EGC

Visualización de Resultados

Autores: Gómez Barrera, Rubén
Ojeda Gutiérrez, Alejandro
Rico Ruíz, Javier
Sánchez Crespo, Juan Luis
Velázquez Caballero,Daniel

Fecha: 18 de diciembre de 2014

Índice

Roles....4
Resumen....5
Introducción....6
  Agora@US....6
  Visualización de resultados....7
  Tecnología....7
Gestión del proyecto y código fuente....10
  Gestión de ramas....10
  Forma de trabajar....11
  Caso práctico....11
    Comandos:....12
Gestión de la construcción y despliegue....13
  Maven....13
    Configuración....13
    Usos....13
  Caso practico....13
Gestión de la construcción e integración continua....15
  Jenkins....15
  Tareas....15
  Caso práctico....17
Gestión de la calidad....18
  Aspectos configurados....18
  Aspectos a configurar....18
  Caso práctico....19
Gestión del cambio, incidencias y depuración....20
  Incidencias....20
  Depuración....20
  Cambios....20
  Caso práctico....20
Mapa de herramientas....21
Integración completa....22
  Introducción....22
  Gestion del código....22
  Estrategia de integración....22
  Herramientas usadas....22
    Docker 22
    Nginx....22
    Rake....23
    Seaport....23
  Automatización de la construcción....23
    Maven....26
    Django....26
    Php....26
  Integración continua....26
  Ventajas y desventajas....27
Conclusiones....28
  Sobre la comunicación....28
  Sobre las herramientas....28

Sobre la estrategia de integración.....28
Enlaces.....29

Roles

Nombre y apellidos	Roles
Juan Luis Sánchez Crespo	Líder
Javier Rico Ruíz	Programador
Daniel Velázquez Caballero	Secretario y Programador
Alejandro Ojeda Gutiérrez	Programador y Administrador
Rubén Gómez Barrera	Secretario y Programador

Líderes: serán los encargados de organizar el grupo. En este caso también ha ejercido de todos los demás roles.

Programadores: serán los encargados de desarrollar aplicaciones.

Secretarios: serán los encargados de crear las actas y los documentos necesarios.

Administradores: serán los encargados de la instalación y configuración de las herramientas necesarias en el servidor.

Resumen

El fin de esta practica no es la creación de una aplicación si no aprender a gestionar la configuración de un proyecto.

Para aprender a manejar la configuración de un proyecto se ha decidido crear entre toda la clase un proyecto sobre la creación de votaciones por internet. Este proyecto ha sido dividido en varios sistemas, donde cada uno es una de las partes del proyecto general.

Nuestro grupo es el encargado de la visualización de los resultados. Esta visualización la realizaremos mediante unas series de gráficas que podrá seleccionar el usuario.

A lo largo del curso, todos los grupos hemos ido realizando tareas. Al principio sin control ni herramientas, lo que ha sido caótico. Este caos se ha ido calmando un poco gracias a la utilización de herramientas que nos ayudan a gestionar la configuración de nuestro proyecto.

Una de las herramientas que nos ha ayudado ha sido Redmine. Redmine es una herramienta para la gestión de proyectos. Esta herramienta nos ayuda con:

- Las comunicaciones tanto entre grupo como internas. Estas comunicaciones pueden ser de incidencias o aclaraciones.
- La creación de tareas y asignación de las mismas, para poder saber quien es el encargado de que.

Integrado con Redmine podemos tener git. Git es un repositorio de código por lo que nos ofrece:

- Centralización del código.
- Control de versiones.

Otra de las herramientas que hemos visto ha sido Jenkins. Jenkins es un software de integración continua. Las utilidades que nos ha ofrecido esta herramienta son:

- Compilación automatizada.
- Despliegue automatizado.

El despliegue y compilación lo hace con ayuda de Maven.

Por ultimo para la calidad del código hemos estado investigando sobre SonarQube. Esta herramienta nos permite ver la calidad y descubrir ciertos fallos.

Introducción

Agora@US

Para aprender a gestionar la configuración de un proyecto, se ha decidido montar uno entre toda la clase, para ello se ha dividido en subsistemas y cada grupo desarrollará uno de ellos.

El proyecto es un sistema de votación por internet bautizado como Agura@US. En este sistema un usuario podrá registrarse y crear una votación, o votar en una votación si se encuentra en su censo. Otra de las opciones que se dará en este sistema será la visualización de estadísticas de las votaciones ya terminadas o la deliberación sobre las mismas.

Tras dividir la aplicación ha resultado la siguiente arquitectura para la aplicación.

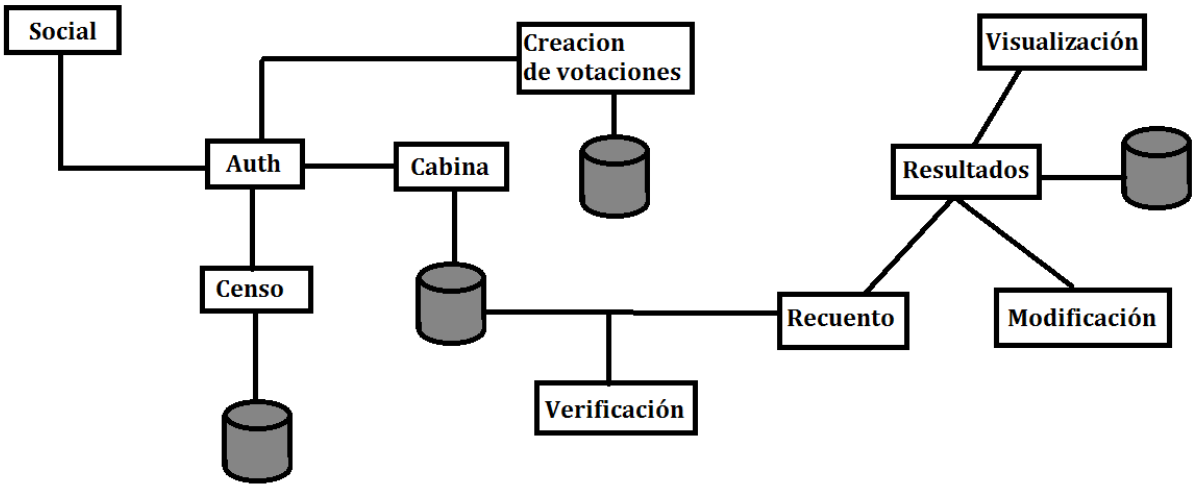


Ilustración 1: Arquitectura de Agora@US

Visualización de resultados

Nuestro grupo es el encargado de desarrollar el sistema de visualización, cuyo propósito es crear gráficas que muestren los resultados de las votaciones a partir de los datos obtenidos de los otros subsistemas.

Para la obtención de datos nos conectamos con dos subsistemas, siempre como clientes. Estos subsistemas son frontend de resultados y creación de votaciones. Más detalladamente:

Creación de votaciones: nos transmitirán un json en el que obtendremos el nombre y el identificador de todas las votaciones finalizadas. Con esta información creamos un menú en el cual el usuario puede elegir la votación de la cual quiere ver las gráficas.

Frontend de resultados: tras enviarles el identificador de la votación en la que estamos interesados, nos transmitirán un json en el que obtendremos los resultados de las votaciones, con los cuales dibujaremos las gráficas.

Tecnología

Para la elaboración de este sistema lo primero que debatimos fue la tecnología a utilizar. Para esto varios miembros del equipo dieron diferente ejemplos, como Spring o HTML con JavaScript.

Tras las exposiciones de cada uno de los miembros de su opción, se decidió afrontar el proyecto con HTML y JavaScript, ya que es más sencillo de desplegar, es una tecnología más simple, consume menos recursos del servidor y existen librerías de JavaScript que dibujan gráficas muy elaboradas.

Ilustración 2: Captura de pantalla del sistema Visualización de Resultados

Gestión del proyecto y código fuente

Para realizar una buena práctica de gestión de proyecto, lo primero que debemos hacer es elegir una herramienta. En nuestro caso para ello estudiamos las ventajas y los contras de tres herramientas: Github, Redmine y Jira.

Tras este estudio previo elegimos como herramienta Redmine, ya que contábamos con la posibilidad de instalarlo en un servidor privado y soporta repositorios como git o svn.

Una vez seleccionado el gestor de proyecto hay que determinar una series de protocolos y políticas de uso de la misma.

Como herramienta de gestión de código decidimos utilizar git, ya que es la más avanzada actualmente y es la que mejor se ajusta al modelo no centralizado de desarrollo de Agora@Us.

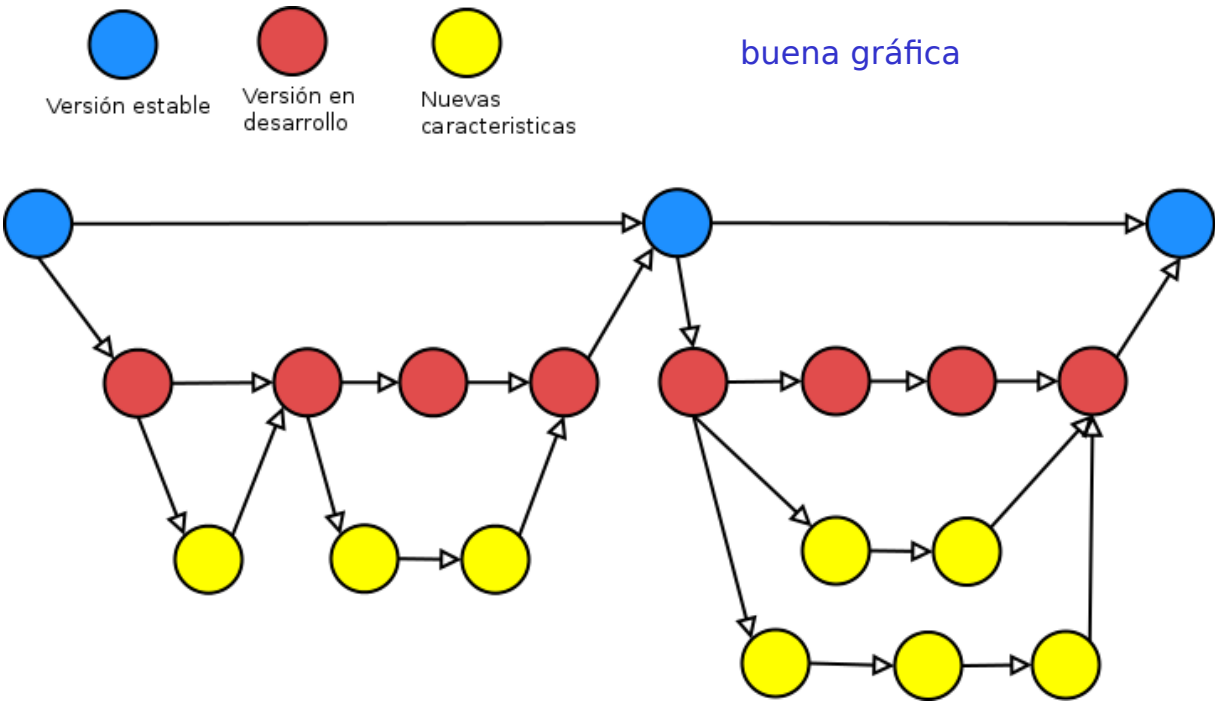
Gestión de ramas

La primera opción que el grupo puso en práctica fue la de trabajar en una única rama dividiendo el código en tres carpetas. Una para el HTML y el CSS, una para la creación de estadísticas y otra para la obtención de datos internos. Tras varias sesiones de trabajo, el grupo tomó conciencia de que no era la mejor forma de trabajar y adoptamos un modelo de gestión diferente.

Se decidió tener una rama “Master” donde se deben tener las versiones estables de nuestro sistema.

En segundo lugar, cuando queramos seguir desarrollando el sistema abriremos una rama nueva a la que llamaremos “Development”. Los cambios realizados sobre esta rama deberán ser insertados en la rama “Master” una vez que haya sido validada y calificada como versión estable del sistema.

Por último, si queremos añadirle una nueva característica o funcionalidad al sistema, abriremos una rama nueva sobre la rama de “Development” y, de forma análoga a la rama de desarrollo y la master, cuando la característica sea desarrollada insertaremos sus cambios sobre la rama “Development”. La principal razón por la que trabajaremos con esta jerarquía de ramas es para no estropear el código que pueda haber en la rama base. De este modo, podemos seguir desarrollando nuestro sistema en la rama de desarrollo sin tocar las versiones estables existentes en la rama master.



buena gráfica

Ilustración 3: Ejemplo gráfico de creación de ramas

Externamente, a nivel del proyecto de Agora@us se está trabajando en proyectos diferentes, con repositorios independientes. Quizás no sea la mejor opción, ya que por ejemplo nosotros podríamos haber creado un proyecto común con el equipo de "Frontend de resultados". Pero, debido a que los objetivos del proyecto no estaban claros al comienzo de éste, se comenzó a desarrollar en repositorios diferentes.

No estamos de acuerdo con esta forma de trabajar, pero una vez empezado el desarrollo es difícil poner de acuerdo a todos los grupos. Uno de los grupos creó un repositorio compartido, pero no todos los grupos participan en el.

Elaborar más por qué no estáis de acuerdo y qué proponéis

Forma de trabajar

Check out: Debido a que en ocasiones trabajarán en paralelo varios integrantes del grupo, se ha decidido que los check out no serán reservados.

Aprobación de los cambios: Serán aprobados por el desarrollador que los ha generado sin la necesidad de validación por parte de otro integrante del grupo.

más detalles en estas partes

Gestión de conflictos: Si git no puede resolver el conflicto automáticamente, el desarrollador que hizo el último 'push' al repositorio será el encargado de resolverlo manualmente.. Para ello hará uso de herramientas Diffs para determinar que parte de código será la definitiva.

Commit: Deberá contener siempre un titulo intuitivo de no mas de 100 caracteres y una descripción más extendida en la que se comente lo que implica ese cambio.

Roles: Se establecen dos roles principalmente en el ámbito de la gestión de código. Por un lado tendremos al administrador (Alejandro Ojeda) y por otro los demás integrantes del grupo que serán simples usuarios.

Caso práctico

Se desea añadir una nueva característica al sistema desarrollado hasta el momento. En concreto,

se quiere añadir un menú desplegable en el que aparezcan todas las votaciones existentes en el sistema de Agora@us. Partimos de la situación en la que la persona o personas que van a desarrollar dicha características se encuentran en la rama “Development”.

En primer lugar, deben crear sobre la rama de desarrollo una nueva rama de característica, en nuestro caso la rama se llama “jsdropdown”. Una vez creada la rama, nos cambiamos a la rama nueva y la enviamos al servidor. Tras ello, se desarrolla la característica que queremos implementar. Durante la implementación, el desarrollador deberá realizar los commits correspondientes a cada uno de los cambios que vaya realizando sobre el sistema. Cuando la funcionalidad esté totalmente desarrollada, el programador deberá realizar un push para enviar todo los cambios al servidor y luego cambiarse a la rama “Development” para realizar un merge de la rama “jsdropdown”. De este modo la nueva característica será insertada en la rama de desarrollo.

Comandos:

```
git branch jsdropdown #(Creación de la nueva rama “jsdropwn”)
git checkout jsdropdown #(Cambio a la rama “jsdropwn”)
git push origin jsdropdown #(Enviar rama al servidor)
git commit -a #(Commit de los cambios realizados)
git push #(Enviar cambios realizados a la copia del servidor)
git checkout jsdropdown #(Cambio a la rama “Development”)
git merge jsdropdown #(Inserción de la nueva característica a la rama “Development”)
```

????

Podría haber más info en este apartado aunque está bien sintetizado.

Falta el ejercicio explítico aunque supongo que el anterior es el ejercicio.

Gestión de la construcción y despliegue

Otras de las tareas que se pueden automatizar son la construcción y el despliegue. Para esto tenemos herramientas como: Maven, Apache ANT, Phing, Scons y Gradle.

Con estas herramientas se crea una configuración, en la que se pueden especificar cosas como la configuración del servidor en el que se va a desplegar, las dependencias que se van a descargar y los repositorios de los que se van a descargar esas dependencias.

Maven

A la hora de intentar desplegar proyectos de otros grupos nos dimos cuenta que podíamos ahorrarnos toda la parte de la configuración, automatizándola con maven ya que contenían ficheros de configuración pom.xml.

Esto es algo que nos facilita muchas cosas a la hora de la integración continua, ya que con llamar a maven, el ya sabe como actuar.

Configuración

Para que maven funcionara correctamente, lo único que tuvimos que hacer tras su instalación, es configurar el servidor de aplicaciones que vamos a utilizar. Para esta tarea, tuvimos que mirar los archivos de configuración (pom.xml) de varios sistemas, para ver que nombres les habían dado a sus servidores de aplicaciones.

Tras saber los nombre que tenían los servidores de aplicaciones, nos fuimos a el settings.xml de maven y añadimos las lineas oportunas para agregar un nuevo servidor de aplicaciones.

Usos

Las tareas en las que hemos utilizado maven son para empaquetar y desplegar aplicaciones de otros sistemas, creadas en Spring.

El empaquetado lo hemos utilizado para poder ejecutar clases que tenían estas aplicaciones, las cuales te inicializaban los datos de la base de datos. Para poder ejecutar estas clases primero, colocados en el lugar donde se encuentra el fichero de configuracion, con el comando :

```
mvn package
```

Lo empaquetamos para a continuación utilizar el comando:

```
mvn exec:java -Dexec.mainClass='utilities.PopulateDatabase'
```

Y ejecutar la clase que queríamos utilizar.

El despliegue lo hemos utilizado para facilitar el despliegue en el servidor Tomcat. Esto es sobre todo utilizado por el servidor de la automatización de la integración continua(Jenkins). Para que esto funcione correctamente, tenemos que tener en el fichero de configuración el nombre del servidor al que se haga referencia en el archivo pom.xml del sistema y estando en el directorio donde se encuentre dicho archivo ejecutar el comando:

```
mvn tomcat7:redeploy
```

Caso practico

Vamos a suponer que vamos a desplegar por primera vez un nuevo sistema realizado en Spring.

Tras la ejecución del comando de despliegue nos salta un fallo por el cual nos damos cuenta de que el nombre del servidor que aparece en el fichero pom.xml no esta en nuestra configuración del Maven.

Para solucionar este error lo primero que tenemos que hacer es irnos al fichero pom.xml del sistema y buscar la etiqueta “<server></server>”, en dicha etiqueta aparece el nombre con el cual ellos se refieren a el servidor de aplicaciones.

Una vez que tenemos el nombre, en el fichero de configuración de Maven settings.xml dentro de la etiqueta “<servers></servers>”, introducimos lo siguiente:

```
<server>
	<id>nameserver</id>
	<username>user</username>
	<password>key</password>
</server>
```

En esta configuración se debe sustituir las siguientes palabras:

nameserver: este es el nombre que obtuvimos antes del fichero pom.xml

user: nombre de un usuario de nuestro Tomcat con permisos para poder desplegar aplicaciones.

key: contraseña del usuario de Tomcat.

También sería mucho mejor que se explicara más y mejor!!!

Gestión de la construcción e integración continua

Si la integración de un proyecto la realizamos continuamente es algo que nos consumira algo de tiempo al principio y muy poco tiempo a lo largo del proyecto si lo realizamos con programas que se encarguen de la integración continua. Si no optamos por la integración continua y dejamos la integración para cuando este terminado todo, la integración se puede convertir en un gran problema.

Para la automatización de la integración continua hay ciertas cosas que son interesantes tener en cuenta como:

Herramienta local o servidor: estas herramientas pueden estar instaladas tanto en local como en un servidor.

Herramienta de construcción: tiene herramientas que no construya el proyecto.

Tecnología: puede ser que la herramienta sea mejor para una cierta tecnología que para otra o que ni soporte alguna tecnología.

Sistema de control de versiones: tiene que ser capaz de integrar sistema de control de versiones como svn o git.

Coste: uno de los principales puntos, ya que desde el punto de vista empresarial estas herramientas son para reducir el coste.

Notificaciones: permita notificar sobre los despliegues.

Jenkins

Hay herramientas de integración continua que satisfacen lo anterior como son: Jenkins, Bamboo, TeamCity y Buildbot. En este caso hemos optado por Jenkins, ya que es una herramienta con una interfaz sencilla y muchos plugins ya desarrollados. En nuestro caso lo hemos instalado en un servidor web, en el cual todos tenemos un usuario creado.

Los plugins más destacados de los que hemos instalados son:

GIT client plugin y GIT plugin: plugins necesarios para que Jenkins pueda descargar códigos de los repositorios basados en git.

Sonar Plugin: nos permite pasar SonarQube a las tareas.

Unicorn Validation Plugin: para validar documento HTML hemos instalado este plugin.

Tareas

Para la integración hemos decidido integrarnos nosotros y los dos sistemas de los que dependemos, que son Frontend de Resultados y Creación y administración de votaciones. Esto es posible ya que estos sistemas tienen sus datos de ejemplo. Para separar nuestras tareas de las otras, hemos creado dos etiquetas. Externas, para las tareas que no nos pertenecen e Internas, para nuestras tareas.

Para el desarrollo del subsistema hemos añadido varias tareas a Jenkins. Entre otras:

- Desplegar nuestro subsistema en un servidor web y validar el código HTML de nuestro subsistema..
- Compilar y desplegar la ultima versión de Creación y Administración de Votaciones en Tomcat para hacer pruebas.

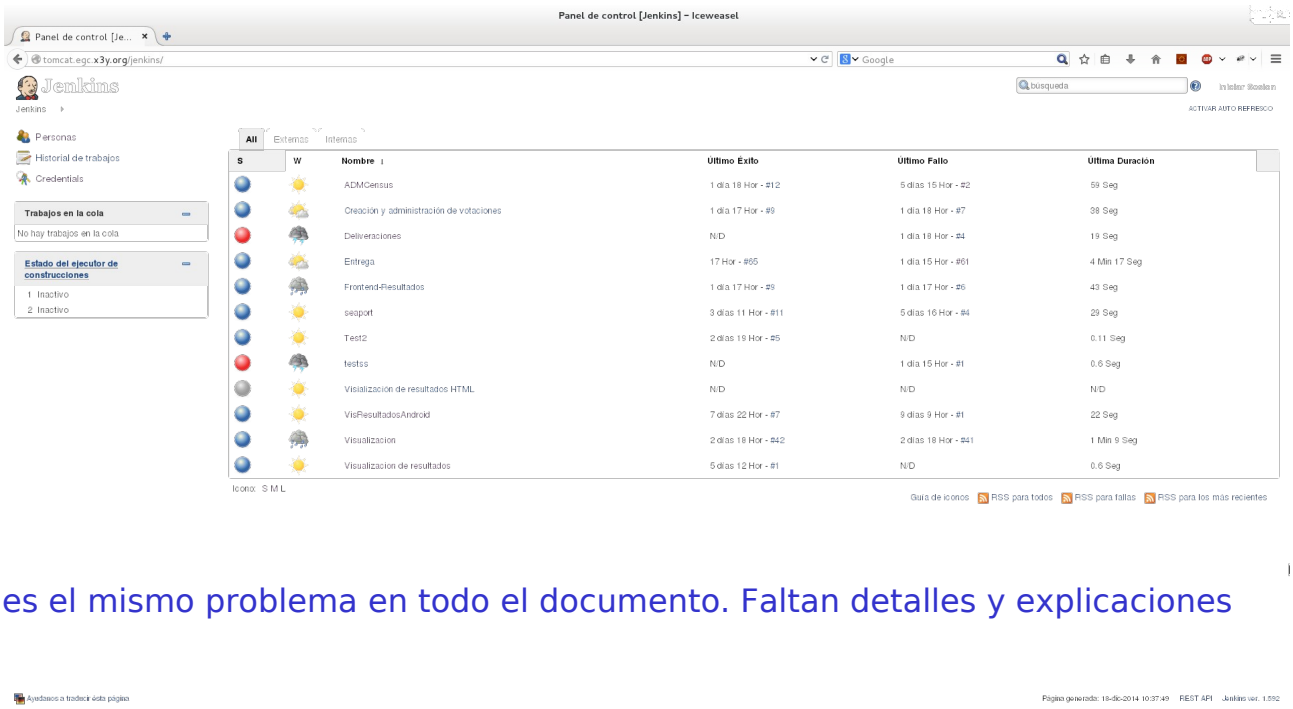
- Compilar y desplegar la ultima versión de Frontend de Resultados en Tomcat para hacer pruebas.

Respecto a las tareas correspondiente a los servicios de los que dependemos, hemos decidido ejecutarlas semanalmente por si hay cambios en su códigos. Estas tareas se descargan y se le pasa maven para desplegarlas en Tomcat. Si estos proyectos tienen algún test también sera ejecutado automáticamente por maven, y si algún test no es superado la aplicación no sera desplegada y aparecerá como fallida en Jenkins .Las bases de datos han sido creadas con anterioridad. Tras desplegar correctamente la aplicación se pasa sonar para comprobar la calidad del código.

Para nuestro servicio, el despliegue sera diariamente. En esta tarea se descarga el código de la rama development. Cuando tenemos el código lo desplegamos en un servidor web y valida el código html. Por ultimo se le pasa sonar para comprobar la calidad del código. En el caso de que la ejecución sea inestable se le mandara un correo a todas los integrantes del grupo.

Ha parte de los dos servicios necesarios, hemos credo dos tareas más para comprobar el correcto funcionamiento del servidor de integración continua. Los otros dos servicios son Deliberaciones y Censo.

En el caso de Censo se despliega correctamente pero Deliberaciones no, ya que le falla uno de los test y maven no lo despliega.



es el mismo problema en todo el documento. Faltan detalles y explicaciones

Ilustración 4: Captura de pantalla de Jenkins

Caso práctico

Vamos a suponer que el servicio de Frontend de Resultados decide dejar de tener ejemplos en sus bases de datos, por lo que tenemos que integrar el servicio de Recuento.

Lo primero sería crear la base de datos, y los usuarios que hicieran falta. Tenemos un script al cual se le pasa el nombre de la base de datos y te la crea con los usuarios que están utilizando todos los grupos que hemos visto.

Una vez tenemos la base de datos ya toca ponernos con Jenkins. Tenemos que darle a crear una

nueva tarea. Ahora pondremos el nombre de la nueva tarea “Recuento” y seleccionaremos copiar una tarea existente. Seleccionaremos unas de las tareas que tienen Spring como ADMCensus y pulsar OK.

En la configuración de la tarea lo único que tenemos que cambiar es el repositorio y en la parte de la configuración del sonar darle otro nombre y clave.

En caso de que no tenga la carpeta test, se tiene que comentar la linea de los test de sonar o dará fallo la ejecución.

Con esto ya hemos agregado una nueva ejecución de otro servicio.

muy poco explicado.

Gestión de la calidad

El grupo ha decidido usar una herramienta durante el proyecto para gestionar la calidad del código generado. Como ya hemos hecho referencia anteriormente esta herramienta es SonarQube. Para su utilización lo hemos instalado en nuestro servidor y hemos descargado el plugin existente para Jenkins. De este modo hemos podido configurar Jenkins para que tras descargar el código, éste sea analizado para comprobar que cumple con ciertos criterios de calidad establecidos como umbrales. No obstante, debido a falta de tiempo no ha dado tiempo de probar la gran parte del potencial que esta herramienta nos puede ofrecer. Es por ello que diferenciaremos entre aquellos criterios de calidad que ha dado tiempo establecer, y aquellos que están pendientes de realizarse de aquí al final de la asignatura.

Aspectos configurados

Umbrales: Sonar nos da la posibilidad de configurar diferentes perfiles para que sean asignados a los diferentes proyectos. En estos perfiles se puede establecer umbrales para una amplia variedad de aspectos del código fuente. Para nuestro proyecto, hasta el momento hemos establecido los siguientes:

	Umbral de advertencia	Umbral de Error
Evidencias bloqueantes	mayor que 0	mayor que 3
Evidencias críticas	mayor que 0	mayor que 5
Evidencias mayores	mayor que 3	mayor que 8
Evidencias menores	mayor que 8	mayor que 15
Bloques duplicados	mayor que 10	mayor que 30
Errores	mayor que 0	mayor que 3

Notificaciones: Los desarrolladores, al subir nuevo código al repositorio, es posible que provoquen que alguno de estos umbrales sean superados. Es por ello, que se ha configurado Sonar para que en estos caso se envíe por correo una notificación a aquel desarrollador que rompa con estas reglas.

Plugins instalados: Para poder realizar análisis sobre nuestro proyecto hemos descargado entre otros los plugins correspondientes para analizar lenguajes web, javascript y css.

Aspectos a configurar

Umbrales: En el apartado anterior hemos decidido una serie de umbrales que están establecidos en nuestro análisis del código. A estos queremos añadirle los siguientes:

- Porcentaje de cobertura de test unitarios.
- Porcentaje de cobertura de test de rendimiento.
- Complejidad del código
- Porcentaje de código comentado.

Otro de los aspectos a tener en cuenta de esta herramienta es la gestión de issues o evidencias de código. En este caso, vemos que cuando sonar detecta alguna evidencia en nuestro proyecto podemos asignársela a alguno de los usuarios para que este sea el responsable de resolverla.

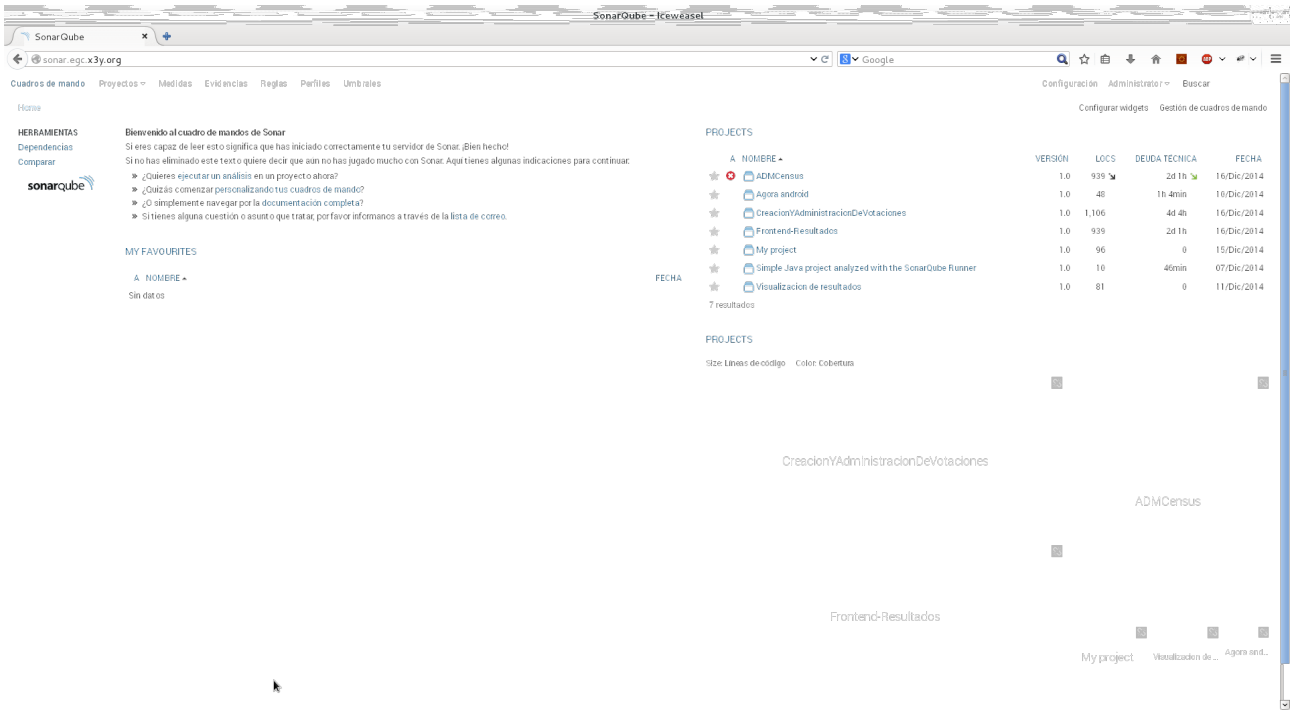


Ilustración 5: Captura de pantallas de SonarQube

Caso práctico

Hemos decidido introducir un nuevo sistema para que pase por SonarQube, pero no tenemos definido umbral para el por lo que tenemos que definir uno.

Lo primero es dirigirnos a Umbrales. Una vez ahí, le damos a crear y nos pedirá el nombre del umbral. Ahora podemos añadir tantas condiciones como queramos a ese umbral.

Hay muchos tipos de condiciones, pero todas se definen igual. Lo primero es elegir la métrica, después elegimos la condición (mayor que, igual que ….) y por ultimo ponemos dos valores, uno para las advertencias y otro para los errores.

Cuando ya tenemos creado nuestro umbral, seleccionamos los proyecto que queremos que tengan dicho umbral.

lo mismo: muy pocas descripciones

Gestión del cambio, incidencias y depuración

Incidencias

La gestión de las incidencias es algo importante y de lo que se tiene que tener un mecanismos en el que quede constancia de las incidencias notificadas y las solucionadas. Esto nos puede ahorrar malentendidos del tipo yo te dije tu me dijiste.

Es cierto que a lo largo de este proyecto no se ha creado un consenso para la resolución de estas incidencias, haciendo cada grupo lo que ha visto oportuno. En nuestro caso con el servidor Redmine, pusimos disponible la opción de que cualquiera pudiera darse de alta y poder comentar en un foro dedicado a los comentarios externos.

Depuración

Respecto a la depuración, no hemos podido realizar pruebas ya que nuestro código solo es HTML, JavaScript y CSS. Lo máximo que podíamos hacer era comprobar que lo realizado funcionaba bien y si se trataba de HTML, pasarle el validado de la W3C.

Por otro lado, el código de los otros grupos, antes de ser desplegado se le ejecutan todos sus test para comprobar que funcionan correctamente, si alguno de estos test no pasa, no se desplegara la aplicación.

Cambios

Los cambios del código decidimos realizarlos en una nueva rama del repositorio. El programador que tenga encargada una nueva funcionalidad tendrá una nueva tarea en el Redmine. Lo primero que tiene que hacer es crear una nueva rama, en la que desarrollará todo el código y una vez haya terminado y haya comprobado que el código funciona y no perjudica a otro código unirá esta rama a la rama de desarrollo.

Respecto a los roles, todo el mundo tiene permiso de modificar cualquier rama, y sera responsable de verificar su propio código.

Caso práctico

Estamos en la situación en la que se desea añadir una nueva funcionalidad a la aplicación, como puede ser un menú desplegable de las votaciones terminadas.

Lo primero que hacemos es hablarlo en grupo y una vez acordado que, como, cuando, cuanto tiempo dispone y quien lo va ha hacer, se crea una tarea en el Redmine.

El seleccionado, creara una rama nueva donde desarrollara la nueva funcionalidad y tras ver que funciona correctamente unirá esta rama a la de desarrollo y dejara la otra abierta, por si tenemos que hacer cambios sobre esa funcionalidad.

Mapa de herramientas

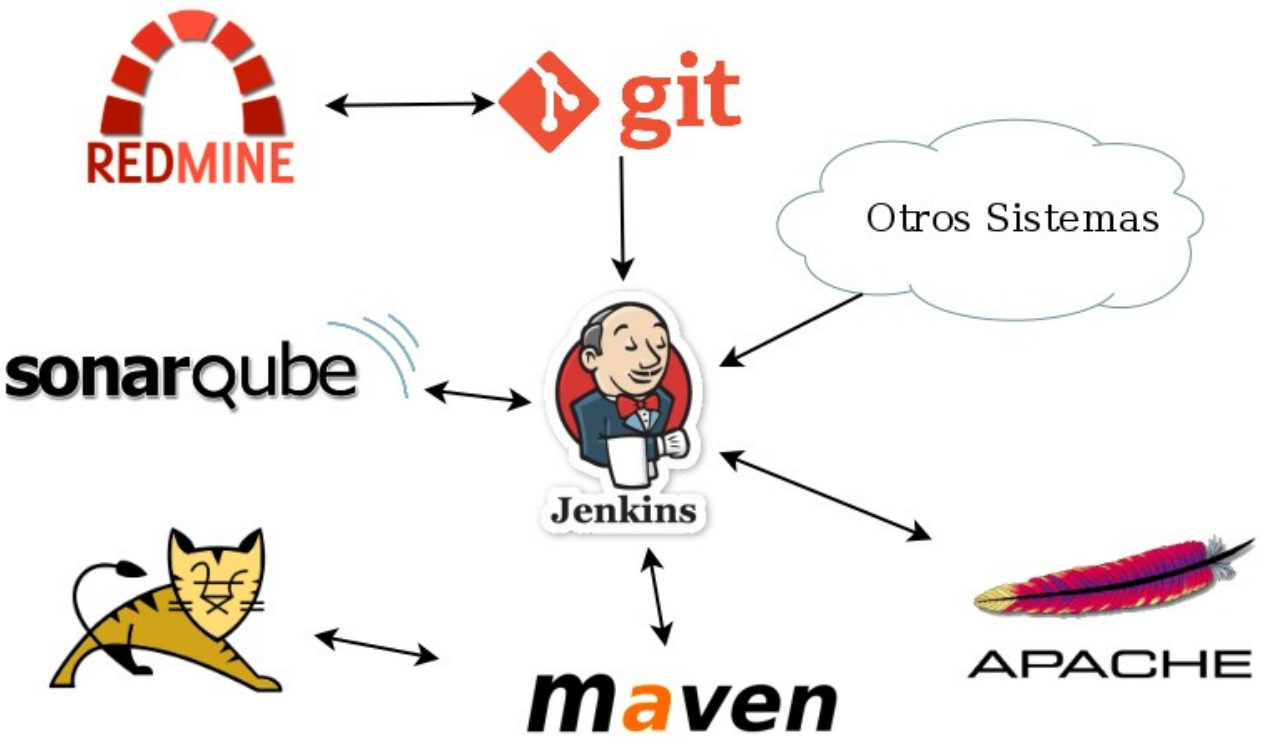


Ilustración 6: Mapa de herramientas

Redmine: herramienta que nos permite administrar nuestros proyectos, gestionar incidencias y comunicaciones. Esta herramienta trae integrada git.

Git: control de versiones integrado en Redmine y del que Jenkins obtendrá nuestro sistema.

Jenkins: servidor de integración continua. Jenkins obtiene los datos de nuestro git o el repositorio que tengan los otros grupos. Estos datos los despliega en Tomcat con ayuda de maven o Apache dependiendo del servicio. Después le pasa SonarQube. En caso de que sea nuestro servicio y el despliegue sea inestable nos mandara un correo.

SonarQube: analizador del código. Obtiene los datos de Jenkins y ofrece los resultados.

Tomcat: contenedor de aplicaciones. Maven desplegará en el las aplicaciones.

Apache: servidor web. Jenkins desplegará en el las aplicaciones web realizadas en HTML, PHP o Phyton.

Maven: encargado del despliegue y empaquetado. Cuando Jenkins le pasa un proyecto, maven empaqueta el proyecto, inicializa la base de datos y lo despliega en Tomcat.

Integración completa

Introducción

Tal y como se expuso en clase, hemos integrado todos los subsistemas desarrollados que hemos podido para que puedan ejecutarse en conjunto en un solo ordenador.

A continuación se detalla todo lo que hemos trabajado y aprendido.

Gestion del código

Para gestionar el código que genera la aplicación compuesta por todos los subsistemas integrados (aplicación final), se ha utilizado un repositorio aparte del de desarrollo.

Aunque entendemos que lo ideal hubiera sido que tanto la integración como el desarrollo se realizará directamente en el repositorio compartido desde el principio, necesitamos hacer modificaciones en los proyectos para integrarlos, modificaciones específicas a nuestra estrategia de integración. Además, no podemos imponer nuestra forma de trabajar a todos los grupos, por ello hemos decidido mantener un repositorio paralelo con los cambios necesarios.

Estrategia de integración

En lugar de instalar todas las aplicaciones en una sola máquina, con los problemas que esto conlleva, por ejemplo, conflictos entre servidores de aplicación por recursos compartidos, como los puertos tcp, hemos decidido instalar cada aplicación en un contenedor, esto es, en una pseudo-máquina virtual que aísla a las aplicaciones entre sí y del resto del sistema operativo.

Así, cada aplicación tendrá su propio sistema de archivos y su propia dirección IP y puertos, eliminando todo conflicto posible sobre recursos de la máquina.

Detallamos las herramientas usadas, así como las ventajas y desventajas de esta estrategia a continuación.

Herramientas usadas

Docker

Es la tecnología que proporciona los contenedores ya mencionados. Inspirada en LXC, utiliza los sistemas de ‘capabilities’, ‘control groups’ y ‘namespaces’ del núcleo del sistema operativo Linux para aislar los contenedores. Es una tecnología nueva y como tal presenta algunos pequeños problemas, pero consideramos que es lo suficientemente útil como para usarla en el proyecto. Estamos utilizando la versión de desarrollo ‘1.3.2-dev, build 353ff40’, por razones de compatibilidad.

Nginx

Lo utilizaremos como servidor proxy para crear la ilusión de que la aplicación se encuentra en su conjunto en un solo dominio.

Rake

Es una herramienta inspirada en GNU Make implementada en Ruby. La utilizaremos para automatizar la construcción, prueba y despliegue del proyecto.

Seaport

Es una extensión para Rake que facilita enormemente la manipulación de contenedores Docker en las tareas de Rake. Ha sido desarrollada en paralelo a este proyecto por un integrante del grupo, que también la utilizará para la automatización de la construcción de su proyecto fin de grado.

Entre otras cosas, Seaport permite realizar la compilación en un entorno conocido (por ejemplo, si un subsistema depende de una versión de JDK y otro de otra versión, es posible compilar ambos subsistemas en una sola máquina, aunque el sistema operativo que ejecuta Seaport no tenga instalada ninguna versión del JDK)

Seaport también permite definir ‘contextos’, es decir, grupos de contenedores que tienen que ejecutarse al mismo tiempo. Esto permite, por ejemplo, ejecutar todos los subsistemas automáticamente, en un entorno aislado, para realizar pruebas de integración.

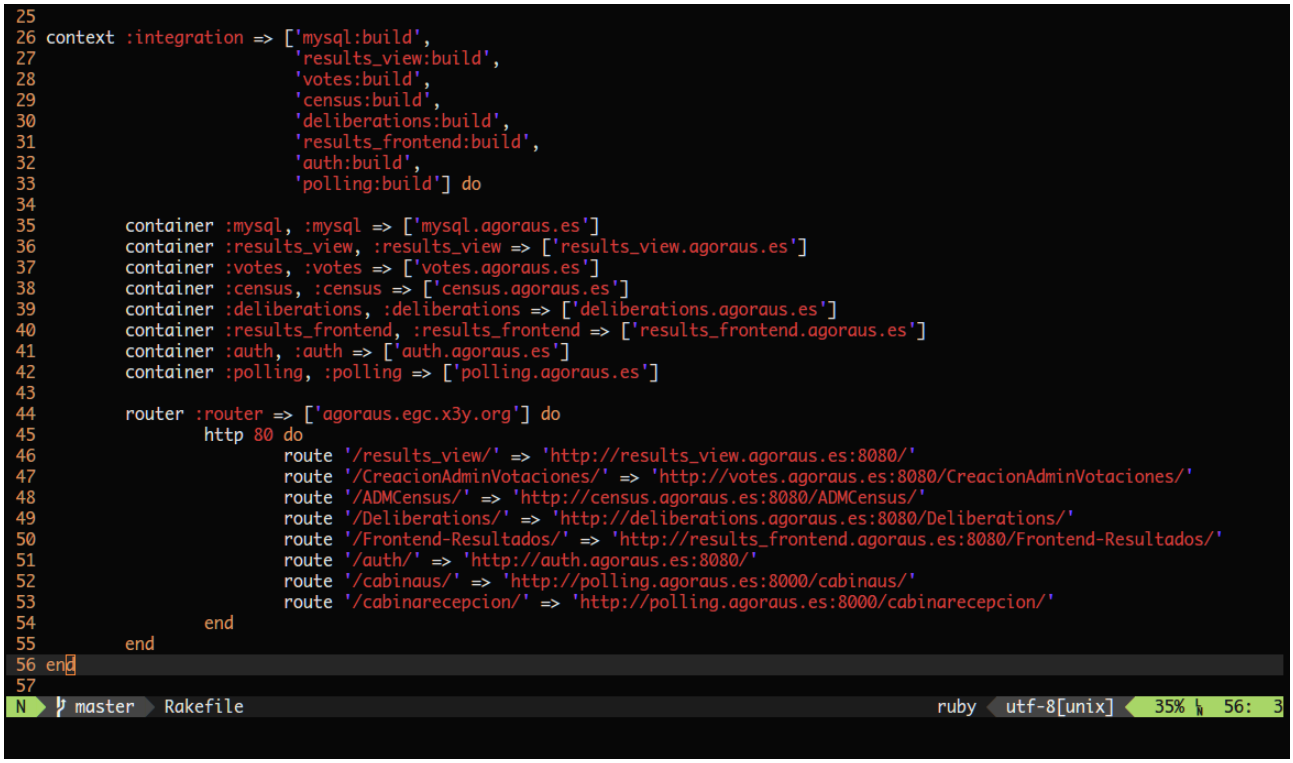
Gracias a esta extensión, podemos compilar, realizar pruebas de unidad, ejecutar la aplicación, y todos sus subsistemas, realizar pruebas de integración, y parar la aplicación, con un solo comando.

Como desventaja señalamos que depende en una versión de desarrollo de Docker dado que la versión estable no dispone de algunas características que necesitamos. En cualquier caso Seaport dispone de una batería de pruebas para comprobar que la versión de Docker instalada funcione como necesitamos así que esto no debería ser un problema.

Automatización de la construcción

Para automatizar la construcción utilizaremos Rake y Seaport. Para ello, hemos añadido un archivo Rakefile general y uno por cada subsistema al repositorio, y en ellos hemos definido todas las tareas necesarias: compilar los war de los proyectos java, pasar las pruebas definidas en maven, generar las imágenes de los contenedores docker, generar los scripts de mysql, ejecutar el proyecto...

Rake permite definir dependencias entre tareas, así como dependencias entre archivos, igual que GNU Make. Configurado correctamente, podemos compilar y ejecutar el proyecto desde cero con un solo comando.



```
25
26 context :integration => ['mysql:build',
27                          'results_view:build',
28                          'votes:build',
29                          'census:build',
30                          'deliberations:build',
31                          'results_frontend:build',
32                          'auth:build',
33                          'polling:build'] do
34
35         container :mysql, :mysql => ['mysql.agoraus.es']
36         container :results_view, :results_view => ['results_view.agoraus.es']
37         container :votes, :votes => ['votes.agoraus.es']
38         container :census, :census => ['census.agoraus.es']
39         container :deliberations, :deliberations => ['deliberations.agoraus.es']
40         container :results_frontend, :results_frontend => ['results_frontend.agoraus.es']
41         container :auth, :auth => ['auth.agoraus.es']
42         container :polling, :polling => ['polling.agoraus.es']
43
44         router :router => ['agoraus.egc.x3y.org'] do
45                 http 80 do
46                         route '/results_view/' => 'http://results_view.agoraus.es:8080/'
47                         route '/CreacionAdminVotaciones/' => 'http://votes.agoraus.es:8080/CreacionAdminVotaciones/'
48                         route '/ADMCensus/' => 'http://census.agoraus.es:8080/ADMCensus/'
49                         route '/Deliberations/' => 'http://deliberations.agoraus.es:8080/Deliberations/'
50                         route '/Frontend-Resultados/' => 'http://results_frontend.agoraus.es:8080/Frontend-Resultados/'
51                         route '/auth/' => 'http://auth.agoraus.es:8080/'
52                         route '/cabinaus/' => 'http://polling.agoraus.es:8000/cabinaus/'
53                         route '/cabinarecepcion/' => 'http://polling.agoraus.es:8000/cabinarecepcion/'
54                 end
55         end
56 end
57
N  master  Rakefile                                   ruby  utf-8[unix]  35%  56:  3
```

Ilustración 7: Captura de pantalla de la definición de Seaport del contexto de la aplicación

También definimos los contenedores que se van a ejecutar, sus nombres de host, y definimos un enrutador HTTP basado en nginx que define el dominio al que accederán los usuarios finales de la aplicación. (En nuestro caso es ‘agoraus.egc.x3y.org’, ya que el dominio ‘x3y.org’ es propiedad de uno de los integrantes del grupo y así podemos acceder al sistema desde internet)

Respecto a los subsistemas individuales, cada uno utiliza sus propias herramientas para construir su proyecto. Entre otras:

Maven

La mayoría de los subsistemas están creados en Spring con ficheros de configuración pom.xml. Gracias a estos fichero podemos configurar fácilmente el despliegue del subsistema gracias ha maven.

Django

El subsistema de cabina de votación, desarrollado en django, utiliza gunicorn como servidor de aplicación, y pip y virtualenv para gestionar las dependencias de su subsistema.

Php

El subsistema de autenticación, desarrollado en Php, debe utilizar forzosamente Apache como servidor web, ya que utiliza archivos .htaccess.

Nosotros lo hemos configurado para que utilice php5-fpm como servidor de aplicación en lugar de mod-php, incrementando así el rendimiento.

Integración continua

Para la integración continua hemos utilizado Jenkins, al igual que en la fase de desarrollo.

Hemos definido una tarea para compilar y probar cada push del repositorio de la integración final. Además, hemos definido una tarea para probar cada push de la librería Seaport que estamos utilizando ya que se está desarrollando en paralelo a este proyecto.

Ventajas y desventajas

Como ventajas de nuestra estrategia señalamos:

- Podemos compilar, probar y ejecutar cualquier versión de la aplicación final en un solo comando, sin tener que instalar 11 subsistemas manualmente, sin tener que instalar todas las versiones de las dependencias de los subsistemas manualmente. Creemos que esta es la razón principal para hacer todo esto y que esta ventaja, por si sola, justifica el coste inicial de montar todo esto.
- Cada subsistema puede tener sus propias dependencias, con sus propias versiones, sin que pueda haber interferencia alguna entre ellos.
- Uno de los principales problemas de este proyecto ha sido que cada grupo ha realizado el desarrollo aislado de los demás. Si se hubiera seguido esta estrategia en todos los grupos desde el principio, dado que arrancar todos los subsistemas esta a golpe de click, podríamos haber ido integrando todo sobre la marcha, y se hubiera detectado al momento si hubiera surgido algún problema que rompiera la integración.
- Dado que cada aplicación esta en su propio contenedor y se comunican entre si por un nombre de host definido, si en algún momento se decide dividir los subsistemas en maquinas físicas distintas es trivial hacerlo.

Como desventajas señalamos:

- Montar toda la automatización tiene un coste inicial considerable de tiempo. Aun así, consideramos que las ventajas justifican este coste inicial.
- Docker es una tecnología nueva y por tanto tiene algunos problemas, algunos difíciles de tratar para alguien que no este familiarizado con ella.
- Además, el uso de Docker puede complicar, aunque no imposibilitar, algunas tareas que serian triviales si no se usara Docker, destacando entre otros el uso de depuradores.

Le sacáis muy poco partido a todo el trabajo hecho con "docker" y demás

Podría mejorar mucho si extendéis esta parte.

Conclusiones

Sobre la comunicación

Con esta practica hemos descubierto lo importante que es la comunicación entre grupos y lo difícil que puede llegar a ser si no se gestiona de alguna manera.

Debido a que no se han establecido unas pautas para todos los grupos, esto ha desencadenado en un caos donde cada grupo definía su manera de trabajar, si la definía, dando lugar a que la integración no ha sido muy buena.

Los aspectos mas importantes que se tenían que tener en cuenta en esas pautas para poder integrarse bien con el resto de los equipos son un canal de comunicación e incidencias, la explicación del fin de su sistema, dar explicaciones de la configuración de su sistema si tiene algunos puntos que no son triviales y crear un repositorio donde todo el mundo pudiera obtener el código que desarrollan.

De los anteriores aspectos, los más importantes desde nuestro punto de vista es la creación de un canal de comunicación e incidencias y un repositorio.

El canal de comunicación e incidencias es muy importante ya que pueden ver aspectos que no están decididos o reflejados o puede haber fallos o cadencias de las que el equipo de ese sistema no este al tanto.

Por otro lado el repositorio de código es muy importante también ya que sin el, el resto de subsistemas no puede obtener el código necesario para poder hacer sus integraciones o para hacer la integración final.

Estas pautas, sobre todo las dos mas destacadas tendría que ser anunciadas en el espacio del grupo correspondiente, donde todos los grupo pudieran con un simple vistazo verlas.

Sobre las herramientas

Otra de los aspectos que hemos visto en esta practica es la importancia de automatizar lo máximo posible el despliegue, la configuración, las pruebas, la inspección del código y la integración.

Hemos estudiado más a fondo herramientas que ya conocíamos como Redmine o git y también hemos dado nuevas herramientas como Jenkins o SonarQube. Estas herramientas nos ayudan a automatizar muchos de los aspectos de nuestro proyecto.

La automatización de estos aspectos puede llevarnos a perder un poco de tiempo al principio, sobre todo las primeras veces, pero es algo que al final nos ahorra muchísimo tiempo, ya que no tenemos que estar preocupándonos de estas cosas, si no que se realizan automáticamente.

Sobre la estrategia de integración

Aunque reconocemos que muchas de las ventajas de la estrategia que hemos utilizado sólo se materializan si se hace desde el principio, con todos los grupos, creemos que es una buena estrategia para el desarrollo de proyectos integrados.

Teniendo todas las ventajas y desventajas en cuenta, creemos que esta metodología es la óptima.

Me gustan las conclusiones. Se nota que habéis aprendido y trabajado. El documento en general se puede mejorar.

Enlaces

Redmine: http://redmine.x3y.org

SonarQube: http://sonar.egc.x3y.org/dashboard/index/

Jenkins: http://tomcat.egc.x3y.org/jenkins/

Repositorio Seaport: git://x3y.org/alex/seaport

Repositorio Entrega: git://x3y.org/egc_entrega