

## Instalación de andino

La instalación avanzada está pensada para usuarios que quieren ver cómo funciona internamente Andino

Para instalar y ejecutar Andino, seguiremos estos pasos:

Paso 1: Clonar repositorio.

```
sudo mkdir /etc/portal
```

```
cd /etc/portal
```

```
sudo git clone https://github.com/MunicipioLomasdeZamora/opendata .
```

Paso 2: Especificar las variables de entorno para el contenedor de postgresql.

NOTA: Debemos usar un dominio válido para la variable `DOMINIO`, de otra forma el envío de mails no funcionará. Postfix requiere un "fully-qualified domain name (FQDN)". Ver [la documentación de Postfix](#) para más detalles.

```
DB_USER=<my user>
```

```
DB_PASS=<my pass>
```

```
DOMINIO=andino.midominio.com.ar
```

```
ANDINO_VERSION=<version que deseamos instalar>
```

```
sudo su -c "echo POSTGRES_USER=$DB_USER > .env"
```

```
sudo su -c "echo POSTGRES_PASSWORD=$DB_PASS >> .env"
```

```
sudo su -c "echo NGINX_HOST_PORT=80 >> .env"
```

```
sudo su -c "echo DATASTORE_HOST_PORT=8800 >> .env"
```

```
sudo su -c "echo maildomain=$DOMINIO >> .env"
```

```
sudo su -c "echo ANDINO_TAG=$ANDINO_VERSION >> .env"
```

```
sudo su -c "echo NGINX_HOST_SSL_PORT=443 >> .env"
```

```
sudo su -c "echo SITE_HOST=$DOMINIO >> .env"
```

Paso 3: Construir y lanzar los contenedor de servicios usando el archivo **latest.yml**:  
`docker-compose -f latest.yml up -d db postfix redis solr`

Paso 4: Construir y lanzar el contenedor de **andino** usando el archivo **latest.yml**:  
`docker-compose -f latest.yml up -d portal`

Paso 5: Inicializar la base de datos y la configuración de la aplicación:

```
EMAIL=admin@example.com
```

```
HOST=datos.gob.ar
```

```
DB_USER=<my db user>
```

```
DB_PASS=<my db pass>
```

```
STORE_USER=<my datastore user>
```

```
STORE_PASS=<my datastore password>
```

```
docker-compose -f latest.yml exec portal /etc/ckan_init.d/init.sh -e  
"$EMAIL" -h "$HOST" \  
  
-p "$DB_USER" -P "$DB_PASS" \  
  
-d "$STORE_USER" -D "$STORE_PASS"
```

Paso 6: Construir el contenedor de **nginx** usando el archivo **latest.yml**:  
`docker-compose -f latest.yml up -d nginx`

## Desinstalar andino

Esta secuencia de comandos va a ELIMINAR TODOS LOS CONTENEDORES, IMÁGENES y VOLUMENES de la aplicación de la vm donde está instalada la plataforma.

Esta operación no es reversible. **Perderás todos tus datos si realizas esta operación.**

```
app_dir="/etc/portal/"

cd $app_dir

docker-compose -f latest.yml down -v

cd ~/

sudo rm $app_dir -r
```

## Checklist para la puesta en producción

Si estás por configurar tu instancia de Andino para un ambiente productivo, tenemos algunas recomendaciones para que sigas y verifiques si tu instancia está bien configurada.

Verificá que tu instancia de Andino tenga un nombre de dominio único y bien configurado

Para el correcto funcionamiento de tu instancia de Andino, es recomendable que la misma tenga un único nombre de dominio asignado.

Una vez que tengas definido el nombre de dominio (ej: `datos.ministerio.gob.ar`) y el mismo se resuelva a la IP pública asignado al *host* de tu Andino (o al *load balancer/frontend server* que opcionalmente tengas) es importante que tu instancia de Andino conozca ese nombre de dominio y que esté configurado para responder al mismo.

Verificar si mi Andino tiene el nombre de dominio configurado correctamente

Para saber si tu instancia de Andino tiene el nombre de dominio correctamente configurado seguí los siguientes pasos, ejecutando el comando en el directorio de instalación de Andino (ej: `/etc/portal`):

```
docker-compose -f latest.yml exec portal grep ckan\.site_url
/etc/ckan/default/production.ini
```

Si tu sitio está bien configurado, el valor del parámetro de configuración `ckan.site_url` deberá coincidir con el nombre de dominio de tu Andino (incluyendo el *schema*, `http` o `https`):

```
ckan.site_url=http://datos.ministerio.gob.ar/
```

Si éste no coincide, deberás modificar el valor del parámetro en la configuración de tu Andino.

Actualizando el nombre de dominio asignado a Andino

Para actualizar el nombre de dominio que tiene tu andino (por ejemplo datos.ministerio.gob.ar) debés ejecutar el siguiente comando:

```
docker-compose -f latest.yml exec portal /etc/ckan_init.d/update_conf.sh  
"ckan.site_url=http://datos.ministerio.gob.ar/";
```

Verificá que el contenedor portal pueda resolver el nombre de dominio asignado a la instancia

Para asegurar el correcto funcionamiento de algunos componentes de la arquitectura de Andino, es necesario que desde dentro de los contenedores Docker que forman parte de la solución, el nombre de dominio asignado a tu Andino pueda ser resuelto correctamente.

Seguir las recomendaciones de seguridad

## **Mantenimiento**

Exploración de la instancia de andino

¿Qué está corriendo docker?

Para obtener una lista de lo que está corriendo actualmente Docker, podemos usar el siguiente comando:

```
docker ps # Tabla de ejecucion actual
```

```
docker ps -q # Listado de IDs de cada contenedor
```

```
docker ps -aq # Listado de IDs de todos los contenedores  
disponibles.
```

## Utilización del archivo latest.yml en los comandos de docker-compose

En múltiples secciones de esta documentación se emplea el uso de comandos que comienzan de la siguiente manera:

```
docker-compose -f latest.yml
```

Es importante recordar que no alcanzaría con especificar el directorio absoluto del archivo `latest.yml`; es necesario ejecutar estos comandos *exactamente en ese mismo directorio*, debido a que ahí también se encuentra el archivo que contiene las variables de entorno (`.env`) ya que es el directorio de instalación de Andino.

### Ingresar al contenedor principal de andino

El contenedor principal de andino, donde se ejecuta la aplicación CKAN, es denominado `portal`. Para ingresar en una sesión de consola en el contenedor, ejecutar:

```
docker-compose -f latest.yml exec portal /bin/bash
```

### Listar todas las Propiedades de cada contenedor

```
docker-compose -f latest.yml ps -q portal solr db | xargs -n 1 |  
while read container; do docker inspect $container; done
```

## Administración de usuarios

### Crear un usuario ADMIN

```
docker-compose -f latest.yml exec portal  
/etc/ckan_init.d/add_admin.sh mi_nuevo_usuario_admin  
email_del_usuario_admin
```

El comando solicitará la contraseña del usuario administrador.

### Listar mis usuarios

```
docker-compose -f latest.yml exec portal /etc/ckan_init.d/paster.sh  
--plugin=ckan user list
```

Ver los datos de un usuario

```
docker-compose -f latest.yml exec portal /etc/ckan_init.d/paster.sh  
--plugin=ckan user nombre-de-usuario
```

Crear un nuevo usuario

```
docker-compose -f latest.yml exec portal /etc/ckan_init.d/paster.sh  
--plugin=ckan user add nombre-de-usuario
```

Crear un nuevo usuario extendido

```
docker-compose -f latest.yml exec portal /etc/ckan_init.d/paster.sh  
--plugin=ckan user add nombre [email=mi-usuario@host.com  
password=mi-contraseña-rara apikey=unsecretomisticoaleible]
```

Eliminar un usuario

```
docker-compose -f latest.yml exec portal /etc/ckan_init.d/paster.sh  
--plugin=ckan user remove nombre-de-usuario
```

Cambiar password de un usuario

```
docker-compose -f latest.yml exec portal /etc/ckan_init.d/paster.sh  
--plugin=ckan user setpass nombre-de-usuario
```

## Usuario administrador de CKAN

Existe un usuario administrador llamado *default*, el cual es utilizado por la aplicación para la ejecución de ciertas funciones que corren de fondo (por ejemplo, la actualización de la caché del data.json). Es muy importante que dicho usuario **siempre** esté disponible ya que, en caso de no estarlo, provocaría un funcionamiento incorrecto en el portal; por lo tanto, se recomienda muy fuertemente no intentar eliminarlo ni desactivarlo.

## Configuraciones de andino

## Modificar el archivo de configuración

El archivo de configuración de andino se llama `production.ini`, y se lo puede encontrar y modificar de la siguiente manera:

```
# Ingresar al contenedor
```

```
cd /etc/portal
```

```
docker-compose -f latest.yml exec portal /bin/bash
```

```
# Una vez adentro, abrimos el archivo production.ini, y buscamos la  
sección que necesita ser modificada
```

```
vim /etc/ckan/default/production.ini
```

```
# Editamos y, luego de salir del contenedor, lo reiniciamos
```

```
docker-compose -f latest.yml restart portal nginx
```

## Cambiar la configuración del SMTP

Por defecto, andino usará un servidor postfix integrado para el envío de emails. Para usar un servidor SMTP propio, debemos cambiar la configuración del archivo `production.ini`. Para lograrlo, podemos hacerlo de dos formas:

1 ) Ingresando al contenedor.

Debemos buscar y editar en el archivo `production.ini` la configuración de email que luce como:

```
## Email settings
```

```
error_email_from=admin@example.com

smtp.server = postfix

#smtp.starttls = False

smtp.user = portal

smtp.password = portal

smtp.mail_from = administrador
```

## 2 ) Ejecutando comandos paster

Suponiendo que nuestro servidor SMTP está en smtp.gmail.com, la dirección de correo del usuario es smtp\_user\_mail@gmail.com, la contraseña de esa dirección de correo mi\_pass y queremos usar "tls", podemos ejecutar los siguientes comandos:

```
docker-compose -f latest.yml exec portal
/etc/ckan_init.d/update_conf.sh "smtp.server=smtp.gmail.com:587";

docker-compose -f latest.yml exec portal
/etc/ckan_init.d/update_conf.sh
"smtp.user=smtp_user_mail@gmail.com";

docker-compose -f latest.yml exec portal
/etc/ckan_init.d/update_conf.sh "smtp.password=mi_pass";

docker-compose -f latest.yml exec portal
/etc/ckan_init.d/update_conf.sh "smtp.starttls=True";

docker-compose -f latest.yml exec portal
/etc/ckan_init.d/update_conf.sh
"smtp.mail_from=smtp_user_mail@gmail.com";

# Finalmente reiniciamos el contenedor

docker-compose -f latest.yml restart portal nginx
```



Tener en cuenta que si se utiliza un servidor SMTP, se debe setear la configuración con **un correo electrónico de @gmail.com**, y que **starttls debe estar en True**.

Cambiar el remitente de los correos electrónicos que envía Andino

Para modificar el remitente de los correos electrónicos que el sistema envía (por ejemplo los de creación de usuarios nuevos o los de olvido de contraseña), se deben seguir los pasos de la sección Cambiar la configuración del SMTP pero modificando el atributo de configuración `smtp.mail_from`.

Cambiar el id del container de Google Tag Manager

Será necesario modificar la configuración en el archivo `production.ini`.

Esta vez, buscaremos la configuración debajo de la sección `[app:main]` (vas a encontrar campos como `"superThemeTaxonomy"` y `"ckan.site.title"`).

El campo que estamos buscando es

`ckan.google_tag_manager.gtm_container_id`.

En caso de no encontrar el campo mencionado, lo podemos agregar:

```
ckan.google_tag_manager.gtm_container_id = { id que necesitás  
guardar }
```

Google Tag Manager

Para configurar el código de seguimiento de Google Tag Manager ejecutar el siguiente comando:

```
docker-compose -f latest.yml exec portal  
/etc/ckan_init.d/update_conf.sh  
"ckan.google_tag_manager.gtm_container_id=<tu código de seguimiento  
GTM>";
```

# Finalmente reiniciamos el contenedor

```
docker-compose -f latest.yml restart portal nginx
```

Cambiar el id del tag de Google Analytics

Será necesario modificar el archivo de configuración `production.ini`.

La sección a buscar luce de esta manera:

```
## Google Analytics

googleanalytics.id = { un id }

googleanalytics_resource_prefix = { un prefix }

googleanalytics.domain = { un dominio }
```

Lo que se debe modificar es el campo `googleanalytics.id`.

### Indexar datasets con Google Dataset Search

Andino cuenta con la posibilidad de utilizar Google Dataset Search para que éste indexe tus datasets. Para el mejor entendimiento de la herramienta, recomendamos que entres a <https://search.google.com/search-console/about>.

Por default, esta configuración se encuentra desactivada. Para activarla, podés ir a *Configuración -> Configuración avanzada -> Google Dataset Search* -> Clickear el checkbox y guardar el cambio.

### Deshabilitar la URL `/catalog.xlsx`

En caso de desear deshabilitar la URL `/catalog.xlsx`, se puede ejecutar el siguiente comando:

```
docker-compose -f latest.yml exec portal
/etc/ckan_init.d/update_conf.sh
"andino.disable_catalog_excel_url=True";
```

En caso de querer restaurarlo, se debe configurar el atributo `andino.disable_catalog_excel_url` con el valor `False`.

### Configuración de la llamada de invalidación de caché

La aplicación puede ser configurada para hacer una llamada HTTP ante cada cambio en los metadatos del portal. Esta llamada (A.K.A. "hook") puede configurarse para ser a cualquier URL, y usando cualquier método HTTP. Se deberá

utilizar un campo llamado `andino.cache_clean_hook`, que tendrá asignada la URL a la cual queremos enviarle requests HTTP que lograrán ese efecto.

Además, el paquete "Andino" provee una configuración de *nginx* que permite recibir esta llamada e invalidar la caché.

Para configurar internamente *nginx* y *andino*, sólo es necesario pasar la opción `--nginx-extended-cache` al momento de usar el script de instalación.

Si nuestra aplicación ya está instalada, podemos seguir los siguientes pasos:

1. Actualizar a la última versión de la aplicación, con el script de actualización.
2. Ir al directorio de instalación `cd /etc/portal`
3. Editar el archivo `.env`
4. Agregar una línea nueva que sea:  
`NGINX_CONFIG_FILE=nginx_extended.conf`
5. Reiniciar el contenedor de *nginx* `docker-compose -f latest.yml up -d nginx`

Luego configuramos el hook de invalidación:

1. Entramos al contenedor del portal: `docker-compose -f latest.yml exec portal bash`
2. Configuramos el hook: `/etc/ckan_init.d/update_conf.sh`  
`andino.cache_clean_hook=http://nginx/meta/cache/purge`
3. Salimos `exit`
4. Reiniciamos el portal: `docker-compose -f latest.yml restart portal nginx`

*Nota: tener en cuenta que, por defecto, se emplea el método PURGE para disparar el hook, lo cual se puede cambiar editando el campo `andino.cache_clean_hook_method` dentro del archivo de configuración `production.ini`.*

## Caché externa

Es posible implementar la caché externa por fuera del paquete *andino*. Para esto, en el servidor que servirá de caché, necesitamos instalar [openresty](#). Esta plataforma web nos permite correr **nginx** y modificar su comportamiento usando [lua](#).

Luego de instalar **openresty**, debemos activarlo para que empiece cada vez que se prenda el servidor:

```
systemctl enable openresty
```

```
systemctl restart openresty
```

Luego de instalar operesty, debemos agregar los archivos de configuración. Primero borramos la configuración de nginx que viene por defecto en /etc/openresty/nginx.conf y agregamos la nuestra:

```
#user  nobody;

worker_processes  1;

#error_log  logs/error.log;

#error_log  logs/error.log  notice;

#error_log  logs/error.log  info;

#pid        logs/nginx.pid;

events {
    worker_connections  1024;
}

http {
    include        mime.types;

    default_type  application/octet-stream;

    #log_format  main  '$remote_addr - $remote_user [$time_local]
"$request" '

    #              '$status $body_bytes_sent "$http_referer" '

    #              '"$http_user_agent" "$http_x_forwarded_for"'

    #access_log  logs/access.log  main;

    sendfile      on;

    #tcp_nopush   on;

    #keepalive_timeout  0;
```

```
    keepalive_timeout 65;

    #gzip on;

    include /etc/nginx/conf.d/*.conf;
}
```

Luego, creamos el directorio donde pondremos la configuración de nuestra caché y creamos el archivo.

```
mkdir -p /etc/nginx/conf.d/

touch /etc/nginx/conf.d/000-andino-cache.conf
```

El archivo 000-andino-cache.conf contendrá lo siguiente. Es necesario cambiar la palabra IP\_A\_ANDINO por la IP donde está andino.

```
proxy_cache_path /tmp/nginx_cache/ levels=1:2 keys_zone=cache:30m
max_size=250m;

proxy_temp_path /tmp/nginx_proxy 1 2;

server_tokens off;

server {

    client_max_body_size 300M;

    location / {

        proxy_pass http://IP_A_ANDINO:80/;

        proxy_set_header X-Forwarded-For $remote_addr;

        proxy_set_header Host $host;

        proxy_cache cache;

        # Disable cache for logged-in users
```

```

proxy_cache_bypass $cookie_auth_tkt;

proxy_no_cache $cookie_auth_tkt;

proxy_cache_valid 30m;

proxy_cache_key $host$scheme$proxy_host$request_uri;

# Ignore apache "Cache-Control" header

# See
https://lists.okfn.org/pipermail/ckan-dev/2016-March/009864.html

proxy_ignore_headers Cache-Control;

# In emergency comment out line to force caching

# proxy_ignore_headers X-Accel-Expires Expires;

# Show cache status

add_header X-Cache-Status $upstream_cache_status;

}

location /meta/cache/purge {

    allow 192.168.0.0/16;

    allow 172.16.0.0/12;

    allow 10.0.0.0/8;

    allow 127.0.0.1;

    deny all;

    if ($request_method = PURGE ) {

        content_by_lua_block {

            filename = "/tmp/nginx_cache/"

            local f = io.open(filename, "r")

            if (f~=nil) then

```

```

        io.close(f)

        os.execute('rm -rf "'..filename..'")

    end

    ngx.say("OK")

    ngx.exit(ngx.OK)

}

}

}

```

Si se está utilizando la configuración SSL, agregar al final del archivo la siguiente sección, reemplazando `_`[el texto que aparezca entre corchetes (y éstos) por el valor adecuado]:

```

server {

    client_max_body_size 300M;

    listen 443 ssl http2;

    listen [::]:443 ssl http2;

    server_name [Tu nombre de dominio];

    ssl_certificate [Path al certificado];

    ssl_certificate_key [Path a la llave del certificado];

    ssl_session_cache shared:SSL:50m;

    ssl_session_timeout 1d;

    ssl_session_tickets off;

    ssl_prefer_server_ciphers on;

    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;

```

```

ssl_ciphers
'ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE-ECDSA-DES-CBC3-SHA:ECDHE-RSA-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:DES-CBC3-SHA:!DSS';

resolver 8.8.8.8 8.8.4.4;

ssl_stapling on;

ssl_stapling_verify on;

ssl_trusted_certificate [Path al certificado];

add_header Strict-Transport-Security "max-age=31536000;
includeSubdomains; preload";

add_header X-Xss-Protection "1; mode=block" always;

add_header X-Content-Type-Options "nosniff" always;

add_header X-Frame-Options "SAMEORIGIN" always;

proxy_ignore_headers Cache-Control;

location / {

    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

    proxy_set_header Host $http_host;

    proxy_set_header X-Forwarded-Protocol https;

    proxy_ignore_headers Cache-Control;

    add_header X-Xss-Protection "1; mode=block" always;

    add_header X-Content-Type-Options "nosniff" always;

```



```

    add_header X-Frame-Options "SAMEORIGIN" always;

    proxy_redirect off;

    if (!-f $request_filename) {

        proxy_pass https://[Tu URL]:443;

        break;

    }

}

}

```

Finalmente, reiniciamos **openresty**: `systemctl restart openresty`.

Ahora que tenemos la caché configurada, necesitamos configurar la llamada, o hook, de invalidación de caché. Para esto, entramos al servidor donde está corriendo andino y corremos:

```

IP_INTERNA_CACHE=<ip interna del servidor de caché>

cd /etc/portal

docker-compose -f latest.yml exec portal
/etc/ckan_init.d/update_conf.sh
"andino.cache_clean_hook=http://$IP_INTERNA_CACHE/meta/cache/purge";

docker-compose -f latest.yml restart portal nginx

```

Si queremos probar la integración, podemos entrar al contenedor de andino y probar invalidar la caché:

```

IP_INTERNA_CACHE=<ip interna del servidor de caché>

cd /etc/portal

docker-compose -f latest.yml exec portal curl -X PURGE
"http://$IP_INTERNA_CACHE/meta/cache/purge";

# => OK

```

**NOTA:** Si estamos usando nuestro andino con un IP y *no* con un dominio, tendremos que cambiar la configuración `ckan.site_url` para que use la IP del servidor donde se encuentra la caché externa.

```
IP_PUBLICA_CACHE=<ip publica del servidor caché>
```

```
cd /etc/portal
```

```
docker-compose -f latest.yml exec portal  
/etc/ckan_init.d/update_conf.sh  
"ckan.site_url=http://$IP_PUBLICA_CACHE";
```

```
docker-compose -f latest.yml restart portal nginx
```

Especificar las licencias a utilizar

Existe un JSON que contiene las licencias a utilizar en el portal, y cuyo path es `/var/lib/ckan/theme_config/licenses.json`. Este archivo está especificado en el campo `licenses_group_url` del archivo de configuración.

Es posible cambiarlo para lograr utilizar un archivo distinto; para ello, hay que cambiar el path por el deseado, teniendo en cuenta las siguientes indicaciones: \*

- Para utilizar un path de un archivo existente en el container, el mismo debe comenzar con el texto `file://`, tal y como ocurre con el path utilizado por default. \*
- Para utilizar una URL, debe comenzar con `http://` o `https://`.

## Configuración de CORS

Cuando es necesario acceder a Andino desde URLs distintas que apuntan a una misma instancia (ej: accediendo a través de un gateway/caché o directamente a la instancia de Andino o usando la IP pública del servidor *host*) es necesario, para el correcto funcionamiento de Andino, configurar parámetros para habilitar CORS (*Cross-Origin Resource Sharing*). Esto se debe a que un Andino debe tener una URL canónica, por lo tanto, las demás URLs utilizadas deben estar en el *whitelist* de CORS de Andino.

Para poder navegar tu Andino usando como URL una que no es la canónica de tu instancia, tenés que realizar dos acciones (los comandos deben ser ejecutados desde el directorio de instalación de Andino; por *default*, */etc/portal*):

1. Habilitar el comportamiento CORS: `docker-compose -f latest.yml exec portal /etc/ckan_init.d/update_conf.sh "ckan.cors.origin_allow_all = false"` (si bien el parámetro de configuración tiene el valor `false`, esto habilita el control de URLs contra el *whitelist*).
2. Agregar las URLs al *whitelist*: `docker-compose -f latest.yml exec portal /etc/ckan_init.d/update_conf.sh "ckan.cors.origin_whitelist=http://localhost:8080 http://127.0.0.1 http://127.0.0.1:8080"` (en el ejemplo se habilitan las URLs `http://localhost:8080`, `http://127.0.0.1` y `http://127.0.0.1:8080`).

Luego reiniciá los contenedores `portal` y `nginx`: `docker-compose -f latest.yml restart nginx portal`.

Tené en cuenta que el script va a reemplazar las URLs existentes en el campo por lo que le pases por parámetro. Si hay URLs que querés mantener en la configuración, buscalas con el siguiente comando y especificalas junto a las nuevas:

```
docker-compose -f latest.yml exec portal bash -c 'grep "ckan.cors.origin_whitelist" /etc/ckan/default/production.ini'
```

Si deseás habilitar **todas** las URLs para CORS (no recomendado por cuestiones de seguridad), en el paso 1 debés pasar el valor `true` para el atributo de configuración `ckan.cors.origin_allow_all` e ignorar el paso 2.

## Configuración del explorador de series de tiempo

Andino tiene instalado el plugin `ckanext-seriestiempoarexplorer`, pero no se encuentra presente entre los plugins activos. Para activarlo, debemos entrar al contenedor de `andino`, editar el archivo `/etc/ckan/default/production.ini` y agregar el `seriestiempoarexplorer` a la lista de plugins. Luego de agregarlo, debemos reiniciar el servidor.

```
cd /etc/portal
```

```
docker-compose -f latest.yml exec portal bash;
```

```
vim /etc/ckan/default/production.ini
```

```
# ...
```

```
apachectl restart
```

Luego, si vamos a la configuración del sitio, podremos apreciar que se agregó una nueva sección "Series" en el apartado "Otras secciones del portal".

Acceso a los datos de andino

Encontrar los volúmenes de mi andino dentro del filesystem del host

```
docker-compose -f latest.yml ps -q andino solr db | xargs -n 1 |  
while read container; do docker inspect -f ' {{.Name}}: {{range  
.Mounts}}{{.Source}}: {{.Destination}}  {{end}} ' $container; done
```

Ver las direcciones IP de mis contenedores

```
docker-compose -f latest.yml ps -q andino solr db | xargs -n 1 |  
while read container; do docker inspect -f ' {{.Name}}: {{range  
.NetworkSettings.Networks}}{{.IPAddress}}{{end}} ' $container; done
```

Ver las variables de entorno que tienen mis contenedores

```
docker-compose -f latest.yml ps -q andino solr db | xargs -n 1 |  
while read container; do docker inspect -f ' {{range $index, $value  
:= .Config.Env}}export {{$value}}{{println}}{{end}} ' $container;  
done
```

Acceder con un cliente de PostgreSQL a las bases de datos

```
docker-compose -f dev.yml exec db psql -U postgres
```

```
# psql \c ckan db default CKAN
```

```
# psql \c datastore_default db datastore CKAN
```

## Eliminar objetos definitivamente

Es bien sabido que, dentro de CKAN, cada vez que borramos algún elemento, en verdad no se borra, sino que pasa a estar inactivo; por lo tanto, tener alguna forma de eliminar elementos de manera definitiva resulta altamente necesario.

## Purgar Organizaciones Borradas

```
curl -X POST http://tu-host/api/3/action/organization_purge -H  
"Authorization:tu-api-key" -F id=id-o-nombre-que-se-desea-borrar
```

## Purgar Grupos Borrados

```
curl -X POST http://tu-host/api/3/action/group_purge -H  
"Authorization:tu-api-key" -F id=id-o-nombre-que-se-desea-borrar
```

## Purgar Datasets Borrados

```
curl -X POST http://tu-host/api/3/action/dataset_purge -H  
"Authorization:tu-api-key" -F id=id-o-nombre-que-se-desea-borrar
```

## Listar nombres de los datasets contenidos en Andino

```
docker-compose -f latest.yml exec portal /etc/ckan_init.d/paster.sh  
--plugin=ckan dataset list | grep -v DEBUG | grep -v count | grep  
-v Datasets | xargs -n2 | while read id name; do echo $name; done
```

## Backups

Es altamente recomendable hacer copias de seguridad de los datos de la aplicación, tanto la base de datos como los archivos de configuración y subidos por los usuarios.

### Backup de la base de datos

Un ejemplo fácil de hacer un backup de la base de datos sería:

```
container=$(docker-compose -f latest.yml ps -q db)
```

```
today=`date +%Y-%m-%d.%H:%M:%S`
```

```

filename="backup-$(date +%Y-%m-%d).gz"

# Creo un directorio temporal y defino dónde generaré el backup
backupdir=$(mktemp -d)

backupfile="$backupdir/$filename"

# Exporto la base de datos

docker exec $container pg_dumpall -c -U postgres | gzip >
"$backupfile"

# Copio el archivo al directorio actual y borro el original

# Podría reemplazar $PWD con mi directorio de backups, como
/etc/portal/backups

mv "$backupfile" $PWD

```

Y para los demás archivos de la aplicación (requiere [jq](#)):

```

backupdir=$(mktemp -d)

today=`date +%Y-%m-%d.%H:%M:%S`

appbackupdir="$backupdir/application/"

mkdir $appbackupdir

container=$(docker-compose -f latest.yml ps -q portal)

docker inspect --format '{{.json .Mounts}}' $container | jq -r
'.[[]][.Name, .Source, .Destination] | @tsv' |

while IFS=$'\t' read -r name source destination; do

    if ls $source/* 1> /dev/null 2>&1; then

```

```

        dest="$appbackupdir$name"

        mkdir -p $dest

        tar -C "$source" -zcvf "$dest/backup_$(date +%Y%m%d).tar.gz" $(ls
$source)

    else

        echo "No file at $source"

    fi

done

tar -C "$appbackupdir../" -zcvf backup.tar.gz "application/"

```

Podría colocarse esos scripts en el directorio donde se instaló la aplicación (ejemplo : /etc/portal/backup.sh) y luego agregar un cron:

Para correr el script cada domingo, podríamos usar la configuración 0 0 \* \* 0 (ver [cron](#) para más información), correr el comando crontab -e y agregar la línea:

```
0 0 * * 0 cd /etc/portal/ && bash /etc/portal/backup.sh
```

## Realizar un backup del file system

```
# Exporto el path al almacenamiento del volumen
```

```
export CKAN_FS_STORAGE=$(docker inspect --format '{{ range .Mounts
}}{{ if eq .Destination "/var/lib/ckan" }}{{ .Source }}{{ end }}{{
end }}' andino)
```

```
# Creo un tar.gz con la info.
```

```
tar -C "$(dirname "$CKAN_FS_STORAGE")" -zcvf  
/ruta/para/guardar/mis/bkps/mi_andino.fs-data_$(date +%F).tar.gz  
"$(basename "$CKAN_FS_STORAGE")"
```

Realizar un backup de la configuración

```
# Exporto el path al almacenamiento del volumen
```

```
export ANDINO_CONFIG=$(docker inspect --format '{{ range .Mounts  
}}{{ if eq .Destination "/etc/ckan/default" }}{{ .Source }}{{ end  
}}{{ end }}' andino)
```

```
# Creo un tar.gz con la info.
```

```
tar -C "$(dirname "$ANDINO_CONFIG")" -zcvf  
/ruta/para/guardar/mis/bkps/mi_andino.config-data_$(date +%F).tar.gz  
"$(basename "$ANDINO_CONFIG")"
```

## Comandos de DataPusher

Deben ser corridos en el directorio de instalación de Andino.

Subir todos los recursos al Datastore

Es posible que existan recursos que no hayan sido subidos al Datastore. Para buscar e intentar subir dichos recursos, ejecutar:

```
docker-compose -f latest.yml exec portal  
/usr/lib/ckan/default/bin/paster --plugin=ckan datapusher submit_all  
-c /etc/ckan/default/production.ini
```



Se preguntará si se desea proceder. Al escribir que sí (y), iniciar la subida de recursos. Para cada uno de ellos, se escribirá su id y luego el status: si subió correctamente, aparecerá "OK"; en caso contrario, "FAIL".

## Seguridad

Mantener seguro un servidor web puede ser una tarea ardua, pero sobre todo es *constante*, ya que *constantemente* se detectan nuevas vulnerabilidades en los distintos softwares. Y un servidor web no es la excepción! En este breve apartado, se darán pequeñas recomendaciones para mantener seguro el servidor, no solo antes posibles atacantes, sino tambien ante posibles fallos del sistema y como efectuar mitigaciones.

Las siguientes recomendaciones pueden ser implementadas fácilmente en un sistema Ubuntu 16.04, el cual es el recomendado (a la fecha), para correr la aplicación.

## HTTPS

HTTPS permite que la conexión entre el *browser* y el servidor sea encriptada y de esta manera segura. Es altamente recomendable usar HTTPS, para mantener la privacidad de los usuarios. El portal de documentación para desarrolladores de Google provee buena información sobre esto:

<https://developers.google.com/web/fundamentals/security/encrypt-in-transit/why-https>

### Plugin ckanext-security

Para las versiones 2.5.7 en adelante, existe un plugin para mejorar la seguridad de CKAN que se puede encontrar en github. Para su utilización, es necesario el uso de certificados SSL, ya sea en Andino mismo (teniendo *https* en el campo *site\_url* de la configuración del portal) o mediante un reverse proxy.

Para este plugin existen dos scripts, los cuales se pueden encontrar en el directorio `/etc/ckan_init.d/security` dentro del contenedor de Andino; uno para activarlo (`enable_ckanext_security.sh`) y otro para desactivarlo (`disable_ckanext_security.sh`). Sólo se necesita ejecutar el que se necesite y reiniciar apache.

## Sistema y librerías

Es *altamente recomendable* mantener el sistema operativo y las aplicaciones que usemos actualizadas. Constantemente se están subiendo *fixes* de seguridad y posibles intrusos podrían aprovechar que las aplicaciones o el mismo sistema operativo estén desactualizados. Periódicamente, podríamos constatar las nuevas

versiones de nuestro software y actualizar dentro de lo posible. Como ejemplo, podemos ver que para Ubuntu 16.04 salió Ubuntu 16.04.2, con algunas correcciones de seguridad. [Ver](#).

## Firewall

**Todo servidor debe tener activado el firewall.** El firewall permitirá denegar (o permitir) el acceso a la red. En un servidor web, el puerto abierto al público deberían ser sólo el 80 (http) y el 443 (https). Además de ese puerto, si la máquina es accedida remotamente mediante un servidor SSH, deberíamos abrir este puerto también, pero con un límite de acceso. La solución es fácilmente implementable con el programa [ufw](#).

## SSH

Los servidores ssh permiten el acceso al servidor remotamente. **No debe permitirse el acceso por ssh mediante usuario y password.** Sólo debe permitirse el acceso mediante clave publica. DigitalOcean tiene una buena guía de cómo configurar las claves públicas [Ver](#).

## Optimización de logging

### Configurar otro logging driver

Por default, docker escribe a un archivo con formato json, lo cual puede llevar a que se acumulen los logs de la aplicación, y estos archivos crezcan indefinidamente. Para evitar esto, se puede configurar el [logging driver](#) de docker. La recomendación es usar `journald` y configurarlo para que los logs sean persistentes.

### Eliminar logs antiguos de Docker

Dentro del normal funcionamiento de la plataforma, se genera una gran cantidad de logs, los cuales, ante una incidencia, son sumamente útiles. Pero, luego de un tiempo, y sabiendo que los mismos se almacenan internamente en Andino, podría ser necesario eliminarlos.

```
sudo su -c "ls /var/lib/docker/containers/ | xargs -n1 | while read
docker_id; do truncate -s 0
/var/lib/docker/containers/${docker_id%/*}/${docker_id%/*}-json.log;
done"
```

Eliminar logs dentro de Andino

```
docker-compose -f latest.yml exec portal truncate -s 0  
/var/log/apache2/*.log
```

## Configuración HTTPS

### Certificado SSL

Andino cuenta con soporte *builtin* de certificados SSL. Es posible instalar Andino con SSL, actualizar una versión de Andino sin SSL y agregarle el soporte para SSL, e inclusive deshabilitar SSL a una instancia con SSL.

Lo único que un administrador de Andino necesita para habilitar SSL es contar con los certificados `.key` y `.crt` para nuestra aplicación y la elección de un puerto libre del host para usar SSL.

Cómo obtener estos archivos está fuera del "scope" de esta documentación.

Si además de un `.crt` se posee un bundle, se debe incluir el contenido del certificado en dicho bundle (tiene que estar al principio, antes del texto ya existente).

Es recomendable asegurarse de que la key y el certificado sean compatibles. Esto se puede lograr de la siguiente manera: 1. Ejecutar `openssl rsa -noout -text -in {path_de_la_key}` 2. Ejecutar `openssl x509 -noout -text -in {path_del_cert}` 3. Asegurarse de que no haya habido ningún error durante los pasos anteriores 4. 1. Buscar el contenido de *modulus* en el resultado del paso 1 2. Buscar el contenido de *Certificate -> Signature Algorithm -> Subject Public Key Info -> Public Key Algorithm -> Modulus* en el resultado del paso 2 5. Comparar los contenidos encontrados Si los contenidos encontrados son idénticos, entonces la key y el certificado son compatibles.

## Configuración de SSL

Tanto la instalación como la actualización de un Andino emplean el uso de un parámetro llamado `nginx_ssl`, el cual puede ser utilizado para especificar que se desea utilizar SSL.

Para especificar el path de los archivos del certificado, se debe utilizar los parámetros `ssl_key_path` y `ssl_cert_path`. Los archivos dentro del contenedor nginx se llamarán *andino.key* y *andino.crt* respectivamente, y el proceso de instalación o actualización los copiará en el directorio `/etc/nginx/ssl`. En caso de que al menos uno de estos archivos no esté, *no se podrá utilizar el archivo de configuración para SSL* y se elegirá el default en su lugar. Hay que asegurarse de que el path de cada archivo sea válido (exista en el host), y que estén especificados en la instalación/actualización.

Un navegador web *no debería* mostrarnos ninguna advertencia si los certificados son correctos.

Un ejemplo de uso dentro del comando de instalación de Andino para los parámetros mencionados:

```
--nginx_ssl  
--ssl_key_path="/home/miusuario/Desktop/mi_archivo_key.key"  
--ssl_cert_path="/home/miusuario/Desktop/mi_archivo_cert.crt"
```

Estos parámetros de configuración deben ser especificados en cada actualización de Andino, para mantener SSL habilitado.

Modificar el puerto

Para la instalación de Andino, el puerto a ser utilizado por default es el 443, pero éste puede ser cambiado mediante el parámetro `nginx_ssl_port` y un valor a elección.

Ejemplo:

```
--nginx_ssl_port=8443
```

Es importante para los administradores saber que Andino tomará el puerto especificado (o el default) ya sea que el portal use o no use HTTPS. En caso de no querer usar HTTPS y que el host tenga el puerto 443 tomado por un servidor web, es requisito especificar un puerto distinto (ejemplo: 8443) que será reservado por Andino, pero no utilizado.

### Realizar cambios en un Andino instalado

Para lograr que Andino implemente la configuración HTTPS, es necesario realizar una actualización de andino y especificar las opciones detalladas en la sección Configuración de SSL.

Estas opciones solo son válidas a partir de Andino release-2.5.2.

### Probar la configuración

Para asegurarse de que Nginx esté utilizando la configuración HTTPS, ejecutar el siguiente comando debería mostrar `nginx_ssl.conf`:

```
docker exec -it andino-nginx bash -c 'echo $NGINX_CONFIG_FILE'.
```

Si se está implementando la configuración HTTPS y los certificados fueron creados correctamente, el explorador debería redirigir cualquier llamada HTTP a HTTPS.

Si se especificó un puerto para SSL, el portal debería permitir el ingreso si el puerto es parte de la URL.

También deberías poder navegar el portal en el puerto SSL seleccionado.

### Renovar certificados SSL

Para renovar los certificados SSL de tu instancia de Andino es tan sencillo como ejecutar una actualización de Andino. Para llevarlo a cabo es necesario que subas los dos archivos que componen el certificado (`.cer` y `.key`) y que ejecutes el comando de actualización de Andino, especificando la opción `--nginx_ssl` y las

opciones que permiten configurar los archivos del certificado como está especificado en la sección Configuración de SSL.

Si deseás mantener la versión de Andino que tenés, debés especificar la opción `--andino_version` con la versión de tu instancia de Andino.

## Deshabilitar SSL

El proceso de deshabilitación de SSL se puede lograr mediante la ejecución del proceso de actualización de Andino `update.py` especificando el parámetro `--andino_version` a una versión igual a la del portal a configurar (es decir, se mantendrá la misma versión), pero no especificando el parámetro `--nginx_ssl`.

De esta manera el script de actualización usará la configuración de Andino que no habilita HTTPS y se habilitará el acceso por el puerto 80.

Ejemplo de uso:

```
sudo wget
https://raw.githubusercontent.com/datosgobar/portal-andino/master/install/updat
e.py

sudo python update.py --andino_version=<versión del andino del
portal>
```

## Configuración DNS

Algunas funcionalidades del Portal Andino requieren que la aplicación web o procesos externos (ej: DataPusher) puedan navegar sin restricciones desde el proceso en el cual corre la aplicación Python a la URL donde está publicado el sitio (definida por el setting `ckan.site_url`).

Esta URL debe poder ser accedida sin problemas de resolución de nombres o de ruteo de IPs desde el mismo contenedor *portal* para el correcto funcionamiento del sitio.

Algunas instancias de Andino han reportado problemas de funcionamiento debido a que la infraestructura donde están alojados no permite esta resolución de nombres de manera correcta. Este artículo describe cómo diagnosticar el problema y propone soluciones al mismo.

Recordá que *todos los comandos de este artículo deben ser ejecutados en el directorio donde instalaste andino, por ejemplo /etc/portal.*

Cómo diagnosticar si tu andino tiene el problema

Para saber si es necesario realizar las configuraciones detalladas en este artículo, podés realizar los siguientes pasos detallados abajo.

1. Obtener el valor de `ckan.site_url`: el valor se puede obtener ejecutando el siguiente comando: `docker-compose -f latest.yml exec portal grep ckan\.site_url /etc/ckan/default/production.ini`. Ese comando debería devolver `ckan.site_url = <URL de tu Andino>`.
2. Evaluar si tu andino puede navegar hasta la URL del sitio. Ejecutá el siguiente comando: `docker-compose -f latest.yml exec portal curl <URL de tu Andino>/data.json`.

Si el segundo paso no devuelve una respuesta en formato *json* con la información del catálogo de tu instancia idéntica a la que obtendrías navegando desde tu navegador a `<URL de tu Andino>/data.json` (por ejemplo, luego de un rato obtenés `curl: (7) Failed to connect to <URL de tu Andino> port 80: Connection timed out`), entonces *debés aplicar la configuración recomendada en este artículo.*

Cómo resolver el problema

La solución del problema se basa en asegurar que dentro de la red de Docker el nombre de dominio de Andino pueda resolverse correctamente a la IP pública del host, a la IP privada dentro de la red en la que está o a la IP interna dentro de la red de Docker.

Para lograrlo, planteamos distintas alternativas.

### **Configuración de DNS públicos**

Este paso no está cubierto por esta documentación, ya que puede ser resuelto mediante formas diversas, dependiendo de la infraestructura con la que contás.

Lo que debés hacer es contactarte con tu administrador de infraestructura y solicitar una IP pública fija y un nombre de dominio para la instancia de tu andino.

Probablemente ya tengas un nombre de dominio asignado a tu instancia de andino, con lo cual podés saltar este paso.

Configurar andino con el nuevo nombre de dominio

Si ya tenés un nombre de dominio asignado para acceder a tu andino, y cuando lo instalaste lo configuraste usando ese nombre de dominio, podés saltar este paso.

Para configurar el nuevo nombre de dominio es necesario actualizar el setting `ckan.site_url` de la instancia de Andino. Esto lo podés lograr con el siguiente comando:

```
docker-compose -f latest.yml exec portal /etc/ckan_init.d/update_conf.sh  
"ckan.site_url=http://<tu nombre de dominio>"
```

Podés verificar que haya quedado bien configurado ejecutando:

```
docker-compose -f latest.yml exec portal grep ckan\.site_url  
/etc/ckan/default/production.ini
```

Para reflejar los cambios, es necesario reiniciar la aplicación web del contenedor `portal`:

```
docker-compose -f latest.yml exec portal apachectl restart
```

Finalmente, si ya tenías datos cargados en tu andino, necesitás regenerar el índice de búsqueda, usando el siguiente comando:

```
docker-compose -f latest.yml exec portal /etc/ckan_init.d/run_rebuild_search.sh
```

Configurar un alias en la red de Docker para el contenedor `nginx`

Si, aún teniendo un nombre de dominio asignado, tu `portal` no puede resolver el mismo a la IP pública del servidor, podés modificar la configuración de la red de Docker usada por Andino para mapear el nombre de dominio de tu instancia al contenedor `nginx`.

**Aclaración:** Esta configuración se realiza por defecto para todas las instancias de Andino desde la versión 2.5. Si tu instancia de Andino fue creada *antes de la versión 2.5*, seguramente quieras realizar estos pasos.

Para asegurarte de que la red interna de los contenedores de Docker que conforman Andino tienen la configuración necesaria para la correcta resolución de nombres, podés seguir los siguientes pasos (todos en el directorio de instalación de Andino, por ejemplo `/etc/portal`):

1. Editá el archivo `.env` y asegurate que el valor del atributo `SITE_HOST` tenga el valor del `hostname` de tu instancia de Andino, sin `http`. Si no encontrás una entrada para `SITE_HOST` en tu archivo `.env`, agregala al final. Por ejemplo debería ser `SITE_HOST=mi-andino.mi-ministerio.gob.ar`.



2. Descargá la última versión de latest.yml: `mv latest.yml latest.yml.bak && wget https://raw.githubusercontent.com/datosgobar/portal-andino/master/latest.yml`. Probablemente ya tengas esta misma versión si actualizaste a Andino 2.5, pero para asegurarnos de que tengas los últimos cambios necesarios para esta configuración es necesario realizar este paso.
3. Recreá el contenedor nginx: `docker-compose -f latest.yml up -d nginx`. Recordá que este paso puede generar algo de *downtime*, por lo que quizás sea prudente realizarlo en algún horario con poco tráfico en Andino.