get, set flags
return
check if build required
get flags
return
check p4 changesets
find new changesets
p4 sync -n CHANGE,now
return
return
sync global change
read P4SYNC
check flags get|set global change
return
sync to changes
p4 sync -maxf ...
return
run build
sys(BuildDaemon)
return
set flags
pass, build_ok ...
return
create\rename snapshots
volume snap create\rename ...
return
wrap up
house keeping, flags logging, finish
return

Developer specific workflows

User workspace creation

This workflow creates a Perforce client based on a build daemon build.

Setup And Process Input

- As part of the setup, when a particular change reaches a level of acceptance (builds, passes basic tests, passes all acceptance tests, etc.), you can create tags for these 'levels'.
- Users can then get clients based off of p4 change numbers that passed a certain level of tests.

Create workspace

A user selects the daemon, which change they want, and the name for the workspace

wsget --change *<codeline>* *<change num>* *<path where client will be created>*

Process:

1) A flexclone is create off the snapshot of the requested change :

   CMD> ssh *<clustername>* ; volume clone create -vserver *<vs name>* -parent-volume *<parent vol name>* -parent-snapshot *<snapshot name>* -flexclone *<cloned client name>*

   The cloned volume name has the form <user>_<clientname>_<change>_<timestamp>

   *At netapp , clients are of the form <user>:<clientname>, the <user> is stripped by wsget since it is already included

2) Ownership of the volume is changed to that of the requestor:

   CMD> ssh *<clustername>* ;
   volume modify -vserver *<vs name>* *<cloned client name>* -comment WSGET -user *<dev user id>* -group *<groupid>* -space-guarantee none -percent-snapshot-space 50 -snapshot-policy bbuser

3) The volume is mounted to a set location (/junction path to user vol/<username>/<volume_name>):

   CMD> ssh *<clustername>* ; volume mount -vserver *<vs name>* *<cloned client name>* </junction path to user vol/<username>/volume_name>

4) A p4 client owned by user is created(E.g. devuser:demo)
   CMD> p4 -p *<PORT>* -c *<devuser:demo>* client -i *<client _spec of daemon_client>*

Here, <client _spec of daemon_client> is the spec of the daemon client- with the client name, client root and owner and host fields replaced accordingly.

5) Change file ownership to user(E.g using chown) – Note in the above step, we only have the volume and the p4 client owned by 'devuser'. Now, need to ensure all the prebuilt objects in workspace are also owned by 'devuser'.

6) p4_flush is called to inform that the current volume is a workspace owned by 'devuser' and is synced to 'changenum'. This step eliminates a sync as we already have content *@<changenum>.*
   CMD> p4_flush *<devuser:demo>* @*<changenum>*

7) We change the volume size to 500gb to ensure enough space is provided
   CMD> ssh < /dev/null -x *<clustername>* "set diag;volume modify -vserver *<vs name>* *<cloned client name>* -size 500GB" 2> /dev/null

8) A symlink pointing to the newly create volume is created within the workspace in which the script was called :
   CMD> ln -s </junction path to user vol/<username>/volume_name> /home/userid/demo

User workspace deletion

This deletes the perforce client based on a build daemon build.

Delete workspace

User workspace deletion is handled by wsdel
E.g.

wsdel <path to client>

Process:

1) Add the wsdel label to the volume (Just to mark it for deletion)

   CMD> ssh < /dev/null -x *<clustername>* "set diag;volume modify -vserver *<vs name>* *<clientname>* -comment WSDEL

2) Unmount volume

   CMD> ssh < /dev/null -x *<clustername>* volume unmount -vserver *<vs name> <clientname>* 2>&1

3) Offline volume

   CMD> ssh *<clustername>* volume modify -vserver *<vs name>* *<clientname>* -state offline

4) Delete volume

   CMD> ssh < /dev/null -x *<clustername>* volume delete -vserver *<vs name>* *<clientname>* 2> /dev/null

5) Delete p4 client

   CMD> p4 -c *<devuser:demo>* client -d -q -f *<devuser:demo>*

User workspace merge

This workflow is needed, as when the user creates a workspace at a certain change, but there are new changes being made to the codeline as the developer works in their client.

Once they are ready to submit the code, they will need to reconcile the new changes coming into the codeline. The merge workflow allows a developer to move his changes down the codeline

- Create a new wsget workspace off the latest build-ok or the specified change
  (Workspace is created using the same method as above)
- Take a snapshot of opened changes in original client
- Apply changes to new client
- Run p4 sync/flush
- Resolve any conflicts using p4 resolve

Below are some of the important features in Wsmerge :

- --change|c <num> : Explicitly specify the change number <num>
- --delete-after : workspace will be deleted after mentioned days
  - Possible values (Min of 1 to Max of 42 days)
- source : Optional argument to specify the daemon, if not given
  - the same would be determined from the source workspace
  - <product>:<daemon>
- fromclientdir : directory of the client to mergef rom

- toclientdir : directory of the newclient to merge to
    - NOTE: if <toclientdir> is not a full path, and the BBWSDIR environment
    - variable is set, the workspace will be located in <$BBWSDIR/toclientdir>.
- p4client : p4 client name created. (format: "username:name)"

Management tasks

Snapshot Purging

Background

A volume can currently hold a maximum of 255 snapshots, thus snapshot purging is needed in order for the daemon to continuously sync & build new changes throughout the day.

Process

Get a list of all active daemons within the current cluster.

Foreach daemon in found in step #1 , get non-busy snapshots for a given daemon volume

```
  CMD>:> volume snap show -vserver <vserver name>  <Daemon name>  -busy false
---Blocks---
    Vserver  Volume  Snapshot            State    Size Total%    Used%
  ----- ------- ----------------------------- -------- -------- ------ -----
    vxx     xxxx_xxx
               2117874           valid    34.75GB    1%   27%
               2117888           valid    82.62GB    1%   47%

 [......................................................................]
               2126204           valid    37.60GB    1%   29%
               2126232           valid    42.73GB    1%   31%
               2126249           valid    77.09GB    1%   45%
```

We have certain snapshots we want to keep, these special snapshots are stored in a file ex – snapshots of change numbers where certain qualification tests have passed- like builds, basic test suite, all acceptance tests, etc

Keep any snapshot defined as : EX:
BUILD_RUNNING,BUILD_OK,BUILD_DONE,TEST_BASIC,TEST_MORE,ALLTESTS_OK,TEST_MORE_PREV , ALLTESTS_OK_PREV

In addition to keeping the above snapshot, we want to keep :

i. Keep All snapshots between the latest 'build_ok' and the latest build
ii. Keep All snapshots between the latest 'test-more' and the latest build-ok , given it is less than 24 hours old
iii. Keep the latest bad snapshot before build ok

3) After the list of purge-able snapshots is collected, we delete the snapshots:

foreach $snapshot in delete list :

```
     volume snapshot delete -vserver <vs name> -volume <volname> -
snapshot <vol_daily>
```

Volume moves

Background

When a flex clone is created off a snapshot, it is created on the same aggregate in which the parent volume resides.

When several users are creating wsget workspaces of a particular snapshot, all the volumes pile up on the single daemon aggregate.

For performance and space reasons, we move all user-created workspaces to a different aggregate (a predefined "user aggregate") on a nightly basis.

Volume move happens as part of bb.Rundameon for busy code-lines
Other volume move kick off as part of daily schedule jobs on stable code-lines to move to other aggregates .
Volume move can be initiated even for moving the inactive workspace to low cost storage like SATA disk

Each cluster has a file which predefines the user aggregates and demon aggregates:

Process

Nightly volume moves run at 'off hours' daily, for each daemon aggregate:

1) Get list of user workspaces residing on the daemon aggregate and sort from oldest to newest

   i. For each workspace(flexclone) , move to least loaded user aggregate (by selecting user aggregate with least # of volumes)

   ii. Move the volume to the user aggregate
   CMD> volume move -vserver *<vs name>* *<volume>* volume *<destination_aggr>* -foreground true

User workspace purging

Background

Users often create several workspaces and forget to delete old workspaces, thus we recommend a workspace retention policy

Process

Periodically, during off hours, for each user workspace on cluster:

A) Check for config file located in workspace (format: <timestamp in epoch seconds >:# of days to be preserved),
Mark for deletion if it has been 14 days since expiration date, else send warning email

B) Check builds activity in client.
If build has not been run in 45 days:

1) Take copy of workspace (open files) and store it under user's home directory
2) Delete the volume:

```
volume delete -vserver <vs name> -volume <volume_name>
```

If a workspace is between 28 && 45 days old:

1) Send an email warning
2) Total 14 warning email messages being sent to the developer/owner to take action on the work-space
    a. If developer decide to keep the workspace, test build execution in the workspace will help.
    b. Owner of the workspace can also delete the ws using wsdel <ws full path>
    c. After 45 days the workspace will be deleted

P4 CLI SNAP

- P4 cli_snap can we used for backing up of the opened files in the workspaces where the workspace can later be deleted results in saving the storage space.
- Cli snap can also be used in disaster recovery for the perforce clients.
- A cli_snap script that would be run once a day during local non-peak hours (during the middle of the night). The cli snaps will be rolled over every day, so that we will NOT have multiple cli snaps for the same client taking up space.
- The script will go through every workspace under the /setup/bb<site>/scratch directory

  If the p4 client has open and or pending files, run "p4 cli_snap > <archive>"

  The archive file will be named as : <clientname>.clisnap
- The clisnap will be run on a linux build server and hence the cli restore should be run on the same platform.

NOTE: If the user has non p4 files (files that were not yet added to p4) in the workspace, these will not be saved.

Recovering from a cli_snap into a fresh client

- Run "wsget" and get a new client.
- cd to the newly created workspace dir.
- On a Linux machine: run "p4 cli_restore -n **<** *{Full_pathname_to_the_archive_file}* "
    - **Notice "<" , this is mandatory after p4 cli_restore -n , else p4 cli_restore will hang**
- When the cli_restore completes successfully, run "p4 sync @changenumber" where changenumber is the change in the P4SYNC file in the new workspace directory.
- Once the sync is complete, then run "p4 resolve" and go through the resolve process you would with any merged client.

Moving Daemons

Background

This script is used to move the daemon volume to a different aggregate
We move the clones instead of split because clone split takes much longer time than clone volmove

Ex:
MoveDaemon *<daemonname>* --fromaggr *aggr1* --toaggr *aggr2* --clonemoveaggr *useragg1* --vserver *<vs_name>*

1. **Setup And Process Input**
2. **Move Daemon Clone Volumes**
    a. Finding clones for daemon volume -Get Clone List For the Parent Volume
    CMD> ssh < /dev/null -x *<clustername>* "set diag;vol clone show -instance -parent-volume *<daemon_name>*"

    b. Moving clone volume to <destination aggregate>
    CMD> ssh < /dev/null -x *<clustername>* volume move start -vserver *<vs_name>* -volume *<clone_volname>* -destination-aggregate user*agg1* -foreground true

3. **Move Daemon Volume**
    CMD> ssh < /dev/null -x *<clustername>* volume move start -vserver *<vs_name>* -volume *<daemon_name>* -destination-aggregate *aggr2* -foreground true

- In order to minimize daemon downtime and have the old daemon still intact while the new one is created, we will create a new daemon with a different name and then rename it to the old name later.
- To do this, we will create a new daemon using a private copy of bb.CreateDaemon modified to create a p4client with a name to contain <builduser>:<site>:bb2:<olddaemonname>. (e.g: For a newdaemon with a name "mainng1", p4 client would be build:svl:bb2:mainng)

- Our intention is to keep the same daemon name and hence the change to the p4client name. (Otherwise, p4client would be created with <builduser>:<site>:bb:<newname>.
- Create the daemon at a changelist before the current test_ok changelist from the globalsync file.
- Now disable the old daemon. (bb_admin - nouse, nobuild).
- Rename old daemon volume to <daemonname.old> (e.g: mainng to mainng.old)
- Unmount the old daemon (e.g: unmount from //share/Setup/<path> /daemon_base)
- Rename new daemon volume to <daemonname>. (e.g: mainng1 to mainng)
- Unmount the new daemon
- Mount it again at the original old daemon location. (at //share/setup/<path>/daemon_base).
- cd to daemon workspace at /setup/bb<site>/daemon/ontapng/<daemon>/daemon_base and update the p4 client root to point to this directory.
- Update the .bbinfo file in the daemon to point to the correct (renamed) volume name. (e.g: from mainng1 to mainng)

Management Scripts

Along with the daemon creation, running daemon etc Setup also has many others features via the listed scripts which provide the valid information/data without having the sudo privileges:

Use bb_admin for administrative tasks associated with daemon

```
    bb_admin <product>:<daemon> [flags]
             --nobuild <comment>         : set nobuild flag       (0 to unset)
             --nouse <comment>           : set nouse flag         (0 to unset)
             --nomgmt <comment>          : set nomgmt flag        (0 to unset)
             --inactive <comment>        : set inactive flag      (0 to unset)
             --forcebuild <comment>      : set forcebuild flag    (0 to unset)
             --p4clean <comment>         : set p4clean flag       (0 to unset)
             --changenum <changeno>      : set p4clean change num (0 to reset)
             --component <component>     : set p4clean component  (0 to reset)
             --failure-mailto <comma seperated email ids>
                                         : set failure-mailto     (0 to unset)
             --success-mailto <comma seperated email ids>
                                         : set success-mailto     (0 to unset)
             --daemon-running <changeno> : mark daemon-running    (0 to unmark)
             --build-running <changeno>  : mark build-running     (0 to unmark)
             --build-done <changeno>     : mark build-done        (0 to unmark)
--nobuild <comment>         : set nobuild flag       (0 to unset)
             --nouse <comment>           : set nouse flag         (0 to unset)
             --nomgmt <comment>          : set nomgmt flag        (0 to unset)
             --inactive <comment>        : set inactive flag      (0 to unset)
             --forcebuild <comment>      : set forcebuild flag    (0 to unset)
             --p4clean <comment>         : set p4clean flag       (0 to unset)
             --changenum <changeno>      : set p4clean change num (0 to reset)
             --component <component>     : set p4clean component  (0 to reset)
             --failure-mailto <comma seperated email ids>
                                         : set failure-mailto     (0 to unset)
             --success-mailto <comma seperated email ids>
                                         : set success-mailto     (0 to unset)
             --daemon-running <changeno> : mark daemon-running    (0 to unmark)
```

```
          --build-running <changeno>  : mark build-running     (0 to unmark)
```

Storage Management

Along with timely volumes purging/cleanup ,also need to monitor space utilization across nodes in cluster.
For storage cluster:

- ➢ A storage cluster can have maximum 24 nodes.
- ➢ Each node can have 100 volumes maximum.

  bb_cluster tool can give a view on current storage utilization. bb_cluster can provide the storage utilization for daemon and user aggregates.
  Bb_cluster can also be used to manage cluster ,i.e adding /deleting nodes/aggregates

e.g
User Aggregates:

```
AGGR    AVAIL (GB)  FILERMODEL  %USED  NUMVOLS  DISKTYPE(S)   DAEMONS

bb01_a  13060      FAS6280     83    628

bb02_a  13370      FAS6280     83    614

bb03_a  13360      FAS6280     83    628

bb04_a  9170      FAS6280     84    498
```

Syntax:

- ➢ bb_cluster [flags]
- ➢ --nobuild <comment>   : set nobuild flag   (0 to unset)
- ➢ --nouse <comment>   : set nouse flag   (0 to unset)
- ➢ --aggr-set-user <aggr>   : add aggr to useraggr list
- ➢ --aggr-set-daemon <aggr>   : add aggr to daemonaggr list
- ➢ --aggr-set-daemon-clone <aggr> : add aggr to daemon_clone_aggr list
- ➢ --aggr-set-inactive <aggr>   : add aggr to inactiveaggr list
- ➢ --disktype   : flag that triggers the inclusion

Advance admin workflows

Where are the daemon logs:

- Under <setup-path> directory and follow through the dir hierarchy of product/daemon as applicable. For example,

- bb.RunDaemon logs are under setup-path daemonname>/<date>.log
- bb_admin_state logs are under / setup-path /bb_admin_state/<product>/<date>.log
- bb_admin logs are under / setup-path/bb_admin/<product>/<date>.log

How to set nobuild to a daemon:

Remove path

- Run /bb_admin <Productname>:<daemonname> --nobuild <comment>";

How to set nobuild to ALL of the daemons at a given site in one sweep:

- Run "bb_cluster --nobuild <comment>";

How to force a build in a daemon when there are no new changes:

- Run bb_admin <Productname>:<daemonname> --forcebuild <comment>"; ® Eg:- "/ bb_admin DOT:Rboilermaker --forcebuild "<comment>"";

- The build behaves exactly as any normal changelist build. The only difference is that this will force the build to the last built changelist even when there are no new changes.

How to start a build from scratch in a daemon: (Run p4clean in the daemon workspace)

- Run "bb_admin <Productname>:<daemonname> --p4clean <comment>";

- This will run p4clean before the next build is run, essentially simulating starting from scratch.

NOTE

- You can combine --p4clean & --forcebuild flags to get a build from scratch even if there are no new submissions.

Where are the daemon changelist directories?

- Under /<path>daemon_base/.snapshot/<changelist> ® <path>/daemon_base/.snapshot/<changelist>

If one site has fallen behind in builds, and we would like to catch up to head of line

- By design, daemon builds will run at the same change levels at all sites.

- If for some reason, one of the sites falls behind and you want to skip the intermediate builds and just skip ahead to head of line, you will need to edit the globalsync file at:

/<path>/setup/global/<product>/<daemonname>/globalsync

DO NOT DO THIS UNLESS THE FOLLOWING CRITERIA ARE MET:

- You know for sure the changelist has failed at multiple sites, you can remove it.
- You know for sure the changelist will NOT become the next test_ok, you can remove it. But this one has to be made sure with the higher powers.

To clean up and remove old daemons

- Use bb.DeleteDaemon to remove daemons which are no longer required. bb.DeleteDaemon will only work on daemons created after Rboilermaker. Every daemon created before Rboilermaker including DOT:main would need to be deleted manually.

```
$ bb.DeleteDaemon -help
Description:
  bb.DeleteDaemon deletes an existing daemon & its associated p4client after detaching
the possible clone volumes associated with the daemon.
Syntax:
   bb.DeleteDaemon <product:daemon> [flags]
     --help         : Prints this usage message.
     --filer        : Filer name.
     --vserver      : vserver name
     --clonemoveaggr : aggregate name to move the daemon clones to
```