

# Scripting

- [docker compose helper scripts](#)
- [SSL Wildcard Letsencrypt Certificate Renewal](#)
- [ntfy.sh notification wrapper](#)
- [folder-backup configurable folder backup wrapper](#)
- [mkv2mp4 ffmpeg/ffprobe wrapper](#)
- [Docker Container Execute on Host Client/Server App](#)
- [Movie Update TimeStamp to Theatrical Release Date](#)
- [Simple Log File Reset/Wipe](#)
- [docker-compose create helper script \(dialog\)](#)

# docker compose helper scripts

[pknw1logo-white.png](#)

`/usr/local/bin/uplog`

## Helper Scripts

The docker helper scripts are to allow more rapid access to frequent requests

- ☐ create `/usr/local/bin/uplog`
- ☐ `chmod +x /usr/local/bin/uplog`

```
#!/bin/bash
# /usr/local/bin/uplog

if [[ $# -eq 0 ]]
then
    docker compose up -d && docker compose logs -f
else
    docker compose -f $1.yml up -d && docker compose -f $1.yml logs -f
fi
```

- ☐ update profile aliases
- ☐ `vi ~/.bash_aliases`
- ☐ logout and back in

```
# additional bash_alias entries for docker management

alias down='docker compose down'
alias up='docker compose up -d'
alias logs='docker compose logs -f'
```

no additional details

## docker-image-prune-3months

a script to remove all un-used docker images over 3 months old

```
#!/bin/bash
# docker-image-prune-3months.sh

echo "BEFORE $(docker image ls |wc -l)"
time docker image prune --all --filter "until=2160h"
echo "AFTER $(docker image ls |wc -l)"
```

## docker-latest-images

pulls the latest docker images and restarts affected containers

```
#!/bin/bash
# docker-latest-images.sh

find /home/docker/services -iname docker-compose.yml -exec docker compose -f "{}" pull \;
find /home/docker/services -mindepth 1 -maxdepth 1 -type d | while read -r DIR
do
  cd "${DIR}"
  TEST=$(docker compose ps -q)
  [ -z $TEST ] && cd "${DIR}" && docker compose up
  [ -z $TEST ] || echo "${DIR} not running"
done
```

## further info

<a href="#"><u>Product Home</u></a>	n/a
<a href="#"><u>Documentation</u></a>	n/a

<b><u>Github</u></b>	n/a
<b><u>DockerHub</u></b>	n/a
<b><u>Misc</u></b>	n/a

# SSL Wildcard Letsencrypt Certificate Renewal

[pknw1logo-white.png](#)

```
/usr/local/bin/renew-ssl-wildcard.sh
```

## SSL Check and Renew Wildcard

uses Letsencrypt with OVH api to request a wildcard certificate

```
#@pknw1
#!/bin/bash
# renew wildcard SSL certs

DOMAIN=$1

function renew() {
sudo docker run -it --rm --name certbot \
    -v "/etc/letsencrypt:/etc/letsencrypt" \
    -v "/var/lib/letsencrypt:/var/lib/letsencrypt" \
    -v "/root/ovh.conf:/ovh.conf" \
    certbot/dns-ovh certonly --dns-ovh --dns-ovh-credentials /ovh.conf \
    --agree-tos -m pknw1@hotmail.co.uk \
    -d *."${DOMAIN}" -d "${DOMAIN}"
}

function merge() {
if [ -f /etc/ssl/private/wildcard-"${DOMAIN}".pem ]; then sudo rm /etc/ssl/private/wildcard-
"${DOMAIN}".pem; fi
sudo find /etc/letsencrypt/live -type l -iname '*pem' -mmin -3 -exec cat "{}" >>
/etc/ssl/private/wildcard-"${DOMAIN}".pem \;
}

function pfx() {
```

```
if [ -f /etc/ssl/private/wildcard- "${DOMAIN} ".pem ]; then sudo openssl pkcs12 -inkey
/etc/ssl/private/wildcard- "${DOMAIN} ".pem -in /etc/ssl/private/wildcard- "${DOMAIN} ".pem -
export -out /etc/ssl/private/wildcard- "${DOMAIN} ".pfx -passout pass:
fi
}

renew
merge
pfx
ntfy.sh ssl renewal complete
```

## **pemCheck script**

This script checks a certificate to check if it is expired or not

```
#!/bin/bash
# /usr/local/bin/pemCheck <certfile>

CURRENT=$(date +%s)
CERT=$(openssl x509 -enddate -noout -in "${1}" | awk -F\= '{print $2}')
CHECK=$(date -d "${CERT}" +%s)

if [[ $CHECK -lt $CURRENT ]]
then
echo "removing ${1}"
rm "${1}"
else
echo "valid cert ${1} expires ${CERT}"
fi
```

asdas

- ☐ script info
- ☐ check mark

## **further info**

<b><u>Product Home</u></b>	<a href="#">Link</a>
<b><u>Documentation</u></b>	<a href="#">Link</a>
<b><u>Github</u></b>	<a href="#">Link</a>
<b><u>DockerHub</u></b>	<a href="#">Link</a>
<b><u>Misc</u></b>	<a href="#">Link</a>

more text more text

# ntfy.sh notification wrapper

[pknw1logo-white.png](#)

```
/usr/local/bin/ntfy.sh
```

## Custom ntfy wrapper

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged.

```
#!/bin/bash
#/usr/local/bin/ntfy.sh

TOPIC="notflix"
LOGFILE="/var/log/ntfy-${TOPIC}"
OUTPUT=""

if [ $# -eq 0 ]
then
    echo "requires at least one arg"
else
    for i in "$@"
    do
        MSG="${MSG} $i"
    done
    OUTPUT=$(curl -s -d "${MSG}" ntfy.sh/"${TOPIC}")
    echo "${OUTPUT}" >> "${LOGFILE}"
    echo "Message${MSG} sent to topic ${TOPIC}"
fi
```

sdfsd



☐ requires local install of ntfy cli

☐

## further info

<b><u>Product Home</u></b>	<a href="#">Link</a>
<b><u>Documentation</u></b>	<a href="#">Link</a>
<b><u>Github</u></b>	<a href="#">Link</a>
<b><u>DockerHub</u></b>	<a href="#">Link</a>
<b><u>Misc</u></b>	<a href="#">Link</a>

more text more text

# folder-backup configurable folder backup wrapper

[pknw1logo-white.png](#)

```
/usr/local/bin/folder-backup
```

## **Custom Backup Script**

Custom script using a configurable persistent local config to archive and compress the folder contents while excluding any file/folder patterns

```
#!/bin/bash
TARGET='/dev/shm'
LOGPATH='/dev/shm'

if [[ $# -eq 0 ]]
then
    SOURCE=$(pwd)
else
    SOURCE="${1}"
fi

if ! [[ -f "${SOURCE}/.backup-options" ]]
then
    FULL_PATH=$(realpath "${SOURCE}")
    SOURCE_FOLDER=$(echo "${FULL_PATH}" | awk -F/ '{print $NF}')

    touch "${FULL_PATH}/.backup-options"
    cat <<EOF > "${FULL_PATH}/.backup-options"
BACKUP_NAME=${SOURCE_FOLDER}
BACKUP_ROOT=${FULL_PATH}
EXCLUDE=(.git .venv)
DESTINATION=${TARGET}
EOF
```

```

echo "created config file .backup-options - update exclusions"
cat "${FULL_PATH}/.backup-options"

else
    source "${SOURCE}/.backup-options"
    excludes=()    # start with an empty array
    for excl in ${EXCLUDE[*]} ; do    # for each extra argument...
        excludes+=("--exclude "$excl")    # add an exclude to the array
    done

    if [[ -f ${DESTINATION}/${BACKUP_NAME}.tar ]]
    then
        BACKUP_OPTIONS="uzvf"
    else
        BACKUP_OPTIONS="czvf"
    fi

    echo $(pwd)
    cat << EOF >> ${LOGPATH}/${BACKUP_NAME}.log

    $(date) start backup
EOF

    tar -${BACKUP_OPTIONS} ${DESTINATION}/${BACKUP_NAME}.tgz ${excludes[@]} * 2>&1 >>
${LOGPATH}/${BACKUP_NAME}.log

    cat << EOF >> ${LOGPATH}/${BACKUP_NAME}.log

    $(date) end backup
EOF
fi

if [ -f ${LOGPATH}/${BACKUP_NAME}.log ]
then
    tail -n20 ${LOGPATH}/${BACKUP_NAME}.log
fi

```

execution

☐ either change to the directory that is targetted for backup or pass the full path as a paramater

☐ 1st run creates the local `.backup-options` file with **no backup**

```
BACKUP_NAME=sso
BACKUP_ROOT=/home/docker/config/sso
EXCLUDE=(.git .venv)
DESTINATION=/dev/shm
```

☐ 2nd and subsequent executions read the backup parameters from the config file and tar/compress the backup

## further info

<b><u>Product Home</u></b>	<a href="#">Link</a>
<b><u>Documentation</u></b>	<a href="#">Link</a>
<b><u>Github</u></b>	<a href="#">Link</a>
<b><u>DockerHub</u></b>	<a href="#">Link</a>
<b><u>Misc</u></b>	<a href="#">Link</a>

more text more text

# mkv2mp4 ffmpeg/ffprobe wrapper

[pknw1logo-white.png](#)

```
/usr/local/bin/mkv2mp4.sh
```

## Quick MKV to MP4 conversion with FFMPEG

Converts from MKV to MP4 to allow rapid streaming for media

```
#!/bin/bash
#   mkv2mp4.sh

function quit() {
    echo "[FAIL] - ${1}"
    exit
}

function success() {
    echo "[ OK ] - ${1}"
}

[[ -z $1 ]] && quit "no source file specified"
! [[ -z $(which ffmpeg) ]] && success "ffmpeg binary located" || quit "unable to locate ffmpeg"
! [[ -z $(which ffprobe) ]] && success "ffprobe binary located" || quit "unable to locate ffprobe"
! [[ -z $(which figlet) ]] && success "figlet binary located" || quit "unable to locate figlet"

INPUT="${1}"
```

```

FULL_SOURCE=$(realpath "${INPUT}")
FOLDER=$(dirname "${FULL_SOURCE}")
SOURCE_FILE=$(echo "${FULL_SOURCE}" | awk -F/ '{print $NF}')
SOURCE_EXT=$(echo "${SOURCE_FILE}" | awk -F. '{print $NF}')
TARGET_FILE=$(echo "${SOURCE_FILE}" | sed 's/.mkv/.mp4/')

cd "${FOLDER}"
figlet MKV2MP4 >> "${FOLDER}/mkv2mp4.log"
cat << EOF >> "${FOLDER}/mkv2mp4.log"

DATE:$(date)
SOURCE:${FULL_SOURCE}
TARGET:${FOLDER}/${TARGET_FILE}

EOF

#[[ -f "${SOURCE_FILE}" ]] && success "source file located" || quit "unable to access source
file"
#touch /tmp/test/.mkv2mp4 && success "folder writeable" || quit "no permission to write to
folder"
#[[ -f "${FOLDER}/${TARGET_FILE}" ]] && quit "target file ${TARGET_FILE} already exists" ||
success "target filename ${TARGET} is available"

#figlet SOURCE >> "${FOLDER}/mkv2mp4.log"
#echo $(date) >> "${FOLDER}/mkv2mp4.log"
#ffprobe "${FULL_SOURCE}" >> "${FOLDER}/mkv2mp4.log" || quit "ffprobe failed to scan file"

#figlet ENCODE >> "${FOLDER}/mkv2mp4.log"
#echo $(date) >> "${FOLDER}/mkv2mp4.log"
ffmpeg -i "${FULL_SOURCE}" -c copy "${TARGET_FILE}" && mv "${FULL_SOURCE}" /tmp/encodes/

#figlet OUTPUT >> "${FOLDER}/mkv2mp4.log"
#echo $(date) >> "${FOLDER}/mkv2mp4.log"
#ffprobe "${FOLDER}/${TARGET_FILE}" >> "${FOLDER}/mkv2mp4.log" || quit "ffprobe failed to scan
file"

#[[ -f "${FOLDER}/${TARGET_FILE}" ]] && mv "${FULL_SOURCE}" /tmp/encodes/

```

```
sudo chmod -R 777 "${FOLDER}"
```

## execution

- ☐ `mkv2mp4.sh </path/to/inputfile.mkv>`
- ☐ reads the source file
  - ☐ uses ffprobe to get metadata
  - ☐ fast re-code the mkv to mp4
  - ☐ replace bad charachters in file and folder names

## further info

<b><u>Product Home</u></b>	<a href="#">Link</a>
<b><u>Documentation</u></b>	<a href="#">Link</a>
<b><u>Github</u></b>	<a href="#">Link</a>
<b><u>DockerHub</u></b>	<a href="#">Link</a>
<b><u>Misc</u></b>	<a href="#">Link</a>

more text more text

# Docker Container Execute on Host Client/Server App

[pknw1logo-white.png](#)

```
/usr/local/bin/pipe-exec
```

```
/usr/local/bin/pipe-response
```

## Client Server App for Host execution from within docker

The app runs in the container and uses fifo pipes to pass commands to the server side of the app running on the docker host; the commands are executed and responses piped back to the container

### Server Side Component

```
#!/bin/bash
# /usr/local/bin/pipe-exec
# server side of app

if ! [[ -p /var/run/exec ]]
then
    [[ -d /var/run/exec ]] && rm -rf /var/run/exec
    mkfifo /var/run/exec || echo "error" && exit 99
fi

if ! [[ -p /var/run/response ]]
then
    [[ -d /var/run/response ]] && rm -rf /var/run/response
    mkfifo /var/run/response || echo "error" && exit 99
fi

while true; do eval "$(cat /var/run/exec)" > /var/run/response; done
```



- ☐ pipe-exec runs on a docker host
- ☐ listens to fifo pipe /var/run/exec
- ☐ creates pipe if doesn't exist
- ☐ executes the data from the pipe on the docker host
- ☐ sends output of the command to fifo pipe /var/run/response
- ☐ ideally run server component as a service

```
# systemd service file to run pipe-exec on startup
# /etc/systemd/system/pipe_exec.service
```

```
[Unit]
```

```
Description=Pipe Exec
```

```
[Service]
```

```
Type=simple
```

```
ExecStart=/usr/local/bin/pipe-exec
```

```
[Install]
```

```
WantedBy=multi-user.target
```

```
#!/bin/bash
```

```
# client side of app
```

```
# /usr/local/bin/pipe-response
```

```
echo "$1 $2 $3" > /var/run/exec
```

```
cat /var/run/response
```

## Client Application for inclusion and execution within docker container

- ☐ pipe-response should be on the docker container filesystem or mapped in as a volume
- ☐ /var/run/exec must be mapped as a volume
- ☐ /var/run/response must be mapped in as a volume

```
# example docker-compose.yml for mapping folders and script
# within the container pass commands as
```

```
# pipe-response curl ifconfig.me
# returns host IP

---

services:
  your_container:
    container_name: your_container
    image: jamesread/your_container
    user: root
    volumes:
      - ./config/your_container:/config
      - /var/run/docker.sock:/var/run/docker.sock
      - /var/run/exec:/var/run/exec
      - /var/run/response:/var/run/response:ro
      - ./config/pipe-response:/usr/bin/pipe-response:ro
    restart: unless-stopped
    networks:
      - admin
    environment:
      - VIRTUAL_HOST=your_container.admin.pknw1.co.uk
      - VIRTUAL_PORT=1337

networks:
  admin:
    external: true
    name: admin
```

## **further info**

<b><u>Product Home</u></b>	<a href="#">Link</a>
<b><u>Documentation</u></b>	<a href="#">Link</a>
<b><u>Github</u></b>	<a href="#">Link</a>

<b><u>DockerHub</u></b>	<a href="#">Link</a>
<b><u>Misc</u></b>	<a href="#">Link</a>

more text more text

# Movie Update TimeStamp to Theatrical Release Date

[pknw1logo-white.png](#)

```
/usr/local/bin/releaseDate
```

## Movie File Date/Time modifier

Reads the selected movie theatrical release date from previously stored metadata and sets the movie date and time to the release date so that it displays orderly in Plex

```
#!/bin/bash
# set releaseDate

TS=$(cat "${1}" | jq '.releaseDate' | sed 's/"//g' | awk -FT '{print $1}' | awk -F- '{print $1$2$3}')0000.00
FOLDER=$(dirname "${1}")
if [ "${TS}" == "0000.00" ] || [ "${TS}" == "null0000.00" ]
then
    echo "${FOLDER} invalid"
else
    echo "${TS} ${FOLDER}" || echo "${FOLDER} invalid"
    touch -a -m -t "${TS}" "${FOLDER}" || "${FOLDER} invalid"
    touch -a -m -t "${TS}" "${FOLDER}/*" || "${FOLDER} invalid"
fi
```

## execution

- ☐ ensure the movie has been scraped
- ☐ requires <movie-name>.nfo to exist
- ☐ parses nfo for release date

☐ updates the file timestamp

## **further info**

<b><u>Product Home</u></b>	<a href="#">Link</a>
<b><u>Documentation</u></b>	<a href="#">Link</a>
<b><u>Github</u></b>	<a href="#">Link</a>
<b><u>DockerHub</u></b>	<a href="#">Link</a>
<b><u>Misc</u></b>	<a href="#">Link</a>

more text more text

# Simple Log File Reset/Wipe

[pknw1logo-white.png](#)

```
/usr/local/bin/wipe
```

## Script Headline

a simple wrapper to truncate any log file or open file by cat'ing /dev/null into the file

```
#!/bin/bash
# /usr/local/bin/wipe

cat /dev/null > $1
```

sdfsd

☐ wipe /var/log/auth.log

☐ service rsyslog restart

## further info

<a href="#">Product Home</a>	<a href="#">Link</a>
<a href="#">Documentation</a>	<a href="#">Link</a>
<a href="#">Github</a>	<a href="#">Link</a>
<a href="#">DockerHub</a>	<a href="#">Link</a>
<a href="#">Misc</a>	<a href="#">Link</a>

more text more text



# docker-compose create helper script (dialog)

[pknw1logo-white.png](#)

```
/usr/local/bin/create-compose.sh
```

## **Script Headline**

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged.

```
# Create a temporary file to store dialog output
tempfile=$(mktemp /tmp/dialog.XXXXXX)
BACKGROUND_TITLE="Docker Compose Configurator"
dir=$(dirname $(pwd) | awk -F/ '{print $NF}')
existing_networks=$(docker network ls |awk '{print $2" "$3}'|grep -v NAME)
existing_users=$(cat /etc/passwd|egrep -v 'nologin|false'| awk -F: '{print $1" "$3}')
existing_user_ids=$(cat /etc/passwd|egrep -v 'nologin|false'| awk -F: '{print ":"$3":"}')
tz=$(cat /etc/timezone)


# Function to get user input with dialog
exec 3>&1;

function status() {
    status_update="${1}"
    echo "${status_update}" >> /tmp/dialog-log
}
```



```

function wrapper() {
    feature="${1}"
    title="${2}"
    dimensions="${3}"
    data "${4}"

    response=$(dialog --backtitle "${BACKGROUND_TITLE} - ${title}" \
        --${feature} "${title}" \
        "${dimensions}" \
        "${data}" \
        2>&1 1>&3 )

    exit_code=$?

    case $exit_code in
        0) status "${2} - success exit code"
            ;;
        3) status "${2} - alternate button"
            ;;
        *) status "${2} - non-zero error code"
            ;;
    esac

    if [[ -z "${response}" ]]
    then
        echo "${response}"
    fi
}

## SERVICE_NAME
    service_name=$(dialog --backtitle "${BACKGROUND_TITLE} - Service Name" --inputbox
'Service Name' 0 0 ${dir} 2>&1 1>&3);

## DOCKER COMPOSE IMAGE and DOCKER IMAGE TEST PULL
    image_name=$(dialog --backtitle "${BACKGROUND_TITLE} - Docker Image Name" --inputbox
'Image Name' 0 0 2>&1 1>&3);

    rm /tmp/image_pull
    image_test=$(docker pull $image_name || echo error > /tmp/image_pull)
    if [[ $(cat /tmp/image_pull) == "error" ]]

```

```

then
    image_name=$(dialog --backtitle "${BACKGROUND_TITLE}" - Docker Error" --inputbox
'Docker Image not found - Try again' 0 0 2>&1 1>&3);
fi

## TOP LEVEL DOMAIN

top_level_domain=$(dialog --backtitle "${BACKGROUND_TITLE}" --title "Top Level Domain" --
inputbox "domain" 8 50 "pknw1.co.uk" 2>&1 1>&3);

## DOCKER CONFIG ROOT FOLDER

config_folder=$(dialog --backtitle "${BACKGROUND_TITLE}" --title "config folder" --
inputbox "enter the docker config root path" 8 50 "/etc/docker/config" 2>&1 1>&3)

## DOCKER NETWORK SELECTION & CREATION

docker_network=$(dialog --backtitle "${BACKGROUND_TITLE}" --menu "Choose Network: " 22 50
15 ${existing_networks} New Network 2>&1 1>&3)
if [[ ${docker_network} == "New" ]]
then
    docker_network=$(dialog --backtitle "${BACKGROUND_TITLE}" --inputbox 'New Network
Name:' 0 0 2>&1 1>&3);
fi

## USER SELECTION

choose_user=$(dialog --backtitle "${BACKGROUND_TITLE}" --menu "Choose User: " 22 35 10
${existing_users} 2>&1 1>&3)
rm /tmp/id

## GROUP SELECTION

for ID in ${existing_user_ids}; do cat /etc/group | grep $ID|awk -F: '{print $1" "$3}' >>
/tmp/id; done
existing_groups=$(cat /tmp/id| sort -u)
select_group=$(dialog --backtitle "${BACKGROUND_TITLE}" --menu "Choose Group: " 22 35 10
${existing_groups} 2>&1 1>&3)

## RESTART POLICY

restart=$(dialog --backtitle "${BACKGROUND_TITLE}" --menu "Choose Restart Option: " 22 35 10
no 'never restart' always 'always start' on-failure 'fail' unless-stopped 'unl' 2>&1 1>&3)

## NGINX-PROXY APP PORT

app_port=$(dialog --inputbox 'App Port for Proxy' 0 0 "8080" 2>&1 1>&3);

```

```

### Volumes and Mounts

active_volumes=$(docker ps --format "{{.Names}}" | xargs -n1 docker inspect -f '{{json
.Mounts}}' | jq -r '.[ ] | "\(.Source):\(.Destination) \(.Source):\(.Destination) off"' |sort -
u)

filtered_volumes=$(echo "${active_volumes}" | egrep -v 'sock|squid|yaml|config' | sed
's/media off/media on/g'| sed 's/fuse off/fuse on/g')

selected_volumes=$(dialog --backtitle "${BACKGROUND_TITLE} - Add predefined volumes" --
extra-button --extra-label "Add Custom Mounts" --checklist "Select Volumes to Map" 20 100 20
$filtered_volumes 2>&1 1>&3);

dialog_exit_code=$?

case $dialog_exit_code in
0) echo "Add Selected Volumes"
    echo ${selected_volumes}
    add_volume="false"
    ;;
1) echo "Cancel"
    add_volume="false"
    ;;
3) echo "Add Custom Mount"
    add_volume="true"
    ;;
*) echo ""
    add_volume="false"
    ;;
esac

volumes=()

for SEL in "${selected_volumes}"
do
    volumes+=("${SEL}")
done

while [[ "${add_volume}" == "true" ]]
do
    vol=$(dialog --backtitle "${BACKGROUND_TITLE} - Add Custom Mount" --separate-widget
$'\n' --title "Add custom mount" --extra-button --extra-label "Add Another" --form "" 0 0 0
"Source:" 1 1 "$source" 1 10 30 0 "Mount:" 2 1 "$mount" 2 10 30 0 2>&1 1>&3 )

```

```

dialog_exit_code=$?
form=$(echo ${vol}| tr ' ' ':')
case $dialog_exit_code in
    0) echo "Add Single Custom"
        volumes+=" ${form}"
        add_volume="false"
        ;;
    1) echo "Cancel"
        add_volume="false"
        ;;
    3) echo "Add More"
        volumes+=" ${form}"
        add_volume="true"
        ;;
    *) echo ""
        add_volume="false"
        ;;
esac
done

## PRIVILEGED MODE
if dialog --clear --backtitle "${BACKGROUND_TITLE}" --title "Privileged Mode" --yesno
"Enable Privileged Mode?" 0 0 ;then priv=true;else priv=false;fi

## DNS OVERRIDE
dns=$(dialog --backtitle "${BACKGROUND_TITLE}" --inputbox 'Override Primary DNS' 0 0
"8.8.8.8" 2>&1 1>&3);

## RESOURCE LIMITING
cpus=$(dialog --backtitle "${BACKGROUND_TITLE}" --inputbox 'Resource Allocate CPUs' 0 0
"1" 2>&1 1>&3);
memory=$(dialog --backtitle "${BACKGROUND_TITLE}" --inputbox 'Resource Allocate Memory' 0
0 "200" 2>&1 1>&3);

## TimeZone
timezone=$(dialog --backtitle "${BACKGROUND_TITLE}" --inputbox 'Select Timezone' 0 0 $tz
2>&1 1>&3);

exec 3>&-;

```

```
rm -f "$tempfile"

# Display results to confirm input
dialog --backtitle "${BACKGROUND_TITLE} - Configuration Summary" --title "Confirmation" --
msgbox "\
    Service Name: $service_name\n\
    Image Name: $image_name\n\
    Top Level Domain: $top_level_domain\n\
    Config Folder: $config_folder\n\
    Network: $docker_network\n\
    User: $choose_user\n\
    Group: $select_group\n\
    Privileged: $priv\n\
    Vhost: "${service_name}.pknw1.co.uk"\n\
    Port: $app_port\n\
    DNS: $dns\n\
    CPU: $cpus\n\
    Memory: $memory\n\
    TimeZone: $timezone\n\
    Volumes: $volumes\n\
    Timezone: $timezone" 20 70
clear
```

sdfsd

- ☐ script info
- ☐ check mark

## **further info**

<b><u>Product Home</u></b>	<a href="#">Link</a>
----------------------------	----------------------

<b><u>Documentation</u></b>	<a href="#">Link</a>
<b><u>Github</u></b>	<a href="#">Link</a>
<b><u>DockerHub</u></b>	<a href="#">Link</a>
<b><u>Misc</u></b>	<a href="#">Link</a>

more text more text