



DIPARTIMENTO DI
INGEGNERIA ELETTRICA
E DELL'INFORMAZIONE

Tema d'anno di

IMAGE PROCESSING



Tutorial ODC

Open Data Cube

Docente

Prof. Andrea Guerriero

Studenti:

La Forgia Vincenzo

Spedalieri Michele

C. d. L. M in Ingegneria Informatica

Sommario

| | |
|---|----|
| Sommario | 2 |
| CAPITOLO 1 | 4 |
| Cos'è l'Open DataCube? | 4 |
| 1.1 Background | 4 |
| 1.2 Purpose | 5 |
| 1.2 Vision..... | 5 |
| 1.3 Partner Istituzionali..... | 6 |
| 1.4 Panoramica | 6 |
| 1.5 Tools e applications..... | 7 |
| CAPITOLO 2 | 9 |
| Installazione Open Data cube | 9 |
| 2.1 Requisiti di sistema..... | 9 |
| 2.2 Update | 9 |
| 2.3 Prerequisiti | 10 |
| 2.4 Installazione ambiente virtuale | 10 |
| 2.5 Librerie GDAL..... | 10 |
| 2.6 CORE | 11 |
| 2.7 Database PostgreSQL | 11 |
| Correzioni temporanee | 13 |
| CAPITOLO 3 | 15 |
| Datacube ingestion Guide | 15 |
| Nota: | 16 |
| 3.1 CREAZIONE E AGGIUNTA DI UNA DEFINIZIONE DI PRODOTTO | 16 |
| 3.3 GENERAZIONE DI METADATI | 18 |
| 3.3.1 Preparazione script metadata | 20 |
| 3.4 INDICIZZAZIONE DEI SET DI DATI NEL DATABASE | 24 |
| 3.5 INGERIRE I SET DI DATI | 25 |
| Note importanti | 28 |
| CAPITOLO 4 | 33 |
| Data Cube Jupyter Notebook | 33 |
| 4.1 Processo di installazione | 33 |
| 4.2 Configurazione..... | 33 |
| CAPITOLO 5 | 35 |
| Data Cube UI Installation | 35 |
| 5.1 Processo di installazione | 35 |
| 5.2 Configurazione del server | 36 |

| | |
|---|----|
| 5.3 Inizializzazione del database..... | 39 |
| NOTA:..... | 39 |
| 5.4 Avviare i workes..... | 40 |
| 5.5 Panoramica del sistema di attività..... | 41 |
| 5.6 Personalizza l'interfaccia utente | 42 |
| Capitolo 6..... | 47 |
| Aggiunta nuovi algoritmi | 47 |
| 6.1 Django Views | 47 |
| 6.2 Django Models | 47 |
| 6.3 Celery tasks | 47 |
| 6.4 Classi Base | 48 |
| 6.5 Generazione di file di base..... | 50 |
| 6.6 Applicazioni di base | 51 |
| NOTA:..... | 51 |
| 6.7 Esempio | 54 |
| 6.8 Applicazioni complesse..... | 55 |

CAPITOLO 1

Cos'è l'Open DataCube?

Open Data Cube (ODC) è un progetto Open Source Geospatial Data Management e Analysis Software che aiuta a sfruttare la potenza dei dati satellitari. Al suo interno, l'ODC è un insieme di librerie Python e database PostgreSQL che ci aiuta a lavorare con dati raster geospaziali. Un set di dati raster è rappresentato da una matrice composta da righe e colonne di pixel (dette anche celle). Ogni pixel rappresenta una regione geografica, ed il valore del pixel rappresenta una caratteristica di tale regione.

Per garantire il suo successo, l'ODC deve stabilire un "marchio" di cui gli utenti possano fidarsi e che deve promuovere un'esperienza utente positiva. Ciò dovrebbe essere reso possibile attraverso lo sviluppo di una comunità ODC open source che è attivamente impegnata e contribuisce al codice di base, condivide gli algoritmi e fornisce supporto reciproco per la risoluzione dei problemi.

1.1 Background

Con il passare degli anni, le nuove generazioni di satelliti EO stanno creando volumi di dati sempre più significativi con una copertura globale così ampia che per molte applicazioni la mancanza di dati non è più un fattore limitante. Un'ampia attività di ricerca e sviluppo ha prodotto nuove applicazioni dei dati, che offrono un significativo potenziale per incidere con un grande impatto a importanti sfide ambientali, economiche e sociali, anche a livello locale, regionale e globale. Tali applicazioni evidenziano il valore di EO, anche se la sfida consiste nel fornire le giuste connessioni tra dati, applicazioni e utenti. Ancora oggi, molti dati satellitari di EO archiviati sono sottoutilizzati nonostante le moderne infrastrutture informatiche e di analisi.

Affrontare questa sfida è difficile per le economie avanzate e ancora più difficile per i paesi in via di sviluppo che hanno interesse a utilizzare i dati satellitari EO. Semplicemente non è tecnicamente fattibile o economicamente conveniente considerare i tradizionali metodi di elaborazione locale e di distribuzione dei dati (ad esempio download di file basati su scene su Internet) e altri problemi come la manipolazione, la conservazione e l'analisi, rimangono ostacoli significativi.

Fortunatamente, proprio come la tecnologia di osservazione della Terra satellitare è progredita in modo significativo, anche la tecnologia dell'informazione è avanzata. Le sfide di gestione e analisi dei dati derivanti dall'enorme aumento dei volumi di dati liberi e aperti possono essere superate con nuove infrastrutture di calcolo, tecnologie e architetture di dati, come ad esempio "Open Data Cube". Tale soluzione ha un grande potenziale per semplificare la distribuzione e la gestione dei dati per i fornitori, riducendo al contempo le barriere tecniche che consentono agli utenti di sfruttare al massimo i dati.

1.2 Purpose

Committee on Earth Observation Satellites (CEOS) è un partner fondatore dell'iniziativa Open Data Cube (ODC) che cerca di fornire una soluzione di architettura dei dati che abbia valore per i suoi utenti globali e aumenti l'impatto dei dati satellitari dell'EO. Tecnologie come l'Australian Geoscience Data Cube (AGDC) e Google Earth Engine (GEE) hanno trasformato la comunità di utenti di dati satellitari EO. In risposta alla domanda degli utenti, tali soluzioni tecnologiche eliminano il peso della preparazione dei dati, producono risultati rapidi e promuovono una comunità globale di collaboratori attivi e coinvolti. Quindi, CEOS si impegna a gestire e contribuire all'architettura ODC come parte della comunità ODC. Cercano di incoraggiare gli altri a partecipare all'iniziativa con l'obiettivo finale di soddisfare le esigenze degli utenti, simili agli obiettivi di AGDC e GEE.

Man mano che il mondo si sviluppa, cresce anche la sua conoscenza e richiesta di dati satellitari EO. I problemi principali per gli utenti sono l'accesso ai dati, la preparazione dei dati e analisi efficienti per supportare le applicazioni degli utenti. Il CEOS, attraverso la sua rete di connessioni globali, ha stabilito che gli utenti globali condividono molte esigenze comuni che possono essere soddisfatte attraverso l'iniziativa ODC.

Alcuni di questi bisogni sono elencati di seguito:

- Ridurre al minimo il tempo e le conoscenze specialistiche necessarie per accedere e preparare i dati satellitari;
- Dati satellitari EO gratuiti e algoritmi applicativi;
- Soluzioni software open source avanzate attraverso i contributi della community;
- Architetture di dati coerenti che consentono la condivisione di codice, strumenti e algoritmi;
- Analisi di serie temporali efficienti per supportare le applicazioni di cambio di terra;
- Utilizzo di più set di dati insieme (ad es. Interoperabilità e complementarità);
- Uso di strumenti GIS comuni (ad es. QGIS, ArcGIS);
- Soluzioni locali e regionali che evitano la dipendenza commerciale e internet;
- Servizio clienti e supporto utenti sostenuti.

1.2 Vision

L'obiettivo dell'ODC è di aumentare l'impatto dei dati satellitari fornendo uno strumento aperto e liberamente accessibile e di incoraggiare la comunità a sviluppare, sostenere e far crescere l'ampiezza e la profondità delle applicazioni. Questa soluzione intende sostenere obiettivi chiave, tra cui la creazione della capacità degli utenti di applicare i dati satellitari EO e il sostegno a programmi di priorità globali, come quelli che si trovano negli obiettivi di sviluppo sostenibile delle Nazioni Unite (UN-SDG) e negli Accordi di Parigi

e Sendai. Ciò dovrebbe essere reso possibile attraverso lo sviluppo di una comunità ODC open source che è attivamente impegnata e contribuisce al codice core.

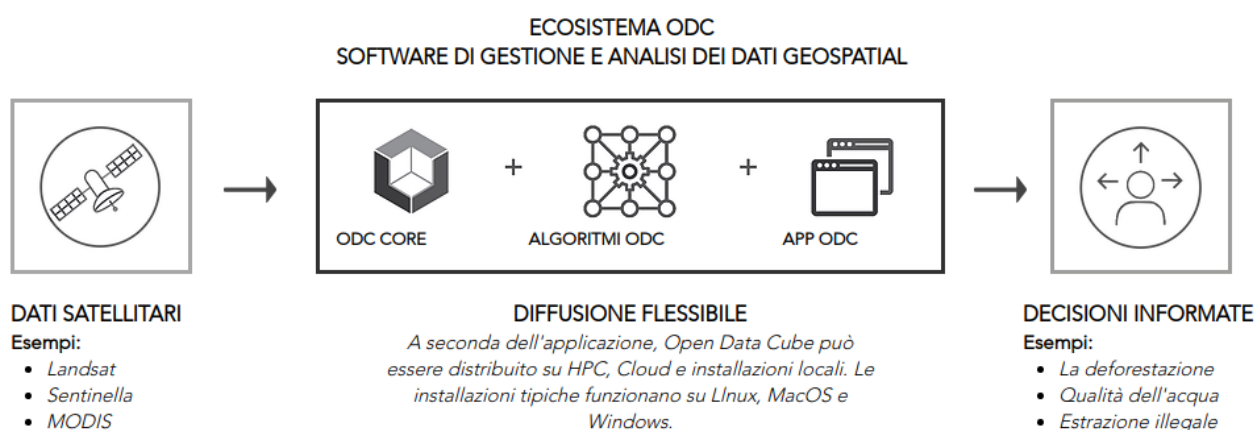
La scalabilità è una delle sfide chiave affrontate dall'ODC. Sebbene gli sforzi iniziali abbiano portato solo a implementazioni di Data Cube su scala nazionale (ad es. Australia, Colombia, Svizzera), ci sono molti altri paesi e organizzazioni globali con grande interesse per l'ODC. Attraverso l'impegno con le organizzazioni delle parti interessate e gli attuali utenti, si prevede che il numero di implementazioni ODC aumenterà e la comunità ODC prospererà con l'aumento dei contributi e dell'impatto. Tali soluzioni open source trarranno vantaggio anche dalle implementazioni precedenti per aumentare l'efficienza riutilizzando i contenuti e applicando le "lezioni apprese" per le nuove distribuzioni.

1.3 Partner Istituzionali

L'iniziativa Open Data Cube è supportata da sei partner istituzionali:

- Geoscience Australia (GA),
- NASA / Comitato per l'osservazione della Terra Satellite (CEOS),
- United States Geological Survey (USGS),
- Commonwealth Scientific and Industrial Research Organization (CSIRO),
- Catapult Satellite Applications e Analytical Mechanics Associates (AMA).

1.4 Panoramica



Open Data Cube (ODC) è una soluzione open source per l'accesso, la gestione e l'analisi di grandi quantità di dati GIS (Geographic Information System), ovvero i dati di osservazione della Terra (EO). Presenta un quadro analitico comune composto da una serie

di strutture e strumenti di dati che facilitano l'organizzazione e l'analisi di ampie raccolte di dati a griglia.

L'Open Data Cube è stato sviluppato per l'analisi di dati di osservazione della terra ricchi in termini temporali, tuttavia la flessibilità della piattaforma consente anche di includere e analizzare altre raccolte di dati a griglia.

Una caratteristica chiave di Open Data Cube è che ogni osservazione unica viene mantenuta, il che contrasta con molti altri metodi usati per gestire ampie raccolte di dati con griglia. Alcuni dei principali vantaggi di ODC sono i seguenti:

- Framework flessibile;
- L'utente mantiene il controllo e la proprietà sui propri dati;
- Passaggio paradigmatico dall'analisi basata su scene a quella basata su pixel;
- Barriera inferiore all'ingresso per l'analisi dei dati del telerilevamento.;

Fornisce le basi di diverse soluzioni di architettura di dati internazionali, da scala regionale a nazionale, come Digital Earth Australia, Africa Data Cube regionale e altri. Data Cube funziona bene con dati ARD (Analysis Ready Data), pre-elaborati e pronti all'uso resi disponibili dai fornitori di dati.

Mentre i provider lavorano per rendere disponibili sul cloud prodotti ARD globali, il Data Cube utilizza tipicamente la raccolta USGS 1 Landsat 8 PDS per le dimostrazioni. Questi dati non sono ARD e non dovrebbero essere usati per analisi scientifiche. Nel 2019, si prevede che i dati ARD diventeranno facilmente disponibili sul Cloud e fino a quel momento l'utente può semplicemente aggiungere e indicizzare i propri dati elaborati.

Il sistema Open Data Cube è progettato per:

- Catalogare grandi quantità di dati di osservazione della Terra;
- Fornire un'API basata su Python per l'interrogazione e l'accesso ai dati ad alte prestazioni;
- Offrire agli scienziati e agli altri utenti una facile capacità di eseguire analisi dei dati esplorativi;
- Consente l'elaborazione scalabile su scala continentale dei dati memorizzati;
- Traccia la provenienza di tutti i dati contenuti per consentire il controllo di qualità e gli aggiornamenti;

ODC sarà sempre un software open source al 100%, gratuito per tutti da utilizzare e rilasciato sotto i termini liberali della licenza Apache 2.0.

1.5 Tools e applications

Il core ODC funge da strato tra i fornitori di dati satellitari e le applicazioni. Esiste una serie di strumenti open source per aiutare gli scienziati a condurre ricerche utilizzando i dati gestiti dall'ODC. Gli strumenti più popolari utilizzati all'interno della comunità che utilizza ODC Core come base sono:

- Strumenti a riga di comando: uno strumento utilizzato da programmatori / sviluppatori per interfacciarsi con l'ODC.
- Open Data Cube Explorer: un'applicazione Web visiva e interattiva che consente agli utenti di esplorare il proprio inventario dei dati disponibili.
- Open Data Cube Stats: un mezzo ottimizzato per la definizione e l'esecuzione di analisi avanzate su sistemi ODC. Questo strumento è orientato verso gli scienziati.
- Interfaccia utente Web (UI): un'applicazione Web che consente agli sviluppatori di mostrare e visualizzare in modo interattivo l'output degli algoritmi.
- Jupyter notebook: documenti di ricerca incentrati sulle tecniche nelle scienze EO. Un notebook contiene un codice eseguibile che descrive in dettaglio come il data cube viene utilizzato in un ambiente di ricerca, e quindi è un materiale di riferimento inestimabile per i nuovi utenti.
- Servizi Web Open Geospatial Consortium (OGC): adattatori che possono collegare applicazioni non-ODC all'ODC.

CAPITOLO 2

Installazione Open Data cube

L'installazione dell'ODC è composta da più parti completamente collegate e interconnesse che servono tutte al corretto funzionamento dell'applicazione e di tutte le funzionalità ad esse collegate. Abbiamo deciso di installare ODC su sistema operativo Ubuntu 18.04, in quanto la guida per Windows era incompleta e conteneva innumerevoli errori.

La guida per l'installazione di ODC è suddivisa in quattro parti:

- Data cube install
- Ingestion
- Notebook install
- UI install

In questo capitolo descriviamo i processi necessari per l'installazione della libreria datacube con tutte le sue dipendenze.

2.1 Requisiti di sistema

Questa sezione presuppone che un utente locale, non un utente amministratore, verrà utilizzato per eseguire tutti i processi. Nella nostra installazione abbiamo usato come utente locale "vincenzo" che corrisponde anche all'amministratore di sistema.

Useremo localuser come nome utente, ma può essere qualsiasi nome che si voglia. Si raccomanda comunque l'uso di localuser in quanto, in parecchi file di configurazione presuppongono l'uso del "localuser". Per utilizzare un nome diverso potrebbe essere necessario modificare diversi file di configurazione aggiuntivi che altrimenti non avrebbero bisogno di modifiche. Non utilizzare caratteri speciali come **è**, **Ä** o **î** per il nome utente che potrebbe causare problemi in futuro. Raccomandiamo che il nome utente sia scritto in minuscolo e utilizzando, se si ha necessità, l'underscore ("_").

2.2 Update

Prima di iniziare l'installazione dei pacchetti, è una buona idea aggiornare il software che recupererà tali pacchetti e le posizioni da cui verranno recuperati. Le linee di seguito aggiorneranno apt-get quindi installare Python3, npm, pip3, e git.

Gli altri pacchetti tmux e htop sono strumenti utili per il monitoraggio delle prestazioni ma non sono necessari. Infine, tentiamo di aggiornare pip3, nel caso in cui la versione non sia

nuova come potrebbe essere. I seguenti comandi vanno inseriti nel terminale e servono appunto a compiere le azioni sopra descritte.

```
sudo apt-get update
sudo apt-get install tmux htop python3-dev python3-pip git
sudo pip3 install --upgrade pip
```

2.3 Prerequisiti

I seguenti comandi creeranno le directory che richiede l'open datacube e imposteranno le loro autorizzazioni in modo che tutti gli utenti possano leggere e scrivere dati.

```
sudo mkdir -p /datacube/{original_data,ingested_data}
sudo chmod -R 777 /datacube/

mkdir -p ~/Datacube
```

2.4 Installazione ambiente virtuale

Successivamente, dobbiamo installare l'ambiente virtuale e attivarlo prima di iniziare a installare pacchetti. Questo è un modo per separare i pacchetti python e mantenere il vostro ambiente operativo non influenzato dalle modifiche apportate a Python dall'ambiente virtuale.

```
sudo pip3 install virtualenv
virtualenv ~/Datacube/datacube_env
source ~/Datacube/datacube_env/bin/activate
```

2.5 Librerie GDAL

I primi due comandi rappresentano la versione GDAL utilizzata da datacube-core 2.4.0, che, al momento della scrittura, non è disponibile sui repository predefiniti utilizzati da apt-get. Nota che si dovrà premere il tasto Invio per eseguire effettivamente il primo comando.

```
sudo add-apt-repository ppa:ubuntugis/ubuntugis-unstable
sudo apt-get update
sudo apt-get install gdal-bin libgdal-dev libnetcdf-dev netcdf-bin libhdf5-serial-dev
hdf5-tools
```

La versione delle librerie GDAL può essere determinata con il comando `gdalinfo --version`. Assicurarsi che corrisponda al vostro pacchetto di associazioni GDAL Python o riceverai un errore relativo a `x86_64-linux-gnu-gcc`. Il prossimo passo richiederà un'installazione compatibile di `gdal`. Di nuovo, come detto prima, eseguire il seguente comando per vedere la versione `gdalinfo` attualmente sulla macchina:

```
gdalinfo --version
```

Esegui il seguente comando dove XXX è la versione del passaggio precedente o il più vicino possibile. Ad esempio, se è stato mostrato 2.4.1 ma non è possibile installarlo, prova 2.4.0 e così via:

```
pip install --global-option=build_ext --global-option="-I/usr/include/gdal" gdal==2.4.0
```

Utilizzare i seguenti comandi per installare le dipendenze Python richieste. Questi pacchetti sono necessari per l'utilizzo del Data Cube, dell'indicizzazione S3 e dei blocchi appunti Data Cube. La versione 1.2.18 di SQLAlchemy viene utilizzata per evitare errori con il comando `datacube -v system init` quando si utilizzano le versioni predefinite e più recenti di SQLAlchemy (almeno la versione 1.3.0b3).

```
pip install rasterio==1.0.2
pip install numpy xarray
pip install shapely scipy cloudpickle Cython netcdf4 boto3 folium hdmedians scikit-image ruamel.yaml
pip install sqlalchemy==1.2.18
pip install pycopg2-binary
```

2.6 CORE

Installa l'ultima versione del core Open Data Cube dal github Open Data Cube Core . È fondamentale selezionare una versione 1.6.10 successiva se si intende utilizzare l'indicizzazione S3. Successivamente, esegui il wheel di sviluppo della configurazione di python.

```
cd ~/Datacube
wget https://github.com/opendatacube/datacube-core/archive/datacube-1.6.1.tar.gz
tar -xf datacube-1.6.1.tar.gz
sudo mv datacube-core-datacube-1.6.1/ datacube-core
cd ~/Datacube/datacube-core
python setup.py develop
```

NOTA: dopo il comando `python setup.py develop` si è presentato l'errore sulla libreria `cachetools` (mancava) e sulla libreria `pyrsistent` (mancava anch'essa), problema risolto con l'installazione di tali librerie con il comando `pip install nomelibreria`.

2.7 Database PostgreSQL

Il database **PostgreSQL** memorizzerà i metadati che punteranno al percorso dei dati. Installa le librerie dei prerequisiti che verranno sfruttate dal ODC.

```
sudo apt-get install postgresql-10 postgresql-client-10 postgresql-contrib-10 libhdf5-serial-dev postgresql-doc-10
```

Nel file di configurazione `/etc/postgresql/10/main/postgresql.conf`, cambiare il parametro `timezone` in `UTC`. Questo parametro dovrebbe essere nella sezione intitolata `CLIENT CONNECTION DEFAULTS`.

Nel file di configurazione `/etc/postgresql/10/main/pg_hba.conf`, cambiare la linea `local` in modo che corrisponda all'esempio seguente: è una delle ultime righe nel file di configurazione. Anche la spaziatura conta, quindi fare attenzione a preservarla.

pg_hba.conf

| | local | all | postgres | ADDRESS | peer |
|--|----------|------|--------------|---------|--------|
| # TYPE | DATABASE | USER | | | METHOD |
| # "local" is for Unix domain socket connections only | | | | | |
| local | all | all | | | md5 |
| # IPv4 local connections: | | | | | |
| host | all | all | 127.0.0.1/32 | | md5 |
| # IPv6 local connections: | | | | | |
| host | all | all | :::1/128 | | md5 |

Ora che le impostazioni di **PostgreSQL** sono state modificate, riavviare il servizio:

```
sudo service postgresql restart
```

Creare un superutente **PostgreSQL** per accedere al database.

```
sudo -u postgres createuser --superuser vincenzo
sudo -u postgres psql -c "ALTER USER vincenzo WITH PASSWORD '1234';"
createdb -U vincenzo datacube
```

Ora, creare il file di configurazione di Data Cube che sarà letto durante l'inizializzazione di Data Cube.

Crea un file a `~/datacube.conf`. Il nome host deve essere impostato su `localhost` o `127.0.0.1`. Se si sta tentando di accedere a un ODC installato su un server, si utilizzerà l'indirizzo IP di quel server seguito dal numero di porta per connettersi.

Esempio: `192.168.1.5:8080` dove `192.168.1.5` trova l'IP del server ed `8080` è il numero della porta. È fondamentale che la password corrisponda alla password specificata quando è stato creato il superutente del database **PostgreSQL**. In caso contrario, Data Cube avrà un errore di autorizzazione. Di seguito viene mostrato il nostro file di configurazione.

```
[datacube]
db_hostname: 127.0.0.1
db_database: datacube
db_username: vincenzo
db_password: 1234
```

Infine, inizializza il database. Se questo passaggio fallisce, è necessario andare a ricontrollare i passaggi precedenti e assicurarsi di aver impostato correttamente tutti i file di configurazione, il database **PostgreSQL** , nonché il superutente e la password del database **PostgreSQL** .

```
datacube -v system init
```

L'output di questo comando dovrebbe essere questo

```
(datacube_env) vincenzo@vincenzo-X556UQK:~$ datacube -v system init
2019-05-08 12:23:23,074 24509 datacube INFO Running datacube command: /home/vincenzo/Datacube/datacube_env/bin/datacube -v system init
Initialising database...
2019-05-08 12:23:23,099 24509 datacube.drivers.postgres._core INFO Ensuring user roles.
2019-05-08 12:23:23,183 24509 datacube.drivers.postgres._core INFO Creating schema.
2019-05-08 12:23:23,184 24509 datacube.drivers.postgres._core INFO Creating tables.
2019-05-08 12:23:23,974 24509 datacube.drivers.postgres._core INFO Adding role grants.
2019-05-08 12:23:23,984 24509 datacube.index.index INFO Adding default metadata types.
Created.
Checking indexes/views.
2019-05-08 12:23:24,146 24509 datacube.drivers.postgres._api INFO Checking dynamic views/indexes. (rebuild views=True, indexes=False)
Done.
```

per verificare la corretta inizializzazione del database inserire il seguente comando

```
datacube system check
```

L'output che ne uscirà sarà più o meno il seguente:

```
Version: 1.6.1
Config files: /home/vincenzo/.datacube.conf
Host: 127.0.0.1:5432
Database: datacube
User: vincenzo
Environment: None
Index Driver: default

Valid connection: YES
```

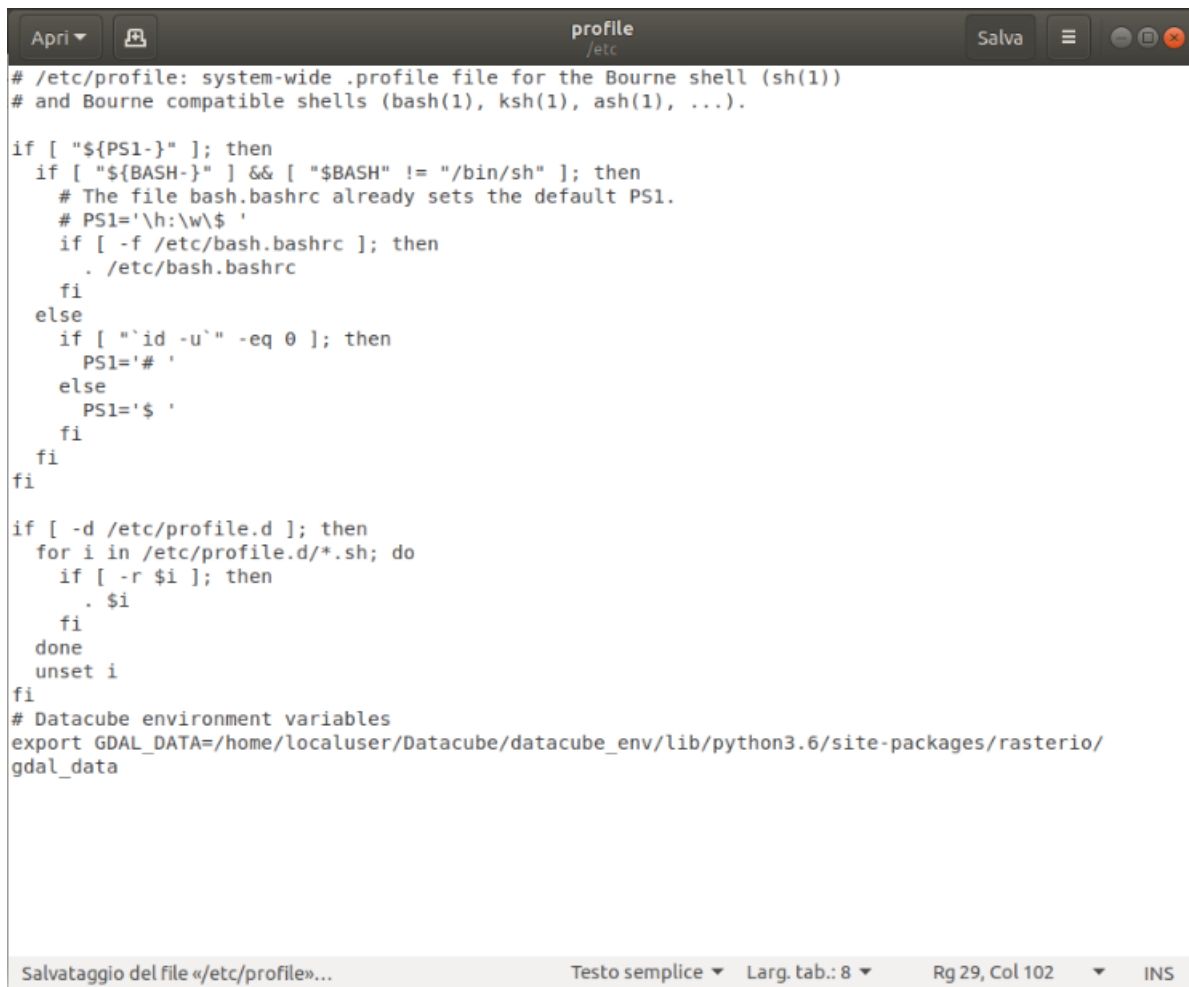
Correzioni temporanee

Al momento, le seguenti righe devono essere aggiunte alla fine del `/etc/profile` file per evitare errori relativi a una variabile d'ambiente `GDAL_DATA` durante il caricamento dei dati:

```
# Datacube environment variables
export GDAL_DATA=/home/localuser/Datacube/datacube_env/lib/python3.6/site-
packages/rasterio/gdal_data
```

e quindi eseguire `source /etc/profile` in un terminale, non è necessario riavviare il sistema per abilitare questa correzione.

Nel nostro caso `localuser` è stato sostituito con `vincenzo`.



The screenshot shows a text editor window titled 'profile /etc'. The window has a dark title bar with a 'Salva' button and standard window controls. The main area contains the content of the /etc/profile file, which is a shell script for Bourne shells. The script sets the PS1 prompt based on whether the user is root or not, and sources other profile files. At the bottom, there is a status bar with the text 'Salvataggio del file «/etc/profile»...', 'Testo semplice', 'Larg. tab.: 8', 'Rg 29, Col 102', and 'INS'.

```
# /etc/profile: system-wide .profile file for the Bourne shell (sh(1))
# and Bourne compatible shells (bash(1), ksh(1), ash(1), ...).

if [ "${PS1:-}" ]; then
  if [ "${BASH:-}" ] && [ "$BASH" != "/bin/sh" ]; then
    # The file bash.bashrc already sets the default PS1.
    # PS1='\h:\w\$ '
    if [ -f /etc/bash.bashrc ]; then
      . /etc/bash.bashrc
    fi
  else
    if [ "`id -u`" -eq 0 ]; then
      PS1='# '
    else
      PS1='$ '
    fi
  fi
fi

if [ -d /etc/profile.d ]; then
  for i in /etc/profile.d/*.sh; do
    if [ -r $i ]; then
      . $i
    fi
  done
  unset i
fi

# Datacube environment variables
export GDAL_DATA=/home/localuser/Datacube/datacube_env/lib/python3.6/site-packages/rasterio/
gdal_data
```

Screenshoot eseguito prima del cambiamento localuser -> vincenzo

CAPITOLO 3

Datacube ingestion Guide

Per indicizzare e importare dati nel Datacube, è necessario soddisfare i seguenti prerequisiti:

- Si dispone di un utente locale che viene utilizzato per eseguire i comandi / applicazioni Data Cube;
- Si dispone di un utente del database che viene utilizzato per connettersi al database 'datacube';
- Il comando 'datacube system init' è stato eseguito correttamente
- Tutto il codice è estratto e si ha un ambiente virtuale nelle directory corrette: `~/Datacube/{data_cube_ui, data_cube_notebooks, datacube_env, agdc-v2}`

Un set di dati raster georeferenziati. Questo può venire sotto forma di:

- Una serie di file GeoTiff per ogni banda
- Un file NetCDF / HDF5 contenente più bande
- Un set di dati formattato BEAM-DIMAP (.dim con .img)
- Altri set di dati raster ampiamente supportati

Metadati associati per il set di dati (XML, dettagli di denominazione dei file, ecc.). Questo include (per lo meno, più è sempre meglio):

- Data di acquisizione
- Livello di elaborazione
- Tipologia di prodotto

Il datasheet associato al tuo prodotto. Questo dovrebbe descrivere le bande, i tipi di dati, le definizioni dei flag, ecc. Se non si ha accesso a una scheda tecnica, è possibile trovare molte delle informazioni dei tipi di dati utilizzando 'gdalinfo'. L'obiettivo principale qui è identificare il tipo di dati di ciascuna banda e il valore dei nodati per ciascuna banda.

Gli unici metadati *richiesti* per i metadati "eo" predefiniti sono la piattaforma, lo strumento, il tipo di prodotto, latitudine, longitudine e tempo.

Il flusso di lavoro di ingestion consiste in:

- Creazione e aggiunta di una definizione di prodotto
- Creazione di uno script di preparazione e generazione di metadati,
- Indicizzazione dei metadati del set di dati,
- Ingerire i dataset indicizzati

Nota:

Quando abbiamo verificato effettivamente l'esistenza delle cartelle `{data_cube_ui, data_cube_notebooks, datacube_env, agdc-v2}` all'interno della directory Datacube ci siamo accorti che mancava la directory "agdc-v2". Questa cartella contiene tutti gli script, tipologia dei dataset e opzioni di ingestioni dei dataset ecc. Abbiamo rimediato a questo errore scaricando la cartella dal seguente link <https://github.com/ceos-seo/agdc-v2>

3.1 CREAZIONE E AGGIUNTA DI UNA DEFINIZIONE DI PRODOTTO

Le definizioni del prodotto definiscono gli attributi per interi set di dati e sono archiviati come file .yaml. Questi attributi includono:

- Un nome breve per il set di dati
- Una breve descrizione di ciò che è il set di dati
- Un tipo di metadati - ne parleremo più avanti
- Metadati del prodotto, tra cui piattaforma, strumento, tipo di prodotto, ecc.
- Qualsiasi numero di misurazioni associate al set di dati e ai relativi attributi.

I primi passi forniscono al set di dati un nome descrittivo, una descrizione che elenca il tipo di dati / preelaborazione / proiezione e un tipo di metadati. Il tipo di metadati può essere personalizzato dagli utenti esperti, ma per semplicità useremo il tipo di metadati EO predefinito.

```
name: ls7_collections_sr_scene
description: Landsat 7 USGS Collection 1 Higher Level SR scene processed using
LEDAPS. 30m UTM based projection.
metadata_type: eo
```

Il name è un nome breve per il set di dati: generalmente ci atteniamo a una stringa separata di maiuscole e minuscole. Questo è il nome che verrà utilizzato per accedere a set di dati di questo tipo a livello di codice e durante l'ingestione. 'eo' è il metadata_type predefinito e la descrizione è completamente definita dall'utente. È possibile creare il proprio tipo di metadati se sono desiderati campi personalizzati o set di metadati, ma per questo esempio ci atterremo al tipo di metadati predefinito.

Si può vedere nel tipo di metadati predefinito che il nome è "eo", corrispondente al nostro tipo di set di dati. Inoltre, si vedrà che il campo del set di dati in quel file include un id, una data di creazione, misurazioni, ecc. Questi campi corrispondono a ciò che deve essere generato nel passaggio successivo e alla struttura del file .yaml / .json - per ora, si tenga presente che ogni definizione di prodotto deve essere associata a un tipo di metadati.

Un elemento facoltativo, 'storage', può essere incluso. Questo può essere visto nello schema del tipo di set di dato. Questo può essere incluso per descrivere il formato dei dati sul disco, inclusi CRS, risoluzione e driver. Questi sono usati nel processo di ingestione per specificare il tipo di proiezione / piastrellatura / file, ma per ora è completamente opzionale. Questa opzione è applicabile *solo* se il set di dati ha una risoluzione e una proiezione costanti. Ad esempio, i dati Landsat utilizzano proiezioni basate su UTM (proiezione di dataset incoerente), mentre i dati GPM sono coerenti nella proiezione WGS84 con una risoluzione

di 0,1 gradi. L'elemento di latitudine è generalmente negativo assumendo che il punto di riferimento sia l'angolo in alto a sinistra.

```
storage:
  crs: EPSG:4326
  resolution:
    longitude: 0.05
    latitude: -0.05
```

Il prossimo elemento del file è il campo dei metadati:

```
metadata:
  platform:
    code: LANDSAT_7
  instrument:
    name: ETM
  product_type: LEDAPS
  format:
    name: GeoTiff
```

L'ultimo elemento (o elenco di elementi) nella definizione del prodotto è la misurazione, vista di seguito.

```
- name: 'sr_band7'
  aliases: [band_7, swir2]
  dtype: int16
  nodata: -9999
  units: 'reflectance'
```

Un modello per un tipo di set di dati può essere trovato di seguito:

```
name: { dataset type name }
description: { dataset description }
metadata_type: eo

metadata:
  platform:
    code: { Platform code }
  instrument:
    name: { Instrument name }
  product_type: { Product type }
  format:
    name: { File type format }

storage:
  crs: { CRS of the dataset, if constant }
  resolution:
    longitude: { Resolution in the x direction }
    latitude: { Resolution in the y direction - usually negative }

measurements:
  - name: { Band name }
    aliases: [{ List of aliases for this band }]
    dtype: { Dataset datatype }
    nodata: { Nodata value for the dataset - from datasheet }
```

```

units: { Band units - string }

{ Any number of measurements }

- name: { Band name }
  aliases: [{ List of aliases for this band }]
  dtype: { Dataset datatype }
  nodata: { Nodata value for the dataset - from datasheet }
  units: 'bit_index'
  flags_definition:
    { Flag name }:
      bits: [{ List of integers - what bits are valid }]
      description: { Flag description }
      values:
        1: { String description of bit }
        2: { String description of bit }
        4: { String description of bit }
        8: { String description of bit }
        16: { String description of bit }
        32: { String description of bit }
        64: { String description of bit }
        128: { String description of bit }

```

Una volta che la definizione del prodotto ha tutte le informazioni richieste, lo aggiungeremo al Datacube. Per il nostro esempio di Landsat 7, questo viene fatto con il seguente comando:

```

datacube -v product add ~/Datacube/agdc-
v2/ingest/dataset_types/landsat_collection/ls7_collections_sr_scene.yaml

```

Questo comando dovrebbe essere eseguito dall'interno dell'ambiente virtuale. Ciò convaliderà la definizione del prodotto e, se valida, indicizzerà nel Datacube. L'output previsto dovrebbe essere simile a quanto segue:

```

2017-04-19 11:23:39,861 21121 datacube INFO Running datacube command:
/home/localuser/Datacube/datacube_env/bin/datacube -v product add ~/Datacube/agdc-
v2/ingest/dataset_types/landsat_collection/ls7_sr_scenes_agdc.yaml
2017-04-19 11:23:40,184 21121 datacube.index.postgres._dynamic INFO Creating index:
dix_ls7_collections_sr_scene_lat_lon_time
2017-04-19 11:23:40,194 21121 datacube.index.postgres._dynamic INFO Creating index:
dix_ls7_collections_sr_scene_time_lat_lon
Added "ls7_collections_sr_scene"

```

3.3 GENERAZIONE DI METADATI

Prima di iniziare questo passaggio, è necessario assicurarsi di avere il set di dati scaricato e archiviato localmente su disco.

Il dataset lo abbiamo scaricato dal seguente link: <http://ec2-52-201-154-0.compute-1.amazonaws.com/datacube/data/LE071950542015121201T1-SC20170427222707.tar.gz>

Decomprimere questa scena nella posizione predefinita con i seguenti comandi:

```

cd /datacube/original_data
mkdir LE071950542015121201T1-SC20170427222707
tar -xzf LE071950542015121201T1-SC20170427222707.tar.gz -C LE071950542015121201T1-
SC20170427222707

```

Ora che abbiamo i dati estratti in una directory con lo stesso nome della scena, possiamo generare il file .yaml dei metadati richiesti. Questo è fatto con gli script Python trovati in ~/Datacube/agdc-v2/ingest/prepare_scripts/*. Sono disponibili numerosi script, tra cui USGS Landsat, Sentinel 1 e ALOS.

Questi script sono responsabili della creazione di un file di metadati .yaml che contiene tutti i campi di metadati richiesti.

```
cd ~/Datacube/agdc-v2/ingest/prepare_scripts/landsat_collection
python usgs_ls_ard_prepare.py /datacube/original_data/LE071950542015121201T1-
SC20170427222707
```

Questi script possono essere eseguiti su una singola directory o su un elenco di directory specificato con un carattere jolly (*). L'output dovrebbe essere simile a quanto mostrato di seguito:

```
2017-06-09 18:24:18,821 INFO Processing /datacube/original_data/LE071950542015121201T1-
SC20170427222707
2017-06-09 18:27:27,356 INFO Writing /datacube/original_data/LE071950542015121201T1-
SC20170427222707/datacube-metadata.yaml
```

Lo script dovrebbe produrre un file del genere, nel nostro esempio il file è il seguente:

```
acquisition:
  aos: '2015-12-12 10:28:11'
  groundstation: {code: _20}
  los: '2015-12-12 10:28:35'
creation_dt: 2015-12-12 00:00:00
extent:
  center_dt: '2015-12-12 10:28:23'
  coord:
    ll: {lat: 7.737183174881767, lon: -3.567831638761143}
    lr: {lat: 7.734497457038338, lon: -1.38312183942985}
    ul: {lat: 9.628719298511964, lon: -3.5706832614261326}
    ur: {lat: 9.625366153010912, lon: -1.375005763660132}
  from_dt: '2015-12-12 10:28:11'
  to_dt: '2015-12-12 10:28:35'
format: {name: GeoTiff}
grid_spatial:
  projection:
    geo_ref_points:
      ll: {x: 437385.0, y: 855285.0}
      lr: {x: 678315.0, y: 855285.0}
      ul: {x: 437385.0, y: 1064415.0}
      ur: {x: 678315.0, y: 1064415.0}
    spatial_reference: PROJCS["WGS 84 / UTM zone 30N",GEOGCS["WGS
84",DATUM["WGS_1984",SPHEROID["WGS
84",6378137,298.257223563,AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG","6326"]],PRIMEM["G
reenwich",0,AUTHORITY["EPSG","8901"]],UNIT["degree",0.0174532925199433,AUTHORITY["EPSG"
,"9122"]],AUTHORITY["EPSG","4326"]],PROJECTION["Transverse_Mercator"],PARAMETER["latitu
de_of_origin",0],PARAMETER["central_meridian",-
3],PARAMETER["scale_factor",0.9996],PARAMETER["false_easting",500000],PARAMETER["false_
northing",0],UNIT["metre",1,AUTHORITY["EPSG","9001"]],AXIS["Easting",EAST],AXIS["Northi
ng",NORTH],AUTHORITY["EPSG","32630"]]
  valid_data:
    coordinates:
      - - [482115.0, 1063065.0]
```

```

- [481425.73252794606, 1062321.589012033]
- [478455.66456685227, 1048971.279519252]
- [443955.6632301671, 887751.2732715366]
- [443415.5353442432, 884930.6421681236]
- [443448.16718427, 884821.583592135]
- [443881.583592135, 884448.16718427]
- [444340.6183618273, 884355.3217041101]
- [633340.6211738006, 856455.3212890928]
- [633970.7573593128, 856365.3015151902]
- [634182.3576451553, 856382.6890596801]
- [634724.3253649862, 857588.6736291265]
- [657884.333851242, 965438.7130952516]
- [672344.3555137333, 1033208.81503327]
- [672525.0, 1034175.0]
- [672404.7186338773, 1035079.0991219141]
- [482115.0, 1063065.0]
type: Polygon
id: 18163152-7388-4607-a75b-1f20e7b70045
image:
  bands:
    pixel_qa: {path: LE07_L1TP_195054_20151212_20161016_01_T1_pixel_qa.tif}
    pixel_qa_conf: {path: LE07_L1TP_195054_20151212_20161016_01_T1_pixel_qa_conf.tif}
    pixel_qa: {path: LE07_L1TP_195054_20151212_20161016_01_T1_pixel_qa.tif}
    radsat_qa: {path: LE07_L1TP_195054_20151212_20161016_01_T1_radsat_qa.tif}
    sensor_azimuth_band4: {path:
LE07_L1TP_195054_20151212_20161016_01_T1_sensor_azimuth_band4.tif}
    sensor_zenith_band4: {path:
LE07_L1TP_195054_20151212_20161016_01_T1_sensor_zenith_band4.tif}
    solar_azimuth_band4: {path:
LE07_L1TP_195054_20151212_20161016_01_T1_solar_azimuth_band4.tif}
    solar_zenith_band4: {path:
LE07_L1TP_195054_20151212_20161016_01_T1_solar_zenith_band4.tif}
    sr_atmos_opacity: {path:
LE07_L1TP_195054_20151212_20161016_01_T1_sr_atmos_opacity.tif}
    sr_band1: {path: LE07_L1TP_195054_20151212_20161016_01_T1_sr_band1.tif}
    sr_band2: {path: LE07_L1TP_195054_20151212_20161016_01_T1_sr_band2.tif}
    sr_band3: {path: LE07_L1TP_195054_20151212_20161016_01_T1_sr_band3.tif}
    sr_band4: {path: LE07_L1TP_195054_20151212_20161016_01_T1_sr_band4.tif}
    sr_band5: {path: LE07_L1TP_195054_20151212_20161016_01_T1_sr_band5.tif}
    sr_band7: {path: LE07_L1TP_195054_20151212_20161016_01_T1_sr_band7.tif}
    sr_cloud_qa: {path: LE07_L1TP_195054_20151212_20161016_01_T1_sr_cloud_qa.tif}
    satellite_ref_point_end: {x: 195, y: 54}
    satellite_ref_point_start: {x: 195, y: 54}
instrument: {name: ETM}
lineage:
  source_datasets: {}
platform: {code: LANDSAT_7}
processing_level: T1
product_type: LEDAPS

```

3.3.1 Preparazione script metadata

Il modo più semplice per generare uno di questi script è utilizzare uno script esistente come base. Duplicare lo `usgs_ls_ard_prepare.py` script e aprilo in un editor di testo. Esistono tre componenti principali nella generazione di questi metadati: analisi degli attributi dai nomi dei file, analisi dei dati dai metadati inclusi (XML o altro, potenzialmente non necessari) e acquisizione degli attributi CRS / proiezione da uno dei file del set di dati.

Il primo passo è l'analisi di tutti i dati utili dal nome del file. Andare alla funzione chiamata `prepare_datasets` nello script di preparazione appena creato.

```

def prepare_datasets(nbar_path):

    fields = re.match((r"(?P<code>LC08|LE07|LT05|LT04)"
                        r"(?P<path>[0-9]{3})"
                        r"(?P<row>[0-9]{3})"
                        r"(?P<productyear>[0-9]{4})"
                        r"(?P<productmonth>[0-9]{2})"
                        r"(?P<productday>[0-9]{2})"
                        r"(?P<collection_number>[0-9]{2})"
                        r"(?P<collection_category>RT|T1|T2)"),
nbar_path.stem).groupdict()

    fields.update({
        'processing_level':
        fields['collection_category'],
        'product_type':
        get_product_type_from_code(fields['code']),
        'creation_dt':
        datetime.datetime(int(fields['productyear']), int(fields['productmonth']),
int(fields['productday']))
    })
    nbar = prep_dataset(fields, nbar_path)
    return (nbar, nbar_path)

```

Qui, `re` viene utilizzato per analizzare i campi dal percorso del file. Modificare la `re.match` chiamata in modo che corrisponda agli attributi dei dati del nostro nome file. Il prossimo elemento sta analizzando i metadati inclusi e generando i dettagli di proiezione / CRS. Trova la definizione della funzione per `prep_dataset`:

```

def prep_dataset(fields, path):
    # getting the list of all images
    images_list = []
    for file in os.listdir(str(path)):
        if file.endswith(".xml") and (not file.endswith('aux.xml')):
            metafile = file
        if file.endswith(".tif") and ("band" in file):
            images_list.append(os.path.join(str(path), file))

    # parsing xml based metadata
    with open(os.path.join(str(path), metafile)) as f:
        xmlstring = f.read()
    xmlstring = re.sub(r'\sxmlns="[^"]+"', '', xmlstring, count=1)
    doc = ElementTree.fromstring(xmlstring)
    satellite = doc.find('.//satellite').text
    instrument = doc.find('.//instrument').text
    acquisition_date = doc.find('.//acquisition_date').text.replace("-", "")
    scene_center_time = doc.find('.//scene_center_time').text[:8]
    center_dt = crazy_parse(acquisition_date + "T" + scene_center_time)
    aos = crazy_parse(acquisition_date + "T" + scene_center_time) -
timedelta(seconds=(24 / 2))
    los = aos + timedelta(seconds=24)
    lpgs_metadata_file = doc.find('.//lpgs_metadata_file').text
    groundstation = lpgs_metadata_file[16:19]
    fields.update({'instrument': instrument, 'satellite': satellite})
    start_time = aos
    end_time = los

    # getting the paths to each of the band datasets

```

```

images = {band_name(satellite, im_path): {'path': str(im_path.relative_to(path))}}
for im_path in path.glob('*.tif')}

#getting the projection details
projdict = get_projection(path / next(iter(images.values()))['path'])
projdict['valid_data'] = safe_valid_region(images_list)

# generating the .yaml document
doc = {
    'id': str(uuid.uuid4()),
    'processing_level': fields["processing_level"],
    'product_type': fields["product_type"],
    'creation_dt': fields["creation_dt"],
    'platform': {
        'code': fields["satellite"]
    },
    'instrument': {
        'name': fields["instrument"]
    },
    'acquisition': {
        'groundstation': {
            'code': groundstation,
        },
        'aos': str(aos),
        'los': str(los)
    },
    'extent': {
        'from_dt': str(start_time),
        'to_dt': str(end_time),
        'center_dt': str(center_dt)
    },
    'format': {
        'name': 'GeoTiff'
    },
    'grid_spatial': {
        'projection': projdict
    },
    'image': {
        'satellite_ref_point_start': {
            'x': int(fields["path"]),
            'y': int(fields["row"])
        },
        'satellite_ref_point_end': {
            'x': int(fields["path"]),
            'y': int(fields["row"])
        },
        'bands': images
    },
    'lineage': {
        'source_datasets': {}
    }
}
populate_coord(doc)
return doc

```

La prima azione consiste nel generare un elenco di immagini e altri file inclusi nell'archivio. Successivamente, i metadati XML vengono aperti e analizzati per i campi desiderati. Se il set di dati non viene fornito con ulteriori metadati o se non si desidera acquisire alcuna informazione non necessaria, è possibile saltare questo passaggio.

```

# getting the list of all images
images_list = []
for file in os.listdir(str(path)):
    if file.endswith(".xml") and (not file.endswith('aux.xml')):
        metafile = file
    if file.endswith(".tif") and ("band" in file):
        images_list.append(os.path.join(str(path), file))

# parsing xml based metadata
with open(os.path.join(str(path), metafile)) as f:
    xmlstring = f.read()
xmlstring = re.sub(r'\sxmlns="[^"]+', '', xmlstring, count=1)
doc = ElementTree.fromstring(xmlstring)
satellite = doc.find('.//satellite').text
instrument = doc.find('.//instrument').text
acquisition_date = doc.find('.//acquisition_date').text.replace("-", "")
scene_center_time = doc.find('.//scene_center_time').text[:8]
center_dt = crazy_parse(acquisition_date + "T" + scene_center_time)
aos = crazy_parse(acquisition_date + "T" + scene_center_time) -
timedelta(seconds=(24 / 2))
los = aos + timedelta(seconds=24)
lpgs_metadata_file = doc.find('.//lpgs_metadata_file').text
groundstation = lpgs_metadata_file[16:19]
fields.update({'instrument': instrument, 'satellite': satellite})
start_time = aos
end_time = los

```

Successivamente, viene generato un elenco di nomi di bande e percorsi per tali bande per il set di dati. Una funzione di supporto `band_name` viene utilizzata per ottenere il nome desiderato in base al percorso del file. Nell'esempio seguente, stiamo usando `glob` per trovare tutti i file .tif e generare un dizionario (banda: percorso per ognuno).

```

# getting the paths to each of the band datasets
images = {band_name(satellite, im_path): {'path': str(im_path.relative_to(path))}
for im_path in path.glob('*.tif')}

```

Si noti che il dizionario delle bande **deve** contenere una voce per ogni misura che si è elencata nel tipo di set di dati. Se si dispone di rosso, verde e blu nel tipo di set di dati, il dizionario delle bande deve includere rosso, verde e blu.

Il prossimo passo è ottenere i punti di proiezione e di riferimento geografico da uno dei file del set di dati:

```

#getting the projection details
projdict = get_projection(path / next(iter(images.values()))['path'])
projdict['valid_data'] = safe_valid_region(images_list)

```

Questo viene fatto usando rasterio - qualsiasi dataset che può essere aperto da rasterio non dovrebbe richiedere alcuna modifica. L'ultimo passo è in realtà la generazione del dict che verrà scaricato in un file .yaml. Alcuni campi non necessari sono stati eliminati di seguito, inclusi i punti di riferimento per l'acquisizione e il satellite.

```

# generating the .yaml document
doc = {
    'id': str(uuid.uuid4()),
    'processing_level': fields["processing_level"],
    'product_type': fields["product_type"],
    'creation_dt': fields["creation_dt"],
    'platform': {
        'code': fields["satellite"]
    },
    'instrument': {
        'name': fields["instrument"]
    },
    'extent': {
        'from_dt': str(start_time),
        'to_dt': str(end_time),
        'center_dt': str(center_dt)
    },
    'format': {
        'name': 'GeoTiff'
    },
    'grid_spatial': {
        'projection': projdict
    },
    'image': {
        'bands': images
    },
    'lineage': {
        'source_datasets': {}
    }
}
populate_coord(doc)

```

Vedrai che lo script ha creato un file .yaml intitolato datacube-metadata.yaml nella stessa directory dei tuoi file di dati. Aprire questo file .yaml e verificare che tutti i campi siano stati compilati e che non vi siano errori.

3.4 INDICIZZAZIONE DEI SET DI DATI NEL DATABASE

Ora che è stata aggiunta una definizione prodotto e un file datacube-metadata.yaml generato per la scena, è giunto il momento di indicizzare il set di dati e associarlo alla definizione del prodotto. Questo viene fatto con un comando datacube dalla CLI. Si noti che l'indicizzazione dell'insieme di dati nel database crea un riferimento assoluto al percorso sul disco: non è possibile spostare il set di dati sul disco dopo l'indicizzazione o non verrà trovato e creerà problemi.

Eseguire il `datacube dataset add` comando sulla directory o sul file di metadati .yaml generato per il set di dati. Questo comando caricherà il file di metadati .yaml e creerà un oggetto di classe Dataset dai contenuti. Quindi, tenterà di far corrispondere il set di dati con una definizione di prodotto utilizzando i metadati forniti.


```
#ensure that you are doing this from within the virtual environment. If not, activate
it with 'source ~/Datacube/datacube_env/bin/activate'
datacube -v dataset add /datacube/original_data/LE071950542015121201T1-SC20170427222707
#or
#datacube -v dataset add /datacube/original_data/LE071950542015121201T1-
SC20170427222707/datacube-metadata.yaml
```

L'output della console risultante sarà simile all'output seguente:

```
2017-06-09 18:32:32,075 5086 datacube INFO Running datacube command:
/home/localuser/Datacube/datacube_env/bin/datacube -v dataset add
/datacube/original_data/LE071950542015121201T1-SC20170427222707/datacube-metadata.yaml
Indexing datasets [#####] 100%2017-06-09 18:32:32,122
5086 datacube-dataset INFO Matched Dataset <id=18163152-7388-4607-a75b-1f20e7b70045
type=ls7_collections_sr_scene location=/datacube/original_data/LE071950542015121201T1-
SC20170427222707/datacube-metadata.yaml>
2017-06-09 18:32:32,123 5086 datacube.index._datasets INFO Indexing 18163152-7388-4607-
a75b-1f20e7b70045
```

Si noti che è possibile eseguire questi strumenti da riga di comando "datacube" sugli input con caratteri jolly, ad esempio `datacube dataset add /datacube/original_data/LE*/*.yaml` nelle directory di elaborazione batch.

Se si verifica un problema con il set di dati, tornare indietro e assicurarsi che i metadati corrispondano tra il tipo di set di dati e i metadati del set di dati. Ciò comprende:

- Misure / bande
- piattaforma
- Strumento
- Tipologia di prodotto

Poiché i riferimenti del set di dati nel database utilizzano percorsi assoluti, assicurarsi che i set di dati siano organizzati e in una posizione che si desidera conservare prima dell'indicizzazione. Ora che abbiamo dimostrato il processo di indicizzazione e l'accesso ai dati, possiamo importare il set di dati.

3.5 INGERIRE I SET DI DATI

L'ingestione è il processo di trasformazione dei dataset originali in un formato più accessibile che può essere utilizzato da Data Cube. Come i passaggi precedenti, l'ingestione si basa sull'utilizzo di un file di configurazione .yaml che specifica tutti i dettagli di input e output per i dati. Il file di configurazione di importazione contiene tutte le informazioni richieste per una definizione di prodotto: utilizza le informazioni per creare una nuova definizione di prodotto e indicizzarla nel Cubo di dati. Successivamente, i set di dati di origine indicizzati che corrispondono ai criteri vengono identificati, affiancati, riproiettati e ricampionati (se necessario) e scritti su disco come unità di archiviazione NetCDF. I metadati richiesti vengono generati e i set di dati appena creati vengono indicizzati (aggiunti) al cubo di dati.

Il file di configurazione di importazione sta essenzialmente definendo una trasformazione tra l'origine e i dati di output: è possibile definire attributi come risoluzione, proiezione, dimensioni delle tile, limiti e misure, insieme al tipo di file e altri metadati in un file di configurazione. Quando si esegue il processo di importazione, il Data Cube determina quali dati si adattano agli attributi dei dati di input per tipo di prodotto e limiti, applica la trasformazione definita e salva il risultato su disco e nel database. Questo processo viene generalmente eseguito quando i dati sono in un formato di file non ottimizzato per l'accesso casuale come GeoTiff - NetCDF è il tipo di file preferito.

Si tenga presente che si dovrà creare un nuovo file di configurazione di importazione in modo che corrisponda ai limiti di scena che hai scaricato. Se non si vuole farlo, eliminare la sezione relativa ai limiti di ingestione dal file di configurazione che stiamo usando come esempio - il nostro prodotto sarà 'ls7_ledaps_general'.

In questa sezione passeremo attraverso un esempio di configurazione di importazione di Landsat 7. Un modello completo può essere trovato in fondo a questa sezione.

Le prime due righe nel file sono:

```
source_type: ls7_collections_sr_scene
output_type: ls7_ledaps_general
```

Ciò indica al processo di importazione di associare i parametri di input ai set di dati associati alla definizione di prodotto "ls7_collections_sr_scene" e di creare una nuova definizione di prodotto denominata "ls7_ledaps_general".

Successivamente, ci sono due parametri che specificano la posizione e i modelli del percorso del file per le unità di memoria appena create:

```
location: '/datacube/ingested_data'
file_path_template:
'LS7_ETM_LEDAPS/General/LS7_ETM_LEDAPS_4326_{tile_index[0]}_{tile_index[1]}_{start_time
}.nc'
```

Questo specifica che il percorso di base per i dati ingeriti è /datacube/ingested_data- questo è stato creato in precedenza e dovrebbe avere 777 permessi.

Gli elementi successivi sono elementi di metadati globali, come mostrato di seguito. Si tenga presente che i valori dello strumento / della piattaforma devono corrispondere ai metadati del set di dati e al tipo di set di dati.

```
global_attributes:
  title: CEOS Data Cube Landsat Surface Reflectance
  summary: Landsat 7 Enhanced Thematic Mapper Plus ARD prepared by NASA on behalf of
CEOS.
  source: LEDAPS surface reflectance product prepared using USGS Collection 1 data.
  institution: CEOS
  instrument: ETM
  cdm_data_type: Grid
  keywords: AU/GA,NASA/GSFC/SED/ESD/LANDSAT,REFLECTANCE,ETM+,TM,OLI,EARTH SCIENCE
```

```
keywords_vocabulary: GCMD
platform: LANDSAT_7
processing_level: L2
product_version: '2.0.0'
product_suite: USGS Landsat Collection 1
project: CEOS
coverage_content_type: physicalMeasurement
references: http://dx.doi.org/10.3334/ORNLDAAAC/1146
license: https://creativecommons.org/licenses/by/4.0/
naming_authority: gov.usgs
acknowledgment: Landsat data is provided by the United States Geological Survey
(USGS).
```

Il prossimo gruppo di campi definisce quale sottoinsieme (se esiste) del set di dati di origine che deve essere utilizzato per il processo di importazione. Se l'intero set di dati di origine deve essere ingerito, questo può essere omesso. Nell'esempio seguente, limitiamo il processo di importazione ai set di dati che corrispondono al tipo di set di dati di input che si trova tra questi limiti. Questi numeri dovrebbero essere in coordinate di latitudine e longitudine WGS84. Nella configurazione generale di ingestione, i limiti di ingestione sono esclusi.

```
ingestion_bounds:
  left: 100.0
  bottom: 5.0
  right: 115.0
  top: 20.0
```

I campi di memoria specificano alcune cose: Proiezione, risoluzione, dimensione della piastrina, dimensione del blocco, ecc.

```
storage:
  driver: NetCDF CF

  crs: EPSG:4326
  tile_size:
    longitude: 0.943231048326
    latitude: 0.943231048326
  resolution:
    longitude: 0.000269494585236
    latitude: -0.000269494585236
  chunking:
    longitude: 200
    latitude: 200
    time: 1
  dimension_order: ['time', 'latitude', 'longitude']
```

Alcune note su questi ingressi sono disponibili di seguito:

- NetCDF CF è l'unico driver di archiviazione corrente supportato
- Il CRS è definito dall'utente - sono supportate tutte le proiezioni trasversali coniche di mercator, sinusoidali o aloni coniche.

- Lo chunking e l'ordine delle dimensioni sono interni alle unità di archiviazione NetCDF: il parametro di chunking può influire sulle prestazioni in base al caso d'uso, ma generalmente lo si lascia in pace.

Note importanti

- Resolution specifica la risoluzione x / y o latitudine / longitudine nelle **unità dell'impostazione crs** - se le unità dei crs sono gradi (ad es. EPSG: 4326), allora si dovrebbe usare latitudine / longitudine qui. Se vengono usati x / y (es. Proiezioni di Mercator trasversali, UTM basato ecc.), Allora dovrebbero leggere x e y invece.
- Tile_size si riferiscono alla piastrellatura dei set di dati di origine: nell'esempio sopra, vengono utilizzati all'incirca 0,94 gradi per entrambi i valori. Ciò significa che data la scena Landsat 7 di 6 gradi quadrati, il processo di ingestione produrrà all'incirca 6 tessere separate. Questo può essere alzato o abbassato lo abbassiamo per dati a risoluzione più alta in modo che le dimensioni dei file siano all'incirca uguali.
- La dimensione della tessera deve essere equamente divisibile per la risoluzione per latitudine e longitudine - questo significa che la dimensione della piastrella di latitudine% la risoluzione della latitudine deve essere **esattamente** zero. Non farlo può comportare un singolo divario di pixel tra le tessere causato da alcuni errori di arrotondamento. Se fossimo ingerendo in Albers 25 m, una dimensione di piastrella valida sarebbe 100000, ma non 100321 in quanto non dividere equamente. Per proiezioni che richiedono gradi frazionari (come sopra), calcoliamo la risoluzione desiderata in entrambe le direzioni x e y, quindi moltiplichiamo per una costante arbitraria (generalmente 3000-4000) per garantire che non vi siano errori di arrotondamento.
- L'ultima parte del file di configurazione è l'informazione di misurazione. Questi assomigliano allo snippet visto qui sotto:

```
- name: blue
  dtype: int16
  nodata: -9999
  resampling_method: nearest
  src_varname: 'sr_band1'
  zlib: True
  attrs:
    long_name: "Surface Reflectance 0.45-0.52 microns (Blue)"
    alias: "band_1"
```

- I punti importanti da notare qui sono che contiene le stesse informazioni (o possono contenere) tutti gli stessi attributi dalla definizione del prodotto, così come le informazioni richieste per mappare le misurazioni della sorgente alle misurazioni ingerite.
- Il campo "src_varname" mappa le misurazioni dei dataset ingerite alle variabili di origine: nel caso precedente, stiamo mappando 'sr_band1' in blu. Src_varname dovrebbe essere impostato sul nome della band dal **file di metadati del set di dati** e il nome dovrebbe essere quello che si desidera venga chiamata la nuova band. Se non si desidera modificare il nome, è possibile inserire lo stesso valore per il nome.
- Di seguito è disponibile un modello di base per la configurazione di importazione:

```

source_type: { Your dataset type name }
output_type: { The new name for the ingested product. This can not be the same as
the source. }

description: { Description of the ingested data }

location: { Base location to store the NetCDF storage units }
file_path_template: { file naming template - standard Python string replacement,
available variables are tile_index[0]/tile_index[1]/start_time }
global_attributes:
  title:
  summary:
  source:
  institution:
  instrument: { This should match your dataset type/metadata }
  cdm_data_type:
  keywords:
  keywords_vocabulary:
  platform: { This should match your dataset type/metadata }
  processing_level: { This should match your dataset type/metadata }
  product_version:
  product_suite:
  project:
  coverage_content_type:
  references:
  license:
  naming_authority:
  acknowledgment:

{ Ingestion bounds in your selected CRS's units (WGS84-> Lat/lon, Albers-
>Meters). If none, delete this block }
ingestion_bounds:
  left:
  bottom:
  right:
  top:

storage:
  driver: NetCDF CF

  crs: { Desired CRS - EPSG }
  tile_size:
    longitude: { Size of each storage unit, in the units of your CRS.
Ensure that this is evenly divisible by your resolution }
    latitude: { Size of each storage unit, in the units of your CRS.
Ensure that this is evenly divisible by your resolution }
  resolution:
    longitude: { Resolution for your dataset in the units of your CRS }
    latitude: { Resolution for your dataset in the units of your CRS.
Latitude should be a negative value}
  chunking:
    longitude: 200
    latitude: 200
    time: 1
  dimension_order: ['time', 'latitude', 'longitude']

measurements:
  - name: { Desired name value - this can be the same as the dataset type }
    dtype: { Desired dtype value - this can be the same as the dataset type }
    nodata: { Desired nodata value - this can be the same as the dataset type }
    resampling_method: nearest
    src_varname: { Band name from dataset type/dataset metadata }

```

```
zlib: True
attrs:
  long_name: {}
  alias: {}
{ Any number of measurements }
```

Ora che abbiamo un file di configurazione di importazione completo, siamo in grado di avviare il processo di importazione. Utilizza il seguente frammento di codice:

```
datacube -v ingest -c ~/Datacube/agdc-
v2/ingest/ingestion_configs/landsat_collection/ls7_collections_sr_general.yaml --
executor multiproc 2
```

Si Nota alcune cose nel comando precedente: -c è l'opzione per il file di configurazione e --executor multiproc abilita il multiprocessing. Si dovrà vedere una quantità significativa di output della console e uno stato di aggiornamento costante fino al termine dell'ingestione. Con le nostre tessere di ~ 1 grado, puoi anche vedere che stiamo producendo 9 tessere per l'acquisizione singola a causa delle nostre impostazioni di piastrellatura. L'output della console può essere visto di seguito:

```
2017-06-11 12:02:31,965 12986 datacube INFO Running datacube command:
/home/localuser/Datacube/datacube_env/bin/datacube -v ingest -c
/home/localuser/Datacube/agdc-
v2/ingest/ingestion_configs/landsat_collection/ls7_collections_sr_general.yaml --
executor multiproc 2
2017-06-11 12:02:32,022 12986 agdc-ingest INFO Created DatasetType ls7_ledaps_general
2017-06-11 12:02:32,044 12986 datacube.index._datasets INFO No changes detected for
product ls7_ledaps_general
2017-06-11 12:02:32,075 12986 agdc-ingest INFO 8 tasks discovered
2017-06-11 12:02:32,083 12986 agdc-ingest INFO Submitting task: (-2, 8,
numpy.datetime64('2015-12-12T10:28:23.000000000'))
2017-06-11 12:02:32,110 12986 agdc-ingest INFO Submitting task: (-4, 10,
numpy.datetime64('2015-12-12T10:28:23.000000000'))
2017-06-11 12:02:32,121 12986 agdc-ingest INFO Submitting task: (-4, 9,
numpy.datetime64('2015-12-12T10:28:23.000000000'))
2017-06-11 12:02:32,122 12986 agdc-ingest INFO Submitting task: (-4, 8,
numpy.datetime64('2015-12-12T10:28:23.000000000'))
2017-06-11 12:02:32,129 12986 agdc-ingest INFO Submitting task: (-2, 9,
numpy.datetime64('2015-12-12T10:28:23.000000000'))
2017-06-11 12:02:32,129 12986 agdc-ingest INFO Submitting task: (-3, 8,
numpy.datetime64('2015-12-12T10:28:23.000000000'))
2017-06-11 12:02:32,130 12986 agdc-ingest INFO Submitting task: (-3, 10,
numpy.datetime64('2015-12-12T10:28:23.000000000'))
2017-06-11 12:02:32,134 12986 agdc-ingest INFO Submitting task: (-3, 9,
numpy.datetime64('2015-12-12T10:28:23.000000000'))
2017-06-11 12:02:32,135 12986 agdc-ingest INFO completed 0, failed 0, pending 8
2017-06-11 12:02:32,136 12997 agdc-ingest INFO Starting task (-2, 8,
numpy.datetime64('2015-12-12T10:28:23.000000000'))
2017-06-11 12:02:32,142 12998 agdc-ingest INFO Starting task (-4, 10,
numpy.datetime64('2015-12-12T10:28:23.000000000'))
2017-06-11 12:02:33,139 12986 agdc-ingest INFO completed 0, failed 0, pending 8
2017-06-11 12:02:34,140 12986 agdc-ingest INFO completed 0, failed 0, pending 8
2017-06-11 12:02:35,142 12986 agdc-ingest INFO completed 0, failed 0, pending 8
2017-06-11 12:02:35,341 12998 datacube.storage.storage INFO Creating storage unit:
/datacube/ingested_data/LS7_ETM_LEDAPS/General/LS7_ETM_LEDAPS_4326_-
4_10_20151212102823000000.nc
```

2017-06-11 12:02:36,076 12997 datacube.storage.storage INFO Creating storage unit:
/datacube/ingested_data/LS7_ETM_LEDAPS/General/LS7_ETM_LEDAPS_4326_-
2_8_20151212102823000000.nc
2017-06-11 12:02:36,145 12986 agdc-ingest INFO completed 0, failed 0, pending 8
2017-06-11 12:02:37,147 12986 agdc-ingest INFO completed 0, failed 0, pending 8
2017-06-11 12:02:37,843 12998 agdc-ingest INFO Finished task (-4, 10,
numpy.datetime64('2015-12-12T10:28:23.000000000'))
2017-06-11 12:02:37,853 12998 agdc-ingest INFO Starting task (-4, 9,
numpy.datetime64('2015-12-12T10:28:23.000000000'))
2017-06-11 12:02:38,148 12986 agdc-ingest INFO completed 1, failed 0, pending 7
2017-06-11 12:02:38,149 12986 datacube.index._datasets INFO Indexing e72bfba8-68e8-
4305-a901-0cc4cddb8114
2017-06-11 12:02:38,199 12986 agdc-ingest INFO completed 0, failed 0, pending 7
2017-06-11 12:02:38,556 12997 agdc-ingest INFO Finished task (-2, 8,
numpy.datetime64('2015-12-12T10:28:23.000000000'))
2017-06-11 12:02:38,563 12997 agdc-ingest INFO Starting task (-4, 8,
numpy.datetime64('2015-12-12T10:28:23.000000000'))
2017-06-11 12:02:39,200 12986 agdc-ingest INFO completed 1, failed 0, pending 6
2017-06-11 12:02:39,201 12986 datacube.index._datasets INFO Indexing d5eb550b-faf7-
4e47-9714-e50a5328c91e
2017-06-11 12:02:39,251 12986 agdc-ingest INFO completed 0, failed 0, pending 6
2017-06-11 12:02:40,252 12986 agdc-ingest INFO completed 0, failed 0, pending 6
2017-06-11 12:02:41,254 12986 agdc-ingest INFO completed 0, failed 0, pending 6
2017-06-11 12:02:42,228 12998 datacube.storage.storage INFO Creating storage unit:
/datacube/ingested_data/LS7_ETM_LEDAPS/General/LS7_ETM_LEDAPS_4326_-
4_9_20151212102823000000.nc
2017-06-11 12:02:42,256 12986 agdc-ingest INFO completed 0, failed 0, pending 6
2017-06-11 12:02:42,643 12997 datacube.storage.storage INFO Creating storage unit:
/datacube/ingested_data/LS7_ETM_LEDAPS/General/LS7_ETM_LEDAPS_4326_-
4_8_20151212102823000000.nc
2017-06-11 12:02:43,257 12986 agdc-ingest INFO completed 0, failed 0, pending 6
2017-06-11 12:02:44,259 12986 agdc-ingest INFO completed 0, failed 0, pending 6
2017-06-11 12:02:45,261 12986 agdc-ingest INFO completed 0, failed 0, pending 6
2017-06-11 12:02:45,728 12998 agdc-ingest INFO Finished task (-4, 9,
numpy.datetime64('2015-12-12T10:28:23.000000000'))
2017-06-11 12:02:45,736 12998 agdc-ingest INFO Starting task (-2, 9,
numpy.datetime64('2015-12-12T10:28:23.000000000'))
2017-06-11 12:02:45,840 12997 agdc-ingest INFO Finished task (-4, 8,
numpy.datetime64('2015-12-12T10:28:23.000000000'))
2017-06-11 12:02:45,847 12997 agdc-ingest INFO Starting task (-3, 8,
numpy.datetime64('2015-12-12T10:28:23.000000000'))
2017-06-11 12:02:46,262 12986 agdc-ingest INFO completed 2, failed 0, pending 4
2017-06-11 12:02:46,263 12986 datacube.index._datasets INFO Indexing 6ab04a04-5d8f-
4898-a7a8-debf170d4a1d
2017-06-11 12:02:46,295 12986 datacube.index._datasets INFO Indexing 0890b27b-1444-
4f7f-96ef-d83e05b31fe8
2017-06-11 12:02:46,310 12986 agdc-ingest INFO completed 0, failed 0, pending 4
2017-06-11 12:02:47,311 12986 agdc-ingest INFO completed 0, failed 0, pending 4
2017-06-11 12:02:48,313 12986 agdc-ingest INFO completed 0, failed 0, pending 4
2017-06-11 12:02:49,315 12986 agdc-ingest INFO completed 0, failed 0, pending 4
2017-06-11 12:02:49,803 12998 datacube.storage.storage INFO Creating storage unit:
/datacube/ingested_data/LS7_ETM_LEDAPS/General/LS7_ETM_LEDAPS_4326_-
2_9_20151212102823000000.nc
2017-06-11 12:02:50,316 12986 agdc-ingest INFO completed 0, failed 0, pending 4
2017-06-11 12:02:50,557 12997 datacube.storage.storage INFO Creating storage unit:
/datacube/ingested_data/LS7_ETM_LEDAPS/General/LS7_ETM_LEDAPS_4326_-
3_8_20151212102823000000.nc
2017-06-11 12:02:51,318 12986 agdc-ingest INFO completed 0, failed 0, pending 4
2017-06-11 12:02:52,319 12986 agdc-ingest INFO completed 0, failed 0, pending 4
2017-06-11 12:02:52,785 12998 agdc-ingest INFO Finished task (-2, 9,
numpy.datetime64('2015-12-12T10:28:23.000000000'))


```
2017-06-11 12:02:52,791 12998 agdc-ingest INFO Starting task (-3, 10,
numpy.datetime64('2015-12-12T10:28:23.000000000'))
2017-06-11 12:02:53,320 12986 agdc-ingest INFO completed 1, failed 0, pending 3
2017-06-11 12:02:53,320 12986 datacube.index._datasets INFO Indexing 751aa80a-9039-
4a69-8270-a93a8b1de886
2017-06-11 12:02:55,762 12998 datacube.storage.storage INFO Creating storage unit:
/datacube/ingested_data/LS7_ETM_LEDAPS/General/LS7_ETM_LEDAPS_4326_-
3_10_20151212102823000000.nc
2017-06-11 12:02:56,122 12986 agdc-ingest INFO completed 0, failed 0, pending 3
2017-06-11 12:02:56,409 12997 agdc-ingest INFO Finished task (-3, 8,
numpy.datetime64('2015-12-12T10:28:23.000000000'))
2017-06-11 12:02:56,415 12997 agdc-ingest INFO Starting task (-3, 9,
numpy.datetime64('2015-12-12T10:28:23.000000000'))
2017-06-11 12:02:57,123 12986 agdc-ingest INFO completed 1, failed 0, pending 2
2017-06-11 12:02:57,124 12986 datacube.index._datasets INFO Indexing c74f0f95-746d-
44ef-825e-7bcfc0171ac1
2017-06-11 12:02:57,154 12986 agdc-ingest INFO completed 0, failed 0, pending 2
2017-06-11 12:02:58,156 12986 agdc-ingest INFO completed 0, failed 0, pending 2
2017-06-11 12:02:58,520 12998 agdc-ingest INFO Finished task (-3, 10,
numpy.datetime64('2015-12-12T10:28:23.000000000'))
2017-06-11 12:02:59,157 12986 agdc-ingest INFO completed 1, failed 0, pending 1
2017-06-11 12:02:59,158 12986 datacube.index._datasets INFO Indexing 445991e2-d90e-
4825-a62b-bc051534a9e7
2017-06-11 12:03:01,590 12997 datacube.storage.storage INFO Creating storage unit:
/datacube/ingested_data/LS7_ETM_LEDAPS/General/LS7_ETM_LEDAPS_4326_-
3_9_20151212102823000000.nc
2017-06-11 12:03:08,043 12986 agdc-ingest INFO completed 0, failed 0, pending 1
2017-06-11 12:03:09,044 12986 agdc-ingest INFO completed 0, failed 0, pending 1
2017-06-11 12:03:10,045 12986 agdc-ingest INFO completed 0, failed 0, pending 1
2017-06-11 12:03:11,046 12986 agdc-ingest INFO completed 0, failed 0, pending 1
2017-06-11 12:03:12,049 12986 agdc-ingest INFO completed 0, failed 0, pending 1
2017-06-11 12:03:13,051 12986 agdc-ingest INFO completed 0, failed 0, pending 1
2017-06-11 12:03:13,244 12997 agdc-ingest INFO Finished task (-3, 9,
numpy.datetime64('2015-12-12T10:28:23.000000000'))
2017-06-11 12:03:14,053 12986 agdc-ingest INFO completed 1, failed 0, pending 0
2017-06-11 12:03:14,053 12986 datacube.index._datasets INFO Indexing 39b9a5a5-840e-
4bc2-ade7-d682ec8a4e56
8 successful, 0 failed
```


CAPITOLO 4

Data Cube Jupyter Notebook

Jupyter Notebook è estremamente utile come strumento di apprendimento e come caso introduttivo per il Datacube. Gli esempi di notebook Jupyter includono molti algoritmi e alcune API di base introduttive di Data Cube. Dopo aver installato tutti i pacchetti richiesti, verificheremo che l'installazione di Data Cube funzioni correttamente.

4.1 Processo di installazione

Il repository Notebook può essere scaricato come segue:

```
cd ~/Datacube
git clone https://github.com/ceos-seo/data_cube_notebooks.git
cd data_cube_notebooks
git submodule init && git submodule update
```

Ora installare i seguenti pacchetti Python:

```
pip install jupyter matplotlib scipy hdmedians rasterstats seaborn sklearn scikit-image
lcmap-pyccd==2017.6.8
pip install bokeh geopandas descartes
```

4.2 Configurazione

Il primo passo è generare un file di configurazione del notebook. Assicuriamoci di essere nell'ambiente virtuale. In caso contrario, attivare con
`source ~/Datacube/datacube_env/bin/activate.`

Quindi eseguire i seguenti comandi:

```
cd ~/Datacube/data_cube_notebooks
jupyter notebook --generate-config
jupyter nbextension enable --py --sys-prefix widgetsnbextension
```

Jupyter creerà un file di configurazione in `~/.jupyter/jupyter_notebook_config.py`. Ora impostare la password e modificare i dettagli del server. Ricordare questa password per riferimenti futuri.

```
jupyter notebook password
```

Ora modificare il file di configurazione del notebook

Jupyter `~/.jupyter/jupyter_notebook_config.py` per includere questi dettagli rilevanti:

```
c.NotebookApp.ip = '0.0.0.0'
```

```
c.NotebookApp.open_browser = False  
c.NotebookApp.port = 8080
```

Salvare il file e quindi eseguire il server notebook con il seguente comando. Se ciò non riesce con un errore di autorizzazioni (`OSError: [...] Permission denied [...]`), eseguire il comando `export XDG_RUNTIME_DIR=""`.

```
cd ~/Datacube/data_cube_notebooks  
jupyter notebook
```

Aprire una pagina Web e accedere all'URL del notebook. Se si esegue il browser dallo stesso computer che ospita i notebook, è possibile utilizzare `localhost:{jupyter_port_num}` come URL, dove `jupyter_port_num` è impostato il numero di porta `c.NotebookApp.port` nel file di configurazione. Se ci si connette da un altro computer, sarà necessario inserire l'indirizzo IP pubblico del server nell'URL (che può essere determinato eseguendo il comando `ifconfig` sul server) al posto di `localhost`. Comparirà un campo password. Immettere la password dal passaggio precedente.

CAPITOLO 5

Data Cube UI Installation

5.1 Processo di installazione

L'interfaccia utente può essere scaricata come segue:

```
cd ~/Datacube
git clone https://github.com/ceos-seo/data_cube_ui.git
cd data_cube_ui
git submodule init && git submodule update
```

Il processo di installazione include sia pacchetti a livello di sistema che pacchetti Python. Si avrà bisogno di avere l'ambiente virtuale attivato per questa intera guida.

Eseguire i seguenti comandi per installare Apache, pacchetti relativi a Apache, Redis e librerie di elaborazione immagini.

```
sudo apt-get install apache2 libapache2-mod-wsgi-py3 redis-server libfreeimage3
imagemagick
sudo service redis-server start
```

Successivamente, si avrà bisogno di vari pacchetti Python che sono responsabili per l'esecuzione dell'applicazione:

```
pip install django==1.11.13 redis imageio django-bootstrap3 matplotlib stringcase
celery
```

Sarà inoltre necessario creare una struttura di directory di base per i risultati:

```
sudo mkdir /datacube/ui_results
sudo chmod 777 /datacube/ui_results
```

L'interfaccia utente di Data Cube invia anche la posta di amministratore, quindi è richiesto un server di posta. Assicurati di configurarlo come un sito internet.

```
sudo apt-get install -y mailutils
```

Apportare le modifiche necessarie a `/etc/postfix/main.cf`:

```
myhostname = {your site name here}
mailbox_size_limit = 0
recipient_delimiter = +
inet_interfaces = localhost
```

e lancia `sudo service postfix restart`.

Per verificare l'installazione e l'installazione del server di posta, eseguire il seguente comando. Cambia `your_email@mail.com` al tuo indirizzo e-mail.

```
echo "Body of the email" | mail -s "The subject line" your_email@mail.com
```

Con tutti i pacchetti sopra installati, è ora possibile passare alla fase di configurazione.

5.2 Configurazione del server

La configurazione della nostra applicazione implica che tutti i nomi utente e le password siano elencati in modo accurato nei file di configurazione richiesti, spostando tali file di configurazione nelle posizioni corrette e abilitando l'intero sistema.

Il primo passo è controllare i file di configurazione di Data Cube e Apache. Se questi non sono già stati configurati, apri `~/Datacube/data_cube_ui/config/.datacube.conf` e assicurarsi che il nome utente, la password e il nome del database corrispondano tutti. Questo dovrebbe essere il nome utente e la password del database **durante il processo di installazione di Data Cube Core**. Se questi dettagli non sono corretti, correggere e salva il file.

Si noti che la nostra applicazione UI utilizza questo file di configurazione per tutto invece che per il `~/datacube.conf` file predefinito.

Successivamente, dovremo aggiornare il file di configurazione di Apache. Apri il file trovato SU `~/Datacube/data_cube_ui/config/dc_ui.conf`:

```
<VirtualHost *:80>
    # The ServerName directive sets the request scheme, hostname and port that
    # the server uses to identify itself. This is used when creating
    # redirection URLs. In the context of virtual hosts, the ServerName
    # specifies what hostname must appear in the request's Host: header to
    # match this virtual host. For the default virtual host (this file) this
    # value is not decisive as it is used as a last resort host regardless.
    # However, you must set it for any further virtual host explicitly.
    #ServerName www.example.com

    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html

    # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
    # error, crit, alert, emerg.
    # It is also possible to configure the loglevel for particular
    # modules, e.g.
    #LogLevel info ssl:warn

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

    # For most configuration files from conf-available/, which are
    # enabled or disabled at a global level, it is possible to
    # include a line for only one particular virtual host. For example the
    # following line enables the CGI configuration for this host only
    # after it has been globally disabled with "a2disconf".
    #Include conf-available/serve-cgi-bin.conf

    # django wsgi
```

```

WSGIScriptAlias / /home/localuser/Datacube/data_cube_ui/data_cube_ui/wsgi.py
WSGIDaemonProcess dc_ui python-home=/home/localuser/Datacube/datacube_env
python-path=/home/localuser/Datacube/data_cube_ui

WSGIProcessGroup dc_ui
WSGIApplicationGroup %{GLOBAL}

<Directory "/home/localuser/Datacube/data_cube_ui/data_cube_ui/">
    Require all granted
</Directory>

#django static
Alias /static/ /home/localuser/Datacube/data_cube_ui/static/
<Directory /home/localuser/Datacube/data_cube_ui/static>
    Require all granted
</Directory>

#results.
Alias /datacube/ui_results/ /datacube/ui_results/
<Directory /datacube/ui_results/>
    Require all granted
</Directory>

</VirtualHost>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet

```

In questo file di configurazione, si noti che tutti i percorsi sono assoluti. Se hai usato un nome utente diverso (diverso da 'localuser'), cambia tutte le istanze di 'localuser' nel tuo nome utente. Questo file presuppone un'installazione standard con un ambiente virtuale situato nella posizione specificata nella documentazione di installazione.

Ora copiamo i file di configurazione dove devono essere. Il `.datacube.conf` file viene copiato nella directory home per coerenza.

```

sudo cp ~/Datacube/data_cube_ui/config/.datacube.conf ~/.datacube.conf
sudo cp ~/Datacube/data_cube_ui/config/dc_ui.conf /etc/apache2/sites-
available/dc_ui.conf

```

Il passo successivo è modificare le credenziali che si trovano nelle impostazioni di Django. Apri il `settings.py` file trovato su `~/Datacube/data_cube_ui/data_cube_ui/settings.py`. Ci sono alcuni piccoli cambiamenti che devono essere fatti per coerenza con le tue impostazioni.

Il `MASTER_NODE` impostazione si riferisce a una configurazione cluster / distribuita. Questo dovrebbe rimanere `'127.0.0.1'` sulla macchina principale, mentre le altre macchine inseriranno qui l'indirizzo IP della macchina principale. Ad esempio, se l'IP pubblico della macchina principale è `52.200.156.1`, i nodi worker entreranno come `52.200.156.1` come `MASTER_NODE` valore.

```
MASTER_NODE = '127.0.0.1'
```

Anche le impostazioni dell'applicazione sono definibili. Cambia l'`BASE_HOST` impostazione nell'URL a cui l'applicazione sarà accessibile. Ad esempio, per il nostro sviluppo interno

abbiamo il server in esecuzione su IP 192.168.100.13, quindi entreremo 192.168.100.13 lì. L'impostazione ADMIN_EMAIL dovrebbe essere l'email che si vuole che l'interfaccia utente invii come e-mail.. L'attivazione e il feedback dell'e-mail saranno inviati dall'indirizzo e-mail qui. L' host e la porta sono configurabili in base a dove si trova il proprio server di posta. Lo lasciamo girare localmente sulla porta 25.

```
# Application definition
BASE_HOST = "localhost:8000/"
ADMIN_EMAIL = "admin@ceos-cube.org"
EMAIL_HOST = 'localhost'
EMAIL_PORT = '25'
```

Quindi, sostituire 'localuser' con qualunque sia l'utente del sistema locale. Questo corrisponde ai valori che hai inserito nel file di configurazione di Apache.

```
LOCAL_USER = "vincenzo"
```

Anche le credenziali del database devono essere inserite qui. Inserisci il nome del database, il nome utente e la password che hai inserito nel tuo .datacube.conf file:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'datacube',
        'USER': 'vincenzo',
        'PASSWORD': '1234',
        'HOST': MASTER_NODE
    }
}
```

Ora che il file di configurazione di Apache è a posto e le impostazioni di Django sono state impostate, ora abileremo il sito e disableremo il default. Usa i comandi elencati di seguito:

```
sudo a2dissite 000-default.conf
sudo a2ensite dc_ui.conf
sudo service apache2 restart
```

Inoltre, a .pgpass è richiesto per la funzionalità Data Cube On Demand. Modifica il .pgpass nella directory di configurazione con il nome utente e la password del database di cui sopra e copiarlo nella directory home dell'utente locale.

```
sudo cp config/.pgpass ~/.pgpass
sudo chmod 600 ~/.pgpass
```

5.3 Inizializzazione del database

Ora che tutti i requisiti sono stati installati e tutti i dettagli di configurazione sono stati impostati, è il momento di inizializzare il database.

Django gestisce automaticamente tutte le modifiche del database attraverso il modello ORM/migrations. Quando ci sono cambiamenti nei `models.py` file, Django li rileva e crea "migrazioni" che apportano modifiche al database in base alle modifiche di Python. Ciò richiede ora un'inizializzazione per creare gli schemi di base.

```
cd ~/Datacube/data_cube_ui
python manage.py makemigrations
{data_cube_ui,accounts,coastal_change,custom_mosaic_tool,fractional_cover,ndvi_anomaly,
slip,task_manager,tsm,water_detection,dc_algorithm,data_cube_manager,cloud_coverage,urb
anization,spectral_indices}
python manage.py makemigrations
python manage.py migrate

python manage.py loaddata db_backups/init_database.json
```

Questa stringa di comandi effettua le migrazioni per tutte le applicazioni e crea tutti gli schemi iniziali del database. L'ultimo comando carica i dati di esempio di default che usiamo, incluse alcune aree, tipi di risultati, ecc.

NOTA:

Se dopo il comando `'python manage.py makemigrations'` comparirà questo genere di output:

```
You are trying to add a non-nullable field 'query_type' to ndvtask without a default; we can't do that (the database needs something to populate existing rows).
Please select a fix:
 1) Provide a one-off default now (will be set on all existing rows with a null value for this column)
 2) Quit, and let me add a default in models.py
Select an option: 1
Please enter the default value now, as valid Python
The datetime and django.utils.timezone modules are available, so you can do e.g. timezone.now
Type 'exit' to exit this prompt
```

Scegliere l'opzione 1 ed inserire il valore `timezone.now`.

Quindi ora, crea un account utente super sull'interfaccia utente per uso personale:

```
python manage.py createsuperuser
```

Ora che abbiamo tutto inizializzato, possiamo vedere il sito e vedere cosa stiamo creando. Visita il sito nel tuo browser Web: tramite IP da un computer esterno o dall'URL `localhost` all'interno della macchina. Ora dovresti vedere una pagina di introduzione. Accedi utilizzando uno dei pulsanti e visualizza lo Strumento Mosaico personalizzato. Vedremo tutte le aree predefinite. Questo non dà accesso a tutte queste aree perché sono esempi senza dati associati. Sarà necessario aggiungere le proprie aree e rimuovere i valori predefiniti.

Visitare il pannello di amministrazione andando a uno `{IP}/admin` o `localhost/admin`. Vedrai una pagina che mostra tutti i vari modelli e valori predefiniti.

5.4 Avviare i workers

Utilizziamo i workers di Celery nella nostra applicazione per gestire l'elaborazione di task asincroni. Usiamo `tmux` gestire più finestre distaccate per eseguire le cose in background. In futuro, passeremo ai processi demone, ma per ora ci piace essere in grado di vedere l'output di debug. Per l'implementazione corrente, vengono utilizzate più istanze di lavoro, una per l'elaborazione di attività generali e una per la funzionalità di gestione di Data Cube. L'operatore del gestore Data Cube ha alcuni parametri specifici che rendono alcune delle operazioni di creazione e cancellazione del database un po' più agevoli.

Apri due nuove sessioni di terminale e attiva l'ambiente virtuale in entrambi:

```
source ~/Datacube/datacube_env/bin/activate
cd ~/Datacube/data_cube_ui
```

Nel primo terminale, eseguire il processo di celery con:

```
celery -A data_cube_ui worker -l info -c 4
```

Nel secondo terminale, eseguire la coda di gestione datacube monouso.

```
celery -A data_cube_ui worker -l info -c 2 -Q data_cube_manager --max-tasks-per-child 1 -Ofair
```

Inoltre, si può eseguire entrambi contemporaneamente usando `celery multi`:

```
celery multi start -A data_cube_ui task_processing data_cube_manager -c:task_processing 10 -c:data_cube_manager 2 --max-tasks-per-child:data_cube_manager=1 -Q:data_cube_manager data_cube_manager -Ofair
```

Per avviare l'utilità di pianificazione, eseguire il seguente comando:

```
celery -A data_cube_ui beat
```

Per testare i workers sarà necessario aggiungere un'area e un set di dati che sono stati inseriti nel database della UI.

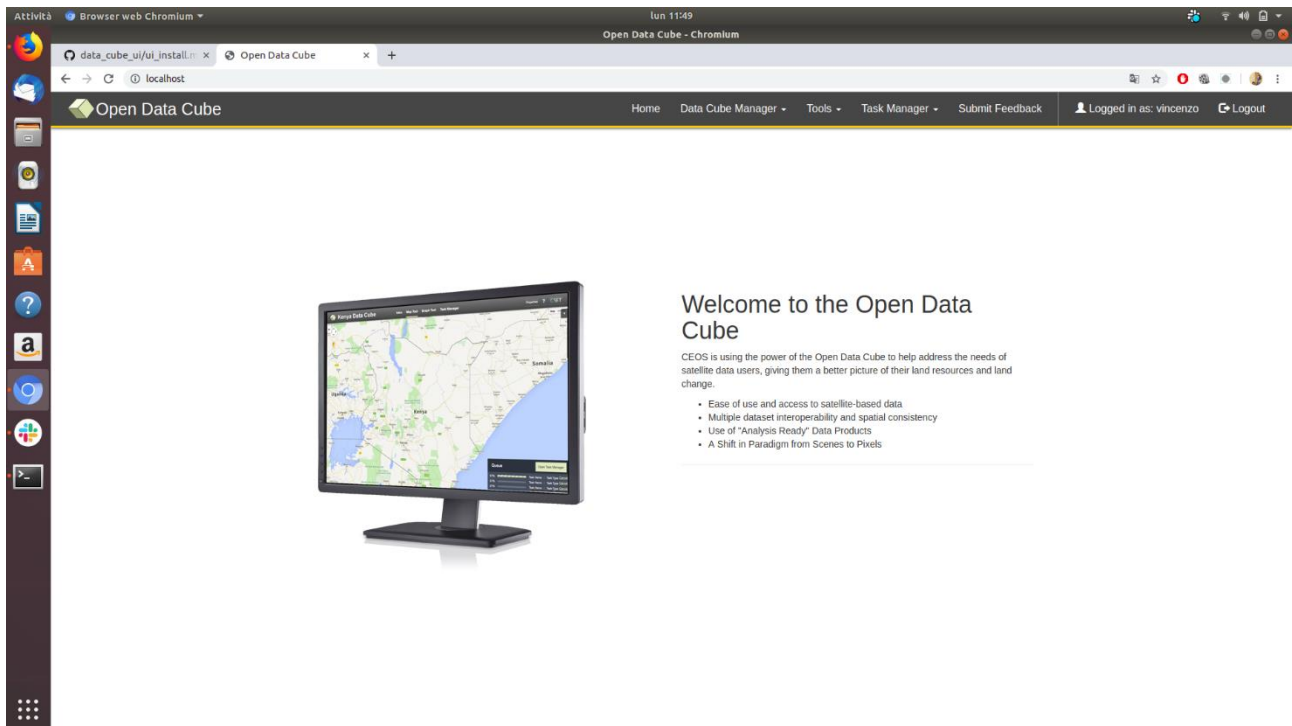
Si può automatizzare questo processo con le seguenti linee di codice

```
sudo cp config/celeryd_conf /etc/default/data_cube_ui && sudo cp config/celeryd /etc/init.d/data_cube_ui
sudo chmod 777 /etc/init.d/data_cube_ui
sudo chmod 644 /etc/default/data_cube_ui
sudo /etc/init.d/data_cube_ui start
```



```
sudo cp config/celerybeat_conf /etc/default/celerybeat && sudo cp config/celerybeat
/etc/init.d/celerybeat
sudo chmod 777 /etc/init.d/celerybeat
sudo chmod 644 /etc/default/celerybeat
sudo /etc/init.d/celerybeat start
```

Comparirà un output che dirà che "localuser" non è giusto come nome utente, andare nei file indicati nei comandi precedenti e cambiare "localuser" con il nome dell'utente, nel nostro caso l'abbiamo cambiato con "vincenzo".



5.5 Panoramica del sistema di attività

Il sistema di lavoro può sembrare complesso all'inizio, ma il flusso di lavoro di base è mostrato di seguito:

- La vista Django riceve i dati del modulo dalla pagina web. I dati di questo modulo vengono elaborati in un modello di query per l'applicazione;
- L'operatore principale di Celery riceve un'attività con un modello di query e recupera tutti i parametri richiesti da questo modello;
- Utilizzando le opzioni di chunking predefinite, l'attività principale di Celery divide i parametri (latitudine, longitudine, tempo) in blocchi più piccoli;
- Questi pezzi più piccoli di (latitudine, longitudine, tempo) vengono inviati ai processi di lavoro di Celery - dovrebbero esserci più processi di lavoro rispetto ai processi principali;
- L'operatore di Celery elabora i dati nei parametri ricevuti e esegue alcune analisi. I risultati vengono salvati su disco e i percorsi vengono restituiti;

- Il processo principale attende fino a quando tutti i blocchi sono stati elaborati, quindi carica tutti i blocchi dei risultati. I blocchi vengono uniti in un unico prodotto e salvati su disco;
- Il processo principale utilizza il prodotto di dati per creare immagini e prodotti di dati e li salva su disco, eliminando tutti i rimanenti prodotti chunk;
- Il processo principale crea un modello di risultati e metadati basato su ciò che è stato appena creato e restituisce i dettagli al browser;

5.6 Personalizza l'interfaccia utente

Per completare la configurazione, sarà necessario creare un'area e un prodotto che hai ingerito. Per questa sezione, formuliamo alcune ipotesi:

- Il nome della tua definizione di prodotto ingerito è 'ls7_ledaps_general'.
- Hai ingerito una scena di Landsat 7.

Per prima cosa, dobbiamo trovare il riquadro di delimitazione della tua area. Apri una shell Django Python e usa i seguenti comandi:

```
source ~/Datacube/datacube_env/bin/activate
cd ~/Datacube/data_cube_ui
python manage.py shell

from utils.data_cube_utilities import data_access_api

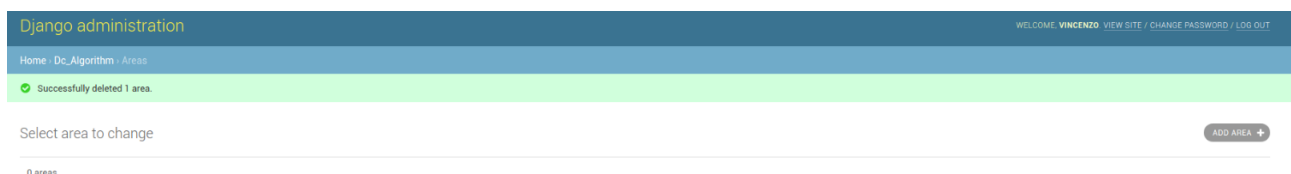
dc = data_access_api.DataAccessApi()

dc.get_datacube_metadata('ls7_ledaps_general', 'LANDSAT_7')
```

Registra le estensioni di latitudine e longitudine. Dovrebbero essere:

```
lat=(7.745543874267876, 9.617183768731897)
lon=(-3.5136704023069685, -1.4288602909212722)
```

Torna alla pagina di amministrazione, seleziona dc_algorithm-> Aree, elimina tutte le aree iniziali, quindi fai clic sul pulsante "Aggiungi area".



Per l'id di area, inserire general perché nel file di ingestione il campo output_type è stato definito come ls7_ledaps_general, se fosse stato ls8_lasrc_puglia il campo id dell'area sarebbe stato puglia invece il nome dell'area può essere qualsiasi combinazione alfanumerica.

Immettere i limiti di latitudine e longitudine in tutti i campi min / max di latitudine / longitudine per entrambi i campi superiore e di dettaglio.

Per tutti i campi delle immagini, inserisci /static/assets/images/black.png- questo darà un'anteprima dell'area nera, ma conterrà comunque i dati che specifichiamo.

Django administration

Home » Dc_Algorithm » Areas » Add area

Add area

Id:

Name:

Latitude min:

Latitude max:

Longitude min:

Longitude max:

Thumbnail imagery:

Satellites:

LANDSAT_7
LANDSAT_8
LANDSAT_5
LANDSAT_5, LANDSAT_7, LANDSAT_8
SENTINEL_1A
LANDSAT_7, LANDSAT_8

+

Hold down "Control", or "Command" on a Mac, to select more than one.

Seleziona LANDSAT_7 nel campo dei satelliti e salva la nuova area.
Torniamo alla pagina principale dell'amministratore e seleziona dc_algorithm-> Applicazioni.

Home » Dc_Algorithm

Dc_Algorithm administration

| DC_ALGORITHM | | |
|--------------------|-----------------------|------------------------|
| Application groups | + Add | Change |
| Applications | + Add | Change |
| Areas | + Add | Change |
| Compositors | + Add | Change |
| Satellites | + Add | Change |

Scegli custom_mosaic_tool e seleziona la tua area nel campo delle aree. Salva il modello ed esci.

Select application to change

Action: ----- Go 0 of 10 selected

| <input type="checkbox"/> | ID | NAME |
|--------------------------|--------------------|-------------------|
| <input type="checkbox"/> | water_detection | Water Detection |
| <input type="checkbox"/> | urbanization | Urbanization |
| <input type="checkbox"/> | tsm | Water Quality TSM |
| <input type="checkbox"/> | spectral_indices | Spectral Indices |
| <input type="checkbox"/> | slip | Slip |
| <input type="checkbox"/> | ndvi_anomaly | NDVI Anomaly |
| <input type="checkbox"/> | fractional_cover | Fractional Cover |
| <input type="checkbox"/> | custom_mosaic_tool | Custom Mosaic |
| <input type="checkbox"/> | coastal_change | Coastal Change |
| <input type="checkbox"/> | cloud_coverage | Cloud Coverage |

10 applications

Change application

Id:

custom_mosaic_tool

Name:

Custom Mosaic

Areas:

general

+

Hold down "Control", or "Command" on a Mac, to select more than one

Satellites:

LANDSAT_7
LANDSAT_8
LANDSAT_5
LANDSAT_5, LANDSAT_7, LANDSAT_8
SENTINEL_1A
LANDSAT_7, LANDSAT_8

+

Hold down "Control", or "Command" on a Mac, to select more than one

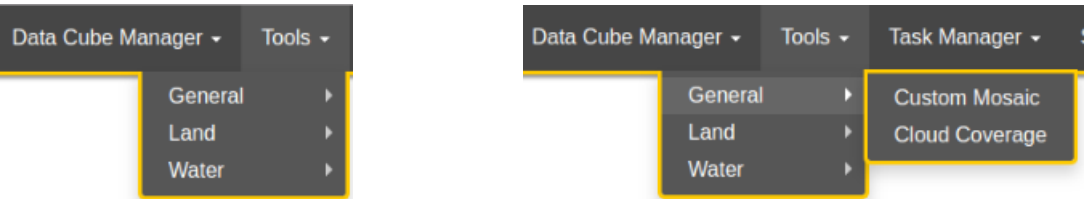
Color scale:

Application group:

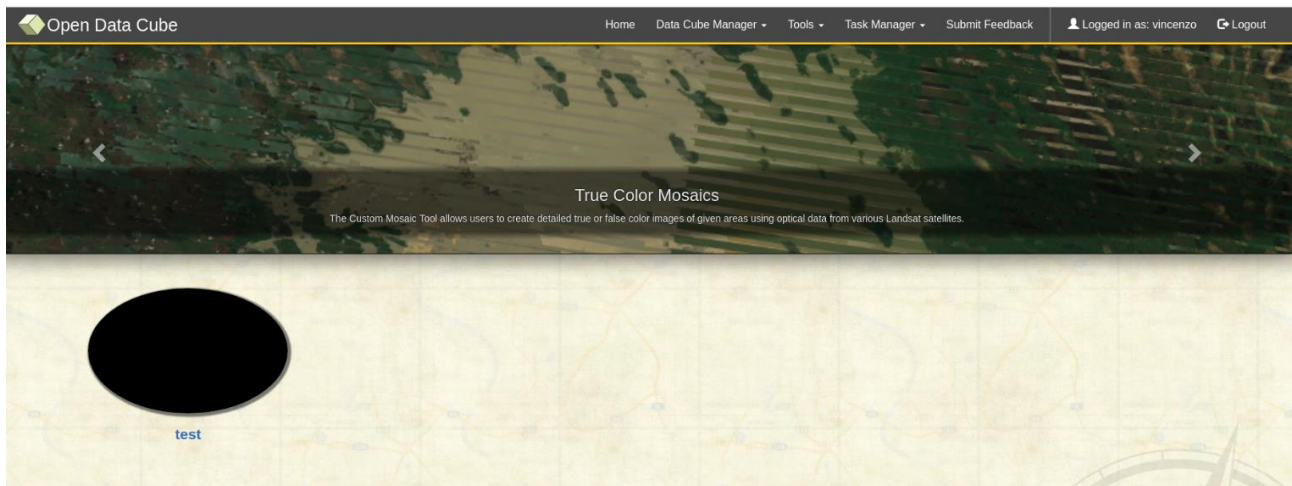
General

✎ +

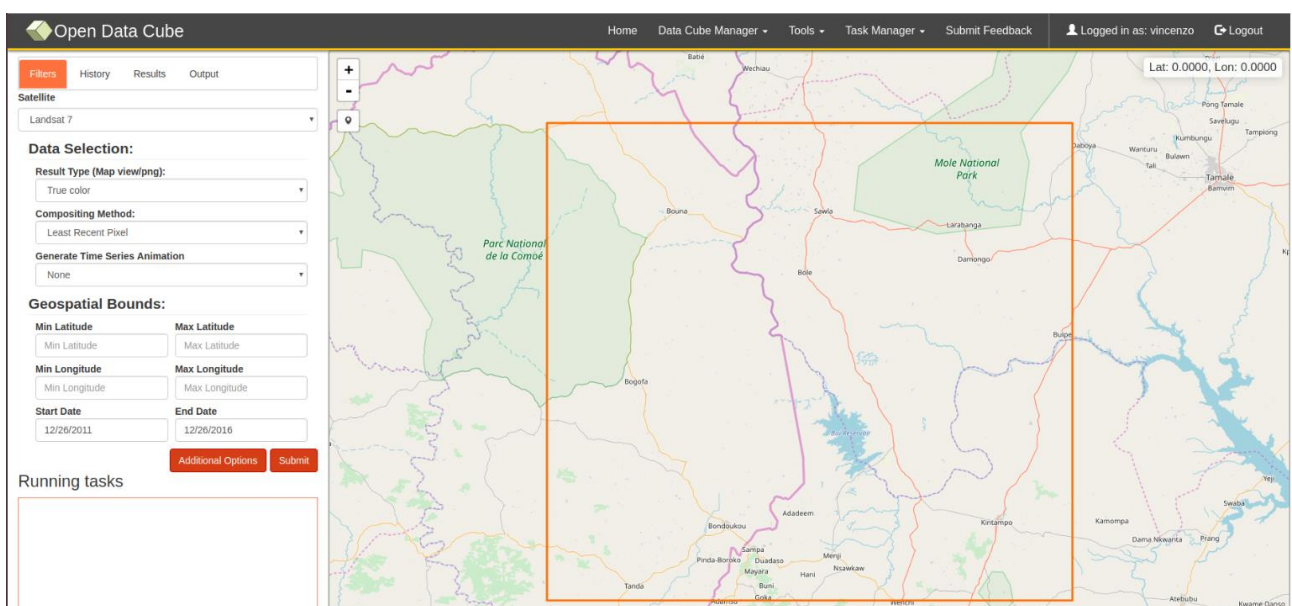
Torna al sito principale e torna allo Strumento mosaico personalizzato.



Vedrai che la tua area è l'unica nella lista

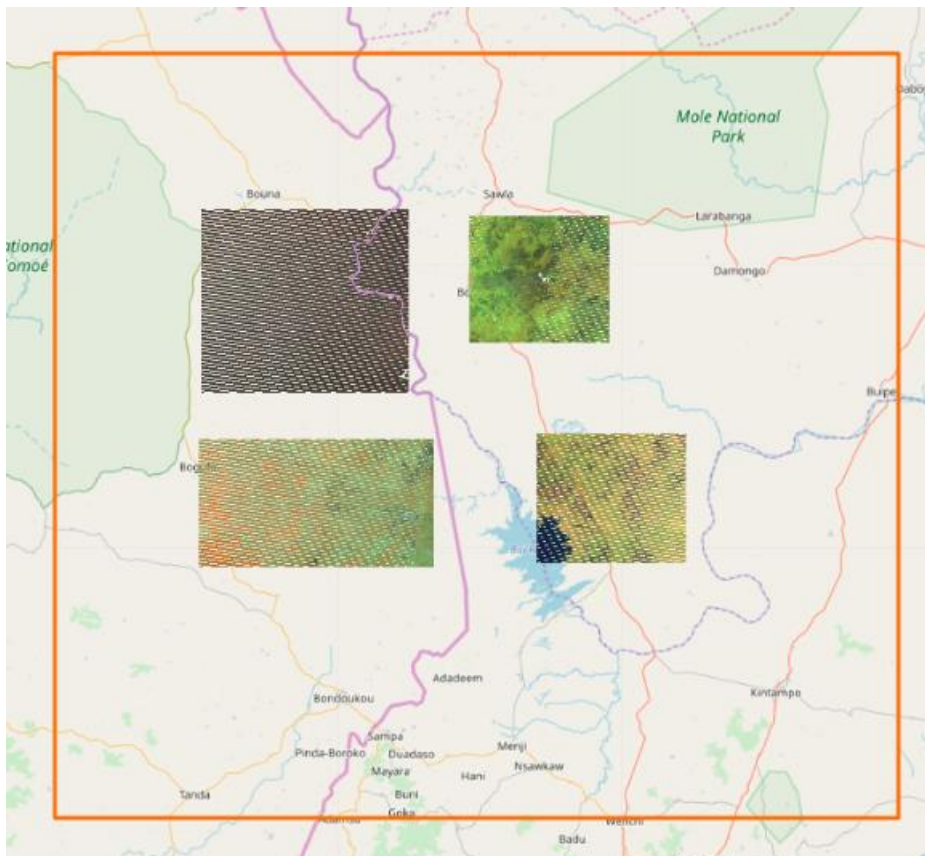


- seleziona quest'area per caricare lo strumento.



Assicurati che i tuoi comandi celery siano in esecuzione, seleziona una area all'interno della area precedente caricate, click su submit. La pagina web dovrebbe caricare un risultato dell'immagine.

Ecco alcuni esempi di elaborazione variando i valori dei campi Result Type, Compositing method.



Capitolo 6

Aggiunta nuovi algoritmi

L'interfaccia utente di Data Cube è progettata per essere estesa rapidamente e facilmente per integrare nuovi algoritmi che operano su set di dati raster basati su Data Cube. Una nuova implementazione dell'algoritmo include tre parti principali:

- Viste Django: associa gli URL alle pagine HTML renderizzate
- Modelli Django: mantiene gli attributi dell'attività principale (parametri, risultati, ecc.)
- Flussi di lavoro Celery - Elabora attività asincrone, riempiendo il modello Django con aggiornamenti e risultati

Sono forniti due comandi `manage.py` per velocizzare questa elaborazione, richiedendo solo modifiche semplici e definite per far funzionare un'applicazione.

6.1 Django Views

Ogni classe di visualizzazione è sottoclasse della classe base `dc_algorithm`, quindi è necessario apportare solo modifiche minori per una nuova pagina. Le funzioni e le variabili che devono essere fornite sono definite nella docstring della classe base e, se vengono omesse, viene sollevata un'eccezione. Il contenuto HTML che viene visualizzato nel browser è controllato qui: forms, panels, ecc...

6.2 Django Models

I modelli Django contengono tutte le informazioni e i parametri necessari per eseguire le operazioni e creare prodotti di output. Ciò include dove salvare i prodotti di output, come suddividere i dati per l'elaborazione e la funzione Python che dovrebbe essere utilizzata per creare i prodotti di output. Tutti i parametri di input e output dovrebbero essere elencati qui.

6.3 Celery tasks

Le attività di Celery eseguono tutte le elaborazioni per gli algoritmi. Le attività utilizzano le informazioni trovate sui modelli per separare i blocchi di dati in dimensioni gestibili, eseguire funzioni di analisi e creare prodotti di output.

Le attività di Celery aprono i modelli con il loro id e analizzano le informazioni richieste. Tutta l'elaborazione effettiva viene eseguita in `tasks.py` per l'app, dal recupero dei dati e dal chunking all'elaborazione dei prodotti di output. È in `tasks.py` dove verrà implementato l'algoritmo attuale.

La pipeline di elaborazione in `tasks.py` è organizzata in modo che ogni attività operi indipendentemente l'una dall'altra: il processo è completamente asincrono e non

bloccante. Le attività sono inoltre organizzate in modo da ridurre la pagina mentre l'attività viene eseguita e ogni attività completa una singola funzione.

Le singole funzioni sono:

- I parametri vengono analizzati dal modello di attività;
- I parametri sono verificati e convalidati;
- I parametri sono suddivisi in pezzi più piccoli e più gestibili per l'elaborazione parallela;
- Una pipeline di elaborazione viene creata dai blocchi di parametri e inoltrata per l'elaborazione. Ciò riguarda sia i blocchi geografici che quelli basati sul tempo;
- Ogni pezzo viene elaborato, con i risultati salvati su disco. I metadati vengono raccolti qui e inoltrati;
- I pezzi sono combinati sia geograficamente che nel tempo, combinando anche i metadati;
- I prodotti di output sono generati e il modello è aggiornato.

6.4 Classi Base

I modelli e le viste di Django consentono l'ereditarietà standard di Python, permettendoci di creare semplicemente una sottoclasse di base comune di `dc_algorithm` per creare rapidamente nuove app.

Di seguito è riportata la classe di base per l'invio di attività. Si noti le ampie docstring che descrivono tutti gli attributi e i parametri richiesti. Inoltre, ci sono funzioni 'getter' che generano `NotImplementedError` quando non è presente un attributo richiesto.

```
class SubmitNewRequest(View, ToolClass):
    """Submit a new request for processing using a task created with form data

    REST API Endpoint for submitting a new request for processing. This is a POST only
    view,
    so only the post function is defined. Form data is used to create a Task model
    which is
    then submitted for processing via celery.

    Abstract properties and methods are used to define the required attributes for an
    implementation.
    Inheriting SubmitNewRequest without defining the required abstracted elements will
    throw an error.
    Due to some complications with django and ABC, NotImplementedError are manually
    raised.

    Required Attributes:
        tool_name: Descriptive string name for the tool - used to identify the tool in
        the database.
        celery_task_func: A celery task called with .delay() with the only parameter
        being the pk of a task model
        task_model_name: Name of the model that represents your task - see models.Task
        for more information
        form_list: list [] of form classes (e.g. AdditionalOptionsForm, GeospatialForm)
        to be used to validate all provided input.

    """
```



```

celery_task_func = None
form_list = None

@method_decorator(login_required)
def post(self, request):
    """Generate a task object and start a celery task using POST form data

    Decorated as login_required so the username is fetched without checking.
    A full form set is submitted in one POST request including all of the forms
    associated with a satellite. This formset is generated using the
    ToolView.generate_form_dict function and should be the forms for a single
satellite.
    using the form_list, each form is validated and any errors are returned.

    Args:
        POST data including a full form set described above

    Returns:
        JsonResponse containing:
            A 'status' with either OK or ERROR
            A Json representation of the task object created from form data.
    """

    user_id = request.user.id

    response = {'status': "OK"}
    task_model = self._get_tool_model(self._get_task_model_name())
    forms = [form(request.POST) for form in self._get_form_list()]
    #validate all forms, print any/all errors
    parameter_set = {}
    for form in forms:
        if form.is_valid():
            parameter_set.update(form.cleaned_data)
        else:
            for error in form.errors:
                return JsonResponse({'status': "ERROR", 'message':
form.errors[error][0]})

        task, new_task = task_model.get_or_create_query_from_post(parameter_set)
        #associate task w/ history
        history_model, __ =
self._get_tool_model('userhistory').objects.get_or_create(user_id=user_id,
task_id=task.pk)
        if new_task:
            self._get_celery_task_func().delay(task.pk)
            response.update(model_to_dict(task))

    return JsonResponse(response)

def _get_celery_task_func(self):
    """Gets the celery task function and raises an error if it is not defined.

    Checks if celery_task_func property is None, otherwise return the function.
    The celery_task_func must be a function callable with .delay() with the only
    parameters being the pk of a task model.

    """
    if self.celery_task_func is None:
        raise NotImplementedError(
            "You must specify a celery_task_func in classes that inherit
SubmitNewRequest. See the SubmitNewRequest docstring for more details."

```

```

    )
    return self.celery_task_func

    def _get_form_list(self):
        """Gets the list of forms used to validate post data and raises an error if it
        is not defined.

        Checks if form_list property is None, otherwise return the function.
        The celery_task_func must be a function callable with .delay() with the only
        parameters being the pk of a task model.

        """
        if self.form_list is None:
            raise NotImplementedError(
                "You must specify a form_list in classes that inherit SubmitNewRequest.
                See the SubmitNewRequest docstring for more details."
            )
        return self.form_list

```

Le classi base del modello funzionano nello stesso modo: gli attributi comuni sono definiti e gli utenti sono liberi di aggiungere o rimuovere campi nelle classi child.

6.5 Generazione di file di base

Sono disponibili due comandi `manage.py` per creare facilmente applicazioni: una per semplici algoritmi (matematica della banda, qualsiasi applicazione che dovrebbe essere eseguita su un mosaico) e una base più flessibile utilizzata per concetti più complessi.

```

python manage.py start_bandmath_app app_name
python manage.py start_dc_algorithm_app app_name

```

Questi comandi fanno alcune cose:

- Copia i file di base associati in `app / nome_app`
- Rinominare tutti i valori di modello per `nome_app` in tutti i file, modelli, viste, moduli, attività, ecc. Di Python.
- Crea un modello `dc_algorithm.models.Application` per la tua nuova app
- Fornisce istruzioni sui passaggi successivi per far funzionare la tua applicazione

Le istruzioni includono per eseguire le migrazioni e inizializzare il database con i nuovi modelli. Questi file di base contengono alcune `TODO`: istruzioni all'interno dei file, che indicano dove sono richiesti gli input.

Le app sono disponibili in due classi principali: app per la matematica e applicazioni più complesse. Le app per la matematica della banda eseguono il compositing sul tempo e sull'area geografica selezionati, quindi eseguono alcune funzioni sul mosaico risultante. L'app `dc_algorithm` è configurata per essere più generale e richiederà più input. Come regola generale, se un algoritmo non ha un componente di serie temporali, è possibile utilizzare l'app per la matematica della banda.

6.6 Applicazioni di base

Per un'applicazione di base, sono necessarie poche modifiche per avere un'app funzionale. Dopo aver generato i file di base con il comando

```
python manage.py start_bandmath mytools
```

comparirà la seguente schermata:

```
Creating app NdvTest
Copied files...
Renamed constants...
Initialized base models...
Done! Follow the instructions below to finish your new apps creation.
Add the string: url(r'^ndvi_test/', include('apps.ndvi_test.urls')) to data_cube_ui/urls.py
Add the string: 'apps.ndvi_test', to data_cube_ui/settings.py in the INSTALLED_APPS list.
Create a GDAL Dem compatible color scale in utils/color_scales/ named ndvi_test
A default red-> green color scale will be used until you create your color scale. Change the path to the color scale in your new models.py - Query
Run 'makemigrations' and 'migrate'
Run 'python manage.py loaddata apps/ndvi_test/fixtures/init.json'
```

Questo output ci informa che per poter ultimare l'aggiunta del nuovo algoritmo dobbiamo svolgere le seguenti azioni.

- aggiungere la stringa `url(r'^mytools/', include('apps.mytools.urls'))` nel file `data_cube_ui/urly.py`;
- aggiungere la stringa `'apps.mytools'` nel file `data_cube_ui/settings.py` nella lista `INSTALLED_APPS`;
- creare un file GDAL compatible color scale in `utils/color_scale` chiamato `mytools`, nel caso in cui non c'è questo file prenderà le impostazioni di colore di default;
- aprire il file `apps/mytools/fixtures/init.json` e modificare `app_name.toolinfo` in `mytools.toolinfo`;
- dopo di che eseguire i seguenti comandi (sempre attivando l'ambiente virtuale di python installato precedentemente):
 - `python manage.py makemigrations mytools`;
 - `python manage.py migrate mytools`;
 - `python manage.py loaddata apps/mytools/fixtures/init.json`.

NOTA:

Se dopo il comando `'python manage.py makemigrations mytools'` comparirà questo genere di output:

```
You are trying to add a non-nullable field 'query_type' to ndvtask without a default; we can't do that (the database needs something to populate existing rows).
Please select a fix:
 1) Provide a one-off default now (will be set on all existing rows with a null value for this column)
 2) Quit, and let me add a default in models.py
Select an option: 1
Please enter the default value now, as valid Python
The datetime and django.utils.timezone modules are available, so you can do e.g. timezone.now
Type 'exit' to exit this prompt
```

Scegliere l'opzione 1 ed inserire il valore `timezone.now` e continuare come i comandi successivi.

Ora avremo tutte le classi di base generate nel database e siamo pronti per implementare l'algoritmo. Aprire il file `tasks.py` e inserire l'espressione matematica della banda nella sezione indicata dalla parola `TODO`.

tasks.py

```
def _apply_band_math(dataset):
    #TODO: apply your band math here!
    return (dataset.nir - dataset.red) / (dataset.nir + dataset.red)

dataset = xr.open_dataset(chunk[0])
dataset['band_math'] = _apply_band_math(dataset)
```

Di default viene impostata la formula per il calcolo della NDVI.

Ora che tutto è completo, per avviare la nuova applicazione bisogna:

- Riavviare di Apache2;
- Riavviare i workers di celery;
- Visitare il sito e selezionare l'app appena aggiunta.

Nota: se dopo aver aggiunto la nuova funzione dovesse uscire il messaggio 500 server error, cambiare il valore DEBUG da False a True in setting.py dopo di che avviare il comando `python manage.py runserver 0.0.0.0:8000` e correggere eventuali errori a livello di codice.

Nel nostro caso abbiamo dovuto cambiare la seguente sezione di codice nel file views.py presente nella cartella dell'app appena aggiunta:

`from datetime import datetime, timedelta` → `import datetime`

inserito la seguente libreria -> `from apps.dc_algorithm.forms import MAX_NUM_YEARS`

da

Class NdviProva(ToolView):

"""Creates the main view for the custom mosaic tool by extending the ToolView class

Extends the ToolView abstract class - required attributes are the tool_name and the

generate_form_dict function.

See the dc_algorithm.views docstring for more details.

"""

tool_name = 'ndvi_prova'

task_model_name = 'NdviProvaTask'

allow_pixel_drilling = True

def generate_form_dict(self, satellites, area):

```

forms = {}

for satellite in satellites:

    forms[satellite.pk] = {

        'Data Selection':

            AdditionalOptionsForm(

                datacube_platform=satellite.datacube_platform, auto_id="{}_{}".format(satellite.pk)),

        'Geospatial Bounds':

            DataSelectionForm(area=area,

                                time_start=satellite.date_min,

                                time_end=satellite.date_max,

                                auto_id="{}_{}".format(satellite.pk))

    }

return forms

```

a

```
class NvdiProva(ToolView):
```

```
    """Creates the main view for the custom mosaic tool by extending the ToolView class
```

```

    Extends the ToolView abstract class - required attributes are the tool_name and the
    generate_form_dict function.

```

```
    See the dc_algorithm.views docstring for more details.
```

```
    """
```

```
    tool_name = 'nvdi_prova'
```

```
    task_model_name = 'NvdiProvaTask'
```

```
    allow_pixel_drilling = True
```

```
    #Questo la cambiamo
```

```
    def generate_form_dict(self, satellites, area, user_id, user_history, task_model_class):
```

```
        forms = {}
```

```
        for satellite in satellites:
```

```
            time_end = satellite.date_max
```

```
earliest_allowed_time = datetime.date(time_end.year - MAX_NUM_YEARS, time_end.month, time_end.day)
```

```
time_start = max(satellite.date_min, earliest_allowed_time)
```

```
forms[satellite.pk] = {
```

```
    'Data Selection':
```

```
        AdditionalOptionsForm(
```

```
            datacube_platform=satellite.datacube_platform, auto_id="{}_{}".format(satellite.pk)),
```

```
        'Geospatial Bounds':
```

```
            DataSelectionForm(user_id=user_id,
```

```
                                user_history=user_history,
```

```
                                task_model_class=task_model_class,
```

```
                                area=area,
```

```
                                time_start=time_start,
```

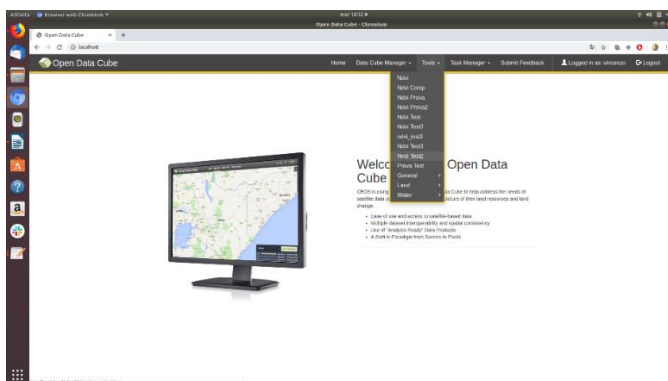
```
                                time_end=time_end,
```

```
                                auto_id="{}_{}".format(satellite.pk))
```

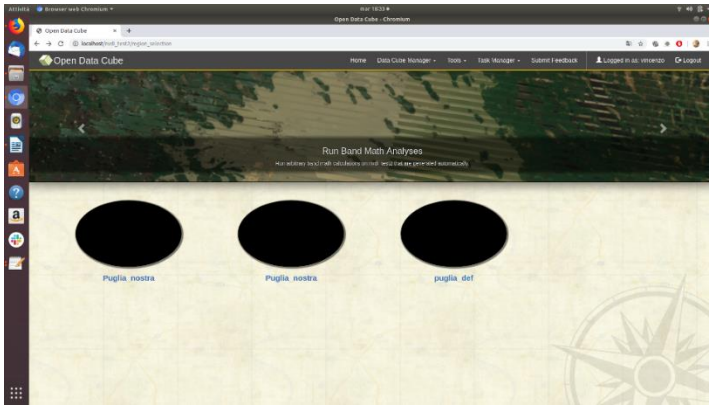
```
    }
```

```
return forms
```

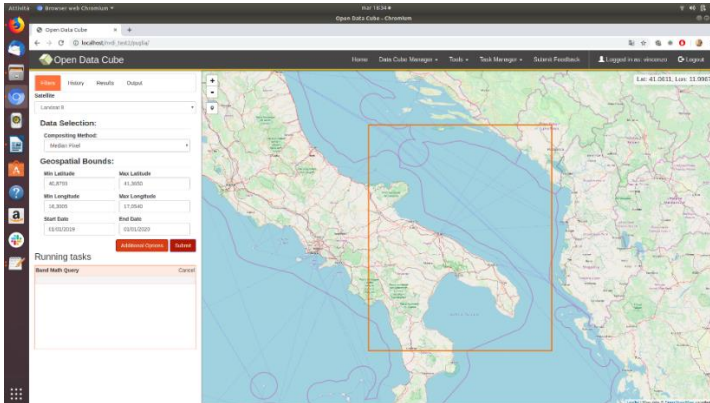
6.7 Esempio



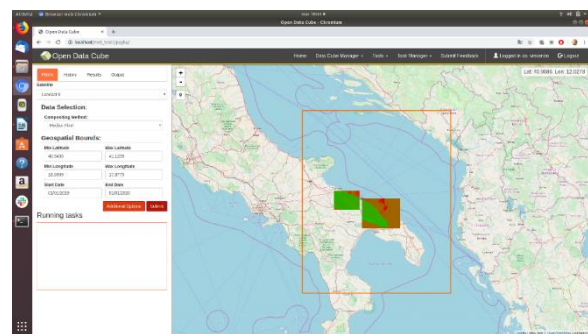
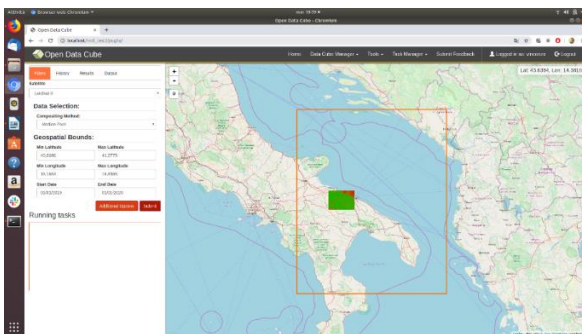
Nel nostro esempio abbiamo cercato di inserire una nuova app dal nome `nvdi_test2` che calcola l'indice NDVI



Abbiamo aggiunto tramite la pagina amministrativa localhost/admin l'area della puglia specificando come id "puglia", come nome puglia_def e aggiungendo questa nuova area nella nuova applicazione con lo stesso procedimento descritto nella sezione 5.6.



Quella che vediamo è l'area della puglia che abbiamo appena aggiunto alla nostra nuova app, ora selezioniamo l'area di cui vogliamo calcolare l'indice, l'intervallo temporaneo e il compositing method e clicchiamo su submit.



Qui vediamo degli esempi di elaborazione sulle varie aree selezionate.

6.8 Applicazioni complesse

Il processo per l'implementazione di un'applicazione avanzata è simile all'app per la matematica della banda. Generare l'applicazione di base utilizzando il comando `manage.py`, ma non eseguire ancora le migrazioni, bisogna aprire tutti i file generati e inserire nelle zone in cui c'è il TODO tutte le informazioni e parametri di cui abbiamo bisogno per il corretto funzionamento della nostra nuova app.

RIFERIMENTI

Per maggiore informazione visitare i seguenti siti internet:

1. <https://www.opendatacube.org/>
2. <https://github.com/ceos-seo>
3. <https://gis.stackexchange.com/questions/tagged/open-data-cube>
4. <http://slack.opendatacube.org/>
5. info@opendatacube.org