

GEOOs – GeoServer

Introducción

GeoServer es el componente de GEOOs encargado de proveer los datos para las capas con variables tipo “Raster” y “Vectoriales”. Las variables tipo raster corresponden a aquellas cuyo valor está asociado a una matriz regular de coordenadas; es decir, cada valor puede representarse como un vector de al menos 3 coordenadas (longitud, latitud, valor-variable). Si se agrega la dimensión tiempo o algún clasificador como profundidad, la cantidad de coordenadas podrían variar.

Las capas vectoriales corresponden a información de objetos vectoriales (polígonos, puntos, líneas) que pueden contener atributos asociados. Los objetos de capas vectoriales pueden además representar un valor de una dimensión de una variable de ZRepo (ver documentación del componente ZRepo de GEOOs). Esto significa que la selección de un objeto en el mapa permite filtrar los resultados de una consulta a cubos de información dentro del modelo de DataWarehousing (cubo de nieve) de ZRepo.

Tanto la información raster como la Vectorial puede ser temporal o a-temporal. La información temporal debe clasificarse dentro de un período, el que podría ser alguno de los siguientes valores: 5 minutos, 15 minutos, 30 minutos, 1 hora, 6 horas, 12 horas, 1 día, 1 mes, 3 meses, 4 meses, 6 meses ó 1 año.

GeoServer ofrece sus servicios al Portal GEOOs mediante un API Rest, lo que significa que puede ser consultado desde cualquier sistema que soporte este estándar.

En este documento se explica cómo configurar una instalación de GeoServer para agregar nuevas capas Raster y Vectoriales. Se muestran ejemplos de consumo de los datos mediante su API Rest y desde el Portal GEOOs. Se recomienda utilizar el Portal como visualizador de los datos, incluso para las etapas de pruebas, ya que permite validar en forma rápida (visualmente) que los datos están correctos.

Instalación del Ambiente

GeoServer puede instalarse y ejecutarse como un microservicio sin dependencias, usando su imagen docker; sin embargo, como se mencionó antes, se recomienda una instalación del Portal y Geoserver juntos, de tal forma de poder visualizar los datos que se están sirviendo.

El documento “**geos-componentes-instalacion**” describe la forma en que se configura un ambiente de producción (a diferencia del ambiente de desarrollo) que sólo contiene al Portal GEOOs, su dependencia Mongo DB y la forma en que se debe configurar, todo usando docker swarm como orquestador de contenedores, bajo el directorio **/opt/geos** (en Linux).

Usando esa instalación como base, se modificará el archivo descriptor de docker (YML) y los directorios de configuración, para crear inicialmente una instalación vacía de geoserver.

Bajo el directorio **/opt/geos/config** agregamos un archivo “**config.hjson**” con la configuración del nuevo GeoServer que instalaremos. El contenido de este archivo es:

```
#  
# GEOOS Sample config file for importing and publishing geospatial data
```

```

#
# Directories '/home/config' (where this file resides), '/home/www' and '/home/data' are
# mapped to your host machine's file system, as volumes in docker contaier. If you run more
# than one service, as data downloaders (noaa-gfs4, for example), consider creating a
docker
# swarm stack. See documentaton at https://github.com/geoos/swarm-examples
#
{
  ## This Server Description
  thisGeoServer:{
    code: geoos-test
    name: Geo-Server de Prueba
    publicURL: http://localhost:8080
    enableTestWebServer: false
  }

  ## Web server to publish REST Services and Web Content to GEOOS Server, Browsers and
other client applications
  webServer:{
    protocol:"http", port 8080
    # keyFile and certFile are required for https. Locations are relative to /home/config
    # keyFile: "certs/my_cert.key"
    # certFile: "certs/my_cert.crt"
  }

  ## Metadata
  ## Providers (data producers or owners)
  providers:{
  }

  ## DataSets
  ## For each dataSet defined here, there should be a config file at /home/config/${dataSet-
code}.hjson
  dataSets:{
  }

  ## Scales
  colorScales:[]
}

```

El formato hjson corresponde a un “JSON de consumo humano”. Representa y se interpreta como un archivo de tipo JSON con menos restricciones. Por ejemplo, los saltos de línea pueden reemplazar y usarse como serparador en lugar de las comas. Los nombres de atributos no necesitan escribirse entre comillas y además es posible agregar comentarios entre los valores y líneas del archivo.

Creamos además un subdirectorio para datos **/opt/geoos/data** y uno para publicación de recursos web: **/opt/geoos/www**, ambos inicialmente vacíos.

A continuación, agregamos el servicio GeoServer a la lista de microservicios que se ejecutan dentro del stack de docker swarm de geoos. Modificamos el archivo **geoos.yml** creado en el documento “**geoos-componentes-instalacion**” para que incluya el servicio “geoserver”:

```
version: '3.6'
services:
  db:
    image: mongo:4.4.1-bionic
    environment:
      MONGO_INITDB_ROOT_USERNAME: admin-user
      MONGO_INITDB_ROOT_PASSWORD: admin-password
    volumes:
      - ./mongo-data:/data/db

  portal:
    image: docker.homejota.net/geoos/portal
    ports:
      - 8091:8090/tcp
    environment:
      MONGO_URL: "mongodb://admin-user:admin-password@db:27017"
      MONGO_DATABASE: geoos
    volumes:
      - ./config:/home/config

  geoserver:
    image: docker.homejota.net/geoos/geoserver
    ports:
      - 8180:8080/tcp
    environment:
      LOG_LEVEL: debug
    volumes:
      - ./data:/home/data
      - ./config:/home/config
      - ./log:/home/log
      - ./www:/home/www
```

Como se observa en la nueva versión del descriptor del stack de servicios, se expone por el puerto 8180 de la máquina host el puerto web del geoserver.

Al iniciar o actualizar el stack de servicios de docker (comando “**docker stack deploy -c geoos.yml geoos**” desde **/opt/geoos**) se puede revisar el estado del servicio usando “**docker service ls**” (dependiendo del poder de la máquina, puede demorar varios minutos). Una vez que el servicio levanta (1/1 instancias en ejecución) éste debe haber creado un archivo de log llamado “**geoserver-**” y terminado con la fecha del día. Ese archivo contiene los detalles de la ejecución del servicio. Se puede revisar la correcta inicialización del servidor en ese archivo. Al iniciarse el servidor web que responde a los servicios del API Rest, el archivo debería indicar:

[info] http web server started at port 8080

La configuración del GeoServer inicialmente está vacía, sin proveedores de datos, sin Datasets y sin escalas de colores. Geoserver provee de un servicio de consulta de metadata (configuración y datos disponibles) de tipo GET, por lo que puede ser consultado directamente desde un browser. Para probar la correcta instalación del GeoServer, apuntamos un browser a la dirección IP de la máquina en donde se instaló geos, con la URL (ejemplo):

<http://192.168.0.19:8180/metadata>

El resultado debería ser un objeto JSON de la forma:

```
{"code":"geos-test","name":"Geo-Server de Prueba","providers":[],"dataSets":[],"colorScales":[]}
```

Bajo el directorio **/opt/geos/data** se deberían haber creado subdirectorios llamados: **“discarded”**, **“files-with-errors”**, **“finished”**, **“import”**, **“tmp”** y **“working”**. Estos directorios son usados por el servidor para procesar y almacenar los archivos de datos correspondientes a cada Dataset.

Las capas con datos se proveen desde GeoServer dentro de “Datasets”. Un Dataset corresponde a un conjunto de datos que se obtienen con la misma periodicidad desde el mismo proveedor de datos. Para publicar una capa de datos, debemos entonces comenzar por definir en el archivo de configuración de GeoServer (**/opt/geos/config/config.hjson**) al nuevo proveedor. En este caso, usaremos un archivo de información vectorial de Aeropuertos de Chile, provisto por la Biblioteca del Congreso Nacional.

Un registro de proveedor de datos para GeoServer incluye su nombre, una URL de página Home y una imagen (logo) de la Institución. Las imágenes se almacenan bajo el directorio **“www”** creado antes, desde donde se publican mediante un servidor web propio al Portal GEOOs o cualquier otra aplicación cliente.

La imagen usada en este ejemplo como logo de la BCN puede descargarse desde:

<https://github.com/geos/geoserver-demo/blob/main/img/bcn.png>

Descargamos esta imagen en el directorio (se debe crear el subdirectorio **“img”** bajo **/opt/geos/www**)

Se modifica el archivo **config/config.hjson** para agregar el proveedor **“bcn”**. La sección **“providers”** de este archivo debe quedar:

```
## Providers (data producers or owners)
providers:{
  bcn:{
    name: "Biblioteca del Congreso Nacional", url:"https://www.bcn.cl"
    # Imagen relativa a /opt/geos/www
    logo:"img/bcn.png"
  }
}
```

Al guardarse el archivo **config.hjson**, éste es automáticamente recargado por el servicio GeoServer, por lo que no es necesario reiniciarlo. Esto puede verificarse en el archivo del **log** correspondiente al día (subdirectorio **/opt/geos/log**, en nuestro ejemplo).

```
15:02:15 [main] updating config from /home/geoserver/config/config.hjson
```

La actualización de la metadata y la imagen publicada puede verificarse apuntando el browser a la IP del servidor.

DataSet Vectoriales

Un Dataset representa un conjunto de capas de datos del mismo tipo (vectorial o raster), asociadas al mismo proveedor, y con la misma temporalidad. Los Datasets vectoriales se importan desde archivos geoJSON y representan objetos como puntos, líneas o polígonos, cada uno de ellos con propiedades.

En el archivo de configuración del GeoServer se debe indicar el código de cada Dataset, su nombre y el código del proveedor.

Se debe agregar el Dataset de prueba a la sección “Datasets” del archivo de configuración “**config.hjson**”.

```
## DataSets
## For each dataSet defined here, there should be a config file at /home/config/${dataSet-code}.hjson
dataSets:{
  bcn-test: {name: "Prueba BCN", provider:"bcn"}
}
```

Si se revisa el archivo **log** luego de guardar los cambios, se obtiene un error, ya que por cada Dataset debe crearse un nuevo archivo de configuración, con el código del Dataset y de tipo hjson.

```
18:31:48 [info] Stopping web server ...
18:31:48 [info] Web Server stopped
```

Bajo el directorio **/opt/geoos/config**, creamos el archivo **bcn-test.hjson**, el que inicialmente tendrá el siguiente contenido:

```
{
  dataSet: {type:"vector", format:"geoJSON"}
  temporality:"none"
  maxFilesInCache: 30
  maxTiledFilesInCache: 30

  filesDefault:{
    Cache: true
    searchTolerance: 0
    options:{
      subjects:["interes"]
    }
  }
}

files: {
}
}
```

La versión actual de GeoServer sólo soporta archivos vectoriales de tipo geoJSON.

Un Dataset de tipo Vectorial define una serie de “**files**”, cada uno representando a una capa de objetos vectoriales. La sección “**filesDefault**” permite definir atributos comunes a todos los archivos que se definan dentro del Dataset. El atributo “**searchTolerance**” permite indicar la cantidad de períodos (de acuerdo a la temporalidad) que se buscarán datos hacia adelante y atrás de la fecha / hora de una consulta, antes de retornar un error cuando no encuentre datos.

Los datos para las capas que GeoServer provee se suministran al servidor como archivos. En el caso de los Datasets vectoriales, estos archivos son geoJSON. Para este ejemplo usaremos un archivo descargado desde la BCN (Biblioteca del Congreso Nacional) y convertido a geoJSON (usando herramientas de GDAL) con datos de los aeropuertos de Chile. Este archivo puede ser descargado desde:

<https://github.com/geoos/geoserver-demo/blob/main/geojson/aeropuertos.geojson>

Dentro del Dataset “**bcn-test**”, se configuran diversos archivos. Cada archivo se obtiene desde una fuente geoJSON. Modificamos la sección “**files**” del archivo **bcn-test.hjson** para agregar la definición del archivo de aeropuertos.

```
files: {
  aeropuertos:{
    cache:true, tiledCache:true, commonName:"Aeropuertos de Chile"
    metadata:{
      idProperty:"objectid_1", nameProperty:"Aerodromo", centroid:true
      copyProperties:{
        codigoComuna:"cod_comuna", categoria:"categoría"
        codigoOACI:"cod_oaci", codigolATA:"cod_iata"
        nombreRegion:"Region", nombreProvincia:"Provincia", nombreComuna:"Comuna:"
      }
    }
    options:{
      getFeatureStyle:{stroke:"black", strokeWidth:1.4, radius:6, fill: "gray"}
      getSelectedFeatureStyle:{stroke:"black", strokeWidth:2.0, fill:"rgba(50,50,250,0.4)", radius:7}
    }
  }
}
```

Se define un nuevo “**file**” con código “**aeropuertos**”. Se indica que el archivo se almacene en cache con tiles (se optimiza por áreas que se retornan al browser por separado, sólo aquellas que se consultan de acuerdo al área visible del mapa)

Se debe indicar que se desea generar información del centroide de cada objeto. Este punto lo usará el Portal GEOOs para mostrar una etiqueta con su nombre y los resultados de las variables observadas asociadas a cada punto.

En primer lugar, debemos analizar el contenido del archivo geoJSON original que deseamos importar para determinar aquellas propiedades o atributos que son de interés y que nos interesa también importar.

Abrimos el archivo original **aeropuertos.geoJSON** en un editor de texto:

Un archivo geoJSON se compone de una lista (arreglo JSON) de objetos “**feature**”. Estos objetos tienen un tipo, una geometría y una serie de propiedades. GeoServer nos permite copiar las propiedades de interés desde los archivos importados. Copiando el contenido del archivo geoJSON en el sitio <https://jsonlint.com/> (o alguna otra herramienta que formatee el texto como JSON) obtenemos:

Cada objeto dentro de un archivo vectorial de GEOOs debe tener una propiedad **"id"** que lo identifique únicamente dentro del archivo. Si el archivo original posee un identificador único, es posible utilizarlo indicando la propiedad **"idProperty"** con el nombre del atributo desde donde debe copiarse esta propiedad.

Debemos indicar además el nombre de la propiedad que usaremos como nombre desplegable de cada uno de los objetos. En este caso, la propiedad **"Aerodromo"** contiene el nombre del aeródromo o aeropuerto que queremos desplegar.

La sección **"copyProperties"** nos permite importar las propiedades deseadas desde el archivo original, cambiando el nombre. El formato de cada una es **"nuevoNombre":"nombreOriginal"**

Los objetos vectoriales tipo **"Point"** son dibujados por el Portal GEOOs como círculos en el mapa. Es posible indicar las propiedades gráficas de estos círculos en estado normal y seleccionado. Estas propiedades son un objeto JSON de acuerdo al **"estilo"** que espera para un círculo, la biblioteca Konva JS (<https://konvajs.org/docs/shapes/Circle.html>) que se usa en las capas vectoriales del portal GEOOs. En este caso, usamos los siguientes estilos:

```
options:{
  getFeatureStyle:{stroke:"black", strokeWidth:1.4, radius:6, fill: "gray"}
  getSelectedFeatureStyle:{stroke:"black", strokeWidth:2.0, fill:"rgba(50,50,250,0.4)", radius:7}
}
```

Una vez que se ha modificado la configuración del archivo **"bcn-test.hjson"** indicando el nuevo **"file"** que se desea manejar, es posible importar el archivo original.

El GeoServer espera que los archivos que se importen cumplan con la siguiente regla para su nombre: **"nombreDataset_nombreFile"**. Si los archivos son temporales, se debe además agregar la fecha, como se indica más adelante.

En nuestro caso, renombramos el archivo original **"aeropuertos.geojson"** como **"bcn-test_aeropuertos.geojson"** y lo copiamos al directorio **geos/data/import**

El resultado de la importación del archivo se debería ver en el archivo de log de GeoServer (**geos/log/geoserver-fecha.log**)

```
11:49:55 [info] File bcn-test_aeropuertos.geojson imported
```

El archivo importado se borra desde la carpeta **"import"** y se crea un nuevo archivo en **geos/data/bcn-test**, también de tipo geoJSON, pero con las propiedades copiada según la configuración que indicamos.

Además, en el archivo **portal.hjson** dentro de la carpeta **geos/config** se debe modificar para incluir la redirección a tu servidor local.

```
# Servers
geoServers:["https://geoserver.geos.org", "http://localhost:8180"]
```

El archivo importado puede ahora ser visualizado como una capa vectorial en el Portal GEOOs.

Las propiedades de los objetos vectoriales pueden utilizarse para modificar dinámicamente el estilo con que ellos se despliegan en el mapa. La definición de los estilos en la configuración de un **"file"**

dentro de un Dataset vectorial, puede ser un objeto JSON (como el caso recién visto) o una función JavaScript que recibe como argumento el objeto vectorial que se despliega.

Podemos definir una función arrow (\Rightarrow) de JavaScript dentro de un atributo multilínea (triple comilla simple en el hJSON, `'''()'''`) que, dependiendo de la categoría del aeropuerto, modifique el color de relleno (**fill**) de cada círculo.

El objeto recibido contiene un atributo llamado **"tags"**, el que es un objeto JSON en donde cada atributo corresponde a una de las propiedades importadas. En nuestro ejemplo, usaremos la propiedad **"categoria"**, la que contiene los valores **"Aeropuerto"** o **"Aerodromo"**. Mostraremos de color rojo aquellos puntos que correspondan a **"Aeropuerto"**. Para ello, modificamos el atributo **"getFeatureStyle"** para que contenga el cuerpo de la función de selección:

```
options:{
  getFeatureStyle:'''(
    f=> {
      if (f.tags.categoria == "Aeropuerto")
        return {stroke:"black", strokeWidth:1.4, radius:6, fill:"red"};
      else
        return {stroke:"black", strokeWidth:1.4, radius:6, fill:"gray"};
    }
  )'''
  getSelectedFeatureStyle:{stroke:"black", strokeWidth:2.0, fill:"rgba(50,50,250,0.4)", radius:7}
}
```

Como se observa en la modificación, se retornará el mismo objeto, pero con su propiedad **"fill"** con valor **"red"** para los **"features"** cuya propiedad (**tag**) llamada **"categoria"** contenga el valor **"Aeropuerto"**. Luego de grabar la modificación, la configuración se actualiza en unos pocos segundos y se puede recargar la página. El proceso de actualización y posibles errores, puede monitorearse en el archivo de log de GeoServer.



El atributo **"centroid:true"** en la configuración del archivo le indica a GeoServer que genere la información del centroide de los objetos a medida que los importa. En el caso de los puntos (como los aeropuertos) el centroide es el mismo punto. Este valor es usado en el portal para mostrar la información del nombre al pasar el cursor sobre el objeto y para representar los valores de las variables (leyendas). Por ejemplo, podemos graficar la altitud (metros sobre el nivel del mar) a la que se encuentra cada aeropuerto, usando la opción **"Observar variables"** al seleccionar la capa en **"Mi Panel"** y seleccionar la variable **"Bathymetry"**.



Luego de unos segundos, se despliegan los valores de la altitud del aeropuerto. Es posible también modificar la escala de colores con que rellena cada punto, para tener una representación visual de la altitud en cada punto.



API REST para Datasets Vectoriales

Un Dataset vectorial se identifica por un archivo de objetos con propiedades geométricas y otros atributos descriptivos (propiedades de cada **"feature"**).

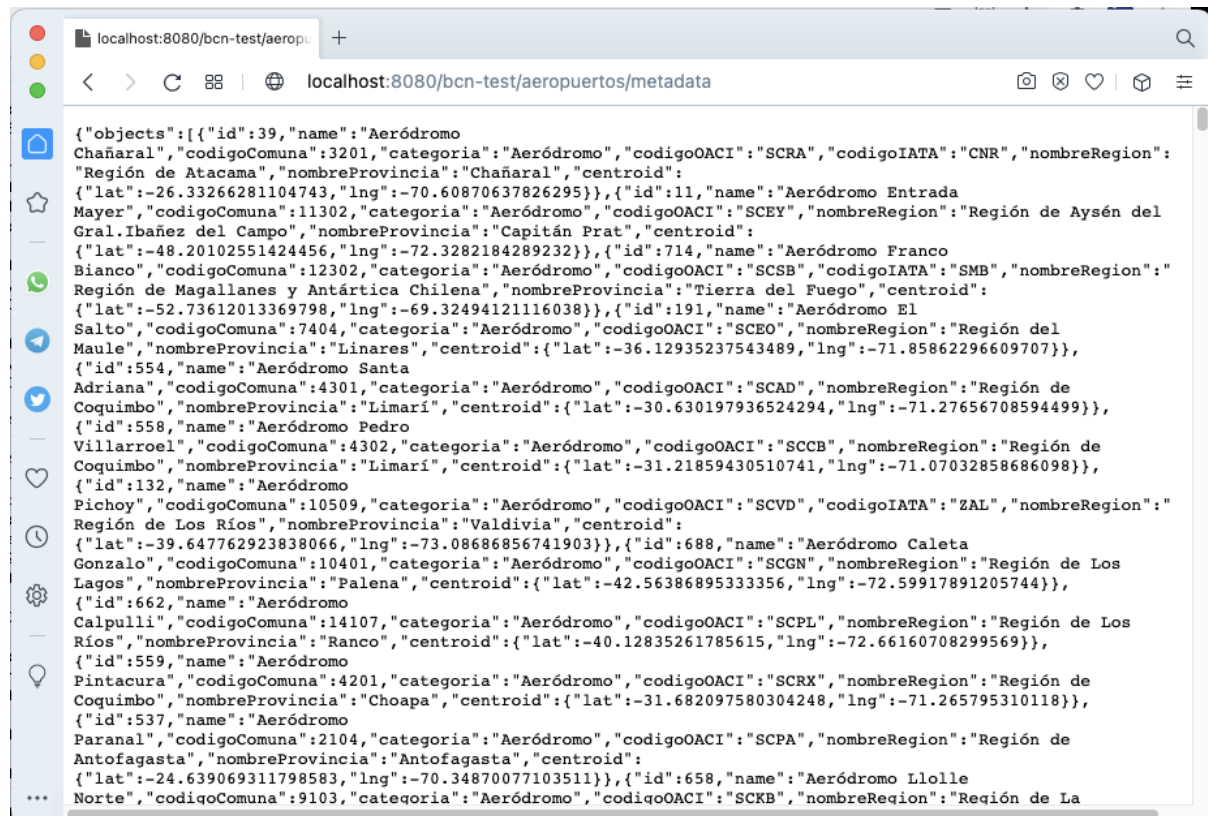
Usando el puerto WEB configurado para el GeoServer, es posible consultar los atributos no geométricos (propiedades) de un Dataset mediante un request GET a la URL.

<http://IP-SERVIDOR:PUERTO/dataSetCode/file/metadata>

En nuestro caso:

<localhost:8180/bcn-test/aeropuertos/metadata>

En el caso del ejemplo anterior:



```
{
  "objects": [
    {
      "id": 39,
      "name": "Aeródromo Chañaral",
      "codigoComuna": 3201,
      "categoria": "Aeródromo",
      "codigoOACI": "SCRA",
      "codigoIATA": "CNR",
      "nombreRegion": "Región de Atacama",
      "nombreProvincia": "Chañaral",
      "centroid": {
        "lat": -26.33266281104743,
        "lng": -70.60870637826295
      }
    },
    {
      "id": 11,
      "name": "Aeródromo Entrada Mayer",
      "codigoComuna": 11302,
      "categoria": "Aeródromo",
      "codigoOACI": "SCEY",
      "nombreRegion": "Región de Aysén del Gral. Ibañez del Campo",
      "nombreProvincia": "Capitán Prat",
      "centroid": {
        "lat": -48.20102551424456,
        "lng": -72.3282184289232
      }
    },
    {
      "id": 714,
      "name": "Aeródromo Franco Bianco",
      "codigoComuna": 12302,
      "categoria": "Aeródromo",
      "codigoOACI": "SCSB",
      "codigoIATA": "SMB",
      "nombreRegion": "Región de Magallanes y Antártica Chilena",
      "nombreProvincia": "Tierra del Fuego",
      "centroid": {
        "lat": -52.73612013369798,
        "lng": -69.32494121116038
      }
    },
    {
      "id": 191,
      "name": "Aeródromo El Salto",
      "codigoComuna": 7404,
      "categoria": "Aeródromo",
      "codigoOACI": "SCEO",
      "nombreRegion": "Región del Maule",
      "nombreProvincia": "Linares",
      "centroid": {
        "lat": -36.12935237543489,
        "lng": -71.85862296609707
      }
    },
    {
      "id": 554,
      "name": "Aeródromo Santa Adriana",
      "codigoComuna": 4301,
      "categoria": "Aeródromo",
      "codigoOACI": "SCAD",
      "nombreRegion": "Región de Coquimbo",
      "nombreProvincia": "Limarí",
      "centroid": {
        "lat": -30.630197936524294,
        "lng": -71.27656708594499
      }
    },
    {
      "id": 558,
      "name": "Aeródromo Pedro Villarroel",
      "codigoComuna": 4302,
      "categoria": "Aeródromo",
      "codigoOACI": "SCCB",
      "nombreRegion": "Región de Coquimbo",
      "nombreProvincia": "Limarí",
      "centroid": {
        "lat": -31.21859430510741,
        "lng": -71.07032858686098
      }
    },
    {
      "id": 132,
      "name": "Aeródromo Pichoy",
      "codigoComuna": 10509,
      "categoria": "Aeródromo",
      "codigoOACI": "SCVD",
      "codigoIATA": "ZAL",
      "nombreRegion": "Región de Los Ríos",
      "nombreProvincia": "Valdivia",
      "centroid": {
        "lat": -39.647762923838066,
        "lng": -73.08686856741903
      }
    },
    {
      "id": 688,
      "name": "Aeródromo Caleta Gonzalo",
      "codigoComuna": 10401,
      "categoria": "Aeródromo",
      "codigoOACI": "SCGN",
      "nombreRegion": "Región de Los Lagos",
      "nombreProvincia": "Palena",
      "centroid": {
        "lat": -42.56386895333356,
        "lng": -72.59917891205744
      }
    },
    {
      "id": 662,
      "name": "Aeródromo Calpulli",
      "codigoComuna": 14107,
      "categoria": "Aeródromo",
      "codigoOACI": "SCPL",
      "nombreRegion": "Región de Los Ríos",
      "nombreProvincia": "Ranco",
      "centroid": {
        "lat": -40.12835261785615,
        "lng": -72.66160708299569
      }
    },
    {
      "id": 559,
      "name": "Aeródromo Pintacura",
      "codigoComuna": 4201,
      "categoria": "Aeródromo",
      "codigoOACI": "SCRX",
      "nombreRegion": "Región de Coquimbo",
      "nombreProvincia": "Choapa",
      "centroid": {
        "lat": -31.682097580304248,
        "lng": -71.265795310118
      }
    },
    {
      "id": 537,
      "name": "Aeródromo Paranal",
      "codigoComuna": 2104,
      "categoria": "Aeródromo",
      "codigoOACI": "SCPA",
      "nombreRegion": "Región de Antofagasta",
      "nombreProvincia": "Antofagasta",
      "centroid": {
        "lat": -24.639069311798583,
        "lng": -70.34870077103511
      }
    },
    {
      "id": 658,
      "name": "Aeródromo Llolle Norte",
      "codigoComuna": 9103,
      "categoria": "Aeródromo",
      "codigoOACI": "SCKB",
      "nombreRegion": "Región de La"
    }
  ]
}
```

Se retorna un objeto JSON con una propiedad **"objects"** que contiene uno a uno los objetos vectoriales con un **"id"**, un **"name"** y el resto de las propiedades copiadas desde el archivo geoJSON original.

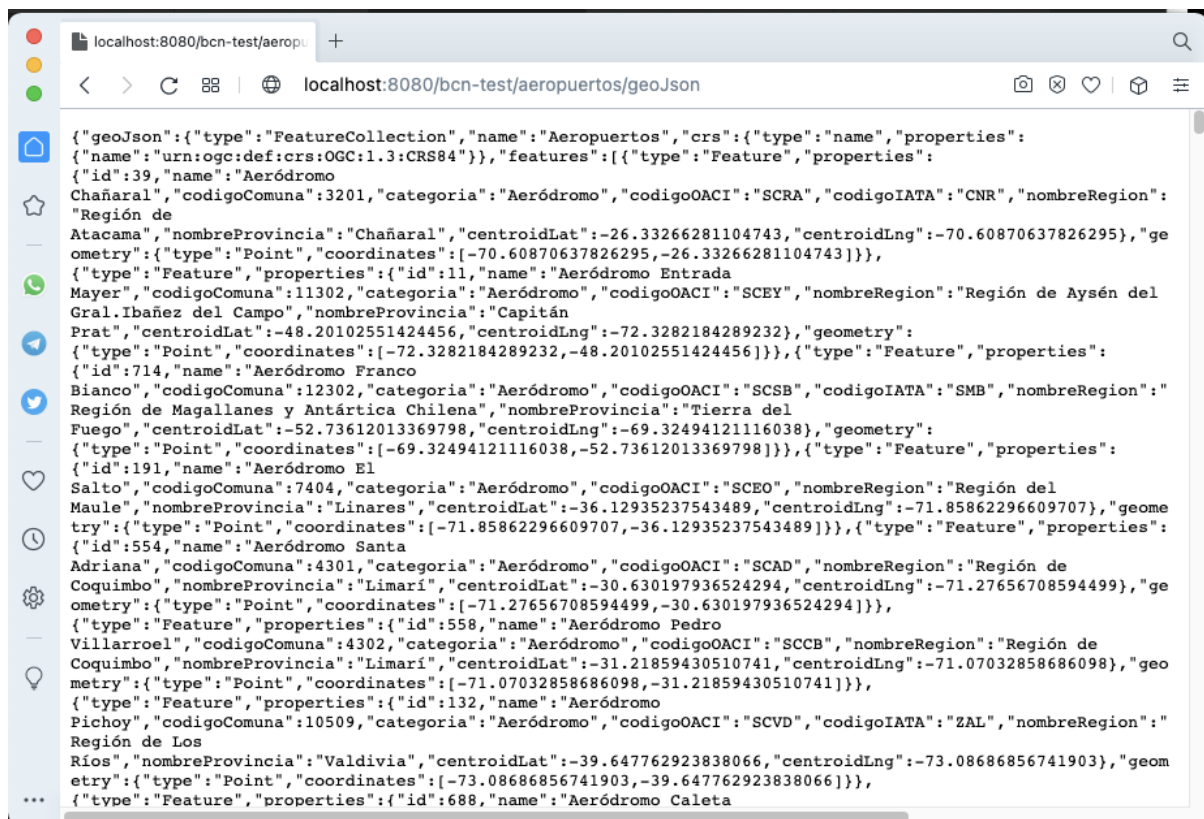
El archivo geoJSON importado (con sus propiedades geométricas y no geométricas) puede también ser consultado mediante el API REST, usando un request GET a la url:

<http://IP-SERVIDOR:PUERTO/dataSetCode/file/geoJson>

En nuestro caso:

<localhost:8180/bcn-test/aeropuertos/geoJson>

El resultado del ejemplo anterior es:



```
{
  "geoJson": {
    "type": "FeatureCollection",
    "name": "Aeropuertos",
    "crs": {
      "type": "name",
      "properties": {
        "name": "urn:ogc:def:crs:OGC:1.3:CRS84"
      }
    },
    "features": [
      {
        "type": "Feature",
        "properties": {
          "id": 39,
          "name": "Aeródromo Chañaral",
          "codigoComuna": 3201,
          "categoria": "Aeródromo",
          "codigoOACI": "SCRA",
          "codigoIATA": "CNR",
          "nombreRegion": "Región de Atacama",
          "nombreProvincia": "Chañaral",
          "centroidLat": -26.33266281104743,
          "centroidLng": -70.60870637826295
        },
        "geometry": {
          "type": "Point",
          "coordinates": [
            -70.60870637826295,
            -26.33266281104743
          ]
        }
      },
      {
        "type": "Feature",
        "properties": {
          "id": 11,
          "name": "Aeródromo Entrada Mayer",
          "codigoComuna": 11302,
          "categoria": "Aeródromo",
          "codigoOACI": "SCEY",
          "nombreRegion": "Región de Aysén del Gral. Ibañez del Campo",
          "nombreProvincia": "Capitán Prat",
          "centroidLat": -48.20102551424456,
          "centroidLng": -72.3282184289232
        },
        "geometry": {
          "type": "Point",
          "coordinates": [
            -72.3282184289232,
            -48.20102551424456
          ]
        }
      },
      {
        "type": "Feature",
        "properties": {
          "id": 714,
          "name": "Aeródromo Franco Bianco",
          "codigoComuna": 12302,
          "categoria": "Aeródromo",
          "codigoOACI": "SCSB",
          "codigoIATA": "SMB",
          "nombreRegion": "Región de Magallanes y Antártica Chilena",
          "nombreProvincia": "Tierra del Fuego",
          "centroidLat": -52.73612013369798,
          "centroidLng": -69.32494121116038
        },
        "geometry": {
          "type": "Point",
          "coordinates": [
            -69.32494121116038,
            -52.73612013369798
          ]
        }
      },
      {
        "type": "Feature",
        "properties": {
          "id": 191,
          "name": "Aeródromo El Salto",
          "codigoComuna": 7404,
          "categoria": "Aeródromo",
          "codigoOACI": "SCEO",
          "nombreRegion": "Región del Maule",
          "nombreProvincia": "Linares",
          "centroidLat": -36.12935237543489,
          "centroidLng": -71.85862296609707
        },
        "geometry": {
          "type": "Point",
          "coordinates": [
            -71.85862296609707,
            -36.12935237543489
          ]
        }
      },
      {
        "type": "Feature",
        "properties": {
          "id": 554,
          "name": "Aeródromo Santa Adriana",
          "codigoComuna": 4301,
          "categoria": "Aeródromo",
          "codigoOACI": "SCAD",
          "nombreRegion": "Región de Coquimbo",
          "nombreProvincia": "Limarí",
          "centroidLat": -30.630197936524294,
          "centroidLng": -71.27656708594499
        },
        "geometry": {
          "type": "Point",
          "coordinates": [
            -71.27656708594499,
            -30.630197936524294
          ]
        }
      },
      {
        "type": "Feature",
        "properties": {
          "id": 558,
          "name": "Aeródromo Pedro Villaruel",
          "codigoComuna": 4302,
          "categoria": "Aeródromo",
          "codigoOACI": "SCCB",
          "nombreRegion": "Región de Coquimbo",
          "nombreProvincia": "Limarí",
          "centroidLat": -31.21859430510741,
          "centroidLng": -71.07032858686098
        },
        "geometry": {
          "type": "Point",
          "coordinates": [
            -71.07032858686098,
            -31.21859430510741
          ]
        }
      },
      {
        "type": "Feature",
        "properties": {
          "id": 132,
          "name": "Aeródromo Pichoy",
          "codigoComuna": 10509,
          "categoria": "Aeródromo",
          "codigoOACI": "SCVD",
          "codigoIATA": "ZAL",
          "nombreRegion": "Región de Los Ríos",
          "nombreProvincia": "Valdivia",
          "centroidLat": -39.647762923838066,
          "centroidLng": -73.08686856741903
        },
        "geometry": {
          "type": "Point",
          "coordinates": [
            -73.08686856741903,
            -39.647762923838066
          ]
        }
      },
      {
        "type": "Feature",
        "properties": {
          "id": 688,
          "name": "Aeródromo Caleta"
        }
      }
    ]
  }
}
```

Geoserver es capaz de generar tiles de vectores (Capas Vector Tiles) según el formato definido en:

https://en.wikipedia.org/wiki/Vector_tiles

Cada “tile” es referenciado (al igual que las capas de mapas con formato **Map-Tile**, usando las coordenadas x,y,z de la especificación. Por ejemplo, la url:

<http://localhost:8080/bcn-test/aeropuertos/tile/6/19/37>

Retorna la lista de aeropuertos dentro de la sección indicada.

Las capas vectoriales publicadas actualmente por GeoServer son públicas. Por ejemplo, la capa de regiones de chile puede obtenerse desde:

<https://geoserver.geoos.org/ine-regional/regiones/geoJSON>

Capas Raster

Los archivos raster soportados hasta el momento por GeoServer son NetCDF y GRIB, y se accede a ellos usando la línea de comando de la herramienta GDAL (<https://gdal.org>). Estos archivos manejan el concepto de “bandas”, las que se importan a GeoServer como variables.

Se usará acá un archivo de ejemplo que corresponde a la salida de la ejecución de un modelo oceanográfico como un archivo NetCDF. El archivo está contenido en la carpeta “nc” del proyecto de ejemplo en github:

<https://github.com/geoos/geoserver-demo>

El primer paso para poder publicar un archivo NetCDF o GRIB2 usando GeoServer es conocer la estructura de este archivo. Esto se puede hacer utilizando los comandos de GDAL, al igual que internamente lo hace GeoServer. La instalación de GDAL puede ser un proceso complejo, dependiente del sistema operativo e instalaciones de otras dependencias que ya puedan existir. Afortunadamente es posible usar la misma instalación que ya usa GeoServer entrando al contenedor Docker (imagen docker en ejecución) en el mismo computador en donde se han ejecutado las pruebas anteriores.

El contenedor docker de GeoServer tiene mapeado un volumen “/home/data” hacia un subdirectorio en el computador de prueba, en particular en los ejemplos se ha usado /home/geoos/data. Lo anterior significa que desde dentro del contenedor en ejecución la ruta “/home/data” apuntará al directorio /opt/geoos/data de nuestro computador. Como queremos analizar la estructura del archivo de ejemplo que usaremos, podemos copiar este archivo (mercatorglorys12v1_gl12_mean_2018-12-31.nc) al directorio /opt/geoos/data y será visibles desde el contenedor.

A continuación, debemos iniciar una sesión de terminal (consola bash de linux) dentro del contenedor en ejecución. Para ello, primero listamos los contenedores activos usando el comando “**docker ps**” y buscamos en la columna “**name**” el que comience por “**geoos_geoserver**”. Copiamos el valor de la primera columna “**Container Id**”.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
d16267c38811	docker.homesjota.net/geos/portal:latest	"docker-entrypoint.s-"	5 hours ago	Up 5 hours	8080/tcp	geoos_portal.1.nc
276f579ab0fd	docker.alpine-ftp-server:latest	"/bin/start_vsftpd.sh"	3 days ago	Up 3 days	21/tcp, 21000-21010/tcp	ftp_server.1.40b0b00f030010000101
eae9732d5fa1	docker.homesjota.net/geos/geoserver:latest	"node index"	7 days ago	Up 7 days	8080/tcp	geoos_geoserver.1.idq7g211antgkwdmu39br0e
bee85b588f9	mongo:4.4.1-bionic	"docker-entrypoint.s-"	7 days ago	Up 7 days	27017/tcp	geoos_db.1.3nd9vddzq701seuzv50k47gdx

Usando el ID copiado, ejecutamos el comando “**bash**” dentro del contenedor identificado por el ID. Esto se consigue con el comando **docker exec -it**, de la forma:

```
root@jotanuc:/opt/geoos# docker exec -it eae9732d5fa1 bash
root@eae9732d5fa1:/usr/src/app#
```

Con el comando anterior iniciamos una sesión de trabajo dentro del contenedor, en donde ya está instalado GDAL y ya están copiados los archivos grib. Para verificar lo anterior, ejecutamos los siguientes comandos:

```
root@f02e0c2f6abd:/usr/src/app# cd /home/data/
root@f02e0c2f6abd:/home/data# ls -la *.nc
-rw-r--r-- 1 root root 49282724 May 23 20:13 mercatorglorys12v1_gl12_mean_2018-12-31.nc
root@f02e0c2f6abd:/home/data# gdalinfo mercatorglorys12v1_gl12_mean_2018-12-31.nc
```

El resultado del último comando (**gdalinfo**) nos muestra que el archivo analizado se compone de una serie de “**SUBDATASETS**”, en particular 11 de ellos.

```

Subdatasets:
SUBDATASET_1_NAME=NETCDF:"mercatorglorys12v1_gl12_mean_2018-12-31.nc":m1otst
SUBDATASET_1_DESC=[1x277x589] ocean_mixed_layer_thickness_defined_by_sigma_theta (16-bit integer)
SUBDATASET_2_NAME=NETCDF:"mercatorglorys12v1_gl12_mean_2018-12-31.nc":siconc
SUBDATASET_2_DESC=[1x277x589] sea_ice_area_fraction (16-bit integer)
SUBDATASET_3_NAME=NETCDF:"mercatorglorys12v1_gl12_mean_2018-12-31.nc":thetao
SUBDATASET_3_DESC=[1x36x277x589] sea_water_potential_temperature (16-bit integer)
SUBDATASET_4_NAME=NETCDF:"mercatorglorys12v1_gl12_mean_2018-12-31.nc":usi
SUBDATASET_4_DESC=[1x277x589] eastward_sea_ice_velocity (16-bit integer)
SUBDATASET_5_NAME=NETCDF:"mercatorglorys12v1_gl12_mean_2018-12-31.nc":sithick
SUBDATASET_5_DESC=[1x277x589] sea_ice_thickness (16-bit integer)
SUBDATASET_6_NAME=NETCDF:"mercatorglorys12v1_gl12_mean_2018-12-31.nc":bottomT
SUBDATASET_6_DESC=[1x277x589] sea_water_potential_temperature_at_sea_floor (16-bit integer)
SUBDATASET_7_NAME=NETCDF:"mercatorglorys12v1_gl12_mean_2018-12-31.nc":vsi
SUBDATASET_7_DESC=[1x277x589] northward_sea_ice_velocity (16-bit integer)
SUBDATASET_8_NAME=NETCDF:"mercatorglorys12v1_gl12_mean_2018-12-31.nc":vo
SUBDATASET_8_DESC=[1x36x277x589] northward_sea_water_velocity (16-bit integer)
SUBDATASET_9_NAME=NETCDF:"mercatorglorys12v1_gl12_mean_2018-12-31.nc":uo
SUBDATASET_9_DESC=[1x36x277x589] eastward_sea_water_velocity (16-bit integer)
SUBDATASET_10_NAME=NETCDF:"mercatorglorys12v1_gl12_mean_2018-12-31.nc":so
SUBDATASET_10_DESC=[1x36x277x589] sea_water_salinity (16-bit integer)
SUBDATASET_11_NAME=NETCDF:"mercatorglorys12v1_gl12_mean_2018-12-31.nc":zos
SUBDATASET_11_DESC=[1x277x589] sea_surface_height_above_geoid (16-bit integer)

```

Supongamos inicialmente que deseamos rescatar la variable **sea_water_salinity**, la que está en el **SUBDATASET** número 10. Para ver más detalles acerca de ese **SUBDATASET**, le indicamos su número al comando **gdalinfo**, con el parámetro **-sd**

```

root@f02e0c2f6abd:/home/data# gdalinfo -sd 10 mercatorglorys12v1_gl12_mean_2018-12-31.nc
Driver: netCDF/Network Common Data Format
Files: mercatorglorys12v1_gl12_mean_2018-12-31.nc
Size is 589, 277
Origin = (-102.04166666666671,-51.958333333333336)
Pixel Size = (0.083333333333333,-0.083333333333333)
Metadata:
  depth#axis=Z
  depth#long_name=Depth
  depth#positive=down

```

Analizando el resultado del comando, podemos ver una sección con el área de datos incluidos en el archivo:

```

Corner Coordinates:
Upper Left  (-102.0416667, -51.9583333)
Lower Left  (-102.0416667, -75.0416667)
Upper Right  (-52.9583333, -51.9583333)
Lower Right  (-52.9583333, -75.0416667)
Center      (-77.5000000, -63.5000000)

```

Como se observa en la salida completa del comando anterior, se muestran 36 bandas. Cada banda corresponde a una capa de datos para la misma área anterior, con algún cambio en otra dimensión. En este caso, comparando los datos de cada banda, vemos que lo que cambia es el atributo **NETCDF_DIM_depth** que representa la profundidad para la que se obtiene el valor del modelo.

```

Band 1 Block=589x1 Type=Int16, ColorInterp=Undefined
NoData Value=-32767
Unit Type: 1e-3
Offset: -0.00152592547237873, Scale:0.00152592547237873
Metadata:
  add_offset=-0.001525925472378731
  cell_methods=area: mean
  long_name=Salinity
  NETCDF_DIM_depth=0.49402499
  NETCDF_DIM_time=604836
  NETCDF_VARNAME=so
  scale_factor=0.001525925472378731
  standard_name=sea_water_salinity
  units=1e-3
  unit_long=Practical Salinity Unit
  _ChunkSizes={1,7,341,720}
  _FillValue=-32767
Band 2 Block=589x1 Type=Int16, ColorInterp=Undefined
NoData Value=-32767
Unit Type: 1e-3
Offset: -0.00152592547237873, Scale:0.00152592547237873
Metadata:
  add_offset=-0.001525925472378731
  cell_methods=area: mean
  long_name=Salinity
  NETCDF_DIM_depth=1.541375
  NETCDF_DIM_time=604836
  NETCDF_VARNAME=so
  scale_factor=0.001525925472378731
  standard_name=sea_water_salinity
  units=1e-3
  unit_long=Practical Salinity Unit
  _ChunkSizes={1,7,341,720}
  _FillValue=-32767
Band 3 Block=589x1 Type=Int16, ColorInterp=Undefined
NoData Value=-32767
Unit Type: 1e-3
Offset: -0.00152592547237873, Scale:0.00152592547237873
Metadata:
  add_offset=-0.001525925472378731
  cell_methods=area: mean
  long_name=Salinity
  NETCDF_DIM_depth=2.645669
  NETCDF_DIM_time=604836
  NETCDF_VARNAME=so

```

Al igual que el caso de las capas temporales, para agregar capas raster, es necesario definir un nuevo Dataset. En el caso de las capas raster, un “Dataset” corresponde a un archivo de origen NetCDF o GRIB2 que puede contener múltiples variables, cada una de ellas clasificadas por otra dimensión (profundidad, por ejemplo).

Observación: GeoServer no maneja la temporalidad dentro del mismo archivo raster (grib2 o NetCDF) de las variables; es decir, la temporalidad no será extraída del tiempo indicado en el archivo según su estándar (base time / forecast time), sino que debe ser parte del nombre del archivo. En el caso en que en el mismo archivo se tengan mediciones o resultados para varias temporalidades, éste archivo debe pre-procesarse y generar un nuevo archivo para cada temporalidad diferente que se desee importar.

Para no crear un nuevo proveedor de datos, se usará acá el mismo “bcn” del ejemplo Vectorial. Agregamos a la sección “Datasets” del archivo `geoos/config/config.hjson` la línea:

```
oce-test:{name:"Prueba Oceanográfica", provider:"bcn"}
```


Al grabar el archivo y monitorear el log del GeoServer, veremos que se despliega un error. Eso es porque se debe recordar que cada Dataset definido en el servidor requiere su propio archivo de configuración. En este caso, debe llamarse **“oce-test.hjson”**. Creamos el archivo con el siguiente contenido:

```
{
  dataSet: {type: "raster", format:"netCDF"}
  #
  # Raster config file
  #
  # Importer config
  # Finished files can be deleted or moved to /home/data/finished

  deleteFinishedFiles: false

  # Temporality associaed to each imported file.
  # All time values in dataSets are UTC
  # Depending on temporality, imported files must follow a name rule to incorporate the date and
time 1,2,3,4,6,12 hours: ${dataSet-code}_YYYY-MM-DD_HH-mm
  # 1 day: ${dataSet-code}_YYYY-MM-DD
  # 1 month ${dataSet-code}_YYYY-MM
  # 1 year ${dataSet-code}_YYYY
  # none (non temporary data): ${dataSet-code}
  # fixed (fixed-period): TODO

  temporality:"none"
  #Grid Query Config
  grid:{
    maxWidth: 150
    maxHeight: 150
    # gdal_translate resamplig algorithm
    # nearest (default),bilinear,cubic,cubicspline,lanczos,average,mode
    resampling: nearest
  }

  #Contour Query Config
  contour:{
    maxWidth: 100
    maxHeight: 100
    # gdal_translate resamplig algorithm
    # nearest (default),bilinear,cubic,cubicspline,lanczos,average,mode
    resampling: bilinear
  }

  ## Variables
  variables: {
    SALINIDAD:{
      selector:{NETCDF_VARNAME:"so", NETCDF_DIM_depth:0.49402499}
      name: "Salinidad a 0.5m"
    }
  }
}
```

```

unit: psu
options:{
  decimals:2
  subjects:["oce"]
  colorScale:{name:"Verde a Rojo", auto:true, clipOutOfRange:false}
  initialVisualizers:{shader:true}
}
}
}
}
}

```

Debido a que el archivo contiene múltiples **DATASETS** y bandas para la misma variable (**SO**, en el archivo original) debemos indicar un **“filtro”** o selectores para que el importador pueda discriminar y encontrar la banda deseada. El selector corresponde a una serie de atributos y valores que se deben satisfacer al mismo tiempo (condición **and**).

El archivo **.nc** original que se descargó desde el proyecto de ejemplo debe ser renombrado para que GeoServer lo reconozca dentro del Dataset que creamos. Debe llamarse igual que el código del Dataset y, como en este caso se definió sin temporalidad, no se agrega marca de tiempo.

Se renombra el archivo a **“oce-test.nc”** y se copia al directorio **“data/import”**. Luego de unos segundos debería desaparecer y el archivo de log del GeoServer mostrar el resultado:

```

17:10:31 [debug] Importing oce-test.nc using NetCDF Raster Importer
17:10:36 [debug] -> Importing SALINIDAD
17:10:37 [info] Creating directory: /home/data/oce-test
17:10:37 [debug] File Imported: oce-test.nc
17:10:37 [debug] Moving file: /home/data/working/oce-test.nc to '/home/data/finished'

```

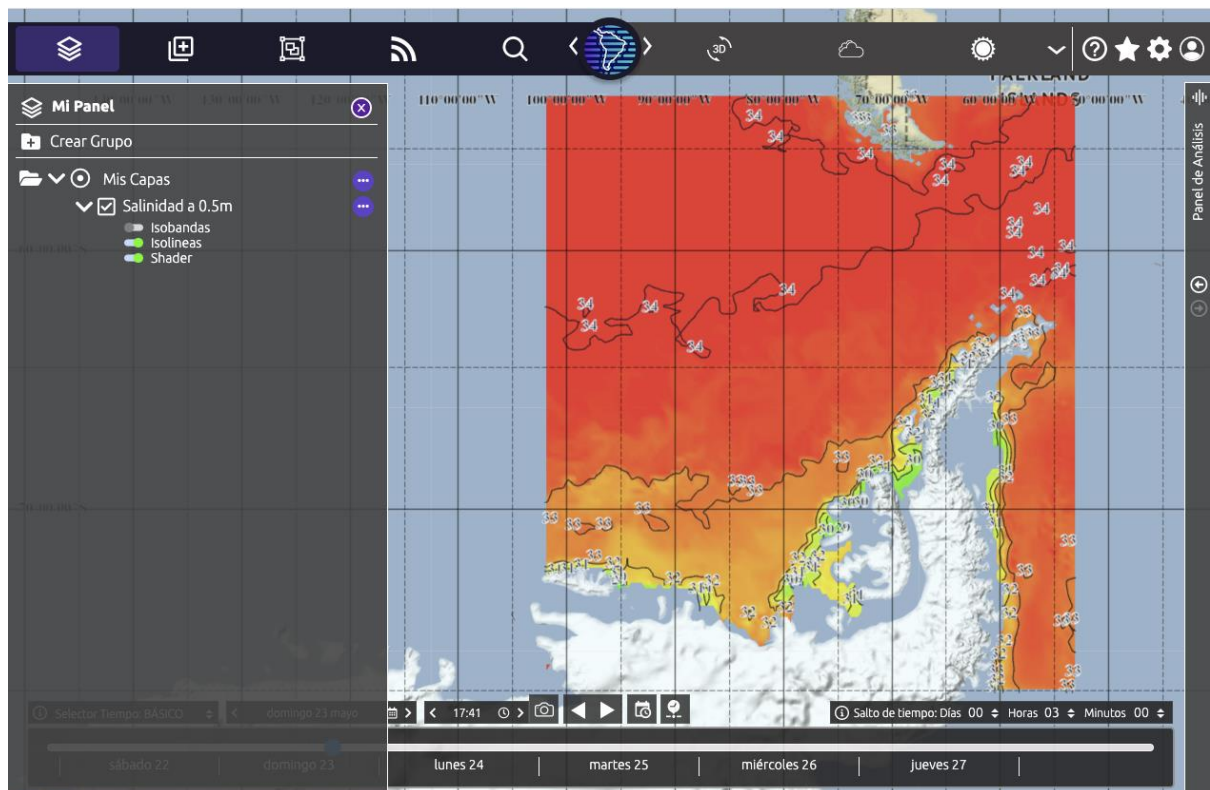
Los visualizadores de variables raster pueden utilizar escalas de colores. Las escalas también se definen dentro del archivo de configuración del GeoServer y pueden usar recursos (archivos que se publican) web, como archivos **“pg”** con rangos de valores y colores para las diferentes escalas. Para este ejemplo, sólo para asegurar que se cuenta con al menos una escala, debemos asegurar que exista el elemento con nombre **“Verde a Rojo”** en la lista que se define en la sección **colorScales** de **config.hjson**:

```

## Scales
colorScales:[{
  name:"verde a rojo", type:"linear-hsl"
  config:{s:"100%", l:"50%"}
}]

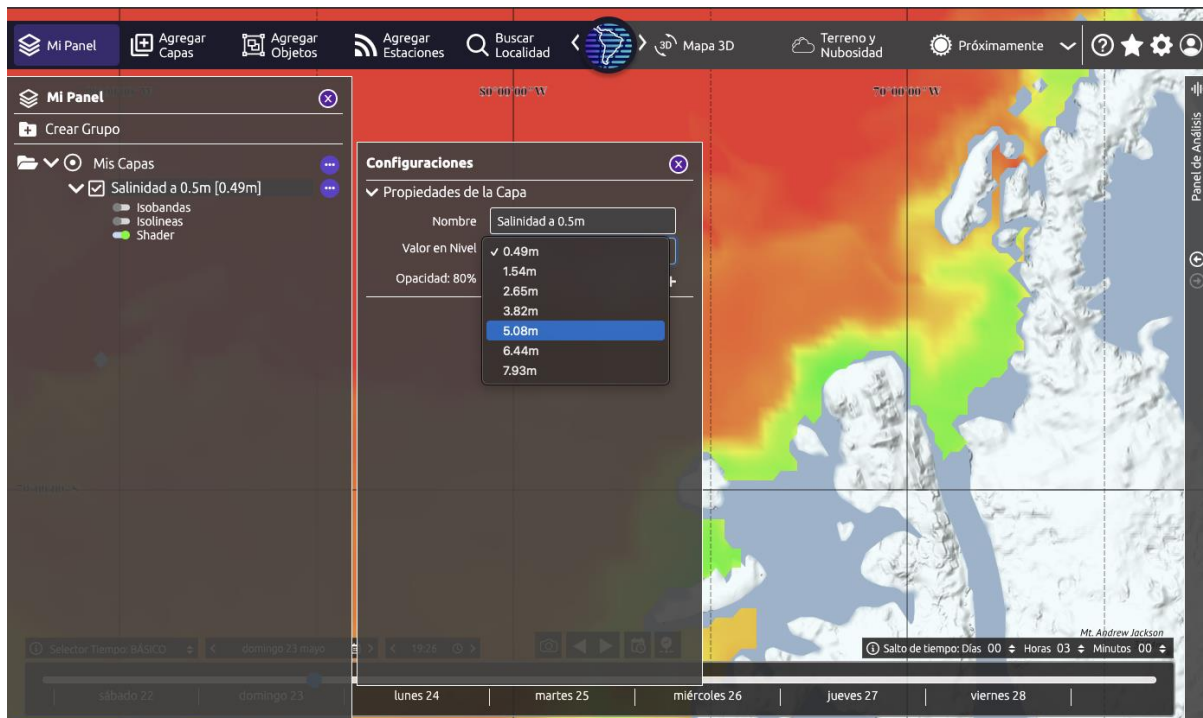
```

Al recargar la página, agregamos la nueva capa **“Salinidad a 0.5m”** que aparece y movemos el mapa a la zona austral de Chile.



Como antes se mencionó, la variable original en el archivo NetCDF se provee en varios niveles de profundidad, en las diferentes bandas del SUBDATASET. GEOOs soporta el concepto de “niveles” asociados a una variable, el que puede utilizarse para representar una dimensión extra (a la ubicación y tiempo). En este caso, podemos rescatar los valores de interés de los niveles que se desea monitorear (se podría hacer con todos, pero seleccionaremos sólo algunos acá por simpleza)

Desde la salida del comando **gdalinfo** antes ejecutado, observando el valor del atributo “**NETCDF_DIM_depth**” seleccionamos algunos valores y cambiamos la configuración de la variable “**Salinidad**” para que se recupere para diferentes niveles. Uno de los atributos de la banda debe corresponder al discriminador del nivel, en este caso, la profundidad. El selector queda entonces sólo con el nombre de la variable, mientras que el atributo de profundidad se usa como discriminador de nivel.



API REST para capas de variables raster

Al igual que las capas vectoriales, los datos de las capas raster pueden ser consultados usando un API REST basada en requests HTTP de tipo GET.

El API REST de consulta de datos raster puede ser invocada desde cualquier sistema (o manualmente mediante un browser o usando comandos como **wget** o **curl**). Debido a que los request son de tipo GET, los parámetros de consulta son parte de la URL.

Todos los valores de tiempo (argumentos: **time**, **startTime**, **endTime**) pueden entregarse como milisegundos UTC (milisegundos desde 01-01-1970) o en formato YYYY-MM-DD-HH:MM, tiempo UTC.

El formato genérico de las consultas por el API REST del GeoServer es el siguiente:

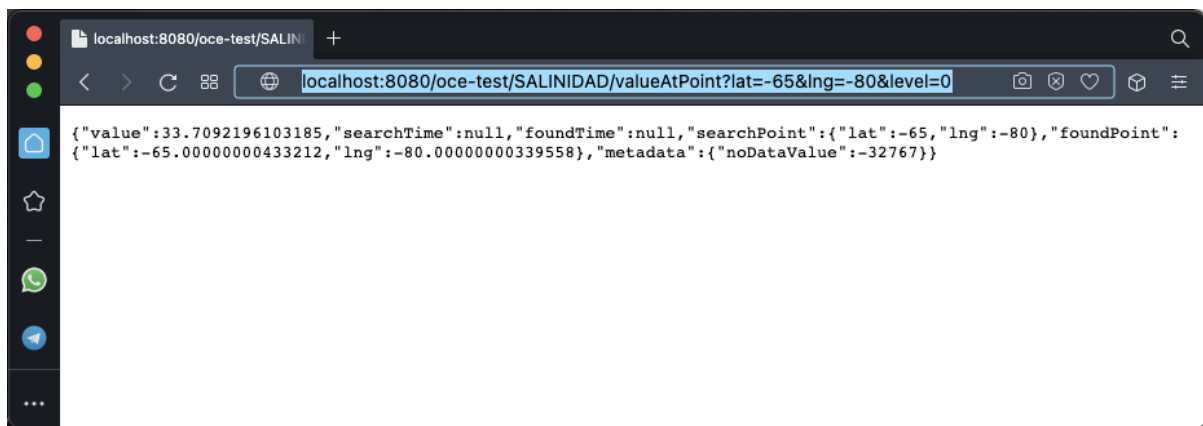
<http://IP-SERVIDOR:PUERTO/dataSetCode/variableCode/query?arg1=xx,arg2=xx>

Variable valueAtPoint

Retorna el valor de la variable en el punto con datos más cercano.

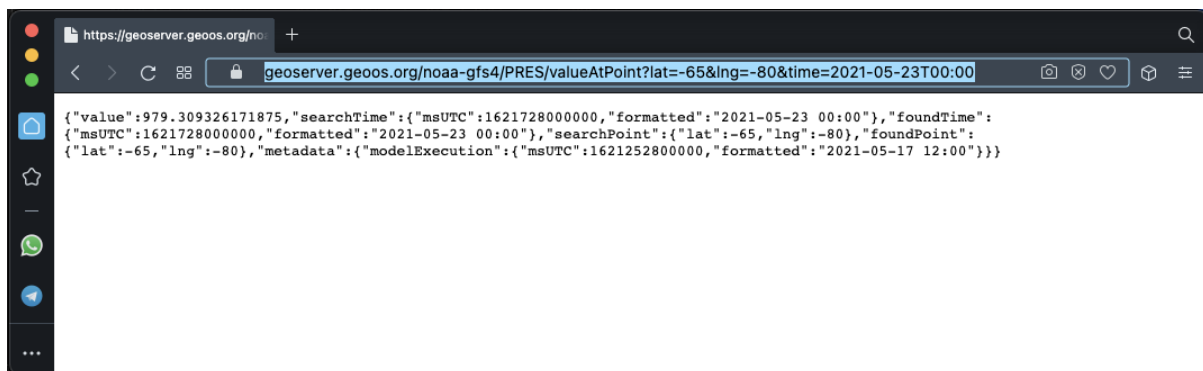
Para el ejemplo anterior:

<http://localhost:8180/oce-test/SALINIDAD/valueAtPoint?lat=-65&lng=-80&level=0>



Para la variable temporal de presión atmosférica al nivel del mar, en el sitio geos.org:

<https://geoserver.geos.org/noaa-gfs4/PRES/valueAtPoint?lat=-65&lng=-80&time=2021-05-23T00:00>



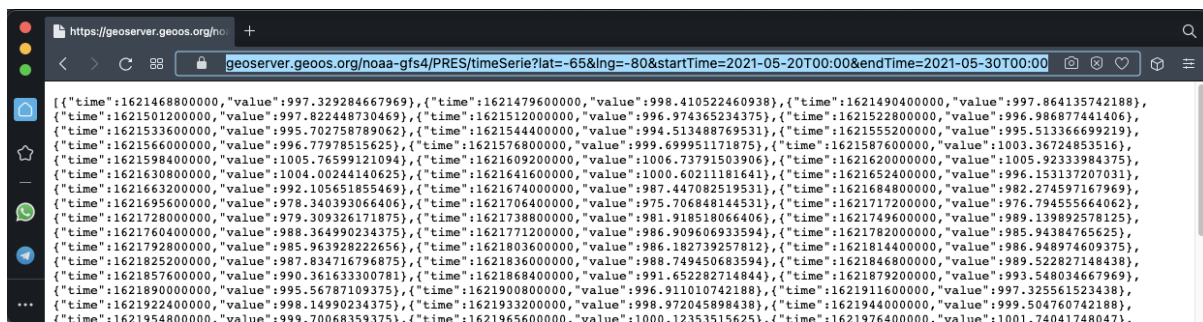
Como argumento se le debe indicar el punto (lat, lng), el tiempo si aplica, y el índice del nivel, si este aplica también. Se retorna el valor, las coordenadas en las que se encontró un dato (de acuerdo a la grilla original del archivo raster) y el tiempo encontrado para el valor (si aplica).

Variable timeSerie

Retorna una serie de tiempo con los valores para un punto (lat,lng) y un período de tiempo determinado. Aplica sólo a variables temporales.

Se retorna un arreglo JSON con los valores en cada tiempo (milisegundos UTC).

<https://geoserver.geos.org/noaa-gfs4/PRES/timeSerie?lat=-65&lng=-80&startTime=2021-05-20T00:00&endTime=2021-05-30T00:00>

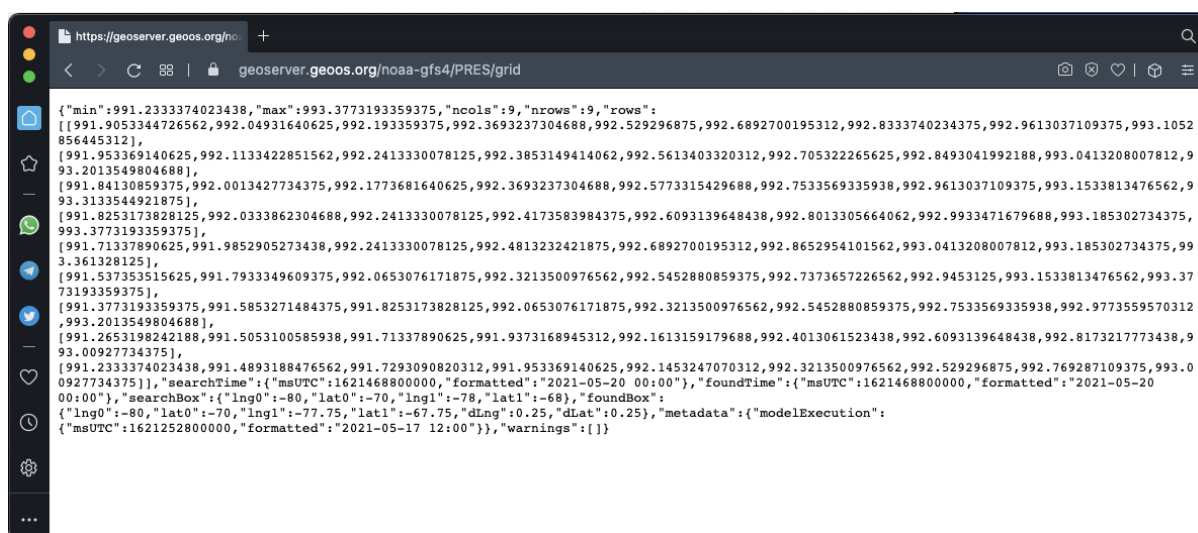


Variable grid

Retorna los valores de la variable para un área indicada, según la resolución espacial original del archivo o escalada (si se cumplen los máximos de datos indicados en la configuración). Se debe indicar el tiempo y nivel, si éstos aplican a cada variable.

Geoserver extenderá los límites del área consultada para incluir los puntos con datos que se consultan y posiblemente algunos en el exterior del área (si los consultados no calzan exactamente con la resolución del archivo). Tanto los límites consultados como los retornados son devueltos como resultados de la consulta, juntos a los valores de la variable en los puntos de la grilla.

<http://geoserver.geoos.org/noaa-gfs4/PRES/grid>



Se incluyen como respuesta los valores mínimo y máximo de la variable en el área y tiempo consultado, el número de filas y de columnas retornados.

Se entrega además el área consultada, el área encontrada con los límites extendidos, el tiempo consultado, el tiempo encontrado y los incrementos en latitud y longitud para la matriz regular encontrada.

Variables Isolines e isobands

Geoserver puede generar dinámicamente un archivo geoJSON con la representación como isolíneas o isobandas de las variables dentro de un área. Los parámetros son los mismos que para la grilla de datos. Excepcionalmente se le puede agregar un parámetro **"increment"** que determina la distancia (en unidades de la variable) entre líneas que se desea usar.

<http://geoserver.geoos.org/noaa-gfs4/PRES/isolines?n=-68&s=-70&e=-78&w=-80&time=2021-05-20T00:00>

