공학
트렌드

소프트웨어공학센터 경영지원TF팀

JDK/WEB/WAS 서버 표준화와 오픈 소스 거버넌스
Part 1 : JDK/WEB/WAS 서버 표준화

2014. 10. 21. [제110호]

Contents

정보통신산업진흥원 부설
nipa 소프트웨어공학센터

Ⅰ. 배경

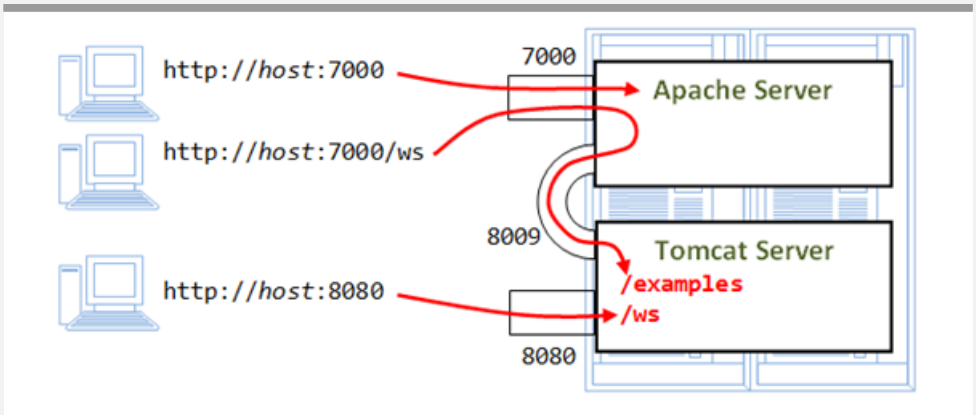
필자는 WEB 서버와 WAS 서버의 고도화 표준을 필자가 다니던 회사에 적용하였다. 그리고 이를 바탕으로 서버에서 배포된 WAS Library를 조사하여 오픈 소스 거버넌스를 진행했다.

WEB/WAS 서버란 ?

WAS (Web Application Server) 서버는 Apache Tomcat 이나 Jetty, Glassfish 서버와 같이 비즈니스 로직(DB를 주로 연결)을 포함한 서버를 말한다. WEB 서버는 WAS 서버 앞에 위치하여 Load를 분산하거나 WAS 일부 서버 장애 시 다른 WAS 서버로 패킷을 전달하여 웹 서비스에 문제가 없도록 하는 서버를 말하며 Apache Httpd나 Nginx를 사용한다.

〈그림 1〉은 기본적인 Aapche Httpd 서버와 Apache Tomat 서버 배치 구조를 보여주고 있다.

그림 1_전통적으로 쓰였던 Apache Httpd 서버와 Tomcat 서버



출처: http://www.ntu.edu.sg/home/ehchua/programming/howto/ApachePlusTomcat_HowTo.html

오픈 소스 거버넌스(Open Source Governance)란?

오픈 소스 라이선스(License)를 사용하는데, 법적 문제가 없는지 체크하는 것 뿐 아니라 보안 및 성능 이슈를 모두 포함한 체크를 진행하는 것을 의미한다.

필자가 소속해 있던 회사는 웹 서비스에서 사용하고 있는 Apache Http / Nginx 서버나 Tomcat 서버가 많았기 때문에 작업이 필요한지 확인하고, 웹 서비스를 개발하고 운영하는 개발자들이 WEB/WAS 서버를 사용함에 있어서 불편한 점이 있는지 확인했다.

그래서 대표적인 웹 서비스 개발자들과 인터뷰를 진행하고, 성능 Benchmark Test를 진행하였다.

첫 번째 인터뷰 시 개발자들이 운영에 관련된 불편함을 가장 많이 호소했다. 운영 부담을 줄일 수 있는 요구사항들로는 설치와 설정과 관련된 개선 사항이 대부분이었다. 설치와 설정에 관련된 내용은 다음과 같다.

1. WEB/WAS에 대한 설치, 설정 표준 가이드가 있으면 좋겠다.
2. WEB/WAS 서버의 설정, 선택 등에 대해서 물어보고 확인할 수 있는 조직이 있으면 좋겠다. 설정 전문가가 필요하다. 설치 및 설정, 가이드, 교육, 결함 리포팅, 버전 업그레이드 권장 조직이 필요하다.
3. WEB/WAS 배포 설정을 표준화하고 개발자들이 배포 설정 작업을 최소한으로 하면 좋겠다.
4. 소프트웨어 업그레이드 할 때에 보안/성능 검증 절차 및 시간이 소요되기 때문에 공통적인 검증 절차를 개발팀이 아닌 거버넌스 조직이 알아서 해주면 좋겠다.
5. WAS 운영에 대한 표준화 된 정책을 통해서 다른 팀으로 전배를 갈때, 전배 갈 팀의 시스템/환경설정 부분에 대한 Learning Curve를 최소화하면 좋겠다.

이렇게 하여 운영에 최소한의 시간을 투자하고 개발에만 전념해서 생산성이 향상되면 매출 향상에도 도움이 될 것이라는 기대감이 존재했다.

두 번째 성능 Benchmark Test를 통해 성능과 웹 서비스에서 많이 사용하는 빈도를 바탕으로 WAS는 Tomcat을 선택하고 WEB서버는 Nginx, Apache Httpd 서버를 선택했다.

그 동안 JDK와 Apache HTTP/Tomcat 서버의 사용 버전, 설치 방법, 사용하는 계정을 리눅스를 잘 아는 개발자에 의존하다 보니, 서비스 부서별로 상이한 환경들이 구축되었다. 환경을 구축한 개발자 대신 투입되는 또 다른 개발자는 새로운 환경을 파악해야 하고, 환경 설정을 익히는데 많은 시간을 소요 할 수밖에 없다. 또한 리눅스를 이해하지 못하는 경우 리눅스에 익숙해지기까지 많은 시간을 할애하게 된다.

또한, 설치 및 설정 배포를 할 때에 개발자가 터미널 프로그램을 이용해서 수작업으로 일일이 서버 한 대씩 접근해서 진행하기 때문에 수정하면서 문제가 발생하는 번거로

움이 많았다. 그리고 긴급 증설 또는 장애 처리와 같은 긴급 이슈들은 개발하는 동시에 진행하기 때문에 대응 속도가 떨어질 수 있었다.

환경 표준화가 되어 서버 설치를 자동화할 수 있다면 서비스 운영비용을 최소화할 수 있다. 서버 설치비용을 위해서 리소스를 따로 쓰지 않아도 되고, 신규 부서에 배정되거나 업무 이동할 때 개발자의 시스템 적응 비용이 적게 든다.

또한 오픈 소스 Library Governance가 된 검증된 버전을 사용하기 때문에 안정성을 확보할 수 있다. 그리고 빌드 / 배포 시스템을 통한 설정 배포를 표준화 디렉토리를 활용하는 까닭에 신속하게 대응할 수 있게 했다. 이를 위해서는 표준 디렉토리를 지정하도록 하였다. 이 디렉토리를 활용하여 웹 서비스의 Library의 정보를 파악하고 서비스별, 호스트별 정보를 수집하도록 하였다. 만약 보안 이슈와 같이 Critical Library를 발견하면 관련 개발자와 조직장에게 알려줄 필요가 있었다.

기존에 발생된 장애 케이스를 분석하고 보안 문제가 발생되지 않도록 문서에 추가해 유용성을 높였고, 운영에 많은 도움이 되는 스크립트를 사용할 수 있도록 데이터 저장소를 활용했다. 웹 서버를 운영하면서 기본적으로 활용될 로그(Log)관리, 웹 서버 재시작(Restart)과 관련된 스크립트 등 운영에 필요한 도구들을 지원하면 웹 개발자가 편안하게 사용하도록 할 수 있다. 웹 서비스를 운영하는데 필요한 스크립트들은 거의 쓰임새가 비슷하기 때문에 하나로 통일된 스크립트를 이용하면 더욱 편하고 효과적으로 운영할 수 있다.

필자는 표준 설치/ 설정 가이드 문서를 작성했다. 표준 설치/설정 가이드 문서는 단순한 설치 표준화와 설정의 가이드 뿐 만 아니라 그 동안 발생했던 보안 문제 또는 장애 패턴을 찾아내고 자주 활용하는 스크립트를 공유해서 개발자들의 생산성을 최대로 높이도록 했다.

II. 장애 사례

많은 웹 서비스 장애들을 처리하면서 장애가 중복적으로 발생한 시스템 류의 이슈들을 모두 정리했다. 필자가 웹 서비스 운영 시 장애 처리했던 경험들을 바탕으로 정리한 것이다. 그 중에 대표적인 것들을 본 원고에서 소개하고자 한다.

시스템

- o WEB 또는 WAS 서버에서 사용하는 처리하는 소켓의 개수가 시스템 설정 파일 오픈 개수보다 많아지면 장애가 발생한다. 프로세스 당 최대 파일 오픈 개수를 리눅스 시스템에서 정의한 디폴트 값 보다 커야 한다.
- o JDK 버전을 업그레이드 하는 과정에서 JAVA_HOME 환경 변수 설정이 과거의 것으로 사용했었다.
- o 오래된 JDK 버전과 하위 Linux kernel 버전에서 메모리 부분 관리가 제대로 안되어 WAS 서버가 제대로 서비스를 하지 못했다.
- o CSH 사용 시 CPU가 100%에 도달하는 일이 발생했다.

WEB (Apache Http)

- o Log 디렉토리 권한이 Root로 되어 있어서 파일을 사용하지 못해 WEB 서버가 실행이 되지 못했다.
- o 로그 파일이 커질 때 Rotation 하지 않아 Disk Full 이슈로 Write가 되지 않아 WEB 서버 장애가 발생했다.
- o mod_jk 버전 업그레이드 후 오동작 했다.
- o mod_jk 버전의 Timeout 미 설정으로 오동작 했다.
- o mod_jk 설정 오타로 오동작 했다.
- o WEB 서버의 성능 이슈 고려 없이 Keep Alive 를 On으로 설정하여 클라이언트의 요청 트래픽을 소화하지 못하고 장애로 이어졌다.
- o 한글 인코딩 문제로 웹에서 한글이 깨졌다.
- o mod_proxy 설정 이슈로 오동작 했다.

WAS (Apache Tomcat)

- o 톰캣 쓰레드 개수를 너무 적게 잡아서 트래픽이 많아졌을 때 웹 서비스를 하지 못했다.
- o L4 또는 L7 health check 파일을 잘 활용하지 못해서 http 패킷을 잘 처리하지 못했다.
- o 로그 파일이 커질 때 Rotation 하지 않아 Disk Full 이슈로 Write가 되지 않아 WAS 서버 장애가 발생했다.
- o war 파일 배포 하자마자 재시작이 되어 classloading 이슈가 발생하여 서비스 장애를 일으켰다.
- o post 요청을 처리할 때 size를 잘 결정하여 너무 큰 요청에 대해서는 WAS가 hang에 빠질 수 있다.

o Session Timeout이 기본 30초로 되어 있는데, 이 때문에 GC(Garbage Collection)가 너무 자주 일어나 서비스 품질이 떨어질 수 있다.

JDK (Java Development Kit)

o JDK 버그가 특정 배포 버전에서 발생한다. 이유 없이 WAS 서버가 메모리 이슈로 장애가 발생한다.

o 긴 GC (Garbage Collection) 발생 시 응답 속도 품질이 떨어지거나 장애로 이어질 수 있다.

o 메모리릭(Memory Leak)이 발생 시 점차 시간이 지나가면 메모리가 부족하여 장애로 이어진다.

WAS 내 디버그(Debug)들은 대부분 처리할 수 있지만, JDK, WEB, WAS 에서 발견되는 이슈는 Linux 시스템의 영향을 많이 받거나 설정 이슈들로 인해서 장애로 이어질 수 있는 부분이 많이 발생할 수 있었다. 필자의 경험을 토대로 많은 이슈들을 쉽게 처리하기 위해서는 기본적인 템플릿을 제공함으로서 최소한의 장애를 예방하는 것이 좋은 방법이라고 생각했다.

Ⅲ. 표준화 내용

필자는 WEB/WAS/JDK 표준 디렉토리 설치 내용과 Apache Httpd 서버와 Tomcat 서버 간의 설정 가이드, Nginx 서버와 Tomcat 서버 설정 가이드, 체크 리스트, 템플릿등을 포함한 내용을 작성하였다.

표준 리눅스 서버를 선택했고, 단일된 계정을 하나 생성해서 그 계정으로는 sudo 권한을 줄 수 있도록 하였다. 리눅시 서버에 접속할 때는 LDAP 계정을 통해서 접근하고 다시 표준 계정으로 접속하여 작업을 할 수 있도록 표준화 했다.

안정화 된 버전인 JDK/Web/WAS 를 미리 설치하였다. JDK, nginx, apache httpd 등을 /apps/ 디렉토리에 설치하도록 했고, 안정화 된 버전으로 먼저 설치한 후, current 라는 Symbolic Link를 걸어 현행화 된 디렉토리임을 명시적으로 표시했다.

```
/apps/jdk
  /apps/jdk/current   -->  oracle-jdk-1.6.0.45
  /apps/jdk/oracle-jdk-1.6.0.45

/apps/web/nginx
  /apps/web/nginx/current  -->  nginx-1.7.1
  /apps/web/nginx/nginx-1.7.1

/apps/web/apache
 /apps/web/apache/current  -->  apache-2.4.9
 /apps/web/apache/apache-2.4.9
```

apache 또는 nginx 의 libs 디렉토리에는 이미 자주 쓰고 잘 활용하는 모든 모듈들을 다 포함시켰다. 이를 통해서 따로 설치하는 작업을 하지 않도록 하였다.

한편 WAS 인 tomcat의 경우는 멀티프로세스 상에서 여러 개의 프로세스로 실행해야 하기 때문에 /apps/서로 다른 디렉토리를 설치한다.

```
/apps/was/tomcat
 /apps/was/tomcat-7.0.50-1
 /apps/was/tomcat-7.0.50-2
 /apps/was/tomcat-7.0.50-3
  ...
```

그리고, 자주 사용하는 운영 스크립트를 모아 /apps/scripts에 저장했다. 자주 사용하는 WEB/WAS 재시작 스크립트, L4/L7 스크립트, 인증서 시간 체크, 문법 체크, ACL 체크 등을 모두 한 번에 모아두었다.

```
/apps/scripts
```

이외 다양한 유틸리티들은 /apps/util 에 저장하도록 하였다.

```
/apps/util
```

물리 서버를 웹 서비스 장비에 활용 시 표준 리눅스 장비에 표준 설치 구조를 지정하고 안정된 버전을 가이드 했다. 그리고 특정 계정으로 권한 관리를 하여 이슈가 없도록 하였고, 설정 등을 가장 성능이 잘 나오는 설정으로 템플릿화 했다.

그리고 내부 Repository에 JDK, WEB, WAS, 유틸리티, 소프트웨어, 설정 파일 등을 모두 모아서 관리를 하였다.

필자는 2장의 장애 내용과 개발 / 운영 경험을 바탕으로 표준 문서 및 가이드를 작성했다. 대략적인 내용을 소개한다.

시스템

- o bash shell 을 기본으로 사용하도록 한다.
- o 안정적인 linux kernel과 gcc/glibc 버전을 표준화 한다.
- o bash history 설정을 두어 어떤 명령어를 썼는지 투명하게 관리한다.
- o 주요 계정 접근 시 시스템 로그를 잘 저장하고 모든 서버의 로그를 하나의 서버로 저장한다.
- o psacct 가동을 통해서 계정 실행 기록 로그를 남기게 한다.
- o ubuntu일 때는 apt-get을, cent os 일 때는 yum 설치 환경을 제공한다.
- o 중요 Utility는 모두 path에 저장하여 실행이 쉽도록 한다.
- o WEB 또는 WAS 서버를 포함하는 모든 프로세스에서 사용하는 처리하는 소켓의 개수를 최대치로 설정했다.
- o JAVA_HOME 환경 변수는 /apps/jdk/current로 지정한다.

WEB (Apache Http)

- o ipcrm 명령어를 포함한 스크립트를 운영 툴에 추가했다. 일부 Apache Http Server의 shared memory segments 가 발생하지 않도록 했다.
- o Log 디렉토리 권한을 주요 계정에서 쓸 수 있도록 한다.
- o 로그 파일 커질 때를 대비하여 파일 rotation을 하는 스크립트를 crond에 등록 하여 disk full 이슈가 발생하지 않도록 한다.
- o mod_jk 버전은 안정적인 버전으로 통일한다.
- o mod_jk 버전의 timeout 을 설정한다.
- o Keep Alive 를 Off를 기본 설정으로 변경하고 필요할 때 수정하도록 한다.
- o 한글 인코딩은 기본 UTF-8로 지정한다.
- o mod_proxy 설정 이슈가 없도록 template을 제공한다.
- o maxclient 값을 최대로 한다.
- o Etag 캐쉬 기능, 압축 전송(mod_defalte)을 활용한다.
- o 불필요한 디렉토리, directory list는 모두 삭제한다.
- o 보안 상 version을 response에 저장하지 않는다.
- o server status(mod_jk포함), WAS의 내부 디렉토리 구조를 보이지 않게 하거나 내부에서만

보게 한다.

o error (4xx, 5xx) 발생 시 error document 템플을 사용한다.

WAS (Apache Tomcat)

o 톰캣 쓰레드 개수를 크게 잡는다. (필요한 공식을 제공하여 WEB 서버와 WAS 서버 간의 thread를 지정할 수 있도록 가이드를 제공한다.)

o L4 또는 L7 health check 파일을 잘 활용할 수 있도록 스크립트로 제공한다.

o 로그 파일 커질 때 rotation 하도록 crond에 추가하여 disk full이 일어나지 않도록 한다.

o war 파일 배포 하자마자 또는 web.xml 파일을 수정 하자마자 WAS가 재시작이 되지 않게 하고 스크립트를 통해서 명시적인 재시작을 할 수 있도록 제공한다.

o maxPostSize는 post 요청 시 size를 미리 템플릿 파일에 지정할 수 있다. 주어진 size보다 큰 요청은 Exception이 발생한다. 만약 제한이 없다면 0으로 지정한다.

o Session timeout이 기본 30초로 되어 있는 것을 최소 값인 1초로 수정한다.

o tomcat 7부터 추가된 불필요한 GC는 하지 않도록 설정(JreMemoryLeakPrevetionListener) 값을 변경한다.

o Cookie 제한 크기를 높게 설정하여 Cookie 크기 문제가 나지 않도록 한다. maxHTTPHeaderSize 값 조절을 통해 Cookie 이슈가 없도록 발생한다.

o admin 디렉토리와 같이 외부에 오픈 될 수 있는 디렉토리와 파일은 모두 삭제한다.

o 8080 과 같은 개발 전용 접근 포트를 외부로 오픈하지 않도록 한다.

o WatchedResource, reloadable 설정을 이용하여 중요 파일 변경에 따른 자동 배포를 설정할 수 있다. 자동 재시작으로 에러가 발생할 수 케이스라면 주의해서 사용한다.

o WEB 서버와의 통신 시 connectTimeout을 지정하여 hang(무한 대기 상태)이 되지 않도록 한다.

JDK (Java Development Kit)

o JDK 버그가 없는 안정적인 버전을 가이드 한다.

o 긴 GC (Garbage Collection) 발생 시에는 thread dump를 발생시켜 로그를 남겨 개발자들이 분석한다. 계속 발생하면 L4/L7 장비에서 제외시켜 WAS의 메모리 덤프를 분석할 수 있는 툴들을 제공한다.

o 메모리 릭(Memory Leak)이 발생할 것이라는 패턴을 찾게 되면 개발자와 운영자에게 자동으로 알린다.

Ⅳ. 결론

필자의 경험을 토대로 이런 작업들을 원활하게 하기 위한 방안들은 다음과 같다. 4개의 전문가 집단과 회사 내 플랫폼 리더십이 필요하다. 설치는 시스템 (또는 인프라) 엔지니어의 도움을, 설정 배포는 배포 시스템 엔지니어의 도움을, 보안 부분은 보안 엔지니어의 도움을, 설치/설정을 주도하는 플랫폼 개발자의 도움을 받는 것이 중요하다.

소프트웨어 특성상 계속 변화하고 있기 때문에 꾸준히 최신 버전을 활용하면서 문제가 발생하는지 파악하고, 전파할 수 있어야 한다. 또한 끊임없이 퍼지는 중요 보안 Library 및 웹 서비스 모듈의 보안성을 늘 체크하도록 한다. 그리고 장애를 일으키는 원인이 시스템 및 Library나 소프트웨어일 경우에는 이에 대한 이슈 전파의 중요성과 필요성을 더욱 절감할 수밖에 없다.

안정적인 버전인 소프트웨어를 쓰도록 권면하도록 한다. 이 또한 애매한 부분이 있어서, 어떤 버전이 안정적인 버전인지 체크하고 설치되어야 하며 개발자/운영자는 잘 숙지하고 진행하도록 해야 한다.

필자는 이런 과정을 통해 표준화 작업을 진행 후, 오픈 소스 거버넌스 (Open Source Governance) 작업을 진행할 수 있었다. 결론적으로는 오픈 소스 거버넌스를 통해 JDK/WEB/WAS 표준화 프로세스가 보완하는 구조를 가지고 있다는 것을 볼 수 있다. 이에 Part 2에서는 오픈 소스 거버넌스에 대해 알아보도록 하겠다.