zCustom.SnapBackup

Version History

* 20131121 – tidy up, Glen Pitt-Pladdy (InterSystems)
* 20131212 – added JournalSwitch functionality, Glen Pitt-Pladdy (InterSystems)
* 20131219 – added LVM2 warning level, Glen Pitt-Pladdy (InterSystems)

# Important

These tools are developed for the needs of the TrakCare UK Infrastructure team and reflect conventions and approaches used to meet the needs of deployments in the UK. They are released on the basis that they may be useful to others within InterSystems.

The development process is on an ad-hoc basis, generally based on adding or updating tools as needed for deployments. There is negligible testing and no QD process nor any sort of managed release process. These are simply rolling snapshots of work in progress and may be in a severely broken state.

Use at your own risk.

# Backup Strategy

Backups are a critical part of any robust business continuity plan, yet widely neglected. When all else fails, backups are the final mechanism for recovery so it is imperative that they can be used to restore systems to fully operational.

This document aims to set out the minimum that should be aimed for when backing up systems running TrakCare. Individual sites should evaluate their specific needs and implement whatever policies and procedures necessary to extend this to a level that they require.

## General good practice

The site “Tao of Backup” (<http://www.taobackup.com/>) lays down fundamental principles of good backup practice (except the last item which is marketing which they are quite clear about). Many are not technical but procedural, however can still be helped by good technical approaches. These are:

### Coverage – backup ALL data

It’s common that only specific “important” data gets backed up and other data excluded from backups, however, while this may be technically sufficient to rebuild the system, in a recovery situation the time wastage for rebuilding the system is a major problem. Additionally, there may be seemingly unimportant data and files that could still be necessary for the running of the system. If these are not backed up they might take time to discover/debug after restore, and even more time recreating them before systems are fully operational again.

There is also considerable detailed configuration and updates that may have been applied all over a system, and the task of rebuilding a system that has been running for some time and has considerable history could be far larger than building a completely new system. This could effectively make recovery a major project on a similar scale to implementing new deployments.

Considering that TrakCare databases are by far the largest volume of data (even on the smallest sites) there is no real reason to not capture the entire system in a backup and have greater certainty and ease of restore.

### Frequency

The time between backups determines how much data may be lost should a restore be done. Typically daily backups are done which is reasonable for the entire databases, however, say hourly backups of journals could be done to reduce the window for loss further.

One good practice is to ensure backups happen automatically. This avoids human error (eg. forgetting to run a backup)

### Separation – ensure at least some backups are off site

If a major disaster occurs (eg. major fire, floods etc.), the site may be destroyed or at least rendered unusable for extended periods of time and backups which are kept sufficiently close to the systems which they backup are likely to also be lost, damaged or inaccessible.

Another factor that is often neglected in this respect is the loss of backup hardware (eg. tape drives). Formats change rapidly and occasions occur frequently where finding a drive to read old media is near impossible. While it may be possible to trawl eBay or system recycling companies for weeks to find a compatible drive second hand, this is not a scenario that is practical in a disaster situation where rapid recovery is necessary. It’s important to ensure that there is a means to read the backups available, even if this means storing a spare backup drive(s) off site as well.

This is not a technical aspect of a good backup strategy, but none the less needs considering by the site or ISC if we are responsible for running the systems.

### History

If gradual corruptions occur or historic data is needed for legal reasons then having old backups of data can be vital. A suitable retention policy and the storage necessary to fulfill that policy needs to be taken into account.

This is not a technical aspect of a good backup strategy, but none the less needs considering by the site or ISC if we are responsible for running the systems.

### Testing – test backups regularly

This is one of the most widely neglected aspects of good backup practice. There are several reasons why this is vital:

* You can’t be sure that backups are working unless you have tested the end-to-end goal: full restore
* You can’t be sure that backups are **still** working unless you test regularly. While initial verification may tell you the initial configuration is good a number of factors can still invalidate backups later:
  + OS upgrades/patches introducing bugs (eg. a known case of an encryption bug which was introduced in an upgrade and was only apparent after systems had been rebooted and encryption re-initialized, rendering backups unreadable)
  + Faults developing with backup hardware that resulted in undetected corruptions to backups
  + Deterioration of media resulting in it becoming unreadable some time after backups were made
  + Configuration changes to backups or the system that invalidate backups

As mentioned under “Separation”, it may be necessary to store a spare drive off-site as well to ensure the backups can be read. Mechanical parts may require periodic use to ensure they operate correctly (eg. circulate lubricants and prevent ceasing) so testing spare backup drives and other equipment is also an important part of this.

### Security

Backups contain sensitive data (in the case of TrakCare, likely the most sensitive civilian data). Precautions need to be taken to ensure that this data does not fall into the wrong hands while still being available when it is needed. Precautions that may should be considered include:

* Encrypting backups – note that the same good practices will also be needed with encryption keys (ie. don’t store them with the backups nor on site)
* Secure shipping and storage of backups – unauthorized physical access to backup media needs to be avoided both for security and to avoid harm to media

### Integrity

There is little point in having backups of corrupted data. Caché has the ability to integrity-check databases which is a good idea to do periodically.

Like with “Frequency”, this should be an automated process. It’s often best not to do integrity checks of all databases with every backup as this can consume considerable IO resources and as databases get bigger can run for a long time, in some cases the time being so long that backups can’t complete in time for the next one. A reasonable frequency may be once a week for each set of databases – ie. split databases up and do some every night or some Saturday night, some Sunday night or similar strategy.

## Capture Mechanism

In light of these good practices, a set of criteria for running backups can be determined:

* Simple – it’s always worth avoiding complex systems which are more prone to faults as well as human error
* Complete – ALL data and files must be captured and restorable
* Easy to restore – when under pressure in a recovery situation, the ease which systems can be brought back into operation is vital
* Low or No Downtime – critical systems like TrakCare should not be taken down unnecessarily. The mechanism must allow the system to continue running with minimum impact from the backup

Caché Online Backup fails the criteria:

* Complete – TrakCare (and the system as a whole) has considerable data that will not be captured by Online Backups
* Easy to restore – The system will not be able to be restored as a whole, and system databases which contain important changes (eg. for Version Control) will be non-trivial to restore

Snapshot Backups fulfills the criteria:

* Simple – can be backed up as a normal filesystem would be and all backup systems can handle this easily
* Complete – as it’s a filesystem backup in the end, the entire system can be captured and restored using this mechanism
* Easy to restore – the entire system can be restored in one shot (or selectively if required), data moved into place, bootloader installed and the system rebooted to bring it back into operation. At this point additional journals could also be applied if needed.
* Low or No Downtime – Snapshots require a brief freeze/thaw of the Write Daemon, but otherwise the system can remain fully operational. It’s worth noting the risk where synchronous mirroring is in use of this tripping a failover, however to date synchronous mirroring is not used for UK TrakCare deployments with OS/VM based HA used instead.

## Snapshot Mechanisms

Depending on the hardware and platform design, there are multiple ways of achieving backups in this way.

### Linux LVM2

All major Linux distributions support this and it provides considerable flexibility as well as a simple to use (single command and quiesces filesystems automatically). It does however have a few warts on some distributions which while easy to work around, none the less need awareness.

The typical scenario to take snapshots (with the Write Daemon frozen) would be:

# lvcreate --name snapshotname --size sizeGB /dev/volumegroup/targetlogicalvolumename

This will produce a ready-to-mount logical volume with the name specified which is a snapshot frozen in time of the target logical volume. Additionally this snapshot is writable which allows “cleanup” scripts to be run on it before it is backed up.

If the size of the snapshot is filled (keeping track of changes against the target volume) the OS automatically destroys it. The original target volume is unaffected. In order to ensure that the snapshot works as expected and the backups can run completely, sufficient size must be chosen so that all the changes occurring while the snapshot is in use can be captured.

While the snapshot is in place there is some write performance impact as any changes have to be written to the snapshot volume (Copy on Write), however unlike some SAN mechanisms there is no catchup/backfill phase after the backup is done.

Removal of the snapshot is similarly simple:

# lvremove /dev/volumegroup/snapshotname

At the point of writing there are race conditions present in some major Linux distributions which intermittently prevent removal of the snapshot and these have to be worked around. The problem appears to relate to udev events destabilizing deactivating the volume. To work around this manually remove the device-mapper first:

# dmsetup remove /dev/volumegroup/snapshotname

Then remove the logical volume:

# lvremove /dev/volumegroup/snapshotname

This process leaves behind the COW device-mapper which then also needs manually removing:

# dmsetup remove /dev/mapper/volumegroup-snapshotname-cow

**IMPORTANT:** In order to be able to use this mechanism, sufficient space needs to be reserved in the Volume Groups to create the snapshot volumes used for backups.

### SAN Snapshots

Leading SANs also provide snapshot mechanisms that may be used for backups. In this case the tools for that specific SAN will need to be used.

A typical process for snapshots (assumes Write Daemon already frozen – this is just the snapshot process) would be:

* Quiesce the filesystem(s)
* Create SAN snapshot
* Un-quiesce filesystem(s)

The snapshot volumes may then be published and mounted for backup.

After backup they are simply unmounted and removed.

### SAN Mirror Split

Another approach possible with SANs is to run a mirror. In this case the process followed (assumes Write Daemon already frozen) would be:

* Quiesce the filesystem(s)
* Break SAN Mirror, the active system remaining running on one half of the mirror
* Un-quiesce filesystem(s)

The inactive part of the mirror can then be used the same as with Snapshots: published and mounted for backup.

After backup they are simply unmounted and resynced back into the mirror.

## Regular Journal Backups

A strategy used by some TrakCare sites is to run regular (say hourly) backups of Journals which can then be applied after restore to minimize the time window for data loss between backups.

This does also require care as there is known to have been problems when an open Journal file is applied due to it not being complete. For this reason it's probably best that if this approach is being used, to switch Journal files before the backup.

# zCustom.SnapBackup Tools

This is a Caché Object Script Class and example shell script to call in and use this backup mechanism. It is important that the site thoroughly tests all components of the backup system to ensure that they are working as expected. This includes testing that faults are detected correctly (eg. remove the Class and/or shut down Caché and ensure the problem is detected and notified by the system).

## zCustom.SnapBackup Class

This class contains Class Methods to control the Write Daemon Freeze/Thaw process and perform sanity checks to ensure it’s been successful, reporting back the status via a file (avoids any problems passing status via csession).

This Class should be loaded into the %SYS Namespace.

### ClassMethod ReportStatus(StatusFile As %String, Contents As %String, Level As %Integer) [ Private ]

This writes the status file for communicating reliably with external scripts calling in. The contents of this file should be a single line in the format:

* “OK” – everything has succeeded cleanly
* “WARNING - some warning message” – this should be used when the Write Daemon has been successfully frozen (safe to proceed), however some unexpected things may have happened
* “ERROR - some error message” – this should be used when the Write Daemon has not been frozen or there is reason to suspect integrity loss and it is unsafe to continue
* “FATAL - some error message” – this should be used when a catastrophic failure has occured and it is unsafe to continue

Without this it is not possible to reliably communicate status with calling scripts:

* Returns from Freeze/Thaw methods do not reliably return status on all versions of Caché currently in use
* csession returns 0 (Unix success) on successful connection, irrespective of successfully executing a command in Caché

### ClassMethod Freeze(StatusFile As %String, TimeOut As %Integer) As %Status

This calls the ##class(Backup.General).ExternalFreeze() method, checking status and confirming that the operation has been successful. It then calls the WriteStatus Class Method to write the status file to the path specified.

### ClassMethod Thaw(StatusFile As %String) As %Status

This calls the ##class(Backup.General).ExternalThaw() method, checking status and confirming that the operation has been successful. It then calls the WriteStatus Class Method to write the status file to the path specified.

### ClassMethod History(StatusFile As %String, LogFile As %String = "") As %Status

This calls the ##class(Backup.General).ExternalSetHistory() method, checking status and confirming that the operation has been successful. It then calls the WriteStatus Class Method to write the status file to the path specified.

It is intended that this is used to ensure that normal operation of Journal purging be maintained by adding the record of Snapshot backups into the Backup History.

### ClassMethod JournalSwitch(StatusFile As %String) As %Status

This calls the ##class(%SYS.Journal.System).SwitchFile() method to switch Journal files, checking status and confirming that the operation has been successful. It then calls the WriteStatus Class Method to write the status file to the path specified.

It is intended that this can be used for switching Journals prior to a regular (say hourly) backup of journals.

## zCustom.SnapBackup.sh Call-in Shell Script

./zCustom.SnapBackup.sh <Freeze|Thaw|History|JournalSwitch> <Instance|\_ALL> [Additional Options]

This is an example Call-in shell script for calling the Class Methods above.

This assumes a user “cachebackup” exists and is configured for Operating System Authentication within all Caché instances this is to be used with. It returns 0 on success and 1 on failure keeping in line with Unix conventions. Both these are configuratble with variables in the top of the script.

The script is called with the operation (Freeze, Thaw, History or JournalSwitch) and the instance name. If \_ALL is used as the instance name then it uses “ccontrol qlist” to determine all registered instances and will perform the operation for all of them.

The script requires all operations to succeed in order for it to succeed, however will always try to execute all operations on the specified instance(s). This ensures that a failure on one instance does not affect others where aborting on the first error may cause further problems in that the operation is then not attempted on other instances where it would succeed. This ensures that the maximum level of integrity is maintained.

Status files are created in the Temp/ directory for each instance and should be removed afterwards. If for any reason the script fails in an unexpected way these files will remain and will be in the format: zCustom.SnapBackup-YYYYMMDD-hhmmss-PID-Operation\_status.txt

### History Function

./zCustom.SnapBackup.sh History <Instance|\_ALL> [Path to a logfile]

The History function updates the Backup record in Caché which is used for Journal Purging and will normally be called with the successful completion of the backup. When the History function is called it may be called with an optional path to a logfile for the backup which will then be made available to view via the SMP.

## snaplib.sh Utility Library / Include

This is a library that may be used to write the Pre- and Post- scripts for backups and includes procedures for various tasks that are done frequently as well as setting sane defaults.

An example of including this would be:

#!/bin/sh -e

. `dirname $0`/snaplib.sh

………………

This should be included first in the scripts to ensure that default variables can then be overridden.

### $CALLIN

This sets the path to the Call-in script (above) that is used to Freeze/Thaw the Write Daemon. It defaults to the same path as the calling script and the default naming of the example Call-in script.

### $EXITFAILURE

Sets the value the script will return on failure. This is normally 0 in keeping with Unix convention.

### $EXITSUCCESS

Sets the value the script will return on success. This is normally 1 in keeping with Unix convention.

### $WORKAROUNDLVM

This sets if workarounds for race conditions with removal of logical volumes on some Linux distributions should be used and is normally set to 1.

### LVMWARNINGLEVEL

This sets the % usage of the LVM2 snapshot when it is remove that will trigger a warning about safety margin. The aim is to alert admins that a significant amount of snapshot space is being used and there is a risk of the snapshot becoming full before the backup completes.

### LVMWARNINGFIAL

If non-zero this will cause linuxlvremovesafe() to return 1 when the above warning level is met. The intention is to draw attention to the problem. This is set to 0 (disabled) by default.

### $MOUNTPOINT

This sets the mountpoint where backup volumes will be mounted and is set to our convention of “/backup”

### heading()

heading HeadingString

Prints formatted heading lines – just standardized for neatness

### makemount()

makemount Device Path ….other options

This creates the path given and mounts the device on that passing through any other arguments through to the mount executable.

### umountall()

umountall Mountpoint

This will parse /proc/mounts and unmounts all mountpoints found below the specified mountpoint. This is useful to automatically unmounts all the mounts on the backup staging area after backups have completed.

### ismounted()

ismounted Mountpoint

This will parse /proc/mounts and counts all mountpoints found below the specified mountpoint. This is useful for determining if the backup staging area is currently in use (ie. avoid the script being called multiple times).

### linuxlvremovesafe()

linuxlvremovesafe Device

This removes a Linux LVM2 logical volume, working around race conditions in some distributions if $WORKAROUNDLVM is set to 1 (see above).

### exitwithstatus()

exitwithstatus $backuperror

This will exit the script with the translated exit status based on the argument it’s given. The argument should be 1 for failures, 0 for success. The values of $EXITFAILURE and $EXITSUCCESS are used along with messages being output.