URL : http://blog.benelog.net/2765024

**[스프링과 클라우드가 무슨 상관이 있을까?](http://blog.benelog.net/2765024" \o "스프링과 클라우드가 무슨 상관이 있을까?)**by 정상혁

* 2011/03/03 07:50
* [개발 이야기](http://blog.benelog.net/category/%EA%B0%9C%EB%B0%9C%20%EC%9D%B4%EC%95%BC%EA%B8%B0)
* [blog.benelog.net/2765024](http://blog.benelog.net/2765024)
* 6 comments

**(덧붙임: 이글을 쓰던 시점의 CloudFoundry는 지금의 CloudFoundry와는 다른 기술이였고,  이전 CloudFoundry는 http://classic.cloudfoundry.com/ 으로 옮기어 갔습니다. )**

**스프링, 클라우드를 내세우다.**

 스프링 프레임워크의 개발사인 스프링소스(SpringSource)와 VMWare는 최근 스프링이 클라우드 환경에 적합한 기술이라고 홍보를 하고 있다. 현재 스프링과 관련이 있는 클라우드 플랫폼은 아래와 같다.

**vFabric**

 VMWare에서 제공하는 프라이빗 클라우드를 위한 솔루션이다. VMWare의 가상화 솔루션과 스프링소스의 미들웨어, 프레임웍, 개발도구 등을 합한 기술 스택을 제공한다.

**Coludfoundry (**[**http://www.cloudfoundry.com/**](http://www.cloudfoundry.com/)**)**

 스프링소스가 인수해서 운영하는 클라우드 서비스이다. 아마존의 AWS 인프라를 이용하고, Tomcat 등의 미들웨어를 제공한다.

**VMForce (**[**http://www.vmforce.com/**](http://www.vmforce.com/)**)**

VMWare와 세일즈포스닷컴이 제휴한 클라우드이다. force.com 데이터베이스를 저장소를 이용하고, 스프링,Tomcat, vSphere 같은 VMWare의 솔루션이 들어간다.



VMFoce 클라우드 (출처 vmforce.com)

**Google App Engine**

 구글과 VMWare의 제휴로 Google App Engine에 대한 지원이 스프링소스의 개발도구에 들어가게 되었다. Spring-roo에서 Google App Engine의 저장소를 쓸 수 있도록 JPA의 Provider로  Datanucleus를 선택할 수 있다.

 2010년 10월에 열린 스프링원 컨퍼런스에서도 클라우드에 대한 홍보는 두드러졌다. 자사의 클라우드 솔루션인 vFabric과 CloudFoundry는 물론, VMForce와 Google App Engine 과도 스프링을 묶어서 홍보했다. 스프링원 2010의 플레티엄 스폰서 세 곳 중 두 곳이 구글과 세일즈포스닷컴이였다는 것을 봐도 그 컨퍼런스의 초점은 알 수 있었다

  스프링은 애플리케이션 프레임워크일 뿐인데, 과연 클라우드와 무슨 관련이 있다는 것일까? 클라우드의 핵심기술은 가상화나 분산 저장소 같은 기술이 아닐까? 클라우드 환경에서 JVM만 제대로 갖추어진다면 스프링이 아닌 스트럿츠나 구글쥬스라고 해도 안 돌아갈 이유는 없지 않은가? 요즘은 서버 여러 대만 쓰는 기술이면 다 클라우드라는 이름을 붙이고 싶어하던데, 스프링도 이런 유행에 묻어 가고 싶은 마음이 아닐까?  여러 가지 의구심이 들만도 하다.

 그러나, 이를 억지 홍보 전략이라고만 생각하면서 그냥 넘어가는 것보다는 왜 VMWare ,구글, 세일즈포스닷컴이 스프링을 클라우드에 끼워 맞추면서 제휴를 하고 있는지를 분석해 본다면 기술의 흐름을 이해하는데 도움이 될 것이다.

**지난 10년, 그리고 수익 모델**

 널리 알려 진 것처럼, 스프링은 EJB의 초기 시대에 Java EE 표준에 대한 대안 기술로 시작되었다. 스프링의 아버지 로드존슨은 초기 EJB의 어려움을 해결한 자신만의 프레임워크를 만들었고, 그 코드를 저서 “Expert one to one J2EE development”에 공개했다. 그 코드가 많은 개발자들의 호응을 얻어서 오픈소스 프로젝트가 되었다. 스프링 프레임워크는 세계 10대 은행 중 9개가 쓰고 있다고 할 정도로 인기를 끌었고[주1], 핵심 개발자들은 사업체를 차리고, 벤처 캐피탈의 투자를 받기에 이른다.

 스프링소스는 창업 이후에 계속 수익 모델을 고심 했을 것이다. 과연 어플리케이션 프레임워크만으로 어떻게 수익을 창출할 것인가? 한 때 스프링소스에서는 엔터프라이즈 서비스라는 것을 만들어서 라이선스를 산 고객에게 좀 더 편하게 마이너패치 버전을 제공하는 정책을 고려하기도 했었다[주2].

 그리고 인수 합병을 통해 프레임워크 이외의 제품군도 확보를 한다. 스프링소스가 처음으로 인수한 업체는 Tomcat과 Apache Httpd의 핵심개발자들이 있는 Covalent라는 업체였다[주3]. 프레임워크와 미들웨어를 통합한 서비스를 제공한다는 명분이였지만, 프레임워크만으로는 수익 창출에 한계가 있었기에 그런 결정을 했을 것이다.

  더욱 놀랍게도 스프링소스 자체가 2009년 4억 2천만 불(당시 환율로 5천3백억원 정도[주4]) 의 가격에 VMWare에 인수된다. 100만원짜리 서버 53만대의 가격이니 그 가격도 놀랍지만 무엇보다 인수자가 의외였다. 오라클이나 IBM처럼 자바 개발자가 친숙한 미들웨어를 많이 파는 업체가 아니였고, 자바와는 멀리 떨어진 듯한 가상화 솔루션으로 유명한 업체인 VMWare였기 때문이다.

  로드존슨은 확장성 있는 사업모델을 고민했다[주5]. 컨설팅과 교육만으로는 인건비를 바탕으로 하는 수익 구조가 된다. 그래서 제품과 서비스를 판매하는 사업 모델도 필요했다. 그런데 제품도 상용 버전과 오픈소스 버전의 차이를 지나치게 크게 둔다면 오픈소스 커뮤니티를 홀대한다는 비판을 받을 수도 있다. 스프링소스에서는 Tc server라는 Tomcat에 기능을 강화한 제품을 판매하고 있다. 그러나 Tc Server에는 없는 버그가 Tomcat에는 있다면 많은 비판이 쏟아질 것이다. 제품 판매에도 어느 정도 한계가 있다 보니 더 확장성 있는 사업 모델인 클라우드 서비스와 스프링을 연결시키려고 했을 것이다. 그래서 클라우드 서비스 업체인 CloudFoundry를 인수했고, 결국 VMWare와 손을 잡았다.

  이러한 사업 배경과는 별도로 스프링과 클라우드를 연관시키는 것이 기술적으로 무리 수는 아닌지, 그 연결에 어떤 노력을 하고 있는지 살펴보아야 할 것이다.

**스프링 포트폴리오의 확장과 클라우드**

 VMWare의 인수 이후로, 스프링 포트폴리오의 발전 속도는 더욱 빨라 지고 있다. 최근 [Spring-Android](http://www.springsource.org/spring-android), [Spring-Hadoop](https://github.com/springsource/spring-hadoop)같은 프로젝트를 보면 자바 기술 중 스프링이 건드리지 않은 영역이 바로 생각나지 않을 정도이다. 이 중 클라우드와 관련이 있는 기술들은 아래와 같이 나누어 정리해 볼 수 있다.

  첫째, 대용량 처리 미들웨어를 직접 스프링에서 인수한 것들이다. AMQP(Advanced Message Queuing Protocol) 바탕의 메시징큐 솔루션인 [RabbitMQ](http://www.rabbitmq.com/)와 데이터그리드 기술인 [GemFire](http://www.springsource.com/products/data-management)가 여기에 해당한다. 비동기 처리와 데이터 캐쉬는 대용량 처리가 많은 클라우드 환경에서 더욱 많이 쓰이는 기술이 될 것이다.

  둘째, 분산 저장소에 대한 API의 래핑(Wrapping)을 제공하는 것이다.  
  [Spring-data](http://www.springsource.org/spring-data)라는 프로젝트 아래에서 많은 저장소를 위한 프로젝트들이 진행되고 있다. 이미 Redis, Riak, CouchDB , MongoD, Neo4j등을 지원한다. 물론 꼭 스프링에서 지원해 주지 않아도 그런 저장소를 쓰는 데에는 문제가 없다. 그런 프로젝트의 의도는 JDBC를 바로 사용하는 것보다 Spring-jdbc를 사용하는 것처럼, 저장소 API도 스프링의 설정과 프로그래밍 방식으로 API를 편리하고 일관성 있게 사용하도록 도와 주는 것이다. 물론 저장소 API들이 JDBC만큼 불편하지는 않겠지만, Template-callback 패턴으로 반복된 Try-catch를 줄일 수 있는 부분 등 개선점이 없지는 않을 것이다.

 그리고 스프링 3.1에 추가되는 캐쉬 추상화도 분산 캐쉬 솔루션을 편하게 활용하도록 해 준다. 트랜잭션 처리 방식과 유사하게 애노테이션으로 캐쉬할 대상과 키를 지정하는 기능이다. EhCache에 대한 구현체는 기본 제공되고 SPI(Service Provider Interface)에 맞추면 다양한 캐쉬가 이 스펙에 맞추어서 활용될 수 있다.

 셋째, 모니터링 기술이다. 클라우드 환경에서는 물리적인 서버에 대해서 사용자가 신경을 쓰지 않더라도 실제로 많은 서버와 미들웨어 인스턴스를 쓰게 된다. 그리고 사용량으로 요금이 결정되므로, 리소스 사용량을 지속적으로 확인할 필요도 생길 수 있다. 사용자가 쓰는 패턴을 빨리 파악을 하는 것도 비용 대비 효과를 판단하는데 도움이 된다. 그래서 모니터링이 전통적인 시스템보다 더욱 중요하다. 스프링소스에서 모니터링 도구인 Hyperic을 인수하고 Spring-insight 같은 도구들을 만들고 있는 것이 이에 대한 대응으로 보여 진다. Spring-insight에서는 Spring MVC의 Controller 정보 등 스프링에만 특화된 정보도 제공하는데, 모니터링이 독립적인 영역이지만 스프링을 썼을 때에 이득을 더 부각시키려는 의도일 것이다.

 메시징큐, 데이터 저장소와 캐쉬, 모니터링 기술들은 기존 환경에 설치해서 사용할 수도 있지만,클라우스 서비스에서 제공하는 기술 스택에 포함되기도 한다. 클라우스 서비스에서도 기본으로 제공되는 솔루션들은 중요한 경쟁력이다. 아마존 클라우드에는 메시징 큐로 SQS(Simple Queue Servcie), 저장소로 SimpleDB가 있고, Google App Engine에서는 캐쉬로 Memcache, 저장소로 빅테이블에 바탕을 둔 DataStore를 쓸 수 있다. VMWare가 제공하는 클라우드 솔루션인 VFabric에는 TcServer의 세션 클러스터링을 Gemfire로 쓰고, Hyperic으로 모니터링을 하도록 구성된다.

최근 스프링 포트폴리오의 확장 중 많은 부분은 클라우드 시대에 맞춘 기술 투자라고 볼 수 있다

**스프링과 클라우드 이식성(Cloud Portability)**

 스프링원 2010 컨퍼런스에서는 VMWare의 클라우드에서 돌아가는 애플리케이션이 Google App Engine에서 똑같이 돌아가는 데모를 보여 줬다.  JVM의 이식성을 생각한다면 어떻게 보면 당연한 결과일 수도 있는데, 이것이 어떤 의미가 있을까?

  클라우드를 도입하려는 쪽에서는 미래에 일어날 수 있는 다양한 상황들을 감안해야 한다. 애플리케이션을 레거시 서버에서 클라우드로, 클라우드에서 다른 클라우드로, 클라우드에서 다시 기존 방식의 서버로의 이전하는 모든 경우가 충분히 일어날 수 있다. 그렇다면 애플리케이션이 특정 실행 환경에 종속적인 부분이 많아 지는 것은 애플리케이션의 소유자에게는 큰 짐이 된다. 그래서 특정 클라우드에 애플리케이션이 묶여 버린다는 것은 얼핏 생각하면 클라우드 사업자에게 유리한 것 같지만, 클라우드의 잠재 사용자에게는 초기의 클라우드 도입을 망설이게 해서, 사용자층이 넓어지는데 부정적인 요인이 된다. 클라우드로 이사하는데 드는 비용이 많다면 기존 애플리케이션을 올리지도 않을 것이다. 그리고 이사 한 후에도 빠져 나오기 힘들다면 사용자가 가격 정책 협상에 불리한 위치가 된다. 그래서 클라우스 사업자는 클라우드 사용자가 기존 애플리케이션을 작은 수정으로 클라우드에 올릴 수 있고, 이사 나갈 때도 쉽게 옮길 수 있다는 것은 강조하는 편이 초기에 시장을 넓히는 데에는 유리할 것이다.

 그런데 클라우드 환경이 정말 기존 애플리케이션을 똑같이 받아줄 수 있을까? 우선 웹어플리케이션이 올라갈 WAS부터 그러기가 쉽지 않다. 클라우드 서비스에서는 한정된 종류의 WAS가 제공된다. Google App Engine은 Jetty를 수정해서 쓰고 있고, VMWare가 관여하는 클라우드인 CloudFoundry, VMForce, VFabric은 당연히 Tomcat의 상용판인 Tc Server를 제공하고 있다. 거기다 클라우드에 올라가는 JVM이나 WAS는 지원하는 스펙이 제약된다. Google App Engine에서는 파일을 직접 쓰지 못하고, 쓰레드나 소켓을 생성할 수 없다. 서블릿 스펙에서도 ServletContext.getNamedDispatcher 을 호출해서 디폴트 서블릿의 이름을 알아내는 메소드가 제대로 동작하지 않는다. 보안 문제나 남용의 여지가 있는 부분은 지원하지 않는 것이다. 레가시 시스템을 클라우드 환경으로 옮기는 상황이라면 기존에 레거시 시스템에서 쓰던 WAS와 클라우드 위의 WAS의 종류가 다를 가능성이 높고, 기존의 WAS가 Java EE Server라면 더욱 그렇다. 더욱이 WAS 자체 혹은 애플리케이션에서 호출하는 기능까지도 제약된다.

  여기서 스프링이 클라우드 사업자들에게도 도움이 될 수 있다. 클라우스 소비자에게 지정된 WAS, 그것도 서블릿 스펙만 지원하는 WAS를 제공한다는 클라우스 서비스의 약점을 스프링이 상쇄해 줄 수 있다. 즉, JavaEE 스펙을 지원하는 서버가 없어도 Tomcat만으로도 객체의 라이프 싸이클 관리와 관계 주입, 선언적 트랜잭션 등을 쓸 수 있다. 그리고 JavaEE server를 쓰는 경우라도 서버가 제공하는 데이터 소스, 트랜잭션 서비스들을 스프링을 거쳐서 사용할 수 있다. 스프링을 통해 간접적으로 Java EE 스펙을 쓴다면, 나중에 Tomcat이나 Jetty로 WAS를 바꿀 때에도 어플리케이션의 적은 부분만 수정하면 된다. 예를 들면 JNDI로 데이터 소스를 찾아오는 부분은 DBCP로 바꾸고, JtaTransationManager를 쓰도록 선언된 Bean선언을 DataSourceTransactionManager으로 바꾸는 정도이다. 그렇게 때문에 스프링을 사용한 어플리케이션은 WAS간의 이식성이 높아지고, WAS선택의 폭이 넒어 진다. WAS의 완충 지대 역할이라 할 수 있다.

  한편으로는 스프링을 활용했을 때 WAS 같은 미들웨어에 대한 종속성은 적어지지만 프레임워크 자체에 종속성이 다시 생기기 때문에, 그것 또한 이식성을 줄이는 일이 아니냐는 생각을 할 수 있다. 하지만 굳이 둘 중에 종속성을 가져야 한다면, WAS보다는 프레임워크에 종속되는 편이 더 낫다고 생각한다. 프레임워크는 하나의 WAS 위에서 여러 개가 공존할 수도 있어서 점진적으로 바꿔나가기도 쉽다. 그리고 애플리케이션이 의존하는 부분을 WAS보다는 프레임워크에 두는 것이 유연성 측면에서는 유리하다. 프레임워크의 버전 업그레이드나 특정 라이브러리 변경은 WAS에 대한 업그레이드보다 간편하기 때문이다. jar파일을 바꾸는 일만 생각한다면 프레임워크의 업그레이드는 Maven의 pom.xml에서 버전 선언 몇 줄만 바꿔 주면 된다. 반면 WAS는 설치 자동화가 되어 있다면 간편하게 모든 서버에 한꺼번에 복사할 수도 있겠지만, 아무래도 프레임워크 업그레이드 보다는 부담되는 일이다. 그리고 스프링은 스프링에 종속적이기 않게 코드를 작성하는 방법들을 많이 제공하고 있다. @Inject 같은 표준 애노테이션이 그 예이다. 이를 적절히 활용한다면 프레임워크에 대한 종속성도 다소 덜어낼 수 있다.

 세일즈포스닷컴, 구글은 스프링소스를 소유한 VMWare와 함께 스프링을 통한 클라우드 이식성(Cloud Portability)을 강조하고 있다. 어떻게 보면 경쟁 관계에 있는 이들이 한 목소리를 내고 있는 것은 초기 시장 확대가 무엇보다 중요하기 때문일 것이다. 그리고 사용자들을 자신들의 플랫폼 만으로 가두는 전략보다는 원한다면 오갈 수도 있는 길을 열어 두는 것이 장기적으로는 이득이라는 믿음을 공유하고 있는 것 같다. 그렇게 해도 될 만큼 핵심 경쟁력인 인프라 기술 등에서는 자신이 있다는 해석도 할 수 있다.

**기술 포털로서의 스프링**

 스프링과 클라우드가 연관되고 있는 또 하나의 측면은, 스프링이 자바 기술 생태계에서 일종의 ‘포털’(Portal) 역할을 하고 있어서라고 분석된다.  
인터넷 포털은 많은 사용자들이 방문하고 여러 정보들을 모아서 사용자에게 일관된 UX를 제공한다. CP(Contents provider)사가 컨텐츠를 포털과 제휴하는 것은, CP사 입장에서는 자사의 컨텐츠를 알릴 수 있고, 포털 입장에서는 방문자에게 풍부한 컨텐츠를 제공해서 트래픽을 더 늘릴 수도 있다는 점에서 양측 모두에 이득이 된다. 그리고 사용자는 포털의 일관된 접근 경로와 UX로 다양한 컨텐츠를 접할 수 있다. 예를 들면 네이버의 휘발류 가격 정보는 석유공사에서 운영하는 Opinet에서 데이터를 가지고 오는 것이지만, 사용자는 네이버 검색창을 통해서 다른 컨텐츠를 볼 때와 같은 화면 스타일과 사용 방법으로 컨텐츠에 쉽게 접근할 수 있다.

포털의 UI 일관화

마찬가지로 스프링은 많은 개발자들이 사용하는 기술이고, 다양한 기술들을 조율해서 일관된 설정 방식과 명명 규칙, API 스타일을 제공한다. API는 Application Programming Interface이니 포털의 User interface가 최종 사용자가 보는 화면이라면 개발자들이 보는 interface는 프레임워크와 라이브러리의 API라 할 수 있겠다. 그리고 스프링과 제휴하는 요소 기술의 보유 업체들은 CP사와 같이 자사의 기술을 알려서 사용자를 늘릴 수 있는 기회를 얻는다. 그리고 스프링소스는 스프링과 연결되는 기술 생태계를 더 풍요롭게 만든다는 이득을 얻는다.  
   
스프링은 다양한 기술을 같은 API로 추상화 시켜서 유연성을 주거나, 비슷한 스타일로 정리해서 개발자들에게 초기 학습 비용을 줄여 준다. 예를 들면 데이터그리드인 Gemfire에서 기존의 스프링의 DB 트랜잭션 관리 인터페이스에 맞춘 GemfireTransactionManager를 제공하는 것이 있다. 그리고 Spring-amqp 같이 최근 추가된 프로젝트도 클래스명, 인터페이스명, 메소드명은 스프링에 익숙한 사람이면 처음 보아도 친숙한 스타일을 느낄 수 있다.

VMWare가 스프링소스를 인수한 일이라던지, 세일즈포스닷컴과의 제휴나 Neo4j, Terracotta같은 저장소, 캐쉬 기술과 스프링이 연결되는 것은 기술적인 시너지를 기대한 측면도 있다. 그러나 그것도 스프링이 많은 사람들에게 익숙한 기술이고, 사람들을 모을 수 있기에 성사된 일들이다. 어느 새 VMWare가 자바 개발자들에게 이전보다 친숙한 기업이 되었으니 스프링소스 인수로 홍보 효과는 충분히 성공을 한 듯하다. 물론 그리고 단순한 홍보 수단 이상의 가치가 있으려면, 소비자들에게  시너지를 체감하게 하는 과제가 남아 있긴 하다.

**비침범적인 설계과 클라우드 시대**

  스프링이 처음 만들어 졌을 때 클라우드 시대를 염두에 두었을 리는 없다. 급작스런 시대의 변화에 스프링도 적응을 해야 하는 상황인데, 현재까지는 클라우드 업체 사이에 활발한 제휴 대상이 되고 있다. 인기 있는 기술이라서 홍보를 위해서 제휴를 하는 측면도 있겠지만, 그래도 스프링이 주는 유연성과 이식성이 클라우드에서도 의미가 있다는 평가를 받아서 투자와 협업 대상이 되었을 것이다.

 이런 현상의 가장 근본적인 이유는 인프라성 코드와 업무 로직 코드를 침범적이기 않게 한다는 스프링의 설계 철학에서 비롯된다고 생각한다. 스프링에서 강조하는 POJO(Plain Old Java Object) 방식의 개발은 특별한 규약에 의존이 없는 자바 코드가 핵심 로직을 담당해서 실행 환경에 덜 의존적인 코드를 만든다.

예를 들면 트랜잭션 처리 같은 인프라성 코드는 WAS가 제공하는 JTA(Java Transaction API) 같은 규약을 쓸 수도 있다. 그렇게 특정 미들웨어에 의존적인 코드는 최대한 한 곳으로 모으는 것이 향후 그 미들웨어가 바뀌었을 때 더 적은 수정을 유발한다. 스프링에서는 JTA에 중립적인 트랜잭션 API(PlatformTransactionManager)를 만들고, 이를 AOP를 통해서 사용해서 결국 JTA를 쓰는 코드를 한 곳에 모으고, 한 두줄 수정으로 다른 방식으로도 바꿀 수 있게 되었다. 다른 예로 Spring security가 Google App Engine의 로그인 인증에도 쓰일 수 있는 이유도  사용자 정보 저장소 인프라에 의존하는 부분이 잘 추상화되어 있기 때문일 것이다[주6].

 거창하게 '클라우드 이식성 '을 위해서가 아니고, 시스템의 변화가 있을 때 적은 수정으로 대응할 수 있도록 고려를 하는 것은 설계의 기본이다. 역할과 책임이 잘 구분된 설계는 시대와 환경을 초월해서 의미가 있다. 결국 유연성, 이식성이라는 것은 잘된 모듈화의 일반적인 결과이지 꼭 특별한 기술을 적용해야 얻어지는 것은 아니다. 스프링의 AOP니 Dependency Injection이니 하는 기술들도 결국 그런 일을 돕기 위해서 존재하는 것일 뿐이다. 스프링 이전에도, 스프링 이후에도 이런 모듈화는 중요하고, 스프링이 클라우드 시대에서도 유행하고 있는 것은 이런 보편적인 설계 원리를 잘 지켰기 때문이라고 할 수 있다.

 클라우드 시대에 애플리케이션 개발은 얼마나 달라질까? 사용하는 저장소나 미들웨어 같은 인프라는 많이 달라진다. 클라우드에 들어오는 새로운 구성요소들에 잘 적응을 하는 것이 처음에는 생산성을 결정하는 요인이 될 것이다. 그러나 그렇다고 애플리케이션을 개발하는 방식이 크게 달라진다고는 생각하지 않는다. 잘 모듈화되어서 역할과 책임이 잘 구분된 코드는 클라우드 시대까지 살아남을 것이고, 그런 코드는 클라우드 시대에도 수정, 추가 비용을 적게 하고, 클라우드 이후 시대까지도 남겨 질 것이다. 프레임워크의 코드이든, 응용 개발자가 짜는 애플리케이션의 코드이든 마찬가지로 말이다.

 덧붙여서, 그렇게 관심과 역할이 잘 정리된 좋은 코드를 만들었다는 것을 무엇으로 증명할 것인가? 바로 테스트 코드를 짜 보는 것이다. 인프라를 관리하는 코드와 업무 규칙의 코드가 섞여있다면, 테스트 코드를 짜기가 힘들 것이고, 그런 코드는 앞으로의 변경에도 더 많은 비용이 드는 코드가 되고 살아남기 힘든 코드가 된다. 특정 프레임워크를 썼다고 좋은 설계가 당연히 바로 나오지는 않는다. 스프링을 쓰면서 테스트하기 쉬운 코드를 짜고 있는지를 돌아보는 것이 스프링을 잘 쓰고 있는지를 가장 잘 확인하는 방법이다.

**정리**

 스프링소스가 클라우드와 관련한 활발한 활동들을 하는 것은 확장성 있는 사업 모델을 찾기 위한 돌파구였고, 포트폴리오와 강화와 각종 제휴로 현재까지는 클라우드를 지원하는 기술 쪽으로도 빠른 발전을 보이고 있다. 그리고 스프링이 제공하는 이식성 덕분에 제한된 WAS를 제공하는 클라우드에 유리한 점이 있고, 넓은 사용자층을 가진 기술 생태계의 포털 역할로 그 가치를 인정받은 것으로 분석된다. 그런데 스프링이 클라우드 시대에도 흥행하고 적응하고 있는 가장 근본적인 이유는 애플리케이션의 핵심 로직과 인프라를 담당하는 부분을 구분한 설계를 하는데 도움이 되는 기술이기 때문이다.

  10년전 한 개인이 만들었던 코드조각이였을 뿐인 스프링이 이제는 업계 흐름을 좌우하는 기술이 되었다. 스프링의 예처럼, 오픈소스는 더 이상 잉여시간이 넘치는 개인들의 습작이 아니고, 기업이 전략적 협업을 하는 매개체이다.

 올해에도 스프링의 소식들이 마구 쏟아질 것이고 그 중 상당 부분은 클라우드와 관련이 있을 것이다. 그들의 전략이 자바 생태계의 참여자 모두에게 혜택이 될 수 있는 방향으로 이어지기를 바란다.

**주**

[주1] 관련 내용이 인용된 기사 <http://www.bloter.net/archives/15878> . 물론 스프링 기술 중 작은부분만 활용하고 있어도 스프링을 쓰는 것으로 집계되었을 것이라고 생각된다.  
[주2] 당시 논의 되었던 라이선스 정책에 대해서는  <http://toby.epril.com/?p=440> 에 자세히 설명되어 있다.  
[주3]로드존슨은 Covalent 인수 배경에 대해서 아래와 같이 밝히고 있다.  
<http://blog.springsource.com/2008/01/29/some-decisions-are-easy-%E2%80%93-like-springsource-acquiring-covalent/>  
[주4] 인수 가격과 원화 환산 금액은 아래 링크를 참조했다  
<http://younghoe.info/1192>

[주5]로드존슨의 사업 모델에 대한 고민은 아래에 발표자료에 언급되어 있다.  
<http://gotocon.com/dl/jaoo-brisbane-2010/slides/RodJohnson_bKeynotebThingsIWishIdKnown.pdf>  
[주6] <http://blog.springsource.com/2010/08/02/spring-security-in-google-app-engine/> 참조

[spring](http://blog.benelog.net/tag/spring), [cloud](http://blog.benelog.net/tag/cloud), [스프링](http://blog.benelog.net/tag/%EC%8A%A4%ED%94%84%EB%A7%81), [클라우드](http://blog.benelog.net/tag/%ED%81%B4%EB%9D%BC%EC%9A%B0%EB%93%9C)