정보시스템 분야
Stack 통합 Test 결과보고서
[Kubernetes]

2015. 06.

nipa 정보통신산업진흥원 National IT Industry Promotion Agency
nipa 공개SW역량프라자 Open Source Software COMPETENCE PLAZA

목 차

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

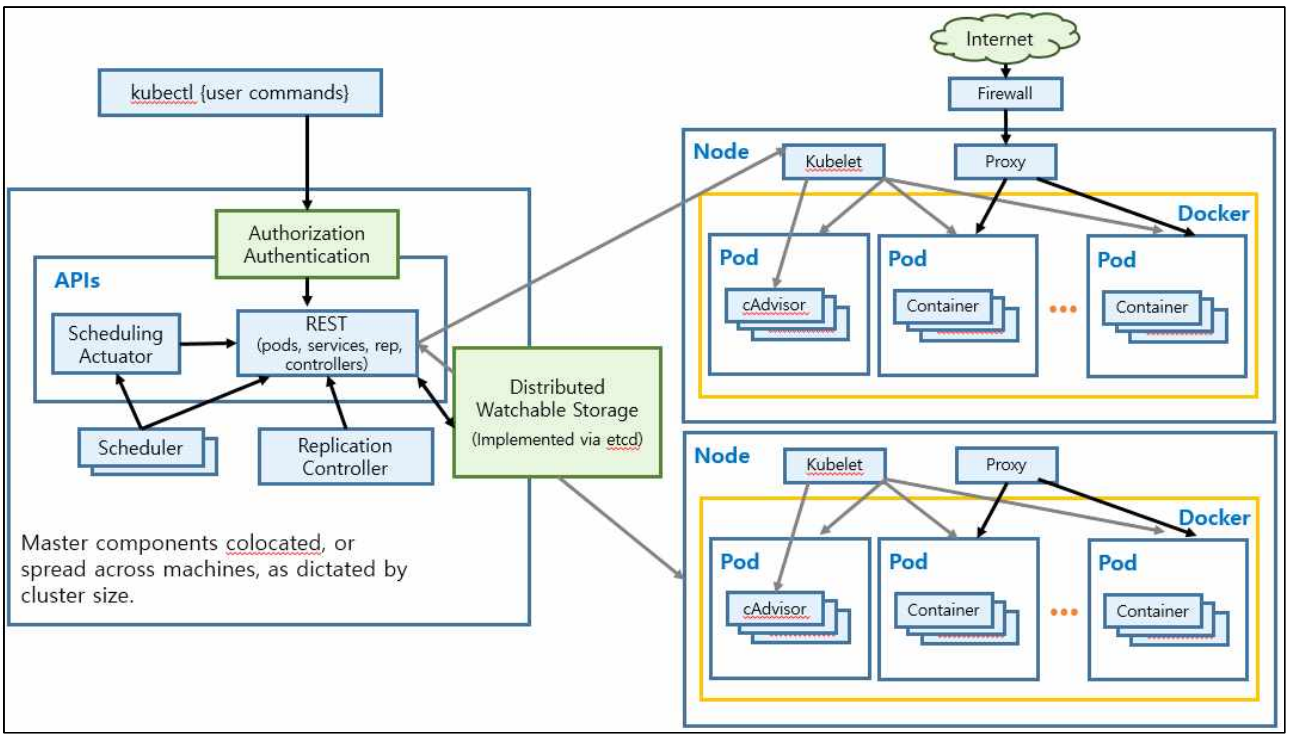
- ㅇ 공개SW로 구성된 Stack이 유기적으로 잘 동작함을 확인
- ㅇ 다양한 Stack 구성에 기반을 둔 테스트를 통해 안정된 Stack 조합 규명
- ㅇ 공개SW 시스템 도입을 위한 Stack 참조모델의 신뢰성 정보로 활용
- ㅇ 공개SW의 신뢰성과 범용성에 대한 사용자 인식 제고

Ⅱ. 테스트 대상 소개

1. Kubernetes 소개

Kubernetes는 Docker 프로젝트의 주요 연관 프로젝트로 Google이 자사의 Backend Container들을 관리하기 위해 고안하여 오픈소스로 발표하였다. Kubernetes 프로젝트는 마이크로소프트(MS), 레드햇, IBM, Docker, CoreOS, Meso Sphere, Salt Stack등이 개발에 참여하고 있다.

Docker 컨테이너의 오픈소스 조직화 관리 시스템(Open Source Orchestration System)인 Kubernetes는 컨테이너의 구성 환경과 역할 구분에 따른 Label 적용 및 어플리케이션 컨테이너의 논리적 그룹 관리 단위인 Pod 개념 적용을 통해 Multi Host 시스템에서 어플리케이션의 배포, 유지보수 및 스케일링이 용이한 효율적인 사용자 환경을 제공한다.



[그림 1. Kubernetes Architecture 개념도]

※ 상기 개념도는 Kubernetes 적용 환경구성에 대한 이해를 돕기 위한 예시 자료임

□ 관련 기술 및 개념 이해

Docker는 2013년 3월 Kubernetes, 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 2. Docker 시스템 구성 개념도]

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

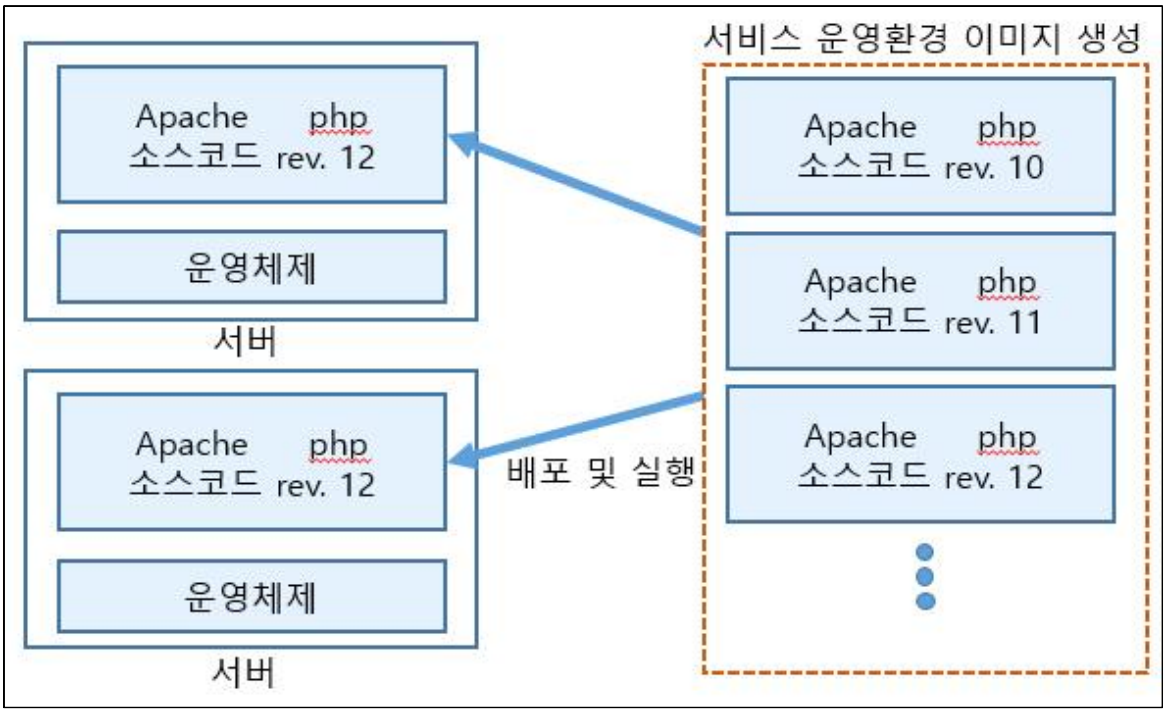
Docker는 운영체제 레벨 가상화(Operating system-level virtualization)의 기능 구현 단위인 컨테이너 안에 있는 애플리케이션 배포를 자동화하는 오픈 소스 엔진으로 Immutable Infrastructure 패러다임이 적용되어 서비스 운영 환경 이미지를 클라우드 플랫폼 상에서 자유로이 생성, 배포 및 운용할 수 있는 사용 환경을 제공한다.

> 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

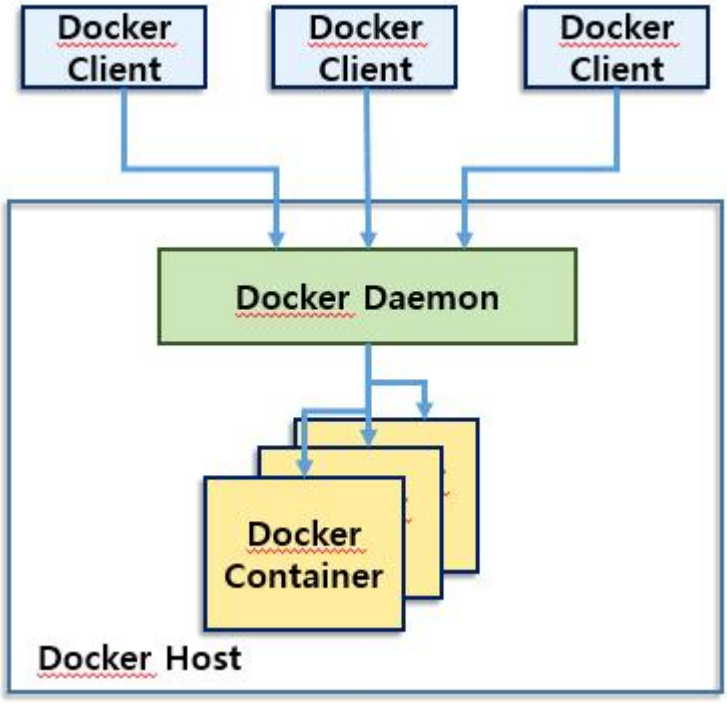


[그림 3. Immutable Infrastructure 개념도]

□ Docker의 주요 기능요소

(1) 도커(Docker) 클라이언트와 서버

도커는 클라이언트-서버 애플리케이션이다. 도커 클라이언트는 도커 서버 혹은 데몬과 통신하며 교대로 작업한다. 커맨드 라인 클라이언트 바이너리인 Kubernetes와 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 4. Docker Architecture 개념도]

(2) 도커(Docker) 이미지

도커 이미지는 소스코드로서 여러 레이어(layer)로 나뉜 형태로 되어 있으며, 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, Kubernetes, Inc는 퍼블릭 레지스트리로 도커 허브(Kubernetes 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 Kubernetes 엔진을 통해 도커의 높은 시스템 호환성을 제공한다.

2. Kubernetes 주요기능 및 특징

Kubernetes는 오픈소스 Infrastructure 플랫폼으로 멀티 호스트 시스템에 대하여 컨테이너화 어플리케이션(Containerized Application)의 관리 환경을 지원하며, 어플리케이션의 배포, 유지보수 및 스케일링을 위한 기초 메커니즘을 제공한다.

Kubernetes의 Architecture는 역할 구분에 따라 etcd, API server와 Scheduler 및 Controller Manager의 기능을 담당하는 Master 서버와 Worker Machine의 역할을 하는 Node 단위인 Minion으로 구성된다.

□ Kubernetes의 특징

> Pod 단위로 컨테이너 그룹화
> Pod Label 처리를 통한 서비스 탐색 및 로드 밸런싱을 허용하는 서비스 추상화
> Pod 인스턴스 설정을 통한 반복제어

□ Kubernetes의 주요 기능요소

(1) etcd

Kubernetes의 자원 정보를 Key/ Value 형태로 저장하는 저장소이며, 클러스트의 구성 정보가 저장된다. etcd의 설정에 따라 해당 기능 및 구성이 모든 노드에 반영된다.

(2) API server

Master 서버의 중심이 되는 것이 API 서버이며, 관리자가 kubectl 명령어를 통해 Pod, Service, Replication Controller 등의 설정을 할 수 있다. Kubernetes 제어수준(control plane) 조절을 위한 전처리(front-end) 역할을 하며 Kubernetes API를 노출시키는 기능을 한다.

(3) Scheduler

새로 생성되어 작업이 할당되지 않은 Pod를 모니터링하며, 필요에 따라 할당 구동시킨다. 또한, Kubectl 명령어를 통해 지시받은 작업을 보유하고 있다가 API server로 전달하는 역할을 담당한다.

(4) Controller Manager

etcd의 Replication Controller 개체를 모니터링하고 변경 사항 발생 시 API 서버를 통해 Pod의 증감 설정을 하는 역할을 담당한다. 예를 들어 관리자가 kubectl 명령어를 통해 replica 수를 10개로 지정하였다면, Pod가 9개가 되는 순간 자동으로 10개로 증가시키고, 11개 이상이 되면 Pod 하나를 감소시켜 지정된 수만큼 유지되도록 관리한다.

(5) Kubelet

Primary Node Agent로 API server 또는 로컬 설정파일(Local Configuration File)을 통해 노드에 할당된 Pod를 모니터링 하며, 노드의 상태 정보를 시스템의 다른 기능 단위에 전달한다.

> Pod가 필요로 하는 Volume을 마운트 함

> Pod의 보안 설정 정보(Secret)를 다운로드 함

> Docker를 통해 Pod의 컨테이너를 구동시킴

> 요청받은 컨테이너의 구동여부 조사(Liveness Probes)를 주기적으로 실행

> 필요시 Mirror Pod를 생성하며 시스템의 연관 기능 단위에 Pod의 상태 정보를 전달

(6) Kube-proxy

Host 시스템의 네트워크 규칙(Network Rule)을 관리(Maintaining)하고 통신접속 전달(Connection Forwarding)을 수행함으로써 Kubernetes

서비스의 추상화를 가능케 한다.

※ 용어/개념 정리

> Cluster : 물리적 또는 가상화 머신과 어플리케이션 구동을 위해 Kubernetes에 의해 사용되는 여타 Infrastructure 자원들의 집합이다.

> Node : Kubernetes를 구동시키는 물리적 또는 가상화 머신으로 노드 상에서 Pod가 스케줄링 될 수 있다.

> Pod : 공유 볼륨(Shared Volume)을 포함하는 공존(co-located) 어플리케이션 컨테이너 그룹으로 배포의 최소 단위이며, 생성 및 스케줄링을 포함한 Kubernetes의 작업, 관리 대상이다.

3. Kubernetes 시스템 요구사항

□ Kubernetes 시스템 요구사항 - Docker 환경구성 기준

OS 구분	정보	필요 환경 요소
Ubuntu	- 12.04 LTS 이상 권장	- 커널 (x86_64 3.8.0 이상 버전) - 디바이스 맵퍼
CentOS/ Fedora	- CentOS 6+ 버전 (64비트) - Fedora core 19+ (64비트)	- 커널 (x86_64 3.8.0 이상 버전) - 디바이스 맵퍼
OS X	- 최신 안정버전 권장	- Boot2Kubernetes (VirtualBox, Kubernetes client 포함)
Windows	- 최신 안정버전 권장	- Boot2Kubernetes (VirtualBox, Kubernetes client 포함)

[표 1. Kubernetes 시스템 요구사항]

※ 상기 시스템 요구사항은 Docker의 시스템 설치 기본 요구 항목이며, Kubernetes는 리눅스 환경(e.g. Ubuntu 12.04+)에서 구성되어 상기 시스템 OS항목 별 구성된 Docker의 클러스터와 연동되어 스케줄링 및 자원을 관리함

※ 상기 시스템 요구사항의 OS 구분 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

- https://github.com/GoogleCloudPlatform/kubernetes/blob/master/docs/design/architecture.md
- https://www.yongbok.net/blog/google-kubernetes-container-cluster-manager/

[2] 설치/ 유저가이드

- https://github.com/GoogleCloudPlatform/kubernetes/tree/master/docs/getting-started-guides

[3] 공식 Repository

- https://github.com/googlecloudplatform/kubernetes

Ⅲ. Stack 통합 테스트

1. 테스트 환경

□ Kubernetes SW 환경

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

[표 2-1. 테스트 SW 환경]

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

※ 상기 Kubernetes 환경 구성 시, GoogleCloudPlatform 기준 이미지 적용함

□ 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 HW조건의 PC 2대로 서버, 클라이언트 환경 구성하여 테스트 진행

2. 테스트 접근 방법

Kubernetes는 Docker 컨테이너의 응용프로그램 배포와 유지 보수 및 확장을 지원하는 관리도구로서 API 서비스를 주관하는 Master 서버와 Master 서버의 명령을 받아 Kubelet 서비스 수행 및 호스트 운영 컨테이너를 관리하는 Minion 기능 단위로 구성된다.
본 테스트는 서버, 클라이언트 환경을 구성하여 도커의 설치 및 운용에 따른 사용자 시나리오를 토대로 Kubernetes 주요 기능 요소의 기능 신뢰성을 확보하며, 환경 구성의 안정성과 Web UI 환경의 호환성을 확인하는 데 주된 테스트 방향성과 목적을 두고 수행되었다.

☐ 탐색적 테스팅(Exploratory Testing)

탐색적 테스팅은 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 테스팅 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 「Kubernetes 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
Kubernetes 기본환경 관리	1	7
Kubernetes 구성 관리	2	24
Kubernetes 운영 관리	2	6
합 계	5	37

[표 3. 테스트 아이디어 현황]

- 기본환경 관리 카테고리에서는 대상 SW의 설치 및 기본 환경구성 측면에서 안정성 확인 및 CLI 모드와 Web 기반 운용환경 별 접속 기능 확인을 중심으로 테스트 진행
- 구성 관리 카테고리에서는 Master 서버 및 Minion 기능 단위 기준에 따른 관리 대상 리소스의 구성과 Pod 단위의 Docker 이미지 관리 기능을 중심으로 테스트 진행
- 운영 관리 카테고리에서는 시스템의 Web 서버 연동에 따라 제공되는 Web UI 상에서 주요 기능 카테고리 별 기능 항목의 동작 정상성 확인 및 Web 환경 호환성 확인을 중심으로 테스트 진행

4. 테스트 수행 결과

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

분류		PASS	FAIL	N/T	N/A
기능	테스트 아이디어				
Kubernetes 기본환경 관리	7	7	0	0	0
Kubernetes 구성 관리	24	24	0	0	0
Kubernetes 운영 관리	6	6	0	0	0

[표 4. 테스트 결과]

- 상기 테스트는 Host OS에 Nginx를 Web 서버로 적용하여 시스템을 구성하였으며, Master 및 Minion 서버 구성에 따른 Kubernetes core 단의 기능 정상성 확보를 중점으로 수행됨

- Kubernetes로 서버, 클라이언트 환경을 구축하고 주요 관리 환경인 CLI 모드와 Web 브라우저 UI에서 로컬 및 원격 Docker 환경에 대한 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

□ Kubernetes 테스트 수행 결과, 공개 SW로 구성된 Stack 환경에서 치명적 결함, 이슈 발생 없이 설치 및 설정 관리됨을 확인하였으며, Docker 연동 환경에 대한 모니터링 및 자원 관리와 이미지 배포 관련 제반 기능들이 각 기능 단위 resource들과 유기적으로 동작함을 확인하였다.

□ Kubernetes는 복수의 Linux 컨테이너로 구성된 클러스터를 하나의 시스템으로 관리하기 위한 오픈소스 프로젝트이다. 컨테이너 간 공존성(co-location)을 제공하는 시스템 관리를 통해 관리자는 멀티 호스트 간에 도커 컨테이너들을 구동시키고 제어하며, 서비스 탐색 및 반복 수행(replication)을 효율적으로 통제할 수 있다.

□ Kubernetes는 JSON 스키마 상에서 동작하는 Swagger Specification을 제공하며, Nginx를 포함한 Web 서버 적용을 통해 Web 브라우저 관리 환경을 제공하므로 Docker 컨테이너 및 볼륨의 그룹 단위 자원에 대한 직관적 접근 및 제어가 가능하다.

※ 참고 자료

[1] Kubernetes 공식 사이트
- http://kubernetes.io/
[2] 설치 가이드/ 유저가이드
- https://github.com/GoogleCloudPlatform/kubernetes/tree/master/docs/getting-started-guides
- https://github.com/GoogleCloudPlatform/kubernetes/tree/master/docs/user-guide
[3] 배포/ 개발 관련
- https://github.com/GoogleCloudPlatform/kubernetes/blob/master/docs/devel/README.md
[4] 포럼/ 커뮤니티
- http://kubernetes.io/community/