SUM-OJS: Scripts para actualizaciones y migraciones de OJS

GONZALO LUJÁN VILLARREAL
PREBI-SEDICI, Universidad Nacional de La Plata
CESGI, Comisión de Investigaciones Científicas
gonzalo@prebi.unlp.edu.ar

PABLO GABRIEL TERRONE
PREBI-SEDICI, Universidad Nacional de La Plata
pabloterrone@sedici.unlp.edu.ar

MARISA RAQUEL DE GIUSTI
PREBI-SEDICI, Universidad Nacional de La Plata
CESGI, Comisión de Investigaciones Científicas
marisa.degiusti@sedici.unlp.edu.ar

EJE TEMÁTICO

Infraestructura tecnológica

RESUMEN

En este trabajo presentamos SUM-OJS, una herramienta desarrollada con el objetivo de acelerar y simplificar las tareas vinculadas a actualizaciones del software Open Journals System (OJS) hacia nuevas versiones, así como también a migraciones hacia nuevos entornos de ejecución. Esta herramienta utiliza una estructura de proyectos basada en Docker y Docker-compose, que permite generar el entorno de ejecución, actualización o migración requerido por cada versión particular de OJS, y una serie de comandos que automatizan muchas de las tareas recurrentes que se deben ejecutar durante las migraciones. Esta ponencia explica la arquitectura interna y forma de

uso de SUM-OJS, e incluye un listado de actualizaciones de OJS realizadas con esta herramienta, junto a los problemas identificados y soluciones aplicadas en cada caso.

PALABRAS CLAVE

Open Journals System, actualización, migración, virtualización.

SUM-OJS: Scripts for updates and migrations of OJS

SECTION

Technological infrastructure

ABSTRACT

In this paper we propose SUM-OJS, a software tool developed to accelerate and simplify common tasks while running upgrades of Open Journals System (OJS), as well as testing migrations towards new environments. This software has been built upon a Docker and Docker-compose project structure, which enables a fast and custom setup of different environments for running, updating or migrating required for each particular OJS version. We have also included a command line tool to run common tasks during migrations. The paper introduces the inner architecture of SUM-OJS and a quick hands-on tutorial, and it also includes a list of upgrades already performed with this tool, with many issues identified and solutions applied in each case.

KEYWORDS

Open Journals System, upgrade, migration, virtualization.

Introducción

Open Journals System (OJS) es un sistema de gestión de publicaciones periódicas de código abierto. Su desarrollo es liderado por el Public Knowledge Project (PKP), y la primera versión fue publicada en el año 2005. Desde entonces, el trabajo mancomunado entre PKP y una creciente comunidad de desarrolladores en todo el mundo ha permitido incorporar cada vez más funciones, optimizar las ya existentes y modernizar las librerías y herramientas sobre las que funciona OJS. Las actualizaciones de OJS se publican de manera periódica, y con cada nueva actualización se incorporan nuevas funcionalidades y se resuelven problemas reportados en versiones anteriores, entre los que se incluyen cuestiones vinculadas a la seguridad, a la performance y uso de recursos, y a fallas en el funcionamiento de determinadas áreas del sistema. En muchas ocasiones, los cambios de versión también requieren realizar cambios en los servicios que utiliza OJS: versión de PHP (por ejemplo de PHP 5.6 a 7.0 y luego a 7.3), motor de almacenamiento en MySQL (por ej. de MyISAM a InnoDB), instalación de librerías adicionales en alguno de los servicios, asignación de memoria, entre otros.

Las actualizaciones de OJS pueden clasificarse en dos categorías: actualizaciones menores, como la actualización desde OJS 3.2.1 hacia OJS 3.2.1-1 y luego hacia 3.2.1-2, que corrigen errores o problemas puntuales, y actualizaciones mayores, por ejemplo desde OJS 3.0 a OJS 3.1 y luego a OJS 3.2, que incorporan un gran número de mejoras y en muchos casos realizan cambios a nivel de arquitectura: cambio de framework MVC, nuevo sistema de gestión de traducciones, cambios en la jerarquía de clases de plugins y/o temas, etc.

Entre las distintas actualizaciones de OJS, puede establecerse un punto de inflexión en el año 2015, con el lanzamiento de OJS 3.0. Esta versión propuso cambios globales en el sistema, que si bien mantenía su esencia en cuanto a flujos de trabajo, roles y servicios esenciales, introducía una interfaz de usuario completamente rediseñada, junto a un nuevo *framework* desarrollado por PKP para todos sus sistemas, y una gran cantidad de opciones de configuración y personalización que otorgaban a OJS mayor flexibilidad para adecuarse a los

requerimientos de las distintas instituciones que lo utilizan. Desde el lanzamiento de OJS 3.0, se han generado más de treinta actualizaciones, cuatro de ellas mayores (3.0, 3.1, 3.2 y 3.3); si bien el lanzamiento de cada nueva versión es coordinado por el propio desarrollador PKP, es importante destacar el rol de un gran número de instituciones académicas, empresas editoriales y usuarios independientes de todo el mundo, quienes realizan sus aportes con código fuente, reportes de errores y traducciones.

Con cada nueva versión, el equipo de PKP empaqueta junto al código fuente una herramienta (*script*) que ejecuta las tareas requeridas para realizar la actualización de la base de datos. Las tareas que realiza este script se encuentran en la incorporación de nuevas opciones de configuración, la creación o modificación de tablas, la modificación de registros tanto por correcciones de errores como por modificaciones en el modelo de datos, el cambio de nombre de archivos y/o directorios, entre otras. Asimismo, antes de ejecutar las tareas de actualización, el script realiza algunos controles vinculados al sistema subyacente, como por ejemplo que la versión de PHP sea adecuada, o que se cuente con los permisos necesarios sobre determinados directorios y archivos.

Una característica interesante de esta herramienta para la actualización es su metodología incremental, que ejecuta los cambios de manera iterativa desde la versión instalada hacia la versión destino. Esto significa que, si una organización posee la versión 3.0 de OJS, y desea pasar a la versión 3.2, la herramienta se encargará de llevar primero desde la versión 3.0 a la versión 3.1, luego de la versión 3.1 a la versión 3.2. En condiciones normales, esta característica resulta muy conveniente, ya que ahorra mucho trabajo, dado que se evita realizar actualizaciones manuales entre las versiones intermedias.

Al momento de realizar una actualización, el proceso puede demorar desde pocos minutos hasta algunas horas. El tamaño de la base de datos, relacionado con la cantidad de revistas, de envíos y de usuarios, así como la capacidad de procesamiento (CPU, memoria, discos), son los factores que más influyen en el tiempo total del proceso de actualización. Una vez finalizado dicho proceso, y en caso de que no se hayan observado errores, deberán revisarse las nuevas

características y funcionalidades de la versión de OJS actualizada, habilitar o re-configurar plugins, y realizar otras tareas de mantenimiento y verificación antes de habilitar el uso o el acceso a los usuarios del sistema. Sin embargo, en caso de detectarse un problema durante la actualización, el script informará al usuario sobre las incidencias encontradas, y queda a criterio de cada usuario decidir la mejor estrategia para corregir el error y volver a intentar la actualización. Este proceso puede requerir numerosas iteraciones, ya que la corrección de un problema puede exponer un segundo problema que antes no era visible. Es difícil determinar el porcentaje de actualizaciones que fallan: en los foros de PKP[1] se observan una cantidad importante de hilos donde se reportan errores de todo tipo, gracias a la participación de la comunidad de usuarios y desarrolladores en la búsqueda de soluciones, que podrían incluso generar cambios en futuras versiones de OJS para corregir los errores detectados.

Para evitar problemas vinculados a la pérdida de información, a la generación de datos inconsistentes o a la imposibilidad de acceder a archivos alojados en el servidor, es conveniente realizar las actualizaciones en un entorno de desarrollo o pruebas, aislado e independiente del servidor en producción. Asimismo, como se mencionó previamente, al identificar un problema en la actualización se deberán realizar las acciones correctivas para resolver dicho problema, para luego volver a intentar la actualización. Sumado a esto, no siempre es posible actualizar directamente hacia la última versión, ya que los errores pueden surgir en alguna etapa intermedia del proceso de actualización. En estos casos, se deben realizar actualizaciones parciales, y realizar correcciones intermedias antes de continuar la actualización. El Gráfico 1 ilustra de manera simplificada este proceso para actualizar desde la versión X a la Z, pasando por la versión intermedia Y.

Luego de realizar muchas actualizaciones entre versiones de OJS, los autores de este trabajo hemos buscado formas de automatizar algunas de las tareas recurrentes, minimizar la tasa de errores y organizar todas las

[1] Hilos en los foros de PKP vinculados problemas de actualización de OJS: https://forum.pkp.sfu.ca/tags/c/questions/5/upgrade

tecnologías (en sus diferentes versiones) requeridas. A partir de aquí surge SUM OJS, una herramienta que facilita la repetición de este proceso de actualización de OJS a partir de diferentes estrategias, como la definición de contextos de ejecución particulares, replicación de entornos para pruebas y automatización de tareas de corrección de errores. Como se indica en el apartado “Casos de uso en contextos reales”, esta herramienta ha sido puesta a prueba en numerosas instalaciones de OJS, que fueron actualizadas a las diferentes versiones disponibles al momento de realizar la actualización, con resultados satisfactorios.

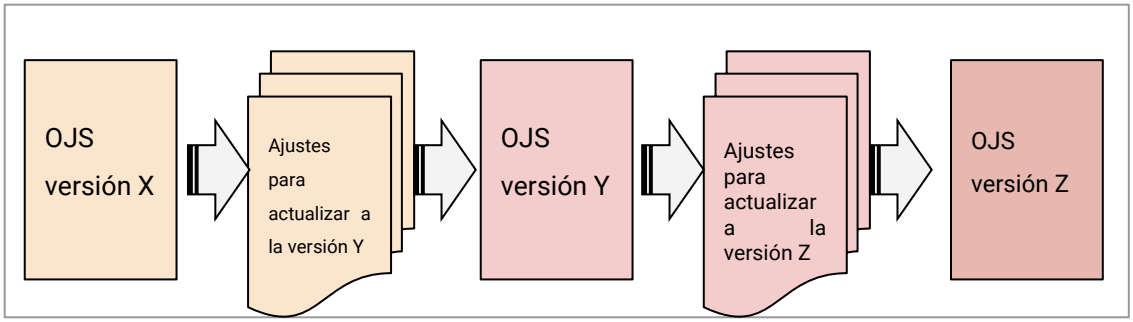

GRÁFICO 1. Etapas en la actualización de OJS desde la versión X a la versión Z (X < Y < Z) (autoría propia)

Desarrollo

Como se mencionó previamente, el proceso de actualización puede demorar mucho tiempo si se trata de grandes instalaciones de OJS. Este proceso se realiza de manera iterativa entre versión y versión, y cada intento de actualización requerirá, como mínimo:

1. Preparar el servidor para ejecutar la actualización (servidor web, versión de PHP, servidor de base de datos).
2. Cargar una base de datos con la versión de OJS en curso.
3. Copiar los archivos subidos por los usuarios del OJS en curso.
4. Realizar los ajustes necesarios para poder pasar a la siguiente versión.
5. Ejecutar la actualización hacia la siguiente versión.

6. **a.** Si la actualización del paso previo falló, revisar la causa, preparar las correcciones para realizar un nuevo intento y volver al paso 1. **b.** Si la actualización del paso previo funcionó, continuar con la actualización hacia la siguiente versión.

Si bien este proceso describe a muy alto nivel las distintas tareas involucradas en cada etapa, cabe destacar que algunas tareas pueden demorar mucho tiempo por su complejidad o volumen de información, y pueden introducir nuevos errores, en especial en los casos que requieran configuraciones específicas en alguno de los servicios del sistema (PHP, servidor web o servidor de base de datos).

Para agilizar este proceso iterativo, hemos implementado un *stack* de ejecución basado en *containers dockers*, junto a unos *scripts bash* y Makefile. El uso de los contenedores permite automatizar o agilizar algunas de las tareas descriptas previamente. A continuación, se brinda una breve introducción a estas tecnologías, y en la siguiente sección se describen los componentes de este *stack*, indicando el rol de cada uno de ellos en proceso de ejecución de actualizaciones y migraciones.

Virtualización basada en contenedores Docker

Docker es una herramienta cada vez más utilizada por la comunidad científica, gracias a su capacidad de ejecutar aplicaciones auto-empaquetadas en entornos aislados[2]. Una de las ventajas de Docker radica en su portabilidad, que le permite distribuir una misma aplicación en un amplio rango de plataformas (Di Tommaso, 2015).

A diferencia de la forma de virtualización completa, en la cual se virtualiza el hardware y se levanta un sistema operativo con todas sus funcionalidades para ejecutar el servicio particular, Docker realiza una virtualización mucho más

[2] Docker: https://www.docker.com/

liviana ya que no virtualiza el hardware y utiliza el núcleo del sistema operativo del host dentro de los contenedores.

Una de las ventajas más evidentes a la hora de utilizar Docker es la habilidad de descargar imágenes pre-construidas que contienen paquetes de software y sus dependencias listos para usarse, evitando así su instalación y configuración manual por parte de los usuarios. Otra ventaja no menos importante es que se ejecuta cada proceso en un contenedor aislado, lo cual evita problemas con otros servicios y asegura que se ejecutará en una configuración predecible, que no cambiará a lo largo del tiempo debido a actualizaciones ni errores de configuración del software.

Organización de servicios con Docker Compose

Cuando se desea ejecutar muchos servicios para crear un entorno de desarrollo, resulta fundamental poder organizarlos. Docker Compose (IBRAHIM, 2021) permite, en un solo archivo, definir un conjunto de servicios que trabajan de forma colaborativa en un entorno aislado. Para esto permite definir aspectos como:

- Mapeo de puertos: cada servicio puede mapear los puertos virtuales que utilizará con los puertos físicos.
- Establecer dependencias: permite indicar el orden de ejecución en los servicios, lo que es especialmente útil cuando un servicio depende de otro para poder funcionar.
- Mapeo de volúmenes: permite definir volúmenes para montar dentro de containers, en el contexto de este trabajo se aprovecha la posibilidad de poder mapear volúmenes con directorios del sistema operativo host.
- Utilización de variables de entorno: muchos servicios necesitan ciertos datos para poder ejecutarse (host, nombre de usuario, contraseña, etc), Docker Compose permite que estos datos se recuperen desde un archivo externo, por lo que resulta fácil cambiar alguno de los datos.

Bash y Makefile

Cuando se trabaja con procesos iterativos, que poseen muchas instrucciones que pueden repetirse más de una vez, es conveniente diseñar mecanismos para automatizar estos procesos. Hacer estas repeticiones de forma manual (copiando y pegando las instrucciones) puede ser problemático ya que algunas veces pueden cometerse errores sintácticos u omitirse alguna instrucción (BAKER, 2020*)*. Sumado a esto, a medida que aumenta la cantidad de instrucciones a ejecutarse, su ejecución manual requerirá cada vez más tiempo. Para evitar estos inconvenientes en este proyecto utilizamos las tecnologías Bash y GNU Make.

Bash, un lenguaje de *scripting*, muy utilizado para la automatización de tareas en entornos GNU/Linux, posee una amplia variedad de instrucciones altamente parametrizables que pueden agruparse en un único archivo (BASH, 2022), lo cual resulta muy útil lograr procesos de migración mucho más rápido y sencillo.

GNU Make utiliza archivos Makefile, como punto de entrada a tareas complejas, agrupando conjuntos de instrucciones o *scripts* en una sola instrucción *make* (MAKE, 2022). En nuestro caso, lo utilizamos para simplificar la invocación a instrucciones de Docker y a los scripts de Bash, generando así código más fácil de escribir, mantener, resultando así en programas más fáciles de utilizar.

A modo de ejemplo, tomamos uno de los comandos incluidos en el Makefile, correspondiente con *make regenerate*. Dicho comando invoca al *script bash upgrader,* al cual le envía los parámetros correspondientes que incluyen la operación a realizar (parámetro -o), la versión desde la que se parte (parámetro -f) y la versión a la que se desea actualizar (parámetro -t). Todo esto se resume en la siguiente línea:

```
./upgrader -o regenerate -f $(from) -t $(to)
```

Internamente, el *upgrader* ejecuta diversas instrucciones para hacer lo siguiente:

1. Elimina la base de datos actual.
2. Crea una nueva base de datos vacía.
3. Carga el backup de la base de datos.
4. Prepara la base de datos para la migración, ejecutando las consultas del post_dump_file.

De esta manera el proceso de migración es mucho más sencillo, se resume en levantar la base de datos (*Make up*), regenerarla (*Make regenerate*) y ejecutar la actualización (*Make upgrade*).

Descripción del *stack*

Como se describió previamente, basados en la experiencia personal, la actualización entre versiones de OJS es un proceso iterativo, que requiere realizar pruebas y ajustes entre las pruebas a fin de alcanzar una actualización *limpia:* sin errores ni durante el proceso de actualización, ni durante las pruebas posteriores a la actualización. Estos ajustes y correcciones pueden clasificarse en dos grandes categorías:

- ***Ajustes a nivel de servidor***: aquí se agrupan los cambios entre versiones de PHP, en la asignación de límites de memoria, configuración de encabezados HTTP en el servidor web, cambios en el tamaño de los buffers del servidor de bases de datos, entre otros.
- ***Ajustes a nivel de base de datos***: involucra todas las correcciones que se hacen sobre la base de datos de OJS, lo que abarca la eliminación de valores nulos, la corrección de referencias incorrectas, la corrección de problemas de codificación de caracteres, entre otros.

Para agilizar la aplicación de ajustes a nivel del servidor, se decidió por una solución basada en contenedores Docker, organizados en una serie de pilas de

ejecución implementadas sobre Docker Compose[3] que incluyen al servidor de bases de datos, al servidor web con PHP, y al software phpMyAdmin para realizar tareas de verificación de datos, búsqueda de errores, etc. La elección de la tecnología de contenerización basada en Docker asegura un modelo de ejecución de *containers* flexible y altamente configurable, que permite emular entornos de desarrollo y ejecución específicos en poco tiempo. Además, se trata de herramientas muy utilizadas, con una gran comunidad de usuarios y una amplia gama de servicios disponibles para los desarrolladores: repositorios de imágenes, documentación, servicios de monitoreo, etc.

A continuación, se muestra un extracto de uno de los archivos docker-compose, en el que se muestran los principales elementos de la configuración:

```
version: '3.7'
services:
  mysql:
      image: mysql:5
      container_name: ${PROJECT_NAME}_mysql
      volumes:
      - ./sql:/var/backups
      environment:
      MYSQL_DATABASE: ${PROJECT_NAME}
  phpmyadmin:
      depends_on:
      - mysql
      image: phpmyadmin/phpmyadmin
      container_name: ${PROJECT_NAME}_phpmyadmin
      environment:
```

[3] Compose specifications, Docker documentation: https://docs.docker.com/compose/compose-file/

```
    ports:
    - '8080:80'
  web:
   #for OJS 3.2
    image: webdevops/php-apache-dev:7.3
    container_name: ${PROJECT_NAME}_web
    volumes:
    - "./public_ojs:/app"
    - "../private_ojs:/var/ojs-data/"
    ports:
    - "80:80"
    environment:
    PHP_MEMORY_LIMIT: 512M
    PHP_DISPLAY_ERRORS: 1
```

EXTRACTO DE CÓDIGO 1. Configuración de servicios de Docker Compose (autoría propia)

La configuración de Docker Compose describe tres servicios: MySQL (motor de base de datos), phpMyAdmin (gestión gráfico de la base de datos) y web (servidor Apache y php). El servicio MySQL está basado en la versión 5, y utiliza un volumen externo (directorio ./sql). El servicio phpMyAdmin se conecta con el servicio MySQL y queda accesible a través del puerto 8080 del host. El servicio web utiliza el servidor web Apache, y el intérprete de PHP versión 7.3; este servicio también utiliza volúmenes externos, uno donde se encuentra el código fuente de OJS (public_ojs) y otro donde se encuentran los archivos subidos por los usuarios (private_ojs). Adicionalmente, el servicio web expone el acceso a OJS a través del puerto 80 del host, e incluye algunas opciones de configuración específicas (512 MB de límite de memoria, y mostrar los errores de php).

Estructura de directorios

La solución propuesta utiliza una estructura de directorios que contiene por un lado los archivos generales del proyecto y, por otro, subdirectorios con archivos particulares para migrar entre distintas versiones de OJS. A continuación, se incluye un extracto de la estructura de directorios propuesta:

```
── .env
├── from3.1-to-3.2
│   ├── config.inc.php
│   ├── docker-compose.yml
│   ├── ojs-3.2.1-3.tar.gz
│   ├── public_ojs
│   └── sql
│     ├── load-data.sql
│     └── post-load-data.sql
├── from3.2-to-3.3
│   ├── docker-compose.yml
│   ├── config.inc.php
│   ├── ojs-3.3.0-9.tar.gz
│   ├── public_ojs
│   └── sql
│     ├── load-data.sql
│     └── post-load-data.sql
├── Makefile
├── private_ojs
├── README.md
└── upgrader
```

LISTADO 1. Estructura de directorios de SUM-OJS (autoría propia)

En esta estructura se observa:

- Un directorio 'private_ojs', que se utiliza para alojar los archivos subidos por los usuarios de OJS.
- Dos directorios ('from3.1-to-3.2' y 'from3.2-to-3.3') que poseen los archivos y directorios necesarios para realizar migraciones entre las versiones 3.1 a 3.2 y 3.2 a 3.3 respectivamente. Dentro de estos directorios se destacan los siguientes objetos:
  - Un archivo 'docker-compose.yml' con la pila de Docker adecuada para la versión de OJS **hacia** la que se está migrando.
  - Un archivo 'config.inc.php', que posee las configuraciones básicas de OJS para la versión **hacia** la que se está migrando.
  - Un directorio SQL, que posee dos archivos:
    - 'load-data.sql', que posee una copia completa de la base de datos del OJS **desde** el que se está migrando;
    - 'post-load-data.sql', que posee el código SQL para ejecutarse sobre la base de datos previa a migrar.
  - Un directorio 'public_ojs', donde debería incluirse el sistema OJS en la versión **hacia** la que se está migrando.
- Un archivo ejecutable llamado *upgrader*, que incluye varias rutinas para utilizar los distintos servicios Docker.
- Un archivo *Makefile*, que ofrece una interfaz de línea de comandos de alto nivel para utilizar las rutinas del programa *upgrader*.
- Un archivo '.env' que posee variables de entorno globales (nombre de base de datos, nombres de archivos utilizados en las distintas etapas, usuario y clave de conexión a la base de datos, etc.)

Modo de uso

Para utilizar este desarrollo, se requiere realizar una configuración inicial por única vez, en la que se preparará todo el entorno para realizar las pruebas de migraciones.

Aspectos a tener en cuenta durante la configuración inicial global

El directorio 'private_ojs' debe contener todos los archivos privados de OJS. Se deberán brindar permisos de lectura y escritura de manera recursiva a dicho directorio, para que las migraciones puedan renombrar y mover archivos entre directorios. Además, el archivo '.env' debe contener las variables de entorno con las que se realizará la migración. Estas variables de entorno son:

#Nombre del proyecto, utilizado para crear contenedores docker. Por ejemplo: PROJECT_NAME=ojs_migrate

#Nombre de la base de datos sobre la que se realizarán las migraciones. Por ejemplo: DATABASE_NAME=ojs_db

#Contraseña del usuario root, necesaria para regenerar la base de datos cuando se solicita. Por ejemplo: ROOT_PASSWORD=root

#Nombre del archivo SQL que contiene la base de datos a importar cada vez que se regenere. Por ejemplo: DUMP_FILE=load-data.sql

#Archivo SQL que contiene otros comandos SQL a ejecutar luego de regenerar la base de datos. Por ejemplo: POST_DUMP_FILE=post-load-data-sql

#Directorio dentro de los contenedores donde se alojarán los archivos SQL. Por ejemplo: DUMP_DIR=/var/backups

EXTRACTO DE CÓDIGO 2. Configuración de variables de entorno de SUM-OJS (autoría propia)

Aspectos a tener en cuenta para configurar y ejecutar una actualización

Para realizar una actualización desde la versión X a la versión Y de OJS, se debe localizar el directorio 'fromX-to-Y'. Por ejemplo, para realizar la

actualización desde la versión 3.1 (X) hacia la versión 3.2 (Y), deberá utilizarse el directorio 'from3.1-to-3.2'. El proyecto posee directorios para realizar actualizaciones entre dos versiones (3.1 a 3.2, y 3.2 a 3.3), pero el esquema propuesto puede extenderse fácilmente para realizar actualizaciones entre cualquier par de versiones, tanto anteriores como posteriores a las ya incluidas.

Antes de realizar la actualización, este directorio debe contener los archivos listos para generar la base de datos y ejecutar la actualización. Para su correcta ejecución, debe:

- Copiar en el directorio from-X-to-Y/sql el archivo con el dump completo de la base de datos del OJS versión X. El nombre del archivo debe ser el especificado previamente en la variable DUMP_FILE del archivo de configuración global .env.
- En caso de ser necesario, se deberá copiar en el directorio from-X-to-Y/sql el archivo que contenga los comandos SQL a ejecutarse luego de importar la base de datos original, y antes de ejecutar la actualización hacia la versión Y. El nombre de este archivo debe ser el especificado previamente en la variable POST_DUMP_FILE.
- Asegurarse de que el *stack* definido en el archivo ''from-X-to-Y/docker-compose.yml' cumple con los requerimientos para ejecutar OJS versión Y. En particular, debe observarse que la versión de PHP sea adecuada en la línea "image: webdevops/php-apache-dev:PHP_VERSION". Por ejemplo, para ejecutar una actualización hacia OJS 3.2, puede utilizarse PHP 7.3, especificada de la siguiente forma: image: webdevops/php-apache-dev:7.3.
- Descomprimir en el directorio 'from-X-to-Y/public_ojs' una copia de OJS versión Y.
- Asignar permisos de lectura y escritura sobre el directorio de cache de OJS: chmod a+rw from-X-to-Y/public_ojs/cache –R.
- Generar un archivo de configuración para OJS versión Y (puede realizarse a partir del archivo TEMPLATE incluido en OJS). Este archivo debe llamarse 'config.inc.php', debe estar alojado dentro del directorio 'public_ojs', y deberá incluir la configuración necesaria para que OJS versión Y se ejecute desde el contenedor Docker. A continuación, se

incluyen algunas de las opciones de configuración más relevantes que deberán tenerse en cuenta:

```
base_url = "http://localhost"
You might also need to set override settings, such as:
base_url[SOME_JOURNAL] = http://localhost/SOME_JOURNAL

[database]

driver = mysqli
host =  mysql
username = root
password = <--- same as password set in .env
name = <--- same as database set in .env

[files]
files_dir = /var/ojs-data/uploads
public_files_dir = public
```

EXTRACTO DE CÓDIGO 3. Configuración de la instalación de OJS (autoría propia)

Una vez realizado esto, se cuenta con un entorno de ejecución de OJS versión Y, y con una copia de la base de datos de OJS versión X. Ahora es hora de utilizar los scripts de automatización de tareas para regenerar la base de datos en versión X, así como también para ejecutar las migraciones.

El formato de los comandos es el siguiente:

make *operación* **from**=X **to**=Y.

Las operaciones disponibles son:

- ***up***: inicia los contenedores docker para migrar OJS desde la versión X hacia la versión Y.
- ***down***: detiene los contenedores docker levantados previamente con up.
- ***regenerate***: regenera la base de datos de la versión X, y ejecuta los scripts SQL previos a la actualización hacia la versión Y.
- ***upgrade***: ejecuta los scripts de actualización hacia la versión Y.

Por ejemplo, para realizar la migración desde OJS 3.1 a OJS 3.2, los comandos a ejecutar serán:

```
make up from=3.1 to=3.2
make regenerate from=3.1 to=3.2
make upgrade from=3.1 to=3.2
make down from=3.1 to=3.2
```

En caso de que la migración falle, será necesario volver a generar la base de datos (utilizando el comando make regenerate from=X to=Y) . Esto eliminará la base de datos actual y la volverá a cargar a partir del backup. Una vez que se restauró, se deberán realizar las correcciones que provocaron el fallo en el intento anterior. Si estas correcciones requieren modificar la base de datos (por ejemplo: eliminar nulls, revisar codificación, etc.), estos comandos deberían incorporarse al archivo POST_DUMP_FILE a fin de automatizar su ejecución en futuras pruebas. Por el otro lado, si las correcciones implican cambios a nivel de la infraestructura (versión de PHP, límites de memoria, configuraciones de MySQL), entonces los cambios deberán aplicarse sobre el archivo 'docker-compose'.

Luego, realizadas estas correcciones, se puede volver a intentar la actualización ejecutando la misma secuencia de comandos. Una vez que la

actualización ha sido exitosa, ya se contará con OJS ver. 3.2 actualizado, y se podrá actualizar hacia la siguiente versión. Sin embargo, en este punto sería deseable realizar una copia de seguridad de la base de datos, ya que, en caso de que la actualización hacia la versión 3.3 falle, podrá volver a intentarse desde la versión 3.2, en vez de volver a intentarlo desde la versión 3.1.

Para realizar una copia de seguridad, puede utilizarse el servicio phpMyAdmin (incluido en 'docker-compose') a través de un navegador web en la dirección (http://localhost:8080), o también puede realizarse desde la línea de comandos, lo que puede resultar más eficiente en muchos casos. Para ello, se ha incluido un comando que realiza estas copias de seguridad y las almacena en el directorio DUMP_DIR, junto a otros comandos que permiten iniciar líneas de comandos en los contenedores MySQL y WEB, a fin de ejecutar operaciones rápidas desde la línea de comandos.

- **make backup**: realiza un backup en el directorio DUMP_DIR.
- **make bash-mysql**: inicia una terminal dentro del contenedor mysql.
- **make bash-php**: inicia una terminal dentro del contenedor web.

Casos de uso en contextos reales

Para verificar el funcionamiento de esta herramienta se realizaron migraciones de distintas instalaciones de OJS. La siguiente tabla resume en orden cronológico (desde mediados de 2019 a principios de 2022) los trabajos de actualización realizados:

REVISTA	DESCRIPCIÓN DE LA INSTALACIÓN	VERSIÓN ORIGEN	VERSIÓN DESTINO
Ameghiniana https://www.ameghiniana.org.ar/	Instalación de OJS con una única revista. Idioma inglés	2.4.8-3	3.2.1-1
PE-APA https://www.peapaleontologica.org.ar/	Instalación de OJS con una única revista. Idiomas inglés y español	2.4.8-3	3.2.1-1
Revista de la Asociación Geológica Argentina https://revista.geologica.org.ar	Instalación de OJS con una única revista. Idioma español	2.3.6-0	3.2.1-1
Revistas de la Sociedad Argentina para el Estudio de los Mamíferos https://ojs.sarem.org.ar/	Instalación de OJS con dos revistas. Idiomas inglés y español	2.4.8-1	3.3.0-6
Revista de Geología Aplicada a la Ingeniería y al Ambiente https://www.editoresasagai.org.ar/ojs/	Instalación de OJS con una única revista. Idioma español	2.4.8.1	3.3.0-7
PE-APA (segunda actualización) https://www.peapaleontologica.org.ar/	Instalación de OJS con una única revista. Idiomas inglés y español	3.2.1-1	3.3.0-8
Portal de Revistas de la UNLP https://revistas.unlp.edu.ar/index	Instalación de OJS con múltiples revistas. Múltiples idiomas	3.1.2-1	3.3.0-9

TABLA 1. Actualizaciones a instalaciones de OJS utilizando SUM-OJS (autoría propia)

En líneas generales, el proceso de migración se realizó de la siguiente manera:

- Para instalaciones previas a OJS 2.4, se realizó una actualización hacia OJS 2.4.8-4. Para ello, se generó un *stack* con PHP 5.6 y MySQL 5.5. Cabe destacar que se utilizó la librería PDO para conectar al servicio MySQL desde PHP.
- Para instalaciones 2.4.x, se realizó primero una actualización hacia OJS 3.0. Para ello, se generó un *stack* con PHP 7.0 y MySQL 5.5. Aquí también se utilizó PDO.

- Para instalaciones 3.0.x, se realizó la actualización hacia 3.1. Para ello se generó un *stack* con PHP 7.2 y MySQL versión 8 (o MariaDB versión 10 o superior). A partir de aquí se comienza a utilizar el conector mysqli.
- Para instalaciones 3.1, se realizaron actualizaciones a OJS 3.2. Para ello se generó un stack con PHP 7.3 ó 7.4 (indistintamente), con MySQL 8/MariaDB 10 y mysqli.
- Para instalaciones 3.2 , se realizaron actualizaciones hacia OJS 3.3. Para ello se generó un stack con PHP 7.4 , con MySQL 8/MariaDB 10 y mysqli.

Como puede observarse, las actualizaciones entre distintas versiones requieren configuraciones y combinaciones de servicios particulares, lo cual fue posible de realizar de manera muy ágil gracias a esta propuesta basada en Docker y Docker-compose. Asimismo, además de los distintos *stacks* generados para cada versión de OJS, es importante destacar que cada actualización requirió sus propias tareas para la corrección de errores, limpieza de datos, ajustes de configuraciones, etc. A lo largo de las distintas actualizaciones, las necesidades particulares de cada caso requirieron ajustes en la herramienta aquí presentada, en particular orientadas a automatizar tareas recurrentes (como por ejemplo regeneración de la base de datos, acceso a la línea de comandos del servicio web o creación de backups).

Si bien existen muchos motivos por los cuales una actualización de OJS puede fallar, luego de aplicar el proceso de actualización en distintos contextos se identificaron algunos factores que determinan la cantidad de “trabajo extra” (ajustes, limpieza de datos, entre otros.) que será necesario para obtener una actualización exitosa:

1. **Volumen de información**: se observó que las instalaciones de OJS con mayor volumen de información resultaron más problemáticas. Los casos las instalaciones *Ameghiniana* (65 GB de datos) y Portal de Revistas de la UNLP (60 GB), son los más representativos aquí.
2. **Antigüedad de la instalación**: el uso de versiones muy antiguas de PHP y MySQL requiere dedicar un esfuerzo adicional, ya que surgen problemas vinculados a sistemas de codificación (UTF8, Latin 8859, etc.) y a

librerías particulares (por ejemplo: el uso de PDO o MySQLi como librería de conexión entre PHP y MySQL). Todas las actualizaciones desde OJS 2.x hacia OJS 3.X presentaron problemas.

3. **Historial de actualizaciones**: las instalaciones que ya superaron varios procesos de actualización a lo largo del tiempo pueden presentar problemas de inconsistencia en los datos (por ejemplo: una revisión sin su revisor, una descarga de archivo sin el correspondiente archivo, etc.). Este tipo de problemas se presentó principalmente en la instalación del Portal de Revistas de la UNLP, que desde su lanzamiento en el año 2008 ya superó muchas actualizaciones (VILLARREAL, 2013, 2014, 2017; PERCIVALE, 2015).

Conclusiones

A lo largo de este trabajo presentamos una herramienta que permite automatizar algunas de las tareas requeridas para realizar procesos de actualización del software OJS. El uso de esta herramienta reduce la complejidad del proceso iterativo de actualización y migración; destacamos la incorporación de las tecnologías de código abierto Docker, Docker-compose, GNU Make y Bash, como herramientas para abstraer y simplificar la cantidad de instrucciones que requiere este proceso, y de emular los distintos entornos de ejecución requeridos para las diferentes versiones de OJS. Si bien esta propuesta no automatiza por completo la actualización de OJS, ya que se encontrarán errores particulares de cada instalación de OJS que deben ser revisados, para luego generar las consultas necesarias que los resuelvan, se reducen notablemente las tareas de reproducción de los procesos de actualización y se minimizan los errores introducidos al evitar la escritura de comandos complejos en cada etapa del proceso.

Asimismo, resulta evidente que es posible llevar esta herramienta a un mayor grado de automatización. Como futuras líneas de investigación y desarrollo, sería muy interesante incorporar a SUM OJS la posibilidad de ejecutar actualizaciones entre versiones muy alejadas de manera desatendida, es decir, poder actualizar por ejemplo de la versión 2.4 a la versión 3.3 con un solo

comando, que internamente ejecutará cada uno de los comandos propuestos aquí para restaurar la base de datos, ejecutar la migración, generar backups y volver a repetir con la siguiente versión de OJS. Es claro que esto requerirá un mayor esfuerzo de configuración por parte del usuario, ya que se deberá generar un entorno virtual para cada versión de OJS a la que se intentará migrar, pero está claro que una vez generados dichos entornos, el proceso podría ejecutarse de manera íntegra en un sólo comando.

Todas las herramientas descritas en este documento están disponibles bajo licencias de código abierto. El código fuente del proyecto se encuentra accesible desde el repositorio GitHub: https://github.com/sedici/script-ojs

Referencias

BAKER, P. (2020). Using GNU Make to Manage the Workflow of Data Analysis Projects. *Journal of Statistical Software*, *94*, 1-46. https://doi.org/10.18637/jss.v094.c01

BROBERG, M. (2019) The birth of the Bash shell. OpenSource.com .https://opensource.com/19/9/command-line-heroes-bash

DI TOMMASO, P., PALUMBO, E., CHATZOU, M., PRIETO, P., HEUER, M. L., & NOTREDAME, C. (2015). The impact of Docker containers on the performance of genomic pipelines. *PeerJ*, *3*, e1273. https://doi.org/10.7717/peerj.1273

DOWNLOAD | PUBLIC KNOWLEDGE PROJECT. https://pkp.sfu.ca/ojs/ojs_download/

GNU MAKE. https://www.gnu.org/software/make/manual/make.html

IBRAHIM, M. H., SAYAGH, M. & HASSAN, A. E. (2021). A study of how Docker Compose is used to compose multi-component systems. *Empir Software Eng 26*, 128. https://doi.org/10.1007/s10664-021-10025-1

PERCIVALE, B. (2015). Actualización del Portal de Revistas de la UNLP: optimizado para móviles. [Blog] 6 de noviembre de 2015 https://blog.sedici.unlp.edu.ar/2015/11/06/actualizacion-del-portal-de-revistas-de-la-unlp-optimizada-para-moviles/

TAGS. PKP/OJS https://github.com/pkp/ojs/tags

VILLARREAL, G. L. (2013). Actualización del Portal de Revistas de la UNLP. [Blog] 8 de marzo de 2013. https://blog.sedici.unlp.edu.ar/2013/03/08/actualizacion-del-portal-de-revistas-de-la-unlp/

VILLARREAL, G. L. (2014). Nuevas características de OJS 2.4.5. [Blog] 28 de octubre de 2014. https://blog.sedici.unlp.edu.ar/2014/10/28/nuevas-caracteristicas-de-ojs-2-4-5/

VILLARREAL, G. L. (2017). Actualización a OJS 3 del Portal de Revistas de la UNLP. [Blog] 7 de febrero de 2017. https://blog.sedici.unlp.edu.ar/2017/02/07/actualizacion-a-ojs-3-del-portal-de-revistas-de-la-unlp/