
정보시스템 분야 Stack 통합 Test 결과보고서 [Docker]

2015. 04.

목 차

I. Stack 통합 테스트 개요	1
1. 목적	1
II. 테스트 대상 소개	2
1. Docker 소개	2
2. Docker 주요기능 및 특징	3
3. Docker 시스템 요구사항	7
III. Stack 통합 테스트	8
1. 테스트 환경	8
2. 테스트 접근 방법	9
3. 테스트 수행	11
4. 테스트 수행 결과	12
IV. 종합	13
※ 참고자료	14
[별첨1] Docker 테스트 케이스	

I. Stack 통합 테스트 개요

공개SW Stack 통합테스트는 여러 공개SW들의 조합으로 시스템 Stack을 구성한 후 Stack을 구성하는 공개SW의 상호 운용성에 중점을 두고 기능 및 성능테스트 시나리오를 개발하여 테스트를 진행한다.

본 통합테스트를 통해 안정적인 Stack 정보를 제공하여 민간 및 공공 정보시스템 개발 및 도입 시 활용될 수 있도록 한다.

1. 목적

□ 공개SW Stack 통합 테스트 수행 목적

- 공개SW로 구성된 Stack이 유기적으로 잘 동작함을 확인
- 다양한 Stack 구성에 기반을 둔 테스트를 통해 안정된 Stack 조합 규명
- 공개SW 시스템 도입을 위한 Stack 참조모델의 신뢰성 정보로 활용
- 공개SW의 신뢰성과 범용성에 대한 사용자 인식 제고

II. 테스트 대상 소개

1. Docker 소개

Docker는 2013년 3월 Docker, Inc(구 dorCloud)에서 출시한 오픈 소스 컨테이너 프로젝트이다.

클라우드 환경에서 다수의 리눅스/유닉스 서버 환경을 설치 및 배포하는 시스템에 대한 어려움과 한계를 극복하기 위해 Immutable Infrastructure 패러다임이 나왔고, 이를 토대로 구현된 Docker는 호스트 OS와 서비스 운영 환경(서버 프로그램, 소스 코드, 컴파일된 바이너리)을 분리하고, 한번 설정한 운영 환경은 변경하지 않고 계속 사용할 수 있는 Immutable 환경을 제공한다.



[그림 1. Docker 시스템 구성 개념도]

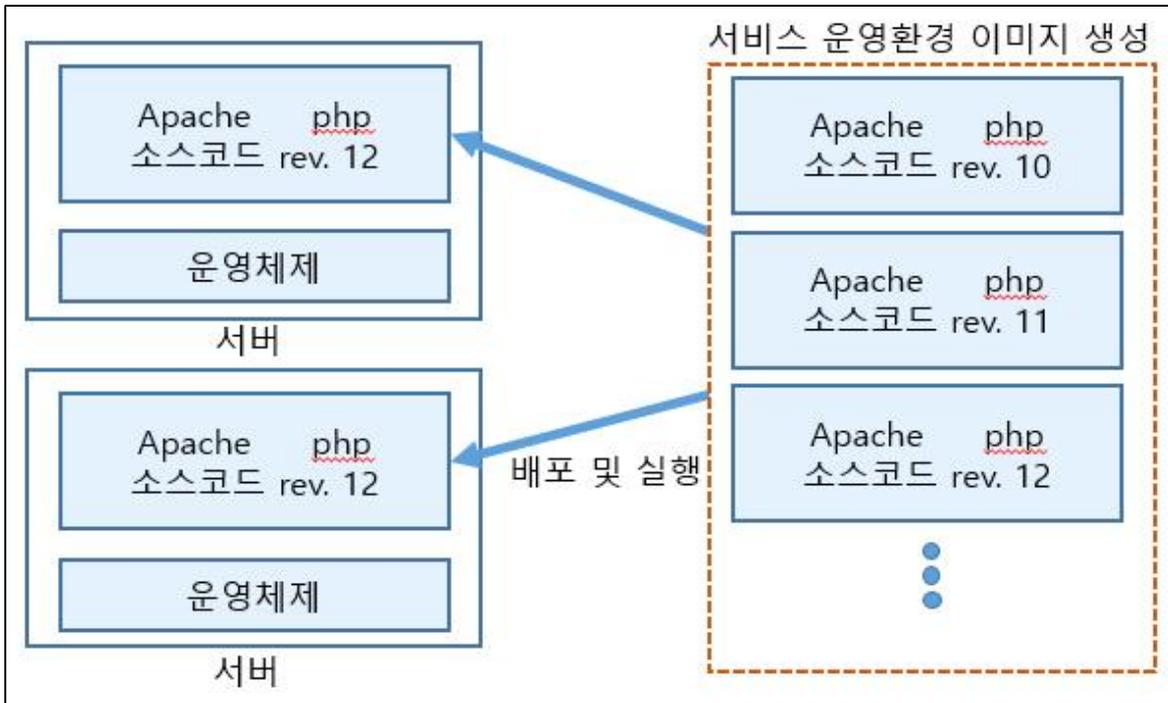
2. Docker 주요기능 및 특징

Docker는 운영체제 레벨 가상화(Operating system-level virtualization)의 기능 구현 단위인 컨테이너 안에 있는 애플리케이션 배포를 자동화하는 오픈 소스 엔진으로 Immutable Infrastructure 패러다임이 적용되어 서비스 운영 환경 이미지를 클라우드 플랫폼 상에서 자유로이 생성, 배포 및 운용할 수 있는 사용 환경을 제공한다.

□ Docker 주요 개념

(1) Immutable Infrastructure

- > 편리한 관리 : 서비스 운영 환경을 이미지로 생성했기 때문에 이미지 자체만 관리하면 됨 (이미지의 중앙 관리를 통해 체계적 배포 및 관리 가능)
- > 확장 : 이미지 하나로 서버를 계속 구성 가능/ 클라우드 플랫폼의 자동 확장(Auto Scaling) 기능과 연동 가능
- > 테스트 용이성 : 개발자의 PC나 테스트 서버에서 이미지를 실행하기만 하면 서비스 환경과 동일한 환경이 구성되기 때문에 테스트가 매우 용이함
- > 호환성(Portability) : 운영체제와 분리된 운영 환경 구성이 가능하므로 어디서든 가볍게 실행 가능한 환경 제공

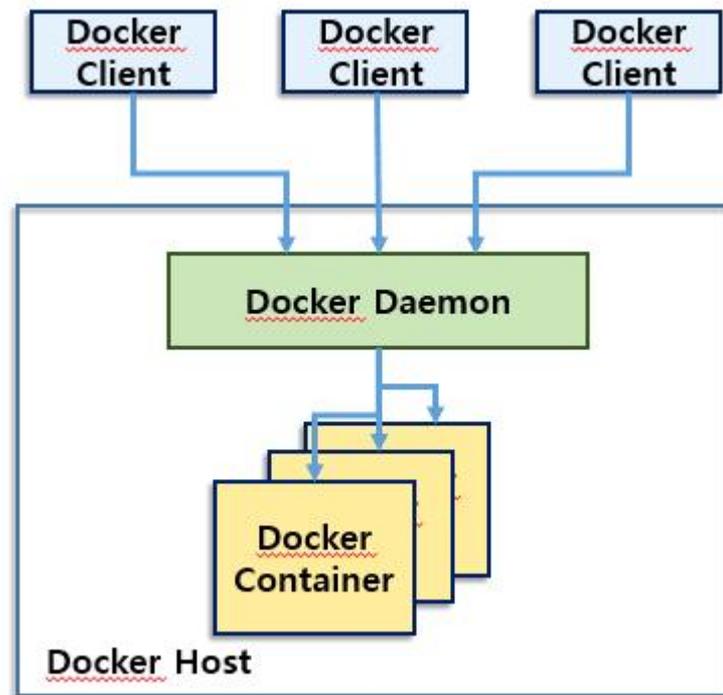


[그림 2-1. Immutable Infrastructure 개념도]

□ Docker의 주요 기능요소 및 특징

(1) 도커 클라이언트와 서버

도커는 클라이언트-서버 애플리케이션이다. 도커 클라이언트는 도커 서버 혹은 데몬과 통신하며 교대로 작업한다. 커맨드 라인 클라이언트 바이너리인 docker와 소스 코드 내에서 도커를 사용할 수 있는 API가 제공되어 하나의 호스트(머신)에서 도커 데몬과 도커 클라이언트를 실행할 수도 있고 다른 호스트에서 실행 중인 리모트 데몬에 로컬 도커 클라이언트를 리모트 연결할 수도 있다.



[그림 2-2. Docker Architecture]

(2) 도커 이미지

도커 이미지는 여러 레이어(layer)로 나뉜 형태로 되어 있으며, 소스코드로서 컨테이너를 런칭(Launching)하는 역할을 한다. 또한, 파일 추가, 명령어 실행, 포트 오픈과 같은 체계적 명령어로 구성되며 공유, 저장, 업데이트 기능을 통해 높은 호환성을 제공한다.

(3) 레지스트리

도커는 구축한 이미지를 레지스트리에 저장한다. 운용 형태에 따라 퍼블릭(public)과 프라이빗(private) 타입으로 구분되며, Docker, Inc는 퍼블릭 레지스트리로 도커 허브(Docker Hub)를 제공한다.

(4) 컨테이너(Container)

화물(cargo)의 개념으로 소프트웨어 이미지를 포함하며 이 소프트웨어

이미지는 실행될 오퍼레이션의 집합으로 구성된다. 예를 들어, 이미지는 생성되고, 시작되며, 중단되고, 재시작되며 삭제될 수 있다. 개발한 애플리케이션과 서비스를 패키징하여 배포할 수 있도록 한 컨테이너 구조는 호스트 OS상에서 동작하는 Docker 엔진을 통해 도커의 높은 시스템 호환성을 제공한다.

3. Docker 시스템 요구사항

□ Docker 시스템 요구사항

OS 구분	정보	필요 환경 요소
Ubuntu	- 12.04 LTS 이상 권장	- 커널 (x86_64 3.8.0 이상 버전) - 디바이스 매퍼
CentOS/ Fedora	- CentOS 6+ 버전 (64비트) - Fedora core 19+ (64비트)	- 커널 (x86_64 3.8.0 이상 버전) - 디바이스 매퍼
OS X	- 최신 안정버전 권장	- Boot2Docker (VirtualBox, Docker client 포함)
Windows	- 최신 안정버전 권장	- Boot2Docker (VirtualBox, Docker client 포함)

[표 1. Docker 시스템 요구사항]

※ 상기 시스템 요구사항 항목 중 Ubuntu 호스트 OS환경에서 UFW(Uncomplicated Firewall)가 적용될 경우, 네트워크 브릿지(network bridge)를 통해 모든 포워드 패킷을 드롭(drop)시키는 default 설정을 허용(Accept)으로 수정 적용이 필요함 (/etc/default/ufw 파일 내 DEFAULT_FORWARD_POLICY="DROP" 항목의 설정 값을 "ACCEPT"로 수정)

※ 관련 상세 정보는 아래의 링크 내역 참조

[1] 기능/특징 세부

- <http://docs.docker.com/>
- <https://www.docker.com/tryit/>

[2] 설치/ 유저가이드

- <http://docs.docker.com/#about-this-guide>
- <https://groups.google.com/forum/#!forum/docker-user>

[3] 공식 Repository

- <https://registry.hub.docker.com/>

III. Stack 통합 테스트

1. 테스트 환경

Docker SW 환경

SW	Version
Ubuntu	12.04 LTS (64비트)
Docker	1.6.2
kernel	3.13.0-52-generic

[표 2-1. 테스트 SW 환경]

※ Ubuntu 환경 구성 시, 설치버전에 상관없이 64비트 체계로 설치되어야하며, kernel은 최소 3.10 버전 이상 조건을 만족하여야 함

Stack 환경

Stack	OS	네트워크 정보 (IP)
A (Server)	Ubuntu 12.04 LTS	121.162.249.93
A (Client)	Ubuntu 12.04 LTS	121.162.249.94

[표 2-2. Stack 환경]

HW 환경

제조사	모델명	CPU	MEM	Disk	NIC
HP	dc7900 CMT	Quad-Core 2.66Ghz~4P	3.6GiB	265GB	Gigabit 1Port

[표 2-3. HW 환경]

※ 동일 사양 HW조건외 PC 2대로 서버, 클라이언트 환경 구성하여 테스트 진행

2. 테스트 접근 방법

Docker는 클라우드 가상 플랫폼으로 컨테이너를 런칭하는 도커 이미지, 운용 형태에 따라 구분 관리되는 레지스트리, 애플리케이션과 서비스를 패키징하여 배포 관리 가능한 환경을 제공하는 컨테이너를 주요 기능 요소로 구성된다.

본 테스트는 서버, 클라이언트 환경을 구성하여 도커의 설치 및 운용에 따른 사용자 시나리오를 토대로 Docker 주요 기능 요소의 기능 신뢰성을 확보하며, 환경 구성의 안정성을 확인하는 데 주된 테스트 방향성과 목적을 두고 수행되었다.

□ 탐색적 테스트(Exploratory Testing)

탐색적 테스트는 테스트 엔지니어의 지적 능력을 최대한 공유, 활용하는 것을 목적으로 하는 테스트 접근법으로 테스트를 수행할 대상을 실행시켜 사용함과 동시에 사용 측면에서 문제가 되는 부분에 집중하여 테스트를 설계 및 계획한다.

이러한 과정은 효율적 진행을 위한 Time Boxing을 통해 수행되므로 테스트 케이스 작성을 최소화할 수 있고, 상대적으로 적은 시간에 집중적인 테스트를 가능하게 한다.

□ 리스크 기반 테스트(Risk based Testing)

리스크 기반 테스트 기법은 테스트 대상에 비해서 테스트 자원이 부족한 경우 효과적이고, 효율적인 테스트 수행을 위해 적용 될 수 있다. 해당 기법은 크게 리스크 식별과 리스크 분석, 그리고 리스크 계획의 세 단계로 구분 진행된다.

리스크 식별 단계에서는 제품 품질관점에서 테스트 대상이 될 항목을 식별하고, 프로젝트나 제품에 대한 리스크 요소를 식별한다.

리스크 분석 단계에서는 장애 발생가능성과 장애로 인한 영향을 식별하고 리스크 우선순위를 결정한다.

마지막으로 리스크 계획 단계에서는 리스크의 우선순위에 따른 대처 방안 및 완화 정책을 수립하며, 이후 테스트 수행 시 커버리지를 고려하여 선택과 집중을 통해 테스트를 수행하게 된다.

□ 시나리오 테스트

시나리오 테스트 기법은 단일 기능에 대한 결함 여부를 확인하는 것이 아니라, 서로 다른 컴포넌트 사이의 상호작용과 간섭으로 발생할 수 있는 결함을 발견하기 위한 기법이다.

3. 테스트 수행

테스트 수행 관련 세부 내역은 별첨 「Docker 테스트케이스」 문서를 참고한다.

□ 탐색적 테스트 현황

이하, 테스트 현황의 차터 항목 구분은 각 기능 아이템 항목 별 세부 구분 항목으로서 사용자 시나리오를 기반으로 정의되었다.

기능 아이템	기본 차터	테스트 아이디어
Docker 기본환경 관리	1	8
Docker 구성 관리	2	14
Docker 운영 관리	3	19
합 계	6	41

[표 3. 테스트 아이디어 현황]

- 기본환경 관리 카테고리에서는 대상 SW의 설치 및 기본 환경구성 측면에서 안정성 확인 및 컨테이너 운용환경 접속 기능을 중심으로 테스트 진행
- 구성 관리 카테고리에서는 컨테이너 구성 관리 및 Docker 이미지 관리 기능을 중심으로 테스트 진행
- 운영 관리 카테고리에서는 Docker 컨테이너의 구동 및 인증 적용에 따른 접속기능과 레지스트리 구성 안정성 확인을 주목적으로 테스트 진행

4. 테스트 수행 결과

Docker 테스트를 수행한 결과 기능 카테고리 별 정리된 내용은 아래와 같다.

분류		PASS	FAIL	N/T	N/A
기능	테스트 아이디어				
Docker 기본환경 관리	8	8	0	0	0
Docker 구성 관리	14	14	0	0	0
Docker 운영 관리	19	19	0	0	0

[표 4. 테스트 결과]

- 컨테이너 환경구성 시 우분투 버전 별 (12.04 LTS, 13.10, 14.04 LTS) 이미지 환경을 적용하여 기능 카테고리 구분에 따른 세부항목들에 대하여 비교 확인 진행함
- Docker로 서버, 클라이언트 환경을 구축하고 로컬 및 원격 레지스트리를 구성하여 이미지 관리 및 컨테이너 구동 관련 주요 기능 동작들을 확인한 결과 환경 구성 및 기능 관련한 심각한 오류 및 치명적 장애 이슈는 발견되지 않음

IV. 종합

- Docker 테스트 수행 결과 공개 SW로 구성된 Stack 환경에서 치명적 결함 이슈 발생 없이 설치, 설정관리되며, 클라우드 가상 플랫폼으로서 이미지 관리 및 배포 관련 제반 기능들이 각 기능 단위 resource들과 유기적으로 동작함을 확인하였다.

- Docker는 컨테이너 구성에 따른 이미지 빌드 및 배포의 체계적 관리 기능을 제공하며, 도커와 호환되는 리눅스 커널과 도커 바이너리가 적용될 수 있는 최소 사양의 호스트 시스템에서 다양한 목적 시스템 구성 및 운용이 가능하도록 유연한 사용자 환경을 제공한다.

- Docker는 이미지 적용만으로 서비스 시스템과 동일한 환경구성이 가능하므로 개발자에게 능동적 테스트 참여가 가능한 환경을 제공한다. 더불어 애플리케이션이 포터블하고 빌드하기 쉬운 컨테이너 기반 환경을 제공하므로 목적 코드 작성부터 테스트를 거쳐 배포 후, 사용되기까지의 빠르고 효율적인 개발 라이프 사이클을 형성할 수 있으므로 다양한 환경에서 협력하기 쉬운 개발환경 조성에 유용한 솔루션이다.

※ 참고 자료

[1] Docker 공식 사이트

- <https://www.docker.com/>

[2] 설치 가이드

- <http://docs.docker.com/installation/#installation>

[3] 배포/ 개발 관련

- <https://hub.docker.com/account/signup/>

- <https://github.com/docker/docker>

[4] 포럼/ 커뮤니티

- <http://blog.docker.com/>

- <http://stackoverflow.com/search?q=docker>

- <https://groups.google.com/forum/#!forum/docker-user>