Vmware 로컬 환경에서
AWS 인프라 3 Tier 아키텍처 구축 프로젝트

VMware 기반 환경에서 LocalStack을 활용한 AWS 시뮬레이션 구현

프로젝트 개요

- 구축 목적

AWS 환경을 로컬에서 최대한 비슷하게 구현하여 클라우드 서비스 테스트를 비용 부담 없이 수행할 수 있는 3 Tier 인프라를 구축했습니다.

- 서비스 내용

특정 OS Image를 선택 후 리소스를 선택해 인스턴스를 생성하는 클라우드 시스템에 영감을 받아 사용자에게 받은 입력 값을 토대로 Kubernetes에서 Deploy를 생성하는 시스템을 구축 했습니다.

또한 Cloudflare Tunnel을 통해 외부 네트워크 환경에서도 접속이 가능하며, 접속한 사용자는 제공된 3개 이미지 중 하나를 선택하고 리소스를 지정한 뒤 생성 요청을 보내면

그 요청이 내부적으로 Deploy 생성으로 이어지는 자동화된 클라우드 인스턴스 생성 서비스 입니다.

사용 기술

- VMware 17.5 Pro (로컬에서 사용할 가상화 플랫폼)
- Local Stack (AWS 서비스 에뮬레이터)
  - ㄴ Lambda
  - ㄴ Gateway API
  - ㄴ IAM
  - ㄴ SQS
- AWS Cli (AWS 서비스 프로비저닝 및 관리)
- MINIO (Local Stack 과 호환되는 객체 스토리지)
- Docker (컨테이너 오케스트레이션)
- Kubernetes (컨테이너 오케스트레이션 플랫폼)
- HAProxy + Cloudflare Tunnel (네트워킹)
- Dynamo DB (로그 적재용 DB)
- Veeam (백업 및 복구 솔루션)

```
root@weblb:/home/vagrant# cloudflared tunnel --url http://localhost:80
2025-11-02T23:51:50Z INF Thank you for trying Cloudflare Tunnel. Doing so, without a Cloudflare account, is a quick way to experiment and try it out. However, be aware that these account-less Tunnels have no u
ptime guarantee, are subject to the Cloudflare Online Services Terms of Use (https://www.cloudflare.com/website-terms/), and Cloudflare reserves the right to investigate your use of Tunnels for violations of s
uch terms. If you intend to use Tunnels in production you should use a pre-created named tunnel by following: https://developers.cloudflare.com/cloudflare-one/connections/connect-apps
2025-11-02T23:51:50Z INF Requesting new quick Tunnel on trycloudflare.com...
2025-11-02T23:51:53Z INF +--------------------------------------------------------------------------------------------+
2025-11-02T23:51:53Z INF |  Your quick Tunnel has been created! Visit it at (it may take some time to be reachable):  |
2025-11-02T23:51:53Z INF | https://focused-assessing-shine-disposal.trycloudflare.com                                 |
2025-11-02T23:51:53Z INF +--------------------------------------------------------------------------------------------+
```

Container Provisioning System

focused-assessing-shine-disposal.trycloudflare.com

🚀 Container Provisioning

컨테이너 이미지와 리소스를 선택하여 새로운 컨테이너를 생성하세요

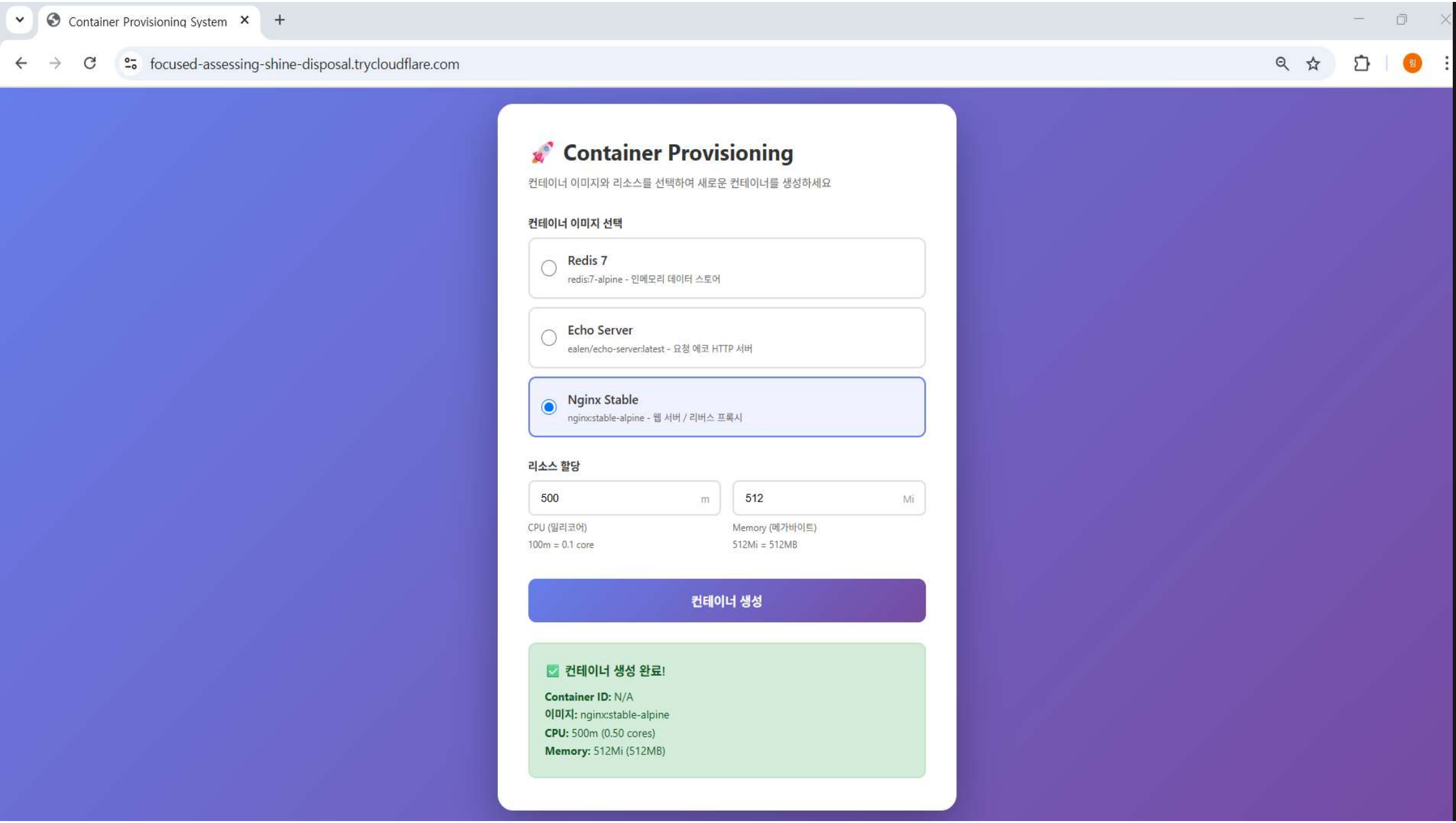
컨테이너 이미지 선택

- ○ **Redis 7**
  redis:7-alpine - 인메모리 데이터 스토어
- ○ **Echo Server**
  ealen/echo-server:latest - 요청 에코 HTTP 서버
- ◉ **Nginx Stable**
  nginx:stable-alpine - 웹 서버 / 리버스 프록시

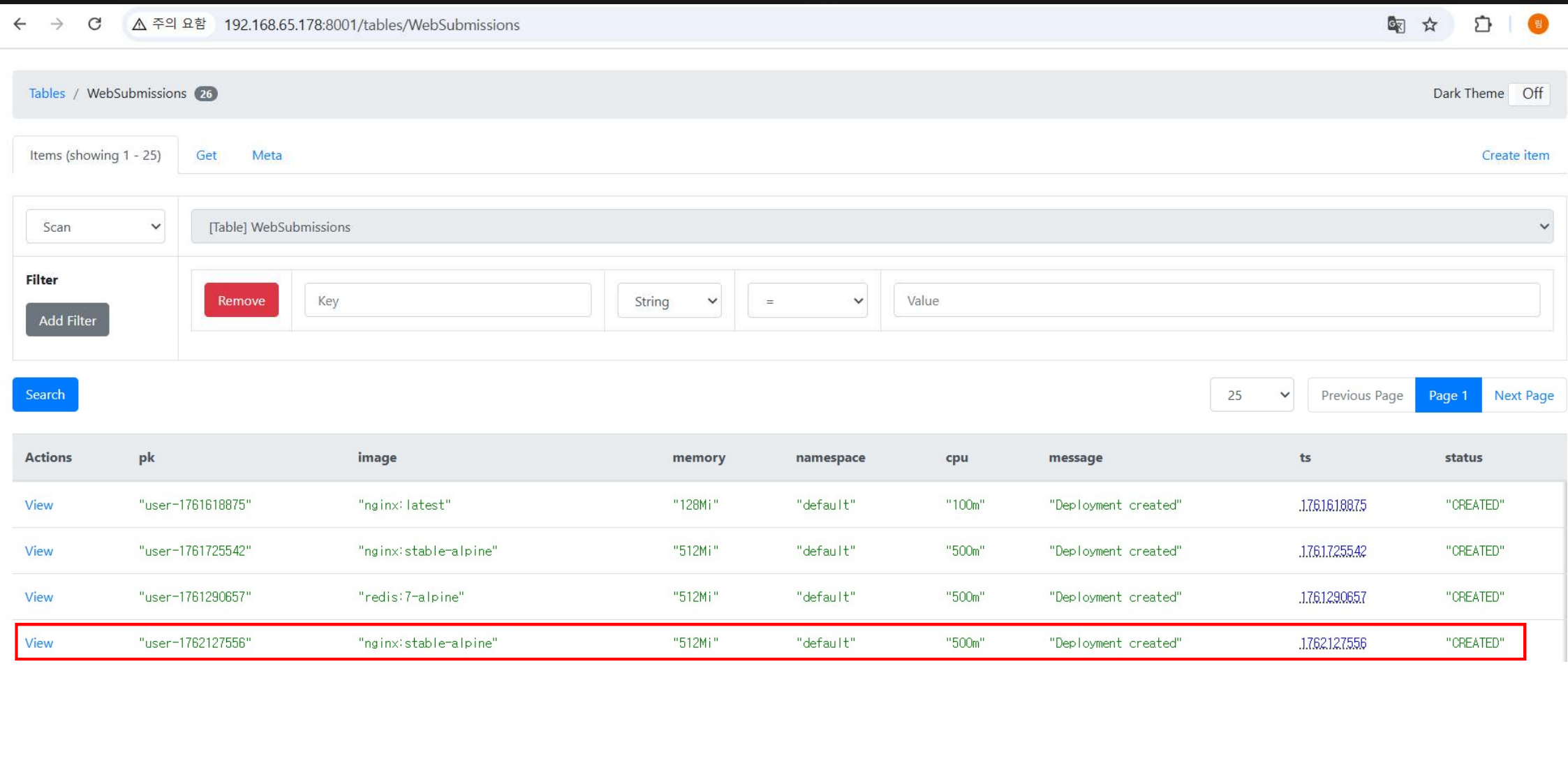
리소스 할당

CPU (밀리코어)	Memory (메가바이트)
500 m	512 Mi
100m = 0.1 core	512Mi = 512MB

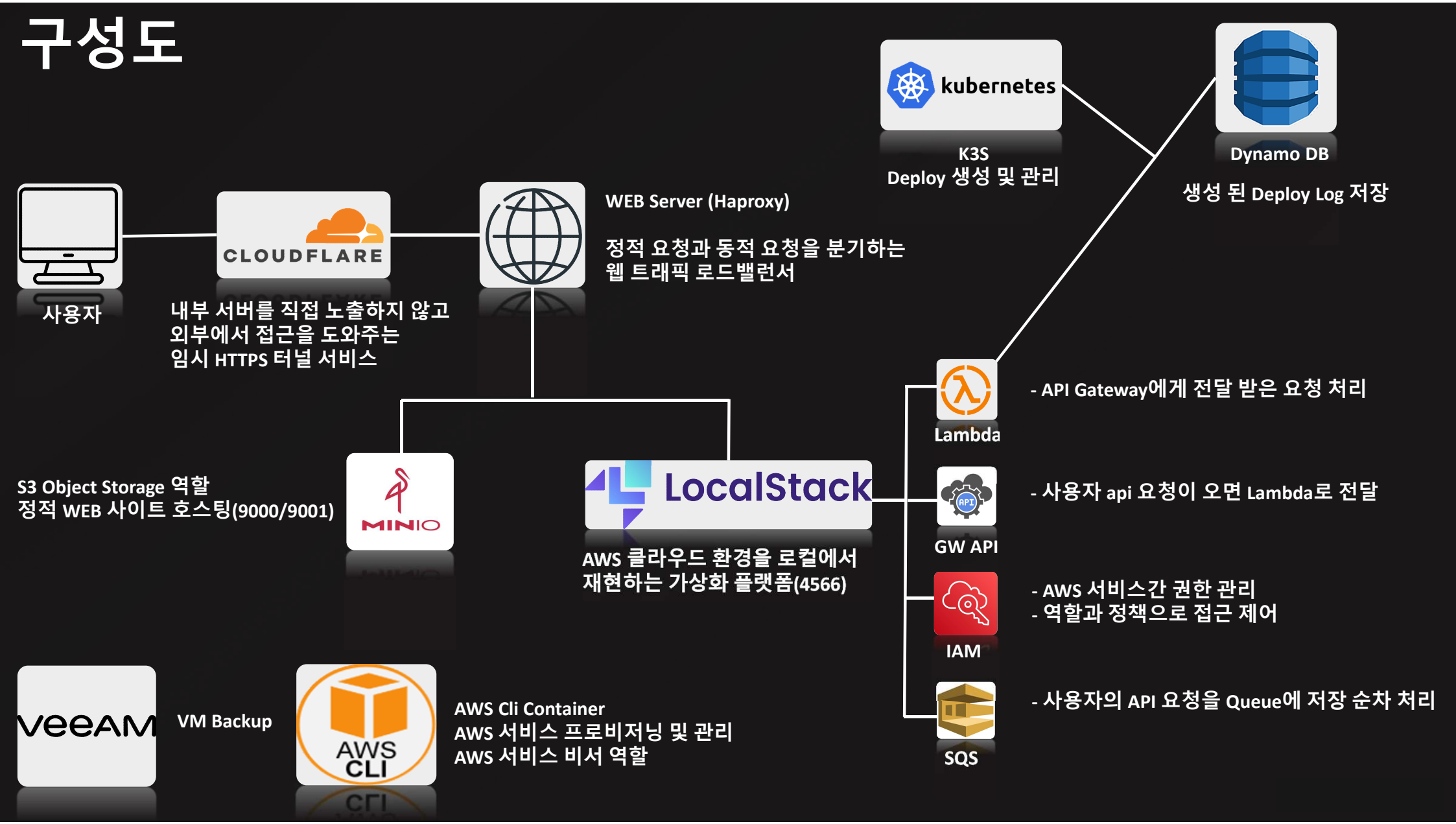
컨테이너 생성


Container Provisioning System
focused-assessing-shine-disposal.trycloudflare.com
Container Provisioning
컨테이너 이미지와 리소스를 선택하여 새로운 컨테이너를 생성하세요
컨테이너 이미지 선택
Redis 7
redis:7-alpine - 인메모리 데이터 스토어
Echo Server
ealen/echo-server:latest - 요청 에코 HTTP 서버
Nginx Stable
nginx:stable-alpine - 웹 서버 / 리버스 프록시
리소스 할당
500
m
512
Mi
CPU (밀리코어)
100m = 0.1 core
Memory (메가바이트)
512Mi = 512MB
컨테이너 생성
컨테이너 생성 완료!
Container ID: N/A
이미지: nginx:stable-alpine
CPU: 500m (0.50 cores)
Memory: 512Mi (512MB)

```
root@kubek3s:/home/vagrant# kubectl get deploy -o wide
NAME              READY   UP-TO-DATE   AVAILABLE   AGE    CONTAINERS   IMAGES               SELECTOR
user-1762127556   1/1     1            1           2m5s   main         nginx:stable-alpine   app=user-1762127556
root@kubek3s:/home/vagrant#
```



구성도

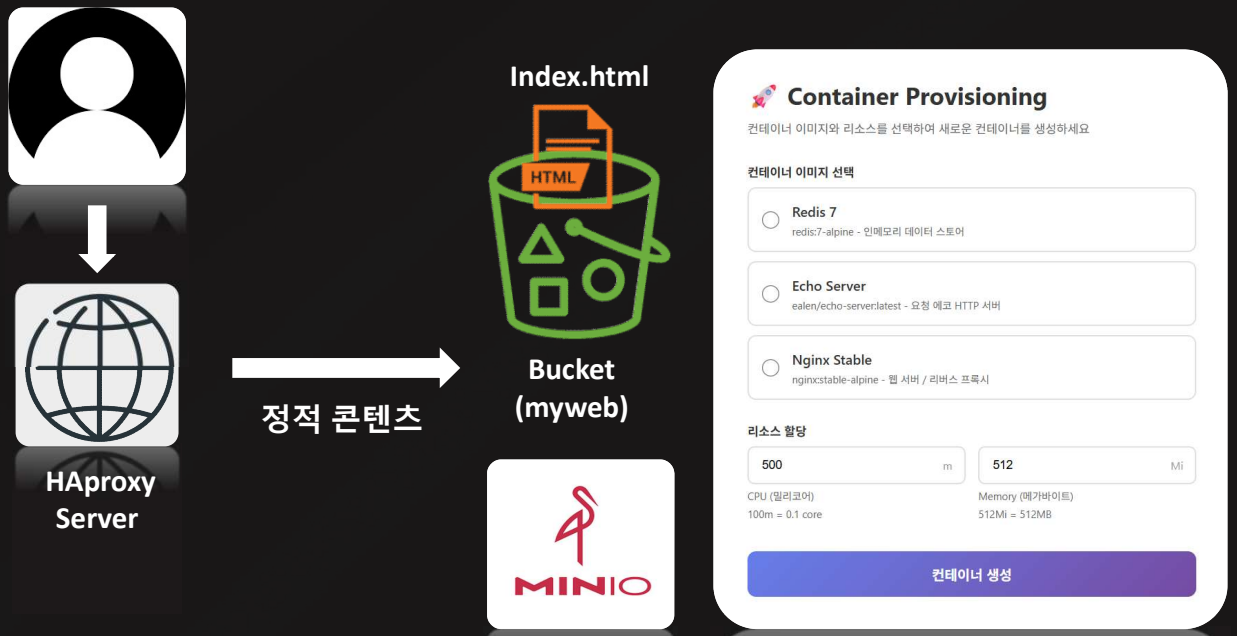

kubernetes
K3S
Deploy 생성 및 관리
Dynamo DB
생성 된 Deploy Log 저장
사용자
CLOUDFLARE
내부 서버를 직접 노출하지 않고
외부에서 접근을 도와주는
임시 HTTPS 터널 서비스
WEB Server (Haproxy)
정적 요청과 동적 요청을 분기하는
웹 트래픽 로드밸런서
Lambda
- API Gateway에게 전달 받은 요청 처리
S3 Object Storage 역할
정적 WEB 사이트 호스팅(9000/9001)
MINIO
LocalStack
AWS 클라우드 환경을 로컬에서
재현하는 가상화 플랫폼(4566)
GW API
- 사용자 api 요청이 오면 Lambda로 전달
IAM
- AWS 서비스간 권한 관리
- 역할과 정책으로 접근 제어
SQS
- 사용자의 API 요청을 Queue에 저장 순차 처리
veeAM
VM Backup
AWS CLI
AWS Cli Container
AWS 서비스 프로비저닝 및 관리
AWS 서비스 비서 역할

MINIO

S3 호환 오브젝트 스토리지

AWS S3와 동일한 환경 제공 하며 정적 웹 리소스 (HTML , CSS , Image)를 별도 WEB 서버 없이

직접 호스팅 가능 & 저장소 및 배포 역할 수행



AWS Cli 에서 MinIO 접근 값을 입력 한다면
원격Bucket 생성 및 정책 지정 파일 업로드 수정 가능

```
sh-5.2# export AWS_ACCESS_KEY_ID=minioadmin
sh-5.2# export AWS_SECRET_ACCESS_KEY=minioadmin
sh-5.2# export AWS_DEFAULT_REGION=us-east-1
```

```
sh-5.2# aws --endpoint-url http://192.168.65.176:9000 s3 mb s3://myapp
make_bucket: myapp
```

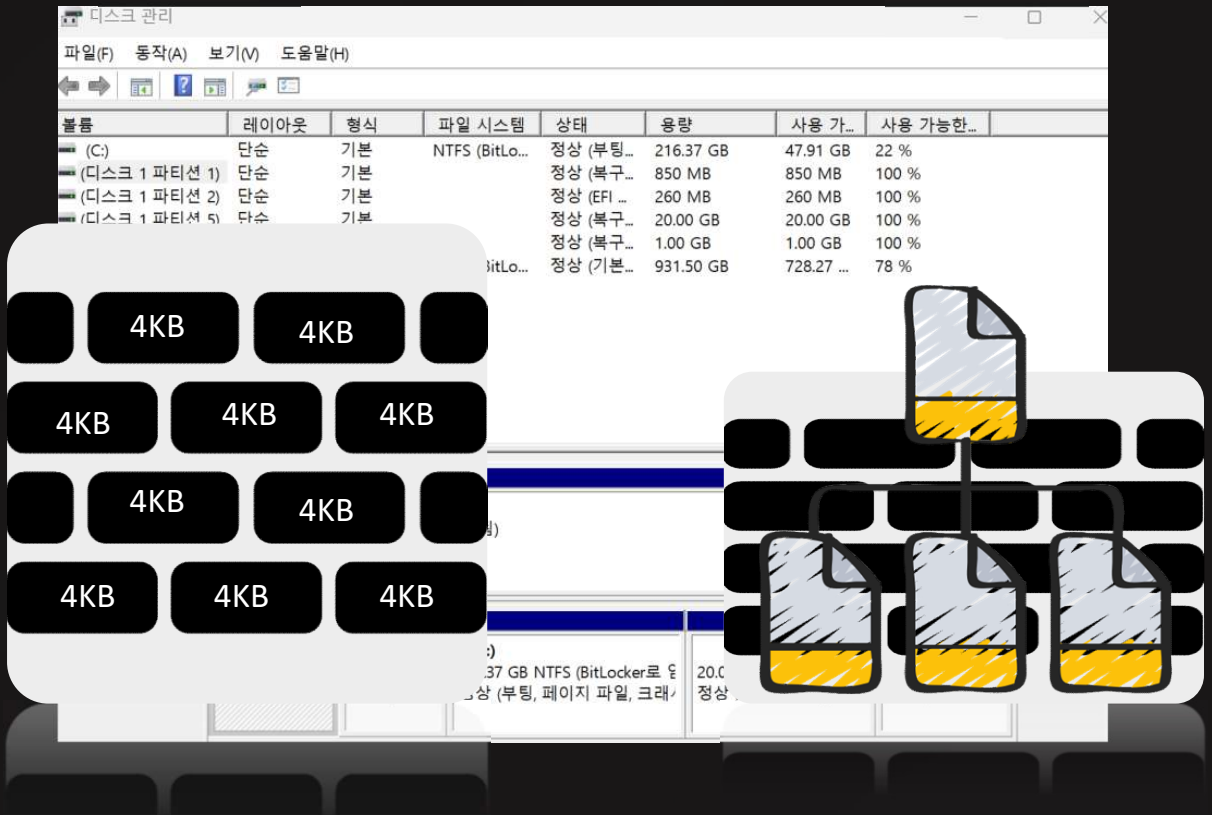
```
drwxrwxrwx  1 vagrant vagrant    0 Oct 23 00:26 ./
drwxr-xr-x 22 root    root    4096 Oct 23 01:37 ../
drwxrwxrwx  1 vagrant vagrant    0 Oct 23 00:27 .vagrant/
-rwxrwxrwx  1 vagrant vagrant 2132 Oct 29 01:10 Vagrantfile*
root@minIO:/vagrant# aws --endpoint-url http://192.168.65.176:9000 \
> s3 cp /vagrant/Vagrantfile s3://myapp/Vagrantfile \
> --content-type "text/plain"
upload: ./Vagrantfile to s3://myapp/Vagrantfile
```

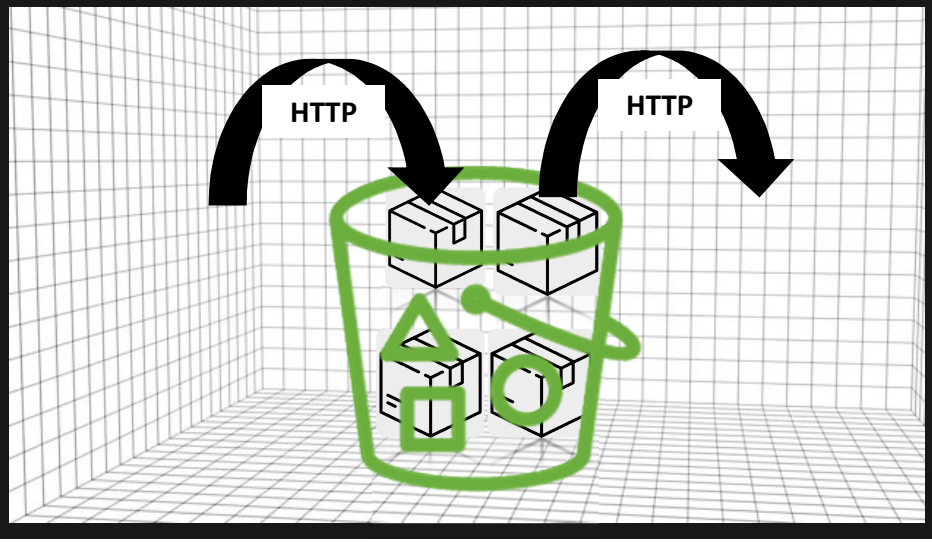
```
--bucket myweb \
--policy '{
  "Version":"2012-10-17",
  "Statement":[{
    "Effect":"Allow",
    "Principal":"*",
    "Action":["s3:GetObject"],
    "Resource":["arn:aws:s3:::myweb/*"]
  }]
}'
```

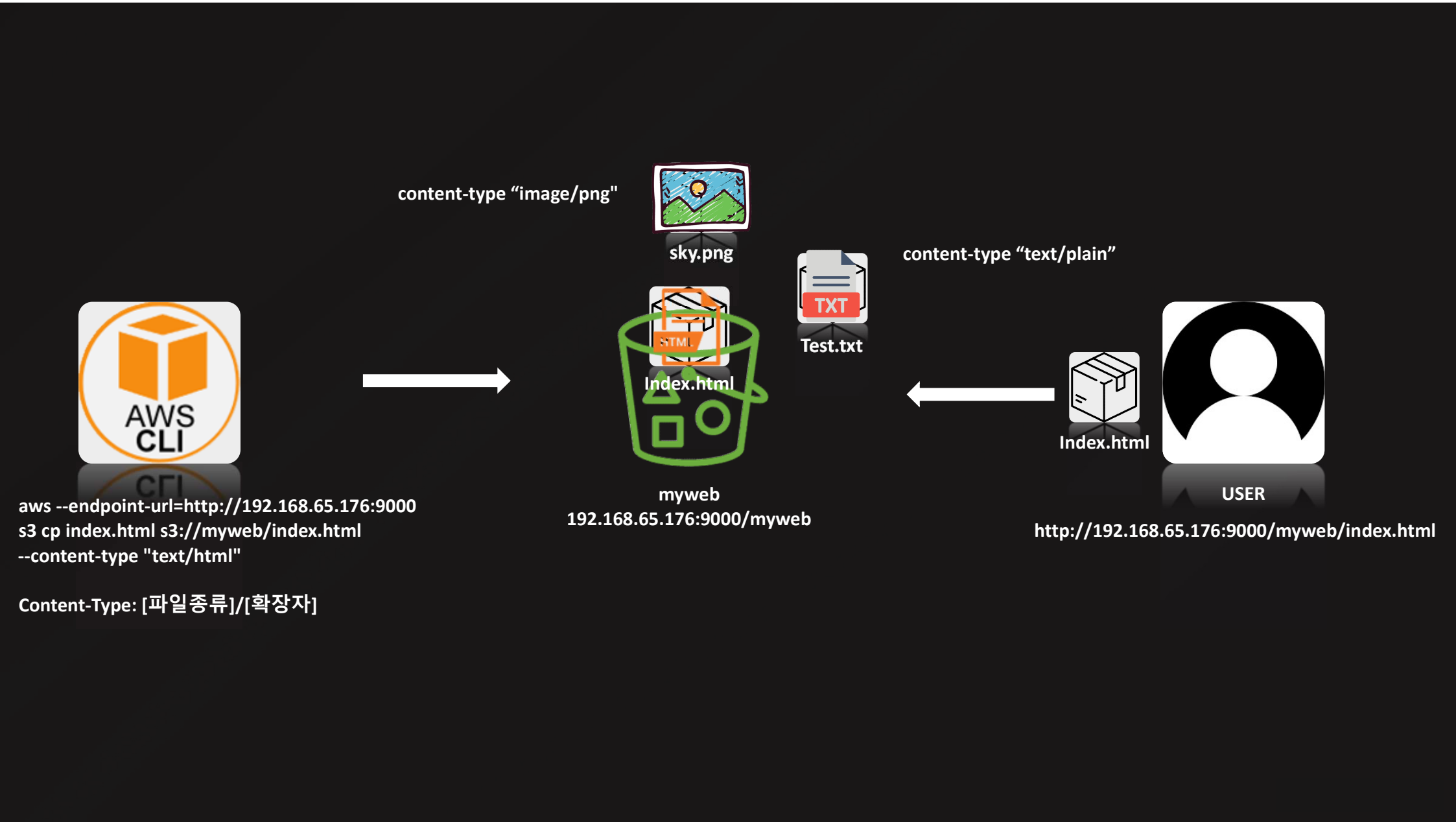
Block
Storage

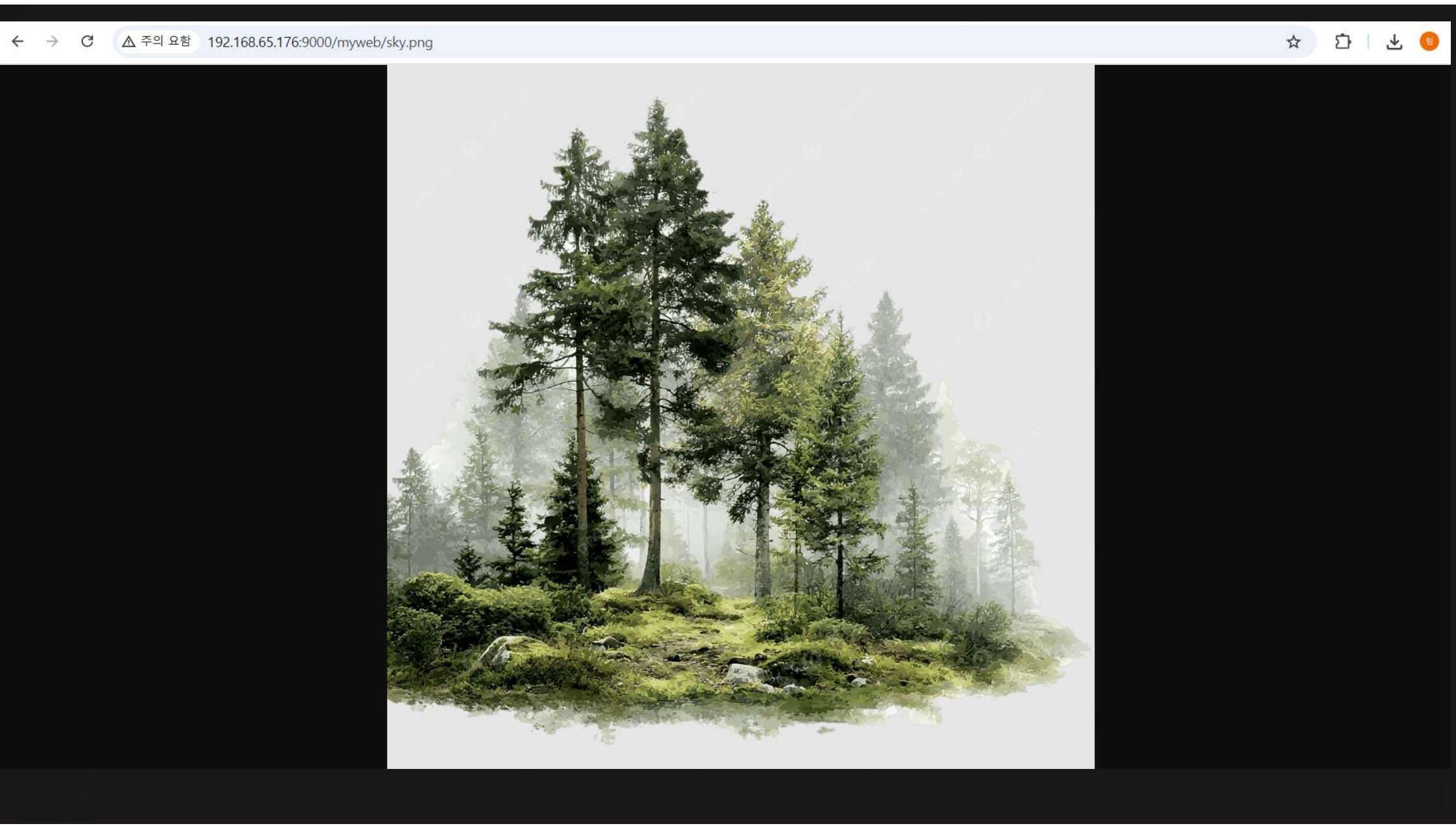
File
Storage

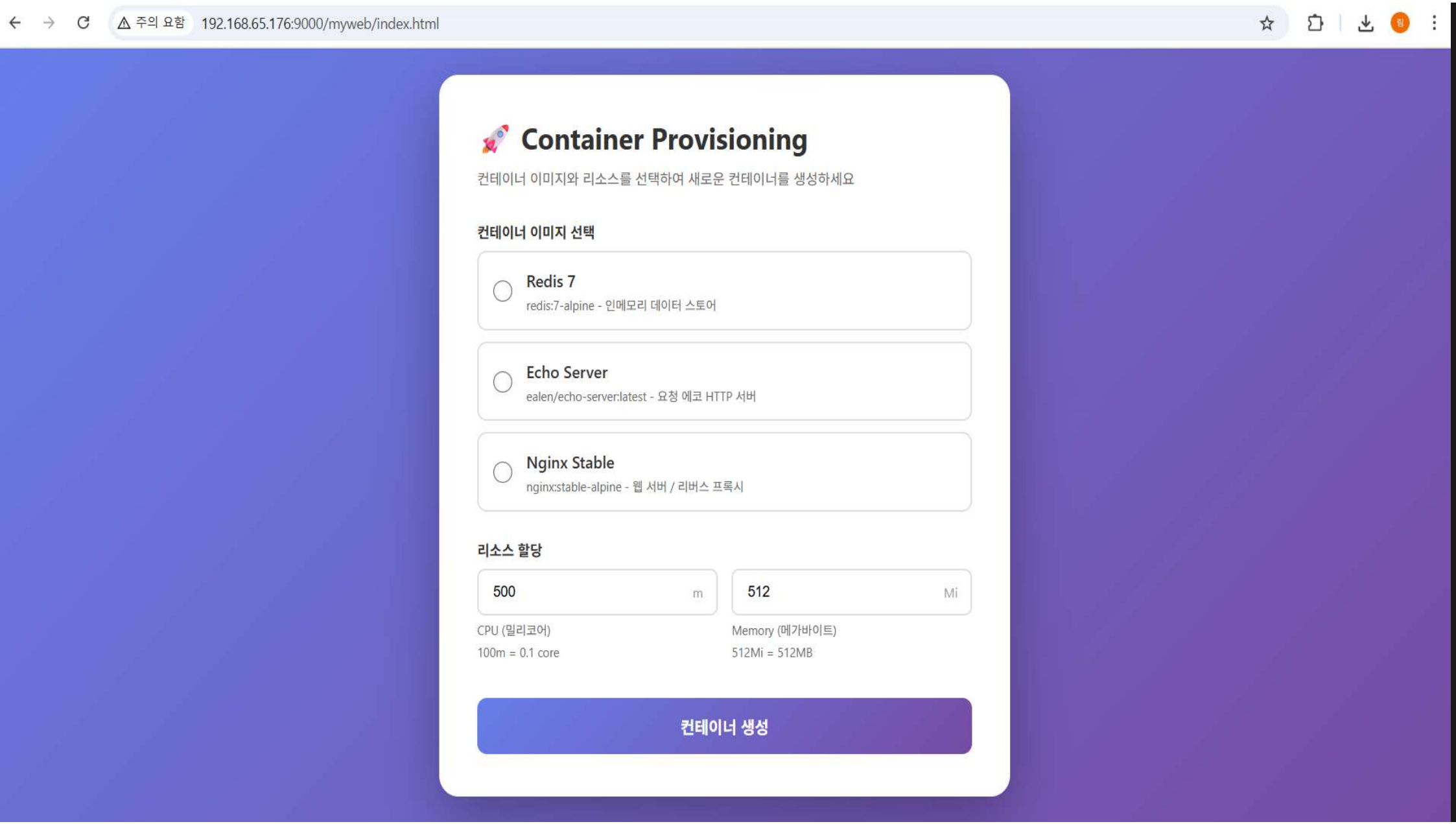
Object
Storage

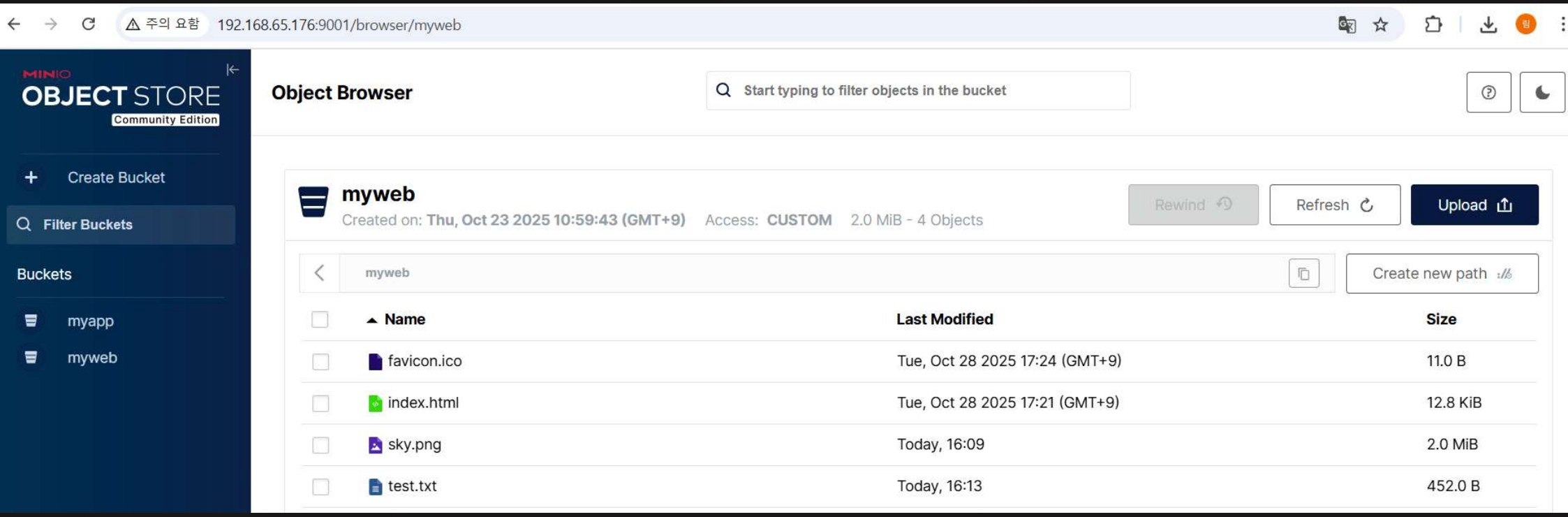





content-type “image/png"
sky.png
content-type “text/plain”
TXT
Test.txt
HTML
Index.html
AWS
CLI
Index.html
USER
aws --endpoint-url=http://192.168.65.176:9000
s3 cp index.html s3://myweb/index.html
--content-type "text/html"
Content-Type: [파일종류]/[확장자]
myweb
192.168.65.176:9000/myweb
http://192.168.65.176:9000/myweb/index.html


주의 요함
192.168.65.176:9000/myweb/sky.png


주의 요함
192.168.65.176:9000/myweb/index.html
Container Provisioning
컨테이너 이미지와 리소스를 선택하여 새로운 컨테이너를 생성하세요
컨테이너 이미지 선택
Redis 7
redis:7-alpine - 인메모리 데이터 스토어
Echo Server
ealen/echo-server:latest - 요청 에코 HTTP 서버
Nginx Stable
nginx:stable-alpine - 웹 서버 / 리버스 프록시
리소스 할당
500
m
512
Mi
CPU (밀리코어)
100m = 0.1 core
Memory (메가바이트)
512Mi = 512MB
컨테이너 생성




```
sh-5.2# aws --endpoint-url=http://192.168.65.176:9000 s3 ls s3://myweb/
2025-10-28 08:24:42         11 favicon.ico
2025-10-28 08:21:16      13063 index.html
2025-10-30 07:09:10    2064418 sky.png
2025-10-30 07:13:39        452 test.txt
```

CLOUDFLARE

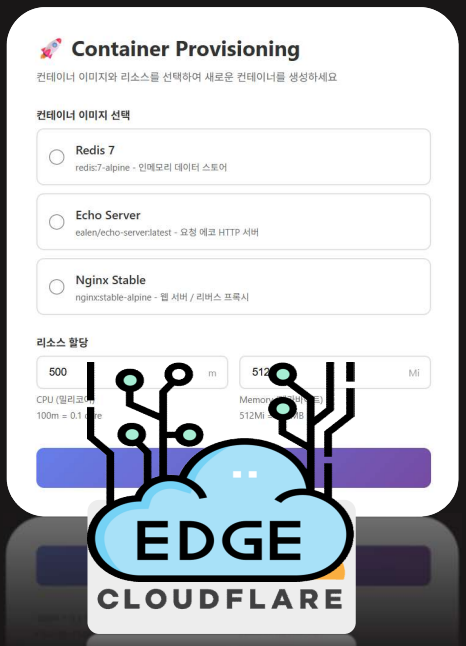
외부와 내부를 연결하는 임시 터널

외부 사용자가 내부(WEB Server)에 접근할 수 있도록
HTTPS 터널을 제공하는 게이트웨이

공인 IP나 IGW가 없어도 로컬 네트워크(192.168.65.0/24)를
임시 도메인으로 외부에 노출

```
root@webtest:/opt/cloudflared# cloudflared tunnel --url http://localhost:80
2025-10-28T08:29:48Z INF Thank you for trying Cloudflare Tunnel. Doing so, without a Cloudflare accoun
t and try it out. However, be aware that these account-less Tunnels have no uptime guarantee, are subj
ervices Terms of Use (https://www.cloudflare.com/website-terms/), and Cloudflare reserves the right to
ls for violations of such terms. If you intend to use Tunnels in production you should use a pre-creat
https://developers.cloudflare.com/cloudflare-one/connections/connect-apps
2025-10-28T08:29:48Z INF Requesting new quick Tunnel on trycloudflare.com...
2025-10-28T08:29:51Z INF +--------------------------------------------------------------------------------
2025-10-28T08:29:51Z INF |  Your quick Tunnel has been created! Visit it at (it may take some time to
2025-10-28T08:29:51Z INF |  https://expression-crucial-avoiding-agreements.trycloudflare.com
2025-10-28T08:29:51Z INF +--------------------------------------------------------------------------------
```

https://expression-crucial-avoiding-agreements.trycloudflare.com
=
http://localhost:80(Haproxy)
=
http://192.168.65.176:9000/myweb/index.html

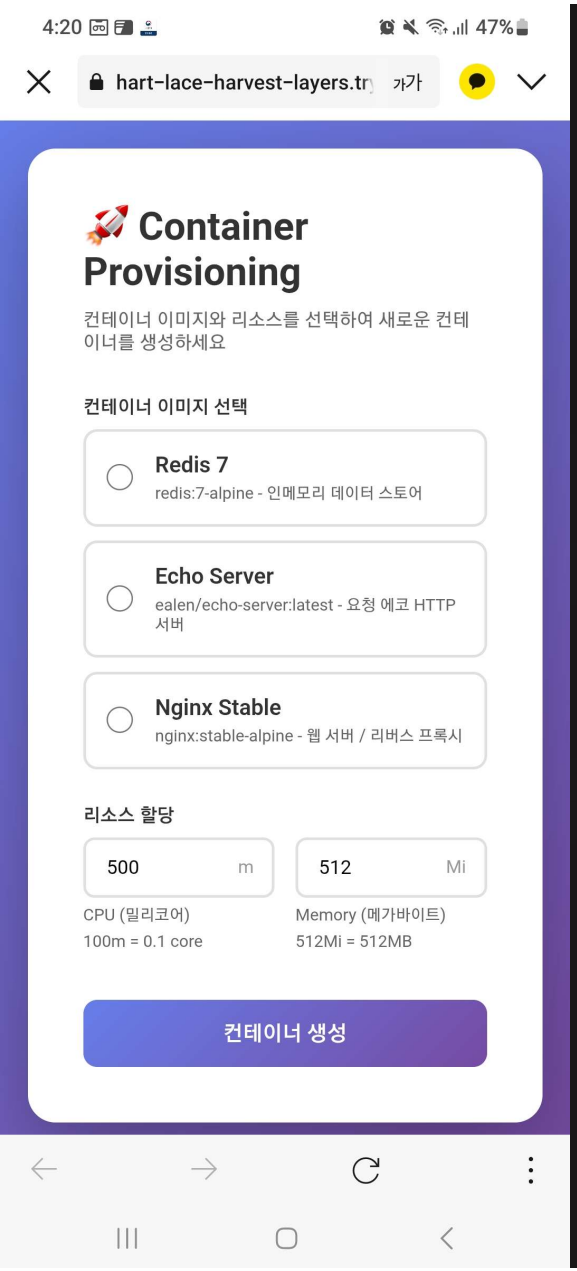


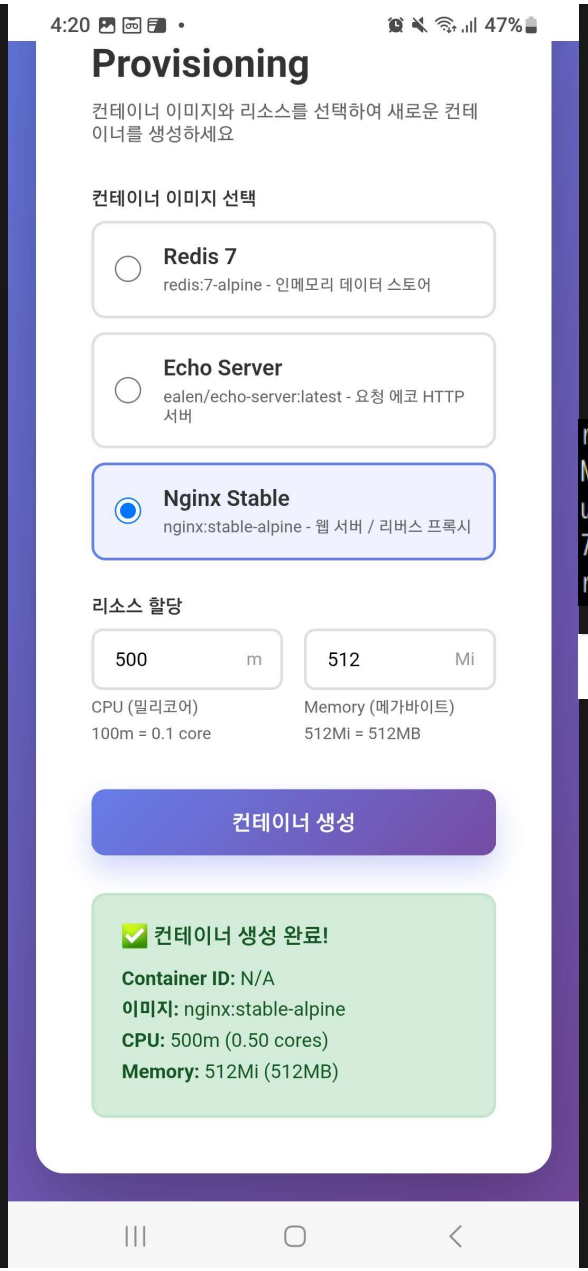
https://expression-crucial-avoiding-agreements.trycloudflare.com

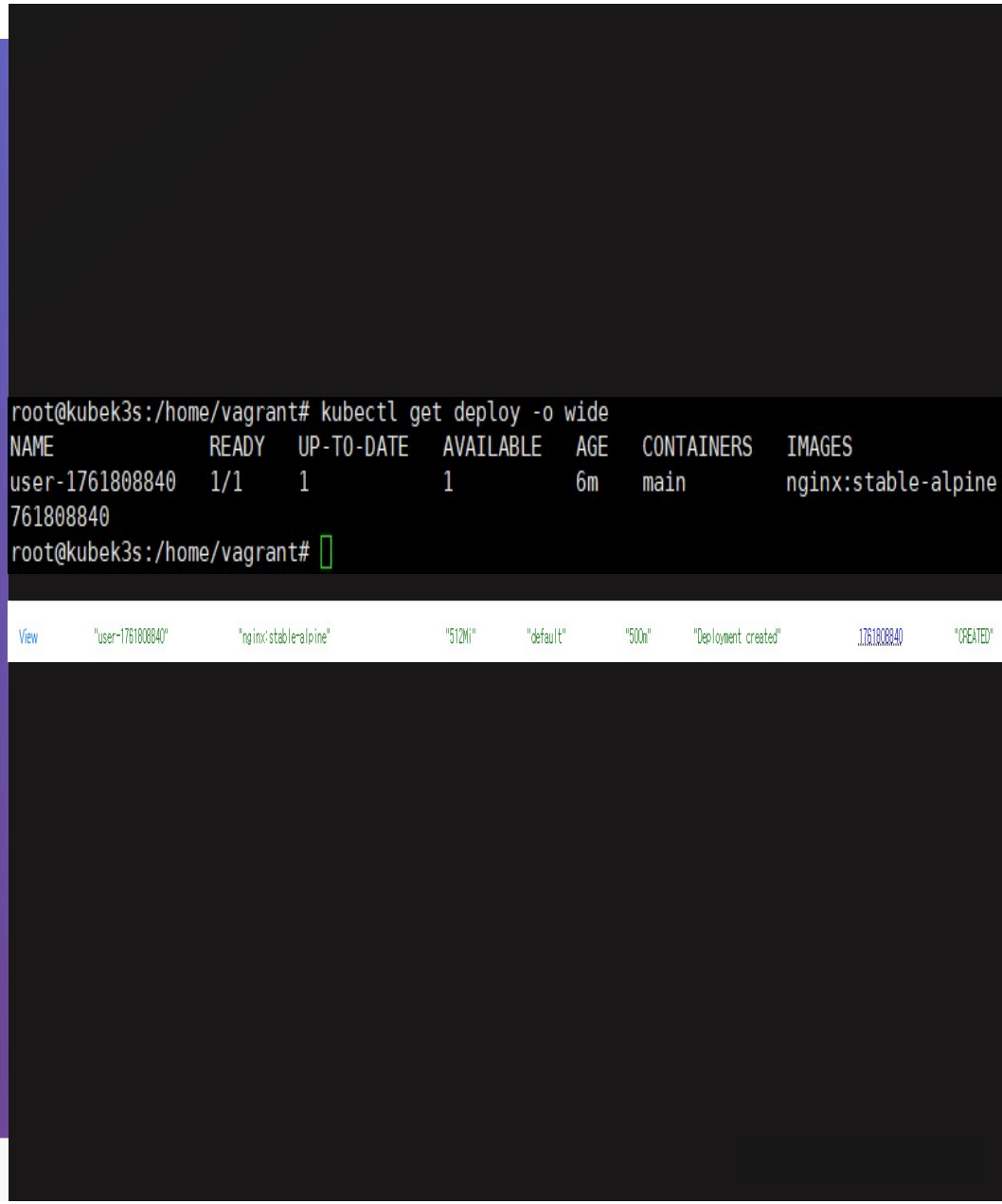
https://expression-crucial-avoiding-agreements.trycloudflare.com

외부 사용자

외부 사용자가 실제로 접속하는 건
MINIO가 아닌
Cloudflare Edge 서버


4:20
47%
hart-lace-harvest-layers.tr
가가
🚀 Container Provisioning
컨테이너 이미지와 리소스를 선택하여 새로운 컨테이너를 생성하세요
컨테이너 이미지 선택
Redis 7
redis:7-alpine - 인메모리 데이터 스토어
Echo Server
ealen/echo-server:latest - 요청 에코 HTTP 서버
Nginx Stable
nginx:stable-alpine - 웹 서버 / 리버스 프록시
리소스 할당
500
m
512
Mi
CPU (밀리코어)
100m = 0.1 core
Memory (메가바이트)
512Mi = 512MB
컨테이너 생성


4:20
47%
Provisioning
컨테이너 이미지와 리소스를 선택하여 새로운 컨테이너를 생성하세요
컨테이너 이미지 선택
Redis 7
redis:7-alpine - 인메모리 데이터 스토어
Echo Server
ealen/echo-server:latest - 요청 에코 HTTP 서버
Nginx Stable
nginx:stable-alpine - 웹 서버 / 리버스 프록시
리소스 할당
500
m
512
Mi
CPU (밀리코어)
100m = 0.1 core
Memory (메가바이트)
512Mi = 512MB
컨테이너 생성
✅ 컨테이너 생성 완료!
Container ID: N/A
이미지: nginx:stable-alpine
CPU: 500m (0.50 cores)
Memory: 512Mi (512MB)


root@kubek3s:/home/vagrant# kubectl get deploy -o wide
NAME READY UP-TO-DATE AVAILABLE AGE CONTAINERS IMAGES
user-1761808840 1/1 1 1 6m main nginx:stable-alpine
761808840
root@kubek3s:/home/vagrant#
View "user-1761808840" "nginx:stable-alpine" "512Mi" "default" "500m" "Deployment created" 1761808840 "CREATED"

WEB Server

정적 웹과 동적 API 트래픽을 분배

Haproxy를 이용하여 사용자의 요청을 받아 정적 콘텐츠는 S3(MinIO)

동적 콘텐츠는 Local Stack으로 전달하는 트래픽 분배 HUB

Cloudflare 임시 도메인
http://localhost:80(Haproxy)
정적 컨텐츠
http://192.168.65.176:9000/myweb/index.html

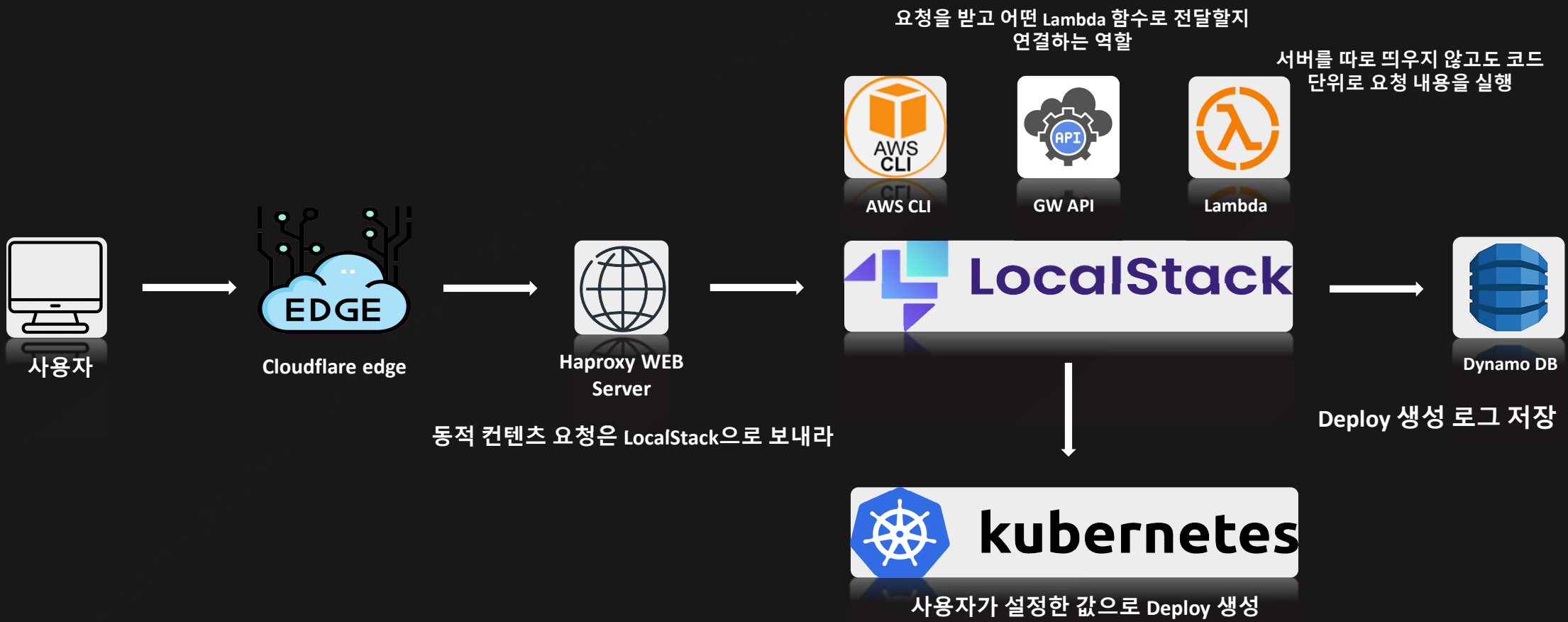
Haproxy WEB Server

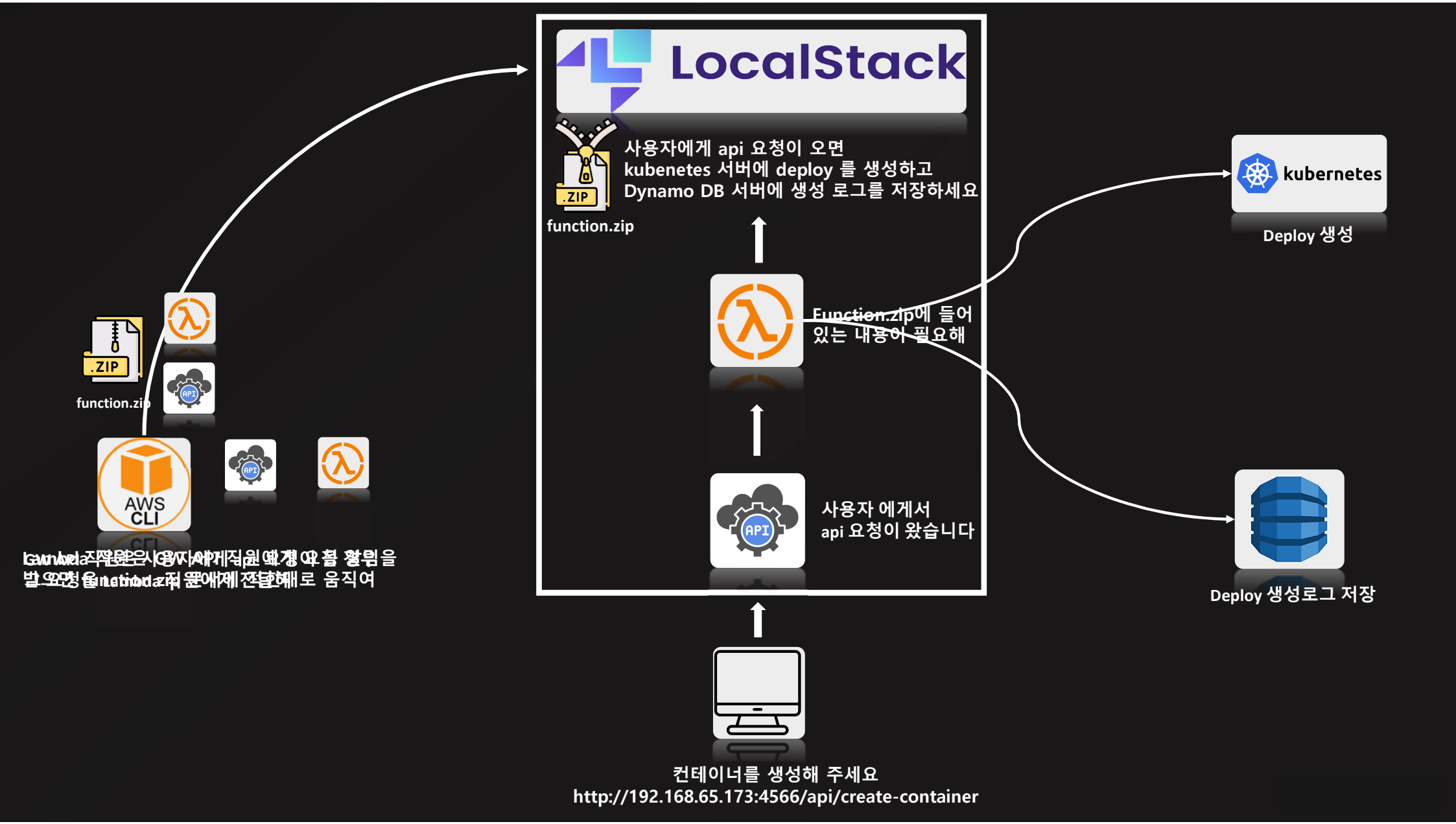
```
global
  daemon

defaults
  mode http
  timeout connect 5s
  timeout client  30s
  timeout server  30s

frontend http_in
  bind *:80
  acl is_api path_beg -i /api
  use_backend api_localstack if is_api
  default_backend static_minio

backend static_minio
  server minio 192.168.65.176:9000  check

backend api_localstack
  server lstack 192.168.65.173:4566 check
```

https://임시도메인/api/create-container

LocalStack

AWS 클라우드 환경을 로컬에서 재현할 수 있는 가상화 플랫폼




LocalStack
사용자에게 api 요청이 오면
kubenetes 서버에 deploy 를 생성하고
Dynamo DB 서버에 생성 로그를 저장하세요
function.zip
.ZIP
function.zip
Function.zip에 들어
있는 내용이 필요해
사용자 에게서
api 요청이 왔습니다
kubernetes
Deploy 생성
Deploy 생성로그 저장
AWS
CLI
컨테이너를 생성해 주세요
http://192.168.65.173:4566/api/create-container

function.zip

app.py
kubeconfig.yaml
requirements.txt

```
aws --endpoint-url=http://192.168.65.173:4566 \
  lambda create-function \
  --function-name createContainerFn \
  --runtime python3.9 \
  --handler app.handler \
  --zip-file fileb://function.zip \
  --role arn:aws:iam::000000000000:role/lambda-role
```

```
body = _parse_event(event)
image  = body.get("image")
cpu    = body.get("cpu")
memory = body.get("memory")
ns     = body.get("namespace", "default")

if not all([image, cpu, memory]):
   return _resp(400, {"error": "Missing required fields
(image/cpu/memory)"})
```

```
import boto3
ddb = boto3.resource("dynamodb", endpoint_url=os.getenv("DDB_ENDPOINT"))
table = ddb.Table(os.getenv("TABLE_NAME", "WebSubmissions"))
table.put_item(Item={
   "pk": created.metadata.name, "ts": int(time.time()),
   "namespace": ns, "image": image, "cpu": cpu, "memory": memory,
   "status": "CREATED", "message": "Deployment created",
})
```

```
from kubernetes import client, config

config.load_kube_config(config_file=os.path.join(os.path.dirname(__file__), "kubeconfig.yaml"))
apps_v1 = client.AppsV1Api()

name = f"user-{int(time.time())}"
container = client.V1Container(
   name="main",
   image=image,
   resources=client.V1ResourceRequirements(
      requests={"cpu": cpu, "memory": memory},
      limits={"cpu": cpu, "memory": memory},
   ),
)
deploy = client.V1Deployment(
   api_version="apps/v1",
   kind="Deployment",
   metadata=client.V1ObjectMeta(name=name),
   spec=client.V1DeploymentSpec(
      replicas=1,
      selector=client.V1LabelSelector(match_labels={"app": name}),
      template=client.V1PodTemplateSpec(
         metadata=client.V1ObjectMeta(labels={"app": name}),
         spec=client.V1PodSpec(containers=[container]),
      ),
   ),
)
created = apps_v1.create_namespaced_deployment(namespace=ns, body=deploy)
```


```
{
  "path": "/create-container",
  "method": "POST",
  "integration": "AWS_PROXY",
  "lambda_uri": "arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/arn:aws:lambda:us-east-1:000000000000:function:createContainerFn/invocations"
}
bash-5.2#
```


```yaml
services:
  localstack:
    image: localstack/localstack:latest
    container_name: localstack
    ports:
      - "4566:4566"
    environment:
      - SERVICES=lambda,apigateway,iam,logs,cloudwatch,sts,s3
      - AWS_DEFAULT_REGION=us-east-1
      - LS_LOG=info
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - ./data:/var/lib/localstack
```

```
root@localstack:/home/vagrant# curl -s http://127.0.0.1:4566/_localstack/health | jq
{
  "services": {
    "acm": "disabled",
    "apigateway": "running",
    "cloudformation": "disabled",
    "cloudwatch": "running",
    "config": "disabled",
    "dynamodb": "disabled",
    "dynamodbstreams": "disabled",
    "ec2": "disabled",
    "es": "disabled",
    "events": "disabled",
    "firehose": "disabled",
    "iam": "available",
    "kinesis": "disabled",
    "kms": "disabled",
    "lambda": "running",
    "logs": "running",
    "opensearch": "disabled",
    "redshift": "disabled",
    "resource-groups": "disabled",
    "resourcegroupstaggingapi": "disabled",
    "route53": "disabled",
    "route53resolver": "disabled",
    "s3": "running",
    "s3control": "disabled",
    "scheduler": "disabled",
    "secretsmanager": "disabled",
    "ses": "disabled",
    "sns": "disabled",
    "sqs": "disabled",
    "ssm": "disabled",
    "stepfunctions": "disabled",
    "sts": "running",
    "support": "disabled",
    "swf": "disabled",
    "transcribe": "disabled"
  },
  "edition": "community",
  "version": "4.9.3.dev62"
}
```

Dynamo DB

HTTP 기반으로 실시간 요청을 DynamoDB에 저장하는 구조

http://192.168.65.178:8001/tables/WebSubmissions

Key – Value 방식으로 Dynamo DB에 저장 된 데이터 확인

Lambda가 컨테이너 생성 로그를
DynamoDB에 저장

user-1762139779를
오픈해줘

boto3는 Python 코드를 HTTP 요청으로 변환해
DynamoDB API에 전달

app.py

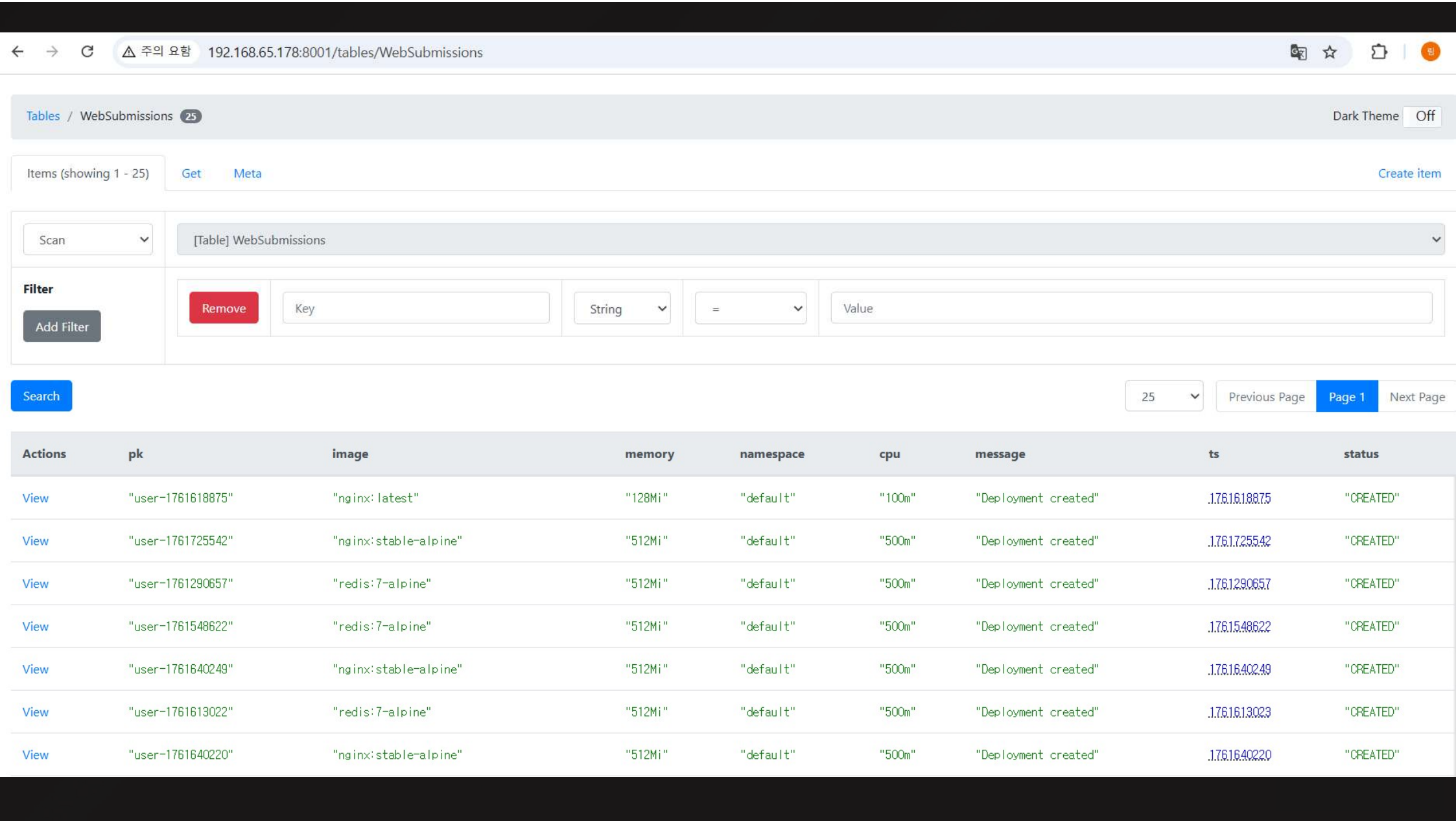
BOTO3

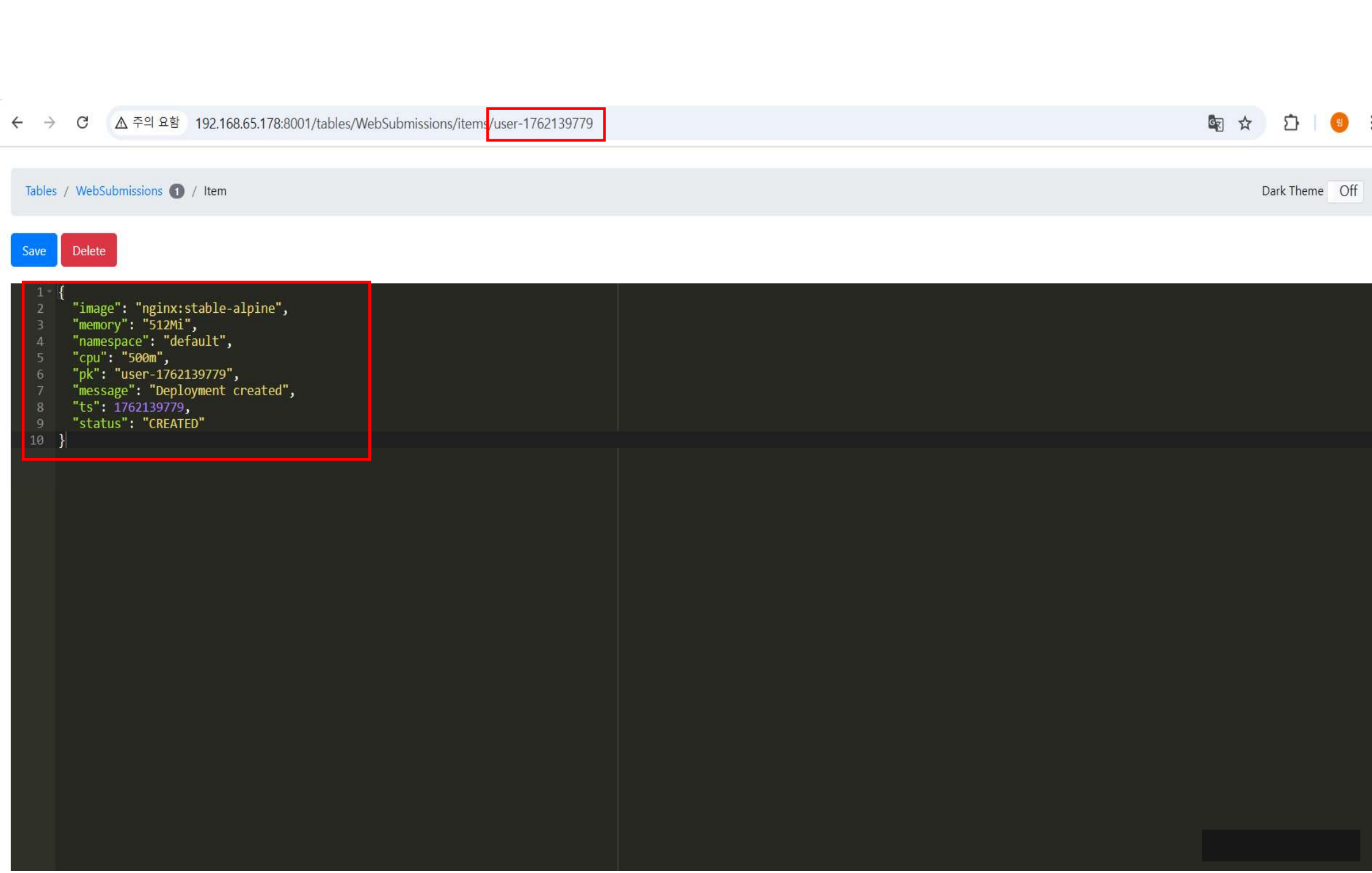
Dynamo DB API 호출

Key → 식별자
(예: user-1762139779)

Value → JSON 데이터

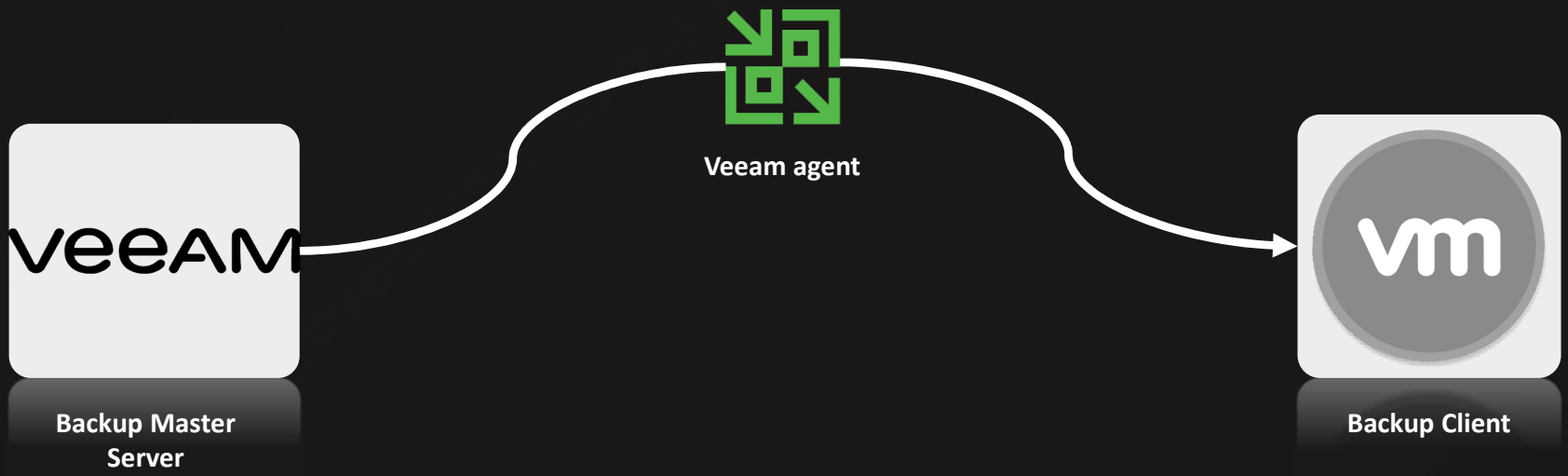

```
{
  "image": "nginx:stable-alpine",
  "cpu": "500m",
  "status": "CREATED",
  ...
}
```

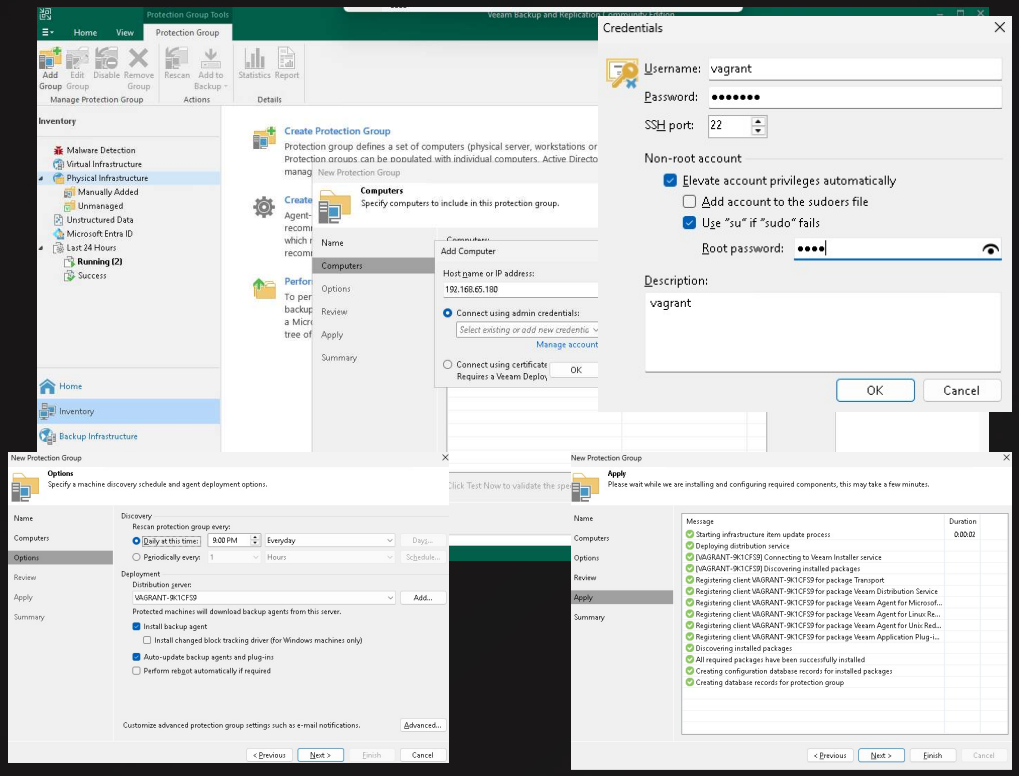

⚠ 주의 요함
192.168.65.178:8001/tables/WebSubmissions
Tables / WebSubmissions 25
Dark Theme Off
Items (showing 1 - 25)
Get
Meta
Create item
Scan
[Table] WebSubmissions
Filter
Add Filter
Remove
Key
String
=
Value
Search
25
Previous Page
Page 1
Next Page
Actions pk image memory namespace cpu message ts status
View "user-1761618875" "nginx:latest" "128Mi" "default" "100m" "Deployment created" 1761618875 "CREATED"
View "user-1761725542" "nginx:stable-alpine" "512Mi" "default" "500m" "Deployment created" 1761725542 "CREATED"
View "user-1761290657" "redis:7-alpine" "512Mi" "default" "500m" "Deployment created" 1761290657 "CREATED"
View "user-1761548622" "redis:7-alpine" "512Mi" "default" "500m" "Deployment created" 1761548622 "CREATED"
View "user-1761640249" "nginx:stable-alpine" "512Mi" "default" "500m" "Deployment created" 1761640249 "CREATED"
View "user-1761613022" "redis:7-alpine" "512Mi" "default" "500m" "Deployment created" 1761613023 "CREATED"
View "user-1761640220" "nginx:stable-alpine" "512Mi" "default" "500m" "Deployment created" 1761640220 "CREATED"

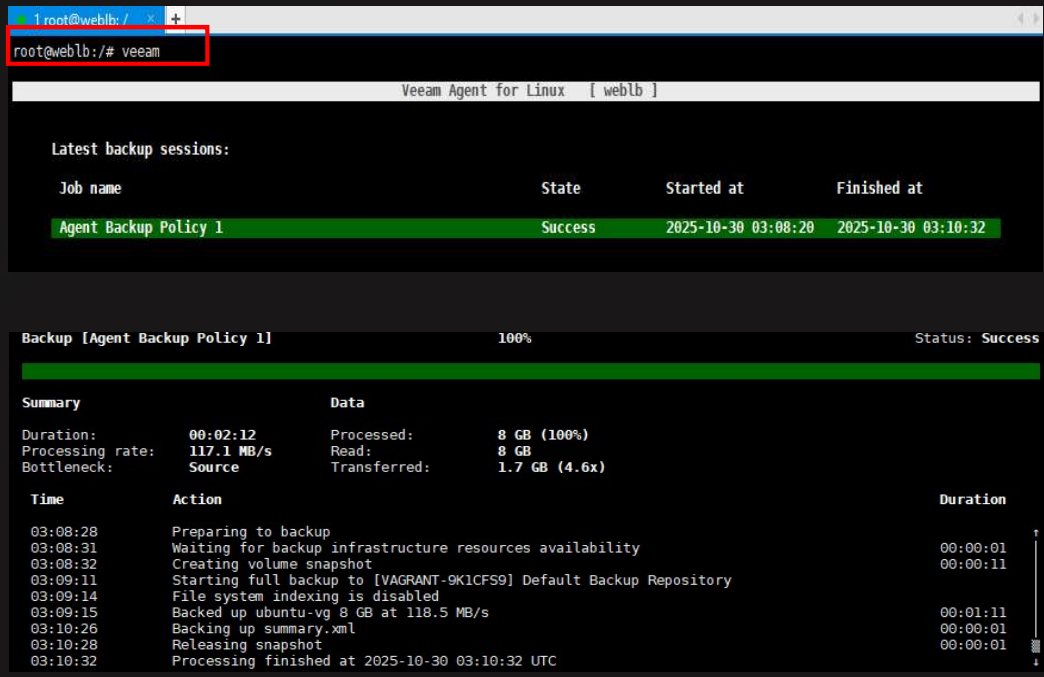

주의 요함
192.168.65.178:8001/tables/WebSubmissions/items/user-1762139779
Tables / WebSubmissions 1 / Item
Dark Theme Off
Save
Delete
{
"image": "nginx:stable-alpine",
"memory": "512Mi",
"namespace": "default",
"cpu": "500m",
"pk": "user-1762139779",
"message": "Deployment created",
"ts": 1762139779,
"status": "CREATED"
}

VEEAM

가상 머신 친화적 백업 소프트웨어

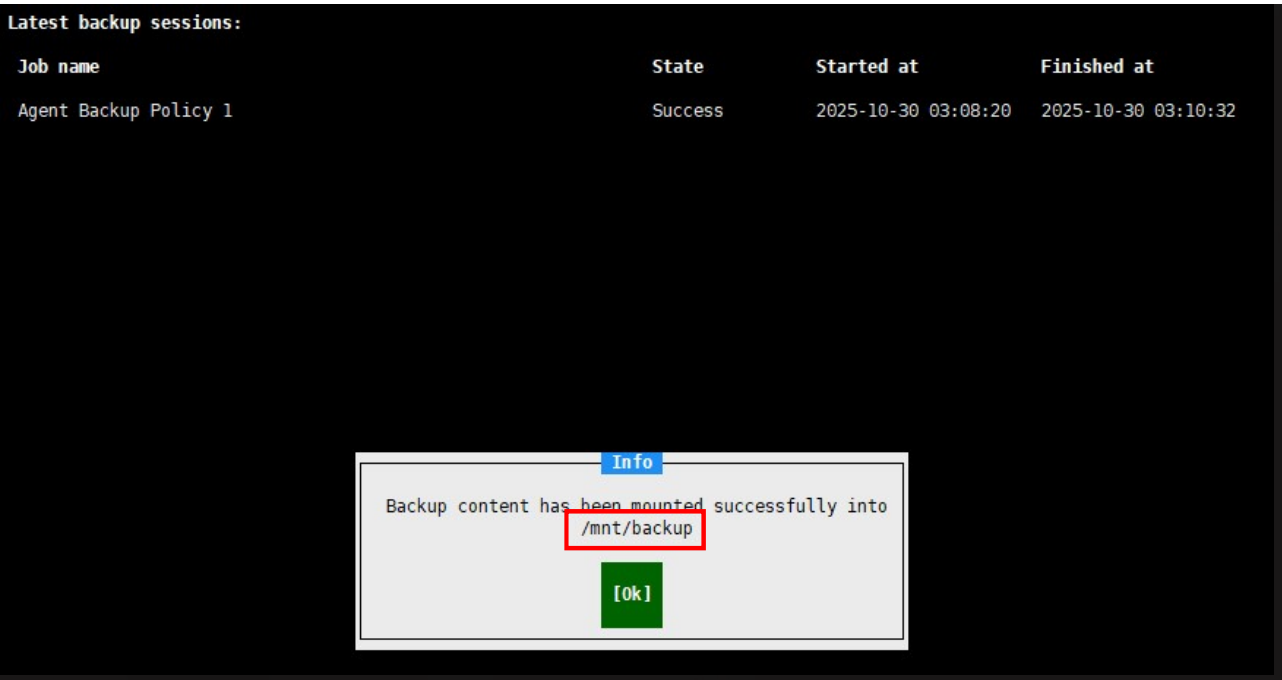


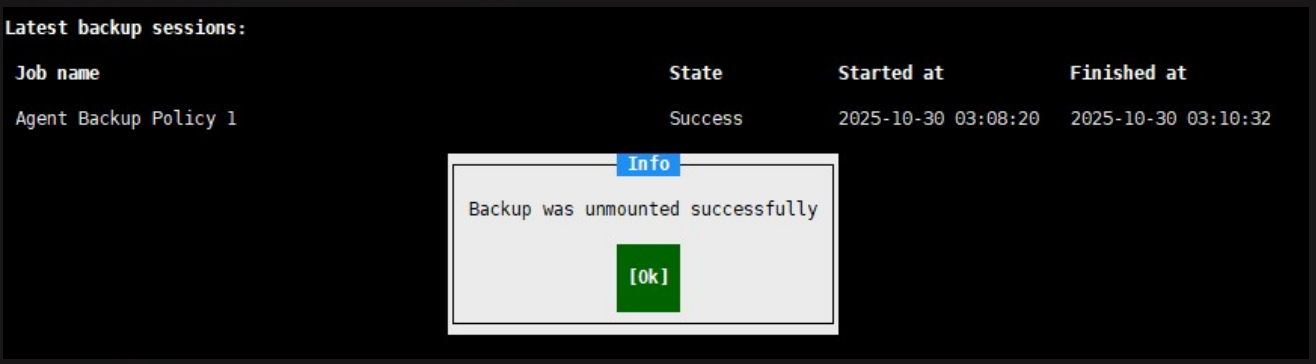




Veeam agent

Backup Client






```
root@weblb:/# cd /mnt/backup/
root@weblb:/mnt/backup# ll
total 3039336
drwxr-xr-x 22 root root        4096 Oct 30 03:08 ./
drwxr-xr-x  5 root root        4096 Oct 30 03:14 ../
lrwxrwxrwx  1 root root           7 Sep 11  2024 bin -> usr/bin/
drwxr-xr-x  2 root root        4096 Jul 28 23:24 boot/
dr-xr-xr-x  2 root root        4096 Sep 11  2024 cdrom/
drwxr-xr-x  4 root root        4096 Sep 11  2024 dev/
drwxr-xr-x 99 root root        4096 Oct 30 03:04 etc/
drwxr-xr-x  3 root root        4096 Jul 28 23:29 home/
lrwxrwxrwx  1 root root           7 Sep 11  2024 lib -> usr/lib/
lrwxrwxrwx  1 root root           9 Sep 11  2024 lib32 -> usr/lib32/
lrwxrwxrwx  1 root root           9 Sep 11  2024 lib64 -> usr/lib64/
lrwxrwxrwx  1 root root          10 Sep 11  2024 libx32 -> usr/libx32/
drwx------  2 root root       16384 Jul 28 23:24 lost+found/
drwxr-xr-x  2 root root        4096 Sep 11  2024 media/
drwxr-xr-x  4 root root        4096 Oct 29 01:24 mnt/
drwxr-xr-x  4 root root        4096 Oct 30 03:02 opt/
drwxr-xr-x  2 root root        4096 Apr 18  2022 proc/
drwx------  6 root root        4096 Oct 30 03:04 root/
drwxr-xr-x 14 root root        4096 Sep 11  2024 run/
lrwxrwxrwx  1 root root           8 Sep 11  2024 sbin -> usr/sbin/
drwxr-xr-x  6 root root        4096 Sep 11  2024 snap/
drwxr-xr-x  2 root root        4096 Sep 11  2024 srv/
-rw-------  1 root root  3112173568 Jul 28 23:25 swap.img
drwxr-xr-x  2 root root        4096 Apr 18  2022 sys/
drwxrwxrwt 15 root root        4096 Oct 30 03:04 tmp/
drwxr-xr-x 14 root root        4096 Sep 11  2024 usr/
drwxr-xr-x  2 root root        4096 Oct 29 01:24 vagrant/
drwxr-xr-x 13 root root        4096 Sep 11  2024 var/
drwxrws---  2 root veeam       4096 Oct 30 03:09 .veeamsnapstorage/
root@weblb:/mnt/backup# pwd
/mnt/backup
root@weblb:/mnt/backup# cd etc/haproxy/
root@weblb:/mnt/backup/etc/haproxy# ll
total 16
drwxr-xr-x  3 root root 4096 Oct 29 08:35 ./
drwxr-xr-x 99 root root 4096 Oct 30 03:04 ../
drwxr-xr-x  2 root root 4096 Oct 29 01:26 errors/
-rw-r--r--  1 root root 1367 Oct 29 02:35 haproxy.cfg
root@weblb:/mnt/backup/etc/haproxy# pwd
/mnt/backup/etc/haproxy
```