TRABAJO DE FIN DE MÁSTER

••••

Andrea Rosado Abad

Directores: Raquel Montorio Llovería y Daniel Borini Alves

Máster Universitario en Tecnologías de la Información Geográfica para la Ordenación del Territorio: Sistemas de Información Geográfica y Teledetección

King's College October 2017

Agradecimientos

And I would like to acknowledge ...

Prólogo

El presente Trabajo Fin de Máster en Tecnologías de la Información Geográfica para la Ordenación del Territorio: SIG y Teledetección, expone todas las tareas que se han realizado durante la colaboración en el Proyecto SIOSE-INNOVA: Innovaciones técnicas y metodológicas en el Sistema de Información sobre Ocupación del Suelo de España (SIOSE) y su aplicación en estudios geográficos. El investigador principal de este proyecto es Alfredo Ramón Morte, que cuenta además con la participación de los miembros del Laboratorio de Geomática y otros profesionales, en colaboración con el equipo de investigación responsable de la base de datos del SIOSE.

Resumen

En este trabajo...

Abstract

Una prueba de acencuación

Índice general

Índice de figuras

Índice de cuadros

1. INTRODUCCIÓN

Puntos de interés:

- What?
- How?
- When?
- Who?
- Where?
- why or for what?

1.1. Análisis de la estructura del paisaje de LU/LC

1.2. Métricas de paisaje

1.3. Software que las calcula

1.4. Objetivos

2. METODOLOGÍA

El desarrollo del software requiere de una metodología de trabajo colaborativo basado en una serie de herramientas de contenerización y orquestación de manera eficiente, extensible y reproducible.

2.1. Software y plataformas

A lo largo de este proyecto se han utilizado una serie de softwares y plataformas para el desarrollo de la nueva herramienta, clasificados en diferentes grupos. Dentro de estos conjuntos se encuentran explicados con más detalle los procesos que corresponden según las aplicaciones que se utilizaron. Estas tareas se llevaban a cabo de forma diaria y se solían efectuar cada cierto tiempo. Estos procesos se dividen en tres: (1) actualización de ficheros, (2) trabajo colaborativo e (3) integración continua.

Antes que nada, se debe de tener en cuenta que los procesos están representados mediante diagramas de secuencia. Por ello, la lectura debe realizar de izquierda a derecha y de arriba a abajo. Las líneas discontinuas corresponden a las notificaciones o mensajes intercambiables que son enviados y recibidos entre distintos actores, y las líneas continuas corresponden a las operaciones que realiza y recibe cualquier actor. Además, con ello podemos relacionar la activación y desactivación del actor cuando éste envía o realiza cualquier operación. Finalmente, entre actores aparecen expresiones como condicionantes o alternativas (alt), opcionales (opt) y bucles o ciclos (loop), es decir, una operación que se repite tantas veces sea necesario o quiera el usuario. Y no menos importante, los actores son los softwares o plataformas utilizados durante el proceso.

2.1.1. Control de versiones

Todas las modificaciones que surgían de uno o varios archivos a lo largo de una tarea, se utilizaba el control de versiones que se encargaba de registrarlas. Gracias a este sistema, se tenía la posibilidad de recuperar una versión antigua en cualquier momento. Para ello se han utilizado dos tipos de sistemas de control:

- **Git**: permite tener un control de las versiones que se han distribuido con código abierto y libre, con mayor rapidez y eficacia, sobre el repositorio (lugar en el que se almacenan todos los archivos necesarios de una investigación) en el que estamos trabajando. Por

otro lado, este sistema proporciona la posibilidad de replicar y actualizar el repositorio de trabajo, además de añadir, eliminar, mover o reemplazar cualquier archivo. La ventaja de este tipo de sistemas es que se puede crear una rama (branch) colaborativa del repositorio original para tener un aseguramiento de los datos, ya que si en cualquier momento se registra algún problema o éstos se dañan, existe la posibilidad de volver a restaurarlos sin ningún problema desde el repositorio original. Así pues, se creó una nueva rama colaborativa paralela[1] a la rama original del proyecto para operar y modificar sin el riesgo de que los archivos originales fueran dañados. Una vez finalizados y revisados todos los cambios, se procedía a unirlos al repositorio original.

- **GitHub**: es una plataforma de desarrollo que alberga proyectos de software, como es el caso del proyecto SIOSE-INNOVA[2], y almacena de forma pública todo el código, trabajando de forma colaborativa utilizando el sistema de control de versiones de Git. En este caso, su presentación es más visual e interactiva, a través de una interfaz web.

Una de las tareas diarias que corresponde a esta subsección es la actualización de ficheros de la extensión. El proceso consistía en que desde la máquina local Git(local) el usuario ejecutaba un comando pull que se encargaba de obtener la última versión del propio repositorio y de revisar si han habido cambios desde la última actualización (este paso es opcional en el caso de que no se tengan los ficheros actualizados pero si se tiene la última versión, no hace falta ejecutar el anterior comando). Cuando la máquina local tenía la última versión, se iniciaba un loop, es decir, el usuario editaba los ficheros de su sistema las veces que eran necesarias. Hechas estas modificaciones, se añadían y actualizaban los ficheros mediante el comando commit, obteniendo a continuación un hash (identificador único) por cada cambio que se había realizado durante el proceso. A partir de aquí, el loop finalizaba si el usuario decidía que no habían más modificaciones que realizar. Finalmente, desde la máquina local, el usuario subía todos los cambios realizados a su propio repositorio y así tener una nueva versión actualizada (ver figura 2.1).

Otro de los procesos, que comparte esta y la siguiente subsección, corresponde al trabajo colaborativo de los usuarios entre sus propios repositorios y el repositorio Upstream (original) del proyecto. Desde el punto de vista más técnico, una de las primeras tareas era realizar un fork, es decir, una copia de todos los ficheros del repositorio Upstream a un nuevo repositorio del propio usuario. La ventaja de esto es que si en cualquier momento el repositorio del usuario sufría algún contratiempo, se podía volver a realizar un fork del repositorio Upstream. A continuación, si no había una copia en la máquina local Git (local), se ejecutaba

[1] https://github.com/andrearosado/pg_landmetrics
[2] https://github.com/siose-innova/pg_landmetrics

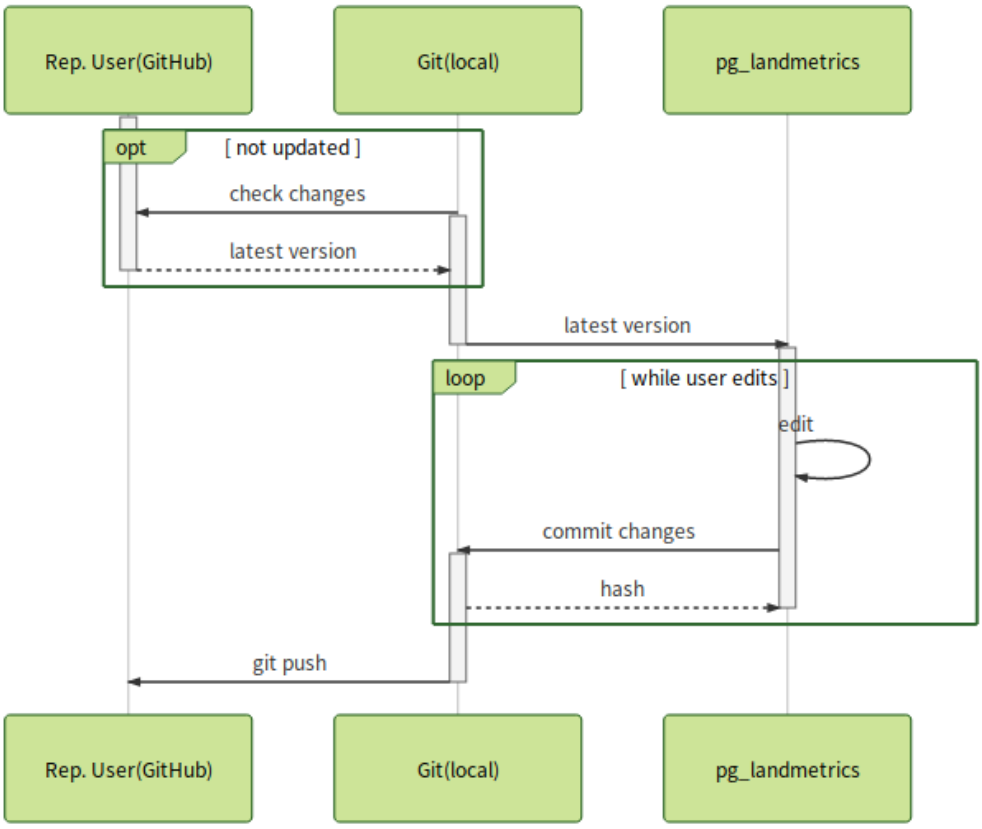

Figura 2.1 Flujo de proceso de actualización de ficheros.

el comando clone para obtener una copia de la última versión de todos los ficheros del repositorio del usuario. En el caso de que los ficheros no estaban actualizados, se ejecutaba una nueva instrucción donde Git(local) revisaba las últimas modificaciones y, si no los tuviera actualizados, se obtenía su última versión (estos últimos procesos son alternativos, ya que depende del estado de los ficheros). A partir de aquí, el sistema de ficheros quedaba actualizado y se iniciaba el proceso de trabajo colaborativo. En este punto, se producía un loop cada vez que el usuario editaba ficheros, añadía y actualizaba en la máquina local y finalmente, recibía un hash correspondiente a cada modificación que se había realizado. Una vez se hechos todos los cambios, éstos se subían al propio repositorio y en este momento el propio repositorio del usuario estaba por delante del repositorio Upstream ya que éste no tenía los últimos cambios guardado y el loop terminaba ya que no habían más modificaciones que realizar. Ahora es cuando interviene el trabajo colaborativo, es decir, desde el propio repositorio del usuario se enviaba una petición al repositorio Upstream para emparejar ambos a través de la función pull request desde la plataforma de GitHub. Si la petición era aceptada por el personal encargado del repositorio Upstream, el usuario ejecutaba el comando merge (unión) y recibía una notificación de sincronización completada. Finalmente los repositorios quedaban sincronizados y los ficheros con la última versión. Se volvían a subir los cambios al propio repositorio del usuario y se recibía una notificación de que los cambios habían sido correctos. Todo esto queda dentro de un loop hasta que el usuario decidía que no habían más cambios que realizar o ficheros que actualizar (ver figura 2.2).

2.1.2. Contenerización y orquestación de servicios

Uno de los objetivos de la extensión era obtener un alto grado de reproducibilidad y, por ello se ha llevado a cabo la virtualización de un sistema operativo a través de contenedores[3] Docker (ver figura 2.3).

Se han utilizado los siguientes elementos:

- **Docker**: es un servicio de integración continua de código abierto que automatiza el despliegue de ficheros de configuración de aplicación en contenedores facilitando la virtualización en el sistema operativo en cualquier máquina con Docker instalado. Este servicio permitió la portabilidad de las aplicaciones y la reproducibilidad del proyecto

[3]El uso de contenedores es relevante para la reproducibilidad, por eso se evalúan los siguientes criterios objetivos: portabilidad (se pueden reproducir en máquinas cuyos contenedores despliegan recursos compartidos), empaquetado (el software y los datos están compilados en una imagen binaria), reutilización de componentes (distribución entre usuarios de las imágenes sin necesidad de progresos intermedios), distribución (distribución de imágenes a través de plataformas) y versionado (integración de sistema de control de versiones en los archivos para la automatización la compilación de datos) (Boettiger, 2015).

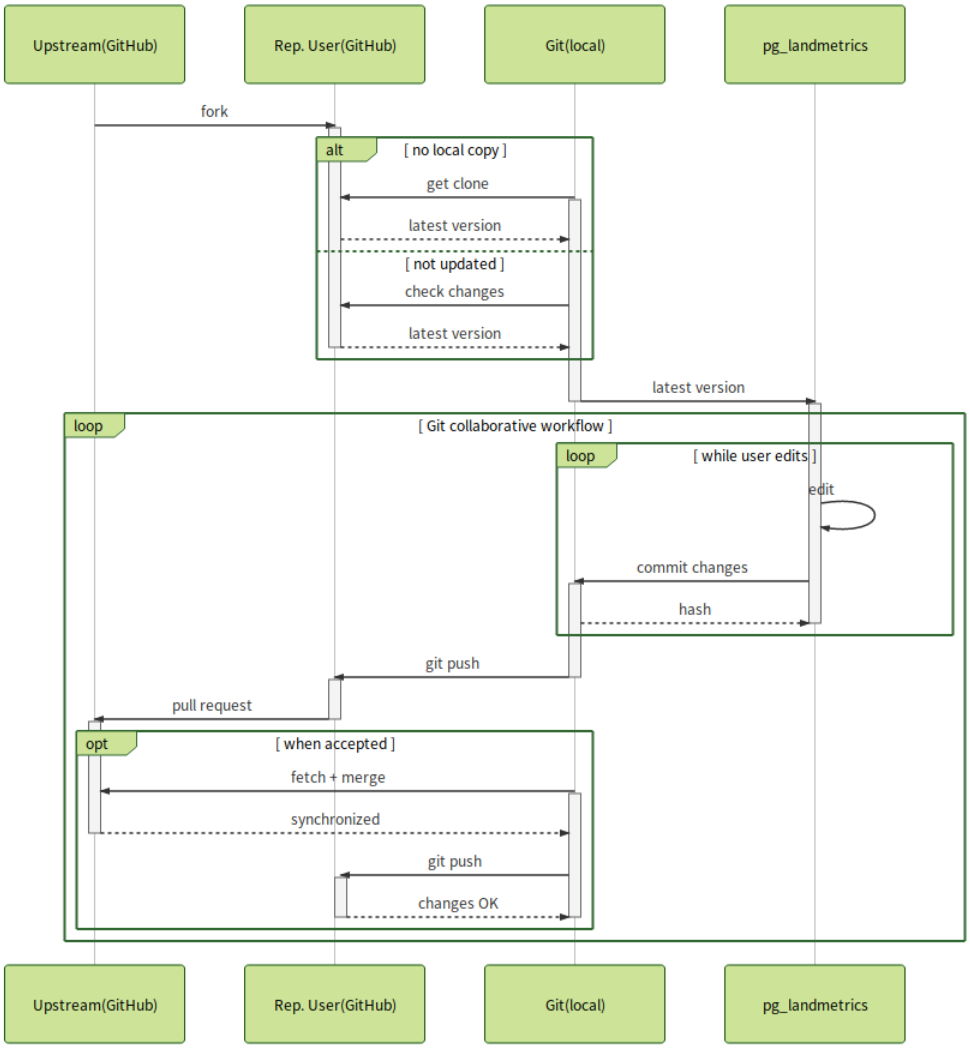

Figura 2.2 Flujo de proceso de trabajo colaborativo entre repositorios.

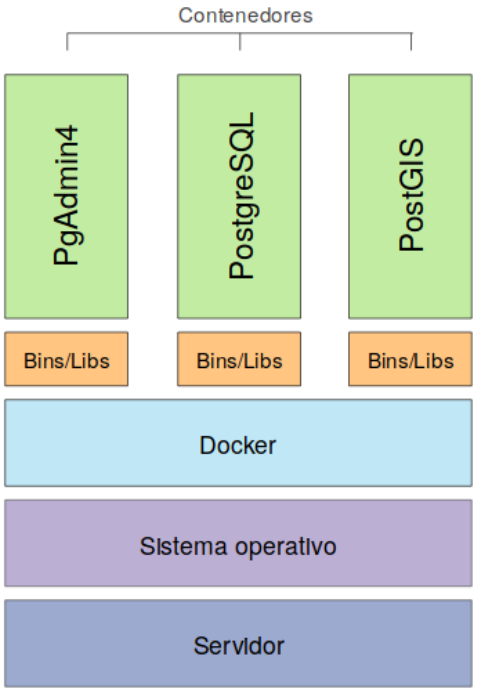

Figura 2.3 Contenerización del sistema operativo.

en un entorno de trabajo con todos los softwares, configuraciones y datos necesarios, con la ventaja de reproducirlo en cualquier máquina.

- **Docker Hub**: es una biblioteca digital donde se almacenan todas las imágenes Docker de aplicaciones en distintos repositorios. Desde aquí se obtuvieron las imágenes de las aplicaciones que se utilizaron.

- **Docker-compose**: es un fichero de configuración que despliega todos los servicios que sean necesarios durante el desarrollo de manera rápida y eficaz.

Otro de los procesos, que corresponde a esta subsección, es la integración continua del proyecto que conlleva la ejecución de control de versiones y la automatización. Desde el punto de vista del usuario, se ejecutaba el comando docker-compose build si la extensión había sido modificada. En el caso de que no estuviera modificada y no fuera necesario volver a construir el Docker-compose, solamente era necesario ejecutar el comando docker-compose up para iniciarlo. A partir de aquí, el Docker-compose orquestaba y se ocupaba de lanzar dos contenedores: PostgreSQL/PostGIS y PgAdmin. Una vez han sido lanzados dichos contenedores y se han recibido las notificaciones de que la conexión había sido satisfactoria, se iniciaba un loop mientras el usuario realizaba consultas a los contenedores. A continuación, ambos contenedores realizaban las consultas entre ambos y luego se enviaban al usuario los resultados de las consultas acompañado de una vista previa de ellos. Cuando ya eran adquiridos todos los resultados deseados, el loop se terminaba y se ejecutaba el comando docker-compose down para que el Docker-compose dejase de funcionar y el usuario recibiese una notificación por parte de este actor de que había sido apagado sin problema (ver figura 2.4).

2.1.3. Extensibilidad

El *makefile* (fichero encargado de organizar todo el código compilado de todos los programas que se deseen utilizar en la extensión) obtiene la capacidad para ampliar las funcionalidades de cualquier aplicación, como es el caso de la siguiente extensión:

- **PostgreSQL/PostGIS**: por un lado, PostgreSQL es un poderoso sistema de bases de datos relacional el cual ayuda a organizar todos los objetos en un conjunto de tablas y, en cuanto a PostGIS es una extensión que añade objetos geográficos a la base de datos relacional de PostgreSQL, pasando a ser una base de datos espacial. Ambas extensiones han permitido la construcción de consultas sobre las bases de datos utilizadas en el proyecto, tanto los de prueba como de los casos de estudio a partir de datos del SIOSE.

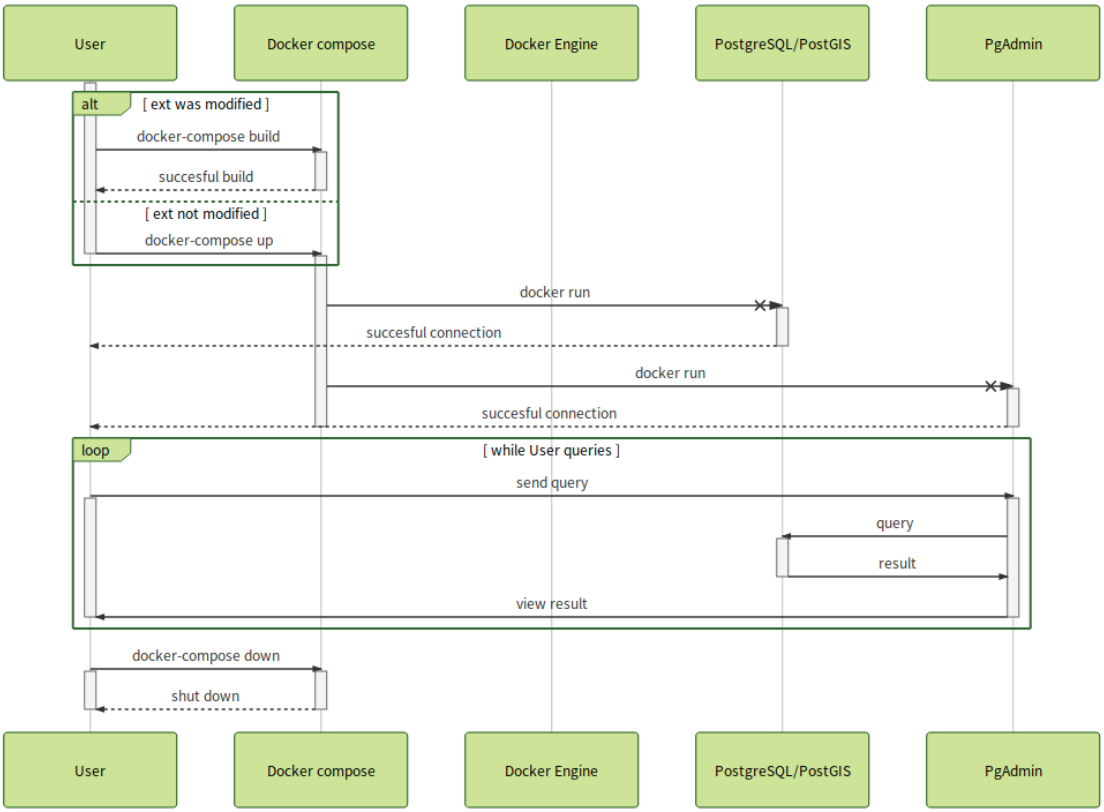

Figura 2.4 Flujo de proceso de integración continua.

Cuadro 2.1 Atributos del primer conjunto de datos.

Nombre	Tipo de campo	Descripción
gid	integer	Identificador único de cada polígono
geom	geometry	Geometría del polígono
category	character varying (text)	Clasificación de la cobertura del suelo
svg_color	character varying (text)	Color según tipo de categoría

2.1.4. Aplicaciones

Tanto para el conjunto de datos como para las consultas y funciones de SQL, se utilizaron aplicaciones que realizan unas determinadas tareas:

- **PgAdmin4**: es una aplicación de código abierto capaz de administrar y gestionar bases de datos PostgreSQL. En este caso, desde una interfaz web donde se han construido todas las consultas, funciones, agregados,... etc.
- **QGIS 2.18**: es una aplicación de escritorio SIG, de código abierto o libre, que analiza, maneja y opera con datos vectoriales, datos ráster y bases de datos. Además facilita la conexión entre las bases de datos espaciales como PostGIS. Gracias a este software se ha elaborado un conjunto de datos que se ha utilizado para comprobar el funcionamiento de las métricas de paisaje.

2.2. Conjunto de datos

Antes de seleccionar las métricas de paisaje que han sido utilizadas para complementar el nuevo software, ha sido necesario elaborar y procesar conjuntos de datos que posteriormente se han utilizado para realizar comprobaciones del funcionamiento de las métricas. Así pues, por una parte se ha creado un paisaje ficticio y por otra, se han utilizado los datos del SIOSE de dos zonas de estudio como casos de estudio reales.

Para el paisaje ficticio, se ha digitalizado desde cero todos los polígonos que comprenden esta zona de estudio a partir de herramientas de geoprocesamiento, geometría, vectorial y edición, utilizando QGIS. En la tabla de atributos del shapefile, cada polígono tiene identificador único, geometría, clasificación según el tipo de cobertura del suelo y color según el tipo de categoría al que pertenece.

En este paisaje se ha optado por una clasificación de etiquetas simples. En cuanto al color de cada tipo de categoría se ha definido y escogido a partir de los 147 colores que presenta

Cuadro 2.2 My caption

Category	svg_color
Agricultural area	lightsalmon
Continuous urban	lightcoral
Discontinuous urban	lightpink
Forest	forestgreen
Main road	gray
Secondary road	lightgrey
River	royalblue
Water body	mediumblue

Cuadro 2.3 Atributos del primer conjunto de datos.

Nombre	Tipo de campo	Descripción
ID_POLYGON	integer	Identificador único de cada polígono
GEOMETRY	geometry	Geometría del polígono
SIOSE_CODE	character varying (text)	Tipo de cobertura de cada polígono
SIOSE_XML	character varying (text)	Información completa del tipo de cobertura al que pertenece cada polígono, en formato XML
SUPERF_HA	character varying (text)	Superficie del polígono en hectáreas
OBSER_C	character varying (text)	Campo información auxiliar (texto)
OBSER_N	character varying (text)	Campo información auxiliar (numérico)
CODBLQ	character varying (text)	Código numérico al bloque de producción del SIOSE al que perte

Scalable Vector Graphics (SVG) Specification[4], además del apoyo de la clasificación de colores que especifica la leyenda del Corine Land Cover.v

Los colores definidos han sido utilizados posteriormente para elaborar y colorear las figuras de la documentación, tanto del trabajo como para la plataforma del proyecto en GitHub, a partir de la función SVG que se ha creado expresamente para ello (véase).

En el segundo conjunto de datos, se han utilizado los datos del SIOSE de dos zonas de estudio distintas para realizar una comprobación de los resultados de las métricas de paisaje a partir de casos reales y comparar ambos paisajes. La estructura de la tabla de atributos se compone de identificadores únicos, geometría, tipos de coberturas, información en XML, superficie, dos campos para almacenar nueva información y códigos numéricos de producción.

[4]http://www.december.com/html/spec/colorsvg.html

Cuadro 2.4 My caption

Nombre	Tipo de campo	Descripción
ID_POLYGON	string	Identificador único de cada polígono
GEOMETRY	geometry	Geometría del polígono
SIOSE_CODE	string	Tipo de cobertura de cada polígono
SIOSE_XML	string	Información completa del tipo de cobertura al que pertenece cada polígono, en formato XML
SUPERF_HA	real	Superficie del polígono en hectáreas
OBSER_C	string	Información auxiliar (texto)
OBSER_N	integer	Información auxiliar (numérico)
CODBLQ	integer	Código numérico al bloque de producción del SIOSE al que pertenece

2.3. Selección de métricas

2.4. Implementación/desarrollo de funciones en PostgreSQL

2.5. Documentación de la extensión

A lo largo del trabajo, se aplican una serie de lenguajes de marcado para la documentación de la extensión. Pero antes se obtienen conocimientos previos sobre ellos y su funcionamiento para utilizarlos durante el proyecto. Los lenguajes utilizados son:

- **Markdown**[5] es un lenguaje ligero capaz de convertir texto plano a lenguaje HTML. Permite una escritura sencilla y conserva un diseño fácil de lectura. Es compatible con muchas plataformas. Este tipo de lenguaje es utilizado para documentar la extensión en la plataforma de GitHub y la descripción de cada una de las métricas de paisaje.

- **TeX**[6] es el lenguaje que se utiliza en el sistema de textos LaTeX y que crea documentos con una alta calidad tipográfica. Se utiliza para escribir artículos o libros científicos, y desde hace tiempo este lenguaje se emplea por un gran número de usuarios. Este tipo de lenguaje es utilizado para redactar este trabajo. Para trabajar con este lenguaje, se utiliza la aplicación Texmaker que se ejecuta desde un contenedor Docker.

- **Scalable Vector Graphics (SVG)** es un lenguaje capaz de crear gráficos basados en vectores escalables a partir de archivos vectoriales en 2D y en formato XML. En esta década muchos de los navegadores web utilizan este tipo de lenguaje para sus gráficos.

[5]https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet
[6]https://www.latex-project.org/

Gracias a este lenguaje, los gráficos no pierden calidad, pueden ser escalables y ocupan menos espacio en la memoria. Este tipo de lenguaje es utilizado para crear las figuras del trabajo.

Por ello, se desarolla una función SQL, como primera propuesta, que dibuja y colorea los gráficos vectoriales que acompañan en el trabajo y en la documentación de la extensión en la plataforma GitHub. Para que las figuras tengan el color correspondiente al tipo de cobertura al que pertenecen, se aplica el código SVG que acompaña a cada polígono en la tabla de atributos del primer conjuntos de datos. La función SVG puede consultarse en el Anexo II.

- **Mermaid**[7] es un lenguaje de secuencia, que utiliza etiquetas similares a las que son empleadas en el lenguaje de marcado, capaz de generar gráficos a partir de texto por medio de JavaScript. Este tipo de lenguaje se ha utilizado para crear los diagramas de secuencia y el diagrama de Gantt.

[7]https://mermaidjs.github.io/

3. RESULTADOS Y DISCUSIÓN

Resultados

Discusión

4. CONCLUSIONES Y TRABAJO FUTURO

Conclusiones

Trabajo

A. HOW TO INSTALL LaTeX

Windows OS

TeXLive package - full version

1. Download the TeXLive ISO (2.2GB) from
   https://www.tug.org/texlive/
2. Download WinCDEmu (if you don't have a virtual drive) from
   http://wincdemu.sysprogs.org/download/
3. To install Windows CD Emulator follow the instructions at
   http://wincdemu.sysprogs.org/tutorials/install/
4. Right click the iso and mount it using the WinCDEmu as shown in
   http://wincdemu.sysprogs.org/tutorials/mount/
5. Open your virtual drive and run setup.pl