Servidor de mapas de alta disponibilidad con Jboss, GeoServer y PostGIS.

C. Moya Diez[(1)], D. Tabernero Pérez[(1)] y C. Toquero Nieto [(1)]

[(1)] Dept. Informática (INCLAMSOFT), INCLAM S.A., Calle Samaria, 4, 28009 Madrid, inclam@inclam.com.

RESUMEN

Nos encontramos con una necesidad cada vez mayor de publicar la información geográfica y georreferenciada que se posee a través de un entorno web. Esta tendencia se refleja en el aumento de los IDE tanto en el sector público como en el privado. Para ello se necesita una infraestructura que pueda soportar grandes cargas, y que sea lo más fiable posible ante los desastres que se puedan producir.

A lo largo de la experiencia adquirida en multitud de proyectos que involucran tanto la información geográfica como la necesidad de tratarla y servirla a los usuarios, hemos estudiado las distintas soluciones, en el ámbito del software libre: servidores de mapas, servidores de aplicaciones, bases de datos con extensiones geoespaciales, organización lógica de los nodos, etc.

Después de analizar las posibles configuraciones hemos diseño una arquitectura que satisface nuestra necesidades: Geoserver como servidor de mapas, alojado en un servidor de aplicaciones, Jboss, para ofrecer una solución de alta disponibilidad (HA) y rendimiento (HP).

Esta arquitectura desarrollada sobre Jboss + Geoserver + Postgis nos permite tener un sistema de alta disponibilidad/capacidad en el que sea muy sencillo mejorar el rendimiento según el volumen de información con el que trabajemos.

Palabras clave: *Alta capacidad, Alta disponibilidad, IDE, Geoserver, Jboss, PostGIS, SIG, software libre.*

INTRODUCCIÓN

En prácticamente todos los proyectos que desarrollamos en INCLAM tenemos una serie de características similares: un volumen de datos enorme, datos de gran complejidad que han de ser tratados y procesados, necesidad de rapidez de respuesta, funcionamiento continuado al ser muchas veces sistemas de control o de alerta temprana. Este entorno nos ha llevado a estudiar e implementar una infraestructura que nos sirva como base a la hora de desarrollar cada nuevo proyecto y que nos permita dar respuesta a las problemáticas anteriormente descritas.

En este documento vamos a tratar de transmitir la experiencia obtenida en el desarrollo de IDEs de alta disponibilidad, usando soluciones de software libre. No existe una arquitectura única para dar respuesta a todos los problemas (nuestra propuesta no deja de ser un desarrollo específico para cubrir nuestras necesidades), pero las herramientas que vamos a describir, permiten adaptaciones para tratar de ofrecer las funcionalidades requeridas en cada caso.

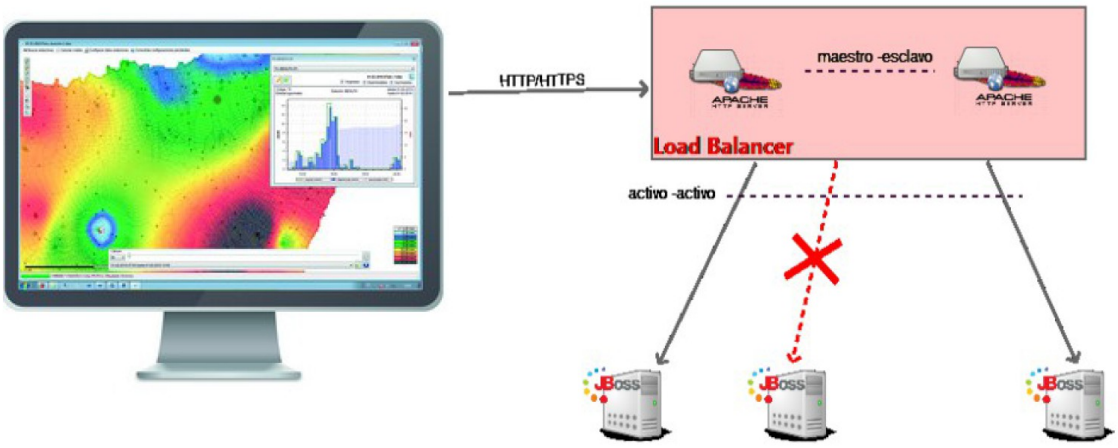

Ilustración 1: Arquitectura base

Esta arquitectura base define un sistema escalable que permite aumentar el rendimiento del procesamiento de datos de forma muy sencilla y mantener la alta disponibilidad de la solución en todo momento.

Partiendo de este modelo de arquitectura, y con experiencia sobre ella, utilizándola en múltiples casos de éxito en distintos proyectos, nos enfrentamos a unas nuevas necesidades.

Necesidades

En las distintas reuniones retrospectivas de muchos de los proyectos que íbamos finalizando nos encontrábamos con una serie de problemáticas repetidas, lo que nos llevó al diseño de una arquitectura común que pudiera dar solución a la mayoría de estos requisitos:

- Minimizar el número de puntos críticos en la infraestructura, puntos que provocan la caída del todo el sistema.
- Minimizar el número de equipos que sostienen a la infraestructura.
- Ofrecer alta disponibilidad, pero a su vez alto rendimiento con todos los nodos activos.
- La información almacenada dentro de la base de datos es altamente volátil, por lo que se necesitan escrituras rápidas.
- Utilización de software libre, de manera prioritaria (aunque no obligatoria)..

ARQUITECTURA GENERAL

Este diseño de arquitectura se basará fundamentalmente en las funcionalidades que nos provee Jboss (actualmente renombrada a Wildfly, a partir de la versión 8.x), tanto en la implementación completa del estándar J2EE como en la versatilidad que ofrece en cuanto a los modos funcionamiento (clusters, granjas de servidores, etc.).

En cuanto a la funcionalidad de servidor de mapas, optamos por la solución más utilizada en entornos Java, que es Geoserver 2.6, que será desplegado en el servidor de aplicaciones Jboss.

En la mayoría de los casos se optará por una base con extensiones geoespaciales para almacenar la información que servirá Geoserver, y siguiendo con la dinámica de utilizar soluciones ampliamente probadas y maduras, se utilizará PostgreSQL junto con la extensión PostGIS (en su rama 2.x).

Ilustración 2: Componentes básicos

Es muy importante destacar, que el resultado tiene que ofrecer interoperatividad entre los componentes y sistemas externos, por lo que todas las soluciones propuestas tendrán que cumplir que se comuniquen mediante protocolos abiertos, documentados y estándares.

SOLUCIONES PROPUESTAS

La arquitectura presentada en el apartado anterior, es muy versátil y permite realizar un proceso de "personalización" para que cada configuración pueda dar mayor relevancia a las peculiaridades del proyecto que va servir. Sistemas complejos, como sistemas de ayuda a la decisión para alarmas para avenidas, tienen como punto crítico la necesidad de funcionar 24x7, y realizar su tarea ante cualquier eventualidad, tanto software como hardware. Mientras que proyectos, de búsqueda de patrones en modelos de cambio climático (utilizando herramientas de Big Data) es necesaria una gran capacidad de cálculo y procesamiento, no tanto de disponibilidad.

Por lo tanto, mediante la arquitectura que presentamos tenemos como objetivos principales obtener:

- Alta disponibilidad: Si un nodo cae, el sistema tiene que continuar funcionando, y cuando se incorpora un nuevo nodo al cluster, tiene que ser transparente al usuario que utiliza el sistema.
- Alta rendimiento: Todos los nodos de la solución tienen que tener un comportamiento activo, es decir, cualquiera de ellos puede responder una petición.

Jboss

Existen varias alternativas en el mercado, incluidas varias de software libre, que nos permiten albergar y servir aplicaciones J2EE de manera estable, segura y administrada. En nuestro caso nos hemos decidido por el servidor Jboss (formalmente conocida como Wildfly en su edición "community") desarrollado por Red

Hat Inc, liberado como software libre, ya que es una de la soluciones más maduras en cuanto a servidor de aplicaciones J2EE, con una gran empresa detrás de su desarrollo junto con un importante soporte de la comunidad del software libre.

Jboss ofrece dos modos de funcionamiento, sobre los que se va a basar cualquier solución que se implemente: standalone y domain.

Standalone

Se arranca el servidor Jboss como una instancia única. Esto significa que no comparte recursos con ninguna otra instancia en el cluster.

Domain

En este modo de funcionamiento se definen un conjunto de procesos Jboss que comparte recursos, funcionando como maestro-esclavo(s).

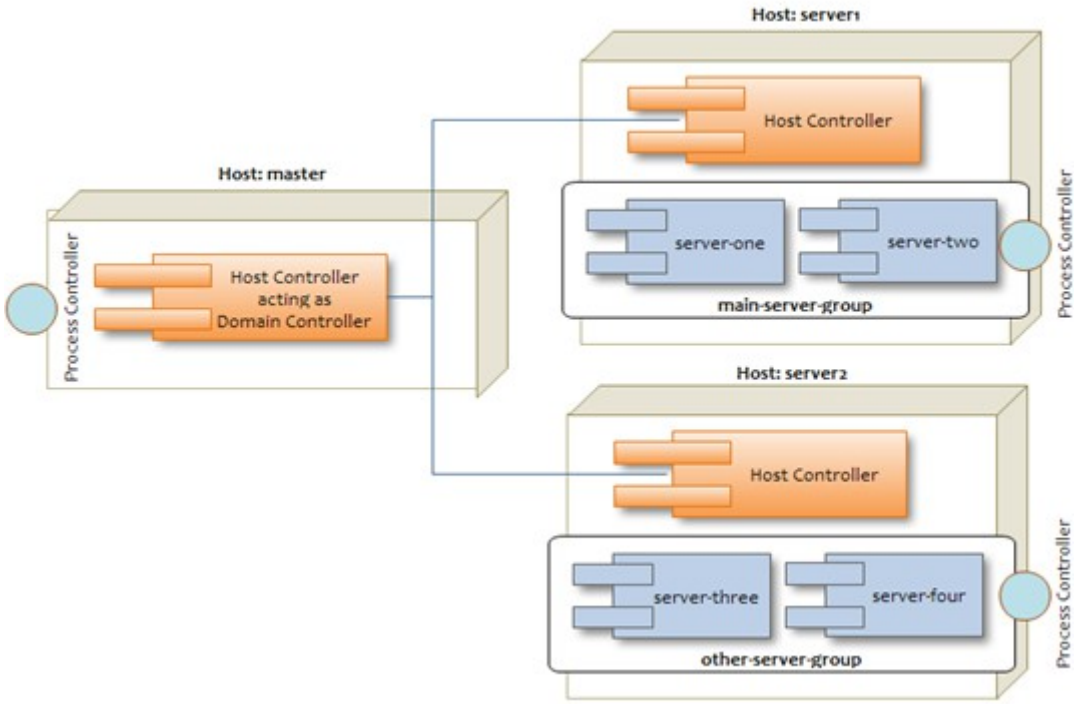

Ilustración 3: Jboss en modo domain

Como se puede ver en la ilustración anterior, en todos los host tiene que haber un "controller" (de color naranja) que es el encargado de la administración a nivel de arquitectura. En los host esclavos pueden ejecutarse una o varias instancias (de color azul), en el maestro se podrían tener instancias pero no se suele hacer, ya que se ocupa exclusivamente de tareas de gestión del cluster.

Se pueden definir "server-groups" a los cuales pertenecerán las instancias, y que permite especificar unas características particulares para cada grupo. Por ejemplo; la maquina virtual de Java que se utiliza, límites de uso de memoria, etc.

Perfiles y clustering

Estos modos de funcionamiento, a su vez permiten utilizar perfiles. Estos perfiles le indican a Jboss que nivel de J2EE se quiere ofrecer en la instancia.

Domain Profile Name	Standalone File Name	Description	Clustered
default	standalone.xml (default)	Java EE6 Web Profile + JCA + JAX-RS + JAX-WS + Javamail + Remote Connectivity	N
ha	standalone-ha.xml	Java EE6 Web Profile + JCA + JAX-RS + JAX-WS + Javamail + Remote Connectivity	Y
full	standalone-full.xml	Java EE6 Full Profile	N
full-ha	standalone-full-ha.xml	Java EE6 Full Profile	Y

Como se puede ver en la tabla, según se arranque con un configuración u otra, se obtienen diferentes niveles de implementación del estándar J2EE, y se activa o desactiva los procesos que permiten el clustering.

En cuanto al clustering que ofrece Jboss, cuando arrancamos con los perfiles “ha”, se activa varios subsistemas como i) “Jgroups”, el cual se encarga de configurar unos canales de comunicación entre cada uno de los nodos que forman el cluster. Habitualmente, se utilizan canales de comunicación TCP y UDP, mediante multicast. Y ii) “modcluster” como balanceador de carga.

Apache

El servidor web Apache se utilizará como frontend, y realizará una funcionalidad de proxy entre el usuario y el servidor Jboss. Se utilizará el módulo “mod_cluster” para conectar con Jboss, y obtener la información del estado del cluster.

Hay que remarcar que Apache no toma ninguna decisión de cara a qué nodo enviar una petición ya que todo esto se configura desde las instancias Jboss.

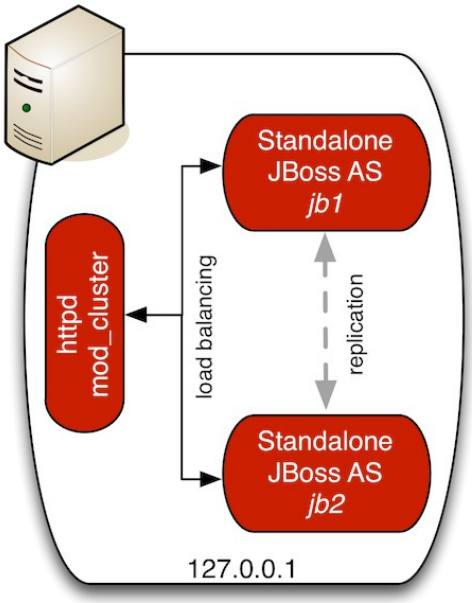

Ilustración 4: Uso de mod_cluster

Pero con la utilización de una sola instancia de Apache se genera un nuevo punto crítico que queremos evitar, por lo que se tiene que optar por alguna solución que permita tener un respaldo en caso que una instancia de Apache caiga. Para ello utilizaremos cualquiera de las múltiples herramientas que permiten clustering en servidores Apache, como puede ser: heartbeat, Pacemaker, o Corosync, por ejemplo.

PostgreSQL

PostgreSQL, es uno de los Sistemas Gestor de Base de Datos (SGDB) más conocidos, y que junto con el plugin PostGIS de extensión geoespacial, es una de las más utilizadas en el ámbito SIG. Ambas piezas de software están muy maduras y se utilizan en la llamada pila de “OpenGeo Stack” para el diseño de estructuras que almacenen grandes cantidades de datos, y que incluyan un componente geoespacial.

Los sistemas actuales de clustering de PostgreSQL se orientan en dos sentidos muy diferenciados:

- Sistemas de activo/pasivo, que nos permiten el uso de servidores de respaldo. Esta configuración nos provoca contar con un punto crítico en el nodo maestro del cluster, que es lo que queremos evitar.
- Sistemas activo/activo pero optimizados para la lectura y no la escritura. Se tienen que utilizar herramientas externas, ya que PostgreSQL no admite clusters activo/activo.

Ambas soluciones no son aceptables para nuestro objetivo, ya que queremos que nuestro sistema soporte alta disponibilidad, alto rendimiento, que permita una actualización muy rápida de los datos y evitar al máximo todos los puntos críticos en el caso de la alta disponibilidad.

NUESTRA CONFIGURACIÓN

Como ya hemos dicho, hay gran multitud de posibles arquitecturas o soluciones que se pueden implementar.

En nuestro caso, buscamos un compromiso entre la alta disponibilidad, sin renunciar al alto rendimiento, minimizando el número de recursos necesarios. También se busca la sencillez en el diseño que se traduce en facilidad a la hora de añadir nuevos nodos dentro de nuestro cluster, y en la construcción del mismo.

A estas unidades que van conformar el cluster, las cuales incorporan todos los servicios (Jboss, Geoserver y PostGIS), mantienen una configuración común (Geoserver tiene la misma configuración en todos los nodos) y que son autocontenidas en cuanto a información (todas las bases de datos tienen la misma información)

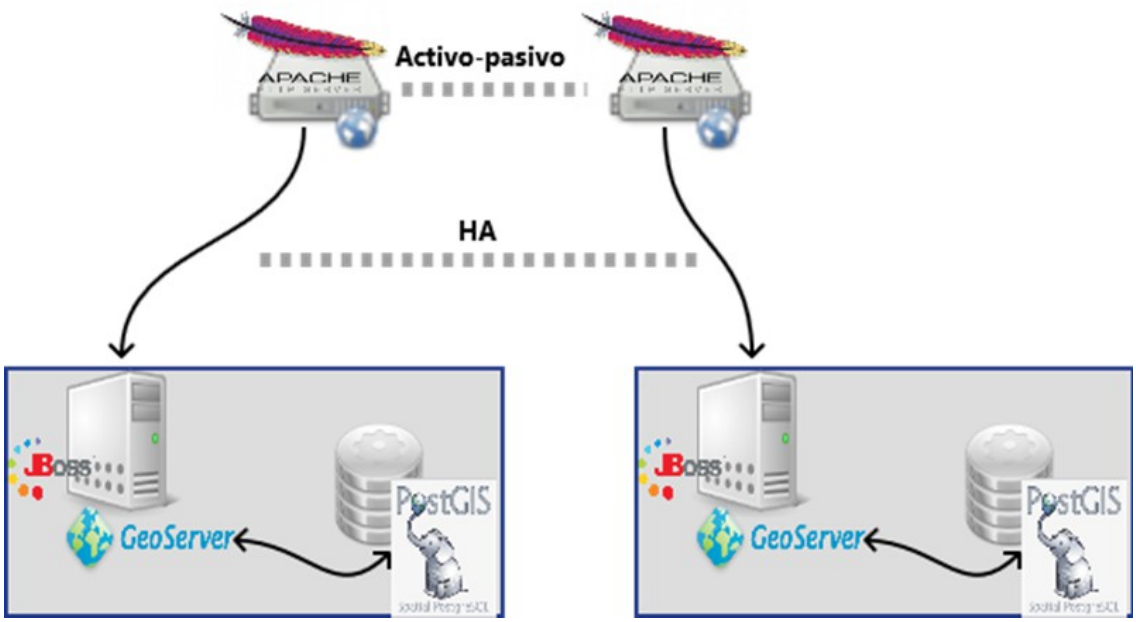

Ilustración 5: Arquitectura desarrollada

La alta disponibilidad se consigue mediante el uso de Apache + mod_cluster, junto con el módulo mod_cluster que corre en Jboss. Se configurará el balanceo de carga mediante la distribución de carga de la CPU, para intentar que ningún nodo tenga una carga superior al resto. Por lo tanto, Apache funciona como un proxy de peticiones hacia los nodos Jboss, pero sin realizar realmente ninguna tarea de balaceo de carga, la configuración de balanceo se hace en la parte Jboss. Para que Apache no sea un punto crítico en cuanto a la alta disponibilidad, usaremos un servidor Apache de respaldo.

Tampoco queremos que la base de datos pueda ser un punto crítico en sistema, por lo que se incluye una instancia independiente en cada uno de los nodos para dar redundancia al sistema de base de datos.

Rendimiento

El rendimiento obtenido, con dicha configuración, se midió utilizando Jmeter, mediante el diseño de unas pruebas que lanzaban peticiones contra Geoserver, utilizando el protocolo WMS, con diferentes BBOX o encuadres. Para usar una prueba más real, se utilizarón diferentes equipos clientes que disponian de Jmeter y que se configuraron para atacar el servicio WMS.

Las características de las pruebas que se lanzarón fueron las siguientes:

- Seis nodos clientes, con Jmeter.
- Cluster de 3 nodos en modo de funcionamiento standalone, con 2 nodos con Apache corriendo, junto con mod_cluster.
- Peticiones WMS a diferentes BBOX

- Uso del formato PNG de respuesta
- Se cargaron dos cartografías de prueba, una vectorial y otra compuesta de ortofotos
- Se lanzaron 200 hilos simultáneos, máximos, en cada nodo (más threads colgaba el equipo cliente)

Tabla 1: Tiempos de respuesta

Hilos	Raster		Vectorial	
	Tiempo medio de respuesta con caché	Tiempo medio de respuesta sin caché	Tiempo medio de respuesta con caché	Tiempo medio de respuesta sin caché
100	0,7 seg.	1,4 seg.	0,4 seg.	0,6 seg.
128	0,8 seg.	1 seg.	0,6 seg.	0,8 seg.
150	1 seg.	1,7 seg.	0,8 seg.	1 seg.
200	1,4 seg.	1,9 seg.	1,3 seg.	1,4 seg.

Pros

Esta arquitectura propuesta tiene como puntos fuertes destacados:

- Es sencillo incorporar otro nodo al cluster, al no incorporar soluciones software complejas, ya que se puede clonar un nodo y con unos cambios mínimos crear una instancia nueva en el cluster.
- Estructura compacta. Un cluster funcionaría con exclusivamente dos nodos.
- Cada nodo es una unidad funciona completa. Tiene todos los servicios para funcionar de manera autónoma, y no hay interdependencias entre nodos.
- Ningún punto crítico con un solo fallo o preparado para responder en alta disponibilidad
- Alta capacidad, todos los nodos trabajarán como uno sólo. Esta funcionalidad es muy sencilla de cambiar dentro de Jboss para que funcione como se desee.
- La caída de un nodo es totalmente transparente al usuario

Contras

A su vez, a lo largo de su uso, hemos encontrado puntos que pueden entenderse como puntos débiles o a mejorar:

- Sincronización de la configuración y datos. Esta arquitectura obliga a desarrollar unas herramientas que permitan mantener la coherencia de los datos y las configuraciones entre todos los nodos que forman parte del mismo. En concreto, el contenido de las bases de datos y la configuración de los servidores Geoserver. Nosotros para solucionar esta problemática hemos utilizado “infinispan” al permitirnos compartir una estructura de datos nombre-valor entre todos los nodos que forman parte del cluster.
- Requiere de monitorización de las instancias PostgreSQL ya que si un nodo tiene su servidor de base de datos caida, no hay método automático para

volver a levantarla. Se pueden usar alguna solución de monitorización como puede ser Pandora FMS, por ejemplo.

OTRAS POSIBLES ARQUITECTURAS

Como hemos intentado remarcar la solución propuesta es simplemente la más útil para nuestros intereses y el tipo de proyectos que desarrollamos. Pero existen infinidad de arquitecturas posibles basadas en la "arquitectura básica" que se ha definido. Por ejemplo en organizaciones muy complejas nunca se usa un proxy software como estamos haciendo nosotros con Apache, sino que se tiene uno hardware que sea el encargado de distribuir las llamadas realizadas.

La arquitectura que se muestra a continuación es una de las más completas, formada por una serie de granjas de Jboss funcionando en modo domain, lo que la dota de una gran capacidad de procesamiento y cálculo. Los esclavos tendrán toda una serie de instancias definidas cada una asociada a un "jboss-group" de tal forma que la alta disponibilidad se establece entre las instancias de estos esclavos que contienen o despliegan las mismas aplicaciones. Esto permite personalizar muchísimo el tipo de balanceo que se quiere por aplicación, así como el tipo de ejecución de la misma y facilitar la distribución geográfica de las granjas.

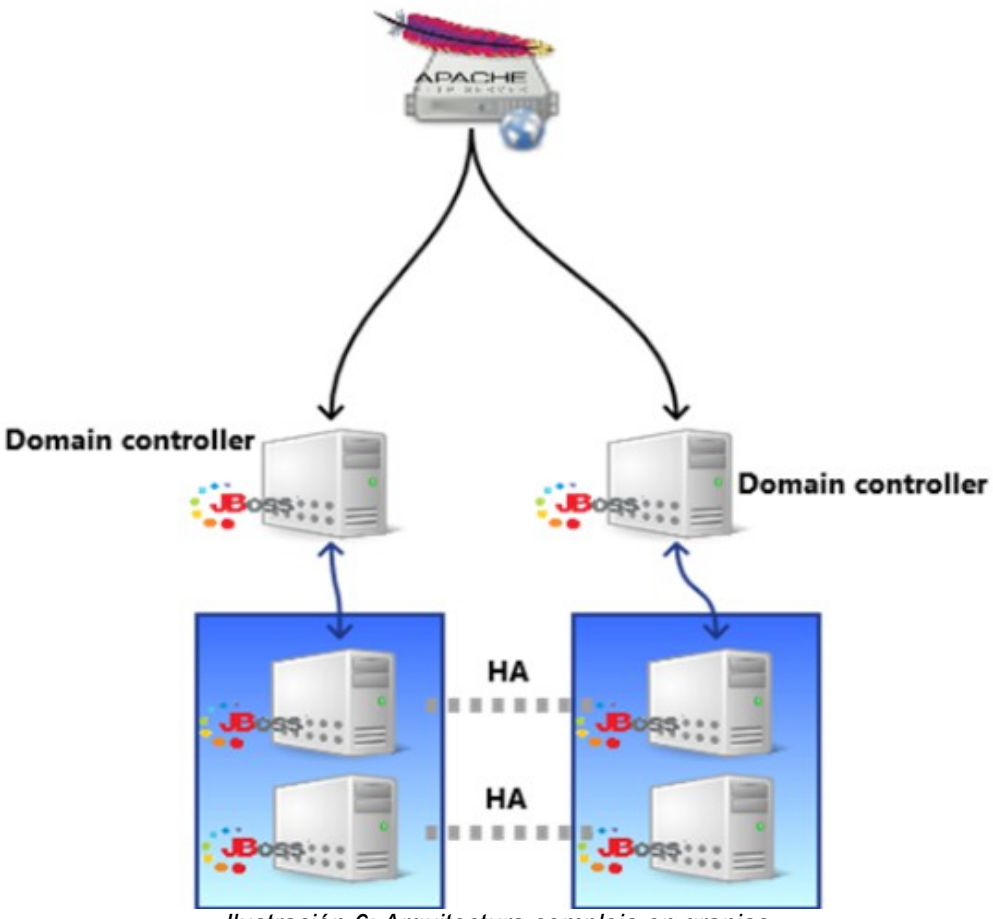

Ilustración 6: Arquitectura compleja en granjas

REFERENCIAS

- Marchioni, Francesco, (2011) “JBoss AS 7 Configuration, Deployment and Administration”, Packt Publishing, ISBN 9781849516785
- Locovella, Stefano; Youngblood; Brian, (2013), “Geoserver, Beginner's Guide”, Packt Publishing, ISBN 9781849516686
- Smith, Gregory, (2010). “PostgreSQL 9.0 High Performance”, Packt Publishing, ISBN 9781849510301
- Teckadmin, (2011) “HighAvailability active-active apache cluster”, URL: https://teckadmin.wordpress.com/2011/07/12/highavailability-active-active-apache-cluster/
- Middleware magic, (2012), “Using mod_cluster with Jboss AS 7.1 cluster”, URL: http://middlewaremagic.com/jboss/?p=2147
- Stack Overflow, (2014), “Does PostgreSQL support Active-Active Clustering with DRBD?”, URL: http://stackoverflow.com/questions/22989249/does-postgresql-support-active-active-clustering-with-drbd
- WILMING, HEINZ, Arquinet, (2012), “MANAGING CLUSTER NODES IN DOMAIN MODE OF JBOSS AS 7 / EAP 6”, URL: http://blog.akquinet.de/2012/06/29/managing-cluster-nodes-in-domain-mode-of-jboss-as-7-eap-6/
- SIMS, SAMUEL, Arquinet, (2012), “SCALABLE HA CLUSTERING WITH JBOSS AS 7 / EAP 6”, URL: http://blog.akquinet.de/2012/07/19/scalable-ha-clustering-with-jboss-as-7-eap-6/
- GIS Stack Exchange, (2014), “How to get JAI + JBoss 7.1.1 + Geoserver 2.3.0 to work?”, URL: http://gis.stackexchange.com/questions/57040/how-to-get-jai-jboss-7-1-1-geoserver-2-3-0-to-work
- Ferraro, Paul; Ban, Bela,”Jboss community: Clustering in AS 7.0”, URL: http://www.jboss.org/dms/AS7/as7webinar/AS7-clustering-webinar.pdf
- Pandora FMS, URL: http://pandorafms.com/