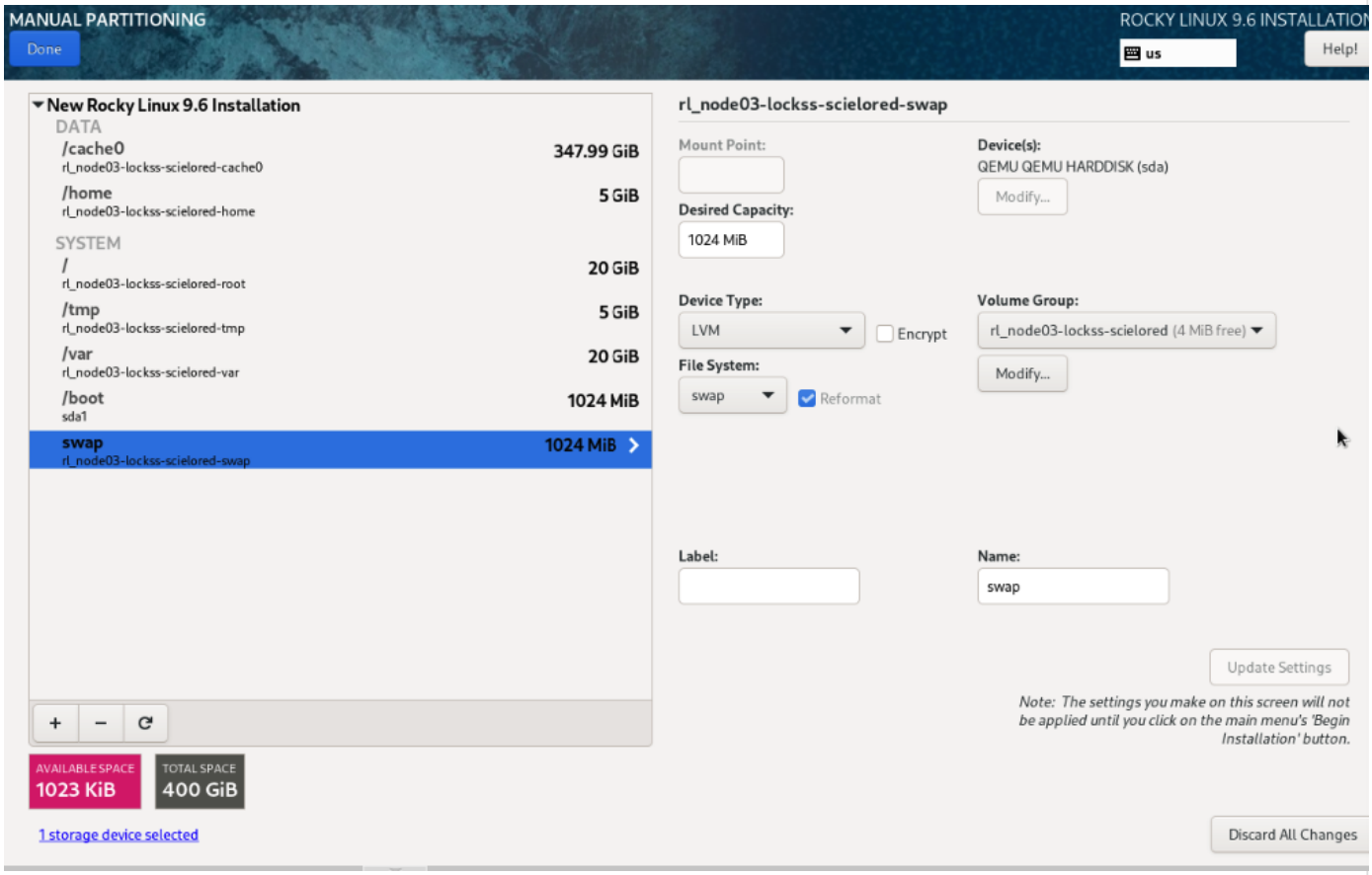
# COMO CONFIGURAR UMA REDE LOCKSS

## CONFIGURAR O SERVIDOR

ESQUEMA DE PARTIÇÃO




Uma vez instalado o sistema operacional vamos instalar os pacotes:

```
yum -y install java-1.8.0-openjdk bind-utils dstat gitiotop lshw lsof lynx  nmap  pciutils rsync smartmontools sysstat tmux wget
```

Vamos agora instalar o pacote do lockss:

```
rpm -Uhiv https://assets.lockss.org/rpm/repo/lockss-daemon-1.78.6-1.noarch.rpm
```

Ajustamos a permissão na pasta /cache0

```
chown lockss. /cache0/
```

## CONFIGURAR CAIXA LOCKSS

```
[root@node03-lockss-scielored lockss]# ./hostconfig
root is configuring
eth0: error fetching interface information: Device not found
LOCKSS host configuration for Linux.
For more information see /etc/lockss/README
Configuring for user lockss
Fully qualified hostname (FQDN) of this machine: [node03-lockss-scielored.scielo.org]
IP address of this machine: [] 192.168.169.166
Is this machine behind NAT?: [N] Y
External IP address for NAT: [] 177.92.116.203
Initial subnet for admin UI access: [192.168.169.0/24]
LCAP V3 protocol port: [9729] 9735
PROXY port: [8080]
Admin UI port: [8081]
Mail relay for this machine: [localhost] mailrelay.scielo.org
Does mail relay mailrelay.scielo.org need user & password: [N]
E-mail address for administrator: [] infra@scielo.org
Path to java: [/usr/bin/java]
Java switches: []
Configuration URL: [http://props.lockss.org:8001/daemon/lockss.xml]
http://200.130.45.61/props/unampln/lockss.xml
Configuration proxy (host:port): [NONE]
Enable config failover: [Y]
Config failover max age: []
Preservation group(s): [prod] unampln
Content storage directories: [] /cache0
Temporary storage directory: [/cache0/tmp]
User name for web UI administration: [] admin
Password for web UI administration user admin: []
Password for web UI administration (again): []

Configuration:
```

```
LOCKSS_CONFIG_VERSION=1
LOCKSS_USER="lockss"
LOCKSS_HOSTNAME=node03-lockss-scielored.scielo.org
LOCKSS_IPADDR=192.168.169.166
LOCKSS_EXTERNAL_IPADDR=177.92.116.203
LOCKSS_V3_PORT=9735
LOCKSS_ACCESS_SUBNET="192.168.169.0/24"
LOCKSS_MAILHUB=mailrelay.scielo.org
LOCKSS_MAILHUB_USER=
LOCKSS_MAILHUB_PASSWORD=
LOCKSS_EMAIL=infra@scielo.org
LOCKSS_JAVA_CMD=/usr/bin/java
LOCKSS_JAVA_SWITCHES=
LOCKSS_JAVA_HEAP=
LOCKSS_PROPS_URL="http://200.130.45.61/props/unampln/lockss.xml"
LOCKSS_PROPS_PROXY="NONE"
LOCKSS_PROPS_SERVER_AUTHENTICATE_KEYSTORE=""
LOCKSS_CONFIG_FAILOVER_ENABLE="Y"
LOCKSS_CONFIG_FAILOVER_MAX_AGE=""
LOCKSS_TEST_GROUP="unampln"
LOCKSS_DISK_PATHS="/cache0"
LOCKSS_ADMIN_USER=admin
LOCKSS_ADMIN_PASSWD=SHA-
256:d17f0af8b8b9ec09d051523e6a3eb76f3c114ab2cc93f330e4d73e0b7f7347e0
LOCKSS_PROXY_PORT=8080
LOCKSS_UI_PORT=8081
LOCKSS_TMPDIR=/cache0/tmp
LOCKSS_CLEAR_TMPDIR=yes
LOCKSS_RELEASE=1.78.6-1
LOCKSS_HOME is
OK to store this configuration: [Y]
Checking content storage dirs
/cache0 exists and is writable by lockss
/var/log/lockss does not exist; shall I create it: [Y]
/cache0/tmp does not exist; shall I create it: [Y]
Done
/tmp/hostconfig.TWkRu: line 518: chkconfig: command not found
LOCKSS will start automatically at next reboot, or you may
 start it now by running   /etc/init.d/lockss start
```

```
/tmp/hostconfig.TWkRu: line 530: mail: command not found
Sending mail failed. Please check mail configuration.
Please also send /tmp/unsent-lockss-config to lockssdiag@lockss.org.
```

Ao executar /etc/init.d/lockss start deu o seguinte erro:

```
[root@node03-lockss-scielored lockss]# /etc/init.d/lockss start
/etc/init.d/lockss: line 8: /etc/init.d/functions: No such file or directory
```

Copiei de um outro servidor o /etc/init.d/functions

```
# -*-Shell-script-*-
#
# functions This file contains functions to be used by most or all
#       shell scripts in the /etc/init.d directory.
#

TEXTDOMAIN=initscripts

# Make sure umask is sane
umask 022

# Set up a default search path.
PATH="/sbin:/usr/sbin:/bin:/usr/bin"
export PATH

if [ $PPID -ne 1 -a -z "$SYSTEMCTL_SKIP_REDIRECT" ] && \
        [ -d /run/systemd/system ] ; then
    case "$0" in
    /etc/init.d/*|/etc/rc.d/init.d/*)
        _use_systemctl=1
        ;;
    esac
fi

systemctl_redirect () {
    local s
    local prog=${1##*/}
    local command=$2
```

```
    local options=""

    case "$command" in
    start)
        s=$"Starting $prog (via systemctl): "
        ;;
    stop)
        s=$"Stopping $prog (via systemctl): "
        ;;
    reload|try-reload)
        s=$"Reloading $prog configuration (via systemctl): "
        ;;
    restart|try-restart|condrestart)
        s=$"Restarting $prog (via systemctl): "
        ;;
    esac

    if [ -n "$SYSTEMCTL_IGNORE_DEPENDENCIES" ] ; then
        options="--ignore-dependencies"
    fi

    if ! systemctl show "$prog.service" > /dev/null 2>&1 || \
            systemctl show -p LoadState "$prog.service" | grep -q 'not-found' ; then
        action $"Reloading systemd: " /bin/systemctl daemon-reload
    fi

    action "$s" /bin/systemctl $options $command "$prog.service"
}

# Get a sane screen width
[ -z "${COLUMNS:-}" ] && COLUMNS=80

# Read in our configuration
if [ -z "${BOOTUP:-}" ]; then
    if [ -f /etc/sysconfig/init ]; then
        . /etc/sysconfig/init
    else
        # verbose ->> very (very!) old bootup look (prior to RHL-6.0?)
        # color ->> default bootup look
```

```
        # other ->> default bootup look without ANSI colors or positioning
        BOOTUP=color
        # Column to start "[  OK  ]" label in:
        RES_COL=60
        # terminal sequence to move to that column:
        MOVE_TO_COL="echo -en \\033[${RES_COL}G"
        # Terminal sequence to set color to a 'success' (bright green):
        SETCOLOR_SUCCESS="echo -en \\033[1;32m"
        # Terminal sequence to set color to a 'failure' (bright red):
        SETCOLOR_FAILURE="echo -en \\033[1;31m"
        # Terminal sequence to set color to a 'warning' (bright yellow):
        SETCOLOR_WARNING="echo -en \\033[1;33m"
        # Terminal sequence to reset to the default color:
        SETCOLOR_NORMAL="echo -en \\033[0;39m"

        # Verbosity of logging:
        LOGLEVEL=1
    fi

    # NOTE: /dev/ttyS* is serial console. "not a tty" is such as
    # /dev/null associated when executed under systemd service units.
    if LANG=C tty | grep -q -e '\(/dev/ttyS\|not a tty\)'; then
        BOOTUP=serial
        MOVE_TO_COL=
        SETCOLOR_SUCCESS=
        SETCOLOR_FAILURE=
        SETCOLOR_WARNING=
        SETCOLOR_NORMAL=
    fi
fi

# Check if any of $pid (could be plural) are running
checkpid() {
    local i

    for i in $* ; do
        [ -d "/proc/$i" ] && return 0
    done
    return 1
```

```
}

__kill_pids_term_kill_checkpids() {
    local base_stime=$1
    shift 1
    local pid=
    local pids=$*
    local remaining=
    local stat=
    local stime=

    for pid in $pids ; do
        [ ! -e  "/proc/$pid" ] && continue
        read -r line < "/proc/$pid/stat" 2> /dev/null

        stat=($line)
        stime=${stat[21]}

        [ -n "$stime" ] && [ "$base_stime" -lt "$stime" ] && continue
        remaining+="$pid "
    done

    echo "$remaining"
    [ -n "$remaining" ] && return 1

    return 0
}

__kill_pids_term_kill() {
    local try=0
    local delay=3;
    local pid=
    local stat=
    local base_stime=

    # We can't initialize stat & base_stime on the same line where 'local'
    # keyword is, otherwise the sourcing of this file will fail for ksh...
    stat=($(< /proc/self/stat))
    base_stime=${stat[21]}
```

```
    if [ "$1" = "-d" ]; then
        delay=$2
        shift 2
    fi

    local kill_list=$*

    kill_list=$(__kill_pids_term_kill_checkpids $base_stime $kill_list)

    [ -z "$kill_list" ] && return 0

    kill -TERM $kill_list >/dev/null 2>&1
    sleep 0.1

    kill_list=$(__kill_pids_term_kill_checkpids $base_stime $kill_list)
    if [ -n "$kill_list" ] ; then
        while [ $try -lt $delay ] ; do
            sleep 1
            kill_list=$(__kill_pids_term_kill_checkpids $base_stime $kill_list)
            [ -z "$kill_list" ] && break
            let try+=1
        done
        if [ -n "$kill_list" ] ; then
            kill -KILL $kill_list >/dev/null 2>&1
            sleep 0.1
            kill_list=$(__kill_pids_term_kill_checkpids $base_stime $kill_list)
        fi
    fi

    [ -n "$kill_list" ] && return 1
    return 0
}

# __proc_pids {program} [pidfile]
# Set $pid to pids from /run* for {program}.  $pid should be declared
# local in the caller.
# Returns LSB exit code for the 'status' action.
__pids_var_run() {
```

```
    local base=${1##*/}
    local pid_file=${2:-/run/$base.pid}
    local pid_dir=$(/usr/bin/dirname $pid_file > /dev/null)
    local binary=$3

    [ -d "$pid_dir" ] && [ ! -r "$pid_dir" ] && return 4

    pid=
    if [ -f "$pid_file" ] ; then
            local line p

        [ ! -r "$pid_file" ] && return 4 # "user had insufficient privilege"
        while : ; do
            read line
            [ -z "$line" ] && break
            for p in $line ; do
                if [ -z "${p//[0-9]/}" ] && [ -d "/proc/$p" ] ; then
                    if [ -n "$binary" ] ; then
                        local b=$(readlink /proc/$p/exe | sed -e 's/\s*(deleted)$//')
                        [ "$b" != "$binary" ] && continue
                    fi
                    pid="$pid $p"
                fi
            done
        done < "$pid_file"

            if [ -n "$pid" ]; then
                    return 0
            fi
        return 1 # "Program is dead and /run pid file exists"
    fi
    return 3 # "Program is not running"
}

# Output PIDs of matching processes, found using pidof
__pids_pidof() {
    pidof -c -o $$ -o $PPID -o %PPID -x "$1" || \
        pidof -c -o $$ -o $PPID -o %PPID -x "${1##*/}"
}
```

```
# A function to start a program.
daemon() {
    # Test syntax.
    local gotbase= force= nicelevel corelimit
    local pid base= user= nice= bg= pid_file=
    local cgroup=
    nicelevel=0
    while [ "$1" != "${1##[-+]}" ]; do
        case $1 in
        '')
            echo $"$0: Usage: daemon [+/-nicelevel] {program}" "[arg1]..."
            return 1
            ;;
        --check)
            base=$2
            gotbase="yes"
            shift 2
            ;;
        --check=?*)
            base=${1#--check=}
            gotbase="yes"
            shift
            ;;
        --user)
            user=$2
            shift 2
            ;;
        --user=?*)
            user=${1#--user=}
            shift
            ;;
        --pidfile)
            pid_file=$2
            shift 2
            ;;
        --pidfile=?*)
            pid_file=${1#--pidfile=}
```

```
        shift
        ;;
    --force)
        force="force"
        shift
        ;;
    [-+][0-9]*)
        nice="nice -n $1"
        shift
        ;;
    *)
        echo $"$0: Usage: daemon [+/-nicelevel] {program}" "[arg1]..."
        return 1
        ;;
  esac
done

# Save basename.
[ -z "$gotbase" ] && base=${1##*/}

# See if it's already running. Look *only* at the pid file.
__pids_var_run "$base" "$pid_file"

[ -n "$pid" -a -z "$force" ] && return

# make sure it doesn't core dump anywhere unless requested
corelimit="ulimit -S -c ${DAEMON_COREFILE_LIMIT:-0}"

# if they set NICELEVEL in /etc/sysconfig/foo, honor it
[ -n "${NICELEVEL:-}" ] && nice="nice -n $NICELEVEL"

# Echo daemon
[ "${BOOTUP:-}" = "verbose" -a -z "${LSB:-}" ] && echo -n " $base"

# And start it up.
if [ -z "$user" ]; then
   $nice /bin/bash -c "$corelimit >/dev/null 2>&1 ; $*"
else
   $nice runuser -s /bin/bash $user -c "$corelimit >/dev/null 2>&1 ; $*"
```

```
    fi

    [ "$?" -eq 0 ] && success $"$base startup" || failure $"$base startup"
}

# A function to stop a program.
killproc() {
    local RC killlevel= base pid pid_file= delay try binary=

    RC=0; delay=3; try=0
    # Test syntax.
    if [ "$#" -eq 0 ]; then
        echo $"Usage: killproc [-p {pidfile} [-b {binary}]] [-d {delay}] {program} [-
signal]"
        return 1
    fi
    if [ "$1" = "-p" ]; then
        pid_file=$2
        shift 2
    fi
    if [ "$1" = "-b" ]; then
        if [ -z $pid_file ]; then
            echo $"-b option can be used only with -p"
            echo $"Usage: killproc [-p {pidfile} [-b {binary}]] [-d {delay}] {program} [-
signal]"
            return 1
        fi
        binary=$2
        shift 2
    fi
    if [ "$1" = "-d" ]; then
        delay=$(echo $2 | awk -v RS=' ' -v IGNORECASE=1 '{if($1!~/^[0-9.]+[smhd]?$/) exit
1;d=$1~/s$|^[0-9.]*$/?1:$1~/m$/?60:$1~/h$/?60*60:$1~/d$/?24*60*60:-1;if(d==-1) exit
1;delay+=d*$1} END {printf("%d",delay+0.5)}')
        if [ "$?" -eq 1 ]; then
            echo $"Usage: killproc [-p {pidfile} [-b {binary}]] [-d {delay}] {program} [-
signal]"
            return 1
        fi
```

```
        shift 2
    fi


    # check for second arg to be kill level
    [ -n "${2:-}" ] && killlevel=$2

    # Save basename.
    base=${1##*/}

    # Find pid.
    __pids_var_run "$1" "$pid_file" "$binary"
    RC=$?
    if [ -z "$pid" ]; then
        if [ -z "$pid_file" ]; then
            pid="$(__pids_pidof "$1")"
        else
            [ "$RC" = "4" ] && { failure $"$base shutdown" ; return $RC ;}
        fi
    fi

    # Kill it.
    if [ -n "$pid" ] ; then
        [ "$BOOTUP" = "verbose" -a -z "${LSB:-}" ] && echo -n "$base "
        if [ -z "$killlevel" ] ; then
            __kill_pids_term_kill -d $delay $pid
            RC=$?
            [ "$RC" -eq 0 ] && success $"$base shutdown" || failure $"$base shutdown"
        # use specified level only
        else
            if checkpid $pid; then
                kill $killlevel $pid >/dev/null 2>&1
                RC=$?
                [ "$RC" -eq 0 ] && success $"$base $killlevel" || failure $"$base
$killlevel"
            elif [ -n "${LSB:-}" ]; then
                RC=7 # Program is not running
            fi
        fi
```

```
    else
        if [ -n "${LSB:-}" -a -n "$killlevel" ]; then
            RC=7 # Program is not running
        else
            failure $"$base shutdown"
            RC=0
        fi
    fi

    # Remove pid file if any.
    if [ -z "$killlevel" ]; then
        rm -f "${pid_file:-/run/$base.pid}"
    fi
    return $RC
}

# A function to find the pid of a program. Looks *only* at the pidfile
pidfileofproc() {
    local pid

    # Test syntax.
    if [ "$#" = 0 ] ; then
        echo $"Usage: pidfileofproc {program}"
        return 1
    fi

    __pids_var_run "$1"
    [ -n "$pid" ] && echo $pid
    return 0
}

# A function to find the pid of a program.
pidofproc() {
    local RC pid pid_file=

    # Test syntax.
    if [ "$#" = 0 ]; then
        echo $"Usage: pidofproc [-p {pidfile}] {program}"
        return 1
```

```
    fi
    if [ "$1" = "-p" ]; then
        pid_file=$2
        shift 2
    fi
    fail_code=3 # "Program is not running"

    # First try "/run/*.pid" files
    __pids_var_run "$1" "$pid_file"
    RC=$?
    if [ -n "$pid" ]; then
        echo $pid
        return 0
    fi

    [ -n "$pid_file" ] && return $RC
    __pids_pidof "$1" || return $RC
}

status() {
    local base pid lock_file= pid_file= binary=

    # Test syntax.
    if [ "$#" = 0 ] ; then
        echo $"Usage: status [-p {pidfile}] [-l {lockfile}] [-b {binary}] {program}"
        return 1
    fi
    if [ "$1" = "-p" ]; then
        pid_file=$2
        shift 2
    fi
    if [ "$1" = "-l" ]; then
        lock_file=$2
        shift 2
    fi
    if [ "$1" = "-b" ]; then
        if [ -z $pid_file ]; then
            echo $"-b option can be used only with -p"
            echo $"Usage: status [-p {pidfile}] [-l {lockfile}] [-b {binary}] {program}"
```

```
            return 1
        fi
        binary=$2
        shift 2
    fi
    base=${1##*/}

    if [ "$_use_systemctl" = "1" ]; then
        systemctl status ${0##*/}.service
        ret=$?
        # LSB daemons that dies abnormally in systemd looks alive in systemd's eyes due to
RemainAfterExit=yes
        # lets adjust the reality a little bit
        if systemctl show -p ActiveState ${0##*/}.service | grep -q '=active$' && \
        systemctl show -p SubState ${0##*/}.service | grep -q '=exited$' ; then
            ret=3
        fi
        return $ret
    fi

    # First try "pidof"
    __pids_var_run "$1" "$pid_file" "$binary"
    RC=$?
    if [ -z "$pid_file" -a -z "$pid" ]; then
        pid="$(__pids_pidof "$1")"
    fi
    if [ -n "$pid" ]; then
        echo $"${base} (pid $pid) is running..."
        return 0
    fi

    case "$RC" in
    0)
        echo $"${base} (pid $pid) is running..."
        return 0
        ;;
    1)
        echo $"${base} dead but pid file exists"
        return 1
```

```
        ;;
    4)
        echo $"${base} status unknown due to insufficient privileges."
        return 4
        ;;
    esac
    if [ -z "${lock_file}" ]; then
        lock_file=${base}
    fi
    # See if /var/lock/subsys/${lock_file} exists
    if [ -f /var/lock/subsys/${lock_file} ]; then
        echo $"${base} dead but subsys locked"
        return 2
    fi
    echo $"${base} is stopped"
    return 3
}

echo_success() {
    [ "$BOOTUP" = "color" ] && $MOVE_TO_COL
    echo -n "["
    [ "$BOOTUP" = "color" ] && $SETCOLOR_SUCCESS
    echo -n $"  OK  "
    [ "$BOOTUP" = "color" ] && $SETCOLOR_NORMAL
    echo -n "]"
    echo -ne "\r"
    return 0
}

echo_failure() {
    [ "$BOOTUP" = "color" ] && $MOVE_TO_COL
    echo -n "["
    [ "$BOOTUP" = "color" ] && $SETCOLOR_FAILURE
    echo -n $"FAILED"
    [ "$BOOTUP" = "color" ] && $SETCOLOR_NORMAL
    echo -n "]"
    echo -ne "\r"
    return 1
}
```

```
echo_passed() {
    [ "$BOOTUP" = "color" ] && $MOVE_TO_COL
    echo -n "["
    [ "$BOOTUP" = "color" ] && $SETCOLOR_WARNING
    echo -n $"PASSED"
    [ "$BOOTUP" = "color" ] && $SETCOLOR_NORMAL
    echo -n "]"
    echo -ne "\r"
    return 1
}

echo_warning() {
    [ "$BOOTUP" = "color" ] && $MOVE_TO_COL
    echo -n "["
    [ "$BOOTUP" = "color" ] && $SETCOLOR_WARNING
    echo -n $"WARNING"
    [ "$BOOTUP" = "color" ] && $SETCOLOR_NORMAL
    echo -n "]"
    echo -ne "\r"
    return 1
}

# Inform the graphical boot of our current state
update_boot_stage() {
    if [ -x /bin/plymouth ]; then
        /bin/plymouth --update="$1"
    fi
    return 0
}

# Log that something succeeded
success() {
    [ "$BOOTUP" != "verbose" -a -z "${LSB:-}" ] && echo_success
    return 0
}

# Log that something failed
failure() {
```

```
    local rc=$?
    [ "$BOOTUP" != "verbose" -a -z "${LSB:-}" ] && echo_failure
    [ -x /bin/plymouth ] && /bin/plymouth --details
    return $rc
}

# Log that something passed, but may have had errors. Useful for fsck
passed() {
    local rc=$?
    [ "$BOOTUP" != "verbose" -a -z "${LSB:-}" ] && echo_passed
    return $rc
}

# Log a warning
warning() {
    local rc=$?
    [ "$BOOTUP" != "verbose" -a -z "${LSB:-}" ] && echo_warning
    return $rc
}

# Run some action. Log its output.
action() {
    local STRING rc

    STRING=$1
    echo -n "$STRING "
    shift
    "$@" && success $"$STRING" || failure $"$STRING"
    rc=$?
    echo
    return $rc
}

# returns OK if $1 contains $2
strstr() {
    [ "${1#*$2*}" = "$1" ] && return 1
    return 0
}
```

```
# Check whether file $1 is a backup or rpm-generated file and should be ignored
# Copy of the function is present in usr/sbin/service
is_ignored_file() {
    case "$1" in
    *~ | *.bak | *.old | *.orig | *.rpmnew | *.rpmorig | *.rpmsave)
        return 0
        ;;
    esac
    return 1
}

# Convert the value ${1} of time unit ${2}-seconds into seconds:
convert2sec() {
  local retval=""

  case "${2}" in
    deci)   retval=$(awk "BEGIN {printf \"%.1f\", ${1} / 10}") ;;
    centi)  retval=$(awk "BEGIN {printf \"%.2f\", ${1} / 100}") ;;
    mili)   retval=$(awk "BEGIN {printf \"%.3f\", ${1} / 1000}") ;;
    micro)  retval=$(awk "BEGIN {printf \"%.6f\", ${1} / 1000000}") ;;
    nano)   retval=$(awk "BEGIN {printf \"%.9f\", ${1} / 1000000000}") ;;
    piko)   retval=$(awk "BEGIN {printf \"%.12f\", ${1} / 1000000000000}") ;;
  esac

  echo "${retval}"
}

# Evaluate shvar-style booleans
is_true() {
    case "$1" in
    [tT] | [yY] | [yY][eE][sS] | [oO][nN] | [tT][rR][uU][eE] | 1)
        return 0
        ;;
    esac
    return 1
}

# Evaluate shvar-style booleans
is_false() {
```

```
    case "$1" in
    [fF] | [nN] | [nN][oO] | [oO][fF][fF] | [fF][aA][lL][sS][eE] | 0)
        return 0
        ;;
    esac
    return 1
}

# Apply sysctl settings, including files in /etc/sysctl.d
apply_sysctl() {
    if [ -x /lib/systemd/systemd-sysctl ]; then
    /lib/systemd/systemd-sysctl
    else
        for file in /usr/lib/sysctl.d/*.conf ; do
            is_ignored_file "$file" && continue
            [ -f /run/sysctl.d/${file##*/} ] && continue
            [ -f /etc/sysctl.d/${file##*/} ] && continue
            test -f "$file" && sysctl -e -p "$file" >/dev/null 2>&1
        done
        for file in /run/sysctl.d/*.conf ; do
            is_ignored_file "$file" && continue
            [ -f /etc/sysctl.d/${file##*/} ] && continue
            test -f "$file" && sysctl -e -p "$file" >/dev/null 2>&1
        done
        for file in /etc/sysctl.d/*.conf ; do
            is_ignored_file "$file" && continue
            test -f "$file" && sysctl -e -p "$file" >/dev/null 2>&1
        done
        sysctl -e -p /etc/sysctl.conf >/dev/null 2>&1
    fi
}

# A sed expression to filter out the files that is_ignored_file recognizes
__sed_discard_ignored_files='/\(~\|\.bak\|\.old\|\.orig\|\.rpmnew\|\.rpmorig\|\.rpmsave\)$
/d'

if [ "$_use_systemctl" = "1" ]; then
        if  [ "x$1" = xstart -o \
              "x$1" = xstop -o \
```

```
               "x$1" = xrestart -o \
               "x$1" = xreload -o \
               "x$1" = xtry-restart -o \
               "x$1" = xforce-reload -o \
               "x$1" = xcondrestart ] ; then

        systemctl_redirect $0 $1
        exit $?
    fi
fi

strstr "$(cat /proc/cmdline)" "rc.debug" && set -x
return 0
```

ao tentar subir novamente o serviço

```
[root@node03-lockss-scielored lockss]# /etc/init.d/lockss start
Reloading systemd:                                         [  OK  ]
Starting lockss (via systemctl):  Failed to start lockss.service: Unit lockss.service not
found.
                                                           [FAILED]
```

crie o script de start

/usr/local/libexec/lockss-start.sh

```
#!/usr/bin/env bash
set -euo pipefail

PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

USER_LOCKSS=lockss
HOME_LOCKSS=/home/lockss
LOG_DIR=/var/log/lockss
LOG_FILE=/var/log/lockss/stdout
LOCKFILE=/var/run/lockss/startdaemon.lock.lockss
KEEPGOING=/home/lockss/KeepGoing

mkdir -p /var/run/lockss "${LOG_DIR}" "${HOME_LOCKSS}"
touch "${LOG_FILE}"
```

```
chown "${USER_LOCKSS}:${USER_LOCKSS}" "${HOME_LOCKSS}" "${LOG_FILE}"
chmod 755 /var/run/lockss "${LOG_DIR}" "${HOME_LOCKSS}"
chmod 644 "${LOG_FILE}"

# não sobe outra cópia por cima
if pgrep -f '^/bin/bash /etc/lockss/startdaemon lockss$' >/dev/null 2>&1; then
  echo "LOCKSS already appears to be running"
  exit 0
fi

rm -f "${LOCKFILE}" "${KEEPGOING}"

/etc/lockss/startdaemon "${USER_LOCKSS}"

sleep 3

if ! pgrep -f '^/bin/bash /etc/lockss/startdaemon lockss$' >/dev/null 2>&1; then
  echo "LOCKSS startdaemon did not stay running" >&2
  exit 1
fi

if ! pgrep -u "${USER_LOCKSS}" -f '^/bin/sh /etc/lockss/rundaemon wait$' >/dev/null 2>&1;
then
  echo "LOCKSS rundaemon did not start" >&2
  exit 1
fi

exit 0
```

Permissão

```
chmod 755 /usr/local/libexec/lockss-start.sh
```

Crie o script de stop

/usr/local/libexec/lockss-stop.sh

```
#!/usr/bin/env bash
set -euo pipefail
```

```
rm -f /home/lockss/KeepGoing || true

pkill -TERM -f '^/bin/bash /etc/lockss/startdaemon lockss$' || true
pkill -TERM -f '^runuser -s /bin/bash - lockss -c ulimit -S -c 0 >/dev/null 2>&1 ;
/etc/lockss/rundaemon wait$' || true
pkill -TERM -u lockss -f '^/bin/sh /etc/lockss/rundaemon wait$' || true
pkill -TERM -u lockss -f java || true

sleep 5

pkill -KILL -f '^/bin/bash /etc/lockss/startdaemon lockss$' || true
pkill -KILL -f '^runuser -s /bin/bash - lockss -c ulimit -S -c 0 >/dev/null 2>&1 ;
/etc/lockss/rundaemon wait$' || true
pkill -KILL -u lockss -f '^/bin/sh /etc/lockss/rundaemon wait$' || true
pkill -KILL -u lockss -f java || true

rm -f /var/run/lockss/startdaemon.lock.lockss || true
```

Permissão:

```
chmod 755 /usr/local/libexec/lockss-stop.sh
```

criei /etc/systemd/system/lockss.service

```
[Unit]
Description=LOCKSS daemon
After=network.target local-fs.target

[Service]
Type=oneshot
RemainAfterExit=yes
ExecStart=/usr/local/libexec/lockss-start.sh
ExecStop=/usr/local/libexec/lockss-stop.sh
TimeoutStartSec=60
TimeoutStopSec=60

[Install]
WantedBy=multi-user.target
```

Agora inicie o daemon

```
systemctl daemon-reload
systemctl status lockss
```

Depois valide:

```
cat /var/run/lockss/startdaemon.lock.lockss
ps -fp "$(cat /var/run/lockss/startdaemon.lock.lockss)"
ls -l /home/lockss/KeepGoing
```

Depois que isso estiver funcionando, confirme sempre:

```
ps -eo pid,ppid,user,cmd | egrep 'startdaemon|rundaemon|[j]ava'
```

Com o serviço rodando, você deve ver **apenas uma cadeia de processos**.

Se aparecerem várias, significa que algum start anterior não foi limpo.

**□□ Resumo honesto:**

O problema não é Rocky 9. O problema é que o startdaemon do LOCKSS foi escrito com um modelo de supervisão próprio (lockfile + KeepGoing + background loops), que não conversa bem com o modelo de gerenciamento de processos do systemd. Então a abordagem correta é **delegar o stop para um script que mata explicitamente os processos**, como você já comprovou que funciona.

## COMPILADO LOCKSS.JAR

O lockss.jar deve ser alterado para a versão compilada pelo Rondineli. Copie o lockss.jar para o servidor e siga:

```
systemctl stop lockss
cd /usr/share/lockss/
mv lockss.jar lockss.jar.original
cp -a /home/rondinelesaad/lockss.jar .
systemctl start lockss
```

Revision #7
Created 1 December 2025 18:43:10 by Rondineli G. Saad
Updated 6 March 2026 18:12:51 by Rondineli G. Saad