Archivos de las aplicaciones. Consideraciones generales.

Dirección General de Sistemas de Información y Comunicaciones

Áreas de Gobierno Tecnológico de SSII y del Dato

Servicio Andaluz de Salud
Consejería de Sanidad, Presidencia y Emergencias

Contenido

Gestión de entregas

Versión: v03r04

Estado: ACTIVO

Entrada en vigor desde: 05 Nov 2025

Obligado cumplimiento desde: 05 Nov 2025

Enlaces relacionados: Gestión de entregas | Archivos de las aplicaciones. Consideraciones generales.

Cumplimiento normativo

Las normas expuestas son de obligado cumplimiento. La STIC podrá estudiar los casos excepcionales los cuales serán gestionados a través de los responsables del proyecto correspondiente y autorizados por el Área de Gobernanza de la STIC. Asimismo cualquier aspecto no recogido en estas normas deberá regirse en primera instancia por las guías técnicas correspondientes al esquema nacional de seguridad y esquema nacional de interoperabilidad según correspondencia y en su defecto a los marcos normativos y de desarrollo software establecidos por la Junta de Andalucía, debiendo ser puesto de manifiesto ante la STIC.

La STIC se reserva el derecho a la modificación de la norma sin previo aviso, tras lo cual, notificará del cambio a los actores implicados para su adopción inmediata según la planificación de cada proyecto.

En el caso de que algún actor considere conveniente y/o necesario el incumplimiento de alguna de las normas y /o recomendaciones, deberá aportar previamente la correspondiente justificación fehaciente documentada de la solución alternativa propuesta, así como toda aquella documentación que le sea requerida por la STIC para proceder a su validación técnica.

Contacto Dpto: Oficina de Calidad

Introducción

Este artículo pretende recoger algunas de las pautas a tener en cuenta a la hora de configurar correctamente los proyectos.

Una de las cuestiones es la estandarización de los nombres. Se deberá tener presente que:

- Toda una aplicación debería de seguir la misma nomenclatura al nombrar los componentes del mismo (sobre todo si comparte tecnología)
- El nombre corto ya no existe, ahora ese campo es "Código ALM" y se debe corresponder con:
    - "Código ALM" == "Componente JIRA" == "Job Jenkins" == "Repositorio Gitlab"... etc
    - El "Código ALM" en CMS y el "Código aplicación" en CMS, NO siempre tienen que coincidir.
- La nomenclatura del "Código ALM" debe ser descriptiva y debe basarse en la convención de nombres de la tecnología del proyecto. Por ejemplo:
    - Java, Node: Todo minúsculas y guiones medios.
    - .Net: camelCase sin guiones.
    - Android/iOS: camelCase sin guiones

> **Modelo aplicación-componente**
>
> En el caso de que se esté siguiendo el modelo aplicación-componente, las configuraciones que sean necesarias se aplicarán a cada uno de los componentes que forman la aplicación.

Aplicaciones Java

Proyectos ANT

build.xml

Actualmente todos los proyectos Java que comiencen su desarrollo, deben estar *"mavenizados*". No obstante, algunos proyectos en mantenimiento, utilizan Ant para la compilación y la construcción.

En estos casos, los campos/etiquetas que son revisables deben cumplir, en el caso del archivo build.xml:

- Project name: <name>$código ALM del aplicativo en la CMS$</name>
- Property name: <*property name="version"*> ***mayor.menor.revisión.entrega*** <*/property*>

A tener en cuenta: hay que incluir las librerías necesarias para la compilación dentro del repositorio, en el subdirectorio auxLibs, y modificar los ficheros de configuración (build.xml) de la compilación para que se use esta ruta.

Proyectos Maven

pom.xml

El principal archivo para este tipo de proyectos es el pom.xml. A continuación se listan las diferentes casuísticas que se pueden presentar.

Si se trata del pom.xml padre:

- <groupId>es.ja.csalud.sas.$path de la aplicación$</groupId> (en minúsculas) donde path == Ambito.NombreSuitedeAplicaciones.CodALMAplicación

  Ejemplo: <groupId>es.ja.csalud.sas.asistencial.sigilum.sigilum_milenium</groupId>

- <artifactId>$código ALM del aplicativo en la CMS$</artifactId> (en minúsculas)

  Ejemplo: <artifactId>ptwanda</artifactId>

- El campo versión consta de 4 dígitos. Los tres primeros corresponden al versión propiamente dicha y el cuatro al número de entrega (son números enteros no negativos, *y* ***NO DEBEN ser precedidos de ceros***).

  <version>mayor.menor.revisión.entrega</versión>
  Ejemplo: <version>1.1.0.4</version>

- <name>$código del aplicativo en la CMS$</name>

  Ejemplo: <name>ptwanda</name>

Si es un proyecto multimódulos, los archivos pom.xml hijo deberán cumplir:

- <artifactId>$código ALM del aplicativo en la CMS$- modulo</artifactId> (en minúsculas)

  Ejemplo: <artifactId>ptwanda-web</artifactId>

- <name>$código ALM del aplicativo en la CMS$-modulo</name>

  Ejemplo: <name>ptwanda-web</name>

Además, se debe de verificar que dentro de la etiqueta <parent> vienen definidas las etiquetas referentes al pom padre tal como se muestra en el siguiente ejemplo:

pom.xml

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.
xsd">
                <parent>
                                <groupId>es.ja.csalud.sas.admonelec</groupId>
                                <artifactId>ptwanda</artifactId>
                                <version>1.1.0.4</version>
                </parent>
                <artifactId>ptwanda-web</artifactId>
                <packaging>war</packaging>
                <name>ptwanda-web</name>
```

Modelo aplicación-componente

Tal y como comentábamos al inicio de este artículo, la convención establecida se aplicará a cada uno de los componentes.

Lo único a tener cuenta es que, en estos casos, el *groupId* llegará hasta el componente:

- <groupId>es.ja.csalud.sas.$path del componente$</groupId> (en minúsculas) donde path == Ambito.NombreSuitedeAplicaciones.CodALMAplicación.CodALMComponente.

- <artifactId>$código ALM del componente en la CMS$</artifactId> (en minúsculas)

Si es un proyecto multimódulos, los archivos pom.xml hijo deberán cumplir:

- <artifactId>$código ALM del componente en la CMS$- modulo</artifactId> (en minúsculas)

- <name>$código ALM del componente en la CMS$-modulo</name>

Por otro lado, y con carácter general, se debe tener en cuenta que:

- **El *pom.xml* no contendrá configuración de profiles** ya que los perfiles serán gestionados por CTI y serán obtenidos de forma automática en el momento de la compilación.
- **El *pom.xml* no contendrá configuración referente a los repositorios de librerías** ya que el único repositorio válido es el del SAS el cual está configurado a nivel de Maven como *mirror* haciendo inutilizables los declarados en el *pom.xml*.

Para el correcto funcionamiento de Sonarqube

Actualmente, se está utilizando el plugin de JaCoCo (Java Code Coverage) para la obtención de la cobertura de los tests. Con el objetivo de que funcione correctamente, es necesario seguir la siguiente pautas en el pom.xml.

- Quitar todas las referencias a Jacoco de los ficheros pom.xml del proyecto. Ni versiones antiguas ni ninguna otra referencia.
- Quitar la opción del pom.xml, dentro del plugin de maven-surefire-plugins:<argLine>-XX:MaxPermSize=256m -Xmx1024M</argLine>
- Si por cuestiones relacionadas con el desborde de la memoria, hubiese que incluir la opción <argLine>-XX: MaxPermSize=256m -Xmx1024M</argLine>, se hará en la sección *"properties"* del pom.xml.

Cobertura con Jasmine y Saga

En proyectos *maven,* multimodulares, que presentan tests tanto en lenguaje Java como JavaScript, hay que añadir una serie de modificaciones con el fin de que los tests JavaScript se ejecuten correctamente y se genere el reporte de cobertura JavaScript en un formato admisible por SonarQube. A continuación se listan los cambios necesarios a realizar, todos ellos en archivo pom.xml del módulo view:

- Incorporar el plugin jasmine-maven-plugin junto con el plugin **saga-maven-plugin** definido en el siguiente profile. De modo que los tests de JavaScript se ejecuten mediante el comando mvn verify y se genere un fichero total-coverage.dat, con la cobertura JavaScript.

  Es preciso tener en cuenta las siguientes etiquetas dentro de la configuración en ambos plugins, para que Saga funcione con Jasmine:

  - Etiqueta ***keepServerAlive*** en el plugin de Jasmine. Esto indica a Jasmine que siga funcionando hasta que la ejecución de Maven se complete (para que Saga pueda recoger los resultados de sus pruebas).
  - Etiqueta ***jsSrcDir*** en el plugin de Jasmine, para indicarle la ruta donde se almacena el código fuente JavaScript.
  - Etiqueta ***baseDir*** en el plugin de Saga, en la que se incluye la dirección de Jasmine utilizada por Saga para conectarse. Se indica un número de puerto ${jasmine.serverPort} establecido por el plugin de Jasmine que varía cuando ejecutan los test.

- Etiqueta ***outputDir*** en el plugin de Saga. Se indicará la ruta en la que se encuentran los informes tras la ejecución de los tests.

Existen otros parámetros opcionales para el plugin de Jasmine pero estas dependerán del proyecto.

jasmine-maven-plugin

jasmine-maven-plugin

```
<plugins>
        <plugin>
                <groupId>com.github.searls</groupId>
                <artifactId>jasmine-maven-plugin</artifactId>
                        <executions>
                                <execution>
                                        <goals>
                                                <goal>test<
/goal>
                    </goals>
                                </execution>
                            </executions>
            <configuration>
                                <jsSrcDir>${project.basedir}
/src/main/webapp/scripts/view</jsSrcDir>
                <keepServerAlive>true</keepServerAlive>
            </configuration>
        </plugin>
</plugins>
```

saga-maven-plugin

saga-maven-plugin

```
<plugins>
        <plugin>
                <groupId>com.github.timurstrekalov</groupId>
                <artifactId>saga-maven-plugin</artifactId>
                        <executions>
                                <execution>
                                        <phase>verify</phase>
                                                <goals>

<goal>coverage</goal>
                        </goals>
                                </execution>
                        </executions>
                        <configuration>

<baseDir>http://localhost:${jasmine.serverPort}</baseDir>
                <outputDir>${project.build.directory}/coverage<
/outputDir>
                        </configuration>
        </plugin>
</plugins>
```

- Añadir el plugin **maven-antrun-plugin**. Este plugin va a trasladar los resultados obtenidos con Saga a un fichero con formato procesable por Sonarqube. Es decir, hará un volcado del contenido del fichero total-coverage.dat a un total-coverage-sonar.dat y añadirá la ruta correcta en la que se encuentra el código JavaScript.

maven-antrun-plugin

maven-antrun-plugin

```
<plugins>
        <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-antrun-plugin</artifactId>
                        <executions>
                                <execution>
                                        <id>saga-to-sonar-lcov<
/id>
                    <phase>verify</phase>
                    <goals>
                                                <goal>run<
/goal>
                                        </goals>
                                        <configuration>
                                                <target>
                                                        <echo
message="setting up sonar lcov file (from saga): ${saga.output.
coverage.sonar.lcov.file} ..."/>
                                <copy file="${saga.output.coverage.
file}" tofile="${saga.output.coverage.sonar.lcov.file}"/>
                                                        <echo
message="replacing '${saga.output.lcov.source.dir.path} with
'SF:src/main/webapp/scripts/view/' in ${saga.output.coverage.
sonar.lcov.file}"/>

<replace file="${saga.output.coverage.sonar.lcov.file}"
token="${saga.output.lcov.source.dir.path}" value="SF:src/main
/webapp/scripts/view/"/>
                                                </target>
                                        </configuration>
                                </execution>
                        </executions>
        </plugin>
</plugins>
```

- También es necesario añadir una serie de propiedades para que los plugins queden correctamente configurados.

apartado "properties"

apartado "properties"

```
<properties>
        <!-- Propiedades del plugin Saga usado por el puglin
Antrun-->
```

```
        <!-- La propiedad que indica la carpeta donde saga
añade los reportes. Ruta donde se añadirán todos los ficheros
de cobertura-->
    <saga.output.dir>${project.build.directory}/coverage</saga.
output.dir>
        <!-- La propiedad que indica la ruta al reporte de
cobertura obtenido tras la ejecución de los tests. Ruta del
reporte de cobertura obtenido-->
        <saga.output.coverage.file>${saga.output.dir}/total-
coverage.dat</saga.output.coverage.file>
        <!-- La propiedad que indica la ruta al reporte de
cobertura modificado con el plugin maven-antrun-plugin-->
        <saga.output.coverage.sonar.lcov.file>${saga.output.
dir}/total-coverage-sonar.dat</saga.output.coverage.sonar.lcov.
file>
        <!-- La propiedad que indica la ruta incorrecta
contenida en el primer reporte.-->
    <saga.output.lcov.source.dir.path>SF:src/</saga.output.
lcov.source.dir.path>
</properties>
```

profile.xml

Como norma general: ***NO SE DEBERÁN USAR LOS PROFILES SALVO AUTORIZACIÓN EXPRESA DEL EQUIPO DE ARQUITECTURA.***

Si la utilización de perfiles, se acordara con el equipo de arquitectura, los distintos *"profiles"* se organizaría en un archivo de configuración externo en el que se definirán los entornos de despliegue, colocándose las propiedades necesarias para el correcto funcionamiento de la aplicación debidamente etiquetadas siguiendo las pautas recogidas en el presente artículo.

Este protocolo para la configuración de las aplicaciones por entorno de despliegue mediante un archivo externo, está enfocada para aquellas aplicaciones realizadas en Tecnología Java EE, utilizando Maven como herramienta de gestión y construcción de proyectos.

Todas las propiedades a utilizar se definirán en el catálogo de etiquetas

Por cada aplicación es obligatio crear un archivo .xml con el código de la aplicación en la CMS (Por ej. dy-rxxi-disp.xml), en el que se colocarán, bajo los distintos perfiles, el valor de todos los parámetros requeridos por la aplicación para su despliegue y correcto funcionamiento. A continuación se muestra un ejemplo de cómo podría ser el archivo.

Ejemplo del fichero de profile.xml

profile.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd
/settings-1.0.0.xsd">
<profiles>
        <profile>
                <id>all</id> <!-- Parámetros generales de la aplicación -->
                        <properties>
                                <entorno>comun</entorno>
                                        <property1>¿ </property1>
                                        <property2>¿ </property2>
```

```
                                        ¿
                        </properties>
        </profile>
        <profile>
                <id>pre</id>
                        <properties>
                                <entorno>PRE</entorno>
                                        <property3>¿ </property3>
                                        <property4>¿ </property4>
                                        ¿
                        </properties>
        </profile>
        <profile>
                <id>pro</id>
                        <properties>
                                <entorno>PRO</entorno>
                                        <property3>¿ </property3>
                                        <property4>¿ </property4>
                                        ¿
                        </properties>
        </profile>
</profiles>
</settings>
```

El etiquetado seguirá el formato "aplicación.servicio.atributo" en el que "aplicación" se refiere al sistema externo al que se conecta, "servicio" llama a cualquiera de los ofertados, y "atributo" es alguna de las características del citado servicio.

Si el parámetro no se refiere a ningún sistema externo, se puede obviar.

Las propiedades no tienen que estar definidas necesariamente por una única etiqueta, permitiéndose su composicion a través de varias etiquetas. Por ejemplo:

- Usuario de BBDD: <jdbc.username>
- Codificación de BBDD: <jdbc.encoding>
- Ruta almacén de certificados de @firma: <afirma.trustStore.url>
- Contraseña para dicho almacén: <afirma.trustStore.password>

El archivo de etiquetado será entregado junto con el código fuente. El valor asignado a cada etiqueta para los perfiles (PRE, PRO, etc.) vendrá vacío ya que serán completados por el personal de CTI.

Aplicaciones Dockerizadas

Aquellos productos que, bajo acuerdo con el Área de Gobernanza, entregan su propio Docker, deben cumplir con las siguientes directrices:

1. **Origen de la Imagen**: La imagen especificada en el Dockerfile debe provenir exclusivamente del registry DML de la STIC. No se permite el uso de repositorios externos con el fin de evitar inconvenientes con Docker Hub.
2. **Especificación de la Imagen Fuente**: La instrucción "FROM" del Dockerfile deberá contener la ruta precisa de la imagen fuente dentro de la DML. Ejemplo: FROM ${REGISTRY}/apine:3.20.0
3. **Incorporación de Nuevas Imágenes**: En caso de que la imagen requerida no esté disponible en la DML, se deberá solicitar su inclusión en el repositorio correspondiente.
4. **Referencia al registry**: El DockerFile deberá contener una referencia al registry que será provista via build argument con el nombre REGISTRY.

Ejemplo Dockerfile

```
ARG REGISTRY

FROM ${REGISTRY}/php:8.3.9-apache-bullseye

...
```

Estas medidas aseguran un entorno controlado y seguro para el desarrollo y despliegue de aplicaciones dockerizadas.

Anexos

ANEXO I: Catálogo de etiquetas

- **Atributos generales**:

  -*username*: Usuario

  -*password*: Contraseña

  -*url*: Cadena de conexión

  -*host*: Servidor

  -*port*: Puerto

  -*idApp*: Identificador de la aplicación

  -*xmlPath*: Ruta de las plantillas

  -*auth*: Autenticación

  -*wls*: WebLogic Server

  -*encoding*: Codificación

  -*version*: Versión

  -*branch*: rama (para el LDAP, por ejemplo)

  -*containerId*:

  -*ws*: Servicio web

  -*n1*: Nodo 1, propiedad dependiente del nodo donde se realiza el despliegue

  -*n2*: Nodo 2, propiedad dependiente del nodo donde se realiza el despliegue

  -*n3*: Nodo 3, propiedad dependiente del nodo donde se realiza el despliegue

  -*balaneador*: Servicio web del balanceador

- **BBDD**: jdbc

  -*driver:* Driver

  -*schema:* esquema de BBDD

  -*dataSource:*

  -*pool:*

  -*dialect:*

-*max_active:*

-*max_wait:*

-*max_idle:*

-*unidad:* pendiente conocer función para decidir si se coloca como general

-*modulo:* pendiente conocer función para decidir si se coloca como general

-*perfil:* pendiente conocer función para decidir si se coloca como general

-*codigoDiraya:* pendiente conocer función para decidir si se coloca como general

- **Logs**: log4j

  -*rootCategory:* Nivel de Logs y Appenders a utilizar

  -*appender:* atributo para los appenders

  -*file:* Ruta para los ficheros de log

- **Correo**: mail

  -*transport.protocol*: Protocolo de comunicación

  -*default.from*: dirección de correo por defecto del remitente

  - *default.to* : dirección de correo por defecto del destinatario

  -*default.subject*: Asunto por defecto.

- **@firma**: afirma

  -*trustStore*: Almacén de certificados

  -*keyStoreType*: Tipo de almacén

  -*service.nativo*: Servicio de @firma (Núcleo)

  -*service.ticket:* Servicio de @firma (fachada de tickets)

  -*generarTicket:* Endpoint del ws que se usa al generar el ticket en la autenticación web

  -*validacionTicket:* Endpoint del ws que se usa al obtener la información de validación del ticket anterior.

  -*comeBack:* Dirección de retorno al terminar la validación del certificado

  -*schemaLocation:* Datos de conexión del interfaz web del módulo de autenticación para realizar la validación del certificado

- **LDAP**

  -*manager.dn:* pendiente conocer función para decidir si se coloca como general

-manager.password (pendiente aclarar si manager se podría colocar como general)

-user.search.base

-group.search.base

- **MACO**: maco

  -codigoModulo:

  -clavePublica: