[INSTSERV][qxz10kv][lpinstiaas02] ~ $ cat /global/instserv/bin/install\_override.sh

#!/bin/bash

#set -x

if (( ${BASH\_VERSION/.\*/} < 4 )); then

echo ":: SLES11: Re-executing with bash 4"

## We honor if the -x for debugging is set and forward it to our reincarnation

if [[ $- == \*x\* ]]; then

X="-x"

else

X=""

fi

set -x

exec /global/instserv/bin/.testssl.sh/.bash.4.4 ${X} $0 $@

fi

# Define some variables

MYHOME=$(dirname $(readlink -f $0) | sed 's#/bin##')

MYNAME=$(basename $0)

MYPID=$$

MYTTY=$(/usr/bin/tty | sed 's#/dev/##g')

export EXECUSER=$( (echo $SUDO\_USER; echo $SU\_USER; logname 2>/dev/null;) | grep -v "^$" | head -n 1)

OVERRIDEDIR=$MYHOME/overrides

OVERRIDELIST=$(find $MYHOME/overrides/\* -maxdepth 0 -type d -exec basename {} \; | egrep -v "^accounts$|^local$")

ACCOUNTLIST=$(find $MYHOME/overrides/accounts/\* -maxdepth 0 -type d -exec basename {} \; 2>/dev/null)

SPECIALLIST=$(find $MYHOME/overrides/special/\* -maxdepth 0 -type d -exec basename {} \; 2>/dev/null)

LOCALLIST=$(find $MYHOME/overrides/local/\* -maxdepth 0 -type d -exec basename {} \; 2>/dev/null)

MD5SUMFILE=MD5SUM

SHORTERROR=0

DEBUG=0

SKIP\_PREPARE=0

STOREINDB=1

YES=0

MERGE\_STDERR=1

NO\_LOG=0

CONTROL\_MASTER=1

DUMMYSEPARATOR="##############################################################################################"

TGTHOST=""

if ! test -x $PWD; then

cd "/global/instserv/bin"

echo "\*\* Warning: called from not traversable directory, options that provide files on CLI do not work this way!"

fi

[[ -f /etc/profile.d/instserv.sh ]] && source /etc/profile.d/instserv.sh

COMMON\_FRAMEWORK="$MYHOME/bin/remote.lib.sh"

if ! ERR=$( bash -n "$COMMON\_FRAMEWORK" 2>&1 ); then

echo "=> Error in common framework '$COMMON\_FRAMEWORK'!"

echo "$ERR"

COMMON\_FRAMEWORK=""

fi

source "$MYHOME/bin/ssh.lib.sh"

INST\_OV\_ERRORS=(

[1]="Invocation error"

[2]="Input missing"

[3]="Already running"

[4]="Aborted by user"

[66]="SLES12 SP2 Host hit"

[100]="MD5 integrity failed"

[101]="Server timed out"

[102]="Multi-Install Error"

[103]="Prepare script aborted"

[104]="Params script aborted"

[105]="Rsync copy failed"

[106]="Main Script failed"

[107]="Postscript failed"

)

if [[ $EUID != 0 ]]; then

echo ":: Running in non-root context, using privilege elevation via sudo"

SUDO\_CMD="sudo --non-interactive"

else

SUDO\_CMD=""

fi

# If variable INSTSERV is not set, use the current hostname

[ -z "${INSTSERV}" ] && INSTSERV=$(uname -n)

# The DB name is derived from the name of the install server

DB=/global/instserv/database/${INSTSERV}.db

# If ORACLE\_HOME is not set and /etc/profile.d/oracle.sh exists, source it

[ -z "${ORACLE\_HOME}" ] && [ -f /etc/profile.d/oracle.sh ] && . /etc/profile.d/oracle.sh

function CallDB {

local QUERY="$\*"

echo -e ".timeout 5000\n${QUERY};" | sqlite3 ${DB}

}

echo "$(date) - $EXECUSER - $(basename $0) $@" >> $MYHOME/logs/install.log 2>/dev/null

# If you provide -r as first parameter, do a removal instead of installation (configure)

# -i (or none): install

# -r : remove

# -c : check

case ${MYNAME} in

remove\_override.sh) ACTION="remove"

ACTOPT="-r"

;;

check\_override.sh) ACTION="check"

ACTOPT="-c"

;;

\*) ACTION="configure"

;;

esac

while [ -n "$1" ]

do

# if PARAMETER $1 contains =, set PARAM to first part (in front of =), OPTION to rest

if echo "$1" | grep -q "="

then

PARAM=$(echo "$1" | cut -d '=' -f 1)

OPTION=$(echo "$1" | cut -d '=' -f 2)

else

PARAM="$1"

OPTION="$2" # maybe this is not required, but we do not know beforehand

fi

case ${PARAM} in

# -h, -? or --help

#-h|--help|-\?)

# Usage

# exit 0

# ;;

-r|--remove)

ACTION="remove"

ACTOPT="-r"

shift

;;

-c|--check)

ACTION="check"

ACTOPT="-c"

shift

;;

-i|--install)

ACTION="configure"

shift

;;

--yes)

YES="1";

shift;

;;

--info)

ACTION="info"

ACTOPT="--info"

TGTHOST="none"

shift

;;

--special)

SPECIAL="special"

shift

;;

--short-error)

SHORTERROR=1

shift

;;

--debug)

DEBUG=1

shift

;;

--no-db)

STOREINDB=0

shift

;;

--no-control-master)

CONTROL\_MASTER=1;

shift;

;;

--skip-prepare)

SKIP\_PREPARE=1

shift

;;

-f|--file)

INPUT\_FILE=${OPTION}

if [ ! -r "${INPUT\_FILE}" ]

then

echo "Input file ${INPUT\_FILE} not found"

exit 2

else

TGTHOST="FILE"

shift

echo "$1" | grep -q "=" || shift

fi

;;

--no-logfile)

NO\_LOG=1;

shift;

;;

--no-stderr-merge)

MERGE\_STDERR=0;

shift;

;;

-\*)

echo "ERROR: Unknown command/flag/option ${PARAM}"

exit 1

;;

\*) if [ -z "${TGTHOST}" ]

then

TGTHOST=${PARAM}

elif [ -z "${OVERRIDE}" ]

then

OVERRIDE=${PARAM}

else

PARAMETER="${PARAMETER} ${1}"

fi

shift

;;

esac

done

source "$MYHOME/bin/trap.lib.sh"

if ((NO\_LOG != 1)); then

source "$MYHOME/bin/logging.lib.sh"

fi

# if we got several hostnames separated with comma, execute this script for each of them

if echo "${TGTHOST}" | grep -q "," || [ "${TGTHOST}" = "FILE" ]

then

MULTIINSTALLRESULT=""

MULTIINSTALLRC=0

# Preserve given parameters

if [ "${SPECIAL}" = "special" ]; then SPECIALOPT="--special"; else SPECIALOPT=""; fi

if [ ${SHORTERROR} -eq 1 ]; then SHORTOPT="--short-error"; else SHORTOPT=""; fi

if [ ${SKIP\_PREPARE} -eq 1 ]; then SKIPOPT="--skip-prepare"; else SKIPOPT=""; fi

if [ "${TGTHOST}" = "FILE" ]

then

TGTHOST=$(cat ${INPUT\_FILE})

fi

for SINGLEHOST in $(echo "${TGTHOST}" | tr ',' ' ')

do

OPTIONS=$(echo "${ACTOPT} ${SPECIALOPT} ${SHORTOPT} ${SKIPOPT}" | tr -s ' ' ' ')

MULTIINSTALLRESULT=$(printf "%s\n%-75s " "${MULTIINSTALLRESULT}" "install\_override.sh ${OPTIONS} ${SINGLEHOST} ${OVERRIDE} ${PARAMETER}" )

$0 ${OPTIONS} ${SINGLEHOST} ${OVERRIDE} ${PARAMETER}

if [ $? -eq 0 ]

then

MULTIINSTALLRESULT=$(printf "%s %s\n" "${MULTIINSTALLRESULT}" " OK")

else

MULTIINSTALLRC=102

MULTIINSTALLRESULT=$(printf "%s %s\n" "${MULTIINSTALLRESULT}" "NOK")

fi

# If a prepare skript exists, we only execute it the first time

SKIPOPT="--skip-prepare"

done

echo

echo "${MULTIINSTALLRESULT}"

echo

exit ${MULTIINSTALLRC}

fi

TGTHOSTNAME=$(echo "${TGTHOST}" | cut -d '.' -f 1)

# Lock to ensure exclusice execution for each host

source "$MYHOME/bin/lock.inc.sh"

lock "${TGTHOSTNAME}" 4

# if no override name was provided or the provided override name ist not part of the override list,

# print usage information together with a list of usable overrides

if [ -z "$OVERRIDE" ] || ( ! echo -e "$OVERRIDELIST\n$ACCOUNTLIST\n$LOCALLIST\n$SPECIALLIST" | sed 's#/accounts##g; s#/local##g; s#/special##g' | grep -q "^$OVERRIDE$" )

then

if [ ${SHORTERROR} -eq 1 ]

then

[ -z "$OVERRIDE" ] && echo "No override name provided" && exit 2

[ -n "${OVERRIDESTATUSFILE}" ] && printf "%s; %s\n" "${OVERRIDE}" "not found" >> ${OVERRIDESTATUSFILE}

STAMP=$(date +"%Y%m%d %H:%M:%S")

printf "%.80s\n" "### ${STAMP} $OVERRIDE ${DUMMYSEPARATOR}"

echo

echo "Override ${OVERRIDE} not found"

echo

STAMP=$(date +"%Y%m%d %H:%M:%S")

printf "%.80s\n\n" "### NOTFOUND - ${STAMP} - End of $OVERRIDE ${DUMMYSEPARATOR}"

exit 2

else

echo

echo "Usage: $0 <HOST>[,<HOST2>[,<HOST3>...]] <OVERRIDE> [<PARAMETERS> ...]"

echo " or $0 -f <FILE WITH HOSTNAMES> <OVERRIDE> [<PARAMETERS> ...]"

echo

echo " Available overrides:"

for override in $OVERRIDELIST

do

echo " $override"

done

if [ -n "$LOCALLIST" ]

then

echo " Local overrides:"

for override in $LOCALLIST

do

echo " $override"

done

fi

if [ -n "$ACCOUNTLIST" ]

then

echo " Accounts:"

for override in $ACCOUNTLIST

do

echo " $override"

done

fi

echo

[ -n "${OVERRIDESTATUSFILE}" ] && printf "%s; %s\n" "${OVERRIDE}" "not found" >> ${OVERRIDESTATUSFILE}

exit 2

fi

fi

# If an override is found in the subdirectories accounts or local, these version have got precedence over the standard tree

REALOVERRIDEDIR=$OVERRIDEDIR

if [ -d "$REALOVERRIDEDIR/special/${OVERRIDE}/" ] && [ -n "${SPECIAL}" ]

then

STANDARDCHANGELOG=$(ls ${REALOVERRIDEDIR}/${OVERRIDE}/CHANGELOG.\* 2>/dev/null)

[ -n "$STANDARDCHANGELOG" ] && STANDARDOVVERSION=$(basename $STANDARDCHANGELOG | cut -d '.' -f 2-)

REALOVERRIDEDIR=$REALOVERRIDEDIR/special

SPECIALCHANGELOG=$(ls ${REALOVERRIDEDIR}/${OVERRIDE}/CHANGELOG.\* 2>/dev/null)

[ -n "$SPECIALCHANGELOG" ] && SPECIALOVVERSION=$(basename $SPECIALCHANGELOG | cut -d '.' -f 2-)

if [ -z "${STANDARDOVVERSION}" ]

then

STANDARDOVVERSION="(new)"

else

STANDARDOVVERSION="instead of ${STANDARDOVVERSION}"

fi

echo "Request to install override ${OVERRIDE} in revision ${SPECIALOVVERSION} ${STANDARDOVVERSION}"

if (( YES == 0 )); then

read -p "Are you sure [y|N]? " ANSWER

if [ -z "${ANSWER}" ] || [ "${ANSWER}" != "y" -a "${ANSWER}" != "Y" ]

then

echo "=> User aborted (4)"

exit 4

fi

fi

else

[ -d "$REALOVERRIDEDIR/local/${OVERRIDE}/" ] && REALOVERRIDEDIR=$REALOVERRIDEDIR/local

[ -d "$REALOVERRIDEDIR/accounts/${OVERRIDE}/" ] && REALOVERRIDEDIR=$REALOVERRIDEDIR/accounts

fi

if [ ! -d "${REALOVERRIDEDIR}/${OVERRIDE}" ]

then

echo "No appropriate override found"

[ -n "${OVERRIDESTATUSFILE}" ] && printf "%s; %s\n" "${OVERRIDE}" "not found" >> ${OVERRIDESTATUSFILE}

exit 2

fi

if [[ -f "${REALOVERRIDEDIR}/${OVERRIDE}/.decom" ]]; then

echo "Override is decomissioned and not available for intallation!"

[ -n "${OVERRIDESTATUSFILE}" ] && printf "%s; %s\n" "${OVERRIDE}" "decomissioned" >> ${OVERRIDESTATUSFILE}

exit 2

fi

if [ "${ACTION}" = "info" ]

then

VARIABLES=$(grep "^# X-BMW" ${REALOVERRIDEDIR}/${OVERRIDE}/var/tmp/overrides/${OVERRIDE}-script | grep -v "X-BMW-Begin" | grep -v "X-BMW-End")

SUPPORTEDOS=$(sed -n '/case .\*PRODUCT.\*in/,/esac/p' ${REALOVERRIDEDIR}/${OVERRIDE}/var/tmp/overrides/${OVERRIDE}-script)

echo "Info on ${REALOVERRIDEDIR}/${OVERRIDE}:"

echo "Meta variables:"

echo "${VARIABLES}" | sed 's/^/ /g'

echo "OS support:"

echo "${SUPPORTEDOS}" | sed 's/^/ /g'

exit 0

fi

STAMP=$(date +"%Y%m%d %H:%M:%S")

printf "%.80s\n" "### ${STAMP} - $OVERRIDE ${DUMMYSEPARATOR}"

SUBSYSTEM=$(basename ${REALOVERRIDEDIR})

if echo "${SUBSYSTEM}" | grep -q "overrides"

then

SUBSYSTEM=" "

else

SUBSYSTEM="(${SUBSYSTEM})"

fi

OVSIGNED="N"

# Check integrity, if MD5SUM file exists

if [ -f "${REALOVERRIDEDIR}/${OVERRIDE}/${MD5SUMFILE}" ]

then

MD5RESULT="$( $SUDO\_CMD /global/instserv/bin/verify\_override.sh "${REALOVERRIDEDIR}/${OVERRIDE}" 2>&1 )"

if [ $? -eq 0 ]

then

echo "Integrity of override confirmed"

OVSIGNED="Y"

else

echo "Integrity mismatch in override:"

echo "${MD5RESULT}" | grep "FAILED"

[ -n "${OVERRIDESTATUSFILE}" ] && printf "%s; %s\n" "${OVERRIDE} ${SUBSYSTEM}" "integrity mismatch" >> ${OVERRIDESTATUSFILE}

STAMP=$(date +"%Y%m%d %H:%M:%S")

printf "%.80s\n\n" "### MD5ERROR - ${STAMP} - End of $OVERRIDE (100) ${DUMMYSEPARATOR}"

exit 100

fi

fi

#

# check if ssh login with key authentication works if not exit

#

if ! m\_start to=${TGTHOST} retvar=TGTHOST verbose=1 waittimeout=120 clib\_tunnel; then

[ -n "${OVERRIDESTATUSFILE}" ] && printf "%s; %s\n" "${OVERRIDE}" "no connection" >> ${OVERRIDESTATUSFILE}

STAMP=$(date +"%Y%m%d %H:%M:%S")

printf "%.80s\n\n" "### NOCONN - ${STAMP} - End of $OVERRIDE (101) ${DUMMYSEPARATOR}"

exit 101

fi

# We temporarily check for SLES12 SP2 (kernel 4.x) and exit on those

KERNEL\_VERSION=$( m\_ssh ${TGTHOST} 'uname -r' | strings | head -n1);

if grep -q -e '^[45][.]' <<<"${KERNEL\_VERSION}"; then

echo "Hoopsie. You have reached Server with a SLES12 SP2 or later! Installing overrides is not supported for those!"

exit 66

fi

# If a prepare-script exists and we install the override, execute the prepare script first

if [[ "$ACTION" = "configure" ]] && [[ ${SKIP\_PREPARE} -eq 0 ]]; then

PREPARE=$( $SUDO\_CMD /global/instserv/bin/run\_override\_script.sh $REALOVERRIDEDIR/$OVERRIDE prepare $TGTHOSTNAME 2>&1 )

RESULT=$?

(( RESULT > 0 )) && printf "=> Prepare Script exited with code: $RESULT\n"

if echo "${PREPARE}"| grep -q "^ABORT INSTALLATION"; then

echo "${PREPARE}" | sed 's/^ABORT INSTALLATION/ /g' ;

STAMP=$(date +"%Y%m%d %H:%M:%S")

printf "%.80s\n\n" "### ABORT - ${STAMP} - End of $OVERRIDE (103) ${DUMMYSEPARATOR}"

exit 103

fi

fi

# If a params-script exists and we install the override, execute the param script first

# The result (parameters for the override script) are stored in PARAMS

if [[ "$ACTION" = "configure" ]]; then

PARAMS=$($SUDO\_CMD /global/instserv/bin/run\_override\_script.sh $REALOVERRIDEDIR/$OVERRIDE params $TGTHOSTNAME )

RESULT=$?

(( RESULT > 0 )) && printf "=> Params Script exited with code: $RESULT\n"

if echo "${PARAMS}" | grep -q "ABORT INSTALLATION"

then

echo "${PARAMS}" | sed 's/ABORT INSTALLATION[[:space:]]\*//g'

STAMP=$(date +"%Y%m%d %H:%M:%S")

printf "%.80s\n\n" "### ABORT - ${STAMP} - End of $OVERRIDE (104) ${DUMMYSEPARATOR}"

exit 104

fi

else

PARAMS=""

fi

RSYNC\_RC=0

# If we want to install an override, rsync the content first

if [ "$ACTION" = "configure" ]

then

$SUDO\_CMD /global/instserv/bin/sync\_override.sh $TGTHOST ${REALOVERRIDEDIR}/${OVERRIDE}

RSYNC\_RC+=$?

# Syncing common framework

[[ -s "$COMMON\_FRAMEWORK" ]] && m\_rsync -a \

"$COMMON\_FRAMEWORK" "$TGTHOST:/var/tmp/overrides/.common.lib.sh"

RSYNC\_RC+=$?

fi

if [ ${RSYNC\_RC} -eq 0 ]

then

# If there was no parameter explicitely given, use the parameters from the -params execution

[ -z "$PARAMETER" ] && PARAMETER=$PARAMS

# Now execute the override script on the remote server (with the appropriate action and parameters

echo "Executing ${OVERRIDE}-script $ACTION $PARAMETER (on $TGTHOST) ..."

[ ${DEBUG} -ne 0 ] && DEBUGOPTS="-x" || DEBUGOPTS=""

MERGER=''

if (( MERGE\_STDERR == 1 )); then

MERGER='2>&1'

fi

m\_ssh "$TGTHOST" /bin/bash <<-%REMOTE

logger -t '=instserv=' -- "Running '${OVERRIDE}-script', mode '$ACTION', parameter '$PARAMETER' from $(uname -n)";

test -f "/var/tmp/overrides/.common.lib.sh" && source "/var/tmp/overrides/.common.lib.sh"

FILE="/var/tmp/overrides/${OVERRIDE}-script"

if file -b "\$FILE" | grep -q 'Bourne-Again shell script text'; then

LANG=C /bin/bash ${DEBUGOPTS} \$FILE $ACTION $PARAMETER ${MERGER}

else

LANG=C \$FILE $ACTION $PARAMETER ${MERGER}

fi

%REMOTE

RC=$?

if [[ $RC -ne 0 ]]; then

printf "=> Main Script exited with code: $RC\n"

RESULT=106

else

RESULT=0

fi

else

RESULT=105

fi

if [ "${ACTION}" == "configure" ]; then

if [[ $RESULT -eq 0 ]]; then

RPMNAME="$( ${SUDO\_CMD} /global/instserv/bin/run\_override\_checkdummyrpm.sh $REALOVERRIDEDIR/$OVERRIDE )"

if [[ -n $RPMNAME ]]; then

m\_ssh "$TGTHOST" rpm -U --ignoresize --force /var/tmp/overrides/tmp/${RPMNAME} 2>&1

RC=$?

if (( RC == 0 )); then

echo "Dummy rpm for override installation successful.."

else

echo "Dummy rpm for override installation failed! rc=$RC"

fi

else

echo "Dummy rpm for override does not exist - skipping"

fi

fi

# Call the postscript script, if it exists

${SUDO\_CMD} /global/instserv/bin/run\_override\_script.sh $REALOVERRIDEDIR/$OVERRIDE postscript install ${TGTHOST} ${RESULT}

POSTRESULT=$?

if [[ $POSTRESULT -ne 0 ]]; then

echo "=> Postscript exited with code: $POSTRESULT"

RESULT=107

fi

fi

if [ "${ACTION}" == "remove" ]

then

m\_ssh "$TGTHOST" rpm -e BMW\_${OVERRIDE}

fi

if [ ${STOREINDB} -eq 1 ] && [ "${ACTION}" == "configure" -o "${ACTION}" == "remove" ]

then

# Store information in database

TIMESTAMP=$(date +"%Y-%m-%d %H:%M:%S")

CHANGELOG=$(ls ${REALOVERRIDEDIR}/${OVERRIDE}/CHANGELOG.\* 2>/dev/null)

[ -n "$CHANGELOG" ] && OVVERSION=$(basename $CHANGELOG | cut -d '.' -f 2-)

echo "Storing information in database ..."

CallDB "REPLACE INTO overrides (hostname,instserv,override,version,signed,parameters,result,installdate) VALUES ('$TGTHOSTNAME','${INSTSERV}','$OVERRIDE','$OVVERSION','$OVSIGNED','$ACTION $PARAMETER','$RESULT','$TIMESTAMP')"

fi

if [ -n "${OVERRIDESTATUSFILE}" ]; then

if [ ${RESULT} -eq 0 ]; then

printf "%s; OK (0)\n" "${OVERRIDE} ${SUBSYSTEM}"

else

printf "%s; NOK (%i) -- %s\n" "${OVERRIDE} ${SUBSYSTEM}" ${RESULT} "${INST\_OV\_ERRORS[$RESULT]:-Unknown}"

fi >> ${OVERRIDESTATUSFILE}

fi

if [ "${RESULT}" = "0" ]

then

LONGRESULT="SUCCESS"

else

LONGRESULT="ERROR"

fi

STAMP=$(date +"%Y%m%d %H:%M:%S")

printf "%.80s\n\n" "### ${LONGRESULT} - ${STAMP} - End of $OVERRIDE (${RESULT}) ${DUMMYSEPARATOR}"

exit ${RESULT}

[INSTSERV][qxz10kv][lpinstiaas02] ~ $

[INSTSERV][qxz10kv][lpinstiaas02] /global/instserv/bin $

[INSTSERV][qxz10kv][lpinstiaas02] /global/instserv/bin $

[INSTSERV][qxz10kv][lpinstiaas02] /global/instserv/bin $ cat check\_dlv\_dgv.sh

#! /bin/bash

# "check\_dlv\_dgv.sh"

# Version 0.12

#

# - Check the Veritas Version Infos and the DLVs of mounted vxfs Filesystems on a server.

# - Check if the minuimum defined DLV is active, and which VxVM Version is necessary for

# the minimum DLV

#

# Modification Log:

# - Feb 18 18:27:21 CET 2021 Version 0.11 GN

# \* Added check for VRTSvcs since there have been issues with only VxVM being upgraded

# to the current version & VRTvcs remaining at the previous release (possible bug).

# \* Replaced the multiple invocations of $(uname -n) with one variable $THIS\_HOST

# \* Replaced redundant echo infos with function info summary

# - Mar 9 11:35:27 CET 2021 Version 0.12 GN

# \* Modified the get\_dlv\_mounted and get\_dgv\_imported functions to not exit so the

# Veritas PLs are still checked even when no DGs/LVs are active on the server

#

# Script Responsibles:

# Until the new Veritas Product owner @ FG-840 has been established the script ownership

# is delegated to the following people:

#

# Suchetan.Raj@partner.bmw.de

# Praveen.DE.Desai@partner.bmw.de

# VijayKumarReddy.Ayyaluri@partner.bmw.de

#

# ###################################################################

# Any new Versions beyond 7.3.4 will need to be added to the case statement.

# i.e. with EVERY NEW VxVM release!

# ###################################################################

#

######################################################################################

# This should work on SLES11-SP1 to SLES12-SP4

# This script was written on private time and is completly unsupported &

# and under the exclusion of all and any liabilities.

# Only use at your own risk.

# OS-Admins are welcome to use and modify this script as they require.

# The script employs read only routines, written to be used with rootscp & rootssh:

# E.g.:

# rootscp check\_dlv.sh <server>:/var/tmp/

# rootssh <server> "/var/tmp/check\_dlv.sh"

######################################################################################

#

# Get SLES Version we a running on:

# Since older versions of SLES11 do not have /etc/os-release and some that do, have different values,

# using the to be obsoleted /etc/SuSE-release unless it does not exist..

# .. in case some rootsh user has mucked about, thy three methods...

#

THIS\_HOST=$(uname -n)

if [ -f /etc/SuSE-release ] ; then

OSRELA=$(awk -F' = ' '$1 == "VERSION" {print $2}' /etc/SuSE-release)

OSRELB=$(awk -F' = ' '$1 == "PATCHLEVEL" {print $2}' /etc/SuSE-release)

OSREL="${OSRELA}${OSRELB}"

OSREL\_NAME="${OSRELA}-SP${OSRELB}"

elif [ -f /etc/os-release ] ; then

OSRELA=$(awk '$1 == "Welcome" {print "SLES"$7}' /etc/issue)

OSRELB=$(awk '$1 == "Welcome" {print $8}' /etc/issue)

OSREL\_NAME=${OSRELA}-${OSRELB}

elif [ -f /etc/issue ] ; then

OSREL\_NAME=$(awk -F'=' '$1 == "VERSION" {print "SLES"$2}' /etc/os-release | tr -d '["]')

else

echo "ERROR determining SLES version... setting to SLES-UNKNOWN"

OSREL\_NAME=SLES-UNKNOWN

fi

# Get the VCS Version data

if ! rpm -q VRTSvcs > /dev/null 2>&1 ; then

VCSCHECK=1

VCSVERS\_FULL="VRTSvcs not installed on $THIS\_HOST"

VCSERROR="INFO: No VxVM/VCS Mismatch $VCSVERS\_FULL"

VCSVERS\_PATCHLEVEL=""

else

# VCS

VCSCHECK=0

VCSVERS\_RMAJOR=$(rpm -q VRTSvcs | awk -F'-' '{print$2}'| awk -F'.' '{print $1}')

VCSVERS\_RMINOR=$(rpm -q VRTSvcs | awk -F'-' '{print$2}'| awk -F'.' '{print $2}')

VCSVERS\_SP=$(rpm -q VRTSvcs | awk -F'-' '{print$2}'| awk -F'.' '{print $3}')

if [ -z "$VxVMVERS\_SP" ] ; then

VxVMVERS\_SP=0

fi

VCSVERS\_PATCHLEVEL=$(rpm -q VRTSvcs | awk -F'-' '{print$2}'| awk -F'.' '{print $4}')

if [ -z "$VCSVERS\_PATCHLEVEL" ] ; then

VCSVERS\_PATCHLEVEL=0

fi

VCSVERS\_MAJOR="${VCSVERS\_RMAJOR}.${VCSVERS\_RMINOR}"

VCSVERS\_MAJMINOR="${VCSVERS\_MAJOR}.${VCSVERS\_SP}"

VCSVERS\_FULL="${VCSVERS\_MAJMINOR}.${VCSVERS\_PATCHLEVEL}"

fi

if ! rpm -q VRTSvxvm > /dev/null 2>&1 ; then

echo "Veritas VRTSvxvm is not installed on this system..."

echo "please check if $THIS\_HOST is correct.."

echo "ERROR ABORT $THIS\_HOST Veritas VRTSvxvm is not installed on this system..."

exit 995

else

# VxVM

VxVMVERS\_RMAJOR=$(rpm -q VRTSvxvm | awk -F'-' '{print$2}'| awk -F'.' '{print $1}')

VxVMVERS\_RMINOR=$(rpm -q VRTSvxvm | awk -F'-' '{print$2}'| awk -F'.' '{print $2}')

VxVMVERS\_SP=$(rpm -q VRTSvxvm | awk -F'-' '{print$2}'| awk -F'.' '{print $3}')

if [ -z "$VxVMVERS\_SP" ] ; then

VxVMVERS\_SP=0

fi

VxVMVERS\_PATCHLEVEL=$(rpm -q VRTSvxvm | awk -F'-' '{print$2}'| awk -F'.' '{print $4}')

if [ -z "$VxVMVERS\_PATCHLEVEL" ] ; then

VxVMVERS\_PATCHLEVEL=0

fi

VxVMVERS\_MAJOR="${VxVMVERS\_RMAJOR}.${VxVMVERS\_RMINOR}"

VxVMVERS\_MAJMINOR="${VxVMVERS\_MAJOR}.${VxVMVERS\_SP}"

VxVMVERS\_FULL="${VxVMVERS\_MAJMINOR}.${VxVMVERS\_PATCHLEVEL}"

fi

if [[ -z ${VxVMVERS\_MAJOR} ]] ; then

echo "ERROR: NO Veritas VRTSvxvm Version found on system $THIS\_HOST"

echo "Check the system manually.... ABORTING.. NO Veritas VRTSvxvm Version found"

exit 99

fi

# Cross Check VRTSvxvm and VRTSvcs Versions

if [[ "$VCSCHECK" -eq 0 ]] ; then

if [[ "$VxVMVERS\_RMAJOR" -ne "$VCSVERS\_RMAJOR" ]] ; then

VCSERROR="ERROR: MISMATCH VCS Major Version $VCSVERS\_RMAJOR does not match VxVM Version $VxVMVERS\_RMAJOR"

elif [[ "$VxVMVERS\_RMINOR" -ne "$VCSVERS\_RMINOR" ]] ; then

VCSERROR="ERROR: MISMATCH VCS Mainor Version $VCSVERS\_MAJOR does not match VxVM Version $VxVMVERS\_MAJOR"

elif [[ "$VxVMVERS\_SP" -ne "$VCSVERS\_SP" ]] ; then

VCSERROR="WARNING: MISMATCH VCS SP Version $VCSVERS\_MAJMINOR does not match VxVM Version ${VxVMVERS\_MAJMINOR}. Check if newer SP release is available"

else

VCSERROR="OK: Veritas VRTSvxvm $VxVMVERS\_MAJMINOR and VRTSvcs $VCSVERS\_MAJMINOR Versions match"

fi

fi

# The BMW standard is $MIN\_DLV

#####MIN\_DLV=15

MIN\_VxVM="${VxVMVERS\_MAJOR}.${VxVMVERS\_SP}"

############################################################################################

# DLV Support Installed VxVM Version: Source https://sort.veritas.com/dgfs\_matrix/fs\_matrix

# 5.1 (lpcaps3/4) supprts up to DLV 7? No longer documented @ Semantec

# 6.0 supports DLV 7-9 & Supports Mount only for vxupgrade minimum DLV 6

# 6.1 supports DLV 7-10 & Supports Mount only for vxupgrade minimum DLV 6

# 6.2 supports DLV 7-10 & Supports Mount only for vxupgrade minimum DLV 6

# 7.0 supports DLV 7-10 & Supports Mount only for vxupgrade minimum DLV 6

# 7.1 supports DLV 7-11 & Supports Mount only for vxupgrade minimum DLV 6

# 7.2 supports DLV 9-12 & Supports Mount only for vxupgrade DLVs 6-8

# 7.3 supports DLV 9-12 & Supports Mount only for vxupgrade DLVs 6-8

# 7.3.1+ apparently supports DLV 9-13 & Supports Mount only for for vxupgrade DLVs 6-8

# 7.4.0 supports DLV 10-14 (not used @ BMW) & Supports Mount only for vxupgrade DLVs 6-9

# 7.4.1 supports DLV 11-15 & Supports Mount only for vxupgrade DLVs 6-10

# 7.4.2 supports DLV 12-16 & Supports Mount only for vxupgrade DLVs 6-11

# 7.4.3 supports DLV 12-16 & Supports Mount only for vxupgrade DLVs 6-11

#

############################################################################################

# DGV Support Installed VxVM Version: Source https://sort.veritas.com/dgfs\_matrix/dg\_matrix

# NOTE: VxVM 7.4.1 (and possibly lower) cannot modify/extend DGVs less than DG Version 160!

# All VxVM Versions fully support the use of (but not modification of) all previous

# DGV versions.

# NOTE: Some VxVM Version infos are missing in the Veritas link, known deviations have been

# added below.

#

# 4.0 Supports DGVs up to max 110

# 4.1 Supports DGVs up to max 120

# 5.0 Supports DGVs up to max 130

# 5.0 Supports DGVs up to max 140

# 5.1 Supports DGVs up to max 150

# 5.1SP1 Supports DGVs up to max 160

# 6.0 Supports DGVs up to max 170 [DCO version 30 CAUTION larger DCO LOG SPACE REQUIRED]

# 6.0.1 Supports DGVs up to max 180

# 6.1 Supports DGVs up to max 190

# 6.2 Supports DGVs up to max 200

# 7.0 Supports DGVs up to max 200

# 7.1 Supports DGVs up to max 220

# 7.2 Supports DGVs up to max 230

# 7.3 Supports DGVs up to max 230

# 7.3.1 Supports DGVs up to max 240

# 7.4 Supports DGVs up to max 260

# 7.4.1 Supports DGVs up to max 280

# 7.4.2 Supports DGVs up to max 290

# 7.4.3 Supports DGVs up to max 300

#

############################################################################################

echo

echo "Checking host $THIS\_HOST SLES Release ${OSREL\_NAME} VxVM Version ${VxVMVERS\_MAJOR}.${VxVMVERS\_SP} PL: ${VxVMVERS\_PATCHLEVEL}"

if [[ ${OSREL} -lt 114 ]] ; then

echo "WARNING SLES ${OSREL\_NAME} is not a Supported Version for migrations to SLES12"

fi

function get\_dlv\_mounted {

# Function to get the DLV of all mounted vxfs filesystems on this server

echo

echo "Checking DiskLayout DLVs .. ${VxVMVERS\_MAJOR}.${VxVMVERS\_SP} Max supported DLV is $MAX\_SUP\_DLV"

echo

if ! df -t vxfs > /dev/null 2>&1 ; then

echo "No xvfs Filesystems mounted on $THIS\_HOST, no DLVs to check/update on this system"

else

for LVOL in $(df -hP -t vxfs | egrep -v ^Filesystem | awk '{print $1}'| sort -n) ; do

for CDLV in $(/opt/VRTS/bin/fstyp -v ${LVOL} | egrep -i "version"| awk '{print $4}') ; do

if [[ "$CDLV" -ge "$MAX\_SUP\_DLV" ]] ; then

echo "LV $LVOL has max DLV $CDLV OK"

else

echo "LV $LVOL has DLV $CDLV ERROR NOK: UPDATE TO DLV $MAX\_SUP\_DLV"

fi

done

done

fi

}

function get\_dgv\_imported {

# Get the DGs and the DGV for each OK if >= MAX\_SUP\_DLV and UPDATE if < MAX\_SUP\_DLV

# The ">" should never happen, but the Veritas documentation is as water tight as swiss cheese.

echo

echo "Checking DiskGroup DGVs .. ${VxVMVERS\_MAJOR}.${VxVMVERS\_SP} Max supported DGV is $MAX\_SUP\_DGV"

echo

VXDGNR="$(vxdg -q list | wc -l)"

if [ "$VXDGNR" -ne 0 ] ; then

for DG in $(vxdg -q list | awk '{print $1}' | sort -n); do

CDGV=$(vxdg -q list $DG | egrep -i ^version:| awk '{print $NF}')

if [ "$CDGV" -ge "$MAX\_SUP\_DGV" ] ; then

echo "DG $DG has max DGV $CDGV OK"

else

echo "DG $DG has DGV $CDGV ERROR NOK: UPDATE TO DGV $MAX\_SUP\_DGV"

fi

done

else

echo "No imported DGs found on $THIS\_HOST, no DGs to check/update on this system"

fi

}

function info\_summary {

echo "Installed VxVM Version: ${VxVMVERS\_MAJOR}.${VxVMVERS\_SP} PL ${VxVMVERS\_PATCHLEVEL} on $THIS\_HOST"

echo "VxVM ${VxVMVERS\_MAJOR}.${VxVMVERS\_SP} supports: DLV ${MIN\_SUP\_DLV}-${MAX\_SUP\_DLV}"

echo "VxVM ${VxVMVERS\_MAJOR}.${VxVMVERS\_SP} supports: DGV MAX ${MAX\_SUP\_DGV}"

if [[ "$VCSCHECK" -eq 0 ]] ; then

echo "VRTvcs Installed Version: $VCSVERS\_MAJMINOR PL $VCSVERS\_PATCHLEVEL on $THIS\_HOST"

else

echo "VRTvcs Installed Version: $VCSVERS\_FULL"

fi

echo

}

case "${VxVMVERS\_MAJOR}" in

5.\*|4.\*)

echo

echo "ERROR: UNSUPPORTED VERITAS VxVM Version ${VxVMVERS\_MAJOR}.${VxVMVERS\_SP} PL: ${VxVMVERS\_PATCHLEVEL} on $THIS\_HOST"

echo "ERROR: THIS SYSTEM CAN ONLY BE APPLICATION (NOT SAN) MIGRATED TO A NEW SERVER ONLY"

echo "INFO: INSTALLED VRTSvcs Version is: $VCSVERS\_FULL"

echo

echo "ERROR: for Info ONLY, this is a completely unsupported VxVM version"

echo "ERROR: Printing a DLV and DGV List ONLY...."

echo

echo "Checking the DLV of mounted vxfs LVOLs:"

for LVOL in $(df -hP -t vxfs | egrep -v ^Filesystem | awk '{print $1}' | sort -n) ; do /opt/VRTS/bin/fstyp -v ${LVOL} | egrep -i "version" | awk -v y=$LVOL '{print y" \t version: "$4}'; done

echo

echo "Checking the DGV of active Disk Groups:"

for i in `vxdg -q list | awk '{print $1}'`; do echo -n $i" ";vxdg -q list $i | grep version;done

echo

echo "ERROR: This info is ONLY JFYI, the system $THIS\_HOST must be APPLICATION (NOT SAN) MIGRATED TO A NEW SERVER"

echo

;;

6.0)

MIN\_SUP\_DLV=7

MAX\_SUP\_DLV=9

if [ "$VxVMVERS\_SP" -ge 1 ] ; then

MAX\_SUP\_DGV=180

else

MAX\_SUP\_DGV=170

fi

info\_summary

get\_dlv\_mounted

get\_dgv\_imported

;;

6.1|6.2)

MIN\_SUP\_DLV=7

MAX\_SUP\_DLV=10

if [ "$VxVMVERS\_MAJOR" == "6.1" ] ; then

MAX\_SUP\_DGV=190

else

MAX\_SUP\_DGV=200

fi

info\_summary

get\_dlv\_mounted

get\_dgv\_imported

;;

7.0)

MIN\_SUP\_DLV=7

MAX\_SUP\_DLV=10

MAX\_SUP\_DGV=200

info\_summary

get\_dlv\_mounted

get\_dgv\_imported

;;

7.1)

MIN\_SUP\_DLV=7

MAX\_SUP\_DLV=11

MAX\_SUP\_DGV=220

info\_summary

get\_dlv\_mounted

get\_dgv\_imported

;;

7.2)

MIN\_SUP\_DLV=9

MAX\_SUP\_DLV=12

MAX\_SUP\_DGV=230

info\_summary

get\_dlv\_mounted

get\_dgv\_imported

;;

7.3)

case "$VxVMVERS\_SP" in

0)

MIN\_SUP\_DLV=9

MAX\_SUP\_DLV=12

MAX\_SUP\_DGV=230

info\_summary

get\_dlv\_mounted

get\_dgv\_imported

;;

1|2|3|4|5|6|7|8|9)

MIN\_SUP\_DLV=10

MAX\_SUP\_DLV=13

MAX\_SUP\_DGV=240

# for 7.3.1 and later support was added for DLV 13, not listed @ https://sort.veritas.com/dgfs\_matrix/fs\_matrix

info\_summary

get\_dlv\_mounted

get\_dgv\_imported

# TEMPORARY EXTRA 7.3.1 PATCH LEVEL CHECK due to current PLEX and ODM issues

if [ ${VxVMVERS\_SP} -eq 1 ] ; then

echo

echo "Extra Patch Level Check for ${VxVMVERS\_MAJOR}.${VxVMVERS\_SP}"

echo

MIN\_VRTSvxvm=3107

IS\_VRTSvxvm=${VxVMVERS\_PATCHLEVEL}

if [ "$IS\_VRTSvxvm" -lt "$MIN\_VRTSvxvm" ] ; then

echo "ERROR: VRTSvxvm Patch Level is $IS\_VRTSvxvm this must be updated to at least $MIN\_VRTSvxvm"

else

echo "OK: VRTSvxvm Patch Level is $IS\_VRTSvxvm"

fi

fi

;;

esac

;;

7.4)

case "$VxVMVERS\_SP" in

0)

MIN\_SUP\_DLV=10

MAX\_SUP\_DLV=14

MAX\_SUP\_DGV=260

info\_summary

get\_dlv\_mounted

get\_dgv\_imported

;;

1)

MIN\_SUP\_DLV=11

MAX\_SUP\_DLV=15

MAX\_SUP\_DGV=280

info\_summary

get\_dlv\_mounted

get\_dgv\_imported

# TEMPORARY EXTRA 7.4.1 PATCH LEVEL CHECK due to current PLEX and ODM issues

echo

echo "Extra Patch Level Check for ${VxVMVERS\_MAJOR}.${VxVMVERS\_SP}"

echo

MIN\_VRTSvxvm=2705

IS\_VRTSvxvm=${VxVMVERS\_PATCHLEVEL}

MIN\_VRTSodm=1700

IS\_VRTSodm=$(rpm -q VRTSodm | awk -F'-' '{print$2}'| awk -F'.' '{print $4}')

MIN\_VRTSvxfs=1700

IS\_VRTSvxfs=$(rpm -q VRTSvxfs | awk -F'-' '{print$2}'| awk -F'.' '{print $4}')

if [ "$IS\_VRTSvxvm" -lt "$MIN\_VRTSvxvm" ] ; then

echo "ERROR: VRTSvxvm Patch Level is $IS\_VRTSvxvm this must be updated to at least $MIN\_VRTSvxvm"

else

echo "OK: VRTSvxvm Patch Level is $IS\_VRTSvxvm"

fi

if [ "$IS\_VRTSodm" -lt "$MIN\_VRTSodm" ] ; then

echo "ERROR: VRTSodm Patch Level is $IS\_VRTSodm this must be updated to at least $MIN\_VRTSodm"

else

echo "OK: VRTSodm Patch Level is $IS\_VRTSodm"

fi

if [ "$IS\_VRTSvxfs" -lt "$MIN\_VRTSvxfs" ] ; then

echo "ERROR: VRTSvxfs Patch Level is $IS\_VRTSvxfs this must be updated to at least $MIN\_VRTSvxfs"

else

echo "OK: VRTSvxfs Patch Level is $IS\_VRTSvxfs"

fi

;;

2)

MIN\_SUP\_DLV=12

MAX\_SUP\_DLV=16

MAX\_SUP\_DGV=290

info\_summary

get\_dlv\_mounted

get\_dgv\_imported

;;

3)

MIN\_SUP\_DLV=12

MAX\_SUP\_DLV=16

MAX\_SUP\_DGV=300

info\_summary

get\_dlv\_mounted

get\_dgv\_imported

;;

\*)

echo "ERROR: An unknown VxVM 7.4.[n] MINOR Release found on $THIS\_HOST ${VxVMVERS\_MAJOR}.${VxVMVERS\_SP}"

echo "This script will need to be updated accordingly to deal with this NEW VxVM 7.4 MINOR release"

echo "ERROR: ABORT $THIS\_HOST unknown VxVM 7.4.[n] MINOR Release"

exit 991

;;

esac

;;

\*)

echo "ERROR: An unknown Veritas VxVM MAJOR Version encountered ${VxVMVERS\_MAJOR}.${VxVMVERS\_SP}"

echo "This script will need to be updated accrdingly to deal with this VxVM release"

echo "ERROR: ABORT $THIS\_HOST unknown Veritas VxVM MAJOR Version"

exit 992

;;

esac

echo

echo "$VCSERROR"

echo

# eof

[INSTSERV][qxz10kv][lpinstiaas02] /global/instserv/bin $

[INSTSERV][qxz10kv][lpinstiaas02] /global/instserv/bin $ cat create\_iso.sh

#!/bin/bash

# Create a boot disk to install a new server

if [[ $EUID != 0 ]]; then

echo "This program required root privileges, please invoke it with sudo!"

exit 1

fi

if [[ -z $SUDO\_USER ]]; then

TARGET\_USER="root"

else

TARGET\_USER="$SUDO\_USER"

fi

# My own PID (for unique temporary files)

PROC=$$

# Host to install

HOST=$1

# Which boot disk template to use (default: default (which is a symlink to a specific boot disk template)

BOOTDISK=$2

[ -z "${BOOTDISK}" ] && BOOTDISK=default

# If variable INSTSERV is not set, use the current hostname

[ -z "${INSTSERV}" ] && INSTSERV=$(uname -n)

# Get my own directory (for pseudo relative path walking)

MYHOME=$(dirname $(readlink -f $0) | sed 's#/bin$##')

# The DB name (for the host keys) is derived from the name of the install server

DB=${MYHOME}/database/${INSTSERV}.db

function CallDB {

local QUERY="$\*"

echo -e ".timeout 5000\n${QUERY};" | sqlite3 ${DB}

}

# Add /sbin to the current PATH (I forgot why ;-))

PATH=$PATH:/sbin

# Where to store the config file

LOCAL\_CFG=${MYHOME}/bootdisks/config/${HOST}.cfg

# Some temporary directories and files

TMP\_DIR=${MYHOME}/tmp/cdrom

TMP\_ISO\_ROOT=${TMP\_DIR}/iso.d.$PROC

TMP\_MNT=${TMP\_DIR}/mnt.$PROC

SOURCE\_FLAVOR=${MYHOME}/bootdisks/templates/${BOOTDISK}

SOURCE\_BOOT\_CD=${SOURCE\_FLAVOR}/cd

SOURCE\_INITRD\_DIR=${SOURCE\_FLAVOR}/initrd

# Where put store the final iso image

IMAGE=${MYHOME}/bootdisks/built/${HOST}.iso

umask 0333

# Prepare the temporary directory structure, copy the config file, the ssh key and the boot disk to the new dir

mkdir -p ${TMP\_DIR} ${TMP\_MNT}/tmp && \

rsync -a ${SOURCE\_INITRD\_DIR}/ ${TMP\_MNT}/ && \

cp -p ${LOCAL\_CFG} ${TMP\_MNT}/tmp/local.cfg && \

cp -a ${SOURCE\_BOOT\_CD} ${TMP\_ISO\_ROOT} || exit 1

grep -q -e ^V6\_IPADDR $LOCAL\_CFG && rm ${TMP\_MNT}/etc/modprobe.d/50-ipv6.conf

# Also copy the roothash from the database to /tmp/roothash

ROOTHASH=$(CallDB "SELECT value FROM config WHERE key='ROOTHASH'")

[ -n "${ROOTHASH}" ] && echo "${ROOTHASH}" > ${TMP\_MNT}/tmp/roothash

# Dive into ${TMP\_MNT} and create an initial ram disk out of it

cd ${TMP\_MNT} && \

find . -mindepth 1 -print0 2>/dev/null| cpio -o0H newc 2>/dev/null| gzip -6nv > ${TMP\_ISO\_ROOT}/initrd 2>/dev/null || exit 1

cd - >/dev/null

# Now we can drop the temporary directory

rm -r ${TMP\_MNT} || exit 1

# And clean up output image, because mkisofs fails if the output is a broken symlink to an already removed iso, mounted by the external option

rm ${IMAGE} 2> /dev/null

# Finally, create a boot iso

if [[ -x /usr/bin/genisoimage ]]; then

MKISO=/usr/bin/genisoimage

elif MKISO=$( which mkisofs 2> /dev/null ); then

: We have found the legacy mkisofs...

else

echo "No genisoimage or mkisofs found!"

exit 1

fi

"${MKISO}" \

-b isolinux.bin \

-c boot.cat \

-no-emul-boot -boot-load-size 4 \

-boot-info-table -o ${IMAGE} \

${TMP\_ISO\_ROOT} >/dev/null 2>&1|| exit 1

# And clean up

rm -r ${TMP\_ISO\_ROOT} || exit 1

chown ${TARGET\_USER}: ${IMAGE}

# Give back the (full) name of the iso image (for reuse in the calling script)

echo "${IMAGE}"

exit 0

[INSTSERV][qxz10kv][lpinstiaas02] /global/instserv/bin $

[INSTSERV][qxz10kv][lpinstiaas02] /global/instserv/bin $ cat secret\_share.sh

#!/bin/bash

#

# -------------------------------------------------------------------------------------------------------

#

# secure\_share.sh - Share new pasword of reset account on a server

#

# Usage: See USAGE()

#

# Revision history:

# 2021-09-01 - Ver 1.0 First recorded version - See VERSION

# 2021-09-06 - Ver 1.1 Password to be safed to a file

# 2021-09-20 - Ver 1.2 Update the script flow and function

# 2021-10-01 - Ver 1.3 Set remote file permissions

# 2021-10-14 - Ver 2.0 Update the -e flag to accept e-mail addresses with "-" & Add password generator

# 2021-10-22 - Ver 3.0 Change destination server for second part of secret

# 2021-10-28 - Ver 3.1 Add missing sudo to command mkdir

# 2021-11-26 - Ver 3.2 E-mail lookup added, multi servers with "-s"

# 2021-12-03 - Ver 3.3 List servers in body and set reply e-mail address

# 2022-01-13 - Ver 3.4 User not in PBIS - adjust email body

# Error if not correct instserv

# Add logging

# 2022-01-14 - Ver 3.5 Change email subject to reflect user that password is changed

# 2022-03-18 Ver 3.6 Fixed path for centrally managed SSH privat key

#

# -------------------------------------------------------------------------------------------------------

PROGNAME=${0##\*/}

VERSION="3.6"

SSH="/usr/bin/ssh"

SCP="/usr/bin/scp"

MAILX="/usr/bin/mailx"

PBIS="/usr/bin/pbis"

LOGGER\_BIN="/usr/bin/logger"

SSH\_USER="qqsecpw"

SSH\_OPTIONS="-i /global/instserv/share/keys/id\_qqsecpw"

TARGET\_SERVER="lpsecpw02.bmwgroup.net"

FILE=".shared\_2\_${RANDOM}\_$(date +'%Y%m%d%H%M')"

LOCAL\_FOLDER="/tmp/sharedpasswords"

BODY="/tmp/email\_body"

REPLY\_MAIL="LinuxOperationsSecretSharingTool@noreply"

# Check for working folder

if ! [[ -d ${LOCAL\_FOLDER} ]]; then

mkdir -p ${LOCAL\_FOLDER}

fi

msg\_logger() { #msg\_logger Step "Message"

${LOGGER\_BIN} -p local3.info --id ${1} -t "=secret\_share=" "${SUDO\_USER}:${2}"

}

# Trap signals

trap "signal\_exit TERM" TERM HUP

trap "signal\_exit INT" INT

USAGE() {

echo -e "${PROGNAME} - ${VERSION}\n\nUsage: $PROGNAME [-h|--help] [-g] -u <username> -t <system\_account> -s [<server>|<server>,<server>,...] -p <password>\n"

}

help\_message() {

USAGE

cat <<- \_EOF\_

Script is used to email the password of a system account (-t) on a server (-s) to the requester (-u).

$PROGNAME

-h | --help : This message

-g : Generate password only

-u <username> : User that requested the reset of technical user

-t <system\_account> : System Account's password that was resetted on remote server

-s <server> : Server where the password was reset, multiple servers can be specified, seprate by comma

-p <password> : Password that is used

\_EOF\_

}

if (($# == 0)); then

USAGE

fi

house\_keeping() {

rm -rf /tmp/sharedpasswords/\*.pwd

rm -rf /tmp/email\_body

}

output() {

# Possible $2 inputs - OUTPUT\_NONE=0

# - OUTPUT\_CONSOLE=1

# - OUTPUT\_LOG=2

# - OUTPUT\_BOTH=3

M\_OUTPUT=$OUTPUT\_BOTH

if [[ ! -z "$2" ]]; then

M\_OUTPUT=$2

fi

#Console output

if [[ $M\_OUTPUT -eq 1 || $M\_OUTPUT -eq 3 ]]; then

printf "$1\n"

fi

#Log output

if [[ $M\_OUTPUT -eq 2 || $M\_OUTPUT -eq 3 ]]; then

if [[ -d /global/instserv/logs ]]; then

printf "`date +"%d-%m-%y %H:%M:%S"` [${server}/$SUDO\_USER/$$] $1\n" | sed -r "s/\x1B\[([0-9]{1,3}(;[0-9]{1,2})?)?[mGK]//g" >> /global/instserv/logs/secret\_share.log

fi

fi

}

error\_exit() {

output "\033[31mError - $1\033[0m" 3

house\_keeping

exit 1

}

signal\_exit() { # Handle trapped signals

case $1 in

INT)

error\_exit "User interrupted program"

;;

TERM)

error\_exit "Program terminated" >&2

;;

\*)

error\_exit "Program terminated on unknown signal"

;;

esac

shift

}

gen\_password() {

# Can only contain @#$%^\*?+/

pass=$(openssl rand -base64 15)

echo -e "${pass}"

}

check\_instserv() {

source /global/instserv/bin/ssh.lib.sh || { error\_exit "Could not source ssh.lib.sh"; }

source /global/instserv/bin/dcodb.lib.sh || { error\_exit "Could not source dcodb.lib.sh"; }

IFS=',' read -r -a server\_list <<< "${server}"

new\_server=""

len=${#server\_list[@]}

for (( i=0; i<${len}; i++ )); do

INSTALLSRV=$( dcodb.host2instserv "${server\_list[${i}]}" ) || { error\_exit "No valid Install server for ${server\_list[${i}]}"; }

if [[ "${INSTALLSRV}" != "$(hostname)" ]]; then

echo -e "\n"

output "\033[31mError - Password cannot be reset for ${server\_list[${i}]}, perform this on ${INSTALLSRV}...\033[0m" 3

else

new\_server+="${server\_list[${i}]},"

fi

done

server=`echo ${new\_server} | sed 's/,$//'`

}

pbis\_usr\_check() {

RES=`$PBIS find-user-by-name ${username} 1>/dev/null 2>&1; echo $?`

if [[ $RES -ne 0 ]]; then

PBIS\_MSG="${username}, don't have access to ${TARGET\_SERVER} to retrieve the second part of the secret.\n\n"

PBIS\_MSG+="Please log an ITSM Incident to the same service that originated the secret generation/reset.\n\n"

PBIS\_MSG+="Please state that they need to provide the second part of the secret in /home/${username}/${FILE} on ${TARGET\_SERVER}\n\n"

else

PBIS\_MSG="The second part is in a text file that has been copied to server, ${TARGET\_SERVER}, under: /home/${username}/${FILE}\n"

PBIS\_MSG+="Please remove /home/${username}/${FILE} from ${TARGET\_SERVER}, after secret was retrieved.\n\n"

PBIS\_MSG+="Please note that this is an automatically generated e-mail. Do not answer to it!\n\n"

PBIS\_MSG+="If you have any problem with the retrieval of the password please open an ITSM Incident to the same service that originated the secret generation/reset\n"

fi

}

email\_passwd() {

pbis\_usr\_check

echo -e "\nE-mail secret to ${email}\n"

SUBJECT="Share secret from Linux Operations for ${tech\_user}"

echo -e "Dear user,\n\nA new secret has been generated for the user ${tech\_user} on\n" > ${BODY}

IFS=',' read -r -a server\_list <<< "${server}"

len=${#server\_list[@]}

for (( i=0; i<${len}; i++ )); do echo -e " ${server\_list[${i}]}" >> ${BODY}; done

echo -e "\nFor a secure transmission, the secret has been splitted in 2 parts that need to be concatenated\n" >> ${BODY}

echo -e "The first part of the secret is: ${email\_part}\n" >> ${BODY}

echo -e $PBIS\_MSG >> ${BODY}

echo -e "Best Regards\nYour Linux Operations Team" >> ${BODY}

cat ${BODY} | ${MAILX} -r ${REPLY\_MAIL} -R ${REPLY\_MAIL} -s "${SUBJECT}" ${email};

}

remote\_passwd() {

echo -e "\n\nPlace ${FILE} on ${TARGET\_SERVER}\n"

SSH\_CMD="${SSH} ${SSH\_OPTIONS} ${SSH\_USER}@${TARGET\_SERVER}"

SSH\_RES=$(${SSH\_CMD} "sudo /usr/bin/mkdir -p /home/${username}; echo \$?")

if [[ ${SSH\_RES} -eq 0 ]]; then

${SCP} ${SSH\_OPTIONS} ${LOCAL\_FOLDER}/${FILE} ${SSH\_USER}@${TARGET\_SERVER}:/tmp/ > /dev/null 2>&1

${SSH\_CMD} "sudo /usr/bin/mv /tmp/${FILE} /home/${username}/${FILE}"

${SSH\_CMD} "sudo /usr/bin/chown -R ${username}:${gid} /home/${username}"

${SSH\_CMD} "sudo /usr/bin/chmod 0700 /home/${username}"

# Check if remote file got correct content

REMOTE\_FILE=$(${SSH\_CMD} "sudo /usr/bin/cat /home/${username}/${FILE}")

if [ "${REMOTE\_FILE}" == "${remote\_part}" ]; then

email\_passwd

house\_keeping

else

error\_exit "Problem to copy password file to remote server, re-run script"

fi

else

error\_exit "Problem to/on ${TARGET\_SERVER}, please report to DL-pam-linux <pam-linux@list.bmw.com>"

fi

}

set\_options() {

while getopts ":u:s:p:g:t:" flag ; do

case "${flag}" in

u) username=${OPTARG}

;;

t) tech\_user=${OPTARG}

;;

s) server=$(echo ${OPTARG} | awk '{print tolower($0)}')

;;

p) password=${OPTARG}

;;

g)

;;

\*) error\_exit "Unknow/Empty option ${OPTARG}"

;;

\? ) error\_exit "Invalid option: -${OPTARG} exiting"

;;

: ) error\_exit "Option -${OPTARG} requires an argument"

;;

esac

done

# Check on variables

email=`/usr/bin/ldapsearch -h ldap.muc -x -D 'cn=QQMAX00,ou=projects,o=BMW,dc=bmwgroup,dc=com ' -b "o=bmw,dc=bmwgroup,dc=com" "(uid=${username})" mail 2> /dev/null | grep -e "^mail:" | awk '{print $2}'`

if [ -z ${email} ]; then

error\_exit "Email for ${username} not found"

fi

if [ -z ${username} ] || [ -z ${password} ] || [ -z ${server} ] || [ -z ${tech\_user} ]; then

output "\nAll of the variables is not set\n" 3

USAGE

exit 1

else

check\_instserv

# Split and setup password

email\_part="${password:0:${#password}/2}"

remote\_part="${password:${#password}/2}"

echo $remote\_part > ${LOCAL\_FOLDER}/${FILE}

/usr/bin/chmod 0600 ${LOCAL\_FOLDER}/${FILE}

gid="qqbmw"

if [[ ${username} =~ ^qq ]]; then

gid="qqbmw"

elif [[ ${username} =~ ^qt ]]; then

gid="qtbmw"

elif [[ ${username} =~ ^qx ]]; then

gid="qxbmw"

elif [[ ${username} =~ ^q[0-9]+ ]]; then

gid="qbmw"

fi

output "username=${username},gid=${gid},email=${email},server=${server},techuser=${tech\_user}" 2

remote\_passwd

exit 0

fi

}

# # Menu: Share Password or decrypt password

while [[ -n $1 ]]; do

case $1 in

-h | --help)

help\_message

exit 0

;;

-g )

gen\_password

exit 0

;;

\*)

set\_options $@

;;

esac

shift

done

exit 0

[INSTSERV][qxz10kv][lpinstiaas02] /global/instserv/bin $

[INSTSERV][qxz10kv][lpinstiaas02] /global/instserv/bin $ cat netscan.sh

#!/bin/bash

# +-----------------------------------------------------------------------+

# | farmnetscan.sh |

# +-----------------------------------------------------------------------+

# | Description |

# | Check the given Farmserver, |

# | and show how many IP´s are used |

# | References |

# | c6000 |

# +---------------+-------------------------------------------------------+

# | Date | Discription |

# +---------------+-------------------------------------------------------+

# | 14.11.2018 | Version 0.1 |

# | 04.12.2018 | Version 0.1.1 (ssh added) |

# +---------------+-------------------------------------------------------+

#

source /global/instserv/bin/ssh.cred.lib.sh

ssh.cred.admin

source /global/instserv/bin/ssh.multiplex.lib.sh

################################# function display\_help ##################################

## displays the helpmessage when the script is called without parameter ##

##########################################################################################

function display\_help() {

# Display a help message

echo "Aufruf: "`echo $0` "[hostname]"

echo "Please name the hostname of a Xenfarm"

echo -e "All supnet connected to the Farm will be scanned and you get an overview of the free and used IP´s."

}

################################## start of the script ###################################

## Here the script starts ##

##########################################################################################

# Check if the first argument was null, and display help and exit if so

if [ -z "$1" ] ; then

# server=$(uname -n)

display\_help

else

subnets="$( sudo c6000 -b $1 | grep IP\_ADDRES | awk -F';' '{print $4}' | cut -d'.' -f1-3 | xargs )"

echo -e "All subnet connected to the Farm " $1 " will be scanned and you get an overview of the free and used IP´s."

m\_ssh $1 /bin/bash <<<'

IPNETS="'$subnets'";

printf "NET\_ID;PINGABLE;NOT\_RESPONDING\n";

for IPNET in $IPNETS; do

OUTPUT=$(

for ip in {10..254}; do

(

if ping -c 1 -w 1 $IPNET.$ip > /dev/null 2>&1; then

echo "pingable: $IPNET.$ip"

fi;

) &

done

wait;

)

ips=$( wc -l <<<"$OUTPUT" )

ipn=$(( 244 - ips ))

#printf "Es sind %s vergeben und %s IP´s im Netz %s frei \n" "$ips" "$ipn" "$IPNET"

printf "%s;%s;%s\n" "$IPNET.0" "$ips" "$ipn"

done

'

fi

[INSTSERV][qxz10kv][lpinstiaas02] /global/instserv/bin $

configure\_veritas.sh hardcode\_dmz\_hosts lpfc\_qd\_adjust restoreinfo.sh sh

[INSTSERV][qxz10kv][lpinstiaas02] /global/instserv/bin $ cat harden-ssh

#!/usr/bin/perl

use strict;

use warnings;

use Data::Dumper;

use lib "/global/instserv/bin/perl";

use BMW::SSH;

use BMW::Tools;

use BMW::Hook;

use BMW::Server;

use BMW::Rudder;

use Data::Compare;

use JSON;

$SIG{\_\_DIE\_\_} = \&ABORT;

$|=1;

chdir("/global/instserv/bin") or die "Could not chdir to /global/instserv/bin: $!";

my $parser = BMW::Hook->new();

my $rc = $parser->parse({

help => "Manage SSH Hardening on managed SLES Systsems",

params => {

'env=s' => "Use custom Rudder environment and overwrite default routine.",

'hostname=s' => "Hostname/FQDN of the target Rudder Node.",

'verbose+' => "Add multiple levels of verbosity",

'quiet' => "Disable info messages",

},

required => [ 'hostname' ],

actions => {

## Initialization at the very beginning.

'check' => {

help => "Check current status of SSH Hardening of system",

},

'preregulated' => {

help => "SET BACK to status PRE-REGULATED - Not to be used other than fallback!",

},

'regulated2022' => {

help => "SET SSH HARDENING TO REGULATED",

},

'secure' => {

help => "DEPRECATED action - SSH Hardening is administered via SODB security flags.Check https://atc.bmwgroup.net/confluence/display/LINUX/Implementing+SSH+Security+Guidelines+on+SLES+Systems",

},

'insecure' => {

help => "DEPRECATED action - SSH Hardening is administered via SODB security flags.Check https://atc.bmwgroup.net/confluence/display/LINUX/Implementing+SSH+Security+Guidelines+on+SLES+Systems",

},

'revert' => {

help => "DEPRECATED action - SSH Hardening is administered via SODB security flags.Check https://atc.bmwgroup.net/confluence/display/LINUX/Implementing+SSH+Security+Guidelines+on+SLES+Systems",

},

},

});

my %options = %{ $parser->result() };

my $action = $parser->action();

my $node\_host = $options{hostname} ;

my $LOGLEVEL;

$LOGLEVEL = $BMW::Tools::LOGLEVEL = $options{quiet} ? 0 : ($options{verbose} // $ENV{LOGLEVEL} // 1);

my $Server = BMW::Server->new($node\_host) || ABORT "Error during retrieval of server $node\_host";

INFO "Found Server: " . $Server->{fqdn} . " / " . ($Server->{RAW}->{S\_LOG\_ID} // '<no slog>'), 2;

DEBUG "UUID in DB: " . ($Server->getRudderUUID // "<undef>"), 2;

my $RudderEnvironment;

if (defined $options{env}) {

$RudderEnvironment = $options{env};

INFO "Overriding Rudder Environment with CLI provided '$RudderEnvironment'";

} else {

$RudderEnvironment = $Server->getRudderEnvironment;

INFO "Determined the environment: " . $Server->getRudderEnvironment, 2;

}

my $RN = BMW::Rudder::Node->new( name => $Server->{fqdn}, environment => $RudderEnvironment );

my $Properties = $RN->getProperty;

die "No node properties found" unless scalar keys %{$Properties};

my ($preregulated, $secure) = ($Properties->{SSHD\_PRE\_REGULATED\_VERSION} ? 1 : 0, $Properties->{SSHD\_INSECURE} ? 0 : 1);

my $key;

if ($action eq 'check') {

my @sshkeys = sort grep {/^SSH/i} keys %{$Properties};

if (@sshkeys) {

WARN "Current Rudder node properties that are set:";

for $key (@sshkeys) {

WARN "- %s = %s", [ $key, $Properties->{$key} // "(not set)" ];

}

if ($preregulated) {

WARN "System state : UNREGULATED (NOT OK)";

INFO "Results in : %s",

[ $secure ?

"SECURE (OK)" :

"INSECURE (Acceptable)"

];

} else {

INFO "System state : REGULATED (OK)";

INFO "System mode : SECURE (OK)" if ($secure);

WARN "System mode : INSECURE (Acceptable)" if (not $secure);

INFO "Justification: %s", [ $Properties->{SSHD\_INSECURE} ne 'true' ? $Properties->{SSHD\_INSECURE} : "<none>" ] if not $secure;

}

} else {

INFO "No dedicated node properties for SSH settings: System inherits defaults or group settings."

}

}

if ($action eq 'secure') {

INFO "DEPRECATED SSH Hardening is administered via SODB security flags.Check https://atc.bmwgroup.net/confluence/display/LINUX/Implementing+SSH+Security+Guidelines+on+SLES+Systems"

}

if ($action eq 'insecure') {

INFO "DEPRECATED SSH Hardening is administered via SODB security flags.Check https://atc.bmwgroup.net/confluence/display/LINUX/Implementing+SSH+Security+Guidelines+on+SLES+Systems"

}

if ($action eq 'preregulated') {

if ($preregulated) {

INFO "System state : Kept in PREREGULATED";

} else {

WARN "System state : PREREGULATED -> REGULATED (changed)";

INFO "Deleting property: SSHD\_PRE\_REGULATED\_VERSION", 2;

$RN->setProperty('SSHD\_PRE\_REGULATED\_VERSION' => "done ia harden-ssh script" // 'true');

}

}

if ($action eq 'regulated2022') {

if ($preregulated) {

WARN "System state : UNREGULATED -> REGULATED (changed)";

INFO "Deleting property: SSHD\_PRE\_REGULATED\_VERSION", 2;

$RN->delProperty('SSHD\_PRE\_REGULATED\_VERSION');

} else {

INFO "System state : Kept in REGULATED";

}

}

if ($action eq 'revert') {

INFO "DEPRECATED SSH Hardening is administered via SODB security flags.Check https://atc.bmwgroup.net/confluence/display/LINUX/Implementing+SSH+Security+Guidelines+on+SLES+Systems"

}

exit 0;

[INSTSERV][qxz10kv][lpinstiaas02] /global/instserv/bin $

[INSTSERV][qxz10kv][lpinstiaas02] /global/instserv/bin $ cat sap\_show\_fs\_layout.sh

#!/bin/bash

set -a

#set -u

#set -x

# title: SAP Server get Storage Requirements

# description: get SAP Server get Storage Requirements

# from the DCODB

# author: walter.kemmerer@partner.bmw.de

# creation date: 23.12.2016

# initialy checked by: Walter Kemmerer

# history: <date> <version> <modification>

usage() {

cat << EOF

`basename $0` - get SAP Server get Storage Requirements from the DCODB

usage: `basename $0` <sap\_server>

EOF

exit 1

}

function enumerate\_addons {

ADDONS\_ARRAY="$(echo "${ADDONS}" | sed -e 's/\,/\n/g'| cut -f1-3 -d";")"

while read -r EACH\_ADDON

do

while IFS=$';' read -r ADDON ADDON\_SID ADDON\_PKGNAME SIZE

do

ADDON\_PKG\_IP="$(host ${ADDON\_PKGNAME}${MIG\_SUFFIX} | egrep -v 'alias|NXDOMAIN|IPv6' | awk '{print $NF}' )"

echo "${ADDON}\_PKGNAME = ${ADDON\_PKGNAME}"

echo " ${ADDON}\_SID = ${ADDON\_SID}"

echo " ${ADDON}\_PKG\_IP = $ADDON\_PKG\_IP"

done <<<"${EACH\_ADDON}"

done <<<"${ADDONS\_ARRAY}"

}

function get\_migration\_mode {

if ( [ "${MIGRATION\_TYPE}" = "None" ] && [ "${MIGRATION}" = "No" ] && ( [ "${MIG\_PACKAGE\_NEEDED}" = "No" ] || [ "${MIG\_PACKAGE\_NEEDED}" = "null" ] ))

then

MIG\_SUFFIX=""

MIGRATION\_MODE=false

elif ( [ "${MIGRATION\_TYPE}" = "LUN\_Migration" ] && [ "${MIG\_PACKAGE\_NEEDED}" = "No" ] && [ "${MIGRATION}" = "Yes" ] )

then

MIGRATION\_MODE=true

MIG\_SUFFIX=""

elif ( [ "${MIG\_PACKAGE\_NEEDED}" = "Yes" ] && [ "${MIGRATION}" = "Yes" ] )

then

MIG\_SUFFIX="n"

MIGRATION\_MODE=false

else

echo "ERROR: got no clear values about Migration Package Names and Addresses"

echo "got the values from the DCODB:

migrationtype ${MIGRATION\_TYPE}

migration ${MIGRATION}

migration\_package\_needed ${MIG\_PACKAGE\_NEEDED}"

echo ""

echo "ABORTING: ... bye!!"

exit 1

fi

}

############################

# SET VARIABLES

############################

MAIN\_SCRIPT=$(/bin/readlink -f $0)

MAIN\_PATH=$(/usr/bin/dirname $MAIN\_SCRIPT)

VM\_IMAGES="\

-I sles11sp1\_x86\_64 -T sap

-I sles11sp2\_x86\_64 -T sap

-I sles11sp3\_x86\_64 -T sap

-I sles11sp4\_x86\_64 -T sap

-I sles11sp5\_x86\_64 -T sap

-I sles12sp2

-I sles12sp3

-I sles12sp4

-I sles12sp5

-I sles12sp6"

PHYS\_IMAGES="\

-I sles11sp1\_x86\_64 -T sap

-I sles11sp2\_x86\_64 -T sap

-I sles11sp3\_x86\_64 -T sap

-I sles11sp4\_x86\_64 -T sap

-I sles11sp5\_x86\_64 -T sap

-I sles11sp5\_x86\_64 -T sap

-I sles12sp2

-I sles12sp3

-I sles12sp4

-I sles12sp5

-I sles12sp6"

VXVM\_IMAGES="\

-I sles11sp1\_x86\_64 -T sap\_vxvm

-I sles11sp2\_x86\_64 -T sap\_vxvm

-I sles11sp3\_x86\_64 -T sap\_vxvm

-I sles11sp4\_x86\_64 -T sap\_vxvm

-I sles11sp5\_x86\_64 -T sap\_vxvm

-I sles11sp5\_x86\_64 -T sap\_vxvm

-I sles12sp2

-I sles12sp3

-I sles12sp4

-I sles12sp5

-I sles12sp6"

red='\e[4;31m'

orange='\e[4;31m'

NC='\e[0m' # No Color

SAP\_SERVER="$1"

if [ -z "${SAP\_SERVER}" ]; then usage ; exit 1 ; fi

if [ -f "/global/instserv/config/local/REST\_DCODB.conf" ] ; then

for i in $(cat /global/instserv/config/local/REST\_DCODB.conf)

do

declare -x "$i"

done

else

echo "ERROR: could not find /global/instserv/config/local/REST\_DCODB.conf"

exit 1

fi

## get DATA from DCODB (SAPEX)

SID\_DATA="$(/usr/bin/curl -s -k \

--user "${username}":"${password}" -H "Accept: text/plain" \

https://dcodb.bmwgroup.net/dcodb/rest/data/sodb/V\_SAP\_OS\_LAYOUT\_BASE\_DATA?servername=${SAP\_SERVER})"

if [ "${SID\_DATA}" = "No rows found for the provided query." ] ; then echo "${SID\_DATA}" ; exit 1 ; fi

if [ -z "${SID\_DATA}" ] ; then echo "INFO: sorry, no data for ${SAP\_SERVER} found in the SAPEX Database" ; exit 1 ; fi

read -r ID SID VCS SERVERNAME SEC\_SERVERNAME SERVER\_TYPE INSTANCE\_TYPE MIGRATION\_TYPE PKGNAME SAPMNT\_SOURCE SST\_DATA\_SOURCE ARCHIVE\_SOURCE TRANS\_SOURCE SAP\_ENVIR INSTANCE\_NR ADDONS DB DBSIZE ORAPKGNAME MIGRATION DISKLAYOUT PUBLISHED OS\_VERSION MIG\_PACKAGE\_NEEDED LAST\_UPDATED LC\_SID DATE\_PUBLISHED < <(echo "${SID\_DATA}" | tail -1 | sed -e 's/ /\_/g' | sed -e 's/\_|\_/ /g')

declare -x SID

declare -x VCS

declare -x SERVERNAME

declare -x SEC\_SERVERNAME

declare -x SERVER\_TYPE

declare -x INSTANCE\_TYPE

declare -x MIGRATION\_TYPE

declare -x PKGNAME

declare -x ADDONS

declare -x DB

declare -x DBSIZE

declare -x ORAPKGNAME

declare -x MIGRATION

declare -x DISKLAYOUT

declare -x OS\_VERSION

declare -x MIG\_PACKAGE\_NEEDED

declare -x LC\_SID

### echo Installation Instructions for the UNIX Admin ####

echo "### SAP Server Installation intructions #######"

get\_migration\_mode

IP\_PKGNAME="$(host ${PKGNAME}${MIG\_SUFFIX} | egrep -v 'alias|NXDOMAIN|IPv6' | awk '{print $NF}')"

if [ -z ${IP\_PKGNAME} ] ; then IP\_PKGNAME="${orange}IP not assigned yet${NC}: inform ready.operate team" ; fi

if ( [ "${DB}" = "Yes" ] && [ "${ORAPKGNAME}" != "null" ] ) ; then

IP\_ORAPKGNAME="$(host ${ORAPKGNAME}${MIG\_SUFFIX} | egrep -v 'alias|NXDOMAIN|IPv6' | awk '{print $NF}')"

if [ -z ${IP\_ORAPKGNAME} ] ; then IP\_ORAPKGNAME="${orange}IP not assigned yet${NC}: inform ready.operate team" ; fi

fi

if [ "${ADDONS}" != "null" ] ; then

ADDONS\_EXPANDED="$(enumerate\_addons)"

else

ADDONS\_EXPANDED="ADDONS = none"

fi

if [ "${SERVER\_TYPE}" = "Virtual" ] ; then

echo "checking for AUTO\_SAN. Please be patient"

AUTO\_SAN\_AVAILABLE=$(/global/instserv/bin/check4autosan.pl "${SAP\_SERVER}" | egrep "is a VM and has AutoSAN:" | wc -l )

FARM\_SERVERS="$(/global/instserv/bin/cmdb ${SAP\_SERVER} -x \

| egrep '^farmservers:' | cut -f2 -d":" | sed -e 's/,//g' | awk '{print $1, $2}' )"

if [ "${OS\_VERSION}" = "SLES11"\* ] ; then

IMAGE\_TEMPL="$(echo "${VM\_IMAGES}" | egrep -i "${OS\_VERSION}")"

else

IMAGE\_TEMPL="$(echo "${VM\_IMAGES}" | egrep -i "${OS\_VERSION}")"

fi

ADDON\_TYPES=$(echo ${ADDONS\_EXPANDED} | sed -e 's/ /|/g')

SAP\_INSTANCES="${INSTANCE\_TYPE}|${ADDON\_TYPES}"

DGLAYOUT=$(echo ${DISKLAYOUT} | sed -e 's/Disk//')

if [ "${OS\_VERSION}" = "SLES11"\* ] ; then

if ( [ "${DB}" = "No" ] && [[ "${INSTANCE\_TYPE}" = "C"[I,S] ]] ) ; then

INSTANCE\_TYPE="${INSTANCE\_TYPE}\_ONLY"

FS\_LAYOUT\_TEMPLATE=$(ls -1 /global/instserv/data/BMW-CFG-SAP-VM-FS-LAYOUT | egrep "template\_" | egrep ${DGLAYOUT} | egrep ${INSTANCE\_TYPE} | head -1 )

elif [ "${ADDONS}" = "null" ] ; then

FS\_LAYOUT\_TEMPLATE=$(ls -1 /global/instserv/data/BMW-CFG-SAP-VM-FS-LAYOUT | egrep "template\_" | egrep ${DGLAYOUT} | egrep ${INSTANCE\_TYPE} | egrep -v "LC|SDB|ONLY" | head -1 )

elif [[ "${ADDONS}" = \*";"\* ]] ; then

ADDON\_LIST=$( echo "${ADDONS}" | sed -e 's/;/\n' | cut -f1 -d",")

FS\_LAYOUT\_TEMPLATE=$(ls -1 /global/instserv/data/BMW-CFG-SAP-VM-FS-LAYOUT | egrep "template\_" | egrep ${DGLAYOUT} | egrep ${INSTANCE\_TYPE} | egrep ${ADDON\_LIST} | head -1 )

fi

else

FS\_LAYOUT\_TEMPLATE="manual"

fi

echo -e "Server Installation: ${red}No${NC} OS Finalization Task"

echo " you will need to run the SAP\_Prereq in this task"

sleep 4

echo ""

if [ "${AUTO\_SAN\_AVAILABLE}" = \*"is a VM and has AutoSAN:"\* ] ; then

echo "AUTO\_SAN Installation: from the approriate install server"

echo "use vm\_create using ${IMAGE\_TEMPL}"

echo "See https://bsswiki.muc/tiki-index.php?page=bss\_unix\_linux\_inst\_base\_virt-autosan&structure=bss\_unix&page\_ref\_id=1141"

else

echo "No AUTO\_SAN available:"

echo "Create Virtual Server on ${FARM\_SERVERS}: vmc create -d \"xxx xxx\" ${IMAGE\_TEMPL} ${SAP\_SERVER}"

fi

echo ""

echo "attach the Data LUN's to the VM"

if [ -z "${FS\_LAYOUT\_TEMPLATE}" ] ; then

echo "ERROR: could not find a Template under /global/instserv/data/BMW-CFG-SAP-VM-FS-LAYOUT/"

echo "Please contact L2"a

elif [ "${FS\_LAYOUT\_TEMPLATE}" = "manual" ] ; then

echo "You will need to manually setup the filesystems using the information below"

else

chmod 755 /global/instserv/data/BMW-CFG-SAP-VM-FS-LAYOUT

if [ -f "/global/instserv/data/BMW-CFG-SAP-VM-FS-LAYOUT/${SAP\_SERVER}.BMW-CFG-SAP-VM-FS-LAYOUT.conf" ]

then

echo "template file exists"

ls -l /global/instserv/data/BMW-CFG-SAP-VM-FS-LAYOUT/${SAP\_SERVER}.BMW-CFG-SAP-VM-FS-LAYOUT.conf

echo "please check if it is valid"

else

cp /global/instserv/data/BMW-CFG-SAP-VM-FS-LAYOUT/${FS\_LAYOUT\_TEMPLATE} /global/instserv/data/BMW-CFG-SAP-VM-FS-LAYOUT/${SAP\_SERVER}.BMW-CFG-SAP-VM-FS-LAYOUT.conf

echo "fillout the copied template:"

ls -l /global/instserv/data/BMW-CFG-SAP-VM-FS-LAYOUT/${SAP\_SERVER}.BMW-CFG-SAP-VM-FS-LAYOUT.conf

echo ".. then install the Override BMW-CFG-SAP-VM-FS-LAYOUT"

fi

fi

elif ( [ "${SERVER\_TYPE}" = "Physical" ] && [ "${DB}" = "Yes" ] && [ "${MIGRATION\_MODE}" = "false" ] ) ; then

IMAGE\_TEMPL="$(echo "${VXVM\_IMAGES}" | egrep -i "${OS\_VERSION}")"

echo "Server Installation: has a Finalization Task"

echo ""

echo "Install Server: install\_server.sh ${IMAGE\_TEMPL} ${SAP\_SERVER}"

echo ""

if [ "${VCS}" = "Yes" ] ; then

TEMPL\_TYPE=cluster

if [[ "${OS\_VERSION}" = "SLES11"\* ]] ; then

echo "install\_override.sh ${SERVERNAME} BMW-SW-VERITAS-Cluster-Suite"

echo "install\_override.sh ${SEC\_SERVERNAME} BMW-SW-VERITAS-Cluster-Suite"

else

echo "INFO: Veritas Cluster Software should auomatically be installed by Rudder"

fi

else

TEMPL\_TYPE=standalone

fi

DGLAYOUT=$(echo ${DISKLAYOUT} | sed -e 's/Disk//')

if [ "${ADDONS}" = "null" ] ; then

FS\_LAYOUT\_TEMPLATE=$(ls -1 /global/instserv/data/CONFIGURE\_VERITAS/INSTALL\_TEMPLATES | egrep "inst\_sap\_" | egrep ${DGLAYOUT} | egrep ${INSTANCE\_TYPE} | egrep ${TEMPL\_TYPE} | egrep -v 'SDB|LC' )

elif [[ "${ADDONS}" = \*";"\* ]] ; then

ADDON\_LIST=$( echo "${ADDONS}" | cut -f1 -d";")

echo ""

FS\_LAYOUT\_TEMPLATE=$(ls -1 /global/instserv/data/CONFIGURE\_VERITAS/INSTALL\_TEMPLATES | egrep "inst\_sap\_" | egrep ${DGLAYOUT} | egrep ${INSTANCE\_TYPE} | egrep ${ADDON\_LIST} | egrep ${TEMPL\_TYPE} )

fi

if [ -z "${FS\_LAYOUT\_TEMPLATE}" ] ; then

echo "no appropriate configure\_veritas.sh template found under:"

echo " /global/instserv/data/CONFIGURE\_VERITAS/INSTALL\_TEMPLATES/"

echo "please contact L2"

else

echo "Create and install a configure\_veritas.sh template:"

echo "cp template from /global/instserv/data/CONFIGURE\_VERITAS/INSTALL\_TEMPLATES/${FS\_LAYOUT\_TEMPLATE}"

echo "to /global/instserv/data/CONFIGURE\_VERITAS"

echo "fill out the template, then run configure\_veritas.sh <path\_to\_file/edited\_template\_file>"

fi

elif ( [ "${SERVER\_TYPE}" = "Physical" ] && [ "${MIGRATION\_MODE}" = "true" ] ) ; then

IMAGE\_TEMPL="$(echo "${VXVM\_IMAGES}" | egrep -i "${OS\_VERSION}")"

echo "Server Installation: has a Finalization Task"

echo ""

echo "Install Server: install\_server.sh ${IMAGE\_TEMPL} ${SAP\_SERVER}"

echo ""

if [ "${VCS}" = "Yes" ] ; then

TEMPL\_TYPE=cluster

if [[ "${OS\_VERSION}" = "SLES11"\* ]] ; then

echo "install\_override.sh ${SERVERNAME} BMW-SW-VERITAS-Cluster-Suite"

echo "install\_override.sh ${SEC\_SERVERNAME} BMW-SW-VERITAS-Cluster-Suite"

else

echo "INFO: Veritas Cluster Software should auomatically be installed by Rudder"

fi

else

TEMPL\_TYPE=standalone

fi

echo "INFO: this a new server for the Miration Typ LUN Import"

echo " so no filesystems need be setup. "

echo " LUN's will be imported from the old servers at a later date."

elif ( [ "${SERVER\_TYPE}" = "Physical" ] && [ "${VCS}" = "No" ] && [ "${INSTANCE\_TYPE}" != "SDB" ] ) ; then

if (( [[ "${INSTANCE\_TYPE}" = [C,D][I,S] ]] || [ "${INSTANCE\_TYPE}" = "TREX" ] || [ "${INSTANCE\_TYPE}" = "DP" ] \

|| [ "${INSTANCE\_TYPE}" = "AR" ] || [ "${INSTANCE\_TYPE}" = "MDM" ] ) && [ "${DB}" = "No" ] ) then

IMAGE\_TEMPL="$(echo "${PHYS\_IMAGES}" | egrep -i "${OS\_VERSION}")"

echo "Server Installation: No OS Finalization Task"

echo ""

echo "Install Server: install\_server.sh ${IMAGE\_TEMPL} ${SAP\_SERVER}"

echo ""

echo "No SAN LUN's: you need to copy the script from:"

INSTSERV=$(uname -n)

if [[ "${INSTSERV}" = \*"cms"\* ]] ; then

ls -1 ${MAIN\_PATH}/fs\_setups/create\_PHYS\* | egrep "${INSTANCE\_TYPE}"

else

ls -1 /global/instserv/data/BMW-SW-SAP\_PREREQ\_2012\_v1/scripts/create\_PHYS\* | egrep "${INSTANCE\_TYPE}"

fi

echo "onto your server, fill out the required fields in the script"

echo "then run the script directly on the server to create the volumes in LVM vglocal"

fi

fi

if [ "${MIGRATION\_MODE}" = "false" ] ; then

if [ "${SERVER\_TYPE}" != "Virtual" ] ; then

echo "fill out the FS Layout template using the information below:"

fi

echo ""

echo "SAP SERVER Information:"

echo ""

echo -e "\

SID = $SID

VCS = $VCS

SERVERNAME = $SERVERNAME

SEC\_SERVERNAME = $SEC\_SERVERNAME

SERVER\_TYPE = $SERVER\_TYPE

INSTANCE\_TYPE = $INSTANCE\_TYPE

PKGNAME = $PKGNAME

IP\_PKGNAME = $IP\_PKGNAME

DB = $DB

ORAPKGNAME = $ORAPKGNAME

IP\_ORAPKGNAME = $IP\_ORAPKGNAME

${ADDONS\_EXPANDED}

DISKLAYOUT = $DISKLAYOUT\

"

echo ""

echo "Diskgroups:"

echo ""

/usr/bin/curl -s -k --user "${username}":"${password}" -H "Accept: text/plain"\

https://dcodb.bmwgroup.net/dcodb/rest/data/sodb/V\_SAP\_OS\_LAYOUT\_DG\_STORAGE?servername=${SAP\_SERVER} |\

egrep -v "DB Schema|DB Table|^$" | sort -t "|" -n --key=2 | sed -e "s/^${SAP\_SERVER} | /${SAP\_SERVER} | dg${PKGNAME}0/" | sed -e "s/^${SAP\_SERVER} | dg${PKGNAME}010/^${SAP\_SERVER} | dg${PKGNAME}10/" | column -t

echo ""

echo "Filesystem Layout:"

echo ""

/usr/bin/curl -s -k --user "${username}":"${password}" -H "Accept: text/plain" \

https://dcodb.bmwgroup.net/dcodb/rest/data/sodb/V\_SAP\_OS\_LAYOUT\_FILESYSTEM?servername=${SAP\_SERVER} |\

egrep -v "DB Schema|DB Table|^$" | sort -t "|" -n --key=5 | awk '{print $1" | "$9" | "$7" | "$5" | "$3}' | sed -e "s/^${SAP\_SERVER} | /${SAP\_SERVER} | dg${PKGNAME}0/" | column -t

echo ""

echo "when Storage & Filesystems have been setup, go ahead and install the SAP\_Prereq:"

elif [ "${MIGRATION\_MODE}" = "true" ] ; then

echo "go ahead and install the SAP\_Prereq:"

fi

if [ "${OS\_VERSION}" = "SLES12SP2" ] ; then

echo "this is an SLES12 Server"

echo "see sles12 wiki"

else

echo "sap\_create\_prereq\_template.sh ${SAP\_SERVER}"

echo "install\_override.sh ${SAP\_SERVER} BMW-SW-SAP\_PREREQ\_2012\_v1 | tee ${HOME}/${SAP\_SERVER}\_preq.log"

echo "egrep 'ERROR|WARN' ${SAP\_SERVER}\_preq.log"

fi

[INSTSERV][qxz10kv][lpinstiaas02] /global/instserv/bin $

[INSTSERV][qxz10kv][lpinstiaas02] /global/instserv/bin $ cat vm\_upgrade\_sles12.sh

#!/bin/bash

################

# description: Reinstalls a SLES11 VM to SLES12

# author: Alexander Brunhirl <alexander.brunhirl@partner.bmw.de>

# information:

################

# global variabls

PATH=${PATH}:/lfs/opt/nagios/bin

[[ -f /etc/profile.d/instserv.sh ]] && source /etc/profile.d/instserv.sh

# usage() - print usage

# @arg@: -

# Return: print usage

function usage() {

cat <<-USAGE

${0##\*/} [check|preparation|reinstall] --server <SERVER FQDN> --disk "<ARRAY-LDEV> <ARRAY-LDEV>"

Supported modules ORACLE and SAP

MODE:

check Check if the requierements are fulfilled

preparation Check farmserver and filesystem layout (ORACLE only)

reinstall Reinstall the server

OPTIONS:

-e|--env Rudder environment (optional)

-s|--server Server name (FQDN)

-d|--disk New 112GB OS disks

-I|--image Target image for the installation (supported: sles12sp2, sles12sp4; default: sles12sp2)

--nobackup Skip fullbackup (optional)

--nonagios Do not rebuild Nagios config after installation (optional)

--nocheck Skip most sanity checks, handle carefully (optional)

--standalone Reinstalls the VM even if only the current Xen Farmserver is SLES12 (non standard)

EXAMPLE:

${0##\*/} check --server ltbssabr01.bmwgroup.net --disk "97609-505a 97718-505a"

${0##\*/} preparation --server ltbssabr01.bmwgroup.net --disk "97609-505a 97718-505a"

${0##\*/} reinstall --server ltbssabr01.bmwgroup.net --disk "97609-505a 97718-505a"

${0##\*/} reinstall --server ltbssabr02.bmwgroup.net --disk "97609-505b 97718-505b" --env QA

USAGE

return 0

}

# log\_msg() - write message into logfile

# @arg@: message

# Return: -

function log\_msg() {

printf "%s\n" "${\*}" >> ${LOG}

}

# err\_msg() - print stderr message and log output

# @arg@ error message

# Return: print error message

function err\_msg() {

printf "%s\n" "ERROR: ${\*}" >&2

log\_msg "ERROR: ${\*}"

}

# std\_msg() - print stdout message and log output

# @arg@: message

# Return: print text

function std\_msg() {

printf "%s\n" "${\*}"

log\_msg "${\*}"

}

# cleanup() - clean up function

# @arg@: -

# Return:

function cleanup() {

if [[ ${?} -eq 0 ]]; then

echo ""

std\_msg "${MODE:-check or reinstallation} successful."

else

echo ""

err\_msg "${MODE:-check or reinstallation} failed, please check the logs."

err\_msg "LOG: ${LOG}"

fi

}

# get\_environment() - get rudder environment

# @arg1: Farmserver (FQDN)

# Return: Rudder Environment

function get\_environment() {

local SRV="${1}"

local OUT=$( get\_server\_dcodb\_infos.pl --server ${SRV%%.\*} --params LOC\_CAMPUS | awk -F '=' 'gsub(/\047/, ""){print $2}' )

[[ -z ${OUT} ]] && { err\_msg "Wrong output, please check 'get\_server\_dcodb\_infos.pl\`."; exit 1; }

case ${OUT} in

10.\*)

echo "BSS12/AMERICAS"

return 0

;;

19.\*)

echo "BSS\_APAC"

return 0

;;

\*)

if get\_server\_dcodb\_infos.pl --server ${SRV%%.\*} --params NET\_ID | awk -F '=' 'gsub(/\047/, ""){print $2}' | grep -q -P "^160\.50\.(?:5[0-9]|60)\.0$"

then

echo "BSS12/MUC\_BANK"

return 0

else

echo "BSS12/EMEA"

return 0

fi

;;

esac

}

# check\_logic() - check if

# @arg1: SERVER (FQDN)

# @arg2: OS Disk 1 (ARRAY-LDEV)

# @arg3: OS Disk 2 (ARRAY-LDEV)

# Return: 0 - successful

# 1 - failed

function check\_logic() {

log\_msg "DEBUG: ${FUNCNAME[0]} ${\*}"

local SERVER=${1}

local DISK1=( ${2%%-\*} ${2##\*-} )

local DISK2=( ${3%%-\*} ${3##\*-} )

# check SAN

if [[ ${DISK1[0]} == ${DISK2[0]} ]]; then

err\_msg "Same SAN ARRAY, please check OS disks for typos."; return 1

fi

if [[ ${DISK1[1]} != ${DISK2[1]} ]]; then

err\_msg "Not the same SAN LDEVs, please check OS disks for typos."; return 1

fi

return ${?}

}

# check\_oracle\_vm() - check if the ORACLE VM meet all the requierements

# @arg1: Server (FQDN)

# Return: 0: successful

# 1-255: failed

function check\_oracle\_vm() {

log\_msg "DEBUG: ${FUNCNAME[0]} ${\*}"

local VM=${1}

rootssh ${VM} '/bin/bash' <<-'SSH' 2>&1 | tee -a ${LOG}

oracle=175

oracle\_audit=50

oracle\_vg=224

SuSE=$( awk '/^VERSION/{ V=$3 } /^PATCHLEVEL/{ P=$3 } END{print V"."P}' /etc/SuSE-release )

case ${SuSE} in

11.2|11.3|11.4) ;;

12.2|12.4) echo "ERROR: Already a SLES12 VM" >&2; exit 1;;

\*) echo "ERROR: Unsupported OS Release on $(uname -n)" >&2; exit 1;;

esac

FSYSTEME=( $( awk '/\/lfs\/oracle\>/ {print $1}' /proc/mounts ) )

FSYSTEME+=( $( awk '/\/lfs\/oracle\_audit\>/ {print $1}' /proc/mounts ) )

LVS\_1=( $( /sbin/lvs --options vg\_name,lv\_name,lv\_size --units g --noheadings ${FSYSTEME[0]} ) )

LVS\_2=( $( /sbin/lvs --options vg\_name,lv\_name,lv\_size --units g --noheadings ${FSYSTEME[1]} ) )

if [[ "${LVS\_1[0]}" != "dgoracle" || "${LVS\_2[0]}" != "${LVS\_2[0]}" ]]; then

echo -e "ERROR: ORACLE Filesystems are not on the correct volume group." >&2

echo -e "SLES12 Oracle Requirements:\nSize\tVG" >&2

echo -e "224GB\t/dev/dgoracle" >&2

exit 1

else

VGS=( $( /sbin/vgs --options vg\_name,vg\_size --units g --noheadings /dev/${LVS\_1[0]} ) )

# lvs is not rounding (49.96g => 49g), hence we add +1GB

if ! [[ $(( ${LVS\_1[2]%.\*} +1 )) -ge ${oracle} && $(( ${LVS\_2[2]%.\*} +1 )) -ge ${oracle\_audit} && ${VGS[1]%.\*} -ge ${oracle\_vg} ]]; then

echo -e "ERROR: ORACLE Filesystem size does not match the requirements." >&2

echo -e "SLES12 Oracle Requirements:\nSize\tDevice\t\t\t\tMountpoint" >&2

echo -e "175GB\t/dev/dgoracle/lvoracle\t\t/lfs/oracle" >&2

echo -e " 50GB\t/dev/dgoracle/lvoracleaudit\t/lfs/oracle\_audit" >&2

exit 1

fi

fi

if [[ "${LVS\_1[1]}" != "lvoracle" || "${LVS\_2[1]}" != "lvoracleaudit" ]]; then

echo -e "ERROR: ORACLE Filesystems are not in the correct logical volume." >&2

echo -e "SLES12 Oracle Requirements:\nSize\tDevice\t\t\t\tMountpoint" >&2

echo -e "175GB\t/dev/dgoracle/lvoracle\t\t/lfs/oracle" >&2

echo -e " 50GB\t/dev/dgoracle/lvoracleaudit\t/lfs/oracle\_audit" >&2

exit 1

fi

# check mounts

if ! /bin/mount -a; then

echo -e "ERROR: Old or missing mounts found please check '/etc/fstab'." >&2

exit 1

fi

SSH

[[ ${PIPESTATUS[0]} -ne 0 ]] && return 1

if [[ "${MODE}" != "preparation" ]]; then

rootssh ${VM} '/bin/bash' <<-'SSH' 2>&1 | tee -a ${LOG}

if ! [[ $( /usr/bin/lsof /lfs/oracle\_audit | grep -v 'rsyslogd' | grep -c 'oracle' ) -eq 0 ]]; then

echo -e "ERROR: It appears ORACLE processes are active under '/lfs/oracle\_audit', please check with the application team." >&2

/usr/bin/lsof /lfs/oracle\_audit | grep -v 'rsyslogd'

exit 1

fi

if ! [[ $( /usr/bin/lsof /lfs/oracle | grep -c 'oracle' ) -eq 0 ]]; then

echo -e "ERROR: It appears ORACLE processes are active under '/lfs/oracle', please check with the application team." >&2

/usr/bin/lsof /lfs/oracle

exit 1

fi

SSH

else

return 0

fi

return ${PIPESTATUS[0]}

}

# check\_sap\_vm() - check if the SAP VM meet all the requirements

# @arg1: Server (FQDN)

# Return: 0: successful

# 1-255: failed

function check\_sap\_vm() {

log\_msg "DEBUG: ${FUNCNAME[0]} ${\*}"

local VM=${1}

rootssh ${VM} '/bin/bash' <<-'SSH' 2>&1 | tee -a ${LOG}

SuSE=$( awk '/^VERSION/{ V=$3 } /^PATCHLEVEL/{ P=$3 } END{print V"."P}' /etc/SuSE-release )

case ${SuSE} in

11.2|11.3|11.4) ;;

12.2|12.4) echo "ERROR: Already a SLES12 VM" >&2; exit 1;;

\*) echo "ERROR: Unsupported OS Release" >&2; exit 1;;

esac

# check if SAP is running

if ! [[ $( awk '/\/usr\/sap\// {system("lsof " $2 " | wc -l")}' /proc/mounts ) -eq 0 ]]; then

echo -e "ERROR: It appears SAP processes are active under '/usr/sap/<SID>', please check with the application team." >&2

awk '/\/usr\/sap\// {system("lsof " $2)}' /proc/mounts

exit 1

fi

if ! [[ $( ipcs -a | awk 'BEGIN{i=0} /adm/&&!/sapadm/{i++} END{print i}' ) -eq 0 ]]; then

echo -e "ERROR: It appears SAP is still running, please check with the application team." >&2

exit 1

fi

# check mounts

if ! /bin/mount -a; then

echo -e "ERROR: Old or missing mounts found please check '/etc/fstab'." >&2

exit 1

fi

SSH

return ${PIPESTATUS[0]}

}

# check\_farmserver() - check if the farmserver meet all the requirements

# @arg1: Farmserver (FQDN)

# @arg2: Server (FQDN)

# @arg3: OS Disk 1 (ARRAY-LDEV)

# @arg4: OS Disk 2 (ARRAY-LDEV)

# Return: 0 - successful

# 1 - failed

function check\_farmserver() {

log\_msg "DEBUG: ${FUNCNAME[0]} ${\*}"

local FARM=${1}

local VM=${2%%.\*}

local DISK1=( ${3%%-\*} ${3##\*-} )

local DISK2=( ${4%%-\*} ${4##\*-} )

local SIZE=112G

local INST12=$( uname -n )

rootssh ${FARM} '/bin/bash' <<-SSH 2>&1 | tee -a ${LOG}

## check OS release

SuSE=\$( awk '/^VERSION/{ V=\$3 } /^PATCHLEVEL/{ P=\$3 } END{print V"."P}' /etc/SuSE-release )

case \${SuSE} in

12.2|12.4)

;;

\*) echo "ERROR: Unsupported Farmserver Release on \$( uname -n )" >&2

echo "INFO: If the exception is granted, the VM can be migrated to the SLES12 side and the script" >&2

echo "INFO: can be executed with the standalone option, for further information please check the help." >&2

exit 1

;;

esac

## check for SAN storage

STORAGE=\$( awk -F= '/^STORAGE/{print \$2}' /lfs/xen/config/vmc.conf )

if [[ "\${STORAGE}" != "SAN" ]]; then

echo "ERROR: Farmserver (\$(uname -n)) seems to be DRBD" >&2

exit 1

fi

## check new OS disk for header and correct size

if ! /lfs/xen/sbin/vmc -b storage | egrep -qE "${DISK1[0]},${DISK1[1]},${SIZE},.\*(${VM}[^\_]|N/A)"; then

/lfs/xen/sbin/vmc -b storage | egrep -E "${DISK1[0]},${DISK1[1]},${SIZE},.\*(${VM}[^\_]|N/A)"

echo "ERROR: SAN disk cannot be found, is in use or size is not correct." >&2; exit 1

fi

if ! /lfs/xen/sbin/vmc -b storage | egrep -qE "${DISK2[0]},${DISK2[1]},.\*(${VM}[^\_]|N/A)"; then

/lfs/xen/sbin/vmc -b storage | egrep -E "${DISK2[0]},${DISK2[1]},.\*(${VM}[^\_]|N/A)"

echo "ERROR: SAN disk cannot be found, is in use or size is not correct." >&2; exit 1

fi

## check SSH connection to SLES12 Installserver

if ! /usr/bin/nc -vz ${INST12} 22 &> /dev/null; then

echo "ERROR: Please check the SSH connection (\$(uname -n) -> ${INST12})." >&2

/usr/bin/nc -vz ${INST12} 22

fi

SSH

return ${PIPESTATUS[0]}

}

# prepare\_vm() - prepare VM for the upgrade

# @arg1: Server (FQDN)

# @arg2: Rudder Environment

# Return: 0: successful

# 1-255: failed

function prepare\_vm() {

log\_msg "DEBUG: ${FUNCNAME[0]} ${\*}"

local VM=${1}

local EV=${2:-missing}

cmdb ${VM%%.\*} |egrep "^OS|status|modul" >> ${LOG}

update\_dcodb\_status.py --set --release SLES12SP2 --status Installation \

--server ${VM} >> ${LOG} || return ${?}

cmdb ${VM%%.\*} |egrep "^OS|status|modul" >> ${LOG}

if [[ ${EV} == "missing" ]]; then

rudderctl seed --force\_polserv \

--hostname=${VM} --quiet || return ${?}

else

rudderctl seed --env=${EV} --force\_polserv \

--hostname=${VM} --quiet || return ${?}

fi

if [[ ${EV} != "missing" ]]; then

if grep -wq ${VM} /global/instserv/config/local/PolicyExceptions.list; then

if egrep -wq "${VM}.\*${EV}" /global/instserv/config/local/PolicyExceptions.list; then

log\_msg "DEBUG: Server already present in local Policy Exception list"

egrep "${VM}.\*${EV}" /global/instserv/config/local/PolicyExceptions.list >> ${LOG}

else

err\_msg "${VM} already in '/global/instserv/config/local/PolicyExceptions.list' with other environment, please check."

return 1

fi

else

sh -c "echo -e \"${VM}\t${EV}\" >> /global/instserv/config/local/PolicyExceptions.list"

egrep "${VM}.\*${EV}" /global/instserv/config/local/PolicyExceptions.list >> ${LOG}

fi

fi

}

# prepare\_sap\_vm() - collect SAP infos

# @arg1: Server (FQDN)

# Return: PKG-Name, PKG-Filesystem, KWOM-LRSQ, MQSeries Client, MQSeries Server, UCMD

function prepare\_sap\_vm() {

log\_msg "DEBUG: ${FUNCNAME[0]} ${\*}"

rootssh ${1} '/bin/bash' <<-'SSH'

PKG="$( awk '/\/global\/.\*\/apps/&&gsub(/\/global\//,"")&&gsub(/\/apps/,""){print $2}' /proc/mounts )"

PKGFS="$( df -hPT /global/${PKG}/apps | awk '/^\/dev/ {print $1}')"

LRSQ=$( rpm -qa | grep -c 'BMW-kwom-lrsq-client' )

MQSC=$( rpm -qa | grep -c 'BMW-SW-mqs-client' )

MQSS=$( rpm -qa | grep -c 'BMW-SW-mqs-server' )

UCMD=$( rpm -qa | grep -c 'BMW-SW-ucmd' )

[[ -z ${PKG} ]] && exit 1

[[ -z ${PKGFS} ]] && exit 1

echo -e "${PKG}\n${PKGFS}\n${LRSQ}\n${MQSC}\n${MQSS}\n${UCMD}"

SSH

return ${PIPESTATUS[0]}

}

# preprare\_farmserver() - update VMC related software

# @arg1: Farmserver

# Return: 0: successful

# 1-255: failed

function prepare\_farmserver() {

log\_msg "DEBUG: ${FUNCNAME[0]} ${\*}"

rootssh ${1} '/bin/bash' <<-'SSH' 2>&1 | tee -a ${LOG} &> /dev/null

REPO=( BMW-SLES12-VMC2 )

RPMS=(

perl-BMW-Helper

perl-BMW-Storage

perl-BMW-VMC

vmc

)

uname -n

zypper ref --repo ${REPO[@]}

zypper up --repo ${REPO[@]} --no-confirm --no-recommends ${RPMS[@]}

SSH

return ${PIPESTATUS[0]}

}

# backup\_vm() - backup the VM

# @arg1: Farmserver

# @arg2: VM (FQDN)

# @arg3: Full backup ('yes' or 'no')

# Return: 0: successful

# 1-255: failed

function backup\_vm() {

log\_msg "DEBUG: ${FUNCNAME[0]} ${\*}"

local FARM=${1%%.\*}

local VM=${2%%.\*}

local BACKUP=${3}

rootssh ${VM} '/bin/bash -x' <<-SSH 2>&1 | tee -a ${LOG} &> /dev/null

## print mounted filesystems and pvinfo

/bin/df -hPT

/sbin/pvs

SSH

if [[ ${BACKUP} == 'no' ]]; then

std\_msg "Skip VM fullbackup"

fi

rootssh ${FARM} '/bin/bash -x' <<-SSH 2>&1 | tee -a ${LOG} &> /dev/null

/lfs/xen/sbin/vmc info ${VM}; \

/lfs/xen/sbin/vmc storage ${VM}; \

if [[ ${BACKUP} != 'no' ]]; then

if [[ "${STANDALONE}" == "yes" ]]; then

/lfs/xen/sbin/vmc -e FARMSERVERS=${FARM} --full backup ${VM} || exit \${?}; \

else

/lfs/xen/sbin/vmc --full backup ${VM} || exit \${?}; \

fi; \

else

echo "DEBUG: skipping backup"; \

fi; \

tar -cvf /lfs/xen/config/${VM}\_config.tar /lfs/xen/config/${VM}/

SSH

return ${PIPESTATUS[0]}

}

# backup\_oracle() - backup ORACLE VM

# @arg1: VM (FQDN)

# Return: 0: successful

# 1-255: failed

function backup\_oracle() {

log\_msg "DEBUG: ${FUNCNAME[0]} ${\*}"

local VM=${1%%.\*}

rootssh ${VM} '/bin/bash -x' <<-'SSH' 2>&1 | tee -a ${LOG} &> /dev/null

/bin/mkdir -p -m 0755 /lfs/oracle/tmp/ || exit 1

/bin/tar -cvf /lfs/oracle/tmp/$(uname -n)\_oracle-backup.tar \

/etc/fstab /etc/oraInst.loc /etc/oratab \

/usr/openv/netbackup/{bp.conf,{ex,in}clude\_list\*}

/bin/sync

SSH

return ${PIPESTATUS[0]}

}

# backup\_sap() - backup SAP VM

# @arg1: VM (FQDN)

# Return: 0: successful

# 1-255: failed

function backup\_sap() {

log\_msg "DEBUG: ${FUNCNAME[0]} ${\*}"

local VM=${1%%.\*}

rootssh ${VM} '/bin/bash' <<-'SSH' 2>&1 | tee -a ${LOG} &> /dev/null

DEST="$( awk '/\/global\/.\*\/apps/ {print $2}' /proc/mounts )"

[[ -z ${DEST} ]] && exit 1

/bin/mkdir -p -m 0755 ${DEST}/migration || exit 1

/bin/tar -cvf ${DEST}/migration/$(uname -n)\_sap-backup.tar \

/var/lib/{hdb,sdb} /etc/{fstab,oraInst.loc,oratab} \

/usr/sap/{hostctrl,sapservices,xandria} /lfs/{AMS,home/oracle} \

/usr/openv/netbackup/{bp.conf,{in,ex}clude\_list\*}

/bin/sync

SSH

return ${PIPESTATUS[0]}

}

# recreate\_vm - delete SLES11 and creates SLES12 VM

# @arg1: Farmserver

# @arg2: VM

# @arg3: "<OS Disk 1> <OS Disk 2>"

function recreate\_vm() {

log\_msg "DEBUG: ${FUNCNAME[0]} ${\*}"

local FARM="${1%%.\*}"

local VM="${2%%.\*}"

local DISKS="${3}"

local IMAGE="${4}"

cmdb ${VM%%.\*} |egrep "^OS|status|modul" >> ${LOG}

rootssh ${FARM} '/bin/bash -x' <<-SSH 2>&1 | tee -a ${LOG} &> /dev/null

CFG=( \$( /lfs/xen/sbin/vmc info ${VM} | awk -F: '/^(Cur.|Max.) (VCPUs|Mem)/ {printf "%d ", \$2}' ) ); \

echo "DEBUG: CFG: \${CFG[@]}"; \

if [[ "${STANDALONE}" != "yes" ]]; then

echo "DEBUG: DEFAULT"; \

/lfs/xen/sbin/vmc stop ${VM} || exit \${?}; \

/lfs/xen/sbin/vmc delete -f ${VM} || exit \${?}; \

sleep 15; \

/lfs/xen/sbin/vmc create -w -d "${DISKS}" -I ${IMAGE} --cpu=\${CFG[0]}/\${CFG[1]} --mem=\${CFG[2]}M/\${CFG[3]}M ${VM} || exit \${?}; \

sleep 45; \

/lfs/xen/sbin/vmc attdisk ${VM} || exit \${?}; \

else

echo "DEBUG: STANDALONE"; \

/lfs/xen/sbin/vmc -e FARMSERVERS=${FARM} stop ${VM} || exit \${?}; \

/lfs/xen/sbin/vmc -e FARMSERVERS=${FARM} delete -f ${VM} || exit \${?}; \

sleep 15; \

/lfs/xen/sbin/vmc -e FARMSERVERS=${FARM} create -w -d "${DISKS}" -I ${IMAGE} --cpu=\${CFG[0]}/\${CFG[1]} --mem=\${CFG[2]}M/\${CFG[3]}M ${VM} || exit \${?}; \

sleep 45; \

/lfs/xen/sbin/vmc -e FARMSERVERS=${FARM} attdisk ${VM} || exit \${?}; \

fi; \

sleep 15

SSH

return ${PIPESTATUS[0]}

}

# restore\_oracle\_parts - mount ORACLE filesystems and restore 'oraInst.loc' and 'oratab'

# @arg1: VM (FQDN)

# Return: 0: successful

# 1-255: failed

function restore\_oracle\_parts() {

log\_msg "DEBUG: ${FUNCNAME[0]} ${\*}"

local VM=${1}

rootssh ${VM} '/bin/bash -x' <<-'SSH' 2>&1 | tee -a ${LOG} &> /dev/null

mkdir -p /lfs/oracle /lfs/oracle\_audit

echo -e '/dev/dgoracle/lvoracle\t/lfs/oracle\txfs\tdefaults,nofail,x-systemd.device-timeout=20\t0 0' >> /etc/fstab

echo -e '/dev/dgoracle/lvoracleaudit\t/lfs/oracle\_audit\txfs\tdefaults,nofail,x-systemd.device-timeout=20\t0 0' >> /etc/fstab

mount /lfs/oracle

if ! grep -wq '/lfs/oracle' /proc/mounts; then

echo "ERROR: Mount of '/lfs/oracle' failed." >&2; exit 1

fi

mount /lfs/oracle\_audit

if ! grep -wq '/lfs/oracle\_audit' /proc/mounts; then

echo "ERROR: Mount of '/lfs/oracle\_audit' failed." >&2; exit 1

fi

tar -xf /lfs/oracle/tmp/$(uname -n)\_oracle-backup.tar --exclude='etc/fstab' -C /

tar -xf /lfs/oracle/tmp/$(uname -n)\_oracle-backup.tar -C /var/tmp/ etc/fstab

awk '!/ext(3|2)|swap|proc|(sys|sub|debug)fs|devpts|\/lfs\/oracle(|\_audit)|ROOT|^\s\*$|^\s\*#/ \

{print $1"\t"$2"\t"$3"\t"$4",nofail,x-systemd.device-timeout=20\t"$5,$6}' /var/tmp/etc/fstab >> /etc/fstab

awk '!/ext(3|2)|swap|proc|(sys|sub|debug)fs|devpts|\/lfs\/oracle(|\_audit)|ROOT|^\s\*$|^\s\*#/ \

{system("mkdir -p " $2)}' /var/tmp/etc/fstab

if ! mount -a; then

echo "WARNING: Mount of ORACLE filesystems failed, please check" >&2

# exit 1

fi

SSH

[[ ${PIPESTATUS[0]} -eq 0 ]] || exit ${?}

return ${?}

}

# restore\_sap\_parts - mount SAP filesystems and restore data

# @arg1: VM (FQDN)

# @arg2: PKG-Name

# @arg3: PKG-Filesystem

# @arg4: KWOM-LRSQ (0=not installed; 1=installed)

# @arg5: MQSeries Client (0=not installed; 1=installed)

# @arg6: MQSeries Server (0=not installed; 1=installed)

# @arg7: UCMD (0=not installed; 1=installed)

# Return: 0: successful

# 1-255: failed

function restore\_sap\_parts() {

log\_msg "DEBUG: ${FUNCNAME[0]} ${\*}"

local VM=${1}

local PKG=${2}

local PKGFS=${3}

local LRSQ=${4}

local MQSC=${5}

local MQSS=${6}

local UCMD=${7}

# install middleware

if [[ ${LRSQ} -eq 1 ]]; then

rudderctl properties-set --key=tag\_APP-KWOM-lrsq-client\_installed \

--value=true --hostname=${VM} || return ${?}

fi

if [[ ${MQSC} -eq 1 ]]; then

rudderctl properties-set --key=tag\_APP-MQS-Client-8\_installed \

--value=true --hostname=${VM} || return ${?}

fi

if [[ ${MQSS} -eq 1 ]]; then

rudderctl properties-set --key=tag\_APP-MQS-Server-8\_installed \

--value=true --hostname=${VM} || return ${?}

fi

if [[ ${UCMD} -eq 1 ]]; then

rudderctl properties-set --key=tag\_SW-Interface-UCMD\_installed \

--value=true --hostname=${VM} || return ${?}

fi

rootssh ${VM} '/bin/bash' <<-SSH 2>&1 | tee -a ${LOG} &> /dev/null

[[ -z ${PKG} ]] && exit 1; \

[[ -z ${PKGFS} ]] && exit 1; \

mkdir -p /global/${PKG}/apps; \

echo -e "${PKGFS}\t/global/${PKG}/apps\txfs\tdefaults,nofail,x-systemd.device-timeout=20\t0 0" >> /etc/fstab; \

mount /global/${PKG}/apps; \

if ! grep -wq "/global/${PKG}/apps" /proc/mounts; then \

echo "ERROR: Mount of \'/global/${PKG}/apps\' failed." >&2; exit 1; \

fi; \

tar -xvf /global/${PKG}/apps/migration/$(uname -n)\_sap-backup.tar \

--exclude='etc/fstab' -C / ; \

tar -xvf /global/${PKG}/apps/migration/$(uname -n)\_sap-backup.tar \

-C /var/tmp/ etc/fstab ; \

mkdir -p /lfs/oracle; \

awk '!/ext(3|2)|swap|proc|(sys|sub|debug)fs|devpts|\/global\/.\*\/apps|ROOT|^\s\*$|^\s\*#/ \

{print \$1"\t"\$2"\t"\$3"\t"\$4",nofail,x-systemd.device-timeout=20\t"\$5,\$6}' /var/tmp/etc/fstab >> /etc/fstab; \

awk '!/ext(3|2)|swap|proc|(sys|sub|debug)fs|devpts|\/global\/.\*\/apps|ROOT|^\s\*$|^\s\*#/ \

{system("mkdir -p " \$2)}' /var/tmp/etc/fstab; \

if ! mount -a; then \

echo "WARNING: Mount of SAP filesystems failed, please check" >&2; \

# exit 1; \

fi

SSH

[[ ${PIPESTATUS[0]} -eq 0 ]] || exit ${?}

return ${?}

}

# postupdate\_vm - update Rudder, DCODB and Nagios

# @arg1: VM (FQDN)

# Return: 0: successful

# 1-255: failed

function postupdate\_vm() {

log\_msg "DEBUG: ${FUNCNAME[0]} ${\*}"

local VM=${1}

rootssh ${VM} '/bin/bash -x' <<-'SSH' 2>&1 | tee -a ${LOG} &> /dev/null

/usr/bin/rudder agent update -f

/usr/bin/rudder agent inventory

## sleep 120 seconds to process the inventory

/bin/sleep 120

/usr/bin/rudder agent run; /usr/bin/rudder agent run; /usr/bin/rudder agent run

## sleep 60 seconds after rudder agent runs

/bin/sleep 60

SSH

[[ ${PIPESTATUS[0]} -eq 0 ]] || return ${?}

## check sudoers/rudder

if [[ "${CHECK}" == "no" ]]; then

std\_msg "Skipping client checks."

else

log\_msg "Check if user 'qqdeploy' is removed"

rootssh ${VM} '/bin/bash' <<-'SSH' 2>&1 | tee -a ${LOG} &> /dev/null

if /usr/bin/sudo -l -U qqdeploy; then

exit 1

else

exit 0

fi

SSH

if [[ ${PIPESTATUS[0]} -ne 0 ]]; then

std\_msg "WARNING: Please check the sudoers/rudder."

std\_msg "WARNING: Mostly you have just to wait for 30 min, if \`rudder agent update' and"

std\_msg "WARNING: \`rudder agent run' still provide some error please contact L2."

fi

fi

## update Nagios

if [[ "${NAGIOS}" == "no" ]]; then

std\_msg "Skipping Nagios config rebuild"

else

update\_nagios\_client -s ${1} -u 2>&1 | tee -a ${LOG} &> /dev/null

if [[ ${PIPESTATUS[0]} -ne 0 ]]; then

std\_msg "WARNING: Could not update Nagios, please check."

std\_msg "WARNING: Check if multiple \`update\_nagios\_client' are running and rerun it later."

fi

fi

cmdb ${VM%%.\*} |egrep "^OS|status|modul" >> ${LOG}

update\_dcodb\_status.py --remove --server ${VM} 2>&1 | tee -a ${LOG} &> /dev/null

cmdb ${VM%%.\*} |egrep "^OS|status|modul" >> ${LOG}

[[ ${PIPESTATUS[0]} -eq 0 ]] || { err\_msg "Could not remove DCODB flag, please check."; return 1; }

return ${?}

}

###

### MAIN

###

function main() {

[[ ${#} -eq 0 ]] && { usage; exit 0; }

ARGS="${\*}"

while [[ ${#} -gt 0 ]]; do

case ${1} in

-h|--help)

usage; exit ${?}

;;

-s|--srv|--server)

[[ -z ${2} ]] && { echo "Missing server use \`${0##\*/} --help' for further information" >&2; exit 1; }

SERVER=${2}

shift 2

;;

-e|--env|--environment)

[[ -z ${2} ]] && { echo "Missing environment use \`${0##\*/} --help' for further information" >&2; exit 1; }

ENVIRONMENT=${2}

shift 2

;;

-d|--disk)

[[ -z ${2} ]] && { echo "Missing disk use \`${0##\*/} --help' for further information" >&2; exit 1; }

DISK1="${2%% \*}"

DISK2="${2##\* }"

[[ -z ${DISK1} || -z ${DISK2} ]] && { echo "Missing disk use \`${0##\*/} --help' for further information" >&2; exit 1; }

shift 2

;;

--reinstall|reinstall)

[[ -z ${MODE} ]] || { echo "Only one mode is allowed use \`${0##\*/} --help' for further information" >&2; exit 1; }

MODE='reinstall'

shift 1

;;

-I|--image)

case ${2} in

sles12sp2) IMAGE=sles12sp2 ;;

sles12sp4) IMAGE=sles12sp4 ;;

\*) { echo "Unknown image use \`${0##\*/} --help' for further information" >&2; exit 1; } ;;

esac

shift 2

;;

--check|check)

[[ -z ${MODE} ]] || { echo "Only one mode is allowed use \`${0##\*/} --help' for further information" >&2; exit 1; }

MODE='check'

shift 1

;;

--preparation|preparation)

[[ -z ${MODE} ]] || { echo "Only one mode is allowed use \`${0##\*/} --help' for further information" >&2; exit 1; }

MODE='preparation'

shift 1

;;

--nobackup)

local BACKUP="no"

shift 1

;;

--nocheck)

local CHECK="no"

shift 1

;;

--nonagios)

local NAGIOS="no"

shift 1

;;

--standalone)

local STANDALONE="yes"

shift 1

;;

\*)

echo "Unknown option use \`${0##\*/} --help' for further information" >&2; exit 1

;;

esac

done

[[ $EUID =~ ^(0|720021)$ ]] || { echo "Please run the script as root or in instserv env" >&2; exit 1; }

[[ -z ${SERVER} ]] && { echo "Missing server use \`${0##\*/} --help' for further information" >&2; exit 1; }

FQDN="$( host ${SERVER} | awk '/has address/{print $1}')"

[[ "${SERVER}" != "${FQDN}" ]] && { echo "Please use the server FQDN" >&2; exit 1; }

[[ -z ${DISK1} || -z ${DISK2} ]] && { echo "Missing disks use \`${0##\*/} --help' for further information" >&2; exit 1; }

[[ -z ${MODE} ]] && { echo "Missing mode use \`${0##\*/} --help' for further information" >&2; exit 1; }

# [[ -z ${ENVIRONMENT} ]] && local ENVIRONMENT="$( get\_environment ${SERVER} )"

[[ -z ${BACKUP} ]] && BACKUP='yes'

[[ -z ${STANDALONE} ]] && STANDALONE='no'

LOGDIR="/global/instserv/logs/vm\_upgrade"

if ! [[ -d ${LOGDIR} ]]; then

echo "Create missing logdir '${LOGDIR}'"

mkdir -p ${LOGDIR} || { echo "Cannot create missing logdir '${LOGDIR}'" >&2; exit 1; }

fi

LOG="${LOGDIR}/${SERVER}\_$( date '+%Y%m%d-%H%m%S' ).log"

log\_msg "DEBUG: User ${SUDO\_USER:-${USER}} executed: ${0##\*/} ${ARGS}"

trap cleanup EXIT

trap "echo; exit 130" SIGINT

std\_msg "For detailed progress and/or errors, please check the log file."

std\_msg "Log file: ${LOG}"

echo

# check module

MODULE=$( cmdb ${SERVER%%.\*} | awk '/^modul:/ {print $2}' )

case ${MODULE} in

ORACLE) ;;

SAP)

if [[ "${MODE}" == "preparation" ]]; then

err\_msg "Preparation is only in module ORACLE supported"

return 1

fi

;;

\*)

err\_msg "Wrong module for ${SERVER}, only module ORACLE and SAP VMs are supported."; return 1

;;

esac

case ${MODE} in

check|reinstall|preparation)

std\_msg "\* Checking CMDB data and SAN array..."

if [[ "${CHECK}" == "no" ]]; then

std\_msg "Skipping CMDB and SAN checks"

else

check\_logic ${SERVER} ${DISK1} ${DISK2} || exit ${?}

fi

FARMSERVERS=( $( cmdb ${SERVER%%.\*} | awk -F: '/^farmservers:/&&gsub(/,/,"") {print $2}') )

if [[ -z ${FARMSERVERS[0]} || -z ${FARMSERVERS[1]} ]]; then

err\_msg "Could not fetch farmserver for ${SERVER}"; return 1

fi

FARMSERVER=$( rootssh ${SERVER} "/bin/xenstore-read /tool/hostname" )

if [[ -z ${FARMSERVER} || ! ${FARMSERVERS[@]} =~ ${FARMSERVER} ]]; then

err\_msg "Current farmserver doesn't match the CMDB farmservers"; return 1

fi

# Check Steps

case ${MODULE} in

ORACLE)

std\_msg "\* Checking if ORACLE is running and the filesystems are ok..."

if [[ "${CHECK}" == "no" ]]; then

std\_msg "Skipping ORACLE checks"

else

check\_oracle\_vm ${SERVER} || exit ${?}

fi

;;

SAP)

std\_msg "\* Checking if SAP is running..."

if [[ "${CHECK}" == "no" ]]; then

std\_msg "Skipping SAP checks"

else

check\_sap\_vm ${SERVER} || exit ${?}

fi

;;

esac

if [[ "${STANDALONE}" == "yes" ]]; then

std\_msg "\* Checking standalone Farmserver (${FARMSERVER}) for SAN disks..."

check\_farmserver ${FARMSERVER} ${SERVER} ${DISK1} ${DISK2} || exit ${?}

else

std\_msg "\* Checking Farmserver (${FARMSERVERS[0]}) for SAN disks..."

check\_farmserver ${FARMSERVERS[0]} ${SERVER} ${DISK1} ${DISK2} || exit ${?}

std\_msg "\* Checking Farmserver (${FARMSERVERS[1]}) for SAN disks..."

check\_farmserver ${FARMSERVERS[1]} ${SERVER} ${DISK1} ${DISK2} || exit ${?}

fi

;;

esac

case ${MODE} in

reinstall)

# Prepare Steps

std\_msg "\* Preprare VM..."

if [[ ${ENVIRONMENT:-missing} == "missing" ]]; then

prepare\_vm ${SERVER} || exit ${?}

else

prepare\_vm ${SERVER} ${ENVIRONMENT} || exit ${?}

fi

if [[ "${STANDALONE}" == "yes" ]]; then

std\_msg "\* Prepare standalone Farmserver (${FARMSERVER})..."

prepare\_farmserver ${FARMSERVER} || exit ${?}

else

std\_msg "\* Preprare Farmserver (${FARMSERVERS[0]})..."

prepare\_farmserver ${FARMSERVERS[0]} || exit ${?}

std\_msg "\* Preprare Farmserver (${FARMSERVERS[1]})..."

prepare\_farmserver ${FARMSERVERS[1]} || exit ${?}

fi

# Update Steps

# set\_vm\_downtime ${SERVER}

std\_msg "\* Backup VM..."

backup\_vm ${FARMSERVER} ${SERVER} ${BACKUP} || exit ${?}

case ${MODULE} in

ORACLE)

std\_msg "\* Backup ORACLE parts..."

backup\_oracle ${SERVER}

if [[ ${?} -ne 0 ]]; then

err\_msg "Could not backup ORACLE files to '/lfs/oracle/tmp'"

exit 1

fi

;;

SAP)

SAP\_INFO=( $( prepare\_sap\_vm ${SERVER} ) )

std\_msg "\* Backup SAP parts..."

backup\_sap ${SERVER}

if [[ ${?} -ne 0 ]]; then

err\_msg "Could not backup SAP files to '/global/<PKG-Name>/apps'"

exit 1

fi

;;

esac

std\_msg "\* Create new ${IMAGE:-sles12sp2} VM..."

recreate\_vm ${FARMSERVER} ${SERVER} "${DISK1} ${DISK2}" ${IMAGE:-sles12sp2} || exit ${?}

case ${MODULE} in

ORACLE)

std\_msg "\* Restore ORACLE data..."

restore\_oracle\_parts ${SERVER} || exit ${?}

;;

SAP)

std\_msg "\* Restore SAP data..."

restore\_sap\_parts ${SERVER} ${SAP\_INFO[@]} || exit ${?}

;;

esac

std\_msg "\* Update Rudder, DCODB and Nagios..."

postupdate\_vm ${SERVER} || exit ${?}

# rm\_vm\_downtime ${SERVER}

;;

esac

exit 0

}

main "${@}"

[INSTSERV][qxz10kv][lpinstiaas02] /global/instserv/bin $

[INSTSERV][qxz10kv][lpinstiaas02] /global/instserv/bin $

[INSTSERV][qxz10kv][lpinstiaas02] /global/instserv/bin $ cat configure\_veritas.sh

#!/bin/bash

if [ $# -eq 0 ]; then

echo "Syntax: $0 <templatefile> [<templatefile> ...]"

exit 1

fi

######################################

# Some variables we'll make use of

######################################

TEMPLATES=/global/instserv/data/CONFIGURE\_VERITAS/vcs

SETDT=/lfs/scripts/set\_downtime

MYSQL=$( which mysql 2>&1 ) || { echo "mysql client not found!"; exit 1; }

source /global/instserv/bin/storage.lib.sh || { echo "could not source storage.lib.sh"; exit 1; }

source "/global/instserv/bin/ssh.cred.lib.sh"

ssh.cred.admin

source "/global/instserv/bin/ssh.multiplex.lib.sh"

##############################################

# Start logging to automatically rotated file

##############################################

LOGFILE=/global/instserv/logs/control\_veritas/$( date "+%Y/%m/%Y-%m-%d\_%H-%M-%S\_$$.log" )

mkdir -p "$( dirname "${LOGFILE}" )" || { echo "could not create dir of logfile '$LOGFILE'"; exit 1; }

touch "${LOGFILE}" || { echo "could not touch logfile '$LOGFILE'"; exit 1; }

exec > >( tee >( strings | gawk '{print strftime("[%Y.%m.%d. %H:%M:%S] ") $0; fflush();}' >> "${LOGFILE}" ) ) 2>&1

############################################

# Make use of some fancy colours

############################################

declare -rx BOLD="$(tput bold)"

declare -rx RED="$(tput bold; tput setaf 1)"

declare -rx GREEN="$(tput bold; tput setaf 2)"

declare -rx YELLOW="$(tput bold; tput setaf 3)"

declare -rx RESET="$(tput sgr0 )"

declare -rx UL="$( tput smul )"

format="%s%-6s%s %s\n"; nl="$(printf "\n")"

function h() { printf "\n\n%s %-80s%s\n\n" "${BOLD}${UL}" "$@" "$RESET" ; }

function p() { printf "%s %-45s " "->" "$@"; }

function e() { printf "%s%-6s%s\n %s\n" "$RED" "[ERROR]" "$RESET" " ${BOLD} \*\*\* $1 ${RESET}"; [[ -n "$2" ]] && echo -e "$2"; exit 1; }

function w() { printf "$format" "$YELLOW" "[WARN]" "$RESET" "${@:+ $@}"; }

function ok() { printf "$format" "$GREEN" "[OK]" "$RESET" "${@:+ $@}"; }

###########################################

# Abstraction of remote access commands

###########################################

function host.ssh() { local chost="$1"; shift; m\_ssh ${chost} "$@" 2>&1; return $?; }

function host.scp() { m\_scp "$@" 2>&1; return $?; }

function host.rsync() { m\_rsync -a "$@" 2>&1; return $?; }

############################################

# Querying Storm-DB routines

############################################

if nmap -q -p "$gs2\_mysql\_port" -oG - "$gs2\_mysql\_server" | grep -q 'filtered/tcp'; then

GS\_REACHABLE=0

function query\_gs() {

# No query possible, ports not open!

return 255;

}

else

GS\_REACHABLE=1

function query\_gs() {

local RC=0;

echo "# @$1"

$MYSQL --skip-column-names --batch -h $gs2\_mysql\_server -P $gs2\_mysql\_port $gs2\_mysql\_db -u $gs2\_mysql\_user -p${gs2\_mysql\_pass} <<-%SQL

SELECT DISTINCT

SUBSTRING(lun FROM 4), arrayserial, storageclass

FROM

v\_storageoverview

WHERE

orderposid LIKE '%$1'

ORDER BY

arrayserial ASC,

lun DESC;

%SQL

return $?

}

fi

#################################

# Cleanup routing for errors

#################################

function cleanup() {

echo "${RESET}"

echo "cleanup..."

[[ -d "$TMPDIR" ]] && rm -rf "$TMPDIR"

[[ -f "${LOGFILE}" ]] && bzip2 -9 "${LOGFILE}"

}

trap cleanup INT QUIT EXIT

# Create temp directory

TMPDIR=$(mktemp -d) || exit 1

chmod 755 "${TMPDIR}"

# check out this cool site for headers :)

# at: http://patorjk.com/software/taag/#p=display&f=Cybermedium&t=Veritas%20Manager%0A

cat <<-%HEADER

${BOLD}

\_ \_ \_\_\_\_ \_\_\_\_ \_ \_\_\_ \_\_\_\_ \_\_\_\_ \_ \_ \_\_\_\_ \_ \_ \_\_\_\_ \_\_\_\_ \_\_\_\_ \_\_\_\_

| | |\_\_\_ |\_\_/ | | |\_\_| [\_\_ |\/| |\_\_| |\ | |\_\_| | \_\_ |\_\_\_ |\_\_/

\/ |\_\_\_ | \ | | | | \_\_\_] | | | | | \| | | |\_\_] |\_\_\_ | \

${RESET}

%HEADER

#################################

# Main loop through the files

#################################

for file in "$@"; do

rm -rf "${TMPDIR}"/\*

h "Processing file '$file'"

p "Checking File validity"

template=$( readlink -f "$file" )

[[ -s "$template" ]] || e "File '$template' does not exist"

filename=$(basename "$template" )

ok "file exist: $filename"

p "Looking for RequestID markers"

RIDS="$( gawk '/^@/ { gsub("@",""); print $0; }' "$template" )" || e "Could not parse RIDs" "$RIDS"

if ((GS\_REACHABLE==0)); then

w "no StormDB connection: ports closed"

else

if [[ -n "$RIDS" ]]; then

ok "found"

p "Starting RID lookup in StormDB"

cp -p "$template" "${TMPDIR}/${filename}"

cp -p "$template" "${template}.orig"

for RID in $RIDS; do

DATA="$( query\_gs $RID 2>&1 )"; RC=$?;

case $RC in

255)

w "no StormDB connection: ports closed";

break;

;;

0) : ;;

\*)

e "Error during mysql query for StormDB" "$DATA"; ;;

esac

echo "$DATA" | gawk -v "RID=^@$RID" '

BEGIN { line=1; }

FILENAME == "-" { a[line++] = $0; next; }

FILENAME != "-" && $0 ~ RID { for (i=1; i<line; i++) { print a[i]; }; next; }

FILENAME != "-" { print }

' - "${TMPDIR}/${filename}" > "${TMPDIR}/${filename}.new"

mv "${TMPDIR}/${filename}.new" "${TMPDIR}/${filename}"

done

if cmp -s "$template" "${TMPDIR}/${filename}"; then

w "no RIDs replaced"

else

mv "${TMPDIR}/${filename}" "$template"

ok "template modified"

fi

else

ok "none found"

fi

fi

p "Fetching Hostlist(s)"

hostlist=( $( gawk '

/^HOST/ { gsub("HOST ",""); print; exit; }

/^node/ { gsub("node ",""); print; exit; }

' $template ) )

[[ -z "$hostlist" ]] && e "No target hosts specified with 'HOST' or 'node' lines"

ok "working on ${hostlist[\*]}"

p "Testing Hosts"

for chost in ${hostlist[\*]}; do

m\_start $chost

WHOAMI=$( host.ssh $chost "whoami" 2>&1 ) || e "Host test failed on '$chost'" "$WHOAMI"

[[ x"$WHOAMI" != x"root" ]] && e "Remote user on '$chost' is not root but '$WHOAMI'"

done

ok "root access verified"

p "Fetching ServiceGroup"

package="$(gawk '/^sg / {print $2}' $template )"

[[ -z "$package" ]] && w "no 'sg'-line found" || ok "found $package"

p "Fetching module for Apps"

appmodules="$(gawk '

/^app / {

module="other";

printf "%s ",$2;

for (i=3; i<=NF; ++i) {

if ($i ~ /module=/) {

gsub("module=","",$i);

module=$i;

}

}

printf "%s\n", module;

}

' $template )"

[[ -z "$appmodules" ]] && w "no apps in template" || ok "fetched"

h "Running remote sanity check"

for chost in ${hostlist[\*]}; do

p "Copy template to remote host '$chost'"

SCP=$( host.scp "$template" ${chost}:"/tmp/${filename}" 2>&1 ) || e "Could not copy files:" "$SCP"

ok "copied"

p "Running sanity check on '$chost'"; echo "";

echo $BOLD

echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

host.ssh ${chost} "/sbin/vxctl /tmp/${filename} check" 2>&1; RC=$?

echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

echo $RESET

p "Sanity check"

(( RC == 0 )) && ok "success" || e "Failed, see for error above"

if [[ -x $SETDT ]]; then

p "Setting Downtime for SAN Mirror Service"

if ! DT=$( $SETDT ${chost} -s os\_linux\_fs\_check\_san\_mirror -m 15 2>&1 ); then

if [[ $DT =~ "ERROR: Input Service"\* ]]; then

w "no such check"

else

w "Error setting Downtime" "$DT"

fi

else

ok "set"

fi

p "Setting Downtime for VXVM Service"

if ! DT=$( $SETDT ${chost} -s os\_linux\_fs\_check\_vxvm -m 15 2>&1 ); then

if [[ $DT =~ "ERROR: Input Service"\* ]]; then

w "no such check"

else

w "Error setting Downtime" "$DT"

fi

else

ok "set"

fi

fi

done

if ! [[ -z "$appmodules" ]]; then

h "Deploying application scripts"

p "Preparing files"

E=0

while read APP MODULE; do

if ! [[ -d "${TEMPLATES}/${MODULE}" ]]; then

E=1;

continue;

fi

for scriptname in 'start' 'stop' 'monitor'; do

SCRIPT="${TEMPLATES}/${MODULE}/${scriptname}"

if ! [[ -e "${SCRIPT}" ]]; then

E=1;

continue;

fi

sed "s/%PACKAGENAME%/$servicegroup/g" "${SCRIPT}" > "${TMPDIR}/${package}\_${APP}\_${scriptname}"

done

done <<<"$appmodules"

chmod a+rx "$TMPDIR"/\*

(( E == 0 )) && ok "staged" || w "some files missing"

p "Distributing files"

for chost in ${hostlist[\*]}; do

MKDIR=$( host.ssh $chost "mkdir -p /lfs/cluster/vcs/" 2>&1 ) || \

e "Failed to create remote /lfs/cluster/vcs/ folder" "$MKDIR"

RSYNC=$( host.rsync --ignore-existing "$TMPDIR/" ${chost}:/lfs/cluster/vcs/ 2>&1 ) || \

e "Sync failed to '$chost'" "$RSYNC"

done

ok "distributed"

fi

h "Starting remote /sbin/vxctl call"

p "Initiating vxctl on ${hostlist}"; echo "";

echo "$BOLD"

echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

host.ssh ${hostlist} "/sbin/vxctl /tmp/$filename" 2>&1; RC=$?;

echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

echo $RESET

p "Evaluating remote script return code"

case $RC in

255) ok "modification ok"; continue ;;

0) ok "creation ok" ;;

\*) e "Failed, see for error above" ;;

esac

## Disabled automatic failover test

# if [[ -n "$package" ]]; then

# h "Starting failover tests"

# p "Starting '$package' on '$hostlist'"

# STARTPKG="$( host.ssh $hostlist "hagrp -online ${package} -sys $hostlist" 2>&1 )" || e "failed" "$STARTPKG"

# ok "started"

#

# p "Waiting for Package to be online..."

# WAITPKG="$( host.ssh ${hostlist} "hagrp -wait ${package} State ONLINE -sys ${hostlist} -time 300 2>&1" )" e "failed" "$WAITPKG"

# ok "online"

#

# ## TODO: Do not do any failover test on dependency of parent pkg

# p "Start failover '$package' to '${hostlist[1]}'"

# FAILOVER="$( host.ssh ${hostlist} "hagrp -switch $package -to ${hostlist[1]} 2>&1" )" || e "failed" "$FAILOVER"

# ok "started"

#

# p "Waiting for failover to complete..."

# WAIT="$( host.ssh ${hostlist} "hagrp -wait $package State ONLINE -sys ${hostlist[1]} -time 300" 2>&1 )" || e "error" "$WAIT"

# ok "success"

#

# p "Starting failback '$package to '${hostlist}'"

# FAILOVER="$( host.ssh ${hostlist} "hagrp -switch $package -to ${hostlist}" 2>&1 )" || e "failed" "$FAILOVER"

# ok "started"

#

# p "Waiting for failover to complete..."

# WAIT="$( host.ssh ${hostlist} "hagrp -wait $package State ONLINE -sys ${hostlist} -time 300" 2>&1 )" || e "error" "$WAIT"

# ok "success"

# fi

done

exit 0

[INSTSERV][qxz10kv][lpinstiaas02] /global/instserv/bin $

[INSTSERV][qxz10kv][lpinstiaas02] /global/instserv/bin $ cat configure\_veritas.sh

#!/bin/bash

if [ $# -eq 0 ]; then

echo "Syntax: $0 <templatefile> [<templatefile> ...]"

exit 1

fi

######################################

# Some variables we'll make use of

######################################

TEMPLATES=/global/instserv/data/CONFIGURE\_VERITAS/vcs

SETDT=/lfs/scripts/set\_downtime

MYSQL=$( which mysql 2>&1 ) || { echo "mysql client not found!"; exit 1; }

source /global/instserv/bin/storage.lib.sh || { echo "could not source storage.lib.sh"; exit 1; }

source "/global/instserv/bin/ssh.cred.lib.sh"

ssh.cred.admin

source "/global/instserv/bin/ssh.multiplex.lib.sh"

##############################################

# Start logging to automatically rotated file

##############################################

LOGFILE=/global/instserv/logs/control\_veritas/$( date "+%Y/%m/%Y-%m-%d\_%H-%M-%S\_$$.log" )

mkdir -p "$( dirname "${LOGFILE}" )" || { echo "could not create dir of logfile '$LOGFILE'"; exit 1; }

touch "${LOGFILE}" || { echo "could not touch logfile '$LOGFILE'"; exit 1; }

exec > >( tee >( strings | gawk '{print strftime("[%Y.%m.%d. %H:%M:%S] ") $0; fflush();}' >> "${LOGFILE}" ) ) 2>&1

############################################

# Make use of some fancy colours

############################################

declare -rx BOLD="$(tput bold)"

declare -rx RED="$(tput bold; tput setaf 1)"

declare -rx GREEN="$(tput bold; tput setaf 2)"

declare -rx YELLOW="$(tput bold; tput setaf 3)"

declare -rx RESET="$(tput sgr0 )"

declare -rx UL="$( tput smul )"

format="%s%-6s%s %s\n"; nl="$(printf "\n")"

function h() { printf "\n\n%s %-80s%s\n\n" "${BOLD}${UL}" "$@" "$RESET" ; }

function p() { printf "%s %-45s " "->" "$@"; }

function e() { printf "%s%-6s%s\n %s\n" "$RED" "[ERROR]" "$RESET" " ${BOLD} \*\*\* $1 ${RESET}"; [[ -n "$2" ]] && echo -e "$2"; exit 1; }

function w() { printf "$format" "$YELLOW" "[WARN]" "$RESET" "${@:+ $@}"; }

function ok() { printf "$format" "$GREEN" "[OK]" "$RESET" "${@:+ $@}"; }

###########################################

# Abstraction of remote access commands

###########################################

function host.ssh() { local chost="$1"; shift; m\_ssh ${chost} "$@" 2>&1; return $?; }

function host.scp() { m\_scp "$@" 2>&1; return $?; }

function host.rsync() { m\_rsync -a "$@" 2>&1; return $?; }

############################################

# Querying Storm-DB routines

############################################

if nmap -q -p "$gs2\_mysql\_port" -oG - "$gs2\_mysql\_server" | grep -q 'filtered/tcp'; then

GS\_REACHABLE=0

function query\_gs() {

# No query possible, ports not open!

return 255;

}

else

GS\_REACHABLE=1

function query\_gs() {

local RC=0;

echo "# @$1"

$MYSQL --skip-column-names --batch -h $gs2\_mysql\_server -P $gs2\_mysql\_port $gs2\_mysql\_db -u $gs2\_mysql\_user -p${gs2\_mysql\_pass} <<-%SQL

SELECT DISTINCT

SUBSTRING(lun FROM 4), arrayserial, storageclass

FROM

v\_storageoverview

WHERE

orderposid LIKE '%$1'

ORDER BY

arrayserial ASC,

lun DESC;

%SQL

return $?

}

fi

#################################

# Cleanup routing for errors

#################################

function cleanup() {

echo "${RESET}"

echo "cleanup..."

[[ -d "$TMPDIR" ]] && rm -rf "$TMPDIR"

[[ -f "${LOGFILE}" ]] && bzip2 -9 "${LOGFILE}"

}

trap cleanup INT QUIT EXIT

# Create temp directory

TMPDIR=$(mktemp -d) || exit 1

chmod 755 "${TMPDIR}"

# check out this cool site for headers :)

# at: http://patorjk.com/software/taag/#p=display&f=Cybermedium&t=Veritas%20Manager%0A

cat <<-%HEADER

${BOLD}

\_ \_ \_\_\_\_ \_\_\_\_ \_ \_\_\_ \_\_\_\_ \_\_\_\_ \_ \_ \_\_\_\_ \_ \_ \_\_\_\_ \_\_\_\_ \_\_\_\_ \_\_\_\_

| | |\_\_\_ |\_\_/ | | |\_\_| [\_\_ |\/| |\_\_| |\ | |\_\_| | \_\_ |\_\_\_ |\_\_/

\/ |\_\_\_ | \ | | | | \_\_\_] | | | | | \| | | |\_\_] |\_\_\_ | \

${RESET}

%HEADER

#################################

# Main loop through the files

#################################

for file in "$@"; do

rm -rf "${TMPDIR}"/\*

h "Processing file '$file'"

p "Checking File validity"

template=$( readlink -f "$file" )

[[ -s "$template" ]] || e "File '$template' does not exist"

filename=$(basename "$template" )

ok "file exist: $filename"

p "Looking for RequestID markers"

RIDS="$( gawk '/^@/ { gsub("@",""); print $0; }' "$template" )" || e "Could not parse RIDs" "$RIDS"

if ((GS\_REACHABLE==0)); then

w "no StormDB connection: ports closed"

else

if [[ -n "$RIDS" ]]; then

ok "found"

p "Starting RID lookup in StormDB"

cp -p "$template" "${TMPDIR}/${filename}"

cp -p "$template" "${template}.orig"

for RID in $RIDS; do

DATA="$( query\_gs $RID 2>&1 )"; RC=$?;

case $RC in

255)

w "no StormDB connection: ports closed";

break;

;;

0) : ;;

\*)

e "Error during mysql query for StormDB" "$DATA"; ;;

esac

echo "$DATA" | gawk -v "RID=^@$RID" '

BEGIN { line=1; }

FILENAME == "-" { a[line++] = $0; next; }

FILENAME != "-" && $0 ~ RID { for (i=1; i<line; i++) { print a[i]; }; next; }

FILENAME != "-" { print }

' - "${TMPDIR}/${filename}" > "${TMPDIR}/${filename}.new"

mv "${TMPDIR}/${filename}.new" "${TMPDIR}/${filename}"

done

if cmp -s "$template" "${TMPDIR}/${filename}"; then

w "no RIDs replaced"

else

mv "${TMPDIR}/${filename}" "$template"

ok "template modified"

fi

else

ok "none found"

fi

fi

p "Fetching Hostlist(s)"

hostlist=( $( gawk '

/^HOST/ { gsub("HOST ",""); print; exit; }

/^node/ { gsub("node ",""); print; exit; }

' $template ) )

[[ -z "$hostlist" ]] && e "No target hosts specified with 'HOST' or 'node' lines"

ok "working on ${hostlist[\*]}"

p "Testing Hosts"

for chost in ${hostlist[\*]}; do

m\_start $chost

WHOAMI=$( host.ssh $chost "whoami" 2>&1 ) || e "Host test failed on '$chost'" "$WHOAMI"

[[ x"$WHOAMI" != x"root" ]] && e "Remote user on '$chost' is not root but '$WHOAMI'"

done

ok "root access verified"

p "Fetching ServiceGroup"

package="$(gawk '/^sg / {print $2}' $template )"

[[ -z "$package" ]] && w "no 'sg'-line found" || ok "found $package"

p "Fetching module for Apps"

appmodules="$(gawk '

/^app / {

module="other";

printf "%s ",$2;

for (i=3; i<=NF; ++i) {

if ($i ~ /module=/) {

gsub("module=","",$i);

module=$i;

}

}

printf "%s\n", module;

}

' $template )"

[[ -z "$appmodules" ]] && w "no apps in template" || ok "fetched"

h "Running remote sanity check"

for chost in ${hostlist[\*]}; do

p "Copy template to remote host '$chost'"

SCP=$( host.scp "$template" ${chost}:"/tmp/${filename}" 2>&1 ) || e "Could not copy files:" "$SCP"

ok "copied"

p "Running sanity check on '$chost'"; echo "";

echo $BOLD

echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

host.ssh ${chost} "/sbin/vxctl /tmp/${filename} check" 2>&1; RC=$?

echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

echo $RESET

p "Sanity check"

(( RC == 0 )) && ok "success" || e "Failed, see for error above"

if [[ -x $SETDT ]]; then

p "Setting Downtime for SAN Mirror Service"

if ! DT=$( $SETDT ${chost} -s os\_linux\_fs\_check\_san\_mirror -m 15 2>&1 ); then

if [[ $DT =~ "ERROR: Input Service"\* ]]; then

w "no such check"

else

w "Error setting Downtime" "$DT"

fi

else

ok "set"

fi

p "Setting Downtime for VXVM Service"

if ! DT=$( $SETDT ${chost} -s os\_linux\_fs\_check\_vxvm -m 15 2>&1 ); then

if [[ $DT =~ "ERROR: Input Service"\* ]]; then

w "no such check"

else

w "Error setting Downtime" "$DT"

fi

else

ok "set"

fi

fi

done

if ! [[ -z "$appmodules" ]]; then

h "Deploying application scripts"

p "Preparing files"

E=0

while read APP MODULE; do

if ! [[ -d "${TEMPLATES}/${MODULE}" ]]; then

E=1;

continue;

fi

for scriptname in 'start' 'stop' 'monitor'; do

SCRIPT="${TEMPLATES}/${MODULE}/${scriptname}"

if ! [[ -e "${SCRIPT}" ]]; then

E=1;

continue;

fi

sed "s/%PACKAGENAME%/$servicegroup/g" "${SCRIPT}" > "${TMPDIR}/${package}\_${APP}\_${scriptname}"

done

done <<<"$appmodules"

chmod a+rx "$TMPDIR"/\*

(( E == 0 )) && ok "staged" || w "some files missing"

p "Distributing files"

for chost in ${hostlist[\*]}; do

MKDIR=$( host.ssh $chost "mkdir -p /lfs/cluster/vcs/" 2>&1 ) || \

e "Failed to create remote /lfs/cluster/vcs/ folder" "$MKDIR"

RSYNC=$( host.rsync --ignore-existing "$TMPDIR/" ${chost}:/lfs/cluster/vcs/ 2>&1 ) || \

e "Sync failed to '$chost'" "$RSYNC"

done

ok "distributed"

fi

h "Starting remote /sbin/vxctl call"

p "Initiating vxctl on ${hostlist}"; echo "";

echo "$BOLD"

echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

host.ssh ${hostlist} "/sbin/vxctl /tmp/$filename" 2>&1; RC=$?;

echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

echo $RESET

p "Evaluating remote script return code"

case $RC in

255) ok "modification ok"; continue ;;

0) ok "creation ok" ;;

\*) e "Failed, see for error above" ;;

esac

## Disabled automatic failover test

# if [[ -n "$package" ]]; then

# h "Starting failover tests"

# p "Starting '$package' on '$hostlist'"

# STARTPKG="$( host.ssh $hostlist "hagrp -online ${package} -sys $hostlist" 2>&1 )" || e "failed" "$STARTPKG"

# ok "started"

#

# p "Waiting for Package to be online..."

# WAITPKG="$( host.ssh ${hostlist} "hagrp -wait ${package} State ONLINE -sys ${hostlist} -time 300 2>&1" )" e "failed" "$WAITPKG"

# ok "online"

#

# ## TODO: Do not do any failover test on dependency of parent pkg

# p "Start failover '$package' to '${hostlist[1]}'"

# FAILOVER="$( host.ssh ${hostlist} "hagrp -switch $package -to ${hostlist[1]} 2>&1" )" || e "failed" "$FAILOVER"

# ok "started"

#

# p "Waiting for failover to complete..."

# WAIT="$( host.ssh ${hostlist} "hagrp -wait $package State ONLINE -sys ${hostlist[1]} -time 300" 2>&1 )" || e "error" "$WAIT"

# ok "success"

#

# p "Starting failback '$package to '${hostlist}'"

# FAILOVER="$( host.ssh ${hostlist} "hagrp -switch $package -to ${hostlist}" 2>&1 )" || e "failed" "$FAILOVER"

# ok "started"

#

# p "Waiting for failover to complete..."

# WAIT="$( host.ssh ${hostlist} "hagrp -wait $package State ONLINE -sys ${hostlist} -time 300" 2>&1 )" || e "error" "$WAIT"

# ok "success"

# fi

done

exit 0

[INSTSERV][qxz10kv][lpinstiaas02] /global/instserv/bin $