Instalación de Docker y Docker Compose

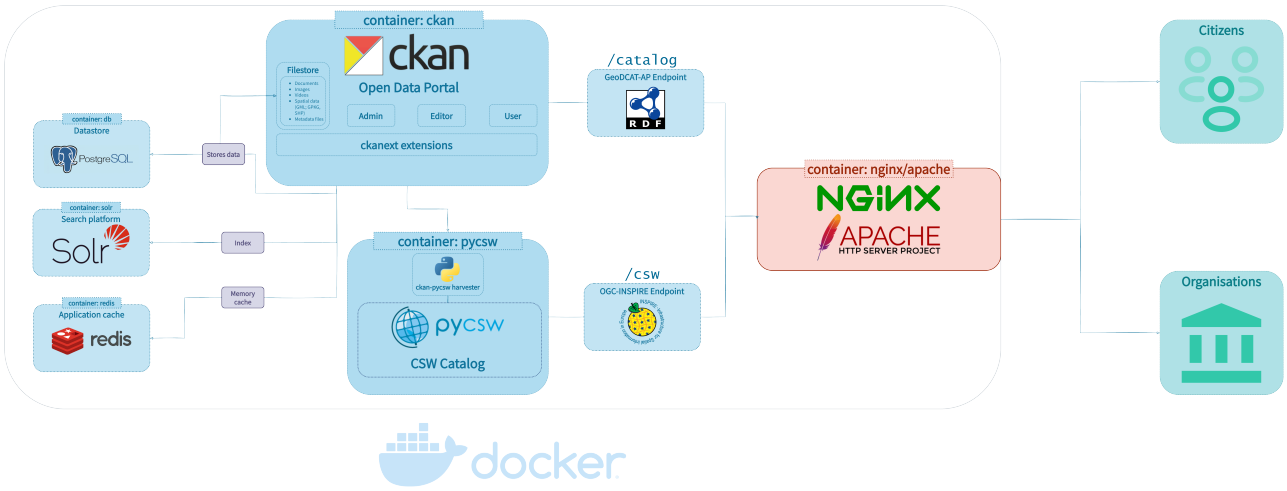
Visión general

Contiene imágenes Docker para los diferentes componentes de CKAN Cloud y un entorno Docker compose (basado en ckan) para el desarrollo y prueba de portales Open Data.

> **TIP**
>
> - Use the **deploy in 5 minutes** to see `ckan-docker` in **5 minutes** !
> - Or use Codespaces to test `ckan-docker` in your browser:
>
> Open in GitHub Codespaces

> **WARNING**
>
> Se trata de una **instalación personalizada de Docker Compose** con extensiones específicas para datos espaciales y metadatos GeoDCAT-AP/INSPIRE perfiles. Para las instalaciones oficiales, eche un vistazo: CKAN: Source installation.

Componentes disponibles



Componentes disponibles:

- CKAN custom multi-stage build con capacidades espaciales desde ckan-docker-spatial, una imagen utilizada como base y construida desde el repo oficial de CKAN. Están disponibles las siguientes versiones de CKAN:

CKAN Version	Type	Docker tag	Notes
2.9.8	custom image	`ghcr.io/mjanez/ckan-spatial:ckan-2.9.8`	Stable version with CKAN 2.9.8
2.9.9	custom image	`ghcr.io/mjanez/ckan-docker:ckan-2.9.9`	Stable version with CKAN 2.9.9
2.9.10	custom image	`ghcr.io/mjanez/ckan-docker:ckan-2.9.10`	Stable version with CKAN 2.9.10

CKAN Version	Type	Docker tag	Notes
2.9.11	custom image	`ghcr.io/mjanez/ckan-docker:ckan-2.9.11`	Stable version with CKAN 2.9.11
2.9.11	latest custom image	`ghcr.io/mjanez/ckan-docker:master`	Latest `ckan-docker` image.

Las imágenes que no son de CKAN son las siguientes:

- PostgreSQL: Imagen personalizada basada en la imagen oficial de PostgreSQL. Los archivos de base de datos se almacenan en un volumen con nombre.
- Solr: Imagen personalizada basada en la imagen oficial de CKAN imagen preconfigurada de Solr. Los datos de índice se almacenan en un volumen con nombre y tienen un esquema espacial personalizado actualizado. [^2]
- Redis: Imagen estándar de Redis
- NGINX: Última imagen estable de nginx que incluye puntos finales SSL y no SSL.
- ckan-pycsw: Imagen personalizada basada en pycsw CKAN harvester ISO19139 para INSPIRE Metadata CSW Endpoint.

Punto final HTTP opcional (`docker-compose.apache.yml`):

- `docker-compose.apache.yml`:
  - Servidor HTTP Apache: Custom image basado en la última imagen httpd estable oficial. Configurado para servir múltiples rutas para el punto final CSW ckan-pycsw (`{CKAN_SITE_URL}/csw`) y CKAN

(`{CKAN_SITE_URL}/catalog`). Sólo HTTP.

El sitio se configura utilizando variables de entorno que puede establecer en el archivo `.env` para un despliegue NGINX y ckan-pycsw (por defecto `.env.example`), o sustituirlo por el archivo `.env.apache.example` para un despliegue Apache HTTP Server utilizando el archivo Docker Compose: `docker-compose.apache.yml`.

Requisitos

`ckan-docker` consiste en un conjunto de contenedores desplegados con docker y `docker compose`.

Prerrequisitos

docker compose *vs* docker-compose

Todos los comandos de Docker Compose en este documento utilizarán la versión V2 de Compose, es decir: `docker compose`. La versión anterior (V1) utilizaba el comando `docker-compose`. Por favor, consulte Docker Compose para más información.

Instalar docker-engine

Siga las instrucciones de instalación de su entorno para instalar Docker Engine.

Para verificar que la instalación de Docker se ha realizado correctamente, ejecute `docker run hello-world` y `docker version`. Estos comandos deberían mostrar para el cliente y el servidor.

Docker/Docker Compose más información

Más información sobre Docker/Docker Compose comandos básicos.

CKAN: Instalación y configuración

Clonar y configurar `.env`

El sitio se configura usando variables de entorno que puedes establecer en el archivo `.env`.

1. Clonar proyecto

```
cd /ruta/mi/proyecto
git clone https://github.com/mjanez/ckan-docker.git & cd ckan-docker
```

2. Copia la plantilla `.env.example` y modifica el `.env` resultante para adaptarlo a tus necesidades.

```
cp .env.ejemplo .env
```

NGINX | **Apache HTTP Server**

Modify the `.env` variables as needed:

```
# Host Ports
CKAN_PORT_HOST=5000
NGINX_PORT_HOST=81
NGINX_SSLPORT_HOST=8443
APACHE_PORT_HOST=81
PYCSW_PORT_HOST=8000

...

#NGINX/APACHE
## Comprueba CKAN__ROOT_PATH y CKANEXT__DCAT__BASE_URI y CKANEXT__SCHEMINGDCAT_GEOMETADATA_BASE_URI. Si no necesita utilizar ubicaciones de dominio, es mejor utilizar la configuración de nginx. Dejar en blanco o utilizar la raíz `/`.
PROXY_SERVER_NAME=localhost
PROXY_CKAN_LOCATION=/catalog
PROXY_PYCSW_LOCATION=/csw

...

# CKAN_SITE_URL = http:/ o https:/ + PROXY_SERVER_NAME. Opcionalmente el APACHE_HOST_PORT si es diferente de 80
CKAN_SITE_URL=http://localhost:81
CKAN__ROOT_PATH=/catalog/{{LANG}}
CKAN_PORT=5000
CKAN__FAVICON=/catalog/base/images/ckan.ico
CKAN__SITE_LOGO=/images/default/ckan-logo.png
```

Modify the `.env` variables as needed:

```
# Host Ports
CKAN_PORT_HOST=5000
NGINX_PORT_HOST=81
```

> **WARNING**
>
> Usando los valores por defecto en el fichero `.env` obtendrá una instancia de CKAN funcionando. Hay un usuario sysadmin creado por defecto con los valores definidos en `CKAN_SYSADMIN_NAME` y `CKAN_SYSADMIN_PASSWORD` (`ckan_admin` y `test1234` por defecto). Todos los ennvars con `API_TOKEN` se regeneran automáticamente cuando se carga CKAN, no es necesario editarlos.
>
> **Esto debe ser obviamente cambiado antes de ejecutar esta configuración como una instancia pública de CKAN.

Ahora está listo para proceder con el despliegue.

Instalar (compilar y ejecutar) CKAN más dependencias

Modo base

Utilice este modo si es un mantenedor y no va a realizar cambios en el código de CKAN o en las extensiones de CKAN.

1. Construye las imágenes:
   ```
   docker compose build
   ```
2. Arranca los contenedores:
   ```
   docker compose up
   ```

Esto iniciará los contenedores en la ventana actual. Por defecto los contenedores se registrarán directamente en esta ventana con cada contenedor utilizando un color diferente. También puede utilizar la opción -d "detach mode" es decir: `docker compose up -d` si desea utilizar la ventana actual para otra cosa. para otra cosa.

> **TIP**
>
> - O construir y desplegar hasta los contenedores:
>   ```
>   docker compose up -d --build
>   ```
> - O utiliza la versión con Apache HTTP Web Server:
>   ```
>   docker compose -f docker-compose.apache.yml up -d --build
>   ```

Aprende más sobre la configuración de este `ckan-docker`

- Copia de seguridad de la base de datos CKAN
- Configurar un servicio docker-compose para que se inicie al arrancar

Modo rápido

Si sólo quieres probar el paquete y ver la funcionalidad general de la plataforma, puedes utilizar la imagen `ckan-docker` del registro de contenedores de Github:

```
# Edita los envvars en el .env a tu gusto e inicia los contenedores.
docker compose -f docker-compose.ghcr.yml up -d --build
```

> **INFO**
> Descargará la imagen pre-construida y desplegará todos los contenedores. Recuerda usar tu propio dominio cambiando `localhost` en el archivo `.env`.

Modo Desarrollo

Utilice este modo si está realizando cambios en el código de CKAN y creando nuevas extensiones o realizando cambios en el código de extensiones existentes. Este modo también utiliza el archivo `.env` para las opciones de configuración.

Para desarrollar extensiones locales utilice el archivo `docker compose.dev.yml`:

Para construir las imágenes:

```
docker compose -f docker-compose.dev.yml build
```

Para poner en marcha los contenedores:

```
docker compose -f docker-compose.dev.yml up
```

> **TIP**
> Consulte CKAN images para obtener más detalles sobre lo que ocurre cuando se utiliza el modo de desarrollo.

Comprobar que los contenedores están en marcha

Al final de la secuencia de inicio de contenedores debería haber 6 contenedores ejecutándose.

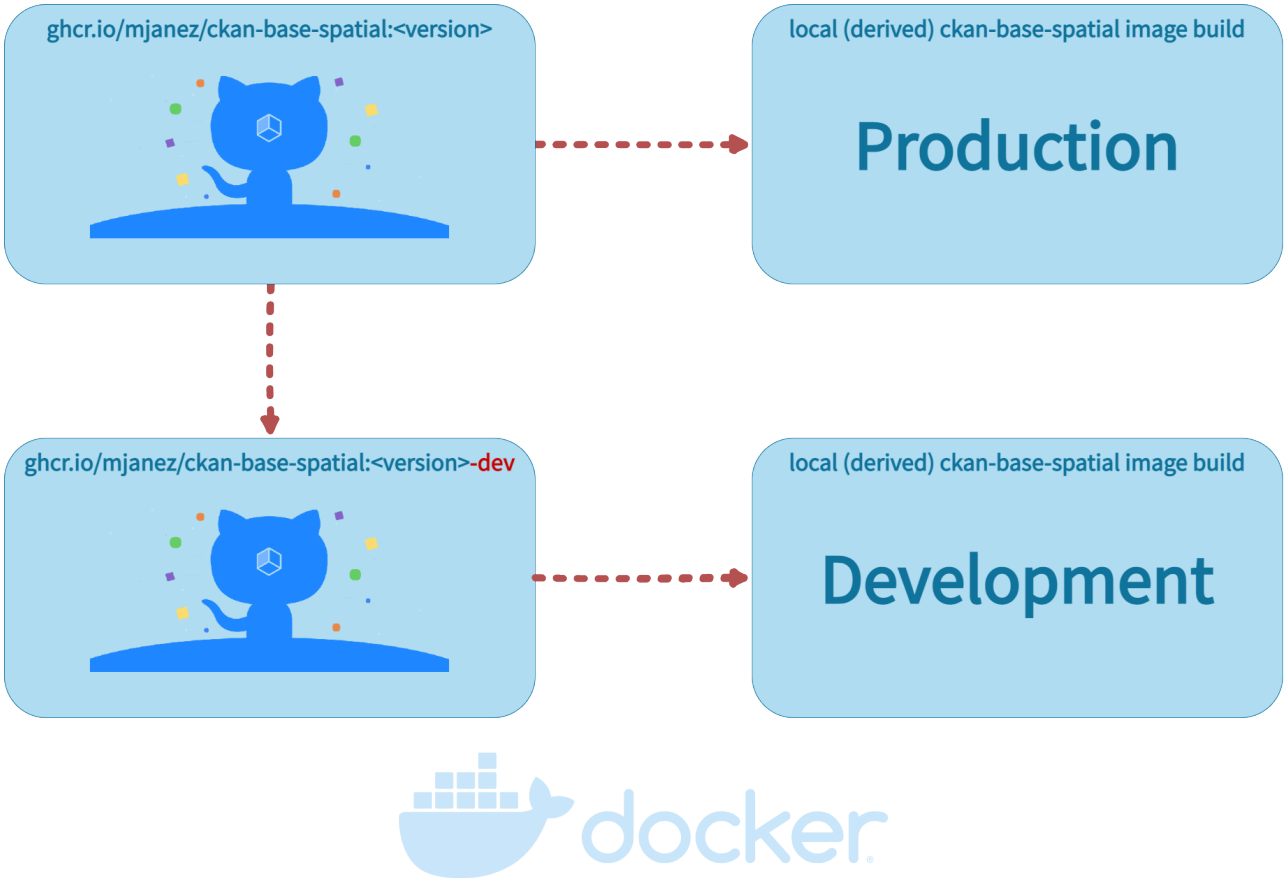
Después de este paso, CKAN debería estar ejecutándose en `http://{PROXY_SERVER_NAME}/{PROXY_CKAN_LOCATION}` y ckan-pycsw en `http://{PROXY_SERVER_NAME}/{PROXY_PYCSW_LOCATION}`, es decir: http://localhost/catalog o http://localhost/csw.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
0217537f717e	ckan-docker-nginx/ ckan-docker-apache	/docker-entrypoint.…	6 minutes ago	Up 4 minutes	80/tcp,0.0.0.0:80->80/ tcp,0.0.0.0:8443->443/ tcp	ckan-docker-nginx-1/ ckan-

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
						docker-apache-1
7b06ab2e060a	ckan-docker-ckan	/srv/app/start_ckan…	6 minutes ago	Up 5 minutes (healthy)	0.0.0.0:5000->5000/tcp	ckan-docker-ckan-1
1b8d9789c29a	redis:7-alpine	docker-entrypoint.s…	6 minutes ago	Up 4 minutes (healthy)	6379/tcp	ckan-docker-redis-1
7f162741254d	ckan/ckan-solr:2.9-solr9-spatial	docker-entrypoint.s…	6 minutes ago	Up 4 minutes (healthy)	8983/tcp	ckan-docker-solr-1
2cdd25cea0de	ckan-docker-db	docker-entrypoint.s…	6 minutes ago	Up 4 minutes (healthy)	5432/tcp	ckan-docker-db-1
9cdj25dae6gr	ckan-docker-pycsw	docker-entrypoint.s…	6 minutes ago	Up 4 minutes (healthy)	8000/tcp	ckan-docker-pycsw-1

ckan-docker: Mejoras

Imágenes CKAN



Los archivos de configuración de la imagen Docker utilizados para construir tu proyecto CKAN se encuentran en la carpeta `ckan/`. Hay dos archivos Docker:

- `Dockerfile`: está basado en `mjanez/ckan-base-spatial:<version>`, una imagen base localizada en el Github Package Registry, que tiene instalado CKAN junto con todas sus dependencias, correctamente configurado y ejecutándose en uWSGI (configuración de producción).

- `Dockerfile.dev`: está basado en `mjanez/ckan-base-spatial:<version>-dev` también localizado en el Registro de Paquetes de Github, y extiende `mjanez/ckan-base-spatial:<version>` para incluir:
    - Cualquier extensión clonada en la carpeta `./src` se instalará en el contenedor CKAN al arrancar Docker Compose (`docker compose up`). Esto incluye instalar cualquier requisito listado en un archivo `requirements.txt` (o `pip-requirements.txt`) y ejecutar `python setup.py develop`.
    - CKAN se inicia ejecutando esto: `/usr/bin/ckan -c /srv/app/ckan.ini run -H 0.0.0.0`.
    - Asegúrese de añadir los plugins locales a la var env `CKAN__PLUGINS` en el archivo `.env`.
- Cualquier cambio personalizado en los scripts que se ejecutan durante el arranque del contenedor puede hacerse en los scripts del directorio `setup/`. Por ejemplo, si quisieras cambiar el puerto en el que se ejecuta CKAN, tendrías que hacer cambios en el archivo yaml de Docker Compose y en el archivo `start_ckan.sh.override`. A continuación, tendría que añadir la siguiente línea a la Dockerfile es decir: `COPY setup/start_ckan.sh.override ${APP_DIR}/start_ckan.sh`. El archivo `start_ckan.sh` de la imagen local sustituirá al archivo `start_ckan.sh` incluido en la imagen base.

> **TIP**
>
> Si obtienes un error como `doesn't have execute permissions`:
>
> ```
> Daemon error response: failed to create shim task: OCI runtime
> create failed: runc create failed: unable to start container
> ```

Puede ser necesario dar permisos de ejecución al fichero en el `Dockerfile`:

```
...
# Anular start_ckan.sh
COPIAR setup/start_ckan.sh.override ${APP_DIR}/start_ckan.sh
EJECUTAR chmod +x ${APP_DIR}/start_ckan.sh
...
```

Mejora de las imágenes CKAN

Ampliación de las imágenes base

Puedes modificar los archivos docker para construir tu propia imagen personalizada adaptada a tu proyecto, instalando cualquier extensión o requisito extra que necesites. Por ejemplo, aquí es donde se actualizaría para utilizar una imagen base CKAN diferente, es decir: `ckan/ckan-base-spatial:<nueva versión>`.

Para realizar pasos extra de inicialización puedes añadir scripts a tus imágenes personalizadas y copiarlos a la carpeta `/docker-entrypoint.d` (La carpeta debería crearse para ti cuando construyas la imagen). Cualquier archivo `*.sh` y `*.py` de esa carpeta se ejecutará justo después de que se ejecute el script de inicialización principal (`prerun.py`) y justo antes de que se inicien los procesos del servidor web y del supervisor.

Por ejemplo, considere la siguiente imagen personalizada:

```
ckan
```

Queremos instalar una extensión como ckanext-validation que necesita crear tablas de base de datos en tiempo de arranque. Creamos un script `setup_validation.sh` en una carpeta `docker-entrypoint.d` con los comandos necesarios:

```
#!/bin/bash

# Create DB tables if not there
ckan -c /srv/app/ckan.ini validation init-db
```

Y luego en nuestro archivo `Dockerfile.dev` instalamos la extensión y copiamos los scripts de inicialización:

```
FROM ckan/ckan-base-spatial:2.9.11

RUN pip install -e git+https://github.com/frictionlessdata/ckanext-validation.git#egg=ckanext-validation && \
    pip install -r https://raw.githubusercontent.com/frictionlessdata/ckanext-validation/master/requirements.txt

COPY docker-entrypoint.d/* /docker-entrypoint.d/
```

> **TIP**
>
> Hay una serie de ejemplos de ampliación comentados en el archivo `Dockerfile.dev`

Aplicación de parches

Cuando construyas las imágenes CKAN específicas de tu proyecto (las definidas en la carpeta `ckan/`), puedes aplicar parches al núcleo de CKAN o a cualquiera de las extensiones creadas. Para ello, cree una carpeta dentro de

`ckan/patches` con el nombre del paquete a parchear (es decir, `ckan/`). paquete a parchear (por ejemplo `ckan` o `ckanext-??`). Dentro puede colocar los archivos de parche que se aplicarán al construir las imágenes. Los parches se aplicarán en orden alfabético, por lo que puede anteponerles un prefijo secuencial si es necesario.

Por ejemplo, vea la siguiente carpeta de imágenes de ejemplo:

```
ckan
├── patches
│   ├── ckan
│   │   ├── 01_datasets_per_page.patch
│   │   ├── 02_groups_per_page.patch
│   │   ├── 03_or_filters.patch
│   └── ckanext-harvest
│       └── 01_resubmit_objects.patch
├── setup
├── Dockerfile
└── Dockerfile.dev
```

INFO

Git diff es un comando para mostrar los cambios entre dos fuentes dentro del repositorio Git. Las fuentes de datos pueden ser dos ramas diferentes, commits, ficheros, etc.

- Muestra los cambios entre el directorio de trabajo y el área de preparación:

  ```
  git diff > [archivo.patch]
  ```

- Muestra los cambios entre el área de preparación y el repositorio:

```
git diff --staged [archivo]`
```

Aplicación de parches en modo desarrollo

Para aplicar parches en modo desarrollo, deberá seguir los siguientes pasos:

1. Asegúrese de que sus parches están ubicados en el directorio `ckan/patches`. Los parches deben organizarse en subdirectorios con el nombre del paquete al que van destinados (por ejemplo, `ckan` o `ckanext-??`). Cada archivo de parche debe terminar con la extensión .patch.

   Por ejemplo, la estructura de directorios podría ser la siguiente:

   ```
   ckan
   ├── patches
   │   ├── ckan
   │   │   ├── 01_datasets_per_page.patch
   │   │   ├── 02_groups_per_page.patch
   │   │   ├── 03_or_filters.patch
   │   └── ckanext-harvest
   │       └── 01_resubmit_objects.patch
   ├── setup
   ├── Dockerfile
   └── Dockerfile.dev
   ```

2. Navegue hasta el directorio `/src`.

3. Aplique los parches utilizando el comando patch:

   ```
   find /path/to/ckan/patches -name '*.patch' -exec patch -p1 < {} \;
   ```

Este comando buscará recursivamente en el directorio `/path/to/ckan/patches` archivos que terminen en `.patch` y los aplicará usando el comando patch. Sustituya `ruta/al/ckan/parches` por la ruta real a su directorio `ckan/parches`.

Crear una extensión en modo desarrollo

Puedes usar las instrucciones ckan extension para crear una extensión CKAN, sólo ejecutando el comando dentro del contenedor CKAN y poniendo como salida la carpeta `src/` montada:

```
docker compose -f docker-compose.dev.yml exec ckan-dev /bin/sh -c "ckan -c /srv/app/ckan.ini generate extension --output-dir /srv/app/src_extensions"
```

A continuación, responda a las preguntas para configurar el plugin:

```
Extension's name [must begin 'ckanext-']: ckanext-newextension
Author's name []: Joe Bloggs
Author's email []: joe@bloggs.com
Your Github user or organization name []: joebloggs
Brief description of the project []: test creating a new extension
List of keywords (separated by spaces) [CKAN]: ckanext-newextension
Do you want to include code examples? [y/N]: y

Written: /srv/app/src_extensions/ckanext-newextension
```

Los nuevos archivos y directorios de extensión se crean en la carpeta `/srv/app/src_extensions/` en el contenedor en ejecución. También existirán en el directorio local src/ ya que el directorio local `/src` está montado como `/srv/`

`app/src_extensions/` en el contenedor ckan. Es posible que tenga que cambiar el propietario de su carpeta para tener los permisos adecuados.

ckan-docker Complementos

Depuración

Depuración de la Instancia de Desarrollo CKAN con VSCode Dev Containers y debugpy

La extensión Visual Studio Code Dev Containers es una potente herramienta que permite a los desarrolladores utilizar un contenedor como entorno de desarrollo completo. Con esta extensión, los desarrolladores pueden abrir cualquier carpeta dentro de un contenedor y aprovechar todas las funciones que ofrece Visual Studio Code. Para ello, los desarrolladores crean un archivo `devcontainer.json` en su proyecto que especifica cómo acceder o crear un contenedor de desarrollo con una pila predefinida de herramientas y tiempo de ejecución. Esto permite a los desarrolladores trabajar en un entorno aislado, asegurando que el entorno de desarrollo es consistente entre los miembros del equipo y que las dependencias del proyecto son fáciles de gestionar.

Desarrollo dentro de un contenedor](https://code.visualstudio.com/assets/docs/devcontainers/containers/architecture-containers.png)

Para configurarlo

1. Instale VSCode.

2. Instala la extensión Remote Development extension para VSCode.

3. En el directorio de tu proyecto, crea un archivo `devcontainer.json`. Este

archivo contendrá la configuración para tu contenedor de desarrollo.

4. En el archivo `devcontainer.json`, especifica la imagen Docker para tu contenedor de desarrollo y cualquier ajuste de configuración adicional, como variables de entorno, puertos a exponer y comandos de inicio.

5. Habilita `debugpy` para tu instancia de desarrollo en tu archivo `.env`:

```
USE_DEBUGPY_FOR_DEV=true
```

6. Arranca los contenedores en modo desarrollo e inicia VS Code.

7. Instala la extensión "Dev Container": pulsa `CTRL+SHIFT+X`, escribe "dev container", pulsa "install".

8. Pulse el botón `Open a Remote Window` en la parte inferior izquierda de la ventana de VS Code.

9. Pulsa `Attach to Running Container...` y selecciona tu contenedor ckan-dev, por ejemplo `ckan-docker-ckan-dev-1`.

10. Haz click en el icono `Run and Debug` en el panel izquierdo y luego `create a launch.json`, selecciona `Python Debugger`, `Remote Attach`, host `localhost` y puerto `5678`.

11. Pulsa `F5` o haz click en el menú `Run` y `Start Debugging`.

Ahora puedes establecer puntos de interrupción y depurar remotamente tu instancia de desarrollo CKAN usando VSCode Dev Containers y debugpy.

pdb

Añada estas líneas al servicio `ckan-dev` en el archivo docker compose.dev.yml

```
ports:
  - "0.0.0.0:${CKAN_PORT}:5000"

stdin_open: true
tty: true
```

Depurar con pdb (ejemplo) - Interactuar con `docker attach $(docker container ls -qf name=ckan)`

comando: `python -m pdb /usr/lib/ckan/venv/bin/ckan --config /srv/app/ckan.ini run --host 0.0.0.0 --passthrough-errors`

Reverse proxy

NGINX

La configuración por defecto de Docker Compose (`docker-compose.yml`) utiliza una imagen NGINX como front-end (es decir: proxy inverso). Incluye HTTPS ejecutándose en el puerto número 8443 y un puerto HTTP (81). Se genera previamente un certificado SSL "autofirmado" y se incluyen los archivos de certificado y clave del servidor. La directiva `server_name` de NGINX (ENV: `PROXY_SERVER_NAME`) y el campo `CN` del certificado SSL se han establecido en 'localhost'. Obviamente, esto no debería utilizarse en producción.

Las ubicaciones del proxy, los puertos y otras opciones de NGINX pueden modificarse en el archivo `.env`:

```
# Puertos de host
NGINX_PORT_HOST=81
NGINX_SSLPORT_HOST=8443

# NGINX
NGINX_PORT=80
NGINX_SSLPORT=443
NGINX_LOG_DIR=/var/log/nginx

# Comprueba CKAN__ROOT_PATH y CKANEXT__DCAT__BASE_URI. Si no necesita
utilizar ubicaciones de dominio, es mejor utilizar la configuración
de nginx. Dejar en blanco o utilizar la raíz `/`.
PROXY_SERVER_NAME=localhost
PROXY_CKAN_LOCATION=/catalog
PROXY_PYCSW_LOCATION=/csw
```

La configuración básica de Docker Compose utiliza una imagen NGINX como front-end (es decir, proxy inverso). Incluye HTTPS ejecutándose en el puerto número 8443. Se genera un certificado SSL "autofirmado" como parte del ENTRYPOINT. La directiva ENV `PROXY_SERVER_NAME`, NGINX `server_name` y el campo `CN` en el certificado SSL se han establecido en 'localhost'. Obviamente, esto no debería utilizarse en producción.

Crear el certificado SSL y los archivos de claves de la siguiente manera: `openssl req -new -newkey rsa:4096 -days 365 -nodes -x509 -subj "/C=DE/ST=Berlin/L=Berlin/O=None/CN=localhost" -keyout ckan-local.key -out ckan-local.crt`. A continuación, los archivos `ckan-local.*` deberán trasladarse al directorio nginx/setup/

Apache HTTP Server

La configuración de Docker Compose (`docker-compose.apache.yml`) utiliza una imagen httpd como front-end. Tiene dos rutas para los servicios ckan (ubicación por defecto: `/catalog`) y ckan-pycsw (ubicación por defecto: `/csw`).

Las ubicaciones del proxy, los puertos y otras opciones del Servidor Web Apache pueden modificarse en el archivo `.env`:

```
# Puertos de host
APACHE_PORT_HOST=81

# Apache HTTP Server
APACHE_VERSION=2.4-alpine
APACHE_PORT=80
APACHE_LOG_DIR=/var/log/apache

# Comprueba CKAN__ROOT_PATH y CKANEXT__DCAT__BASE_URI. Si no necesita
utilizar ubicaciones de dominio, es mejor utilizar la configuración
de nginx. Dejar en blanco o utilizar la raíz `/`.
PROXY_SERVER_NAME=localhost
PROXY_CKAN_LOCATION=/catalog
PROXY_PYCSW_LOCATION=/csw
```

envvars

La extensión ckanext-envvars se utiliza en el repositorio base Docker de CKAN para construir las imágenes base. Esta extensión comprueba si las variables de entorno se ajustan a un formato esperado y actualiza la configuración CKAN correspondiente con su valor.

Para que la extensión identifique correctamente qué claves env var se corresponden con el formato utilizado para el objeto config, las claves env var deben tener el siguiente formato:

Todo en mayúsculas
Sustituya los puntos ('.') por dos guiones bajos ('__')
Las claves deben empezar por 'CKAN' o 'CKANEXT', si no es así, puede anteponer '`CKAN___`'.

Por ejemplo:

- `CKAN__PLUGINS="envvars image_view text_view recline_view datastore datapusher"`
- `CKAN__DATAPUSHER__CALLBACK_URL_BASE=http://ckan:5000`
- `CKAN___BEAKER__SESSION__SECRET=CHANGE_ME`

Estos parámetros se pueden añadir al archivo `.env`.

Para más información, consulte ckanext-envvars

> ⚠ **PRECAUCIÓN**
>
> Si el despliegue se encuentra bajo un proxy, como en un entorno corporativo, para evitar errores al resolver urls con container_names/ hostnames asociados al contenedor en redes internas, debe utilizarse la variable `no_proxy`, en minusculas, con los nombres de los servicios/ contenedores, la ip de la red de docker, etc.
>
> ej:
>
> ```
> no_proxy="127.0.0.1,192.168.192.0/23,172.0.0.0/
> 0,redis,solr,${DB_CONTAINER_NAME}"`
> ```

Datastore

La base de datos Datastore y el usuario se crean como parte de los scripts de entrada para el contenedor db.

xloader

Este despliegue reemplaza DataPusher con XLoader usando Supervisor, más información sobre otras alternativas en la página wiki para esto: https://github.com/ckan/ckan-docker/wiki/Replacing-DataPusher-with-XLoader

ckan-pycsw

ckan-pycsw es un entorno docker compose (basado en pycsw) para desarrollo y pruebas con portales CKAN Open Data.

Componentes disponibles:

- **pycsw**: La aplicación pycsw. Una implementación del servidor OARec y OGC CSW escrita en Python.
- **ckan2pycsw**: Software para lograr la interoperabilidad con los portales de datos abiertos basados en CKAN. Para ello, ckan2pycsw lee datos de una instancia utilizando la API CKAN, genera metadatos ISO-19115/ISO-19139 utilizando pygeometa, o un esquema personalizado que se basa en un esquema CKAN personalizado, y rellena una instancia pycsw que expone los metadatos utilizando CSW y OAI-PMH.

Consumidores de cosecha en un CKAN desplegado

ckanext-harvest supervisor permite recolectar metadatos de múltiples fuentes en un despliegue de producción. Aquí está desplegado por un consumidor trabajador en el contenedor `ckan`, también la extensión `ckanext-harvest` y

otros recolectores personalizados (`ckanext-schemingdcat` o `ckanext-dcat`) están incluidos en las imágenes docker de CKAN.

Para habilitar los harvesters necesitas configurar en el archivo `.env` la variable `CKAN__PLUGINS` con el plugin `harvest`: https://github.com/mjanez/ckan-docker/blob/a18e0c80d9f16b6d9b6471e3148d48fcb83712bd/.env.example#L126-L127

ckan-docker: Consejos y trucos

CKAN. Copias de seguridad

PostgreSQL ofrece las herramientas de línea de comandos `pg_dump` y `pg_restore` para volcar y restaurar una base de datos y su contenido a/desde un archivo.

Servicio de copia de seguridad para el contenedor db

1. Crea un nuevo archivo llamado `ckan_backup_custom.sh` y ábrelo en tu editor de texto preferido.

2. ii. Añade el siguiente código al script, sustituyendo los marcadores de posición por tus valores reales:

```
#!/bin/bash

# Establecer las variables necesarias
CONTAINER_NAME="db"
DATABASE_NAME="ckandb"
POSTGRES_USER="postgres"
POSTGRES_PASSWORD="your_postgres_password"
BACKUP_DIRECTORY="/path/to/your/backup/directory"
DATE=$(date +%Y%m%d%H%M%S)
```

3. Sustituya los siguientes marcadores de posición por sus valores reales:
    - `your_postgres_password`: La contraseña del usuario PostgreSQL.
    - `/path/to/your/backup/directory`: La ruta al directorio donde desea almacenar los archivos de copia de seguridad.

> **WARNING**
> Si ha cambiado los valores del contenedor PostgreSQL, la base de datos o el usuario, cámbielos también. Compruebe que el paquete `zip` está instalado: `sudo apt-get install zip`.

4. Guarde y cierre el archivo.

5. Haz ejecutable el script:

```
chmod +x ckan_backup_custom.sh
```

6. Abre el `crontab` para el usuario actual:

```
crontab -e
```

7. Añade la siguiente línea para programar la copia de seguridad para que se ejecute diariamente a medianoche (ajusta la programación según sea necesario):

```
0 0 * * * /path/to/your/script/ckan_backup_custom.sh
```

> **NOTE**

> Sustituye `/ruta/a/tu/script` por la ruta real al script `ckan_backup_custom.sh`.

8. Guarde y cierre el archivo.

El cronjob está ahora configurado y realizará copias de seguridad de su base de datos PostgreSQL CKAN diariamente a medianoche utilizando el formato personalizado. Las copias de seguridad se almacenarán en el directorio especificado con la marca de tiempo en el nombre del archivo.

> **NOTE**
>
> Scripts de ejemplo para realizar copias de seguridad de CKAN: `doc/scripts`

Restaurar una copia de seguridad

Si necesita utilizar una copia de seguridad, restáurela:

1. Primero limpie la base de datos. **Precaución, esto borrará todos los datos de tu base de datos CKAN.

```
docker exec -it ckan /bin/bash -c "export TERM=xterm; exec bash"

# Borrar todo en la base de datos CKAN, incluyendo las tablas,
para empezar de cero
ckan -c $CKAN_INI db clean
```

2. Después de limpiar la base de datos debes hacer inicializarla o importar un volcado creado previamente.

```
docker exec -i -e PGPASSWORD=$POSTGRES_PASSWORD
$POSTGRESQL_CONTAINER_NAME pg_restore -U $POSTGRES_USER --clean --
if-exists -d $DATABASE_NAME < /path/to/your/backup/directory/
ckan.dump
```

3. Reinicie el contenedor `ckan`.

CKAN. Gestionar nuevos usuarios

- Crea un nuevo usuario desde el host Docker, por ejemplo para crear un nuevo usuario llamado `user_example`

  ```
  docker exec -it <container-id> ckan -c ckan.ini user add
  user_example email=user_example@localhost

  # Usuario administrador
  docker exec -it <container-id> ckan -c ckan.ini sysadmin add
  admin_example email=admin_example@localhost name=admin_example
  ```

  Para eliminar el usuario 'user_example

  ```
  docker exec -it <container-id> ckan -c ckan.ini user remove
  user_example`
  ```

- Crea un nuevo usuario desde el contenedor ckan. Necesitarás obtener una sesión en el contenedor en ejecución

  ```
  ckan -c ckan.ini user add usuario_ejemplo
  email=usuario_ejemplo@localhost`
  ```

  Para eliminar el usuario 'user_example

```
ckan -c ckan.ini user remove usuario_ejemplo`
```

Docker Compose. Configurar un servicio docker compose para que se inicie al arrancar

Para que Docker Compose se ejecute automáticamente al reiniciar una máquina, puedes seguir los siguientes pasos:

1. Crea un archivo de servicio systemd para Docker Compose. Puede crear un archivo llamado `ckan-docker-compose.service` en la carpeta `/etc/systemd/system/` con el siguiente contenido:

   ```
   [Unit]
   Description=CKAN Docker Compose Application Service
   Requires=docker.service
   After=docker.service

   [Service]
   User=docker
   Group=docker
   Type=oneshot
   RemainAfterExit=yes
   WorkingDirectory=/path/to/project/ckan-docker/
   ExecStart=/bin/docker compose up -d
   ExecStop=/bin/docker compose down
   TimeoutStartSec=0

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

2. Sustituye `/path/to/project/ckan-docker/` por la ruta donde se encuentra el fichero `docker-compose.yml` de tu proyecto y y comprueba la ruta al binario de docker compose en ejecución y parada: `/bin/docker`. Cambia

también el `User` / `Group` para ejecutar el servicio.

3. Carga el archivo de servicio systemd con el siguiente comando:

```
sudo systemctl daemon-reload
```

4. Permite que el servicio se inicie automáticamente al arrancar la máquina:

```
sudo systemctl enable ckan-docker-compose
```

5. Ahora puede iniciar el servicio con el siguiente comando:

```
sudo systemctl start ckan-docker-compose
```

6. Si desea detener o comprobar el estado del servicio, utilice los siguientes comandos:

```
# Detener el servicio
sudo systemctl stop ckan-docker-compose

# Comprobar el estado
sudo systemctl status ckan-docker-compose
```

Instalación de Ansible

Resumen

CKAN es una poderosa plataforma de datos abiertos que proporciona un conjunto de herramientas para facilitar la publicación, compartición, búsqueda y uso de datos. Este documento proporciona una guía paso a paso para utilizar un playbook de Ansible para el despliegue de un CKAN personalizado para la gestión de datos espaciales en diferentes entornos.

ckan-ansible

`ckan-ansible` es un playbook personalizado de Ansible para desplegar CKAN en diferentes entornos. El playbook está diseñado para automatizar la instalación y configuración de CKAN, PostgreSQL, Solr, Redis y NGINX. El playbook está diseñado para ser flexible y personalizable, permitiéndote configurar el despliegue para adaptarlo a tus requisitos específicos.

Despliegues disponibles para los siguientes sistemas operativos:

- RedHat Enterprise Linux 9 (RHEL 9)
- RedHat Enterprise Linux 8 (RHEL 8)
- [WIP] Debian 12 (Bookworm)
- [WIP] Ubuntu 20.04 (Focal Fossa)

> ⚠ **WARNING**
>
> Esta es una **instalación personalizada** con extensiones específicas para datos espaciales y metadatos GeoDCAT-AP/INSPIRE perfiles. Para

> instalaciones oficiales, consulte CKAN Documentación: Instalación.

Componentes disponibles:

CKAN es una plataforma líder mundial de portal de datos de código abierto que proporciona un conjunto de herramientas para agilizar la publicación, compartición, búsqueda y uso de datos. CKAN es una solución de software completa lista para usar que hace que los datos sean accesibles y utilizables, proporcionando herramientas para agilizar la publicación, compartición, búsqueda y uso de datos (incluyendo el almacenamiento de datos y la provisión de robustas APIs de datos). CKAN está dirigido a los publicadores de datos (gobiernos nacionales y regionales, empresas y organizaciones) que quieren hacer sus datos abiertos y disponibles.

Los componentes no-CKAN son los siguientes:

- PostgreSQL: Un poderoso sistema de base de datos objeto-relacional de código abierto. CKAN utiliza PostgreSQL para almacenar sus datos. En una instalación de CKAN, necesitarías configurar una base de datos PostgreSQL y configurar CKAN para usarla.
- Solr: Una popular plataforma de búsqueda de código abierto del proyecto Apache Lucene. CKAN utiliza Solr como su motor de búsqueda. Cuando creas o actualizas conjuntos de datos, recursos u otros objetos en CKAN, actualiza el índice de Solr. Luego, cuando buscas en CKAN, consulta a Solr y muestra los resultados. Los datos del índice de Solr se almacenan en un volumen nombrado, similar a los datos de PostgreSQL. En este despliegue, Solr tiene un esquema espacial personalizado que permite búsquedas geográficas.
- Redis: Una tienda de estructura de datos en memoria de código abierto, utilizada como base de datos, caché y broker de mensajes. CKAN utiliza Redis como broker de mensajes para sus trabajos en segundo plano.

Cuando CKAN necesita realizar una tarea que puede llevar mucho tiempo, como actualizar el índice de búsqueda para un gran número de conjuntos de datos, añade un trabajo a la cola de Redis, que luego puede ser procesado en segundo plano.

- NGINX: Un servidor HTTP y proxy inverso de alto rendimiento, gratuito y de código abierto. CKAN utiliza NGINX como un proxy inverso para enrutar las solicitudes HTTP entrantes a la aplicación CKAN. NGINX también puede servir archivos estáticos, manejar la encriptación SSL y balancear las solicitudes entre múltiples instancias de CKAN. La configuración de NGINX para CKAN incluye tanto puntos finales SSL como no SSL, permitiendo una comunicación segura a través de HTTPS.

Requisitos

```
import Tabs from '@theme/Tabs';
import TabItem from '@theme/TabItem';
import Prism from 'prismjs';

# Prerrequisitos
Antes de poder usar  [`ckan-ansible`](https://github.com/mjanez/ckan-
ansible), necesitas instalar:
-   [Ansible](https://docs.ansible.com/ansible/latest/
installation_guide/intro_installation.html)
```

CKAN: Instalación y configuración

Despliegue de CKAN con Ansible

Clona este repositorio en tu máquina local y edita las variables del archivo `playbook/host_vars/production_01.yml`:

```
git clone https://github.com/mjanez/ckan-ansible.git && cd ckan-ansible
```

> ⚠️ **CONFIGURACIÓN DEL PLAYBOOK**
>
> Recuerda cambiar las variables de configuración de CKAN antes de ejecutar el playbook de Ansible. Específicamente la información del usuario/contraseña del host (`ansible_user`, `ansible_password`, etc.) y la configuración de CKAN: `ckan_sysadmin_name`, `ckan_sysadmin_password` y `ckan_sysadmin_email`. También el `proxy_server_name` y `nginx_port` para un despliegue correcto.

Edita las variables de los hosts en la carpeta `inventory` y añade las direcciones IP o el `hostname` de los servidores de despliegue objetivo para el entorno específico.

Personaliza las configuraciones de despliegue en `host_vars/*` para que coincidan con tus requisitos. Modifica cualquier variable necesaria, como las credenciales de la base de datos, las versiones de CKAN y otros ajustes específicos.

> ⚠ **AUTENTICACIÓN SSH**
>
> También, si estás utilizando una autenticación de contraseña SSH para repositorios privados, crea un par de claves SSH y copia las claves en `./playbook/roles/common/files/keys`. Los nombres de los archivos del par de claves deben comenzar con id_ (por ejemplo, `id_rsa` + `id_rsa.pub`)

Ejemplo de despliegue

1. Selecciona el entorno que quieres desplegar, por ejemplo: `rhel`.

2. Edita el archivo `playbook/host_vars/production_01.yml`. Indica la ruta a la clave privada SSH si no estás utilizando autenticación por contraseña (`ansible_ssh_private_key_file`/`ansible_ssh_pass`).

3. Ejecuta el playbook de Ansible para desplegar CKAN en el servidor objetivo. El siguiente comando desplegará CKAN en el servidor objetivo utilizando la configuración del playbook. La bandera `-vvv` se utiliza para una salida detallada:

   ```
   # Ubicación del archivo ansible.cfg basado en el directorio
   clonado
   export ANSIBLE_CONFIG=$(pwd)/playbook/ansible.cfg

   # Ubicación si ckan-ansible se clona en el directorio home
   export ANSIBLE_CONFIG=$HOME/ckan-ansible/playbook/ansible.cfg

   # Ejecuta el playbook de ansible, Verbose con  -vvv
   ansible-playbook $HOME/ckan-ansible/playbook/playbook.yml
   ```

   La variable de entorno `ANSIBLE_CONFIG` se utiliza para especificar la ubicación del archivo `ansible.cfg`. Esto es útil cuando tienes varias configuraciones de Ansible y quieres especificar cuál usar, por ejemplo,

`rhel-9`, `ubuntu-20.04`, etc.

> **HOST_VARS**
>
> El archivo `*/host_vars/*.yml` contiene variables de configuración personalizables para el despliegue, incluyendo las credenciales de la base de datos, la versión de CKAN y la configuración del servidor web. Revisa y modifica estas antes de ejecutar el playbook de Ansible.

Comprobar que los servicios están en funcionamiento

Después del despliegue, puedes comprobar el estado de los servicios utilizando el comando `supervisorctl`. Este comando proporciona una interfaz de línea de comandos al sistema de control de procesos Supervisor, que te permite controlar y monitorizar tus servicios.

Los servicios generados por ckan-Ansible incluyen:

- `ckan`, un sistema de gestión de datos de código abierto para alimentar hubs de datos y portales de datos que se sirve a través de uWSGI y NGINX.
- `ckan-pycsw`, un servicio web completo para catalogar datos geoespaciales. Software para lograr la interoperabilidad con los portales de datos abiertos basados en CKAN. Para hacer esto, `ckan2pycsw` lee datos de una instancia usando la API de CKAN, genera metadatos INSPIRE ISO-19115/ISO-19139 [^3] usando pygeometa, o otro esquema personalizado, y llena una instancia de pycsw que expone los metadatos usando CSW y OAI-PMH.
- `ckan-xloader`, trabajador para cargar rápidamente datos en DataStore. Un reemplazo para DataPusher.

- Trabajadores utilizados para la recolección remota. El trabajador `ckan_harvester_run` se utiliza para ejecutarse periódicamente, los recolectores y el trabajador `ckan_harvester_fetch` se utiliza para recuperar los datos de los servidores remotos y prepararlos para el trabajador `ckan_harvester_run`. También el trabajador `ckan_harvester_gather` se utiliza para identificar los recursos remotos que necesitan ser recolectados. Finalmente, el trabajador `ckan_harvester_clean_log` se utiliza para limpiar periódicamente los registros de los recolectores.

Para comprobar el estado de estos servicios, puedes utilizar el comando `supervisorctl status`. Aquí tienes un ejemplo:

```
$ supervisorctl status
ckan_fetch_consumer              RUNNING   pid 2684195, uptime 0:00:50
ckan_gather_consumer             RUNNING   pid 2684193, uptime 0:00:50
ckan_harvester_clean_log         STOPPED   Not started
ckan_harvester_run               EXITED    May 07 01:12 PM
ckan_pycsw                       RUNNING   pid 2684197, uptime 0:00:50
ckan_uwsgi:ckan_uwsgi-00         RUNNING   pid 2684194, uptime 0:00:50
ckan_xloader:ckan_xloader-00     RUNNING   pid 2684198, uptime 0:00:50
```

API de CKAN: Conceptos básicos

INFO

`params`: Parámetros para pasar a la función de acción. Los parámetros son específicos para cada función de acción.

- `fl` (texto): Campos del conjunto de datos a devolver. El parámetro controla qué campos se devuelven en la consulta solr. `fl` puede ser `None` o una lista de campos de resultado, como: `id,name,extras_custom_schema_field`.

  Ejemplo Todos los conjuntos de datos con los campos `id`, `name`, `title` y un campo de esquema personalizado `extras_inspire_id`:

  ```
  {instancia-ckan}/api/3/action/
  package_search?fl=id,name,title,extras_inspire_id
  ```

- `fq` (texto): Cualquier consulta de filtro para aplicar.

  Ejemplo Todos los conjuntos de datos que tienen la etiqueta `economy`:

  ```
  http://demo.ckan.org/api/3/action/package_search?fq=tags:economy
  ```

- `rows` (int): El número máximo de filas (conjuntos de datos) coincidentes para devolver. (opcional, por defecto: `10`, límite superior: `1000` a menos que se establezca en la configuración del sitio `ckan.search.rows_max`)

> Más info: Documentación API de CKAN y data.gov.uk

Listar conjuntos de datos por campos

Solicitud: `{instancia-ckan}/api/3/action/package_search?fl=id,extras_publisher_name`

Response:


```
{
  "help": "{ckan-instance}/api/3/action/help_show?name=package_search",
  "success": true,
  "result": {
    "count": 32,
    "facets": {},
    "results": [
      {
        "id": "e4a607d0-0875-4043-b8c7-36f731ba5ca8",
        "publisher_name": "Example publisher"
      },
      {
        "id": "5319a6b3-f439-4f53-9732-71699b9f62c8",
        "publisher_name": "Example publisher"
      },
      {
        "id": "02a30269-7665-4f6a-a43d-c288003f5cbb",
        "publisher_name": "Example publisher"
      }
    ],
    "sort": "score desc, metadata_modified desc",
    "search_facets": {}
  }
}
```


Todos los conjuntos de datos de la organización (con algunos campos)

Solicitud: `{ckan-instance}/api/3/action/package_search?fq=organization:iepnb&fl=id,name,extras_alternate_identifier&rows=100`

Response:


```
{
  "help": "{ckan-instance}/api/3/action/help_show?name=package_search",
  "success": true,
  "result": {
    "count": 56,
    "facets": {},
    "results": [
      {
        "id": "fe757d64-436c-482d-b65b-f24348139fd6",
        "name": "example_dataset_1",
        "alternate_identifier": "IDEXAMPLEDATASET1"
      },
      {
        "id": "fc21c1a5-4c02-4157-9d2f-9a2cd200f908",
        "name": "example_dataset_2",
        "alternate_identifier": "IDEXAMPLEDATASET2"
      },
      {
        "id": "fb326c11-18d4-4ee1-aa23-a40cb90cf8d8",
        "name": "example_dataset_3",
        "alternate_identifier": "IDEXAMPLEDATASET3"
      }
    ],
    "sort": "score desc, metadata_modified desc",
    "search_facets": {}
  }
}
```


Toda la información sobre un conjunto de datos por campo

Solicitud: `{ckan-instance}/api/3/action/package_search?q=name:"spa_example_dataset_1_2023"`

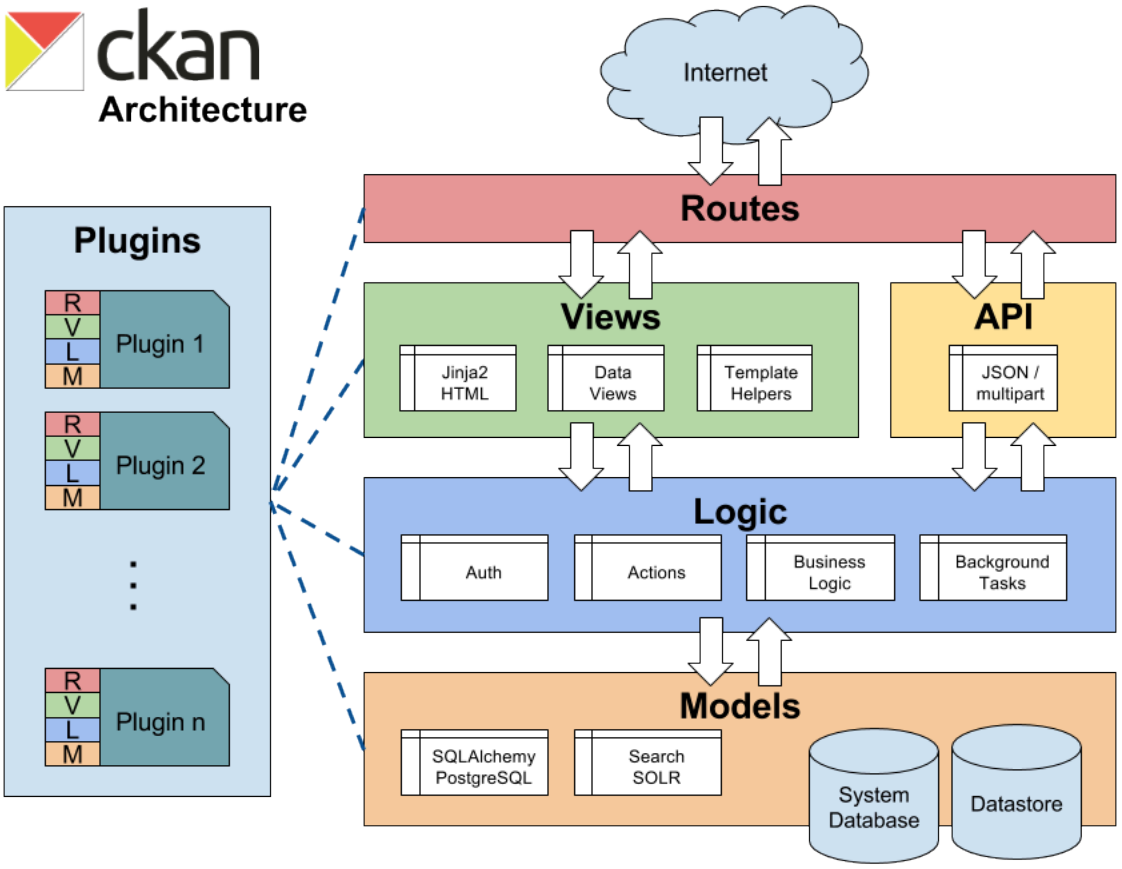
Response:


```
{
  "help": "https://demo.ckan.org/api/3/action/help_show?name=package_search",
  "success": true,
```

CKAN: Desarollo

Arquitectura del código de CKAN

La arquitectura de código de CKAN está diseñada con ciertos estándares de codificación en mente para mantener la consistencia con su diseño y arquitectura previstos.



TIP

> Más información en: CKAN Code Architecture

Blueprints

CKAN se basa en Flask y utiliza Blueprints. Los blueprints predeterminados se definen junto con las vistas en `ckan.views` y se extienden con la interfaz de plugin `ckan.plugins.interfaces.IBlueprint`.

Vistas (Views)

Las vistas procesan las solicitudes leyendo y actualizando datos con funciones de acción y devuelven una respuesta renderizando plantillas Jinja2. Las vistas de CKAN se definen en `ckan.views` y las plantillas en `ckan.templates`.

Funciones de ayuda de plantilla (Helpers)

Las funciones de ayuda de plantilla se utilizan para el código que se reutiliza con frecuencia o el código que es demasiado complicado para incluirse en las plantillas mismas. Nunca deben realizar consultas costosas o actualizar datos.

Funciones de acción (Actions)

Cuando algún código quiere obtener, crear, actualizar o eliminar un objeto del modelo de CKAN, debe hacerlo llamando a una función del paquete `ckan.logic.action`, y *no* accediendo directamente a `ckan.model`.

Lógica (Logic)

La lógica incluye funciones de acción, funciones de autenticación, tareas en segundo plano y lógica de negocio. Las funciones de acción tienen una interfaz

uniforme que acepta un diccionario de listas de cadenas simples, diccionarios o archivos. Devuelven diccionarios simples o lanzan una de un pequeño número de excepciones.

Modelos (Models)

Idealmente, SQLAlchemy solo debe usarse dentro de `ckan.model` y no desde otros paquetes como `ckan.logic`.

Deprecación (Deprecated)

Cualquier cosa que pueda ser utilizada por extensiones, temas o clientes de API necesita mantener la compatibilidad hacia atrás en el sitio de llamada. Para deprecar una función, use el decorador `ckan.lib.maintain.deprecated` y agregue "deprecated" a la cadena de documentación de la función.

Guías de contribución

CKAN es un software libre de código abierto y se agradecen las contribuciones, ya sean informes de errores, código fuente, documentación o traducciones. Las siguientes secciones te guiarán a través de nuestros procesos para hacer diferentes tipos de contribuciones a CKAN:

- Traduciendo CKAN
- Probando CKAN
- Escribiendo documentación
- Arquitectura de código de CKAN
- Estándares de codificación CSS
- Estándares de codificación HTML
- Estándares de codificación JavaScript
- Estándares de codificación Python

- Internacionalización de cadenas
- Manejo de Unicode
- Estándares de codificación de pruebas
- Directrices de desarrollo de frontend
- Migraciones de base de datos
- Actualizando las dependencias de CKAN

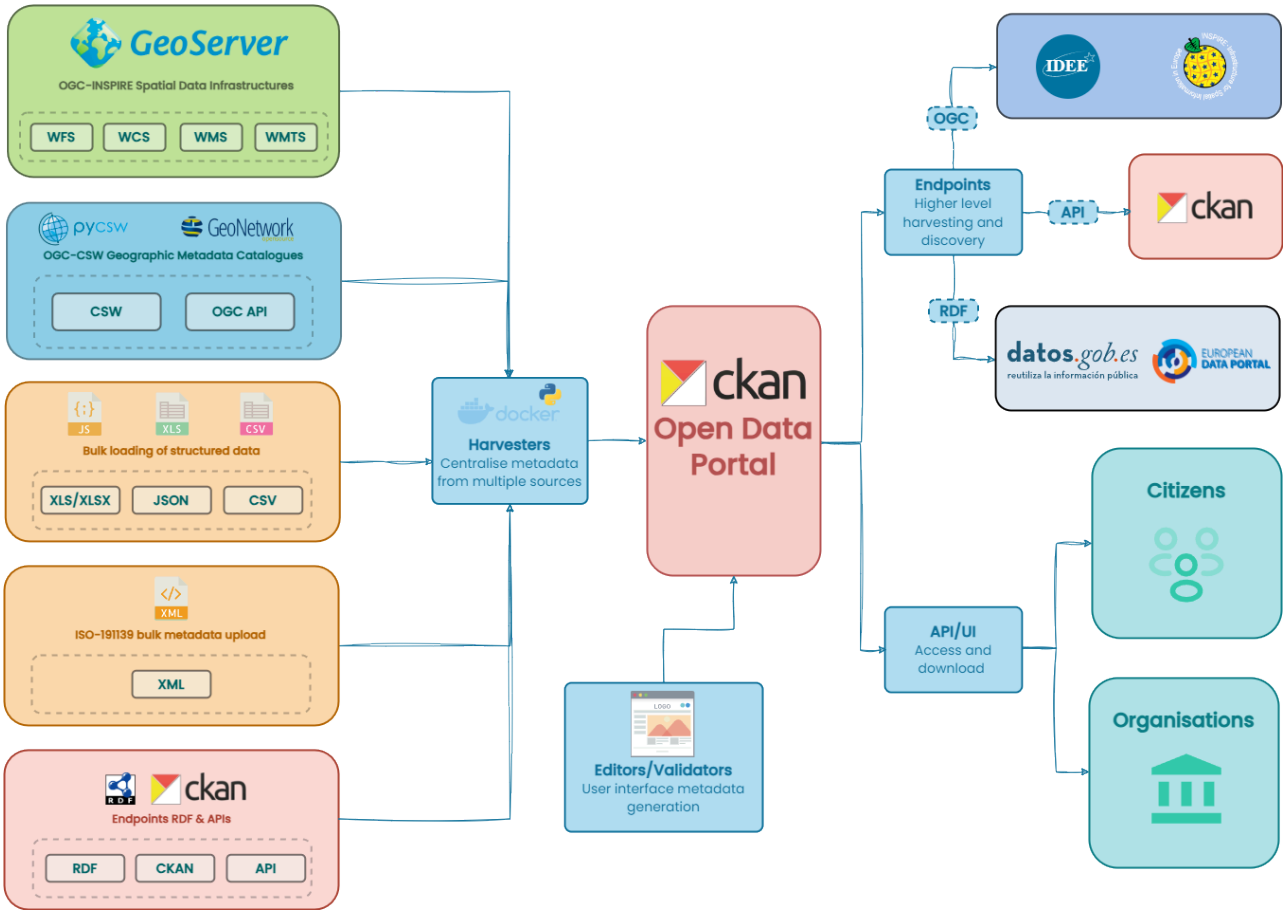
Cosechadores CKAN

`ckanext-harvest`, es una poderosa extensión para CKAN, una plataforma líder de portal de datos de código abierto. Esta extensión proporciona un marco común de recolección que permite a las extensiones de CKAN recopilar datos de varias fuentes y traerlos a la instancia de CKAN.

El proceso de recolección implica dos pasos principales: recolección y búsqueda. La etapa de recolección identifica todos los objetos (conjuntos de datos, archivos, etc.) que necesitan ser recolectados de la fuente, mientras que la etapa de búsqueda recupera los objetos identificados.

El Recolector de CKAN admite una variedad de fuentes y formatos de recolección, y se puede extender para admitir más. También proporciona tanto una Interfaz de Línea de Comandos (CLI) como una Interfaz de Usuario Web (WUI) para administrar fuentes de recolección y trabajos, lo que lo convierte en una herramienta versátil para la recolección y gestión de datos.

La extensión requiere CKAN v2.0 o posterior y puede usar Redis o RabbitMQ como su backend. También ofrece una gama de opciones de configuración para personalizar el proceso de recolección según sus necesidades.



Recolector personalizado

Resumen

Un Recolector en el contexto de CKAN, es un componente o proceso que es responsable de obtener, recopilar e importar datos de varias fuentes de datos externas a la instancia de CKAN.

El Recolector opera en dos etapas principales: la etapa de recolección y la etapa de búsqueda. En la etapa de recolección, el Recolector identifica todos los objetos (como conjuntos de datos o archivos) que necesitan ser recolectados de la fuente. En la etapa de búsqueda, el Recolector recupera los

objetos identificados y los trae a la instancia de CKAN.

Los Cosechadores pueden ser configurados para trabajar con una variedad de fuentes de datos y formatos, lo que los convierte en una herramienta versátil para poblar una instancia de CKAN con datos. Se pueden usar para automatizar el proceso de recolección de datos, reduciendo la necesidad de entrada manual de datos y asegurando que la instancia de CKAN se actualice regularmente con datos frescos de la fuente.

Además de obtener e importar datos, los Cosechadores también proporcionan características como notificaciones de errores, configuraciones de tiempo de espera de trabajos y la capacidad de evitar sobrescribir ciertos campos, ofreciendo una solución integral para las necesidades de recolección de datos en CKAN.

Desarrollar un recolector

Para desarrollar un nuevo recolector, deberá seguir estos pasos:

1. Cree un nuevo archivo Python en su directorio de extensión `ckanext/myplugin/harvesters/`. El nombre del archivo debe ser descriptivo del tipo de fuente de datos que el recolector procesará. Por ejemplo, si está creando un recolector para extraer datos de una API REST, podría nombrar el archivo `rest_api_harvester.py`.

2. En este archivo, deberá definir una nueva clase que herede de `ckanext.harvest.harvesters.base.HarvesterBase`. Esta clase debe implementar al menos los siguientes métodos:

    - `info(self)`: Este método debe devolver un diccionario con información sobre el recolector. Debe incluir al menos las claves 'name', 'title' y 'description'.

- `gather_stage(self, harvest_job)`: Este método es responsable de recopilar los identificadores de los objetos a recolectar y debe devolver una lista de estos identificadores.

- `fetch_stage(self, harvest_object)`: Este método recibe un objeto de recolección (identificado por uno de los identificadores devueltos en el método `gather_stage`) y debe devolver un diccionario con los datos del objeto.

- `import_stage(self, harvest_object)`: Este método recibe un objeto de recolección (con los datos ya extraídos en el método `fetch_stage`) y debe crear o actualizar el conjunto de datos correspondiente en CKAN.

Aquí hay un ejemplo de cómo podría ser un recolector básico:

```
from ckanext.harvest.harvesters.base import HarvesterBase

class RestAPIHarvester(HarvesterBase):
    def info(self):
        return {
            'name': 'rest_api',
            'title': 'Recolector de API REST',
            'description': 'Un recolector para APIs REST'
        }

    def gather_stage(self, harvest_job):
        # Aquí iría el código para recopilar los identificadores de
los objetos a recolectar de la API REST.
        return []

    def fetch_stage(self, harvest_object):
        # Aquí iría el código para extraer los datos del objeto de la
API REST.
        return True
```

3. Una vez que haya definido su clase de recolector, necesita registrarlo para que CKAN lo reconozca. Esto se hace en el archivo `plugin.py` en el directorio `ckanext/harvest/`. Aquí, debe importar su clase de recolector y agregarla a la lista de `harvesters` en el método `after_map` de la clase `HarvestPlugin`.

```
from ckanext.harvest.harvesters import HarvesterBase,
rest_api_harvester

class HarvestPlugin(plugins.SingletonPlugin):
    plugins.implements(plugins.IConfigurer)
    plugins.implements(plugins.IRoutes, inherit=True)

    harvesters = []

    def after_map(self, map):
        self._register_harvesters()
        return map

    def _register_harvesters(self):
        self.harvesters.append(rest_api_harvester.RestAPIHarvester())
```

4. Finalmente, necesita reiniciar CKAN para que los cambios surtan efecto.

> ⚠ **PRECAUCIÓN**
>
> Tenga en cuenta que este es un ejemplo muy básico. Dependiendo de la fuente de datos, es posible que deba implementar lógica adicional en los métodos `gather_stage`, `fetch_stage` y `import_stage`. Además, es posible que deba implementar otros métodos, como `validate_config` (para validar la configuración del recolector) o `get_original_url` (para obtener la URL original de los objetos recolectados).