

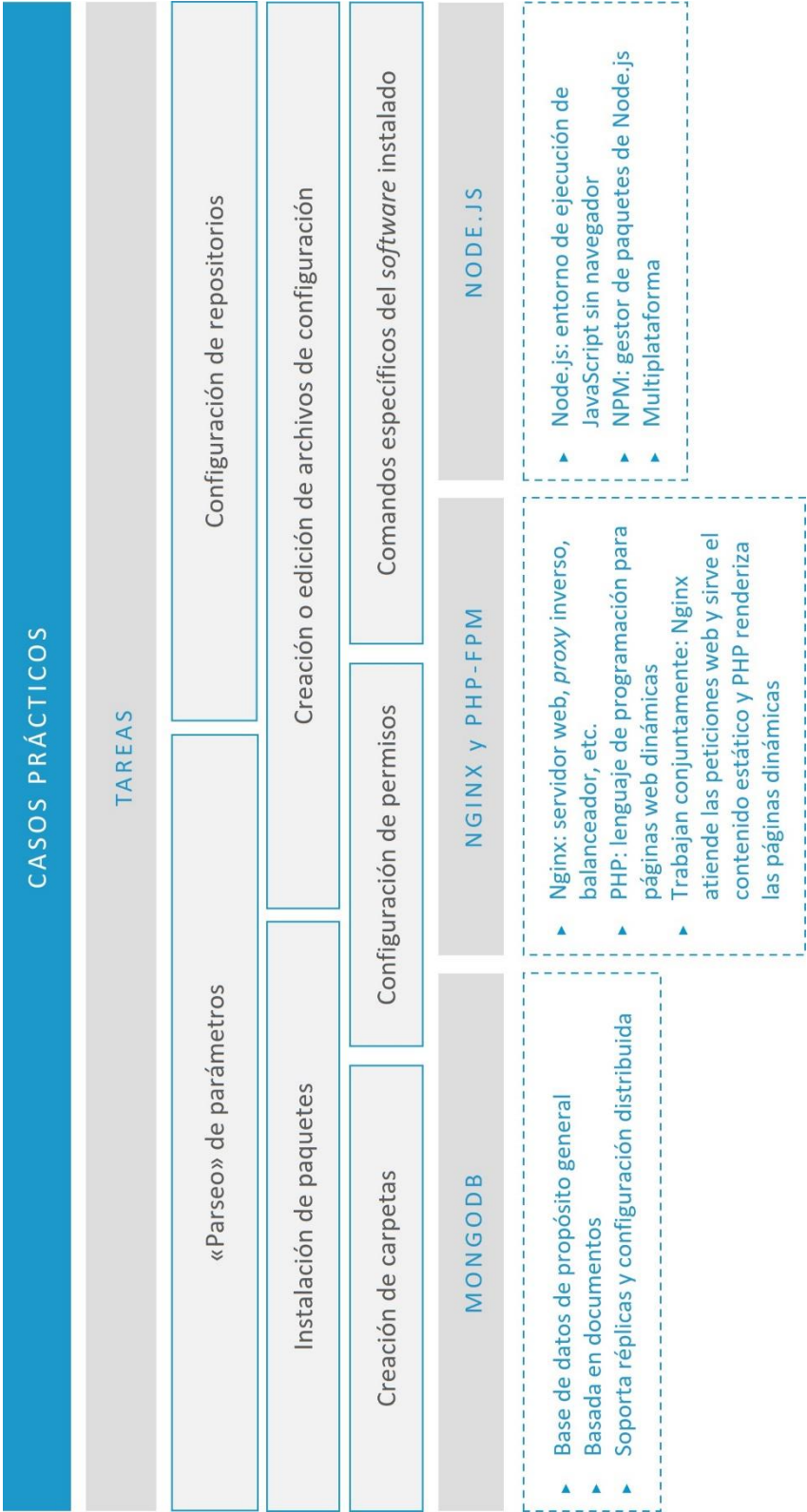
Administración de Sistemas en la Cloud

---

# Automatización de instalación y configuración en Linux

# Índice

Esquema	3
Ideas clave	4
4.1. Introducción y objetivos	4
4.2. Caso práctico: instalación de MongoDB	4
4.3. Caso práctico: instalación de PHP-FPM	16
4.4. Caso práctico: aplicación genérica de NodeJS	25
4.5. Caso práctico: automatización en AWS	31
4.6. Referencias bibliográficas	36
A fondo	39
Test	40



Esquema

## 4.1. Introducción y objetivos

Los conceptos de Bash son la primera piedra en el objetivo de automatizar las tareas administrativas en un servidor Linux. Cada comando y cada utilidad no hacen mucho por sí solos, pero, una vez combinados en un *script*, es posible automatizar cualquier instalación, configuración y despliegue necesario.

En este tema, se presentan ejemplos de *scripts* de instalación de **tres aplicaciones de servidor**: la base de datos MongoDB, el *framework* de páginas web dinámicas PHP-FPM y una aplicación genérica de NodeJS. En todos los casos, se analizará el *script* paso a paso. Un cuarto caso práctico demuestra cómo integrar un *script* de instalación en un despliegue automático de una instancia de la nube.

Los **objetivos** que se pretenden conseguir son:

- ▶ Tomar contacto con *scripts* complejos.
- ▶ Aprender a unir comandos de Bash y utilidades de línea de comandos para construir *scripts* avanzados.
- ▶ Familiarizarse con la documentación de aplicaciones de servidor.

## 4.2. Caso práctico: instalación de MongoDB

MongoDB es una base de datos de propósito general basada en [documentos](#). Soporta alta disponibilidad gracias a su funcionalidad de réplicas y consultas distribuidas en las instalaciones en modo *shard*. La instalación de un servidor *standalone* (en el sentido de que no estará conectado a otros servidores para replicar datos) es

relativamente sencilla y no hace falta más que seguir los pasos indicados en la documentación de [MongoDB](#). MongoDB ofrece una edición [Community](#) sin coste (y, por tanto, también sin soporte más allá de los foros).

A continuación, en el vídeo *Paso a paso de instalación MongoDB en Linux*, se verá una explicación y ejecución paso a paso del *script* de instalación y configuración de MongoDB en Linux.



Accede al vídeo

### ***Script***

Este *script* se basa en la documentación oficial de MongoDB. Usa diversas utilidades GNU y funciones de Bash para automatizar lo más posible la instalación. Tras el *script*, se detallan los apartados en los que está organizado.

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

```
set -e
```

```
logger "Arrancando instalacion y configuracion de MongoDB"
```

```
USO="Uso : install.sh [opciones]"
```

```
Ejemplo:
```

```
install.sh -u administrador -p password [-n 27017]
```

```
    Opciones:
```

```
        -u usuario
```

```
        -p password
```

```
        -n numero de puerto (opcional)
```

```
        -a muestra esta ayuda
```

```
"
```

```
function ayuda() {
```

```
    echo "${USO}"
```

```
    if [[ ${1} ]]
```

```

then
    echo ${1}
fi
}

# Gestionar los argumentos
while getopts ":u:p:n:a" OPCION
do
    case ${OPCION} in
        u )  USUARIO=$OPTARG
              echo "Parametro USUARIO establecido con '${USUARIO}'";;
        p )  PASSWORD=$OPTARG
              echo "Parametro PASSWORD establecido";;
        n )  PUERTO_MONGOD=$OPTARG
              echo "Parametro PUERTO_MONGOD establecido con '${PUERTO_MONGOD}'";;
        a ) ayuda; exit 0;;
        : ) ayuda "Falta el parametro para -$OPTARG"; exit 1;; \?) ayuda "La
opcion no existe : $OPTARG"; exit 1;;
        esac
    done

    if [ -z ${USUARIO} ]
    then
        ayuda "El usuario (-u) debe ser especificado"; exit 1
    fi

    if [ -z ${PASSWORD} ]
    then
        ayuda "La password (-p) debe ser especificada"; exit 1
    fi

    if [ -z ${PUERTO_MONGOD} ]
    then
        PUERTO_MONGOD=27017
    fi

    # Preparar el repositorio (apt-get) de mongodb añadir su clave apt
    apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv
    4B7C549A058F8B6B

```

```

echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu
xenial/mongodb-org/4.2 multiverse" | tee
/etc/apt/sources.list.d/mongodb.list

if [[ -z "$(mongo --version 2> /dev/null | grep '4.2.1')" ]]
then
    # Instalar paquetes comunes, servidor, shell, balanceador de shards y
herramientas
    apt-get -y update \
    && apt-get install -y \
        mongodb-org=4.2.1 \
        mongodb-org-server=4.2.1 \
        mongodb-org-shell=4.2.1 \
        mongodb-org-mongos=4.2.1 \
        mongodb-org-tools=4.2.1 \
    && rm -rf /var/lib/apt/lists/* \
    && pkill -u mongodb || true \
    && pkill -f mongod || true \
    && rm -rf /var/lib/mongodb
fi

# Crear las carpetas de logs y datos con sus permisos
[[ -d "/datos/bd" ]] || mkdir -p -m 755 "/datos/bd"
[[ -d "/datos/log" ]] || mkdir -p -m 755 "/datos/log"

# Establecer el dueño y el grupo de las carpetas db y log
chown mongodb /datos/log /datos/bd
chgrp mongodb /datos/log /datos/bd

# Crear el archivo de configuración de mongodb con el puerto solicitado
mv /etc/mongod.conf /etc/mongod.conf.orig
(
cat <<MONGODB_CONF
# /etc/mongod.conf
systemLog:
    destination: file
    path: /datos/log/mongod.log
    logAppend: true
storage:
    dbPath: /datos/bd

```

```
engine: wiredTiger
journal:
  enabled: true
net:
  port: ${PUERTO_MONGOD}
security:
  authorization: enabled
MONGOD_CONF
) > /etc/mongod.conf
```

```
# Reiniciar el servicio de mongod para aplicar la nueva configuracion
systemctl restart mongod
```

```
logger "Esperando a que mongod responda..."
sleep 15
```

```
# Crear usuario con la password proporcionada como parametro
```

```
mongo admin << CREACION_DE_USUARIO
```

```
db.createUser({
  user: "${USUARIO}",
  pwd: "${PASSWORD}",
  roles:[{
    role: "root",
    db: "admin"
  },{
    role: "restore",
    db: "admin"
  }] })
```

```
CREACION_DE_USUARIO
```

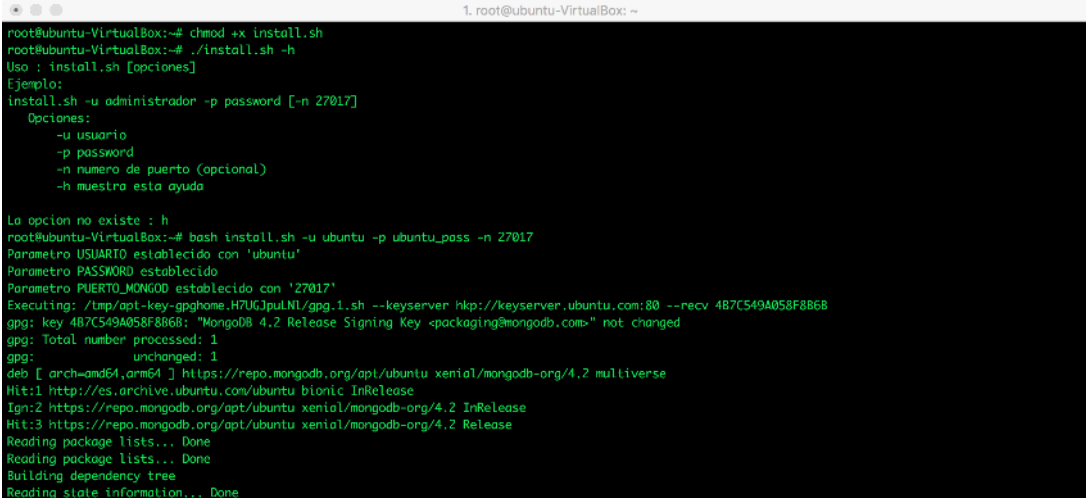
```
logger "El usuario ${USUARIO} ha sido creado con exito!"
```

```
exit 0
```



## Ejecución

Tal como indican los comentarios al principio del *script*, los parámetros «usuario» y «password» son obligatorios, mientras que el «puerto» es opcional. En la Figura 1, se activa el *flag* `x` del *script*, para poder ejecutarlo directamente, y se invoca la ayuda. A continuación, se ejecuta con Bash (sería equivalente invocarlo directamente) con los parámetros de instalación: `bash install.sh -u ubuntu -p ubuntu_pass -n 27017`.



```
1. root@ubuntu-VirtualBox: ~
root@ubuntu-VirtualBox:~# chmod +x install.sh
root@ubuntu-VirtualBox:~# ./install.sh -h
Usa : install.sh [opciones]
Ejemplo:
install.sh -u administrador -p password [-n 27017]
Opciones:
  -u usuario
  -p password
  -n numero de puerto (opcional)
  -h muestra esta ayuda

La opcion no existe : h
root@ubuntu-VirtualBox:~# bash install.sh -u ubuntu -p ubuntu_pass -n 27017
Parametro USUARIO establecido con 'ubuntu'
Parametro PASSWORD establecido
Parametro PUERTO_MONGODB establecido con '27017'
Executing: /tmp/apt-key-gpg/home.h7UGJpuNL/gpg.1.sh --keyserver hkp://keyserver.ubuntu.com:80 --recv 4B7C549A058F8868
gpg: key 4B7C549A058F8868: "MongoDB 4.2 Release Signing Key <packaging@mongodb.com>" not changed
gpg: Total number processed: 1
gpg:      unchanged: 1
deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/4.2 multiverse
Hit:1 http://es.archive.ubuntu.com/ubuntu bionic InRelease
Ign:2 https://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/4.2 InRelease
Hit:3 https://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/4.2 Release
Reading package lists... Done
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

Figura 1. Ejecución del *script* de instalación de MongoDB. Fuente: elaboración propia.

## Shebang

El *script* comienza con el *shebang* para poder ejecutarlo directamente como un *script* de Bash.

```
#!/bin/bash
set -e
```

A continuación, se hace uso del **comando** `set` de Bash, que modifica comportamientos de la consola. Concretamente, `set -e` saldrá del *script* si cualquiera de los comandos falla. Este *script* es muy sencillo y las únicas situaciones de error que comprueba son los parámetros incorrectos. Si uno de los comandos falla, es

preferible no seguir adelante, ya que se ejecutarían comandos que asumen que las líneas anteriores han terminado con éxito.

### Función de ayuda

El texto de ayuda se imprime con una función, en vez de con un simple `echo`, para poder imprimir un mensaje adicional si se ejecuta con un parámetro. Así, si se ejecuta `./install -h`, solo se imprime la ayuda, pero, si se introduce un parámetro que no existe, se imprime la ayuda seguida de un mensaje de error específico.

```
function ayuda() {  
    echo "${USO}"  
    if [[ ${1} ]]  
    then  
        echo ${1}  
    fi  
}
```

### Captura de parámetros

El *script* no es una pieza de código fija que instala el *software* siempre de la misma manera. Los parámetros permiten personalizar la ejecución de un *script* para que sea útil en múltiples situaciones, sin necesidad de alterar el código. Al fin y al cabo, el *script* puede ser usado por administradores sin conocimientos avanzados de Bash y, aun así, ser igualmente útil.

Los parámetros están normalmente disponibles en las variables `$1`, `$2`, `$3`, etc. o en el *array* `$@`. El código para «parsear» los parámetros a partir de estas variables sería muy farragoso y esta tarea es tan habitual que hay utilidades que lo facilitan mucho. Este *script* usa el comando [getopts](#) de Bash: para cada elemento de `":u:p:n:h"`, `getopts` guarda el valor del parámetro en la variable `OPTION`. Entonces, la expresión `case` facilita la ejecución de unas líneas de código para cada valor de `OPTION`. Es equivalente a una estructura `if-else`, pero más fácil de leer.

```

while getopts ":u:p:n:h" OPCION
do
    case ${OPCION} in
        u )  USUARIO=$OPTARG
              echo "Parametro USUARIO establecido con '${USUARIO}'";;
        p )  PASSWORD=$OPTARG
              echo "Parametro PASSWORD establecido";;
        n )  PUERTO_MONGOD=$OPTARG
              echo "Parametro PUERTO_MONGOD establecido con '${PUERTO_MONGOD}'";;
        h )  ayuda; exit 0;;
        : )  ayuda "Falta el parametro para -$OPTARG"; exit 1;;
        \?)  ayuda "La opcion no existe : $OPTARG"; exit 1;;
    esac
done

```

Al terminar este bloque, los parámetros del *script* están disponibles en **variables** de Bash. Esto abre la puerta a simplificar el *script*, de manera que espere los parámetros directamente como variables de entorno. Este método obliga al usuario a definir las variables antes de invocar el *script*, pero puede ser útil según los requisitos del *script*, por ejemplo, un *script* principal puede invocar un segundo *script*: el primer *script* leería los parámetros y definiría las variables de entorno y el segundo no tendría más que leer las variables definidas del primero.

## Validación

Siempre es necesario validar los datos de entrada, tanto si el *script* acepta parámetros como si espera los valores de entorno. La siguiente sección hace lo siguiente:

- ▶ Si el usuario o la contraseña no están definidos, imprime un mensaje de error y sale con un código de salida diferente de 0.
- ▶ Si el puerto no está definido, lo fija a un valor por defecto de 27017.

```

if [ -z ${USUARIO} ]
then
    ayuda "El usuario (-u) debe ser especificado"; exit 1

```

```

fi

if [ -z ${PASSWORD} ]
then
    ayuda "La password (-p) debe ser especificada"; exit 1
fi

if [ -z ${PUERTO_MONGOD} ]
then
    PUERTO_MONGOD=27017
fi

```

El comando `[` es un alias del comando `test`. Esta utilidad se usa para comprobar diferentes situaciones: si existe un fichero, si una variable de entorno que está definida, etc. De hecho, el *script* usa el *flag* `-z`, que comprueba si una cadena de texto tiene longitud cero (Robbins, 2010).

## Repositorios e instalación

El siguiente bloque añade el repositorio de apt oficial de MongoDB para poder instalar la **última versión disponible** (4.2 en el momento de escribir esta sección). Las distribuciones suelen incorporar versiones anteriores (3.6 en el caso de [Ubuntu 18.04](#)).

```

# Preparar el repositorio (apt-get) de mongodb añadir su clave apt
apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv
4B7C549A058F8B6B
echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu
xenial/mongodb-org/4.2 multiverse" | tee
/etc/apt/sources.list.d/mongodb.list

```

A continuación, se instalan el servidor y las herramientas de línea de comandos, **solo si el servidor no está ya instalado**. La instalación es un proceso largo, así que es útil evitarlo si el *software* ya está instalado. Además, esta condición hace que el *script* sea **idempotente**. La idempotencia es una cualidad deseable en un *script*, por su

capacidad de recuperación automática: si el sistema falla durante la ejecución de un *script* idempotente por causas ajenas al *script*, una segunda ejecución puede llegar a finalizarse correctamente y, además, evitará ejecutar de nuevo procesos que han finalizado en la primera ejecución.

```
if [[ -z "$(mongo --version 2> /dev/null | grep '4.2.1')" ]]
then
    # Instalar paquetes comunes, servidor, shell, balanceador de shards y
herramientas
    apt-get -y update \
    && apt-get install -y \
        mongodb-org=4.2.1 \
        mongodb-org-server=4.2.1 \
        mongodb-org-shell=4.2.1 \
        mongodb-org-mongos=4.2.1 \
        mongodb-org-tools=4.2.1 \
    && rm -rf /var/lib/apt/lists/* \
    && pkill -u mongodb || true \
    && pkill -f mongod || true \
    && rm -rf /var/lib/mongodb
fi
```

## Configuración

El siguiente paso consiste en configurar los directorios que alojarán los ficheros de base de datos y de log. De nuevo, se hace uso del comando `test` para no crear las carpetas, si ya existen.

```
# Crear las carpetas de logs y datos con sus permisos
[[ -d "/datos/bd" ]] || mkdir -p -m 755 "/datos/bd"
[[ -d "/datos/log" ]] || mkdir -p -m 755 "/datos/log"

# Establecer el dueño y el grupo de las carpetas db y log
chown mongodb /datos/log /datos/bd
chgrp mongodb /datos/log /datos/bd
```

La configuración del servidor de MongoDB, al igual que muchos otros paquetes de *software*, consiste en un [fichero de configuración](#). El *script* incluye un [heredoc](#), es decir, el contenido del fichero de configuración, que puede contener un número arbitrario de líneas, está embebido en el propio *script*. El identificador MONGODB\_CONF delimita el contenido del fichero y el comando cat se encarga de escribirlo en la ruta de destino. El contenido del fichero inserta valores de variables de entorno con \${PUERTO\_MONGODB}.

```
(
cat <<MONGODB_CONF
# /etc/mongod.conf
systemLog:
  destination: file
  path: /datos/log/mongod.log
  logAppend: true
storage:
  dbPath: /datos/bd
  engine: wiredTiger
  journal:
    enabled: true
net:
  port: ${PUERTO_MONGODB}
security:
  authorization: enabled
MONGODB_CONF
) > /etc/mongod.conf
```

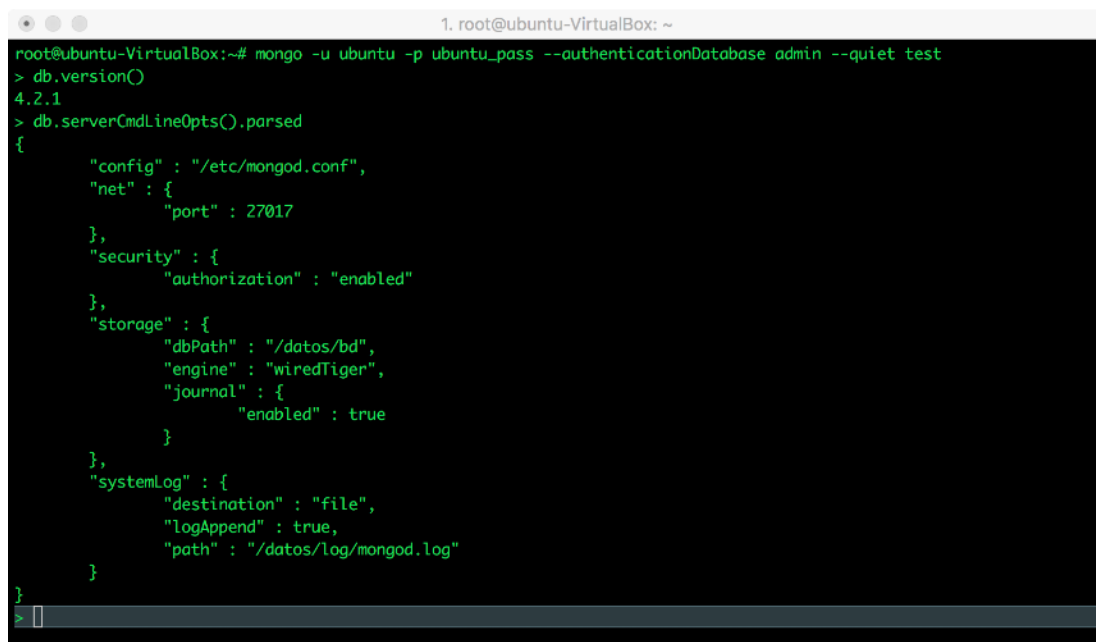
### Creación del usuario por defecto

Una instalación nueva no incluye ningún usuario, así que el *script* se encarga de crear uno a partir de los parámetros proporcionados por el usuario. Para esto, se hace uso de la consola de MongoDB, mongo. Aparte de ofrecer una línea de comandos interactiva, mongo acepta comandos nativos. Para esto, se usa otro HEREDOC, que en este caso se inyecta en el comando mongo a través de la entrada estándar.

```
# Crear usuario con la password proporcionada como parametro
mongo admin << CREACION_DE_USUARIO
db.createUser({
  user: "${USUARIO}",
  pwd: "${PASSWORD}",
  roles:[{
    role: "root",
    db: "admin"
  },{
    role: "restore",
    db: "admin"
  ] })
CREACION_DE_USUARIO
```

## Comprobación

Si todo ha ido bien, el *script* ejecuta `exit 0` para notificar que ha terminado con éxito. La manera de comprobar que el servidor esta correctamente configurado sería iniciar sesión con los datos del usuario, tal como muestra la Figura 2.



```
1. root@ubuntu-VirtualBox: ~
root@ubuntu-VirtualBox:~# mongo -u ubuntu -p ubuntu_pass --authenticationDatabase admin --quiet test
> db.version()
4.2.1
> db.serverCmdLineOpts().parsed
{
  "config" : "/etc/mongod.conf",
  "net" : {
    "port" : 27017
  },
  "security" : {
    "authorization" : "enabled"
  },
  "storage" : {
    "dbPath" : "/datos/bd",
    "engine" : "wiredTiger",
    "journal" : {
      "enabled" : true
    }
  },
  "systemLog" : {
    "destination" : "file",
    "logAppend" : true,
    "path" : "/datos/log/mongod.log"
  }
}
```

Figura 2. Inicio de sesión en MongoDB tras la ejecución del *script*. Fuente: elaboración propia.

## 4.3. Caso práctico: instalación de PHP-FPM

Este segundo *script* de ejemplo instalará un servidor PHP sobre Nginx. **PHP** es un lenguaje interpretado muy extendido en el desarrollo de aplicaciones web dinámicas. Algunos de los administradores de contenidos basados en PHP son Wordpress, Drupal y Moodle. Se puede instalar como un servidor web propio o junto a un servidor web específico, como Apache, IIS o [Nginx](#).

**Nginx** es un servidor web que también puede usarse como *proxy* inverso, balanceador de nivel 5 y caché de HTTP (*hypertext transfer protocol*) (Nginx, s. f.). Permite servir, por ejemplo, contenido estático por sí mismo y contenido dinámico mediante una integración con otro *software*. Nginx Inc., la compañía detrás de Nginx, ofrece una versión de Nginx de pago con más funcionalidad. Este *script* instalará la versión gratuita.

El *script* de PHP es similar en estructura al *script* de instalación de MongoDB, pero incluye algunas novedades, como la ejecución condicionada a la plataforma y la edición de fichero de configuración al vuelo.

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

```
set -e
```

```
logger "Arrancando instalacion y configuracion de PHP-FPM"
```

```
USO="Uso : install.sh [opciones]"
```

```
Ejemplo:
```

```
install.sh -s mysite.com
```

```
Opciones:
```

```
-s nombre del site en nginx
```

```
-a muestra esta ayuda
```

```
"
```

```
function ayuda() {
```



```

echo "${USO}"
if [[ ${1} ]]
then
    echo ${1}
fi
}

# Gestionar las opciones
while getopts ":u:g:l:a" OPCION
do
    case ${OPCION} in
        s ) NGINX_SITE=$OPTARG
            logger "Parametro NGINX_SITE establecido con '${NGINX_SITE}'"
;;
        a ) ayuda; exit 0;;
        : ) ayuda "Falta el parámetro para -$OPTARG"; exit 1;;
        \?) ayuda "La opción no existe : $OPTARG"; exit 1;;
    esac
done

if [ -z ${NGINX_SITE} ]
then
    NGINX_SITE="php.local"
fi

# Instalar PHP-FPM usando apt-get o yum dependiendo de la plataforma
if [[ -e /etc/redhat-release || -e /etc/system-release ]]
then

    # Solo se soporta la instalacion en la release 7 de RedHat y CentOS
    OS=$(rpm -q --whatprovides redhat-release | cut -d"- " -f1)
    RELEASE=$(rpm -q --whatprovides redhat-release | cut -d"- " -f3)

    if [[ "${OS}" != "centos" && "${OS}" != "redhat" ]]
    then
        logger "La instalacion solo esta soportada en RedHat y CentOS"
        exit 2
    fi

    if [[ "${RELEASE}" != "7" ]]

```

```

then
    logger "La instalacion solo esta soportada en CentOS y RedHat release
7"
    exit 3
fi

# Configuracion de SELinux
setenforce permissive

# Preparacion de yum con los repositorios de nginx y php-fpm
(
cat << NGINX_REPO
[nginx]
name=nginx repo
baseurl=http://nginx.org/packages/mainline/rhel/7/\$basearch/
gpgcheck=0
enabled=1
NGINX_REPO
) > /etc/yum.repos.d/nginx.repo

yum install -y epel-release
yum install -y http://rpms.remirepo.net/enterprise/remi-release-7.rpm
yum install -y yum-utils
yum-config-manager --enable remi-php72
yum update -y

# Instalacion y configuracion de nginx
yum install -y nginx

(
cat << NGINX_CONF
server {
    listen      80;
    root /var/www/html;
    index index.php index.html index.htm;
    server_name $NGINX_SITE;

    location / {
        try_files \$uri \$uri/ =404;
    }
}

```

```

        location ~ \.php$ {
            fastcgi_pass    unix:/var/run/php/php7.2-fpm.sock;
            fastcgi_index   index.php;
            fastcgi_param   SCRIPT_FILENAME
\${document_root}\${fastcgi_script_name};
            include         fastcgi_params;
        }

    }
}
NGINX_CONF
) > /etc/nginx/conf.d/$NGINX_SITE.conf

rm /etc/nginx/conf.d/default.conf

systemctl start nginx

# Instalacion y configuracion de php-fpm
yum install -y php72 php72-php-fpm php72-php-gd \
    php72-php-json php72-php-mbstring php72-php-mysqlnd \
    php72-php-xml php72-php-xmlrpc php72-php-opcache

mkdir /var/run/php
chown nginx:nginx /var/run/php/

sed -i "s/user    =.*/user    =    nginx/g" /etc/opt/remi/php72/php-
fpm.d/www.conf
sed -i "s/group    =.*/group    =    nginx/g" /etc/opt/remi/php72/php-
fpm.d/www.conf
sed -i "s/listen    =.*/listen    =    \\/var\\/run\\/php\\/php7.2-fpm.sock/g"
/etc/opt/remi/php72/php-fpm.d/www.conf
sed -i "s/.*/listen.owner    =.*/listen.owner    =    nginx/g"
/etc/opt/remi/php72/php-fpm.d/www.conf
sed -i "s/.*/listen.group    =.*/listen.group    =    nginx/g"
/etc/opt/remi/php72/php-fpm.d/www.conf
sed -i "s/.*/listen.mode    =.*/listen.mode    =    0660/g"
/etc/opt/remi/php72/php-fpm.d/www.conf

systemctl enable php72-php-fpm.service
systemctl start php72-php-fpm.service
# Limpieza de archivos temporales tras la instalacion

```

```

yum clean all

else

    # configuracion de repositorios
    export DEBIAN_FRONTEND=noninteractive
    apt-get install software-properties-common
    add-apt-repository -y ppa:ondrej/nginx-mainline
    apt-get install -y nginx php7.2 php7.2-fpm

    # Crear el archivo del site de nginx
(
cat << NGINX_CONF
server {
    listen 80;
    root /var/www/html;
    index index.php index.html index.htm;
    server_name $NGINX_SITE;

    location / {
        try_files $uri $uri/ =404;
    }

    location ~ \.php$ {
        include snippets/fastcgi-php.conf;
        fastcgi_pass unix:/var/run/php/php7.2-fpm.sock;
    }
}
NGINX_CONF
) > /etc/nginx/sites-available/$NGINX_SITE

rm /etc/nginx/sites-enabled/default
sudo ln -s /etc/nginx/sites-available/$NGINX_SITE /etc/nginx/sites-
enabled/$NGINX_SITE

systemctl restart nginx

# La configuracion por defecto de PHP-FPM es suficiente
fi

```

```
# Creacion de un fichero de prueba en la raiz
# Sera accesible tras la instalacion en http://<ip_del_servidor>/info.php
mkdir -p /var/www/html
echo "<?php phpinfo(); ?>" > /var/www/html/info.php
id -u nginx && chown -R nginx:nginx /var/www || chown -R www-data:www-data
/var/www
chmod -R 755 /var/www

logger "Instalacion completada con exito; visite http://<ip>/info.php para
comprobarlo"

exit 0
```

## Detección de plataforma

Aunque el núcleo de Linux pueda ser el mismo, las diferentes distribuciones en la industria siguen diferentes enfoques en cuanto a la instalación de paquetes, configuración del sistema, etc. Incluso dentro de una misma distribución, dos versiones pueden diferir tanto como dos distribuciones diferentes. Por tanto, es habitual que los *scripts* de instalación sean capaces de **detectar en qué plataforma se están ejecutando**. El código puede estar preparado para ejecutarse en diferentes plataformas o terminar con error si se ejecuta en una plataforma para la que no está preparada.

Este *script* incluye ejemplos de ambos modelos: por un lado, detecta si se está ejecutando en una distribución basada en RedHat. En tal caso, termina con error si no se trata de un RedHat o CentOS versión 7. Si no es una distribución basada en RedHat, el *script* asume que se está ejecutando en una distribución Ubuntu 18.

```
# Instalar PHP-FPM usando apt-get o yum dependiendo de la plataforma
if [[ -e /etc/redhat-release || -e /etc/system-release ]]
then

    # Solo se soporta la instalacion en la release 7 de RedHat y CentOS
    OS=$(rpm -q --whatprovides redhat-release | cut -d"- " -f1)
```

```

RELEASE=$(rpm -q --whatprovides redhat-release | cut -d"-" -f3)

if [[ "${OS}" != "centos" && "${OS}" != "redhat" ]]
then
    logger "La instalacion solo esta soportada en RedHat y CentOS"
    exit 2
fi

if [[ "${RELEASE}" != "7" ]]
then
    logger "La instalacion solo esta soportada en CentOS y RedHat release
7"
    exit 3
fi

# ... prosigue la instalación en RedHat y CentOS

else

    # ... prosigue la instalación en Ubuntu

fi

```

## Instalación en CentOS

El **esquema de la instalación** en CentOS no difiere mucho de la instalación en Ubuntu: configurar repositorios, instalación de paquetes, configurar y arrancar servicios. La principal diferencia es que el gestor de paquetes es yum en vez de apt-get. CentOS incluye una configuración por defecto de [SELinux](#) demasiado restrictiva para el funcionamiento de PHP en conjunción con Nginx, así que hay que incluir el comando `setenforce permissive`.

Finalmente, aunque las versiones de Nginx y PHP-FPM son las mismas, los paquetes de instalación de cada plataforma tienen configuraciones por defecto diferentes. Por ejemplo, la configuración de PHP-FPM en Ubuntu no requiere modificaciones, pero la configuración en CentOS requiere reconfigurar el *socket* Unix y los usuarios con los

que se ejecuta el proceso. En vez de escribir el fichero de configuración entero, el comando [sed](#) edita el fichero al vuelo, editando solo las líneas necesarias.

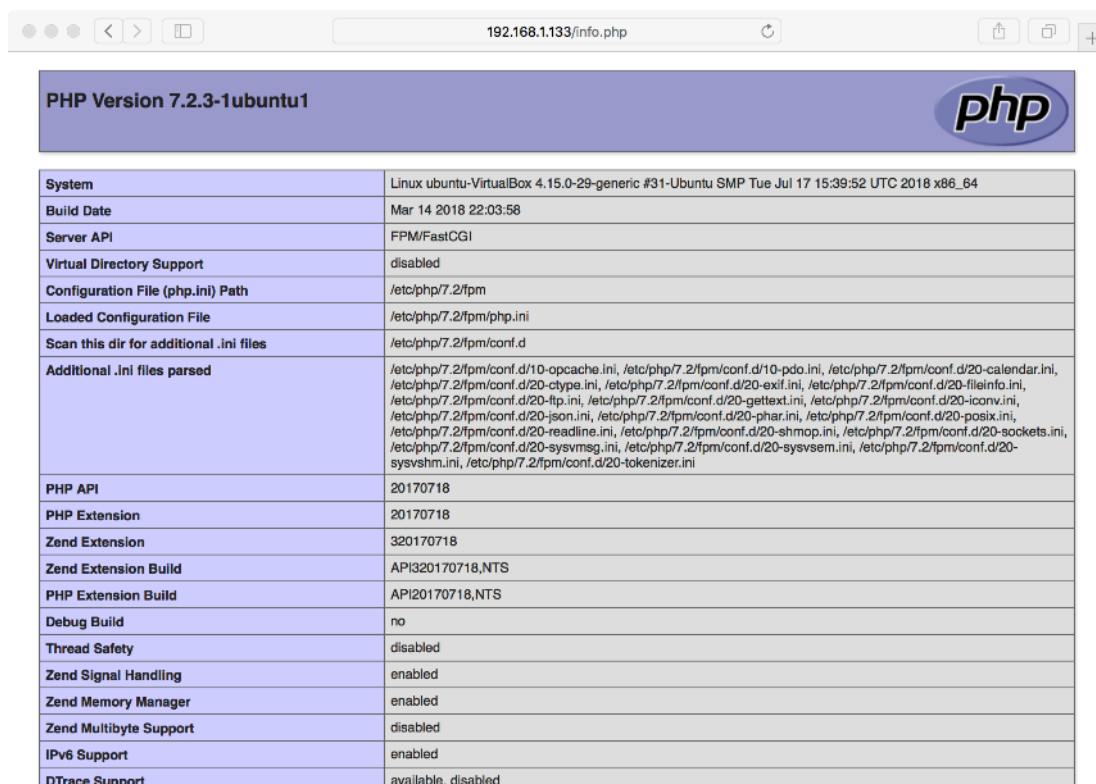
```
sed -i "s/user =.*/user = nginx/g" /etc/opt/remi/php72/php-fpm.d/www.conf
sed -i "s/group =.*/group = nginx/g" /etc/opt/remi/php72/php-fpm.d/www.conf
sed -i "s/listen =.*/listen = \\/var\\/run\\/php\\/php7.2-fpm.sock/g" /etc/opt/remi/php72/php-fpm.d/www.conf
sed -i "s/*.listen.owner =.*/listen.owner = nginx/g" /etc/opt/remi/php72/php-fpm.d/www.conf
sed -i "s/*.listen.group =.*/listen.group = nginx/g" /etc/opt/remi/php72/php-fpm.d/www.conf
sed -i "s/*.listen.mode =.*/listen.mode = 0660/g" /etc/opt/remi/php72/php-fpm.d/www.conf
```

El proceso Nginx se ejecuta con el usuario `nginx` por defecto en CentOS, pero con el usuario `www-data` en Ubuntu. Ya que la creación del fichero de ejemplo está fuera de la condición en función de la plataforma, se usa una expresión `if-else` en la misma línea: si el comando `id -u nginx` termina satisfactoriamente, se ejecuta el comando a continuación de `&&`. Si termina en error, se ejecuta el comando a continuación de `||`. El comando `id` devuelve el identificador del nombre de usuario y falla si el usuario no existe.

```
id -u nginx && chown -R nginx:nginx /var/www || chown -R www-data:www-data /var/www
```

## Comprobación

El fichero `info.php`, creado por el *script*, contiene la función `phpinfo()` de PHP, que imprime una larga lista de opciones de instalación y módulos en ejecución. Si se recupera el fichero con un navegador y se obtiene un fichero de texto plano con el contenido `<?php phpinfo(); ?>`, la instalación no ha funcionado. Sin embargo, si se obtiene una página como la de la Figura 3, la instalación terminó con éxito.




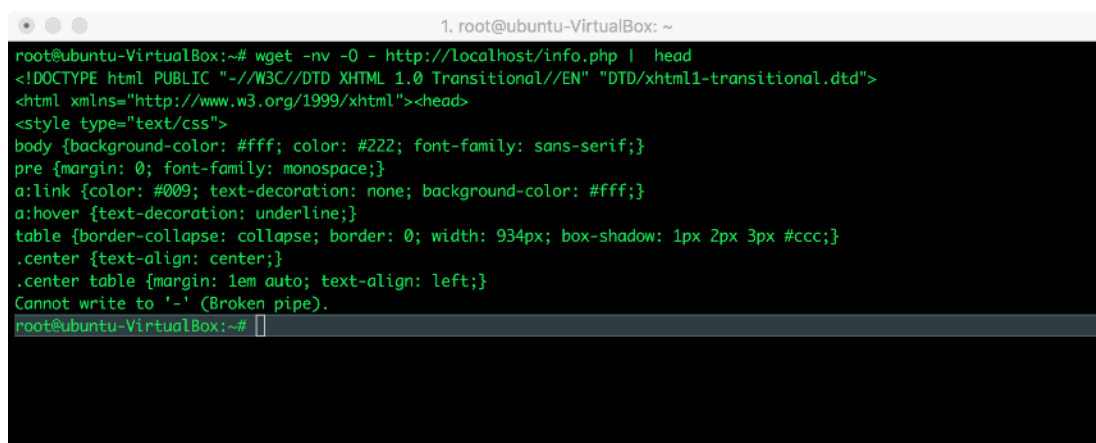
<div> <div>PHP Version 7.2.3-1ubuntu1</div>  </div>	
System	Linux ubuntu-VirtualBox 4.15.0-29-generic #31-Ubuntu SMP Tue Jul 17 15:39:52 UTC 2018 x86_64
Build Date	Mar 14 2018 22:03:58
Server API	FPM/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.2/fpm
Loaded Configuration File	/etc/php/7.2/fpm/php.ini
Scan this dir for additional .ini files	/etc/php/7.2/fpm/conf.d
Additional .ini files parsed	/etc/php/7.2/fpm/conf.d/10-opcache.ini, /etc/php/7.2/fpm/conf.d/10-pdo.ini, /etc/php/7.2/fpm/conf.d/20-calendar.ini, /etc/php/7.2/fpm/conf.d/20-ctype.ini, /etc/php/7.2/fpm/conf.d/20-exif.ini, /etc/php/7.2/fpm/conf.d/20-fileinfo.ini, /etc/php/7.2/fpm/conf.d/20-ftp.ini, /etc/php/7.2/fpm/conf.d/20-gettext.ini, /etc/php/7.2/fpm/conf.d/20-iconv.ini, /etc/php/7.2/fpm/conf.d/20-json.ini, /etc/php/7.2/fpm/conf.d/20-libxml.ini, /etc/php/7.2/fpm/conf.d/20-mbstring.ini, /etc/php/7.2/fpm/conf.d/20-phar.ini, /etc/php/7.2/fpm/conf.d/20-posix.ini, /etc/php/7.2/fpm/conf.d/20-readline.ini, /etc/php/7.2/fpm/conf.d/20-shmop.ini, /etc/php/7.2/fpm/conf.d/20-sockets.ini, /etc/php/7.2/fpm/conf.d/20-sysvmsg.ini, /etc/php/7.2/fpm/conf.d/20-sysvsem.ini, /etc/php/7.2/fpm/conf.d/20-sysvshm.ini, /etc/php/7.2/fpm/conf.d/20-tokenizer.ini
PHP API	20170718
PHP Extension	20170718
Zend Extension	320170718
Zend Extension Build	API320170718,NTS
PHP Extension Build	API20170718,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	enabled
Zend Memory Manager	enabled
Zend Multibyte Support	disabled
IPv6 Support	enabled
DTrace Support	available, disabled

Figura 3. Página de Phpinfo tras la instalación. Fuente: elaboración propia.

También es posible comprobar la instalación localmente en la misma línea de comandos, usando `wget`, como en la **¡Error! No se encuentra el origen de la referencia.** 4.



```

1. root@ubuntu-VirtualBox: ~
root@ubuntu-VirtualBox:~# wget -nv -O - http://localhost/info.php | head
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"><head>
<style type="text/css">
body {background-color: #fff; color: #222; font-family: sans-serif;}
pre {margin: 0; font-family: monospace;}
a:link {color: #009; text-decoration: none; background-color: #fff;}
a:hover {text-decoration: underline;}
table {border-collapse: collapse; border: 0; width: 934px; box-shadow: 1px 2px 3px #ccc;}
.center {text-align: center;}
.center table {margin: 1em auto; text-align: left;}
Cannot write to '-' (Broken pipe).
root@ubuntu-VirtualBox:~#

```

Figura 4. Comando `wget` recuperando la página de Phpinfo. Fuente: elaboración propia.



## 4.4. Caso práctico: aplicación genérica de NodeJS

[NodeJS](#) es un **entorno de ejecución** de JavaScript multiplataforma y de **código libre**. Aunque JavaScript nació como un lenguaje de *scripting* para navegadores web, NodeJS ejecuta el código fuera del navegador. Los desarrolladores pueden usarlo para programar herramientas de línea de comandos y herramientas de servidor. Este último caso es muy habitual: hay *frameworks* como [ExpressJS](#), también conocido como Express, que facilitan el desarrollo de servidores web dinámicos al estilo de PHP.

[NPM](#) (*nearly picked makefiles*) es un gestor de paquetes JavaScript por defecto de NodeJS. Además de una herramienta de línea de comandos pensada para gestionar las dependencias de las aplicaciones de NodeJS, el ecosistema de NPM tiene a su disposición un repositorio público de paquetes, similar en su naturaleza a los repositorios de apt-get o yum. NPM se integra con NodeJS de tal modo que los desarrolladores definen las dependencias de la aplicación en el fichero package.json y NPM las instala y las expone al código local al arrancar la aplicación.

Este *script* instalará no solo NodeJS y NPM, sino que está preparado para descargar el código de una aplicación desde un repositorio de Github.

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

```
AYUDA="Las variables APP_ENTRY_POINT, APP_FOLDER y CLONE_URL deben  
tener valores validos."
```

```
Ejemplo:
```

```
export APP_ENTRY_POINT=bin/www  
export APP_FOLDER=/opt/app  
export CLONE_URL=https://github.com/plesk/node-express  
export BRANCH=master  
"
```

```
# verificacion de parametros
```

```

if [[ -z ${APP_ENTRY_POINT} || -z ${APP_FOLDER} || -z ${CLONE_URL} ]] ;
then
    echo "${AYUDA}"
    exit 1
fi

if [[ -z ${BRANCH} ]]
then
    BRANCH=master
fi

echo "Arrancando instalacion de $APP_ENTRY_POINT@$BRANCH en $APP_FOLDER"

# configuracion para githubb por SSH
mkdir -p ~/.ssh
if [ ! -f ~/.ssh/config ]; then

    cat > ~/.ssh/config << SSH_CONFIG
Host github.com
    StrictHostKeyChecking no
    UserKnownHostsFile=/dev/null
SSH_CONFIG

fi

# instalacion de curl y git en funcion de la distribucion
if [[ -e /etc/redhat-release || -e /etc/system-release ]]; then
    yum -y check-update
    yum install git curl -y
else
    apt-get -y update
    apt-get install git curl -y
fi

# metodo alternativo de deteccion de distribucion
# instalacion de nodeJS y npm
if [[ -x /usr/bin/yum ]]; then
    OS=$(rpm -q --whatprovides redhat-release | cut -d"-" -f1)

    case $OS in

```

```

system)
    yum -y install openssl-devel nodejs npm --enablerepo=epel
    ;;
*)
    yum -y install epel-release
    yum -y install openssl-devel nodejs npm
    ;;
esac
else
    if [[ ! -z $(uname -a | grep -i ubuntu) ]]; then curl -sL
https://deb.nodesource.com/setup_12.x | sudo bash -; fi
    apt-get -y install nodejs
fi

# herramienta para ejecutar una script indefinidamente
# la aplicacion no sera un servicio, pero se comportara como tal
npm install forever -g

# descarga del codigo del repositorio
[[ -z "$APP_FOLDER" ]] && TARGET_DIRECTORY=$(basename "$CLONE_URL" .git) ||
TARGET_DIRECTORY=$APP_FOLDER

if [ ! -d $TARGET_DIRECTORY ]; then
    # clonado inicial
    if [ -z '$BRANCH' ]; then
        git clone $CLONE_URL $APP_FOLDER
    else
        git clone -b $BRANCH $CLONE_URL $TARGET_DIRECTORY
    fi
else
    # actualización de nuevos commits
    cd $TARGET_DIRECTORY
    git remote set-url origin {{ CLONE_URL }}
    git clean -f
    git fetch --all
    git fetch --tags
    if [ -z '$BRANCH' ]; then
        git reset --hard origin
    else
        git reset --hard "origin/$BRANCH"
    fi
fi

```

```

        fi

        git pull origin $BRANCH
    fi

    # instalacion de las dependencias de nodeJS
    pushd $APP_FOLDER
    npm install
    popd

    # parada de los procesos de forever
    PATH="/usr/local/sbin:/usr/local/bin"
    ps aux | grep $APP_FOLDER/$APP_ENTRY_POINT | grep -v grep -q
    if [ $? -eq 0 ]; then
        forever stopall
    fi

    # arranque del script de entrada de la aplicacion
    forever --minUptime 1000 --spinSleepTime 1000 \
        start $APP_FOLDER/$APP_ENTRY_POINT

```

## Parámetros

Al contrario de los otros *scripts*, este no acepta parámetros por la línea de comandos, sino que espera que el usuario los haya fijado como **variables de entorno**. Por tanto, no hay «parseo» de comandos, pero sí comprobación de estos. Uno de los parámetros, `BRANCH`, es opcional y toma el valor por defecto de `master`.

```

# verificacion de parametros
if [[ -z ${APP_ENTRY_POINT} || -z ${APP_FOLDER} || -z ${CLONE_URL} ]]; then
    echo "${AYUDA}"
    exit 1
fi

if [[ -z ${BRANCH} ]]
then
    BRANCH=master

```

```
fi
```

## Instalación multidistribución

El *script* está preparado para instalar la aplicación, tanto en distribuciones RedHat como Ubuntu. En la primera instalación de Git y Curl, el *script* hace uso de una comprobación similar a la vista en los *scripts* anteriores:

```
if [[ -e /etc/redhat-release || -e /etc/system-release ]]; then
```

Más adelante, sin embargo, aprovecha otra diferencia entre las distribuciones: la existencia del fichero `/usr/bin/yum`.

```
if [[ -x /usr/bin/yum ]]; then
```

## Descarga de la aplicación y dependencias

Una vez que NodeJS y NPM están instalados, el resto del *script* no depende de la distribución. En primer lugar, NPM instala el paquete `forever`, que se puede usar para configurar aplicaciones de NodeJS como si fueran servicios.

```
npm install forever -g
```

A continuación, el *script* descarga la aplicación desde el repositorio de Github. Hay dos opciones: si la aplicación no existe, el *script* clona el repositorio de cero, pero, si ya ha sido instalada, actualiza la rama seleccionada al último *commit*.

```
# descarga del código del repositorio
[[ -z "$APP_FOLDER" ]] && TARGET_DIRECTORY=$(basename "$CLONE_URL" .git) ||
TARGET_DIRECTORY=$APP_FOLDER

if [ ! -d $TARGET_DIRECTORY ]; then
    # clonado inicial
    if [ -z '$BRANCH' ]; then
```

```

        git clone $CLONE_URL $APP_FOLDER
    else
        git clone -b $BRANCH $CLONE_URL $TARGET_DIRECTORY
    fi
else
    # actualización de nuevos commits
    cd $TARGET_DIRECTORY
    git remote set-url origin {{ CLONE_URL }}
    git clean -f
    git fetch --all
    git fetch --tags
    if [ -z '$BRANCH' ]; then
        git reset --hard origin
    else
        git reset --hard "origin/$BRANCH"
    fi

    git pull origin $BRANCH
fi

```

La instalación de las dependencias de NodeJS se hace a continuación con NPM. Los comandos pushd y popd sirven para cambiar el directorio de trabajo a uno diferente, para luego volver al original.

```

# instalacion de las dependencias de nodeJS
pushd $APP_FOLDER
npm install
popd

```

Dado que la aplicación puede estar en ejecución, el *script* detiene cualquier ejecución de procesos de forever, siempre y cuando la aplicación esté en ejecución.

```

# parada de los procesos de forever
PATH="$PATH:/usr/local/sbin:/usr/local/bin"
ps aux | grep $APP_FOLDER/$APP_ENTRY_POINT | grep -v grep -q
if [ $? -eq 0 ]; then
    forever stopall
fi

```

Finalmente, el *script* hace uso de *forever* para arrancar la aplicación. En este punto, es posible verificar si la aplicación ha arrancado satisfactoriamente. Por ejemplo, si se instala la aplicación de prueba de [Github](#), la configuración por defecto arranca el servidor HTTP en el puerto 3000, por lo que una simple vista con un navegador puede servir de verificación.

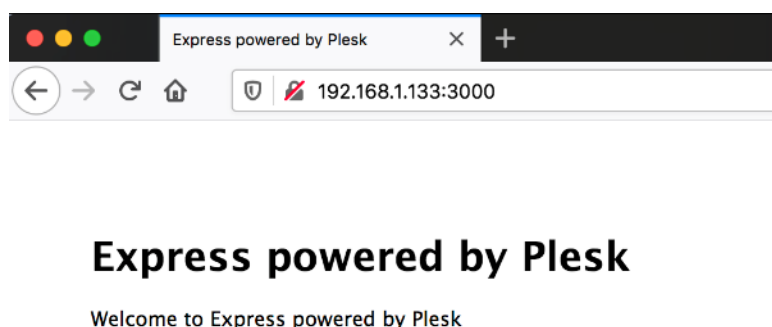


Figura 5. Comprobación de la aplicación NodeJS. Fuente: elaboración propia.

## 4.5. Caso práctico: automatización en AWS

Los *scripts* de los ejemplos anteriores se han demostrado en una ejecución interactiva, pero nada impide que se ejecuten desde una **herramienta de automatización externa**. Por ejemplo, Ansible o Puppet podrían usarlos, aunque en esos casos sería recomendable usar las directivas propias de las herramientas para aprovechar su potencial.

En este caso práctico, se va a hacer uso del *user data* de las instancias de AWS (Amazon Web Services) EC2 (AWS, s. f.). Esta característica es un trozo de código que se puede usar para automatizar tareas y ejecutar *scripts* una vez que arranca la instancia. Esto significa que no es necesario siquiera iniciar sesión en el sistema operativo de la instancia para ejecutar los comandos. Si la instancia se despliega como parte de un grupo de autoescalado o como respuesta a una llamada de API

(*application programming interface*), tanto el proceso de despliegue como el de configuración de la aplicación se habrán llevado a cabo sin intervención humana.

Las instancias Linux de EC2 aceptan dos tipos de *user data*: *scripts* de *shell* y directivas de Cloud-init. Este caso práctico demostrará ambos métodos, con la instalación de un servidor web con la pila LAMP (Linux, Apache, MySQL y PHP) en una instancia de tipo Amazon Linux 2.

### User data con *scripts* de *shell*

El *script* que se va a usar como ejemplo es el siguiente:

```
#!/bin/bash
yum update -y
amazon-linux-extras install -y lamp-mariadb10.2-php7.2 php7.2
yum install -y httpd mariadb-server
systemctl start httpd
systemctl enable httpd
usermod -a -G apache ec2-user
chown -R ec2-user:apache /var/www
chmod 2775 /var/www
find /var/www -type d -exec chmod 2775 {} \;
find /var/www -type f -exec chmod 0664 {} \;
echo "<?php phpinfo(); ?>" > /var/www/html/phpinfo.php
```

Es mucho más sencillo que los anteriores por dos razones: la distribución es conocida, por lo que no hay que considerar diferentes gestores de paquetes, y el *script* no acepta parámetros de usuario. Aunque se han obviado algunas comprobaciones de error para simplificarlo, el *script* es totalmente funcional.

Añadir estas tareas durante el arranque de la instancia incrementa el tiempo que esta tarda en estar disponible. En un despliegue manual, es necesario tener esto en cuenta antes de poder lanzar pruebas.



Los *scripts* de *shell* de *user data* deben empezar con el *shebang*, `#!/bin/bash` (o la *shell* necesaria, si procede). Estos *scripts*, además, se ejecutan como `root` y en modo no interactivo, por lo que:

- ▶ No es necesario ejecutar ninguno de los comandos con `Sudo`.
- ▶ Los comandos no pueden esperar entrada de datos del usuario.

El *log* de salida `/var/log/cloud-init-output.log` contendrá la salida estándar del *script*. En caso de error, este *log* se puede usar para verificar la salida de los comandos. Además, el *script* se copia a la ruta `/var/lib/cloud/instances/<instance-id>/` al procesarlo, por lo que es posible comprobar que el *script* que se ha ejecutado es efectivamente el deseado.

En un despliegue en la consola web, el *user data* se introduce en el paso de selección de los detalles de la instancia. Se puede seleccionar un fichero con el *script* o introducir el contenido del *script* directamente, tal como se indica en la Figura 6.

▼ Advanced Details

Metadata accessible ⓘ Enabled

Metadata version ⓘ V1 and V2 (token optional)

Metadata token response hop limit ⓘ 1

User data ⓘ ☒ As text ☐ As file ☐ Input is already base64 encoded

```
#!/bin/bash
yum update -y
amazon-linux-extras install -y lamp-mariadb10.2-php7.2 php7.2
yum install -y httpd mariadb-server
systemctl start httpd
systemctl enable httpd
usermod -a -G apache ec2-user
chown -R ec2-user:apache /var/www
chmod 2775 /var/www
find /var/www -type d -exec chmod 2775 {} \;
find /var/www -type f -exec chmod 0664 {} \;
echo "<?php phpinfo(); ?>" > /var/www/html/phpinfo.php
```

Figura 6. *User data* en la consola web. Fuente: elaboración propia.

El grupo de seguridad deberá permitir el acceso por HTTP, ya que la instancia va a alojar un servidor web.

Type ⓘ	Protocol ⓘ	Port Range ⓘ	Source ⓘ	Description ⓘ
SSH ▾	TCP	22	My IP 139.47.100.40/32	e.g. SSH for Admin
HTTP ▾	TCP	80	Custom 0.0.0.0/0, ::/0	e.g. SSH for Admin

Figura 7. Grupo de seguridad para servidor web. Fuente: elaboración propia.

Una vez arrancada la instancia, la vista de detalles contiene tanto la IP pública como un nombre DNS (*domain name system*) también público. Este nombre podrá usarse para acceder a la página de prueba `info.php` creada por el *script*.

search : i-Obad4580200aa667e Add filter						
Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status
	i-Obad4580200aa667e	t2.micro	eu-west-1b	running	Initializing	None
Instance: i-Obad4580200aa667e Public DNS: ec2-54-77-237-112.eu-west-1.compute.amazonaws.com						
Description	Status Checks	Monitoring	Tags			
Instance ID	i-Obad4580200aa667e	Public DNS (IPv4)	ec2-54-77-237-112.eu-west-1.compute.amazonaws.com			
Instance state	running	IPv4 Public IP	54.77.237.112			
Instance type	t2.micro	IPv6 IPs	-			
Finding	Opt-in to AWS Compute Optimizer for	Elastic IPs				

Figura 8. Instancia arrancada. Fuente: elaboración propia.

PHP Version 7.2.29	
System	Linux ip-172-31-8-67.eu-west-1.compute.internal 4.14.173-137.amzn2.x86_64 #1 SMP Wed Apr 1 18:06:08 UTC 2020 x86_64
Build Date	Mar 27 2020 17:08:35
Server API	FPM/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc
Loaded Configuration File	/etc/php.ini
Scan this dir for additional .ini files	/etc/php.d
Additional .ini files parsed	/etc/php.d/20-bz2.ini, /etc/php.d/20-calendar.ini, /etc/php.d/20-ctype.ini, /etc/php.d/20-exif.ini, /etc/php.d/20-fileinfo.ini, /etc/php.d/20-ftp.ini, /etc/php.d/20-gettext.ini, /etc/php.d/20-iconv.ini, /etc/php.d/20-json.ini, /etc/php.d/20-mysqld.ini, /etc/php.d/20-pdo.ini, /etc/php.d/20-phar.ini, /etc/php.d/20-sockets.ini, /etc/php.d/20-sqlite3.ini, /etc/php.d/20-tokenizer.ini, /etc/php.d/20-zip.ini, /etc/php.d/25-curl.ini, /etc/php.d/30-mysql.ini, /etc/php.d/30-pdo_mysql.ini, /etc/php.d/30-pdo_sqlite.ini

Figura 9. Página de prueba cargada desde el DNS público de la instancia. Fuente: elaboración propia.

## User data con directivas de Cloud-init

[Cloud-init](#) es un estándar de facto de la industria para la inicialización de instancias de nube multiplataforma. Está soportado por la gran mayoría de proveedores de nube pública y plataformas de nube privada. AWS hace uso de Cloud-init activamente para configurar las claves públicas de inicio de sesión en el fichero `~/.ssh/authorized_keys` del usuario `ec2-user`. En el ejemplo anterior, aunque el usuario proporciona un *script* de Bash directamente, el *user data* hace uso de Cloud-init para ejecutarlo.

AWS soporta el uso de directivas de Cloud-init directamente. Esto da más control al administrador, que puede establecer más parámetros de configuración, no solo el *script* a ejecutar. AWS identifica que el tipo de contenido del *user data* es Cloud-init cuando la primera línea es `#cloud-config`. Las siguientes líneas consiguen el mismo objetivo de instalación del servidor web.

```
#cloud-config
repo_update: true
repo_upgrade: all

packages:
- httpd
- mariadb-server

runcmd:
- [ sh, -c, "amazon-linux-extras install -y lamp-mariadb10.2-php7.2 php7.2" ]
- systemctl start httpd
- sudo systemctl enable httpd
- [ sh, -c, "usermod -a -G apache ec2-user" ]
- [ sh, -c, "chown -R ec2-user:apache /var/www" ]
- chmod 2775 /var/www
- [ find, /var/www, -type, d, -exec, chmod, 2775, {}, \; ]
- [ find, /var/www, -type, f, -exec, chmod, 0664, {}, \; ]
- [ sh, -c, 'echo "<?php phpinfo(); ?>" > /var/www/html/phpinfo.php' ]
```

El último bloque es muy parecido al *script* de Bash, con la diferencia de que cada línea pasa a ser un *array* de parámetros. La principal diferencia es que los paquetes de `httpd` y `mariadb-server` no se instalan con `yum`, sino a través de la directiva `packages`, específica de [Cloud-init](#). Otras directivas permiten la creación de usuarios, creación de ficheros de configuración, configuración de certificados de autoridades de confianza, etc.

El despliegue de la instancia es idéntico al ejemplo anterior, salvo que el contenido de *user data* contiene la directiva de Cloud-init, tal como muestra la Figura 10.

▼ Advanced Details

Metadata accessible		Enabled
Metadata version		V1 and V2 (token optional)
Metadata token response hop limit		1
User data		<input checked="" type="radio"/> As text <input type="radio"/> As file <input type="checkbox"/> Input is already base64 encoded

```
#cloud-config
repo_update: true
repo_upgrade: all

packages:
- httpd
- mariadb-server

runcmd:
- [ sh, -c, "amazon-linux-extras install -y lamp-mariadb10.2-php7.2 php7.2" ]
- systemctl start httpd
- sudo systemctl enable httpd
- [ sh, -c, "usermod -a -G apache ec2-user" ]
- [ sh, -c, "chown -R ec2-user:apache /var/www" ]
- chmod 2775 /var/www
- [ find, /var/www, -type, d, -exec, chmod, 2775, {}, \; ]
- [ find, /var/www, -type, f, -exec, chmod, 0664, {}, \; ]
- [ sh, -c, 'echo "<?php phpinfo(); ?>" > /var/www/html/phpinfo.php' ]
```

Figura 10. *User data* con *script* de Cloud-init. Fuente: elaboración propia.

## 4.6. Referencias bibliográficas

AWS. (s. f.). *Run commands on your Linux instance at launch*.

<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/user-data.html>

Cloud-init. (s. f.). *Cloud config examples*.

<https://cloudinit.readthedocs.io/en/latest/topics/examples.html>

Cloud-init. (s. f.). *Cloud-init documentation*.

<https://cloudinit.readthedocs.io/en/latest/index.html#>

Github. (s. f.). *plesk/node-express*. <https://github.com/plesk/node-express>

GNU. (s. f.). *Bourne Shell Builtins*.

[https://www.gnu.org/software/bash/manual/html\\_node/Bourne-Shell-Builtins.html](https://www.gnu.org/software/bash/manual/html_node/Bourne-Shell-Builtins.html)

GNU. (s. f.). *GNU «sed»*. <https://www.gnu.org/software/sed/manual/sed.txt>

GNU. (s. f.). *Redirections*.

[https://www.gnu.org/software/bash/manual/html\\_node/Redirections.html](https://www.gnu.org/software/bash/manual/html_node/Redirections.html)

GNU Operating System. (s. f.). *GNU Bash*. <https://www.gnu.org/software/bash>

MongoDB. (s. f.). *Choose which type of deployment is best for you*.

<https://www.mongodb.com/try/download/community>

MongoDB. (s. f.). *Configuration File Options*.

<https://docs.mongodb.com/manual/reference/configuration-options/>

MongoDB. (s. f.). *Install MongoDB Community Edition on Ubuntu*.

<https://docs.mongodb.com/manual/tutorial/install-mongodb-on-ubuntu/>

MongoDB. (s. f.). *Welcome to the MongoDB Documentation*.

<https://docs.mongodb.com>

Nginx. (s. f.). *Products*. <https://docs.nginx.com/>

Página de Express (<https://expressjs.com/>).

Página de NodeJS (<https://nodejs.org/en/>).

Página de NPM (<https://www.npmjs.com/>).

PHP. (s. f.). *Installation on Unix systems*.

<https://www.php.net/manual/en/install.unix.php>

PHP. (s. f.). *PHP Manual*. <https://www.php.net/manual/en/>

Robbins, A. (2010). *Bash Pocket Reference*. O'Reilly Media.

SELinux Project. (s. f.). *Main Page*. [https://selinuxproject.org/page/Main\\_Page](https://selinuxproject.org/page/Main_Page)

Ubuntu. (s. f.). *Package: mongobd (1:3.6.3-0ubuntu1) [universe]*.

<https://packages.ubuntu.com/bionic/database/mongodb>

## Documentación de MongoDB

MongoDB. (s. f.). *Configuration File Options*.

<https://docs.mongodb.com/manual/reference/configuration-options/index.html>

Los ejemplos anteriores sirven como punto de partida, pero la configuración es demasiado simple para un despliegue en producción. La documentación de MongoDB es muy completa. Repasar las opciones de configuración del servidor puede dar una idea sobre la cantidad de opciones que pueden ser necesarias configurar en una instalación real.

## Documentación de Nginx

Nginx. (s. f.). *PHP FastCGI Example*.

<https://www.nginx.com/resources/wiki/start/topics/examples/phpfcgi/>

La documentación de Nginx es también muy completa e incluye ejemplos similares de integración con un motor de PHP como el propuesto en este tema. Estudiar una instalación ligeramente diferente es útil para aprender diferentes puntos de vista de una misma implementación.

1. ¿Qué código de salida significa que el proceso ha terminado sin error?
  - A. 0.
  - B. 1.
  - C. -127.
  - D. Todos los anteriores.
  
2. ¿Qué cadena de texto delimita un heredoc?
  - A. NGINX\_CONF .
  - B. Cualquiera, siempre que la de inicio sea igual que la de final.
  - C. Cualquiera, Bash reconoce el fin del documento automáticamente.
  - D. Cualquiera, pero debe ser en mayúsculas.
  
3. ¿Qué significa el *flag* -p en el comando `mkdir`?
  - A. Crear la carpeta con permisos de root.
  - B. Borrar la carpeta.
  - C. No tiene efecto.
  - D. Crear la ruta de carpetas, si no existe.
  
4. ¿Qué diferencia hay entre `apt-get` y `yum`?
  - A. Ninguna, cumplen la misma función.
  - B. `yum` es un alias de `apt-get`.
  - C. `apt-get` es un gestor de paquetes en distribuciones Debian, mientras que `yum` es el gestor en distribuciones RedHat y Fedora.
  - D. `apt-get` es un gestor de paquetes en distribuciones Ubuntu exclusivamente, mientras que `yum` es el gestor en el resto de las distribuciones Debian.



5. ¿Qué modificador del comando `test` comprueba si existe un fichero?
- A. `-e`.
  - B. `-z`.
  - C. `-E`.
  - D. `-h`.
6. ¿Qué comando de gestión de parámetros usa los *scripts* expuestos en el tema?
- A. `getoptions`.
  - B. `getopts`.
  - C. Usan las variables normales de Bash: `$1`, `$2`, etc.
  - D. `while`.
7. ¿Qué hace el comando `mv /etc/mongod.conf /etc/mongod.conf.orig`?
- A. Hace una copia de seguridad del fichero de configuración por defecto.
  - B. Borra el fichero de configuración por defecto para evitar la colisión con el fichero generado en el *script*.
  - C. Crea el fichero de configuración por defecto.
  - D. A y B son correctas.
8. ¿Cómo debe empezar el contenido del *user data* de AWS para que sea considerado como sentencia de Cloud-init?
- A. `#cloud-config`.
  - B. `#cloudinit`.
  - C. `#!/bin/bash`.
  - D. `<powershell>`.

9. ¿Qué limitaciones hay en los *scripts* del *user data*?
- A. Solo pueden contener diez líneas como máximo.
  - B. Solo pueden estar escritos en Python.
  - C. No pueden esperar entrada de datos del usuario.
  - D. No pueden imprimir datos por pantalla.
10. ¿Cuál es el gestor de paquetes de NodeJS?
- A. apt-get.
  - B. yum.
  - C. npm.
  - D. pip.