쿠버네티스 in Action

쿠버네티스 개요

01. 쿠버네티스 소개
02. 도커와 쿠버네티스 첫걸음

1. 쿠버네티스 소개

- 최근 소프트웨어의 개발과 배포의 변화 이해
- 애플리케이션을 격리하고 컨테이너를 사용해 실행 환경 차이 줄이기
- 쿠버네티스에서 사용되는 컨테이너와 도커의 이해
- 쿠버네티스로 개발자와 시스템 관리자의 작업 간소화하기

1.1. 쿠버네티스 시스템 필요이유

1.1.1 모노리스 애플리케이션에서 마이크로서비스로 전환
- ✓ 마이크로서비스로 애플리케이션 분할
- ✓ 마이크로 서비스 확장 배포
- ✓ 환경요굿사항의 다양성

1.1.2 개발환경과 운영환경(프로덕션환경) 일관된 환경제공

1.1.3 지속적인 배포로 전전환(데브옵스, 노옵스)

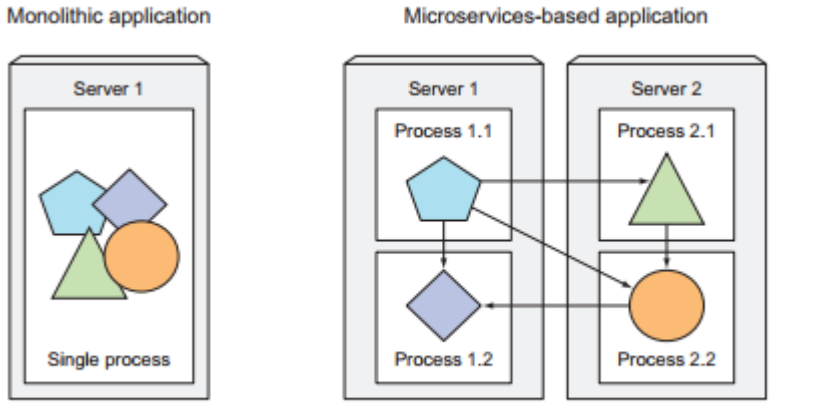


Figure 1.1 Components inside a monolithic application vs. standalone microservices

Devops

1.2. 컨테이너 기술 소개

1.2.1 컨테이너의 이해

- ✓ 리눅스 컨테이너 기술로 구성요소 격리
- ✓ 컨터이너 기술이 가상머신보다 동일한 하드웨어에서 더 많은 수의 소프트웨어 구성요소 실행
- ✓ 컨테이너 격리
  - 리눅스 네임스페이스 : 마운트(mnt),프로세스아이디(pid),네트워크(net),프로세스간통신(ipc),호스트와 도메인이름(uts),사용자ID(user)
  - 리눅스 컨트로 그룹(cgroup) :프로세스 가용 리소스 제한(cpu,메모리, 네트워크 등)

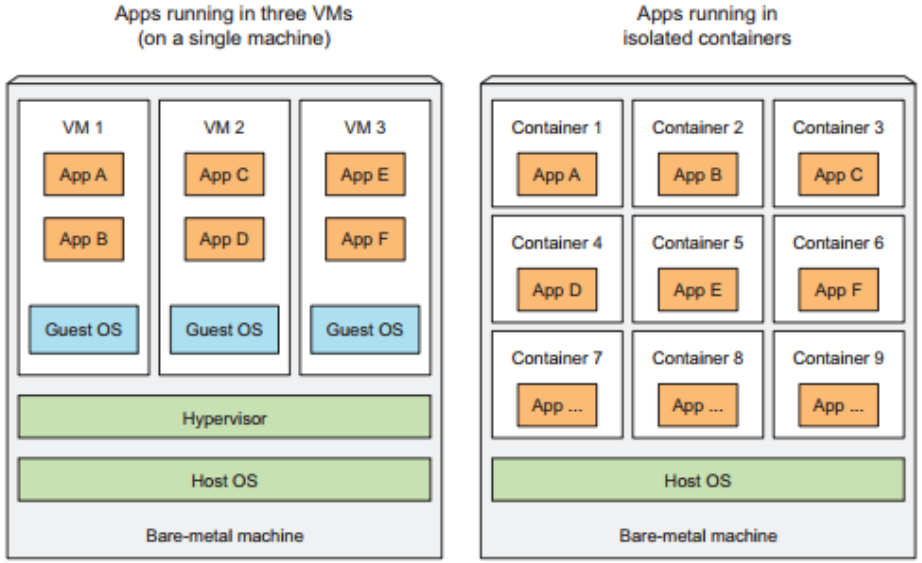


Figure 1.4 Using VMs to isolate groups of applications vs. isolating individual apps with containers

1.2. 컨테이너 기술 소개

1.2.2 도커 컨테이너 플랫폼 소개

✓ 주요 개념

- 이미지 : 애플리케이션과 실행환경 패키지
- 레지스트리 : 도커 이미지 저장, 이미지 공유 저장소
- 컨테이너 : 도커 기반 컨테이너 이미지에서 생성된 리눅스 컨테이너

✓ 도커 이미지 빌드 배포 실행

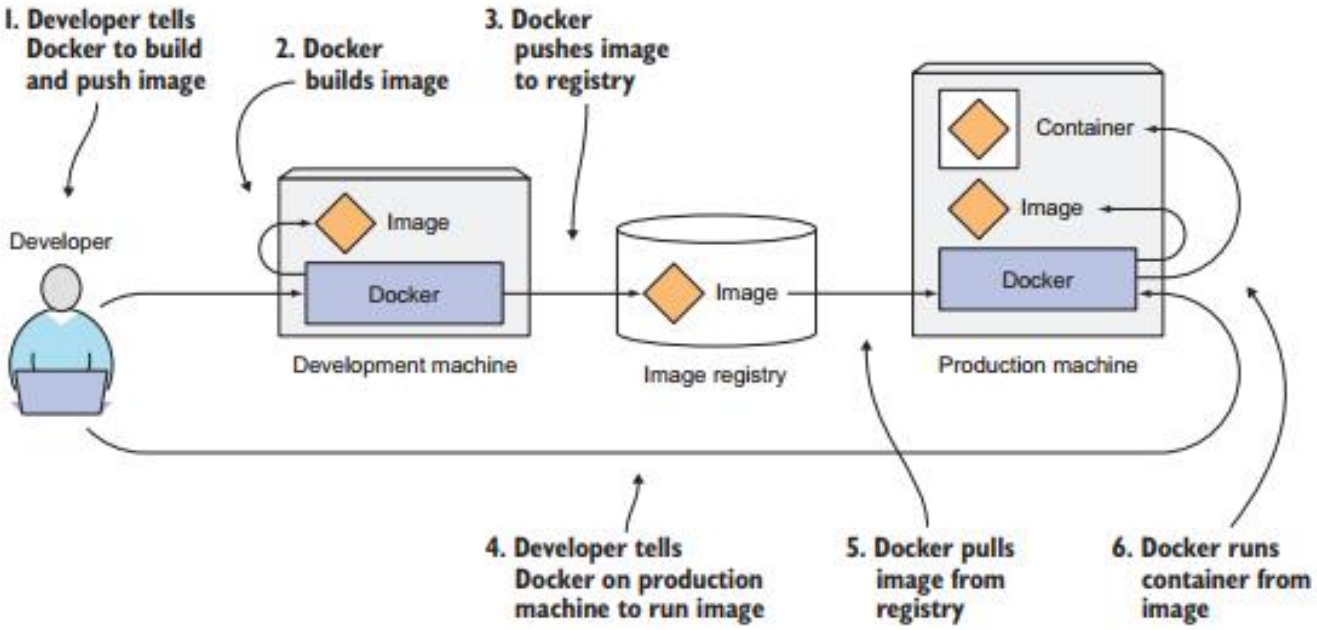


Figure 1.6 Docker images, registries, and containers

1.3. 쿠버네티스 소개

1.3.1 쿠버네티스 개요

✓ 쿠버네티스 : 컨테이너화된 애플리케이션을 쉽게 배포하고 관리할 수 있는 소프트웨어 시스템으로 개발자가 애플리케이션 핵심 기능에 집중하고 운영팀이 효과적으로 리소스를 활용하도록 지원

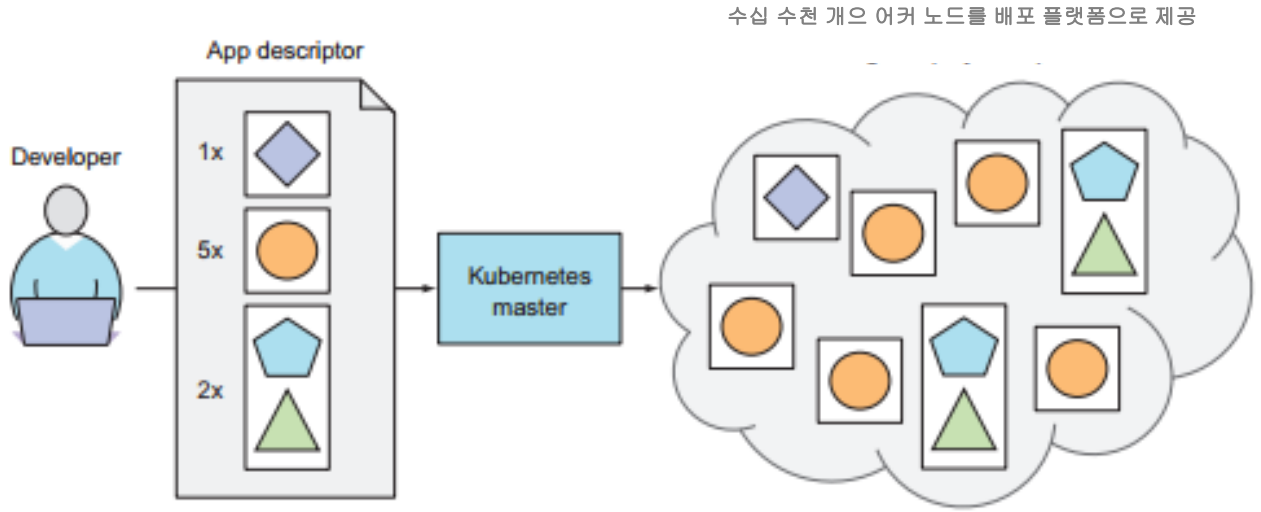


Figure 1.8 Kubernetes exposes the whole datacenter as a single deployment platform.

1.3. 쿠버네티스 소개

1.3.2 쿠버네티스 클러스터 아키텍처

- ✓ 노드
  - 마스터노드 : 쿠버네티스 전체 시스템을 제어하고 관리
  - 워커노드 : 실제 배포되는 컨테이너 애플리케이션 실행
- ✓ 컨트롤플레인
  - Etcd : 클러스터 구성요소를 저장하는 분산 데이터 저장소
  - API 서버 : 사용자, 컨트롤 플레인 구성요소와 통신
  - 컨트롤메니저 : 구성요소 복제본, 워커 노드 추적, 노드 장애 처리 등 클러스터 단 기능 수행
  - 스케쥴러 : 애플리케이션 배포
- ✓ 워커노드
  - Kubelet: API서버 통신 노드 컨테이너 관리
  - 컨테이너 런타임 : 컨테이너 실행 도커
  - Kube-proxy:애플리케이션 구성요소간 네트워크 트래픽 로드밸런싱하는 서비스

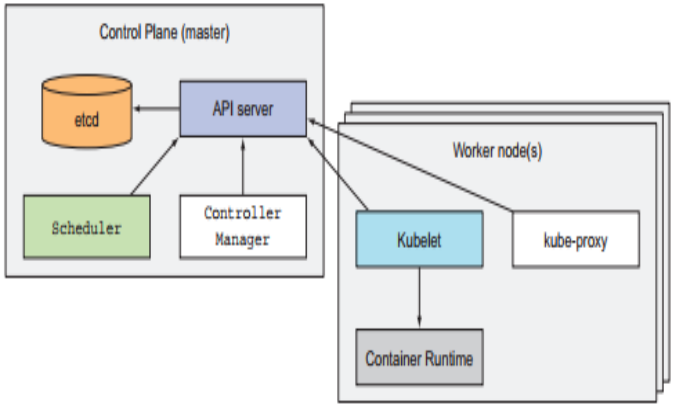


Figure 1.9 The components that make up a Kubernetes cluster

1.3. 쿠버네티스 소개

1.3.3 쿠버네티스 애플리케이션 실행

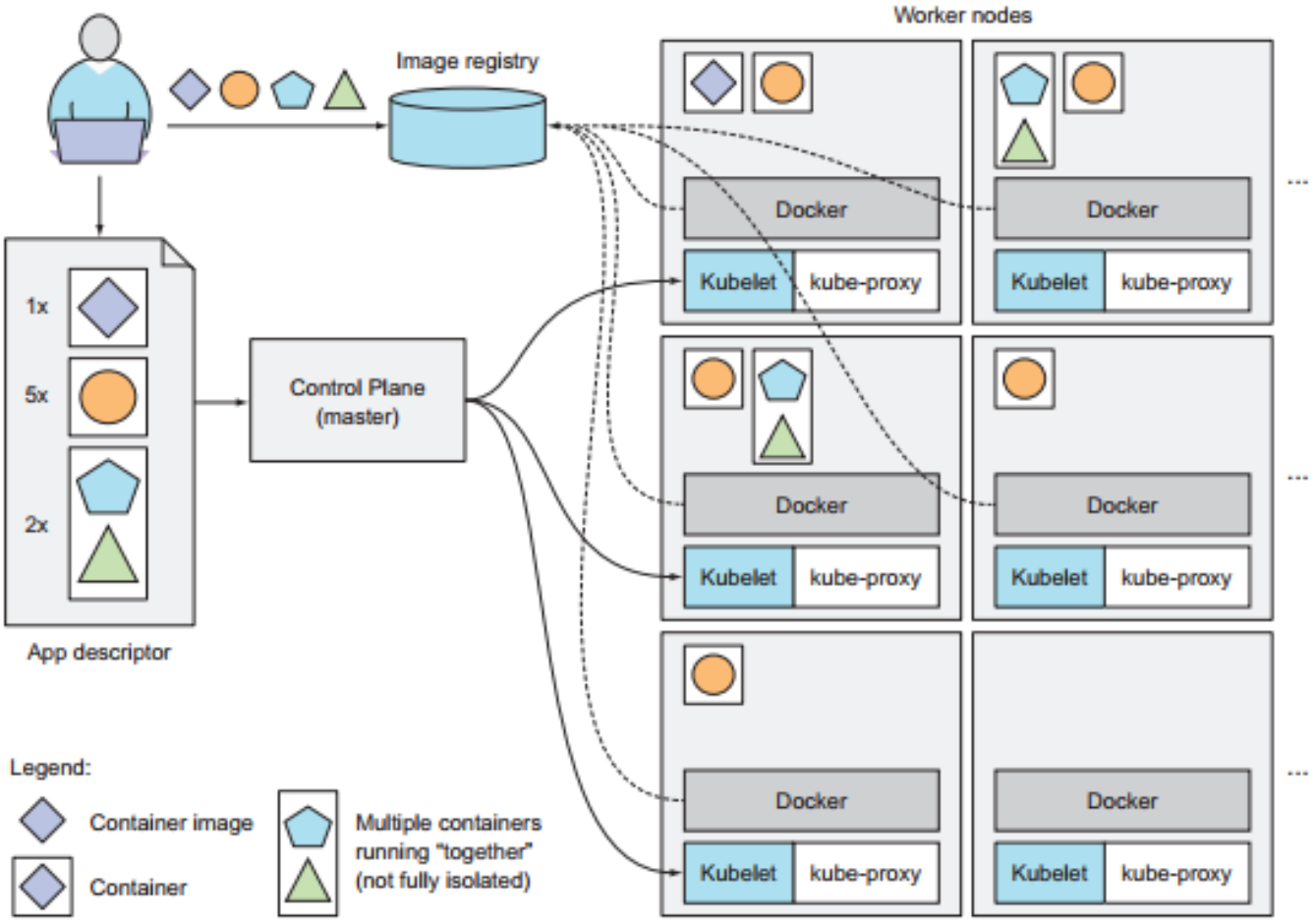


Figure 1.10 A basic overview of the Kubernetes architecture and an application running on top of it

1.3. 쿠버네티스 소개

1.3.4 쿠버네티 장점

- ✓ 애플리케이션 배포 단순화
- ✓ 하드웨어 활용도 높음
- ✓ 상태확인과 자가치유
- ✓ 오토 스케일링
- ✓ 애플리케이션 개발 단순화

2. 도커와 쿠버네티스 첫걸음

- 도커를 사용한 컨테이너 이미지 생성, 실행, 공유
- 로컬에 단일 노드 쿠버네티스 클러스터 실행
- 구글 쿠버네티스 엔진에서 쿠버네티스 클러스터 설치
- kubctl CLI 클라이언트 설정과 사용
- 쿠버네티스에서 애플리케이션의 배포와 수평 스케일링

2.1. 도커를 사용한 컨테이너 이미지 생성, 실행, 공유하기

2.1.1 도커 설치와 Hello World 컨테이너 실행하기

✓ docker run busybox echo “Hello World” (docker run [image이름]:[tag])

CLOUD SHELL
터미널 (steady-velocity-273010)

```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to steady-velocity-273010.
Use “gcloud config set project [PROJECT_ID]” to change to a different project.
ejlee@cloudshell:~ (steady-velocity-273010)$ pwd
/home/ejlee
ejlee@cloudshell:~ (steady-velocity-273010)$ docker run busybox echo "Hello World"
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
d9cbbca60e5f: Pull complete
Digest: sha256:a7766145a775d39e53a713c75b6fd6d318740e70327aaa3ed5d09e0ef33fc3df
Status: Downloaded newer image for busybox:latest
Hello World
ejlee@cloudshell:~ (steady-velocity-273010)$
```

*busybox : echo, ls, gzip 등과 같은 표준 unix 명령줄 도구 모음 단일 실행파일 http://hub.docker.com 공개된 이미지 검색가능

✓ 동작원리

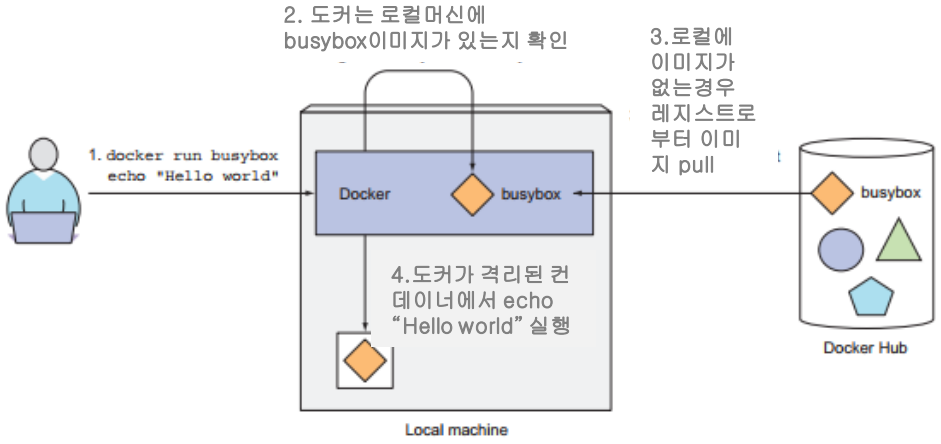


Figure 2.1 Running echo “Hello world” in a container based on the busybox container image

2.1. 도커를 사용한 컨테이너 이미지 생성, 실행, 공유하기

2.1.2 애플리케이션 이미지 생성

✓ Node.js 애플리케이션 app.js 구현

```
1   const http=require("http");
2   const os=require("os");
3
4   console.log("Kubia server starting...");
5
6   var handler=function(request, response){
7       console.log("Received request from "+request.connection.remoteAddress);
8       response.writeHead(200);
9       response.end("You've hit"+os.hostname()+"\n");
10  };
11
12  var wwww = http.createServer(handler);
13
14  wwww.listen(3000);
```

✓ 이미지를 위한 Dockerfile 생성

```
1   FROM node:7
2   ADD app.js /app.js
3   ENTRYPOINT ["node", "app.js"]
```

✓ 컨테이너 이미지 생성

docker build [OPTION] PATH|URL|-

docker build -t kubia:0.1 .

2.1. 도커를 사용한 컨테이너 이미지 생성, 실행, 공유하기

✓ 컨테이너 이미지 생성과 원리
docker build –t kubia .

```
ejlee@cloudshell:~/02 (steady-velocity-273010)$ docker build -t kubia .
Sending build context to Docker daemon  3.072kB
Step 1/3 : FROM node:7
7: Pulling from library/node
ad74af05f5a2: Pull complete
2b032b8bbe8b: Pull complete
a9a5b35f6ead: Pull complete
3245b5a1c52c: Pull complete
afa075743392: Pull complete
9fb9f21641cd: Pull complete
3f40ad2666bc: Pull complete
49c0ed396b49: Pull complete
Digest: sha256:af5c2c6ac8bc3fa372ac031ef60c45a28
Status: Downloaded newer image for node:7
 ---> d9aed20b68a4
Step 2/3 : ADD app.js /app.js
 ---> 7972b790f81c
Step 3/3 : ENTRYPOINT ["node", "app.js"]
 ---> Running in 7b68aea13974
Removing intermediate container 7b68aea13974
 ---> 58beba861713
Successfully built 58beba861713
Successfully tagged kubia:latest
```

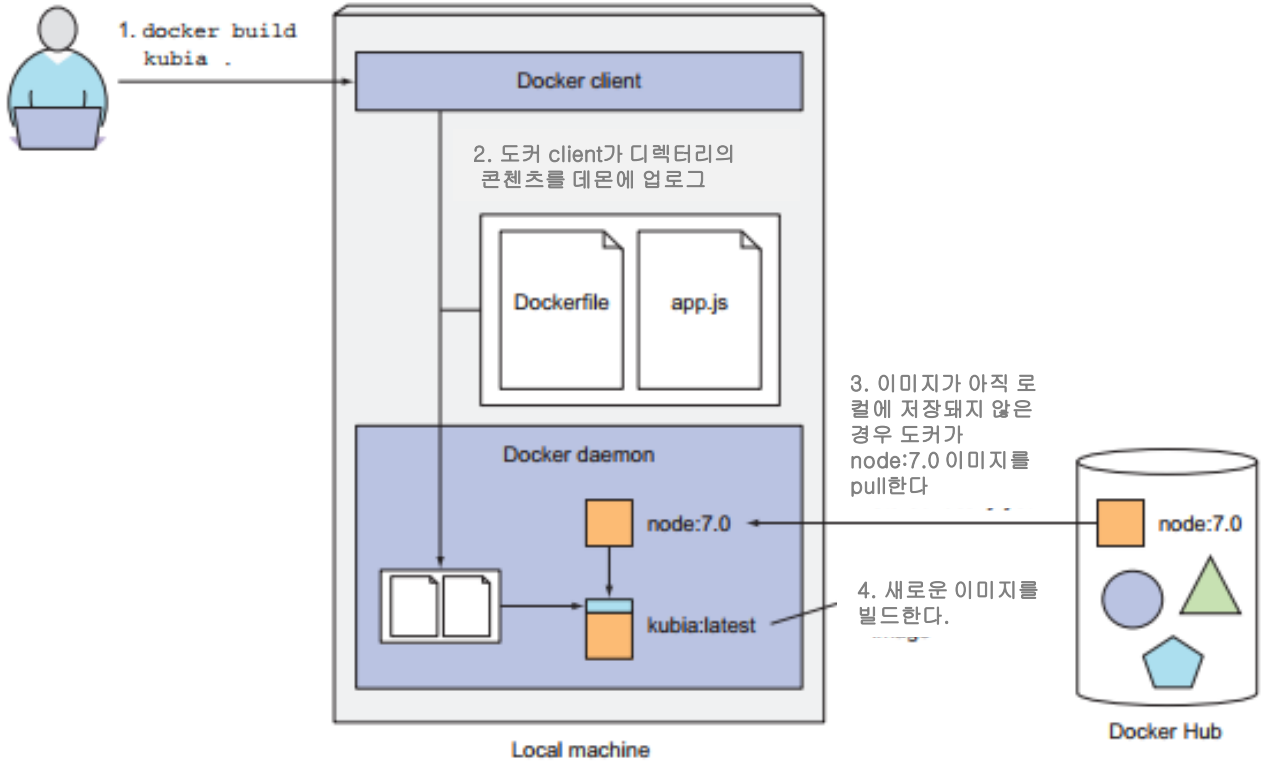


Figure 2.2 Building a new container image from a Dockerfile

2.1. 도커를 사용한 컨테이너 이미지 생성, 실행, 공유하기

✓ 컨테이너 이미지 조회 실행
docker images

```
ejlee@cloudshell:~/02 (steady-velocity-273010)$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
kubia               latest              58beba861713        34 minutes ago      660MB
node                7                   d9aed20b68a4        2 years ago         660MB
```

docker run --name kubia-container -p 8080:8080 -d kubia

```
ejlee@cloudshell:~/02 (steady-velocity-273010)$ docker run --name kubia-container -p 8080:8080 -d kubia
948ca37ab71d2646571d1bd888e78f3c4efd1dc28f30e62937658956d54b0c58
```

✓ 애플리케이션 접근(GCP shell을 이용하는 경우 방화벽 open)
curl localhost:8080

✓ 실행중인 컨테이너 확인
docker ps (또는 docker inspect kubia-container)

```
ejlee@cloudshell:~/02 (steady-velocity-273010)$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS                    NAMES
948ca37ab71d        kubia               "node app.js"       12 minutes ago      Up 12 minutes       0.0.0.0:8080->8080/tcp   kubia-container
```

docker exec -it kubia-container bash (i:표준입력, t:pseudo 터미널할당)

```
ejlee@cloudshell:~/02 (steady-velocity-273010)$ docker exec -it kubia-container bash
root@948ca37ab71d:/#
```

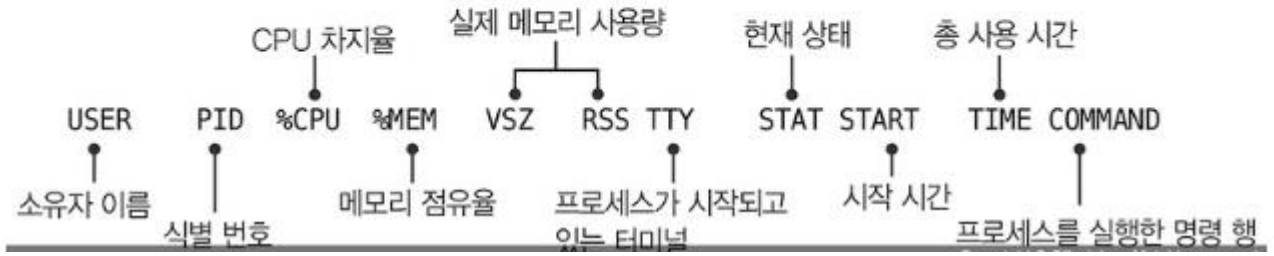
2.1. 도커를 사용한 컨테이너 이미지 생성, 실행, 공유하기

✓ 내부 컨테이너 탐색

ps aux

```
root@948ca37ab71d:/# ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  1.2 614436 25980 ?        Ssl  10:27   0:00 node app.js
root          11  0.0  0.1  20248  3220 pts/0    Ss   11:00   0:00 bash
root          16  0.0  0.1  17504  2056 pts/0    R+   11:04   0:00 ps aux
```

ps aux | grep 프로세스 이름



✓ 컨테이너 중지와 삭제

docker stop kubia-container (중지)

docker rm kubia-container (삭제)

```
ejlee@cloudshell:~/02 (steady-velocity-273010)$ docker stop kubia-container
kubia-container
ejlee@cloudshell:~/02 (steady-velocity-273010)$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                          PORTS               NAMES
948ca37ab71d        kubia               "node app.js"       About an hour ago   Exited (137) 15 minutes ago                         kubia-container
ejlee@cloudshell:~/02 (steady-velocity-273010)$ docker rm kubia-container
kubia-container
ejlee@cloudshell:~/02 (steady-velocity-273010)$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
```

2.1. 도커를 사용한 컨테이너 이미지 생성, 실행, 공유하기

2.1.3 컨테이너 외부 공유

- ✓ 컨테이너 공유는 도커허브나 Quay.io 또는 구글 컨테이너 레지스트리 등 외부 저장소에 푸시
- ✓ 도커허브 규칙에 맞게 이미지 태그 지정(tag명은 도커허브 아이디)
  docker tag kubia openeg/kubia
  docker images

```
ejlee@cloudshell:~/02 (steady-velocity-273010)$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
kubia               latest              58beba861713        3 hours ago         660MB
luksa/kubia         latest              58beba861713        3 hours ago         660MB
openeg/kubia        latest              58beba861713        3 hours ago         660MB
node                7                   d9acd20b60a1        2 years ago         660MB
```

- ✓ 도커허브에 이미지 푸시
  docker login
  docker push openeg/kubia

```
ejlee@cloudshell:~/02 (steady-velocity-273010)$ docker push openeg/kubia
The push refers to repository [docker.io/openeg/kubia]
b451cb7d10d5: Preparing
ab90d83fa34a: Preparing
8ee318e54723: Preparing
e6695624484e: Preparing
da59b99bbd3b: Preparing
5616a6292c16: Waiting
f3ed6cb59ab0: Waiting
```

- ✓ 다른 머신에서 이미지 실행
  docker run –p 8080:8080 –d openeg/kubia

2.2. 쿠버네티스 클러스터 설치

2.2.1 minikube를 활용한 단일 노드 쿠버네티스 실행하기

✓ minikube 설치
https://kubernetes.io/docs/tasks/tools/install-minikube/

✓ Minikube로 쿠버네티스 클러스터 시작
minikube start

```
ejlee@cloudshell:/usr/local/bin (steady-velocity-273010)$ minikube start
* minikube v1.10.1 on Debian 9.12
* Automatically selected the docker driver
* Starting control plane node minikube in cluster minikube
* Pulling base image ...
* Downloading Kubernetes v1.18.2 preload ...
    > preloaded-images-k8s-v3-v1.18.2-docker-overlay2-amd64.tar.lz4: 525.43 MiB
* Creating docker container (CPUs=2, Memory=1995MB) ...
* Preparing Kubernetes v1.18.2 on Docker 19.03.2 ...
  - kubeadm.pod-network-cidr=10.244.0.0/16
! initialization failed, will try again: run: /bin/bash -c "sudo env PATH=/var/lib/minikube/binaries/v1.1
le--var-lib-minikube,DirAvailable--var-lib-minikube-etcd,FileAvailable--etc-kubernetes-manifests-kube-sch
l,FileAvailable--etc-kubernetes-manifests-etcd.yaml,Port-10250,Swap,SystemVerification,FileContent--proc-
stdout:
[init] Using Kubernetes version: v1.18.2
[preflight] Running pre-flight checks
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your internet connection
[preflight] You can also perform this action in beforehand using 'kubeadm config images pull'
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
```

✓ kubectl 쿠버네티스 클라이언트 설치
https://kubernetes.io/docs/tasks/tools/install-kubectl/#install-kubectl-on-linux

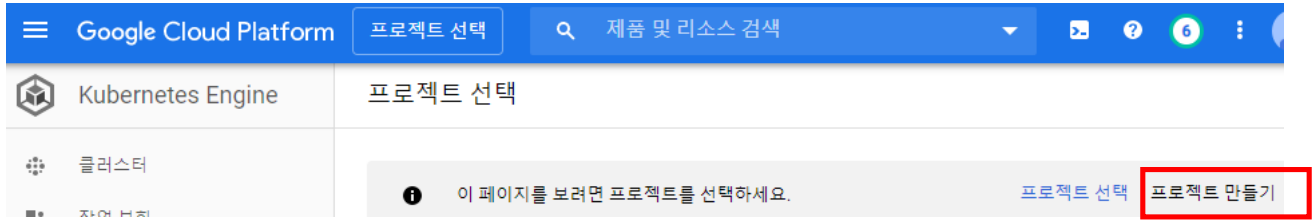
2.2. 쿠버네티스 클러스터 설치

2.2.2 구글 쿠버네티스 엔진을 활용한 관리형 쿠버네티스 클러스터 사용하기

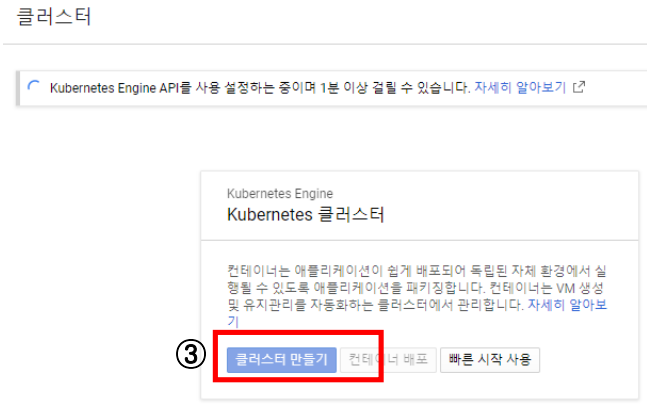
✓ 구글 kubectl 클라우드 프로젝트 설정과 필요한 클라이언트 바이너리 다운로드
https://cloud.google.com/kubernetes-engine/docs/quickstart

1.구글 계정 생성
2.프로젝트 생성
3.빌링활성화
4.쿠버네티스 엔진 API 활성화
5.구글 클라우드 SDK 설치(gcloud 명령행 도구 포함)
6. kubectl명령행 도구 설치 (gcloud components install kubectl)

①



②



③

2.2. 쿠버네티스 클러스터 설치

✓ 클러스터 작용

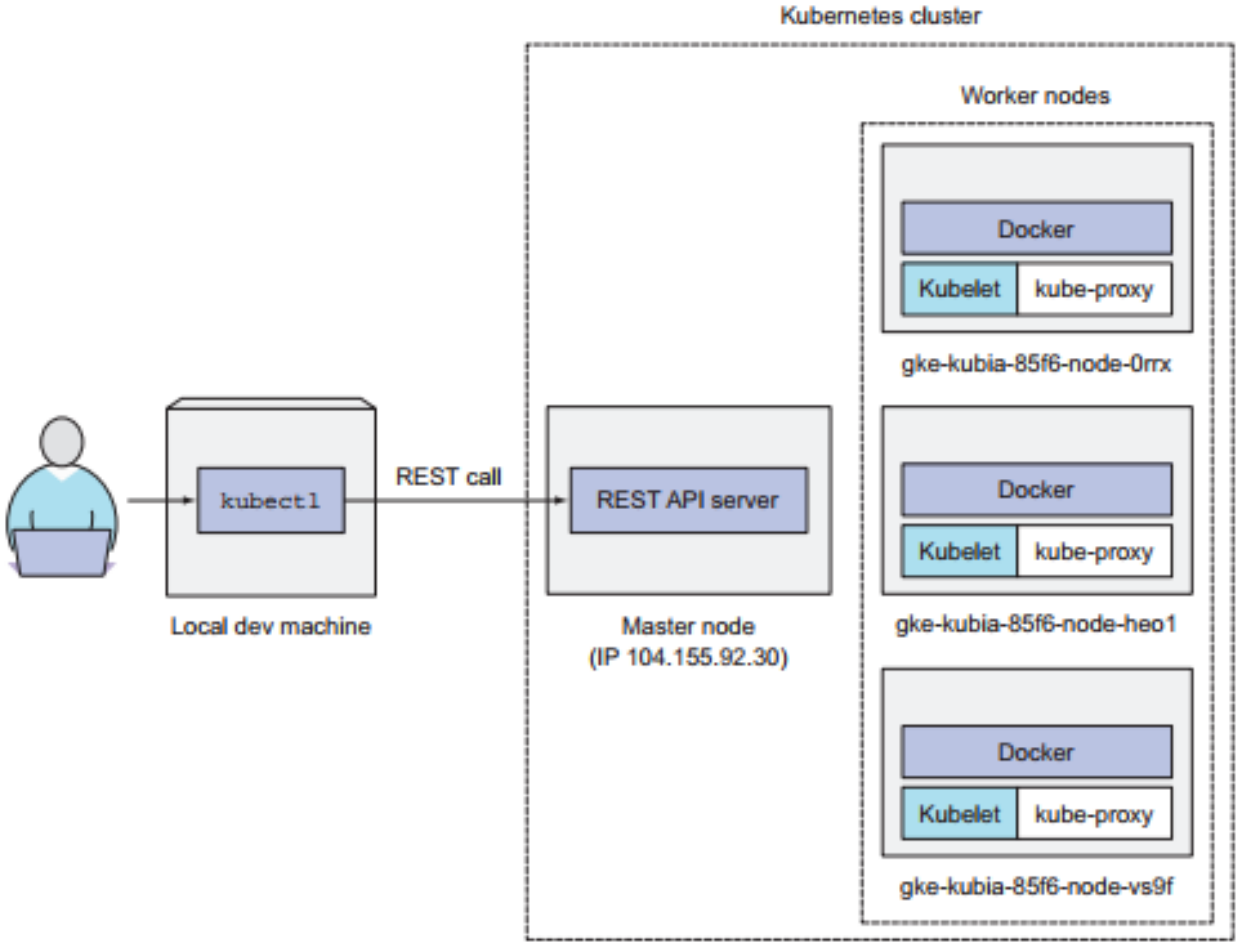


Figure 2.4 How you're interacting with your three-node Kubernetes cluster

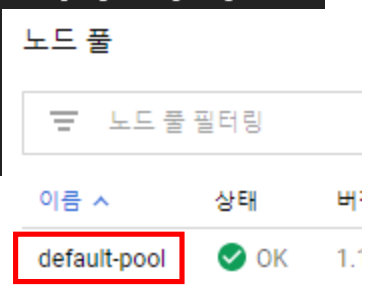
2.2. 쿠버네티스 클러스터 설치

✓ 클러스터 동작 상태 확인

gcloud container clusters get-credentials cluster-1 --zone us-central1-c --project [project-name]

kubectl get nodes

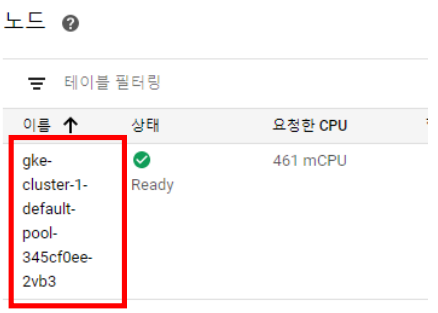
```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to openeg-43795.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
ejlee@cloudshell:~ (openeg-43795)$ gcloud container clusters get-credentials cluster-1 --zone us-central1-c --project openeg-43795
Fetching cluster endpoint and auth data.
kubeconfig entry generated for cluster-1.
ejlee@cloudshell:~ (openeg-43795)$ kubectl get nodes
NAME                                          STATUS   ROLES    AGE   VERSION
gke-cluster-1-default-pool-345cf0ee-2vb3   Ready    <none>   34m   v1.14.10-gke.27
gke-cluster-1-default-pool-345cf0ee-7x9m   Ready    <none>   34m   v1.14.10-gke.27
gke-cluster-1-default-pool-345cf0ee-9sbh   Ready    <none>   34m   v1.14.10-gke.27
```



✓ 오브젝트 세부정보 가져오기

kubectl describe node [node-name]

```
ejlee@cloudshell:~ (openeg-43795)$ kubectl describe node gke-cluster-1-default-pool-345cf0ee-2vb3
Name:               gke-cluster-1-default-pool-345cf0ee-2vb3
Roles:              <none>
Labels:             beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/fluentd-ds-ready=true
```



2.3. 쿠버네티스에 첫 번째 애플리케이션 실행하기

2.3.1 쿠버네티스에 애플리케이션 실행하기

✓ 애플리케이션 배포

kubectl create deployment hello-server --image=gcr.io/google-samples/hello-app:1.0

```
ejlee@cloudshell:~/02 (openeg-43795)$     kubectl create deployment hello-server --image=gcr.io/google-samples/hello-app:1.0
deployment.apps/hello-server created
```

(kuberctl run kubia --image=openeg/kubia --port=8080 --generator=run/v1 replicationcontroller "kubia" created 또는 kuberctl created -f ../kubia.yaml)

✓ 인터넷에 배포된 애플리케이션 노출

kubectl expose deployment hello-server --type LoadBalancer --port 80 --target-port 8080

```
ejlee@cloudshell:~/02 (openeg-43795)$ kubectl expose deployment hello-server --type LoadBalancer --port 80 --target-port 8080
service/hello-server exposed
```

(kubectl expose rc kubia --type=LoadBalancer --name kubia-http service "kubia-http" exposed)

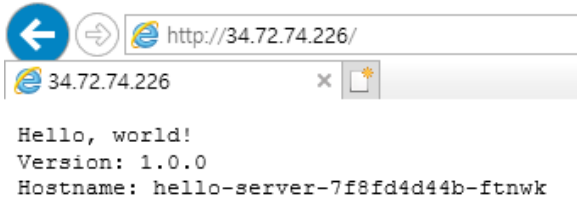
✓ 애플리케이션 검사 사용

kubectl get pods (실행중이 pod검사)

kubectl get service hello-server(서비스 검사)

http://external-ip/ (외부 IP와 노출된 포트를 사용하영 요청

```
ejlee@cloudshell:~/02 (openeg-43795)$ kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
hello-server-7f8fd4d44b-ftnwk   1/1     Running   0          3m40s
ejlee@cloudshell:~/02 (openeg-43795)$ kubectl get service hello-server
NAME           TYPE           CLUSTER-IP    EXTERNAL-IP    PORT(S)        AGE
hello-server   LoadBalancer   10.8.11.134   34.72.74.226   80:32589/TCP   2m11s
```



2.3. 쿠버네티스에 첫 번째 애플리케이션 실행하기

✓ 파드(pod)

- 쿠버네티스 애플리케이션의 기본 실행 단위
- 쿠버네티스 객체 모델 중 만들고 배포할 수 있는 가장 작고 간단한 단위
- 클러스터 에서의 Running 프로세스
- 애플리케이션 컨테이너(또는 다중 컨테이너), 저장소 리소스, 특정 네트워크 정체성(IP 주소) 및 컨테이너가 동작하기 위해 만들어진 옵션들 캡슐화
- 배포의 단위

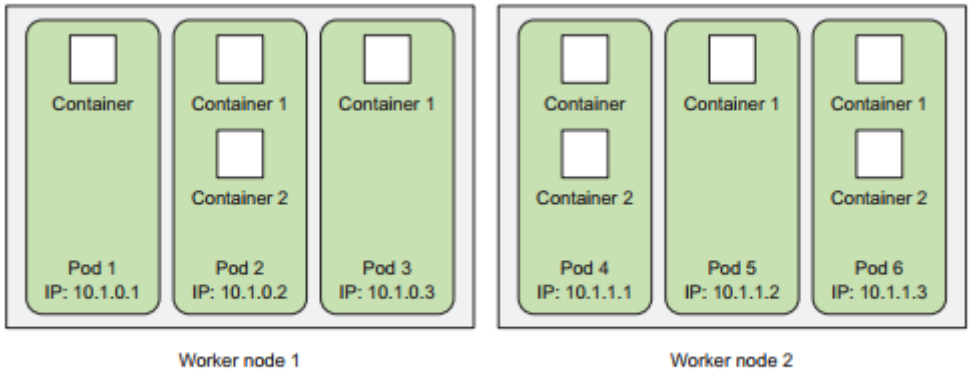
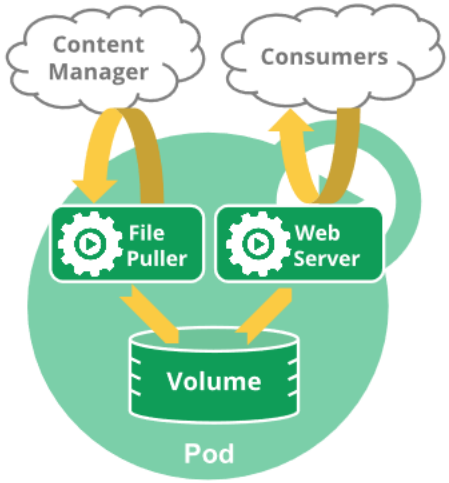


Figure 2.5 The relationship between containers, pods, and physical worker nodes



다중컨테이너 관리

출처: https://kubernetes.io/ko/docs/concepts/workloads/pods/pod-overview/

2.3. 쿠버네티스에 첫 번째 애플리케이션 실행하기

✓ 동작원리

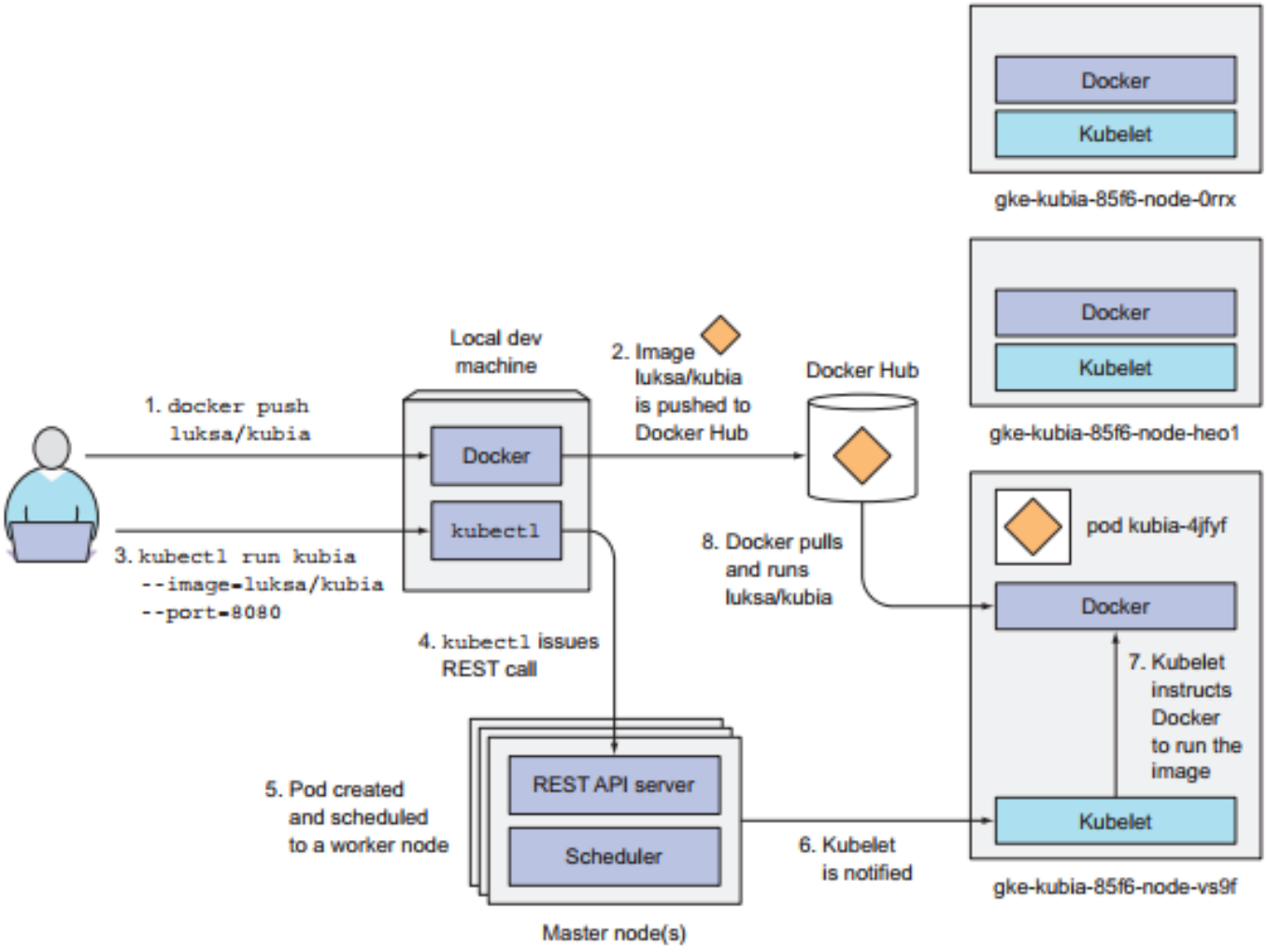


Figure 2.6 Running the `luksa/kubia` container image in Kubernetes

2.3. 쿠버네티스에 첫 번째 애플리케이션 실행하기

✓ 서비스 삭제
kubectl delete service hello-server

✓ 클러스터 삭제
gcloud container clusters delete [cluster-name]