

EC2 to ECS

Migration 삽질기

Backend Engineer at PeopleFund

장원준

Who am I?

장원준

숭실대 소프트웨어학부 17학번

소프트웨어 마에스트로 9기

피플펀드에서 Backend Engineer로 병역특례중



STARTUP TECH CHALLENGE

이런 것들을 이야기 해보려 합니다.



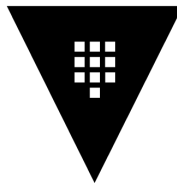
Amazon
EC2



AWS Codebuild

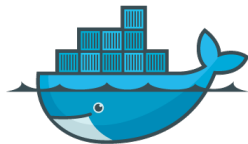


AWS Cloudwatch



HashiCorp

Vault



docker



AWS CodePipeline



AWS Lambda

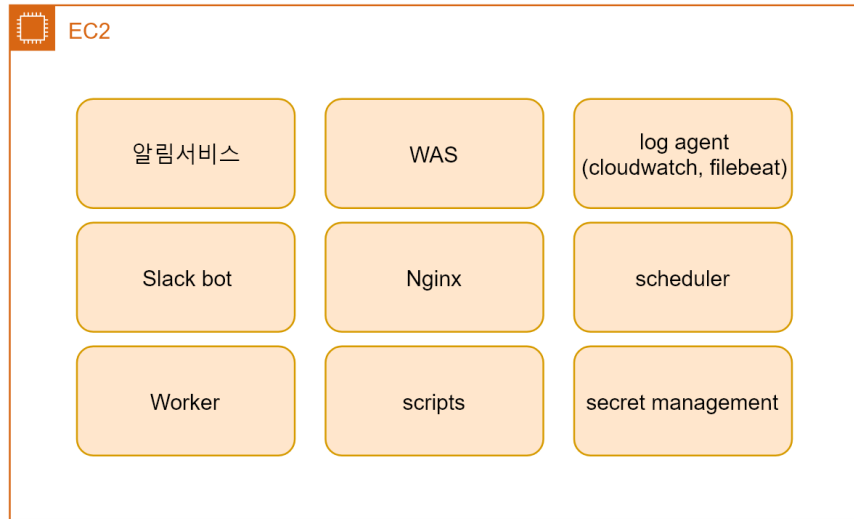


elasticsearch

On Demand EC2 system

한 EC2에 모든 기능 및 서비스가 옹기종기 모여있는 EC2.

하나하나 Legacy system을 container 기반 시스템으로 옮겨봅시다

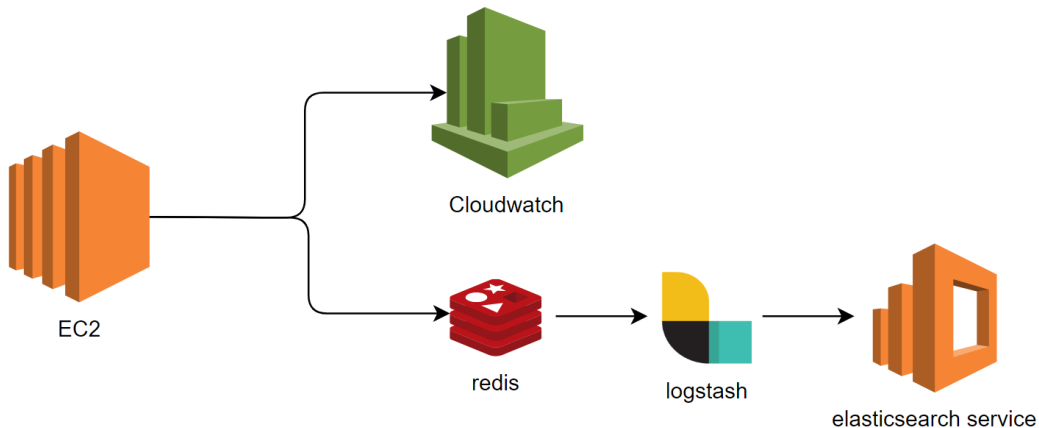


Log

기존 로그 시스템

Dockerizing을 시작하면서, 기존 log system을 확인해봄.

EC2 위에서 cloudwatch 와 ELK + filebeat로 로그를 보내는중.



기존 로그 시스템

XX 로그는 어디있을까요?

-> cloudwatch를 탐색한다.

-> kibana를 탐색한다.

-> 각 서버를 뒤져가며 vi some_log.log 를 열심히 검색한다



celery worker log



syslog

django log



에러 로그를 확인해야해요!

Traceback log가 한줄 한줄 찢려서 나옴.

multi line configuration이 안되어있는 상태

```
▶ 07:56:29 Traceback (most recent call last):
▶ 07:56:29 File "/usr/local/lib/python3.6/site-packages/celery/app/trace.py", line 382, in trace_task
▶ 07:56:29 R = retval = fun(*args, **kwargs)
▶ 07:56:29 File "/usr/local/lib/python3.6/site-packages/celery/app/trace.py", line 641, in __protected_call__
▶ 07:56:29 return self.run(*args, **kwargs)
▶ 07:56:29 File "/usr/local/lib/python3.6/site-packages/sentry_sdk/integrations/celery.py", line 66, in _inner
▶ 07:56:29 reraise(*_capture_exception())
▶ 07:56:29 File "/usr/local/lib/python3.6/site-packages/sentry_sdk/_compat.py", line 44, in reraise
▶ 07:56:29 raise value
▶ 07:56:29 File "/usr/local/lib/python3.6/site-packages/sentry_sdk/integrations/celery.py", line 64, in _inner
▶ 07:56:29 return f(*args, **kwargs)
▶ 07:56:29 File [REDACTED]
▶ 07:56:29 raise ValueError [REDACTED]
▶ 07:56:29 ValueError: [REDACTED]
```

로그 시스템 고치기 - multi line

날짜 prefix가 없으면 이전 line 과 병합하여 로그를 쌓도록 awslog config 추가.

Task definition

```
{
  "containerDefinitions": [
    {
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-group": "/ecs/celery-docker",
          "awslogs-region": "ap-northeast-2",
          "awslogs-stream-prefix": "flower",
          "awslogs-datetime-format": "\\[%Y-%m-%d %H:%M:%S"
        }
      },
      "dnsSearchDomains": [],
      "entryPoint": [],
      "portMappings": [
```

Docker compose

```
logging:
  driver: awslogs
  options:
    awslogs-region: ${AWSLOGS_REGION}
    awslogs-group: ${AWSLOGS_GROUP}
    awslogs-datetime-format: '\\[%Y-%m-%d %H:%M:%S'
    tag: celery.{{.Name}}
```


Log pipeline - ELK

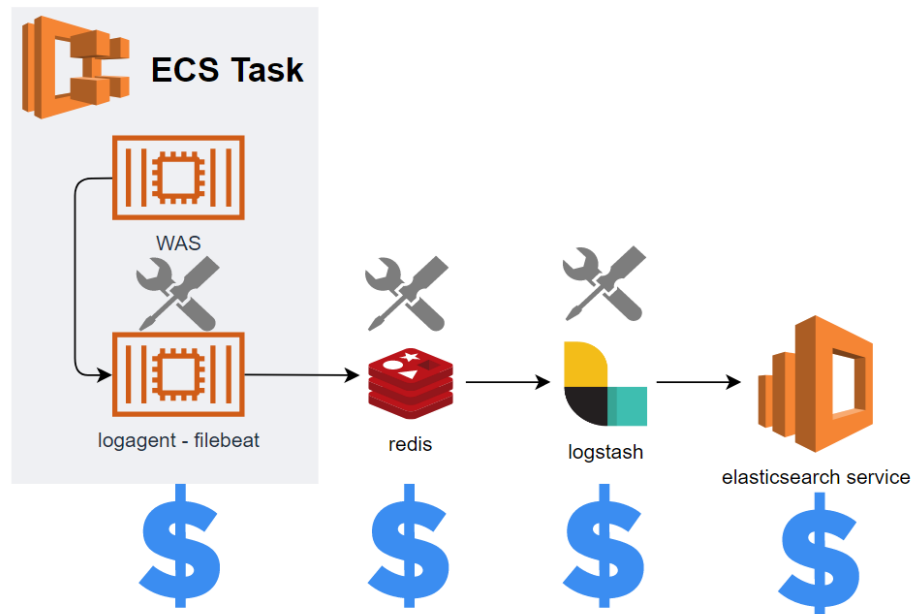
첫 번째로 시도 했던것 은
ELK로 모든 로그를 넣는것.

Log -> filebeat agent container -> redis

redis -> logstash -> elasticsearch

장점: 빠른 중앙화된 로그 수집.

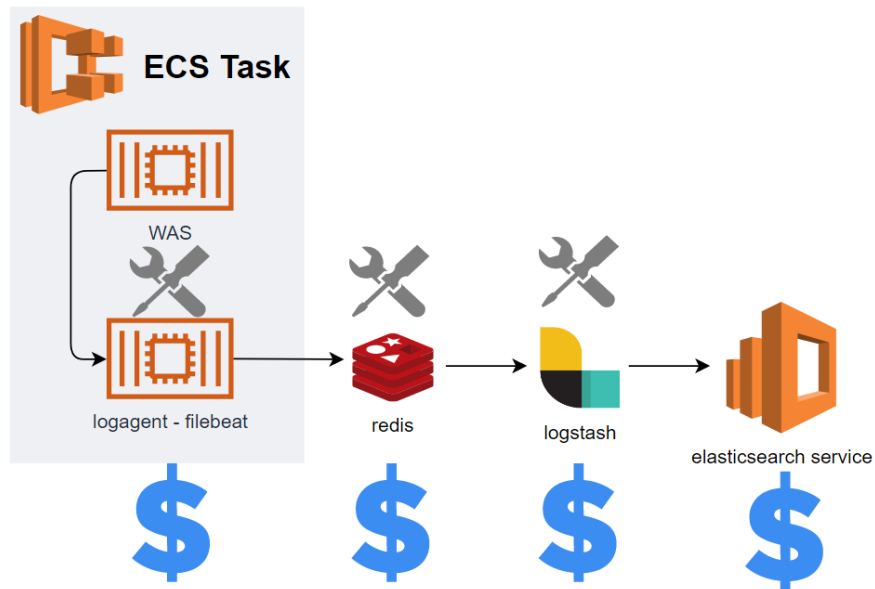
단점: 관리 포인트가 많아짐. 비용 증가



Log pipeline - ELK

logstash -> elasticsearch 연결이 자주 끊겨서 로그 유실.

Attempted to send a bulk request to elasticsearch, but no there are no living connections in the connection pool.
Perhaps Elasticsearch is unreachable or down?

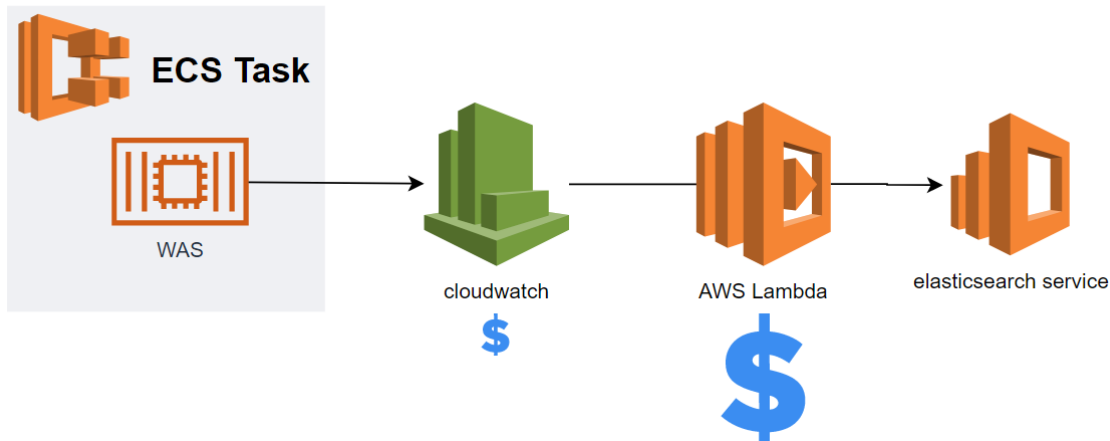


Log pipeline - cloudwatch + lambda

Cloudwatch로 로그를 보낸 후, lambda
를 사용하여 elasticsearch로 stream

장점: 관리 포인트 적음.

단점: 로그 수에 따른 선형적인 과금 구조

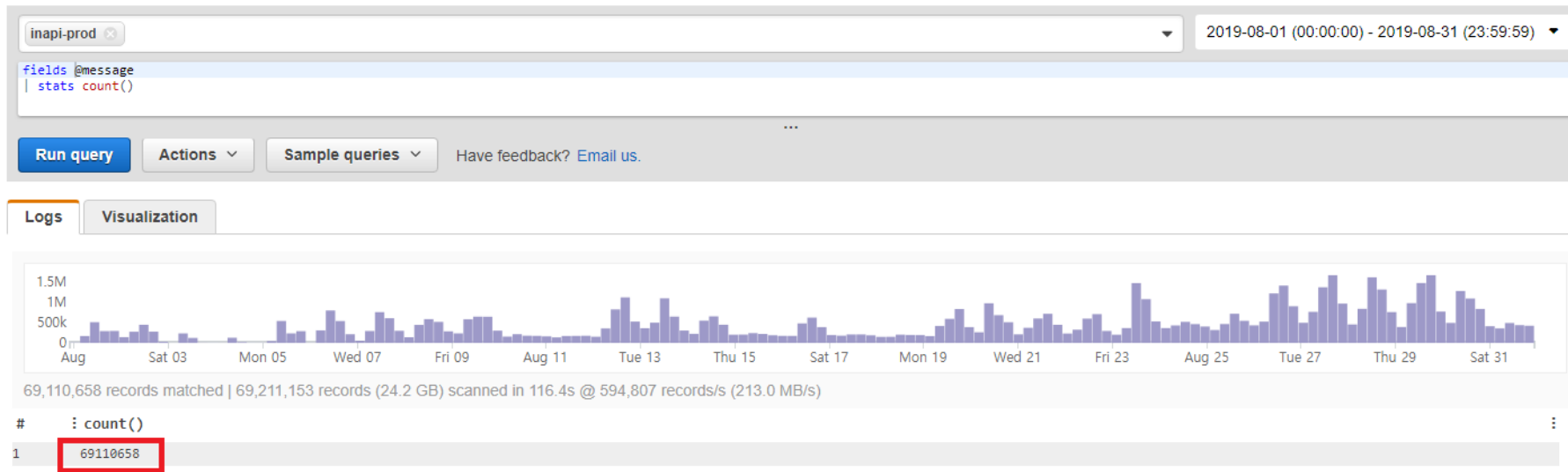


lambda stream 과금 계산

1달동안 대략 7억건의 log 발생.

lambda pricing 계산시 1달에 27~28 달러 과금.

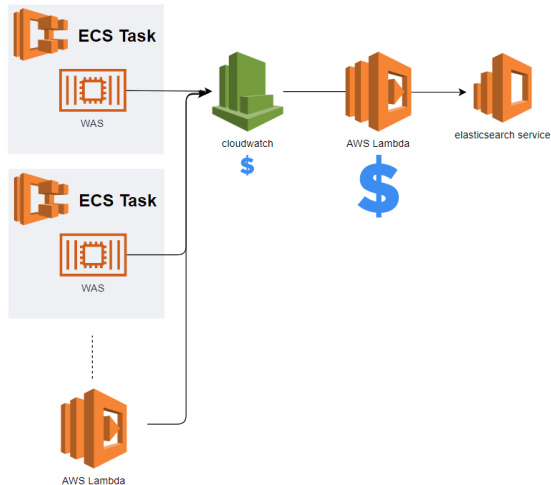
logstash 와 redis 가격 단순 계산: $15.048 + 37.44 =$ 대략 52~53달러



Lambda stream 확장성

cloudwatch log group, log stream으로
index 를 달아줄 수 있음.

하나의 lambda function 정의를 통해 모든
cloudwatch log elasticsearch로 통합 가능.



```
function transform(payload) {  
  if (payload.messageType === 'CONTROL_MESSAGE') {  
    return null;  
  }  
  
  var bulkRequestBody = '';  
  
  payload.logEvents.forEach(function(logEvent) {  
    var timestamp = new Date(1 * logEvent.timestamp);  
  
    // index name format: logGroup-logStream-YYYY.MM.DD  
    var indexName = [  
      'log-' + payload.logGroup.replace(/\\|\/|\?|"|<|>|\\|/gi, "-")  
      + '-' + payload.logStream.replace(/\\|\/|\?|"|<|>|\\|/gi, "-")  
      + '-' + timestamp.getUTCFullYear(),           // year  
      ('0' + (timestamp.getUTCMonth() + 1)).slice(-2), // month  
      ('0' + timestamp.getUTCDate()).slice(-2)       // day  
    ].join('.');  
  
    var source = buildSource(logEvent.message, logEvent.extractedFields);  
    source['@id'] = logEvent.id;  
    source['@timestamp'] = new Date(1 * logEvent.timestamp).toISOString();  
    source['@message'] = logEvent.message;  
    source['@owner'] = payload.owner;  
    source['@log_group'] = payload.logGroup;  
    source['@log_stream'] = payload.logStream;
```

Metric

STARTUP TECH CHALLENGE

Cloudwatch

기본적인 Metric은 Cloudwatch에서 서비스별로 확인할 수 있음.

CloudWatch: [Elastic Container Service](#) ▾

Time range 1h 3h 12h 1d 3d 1w custom ▾

Actions ▾



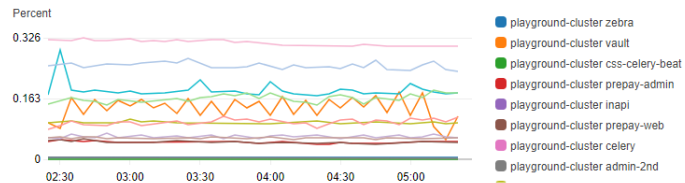
All resources ▾

Service dashboard ▾

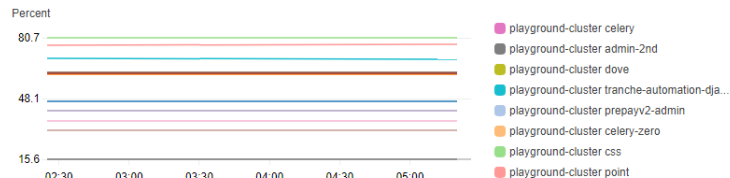
ALARM 0 INSUFFICIENT DATA 0 OK 0

Service Metrics

CPU Utilization Average

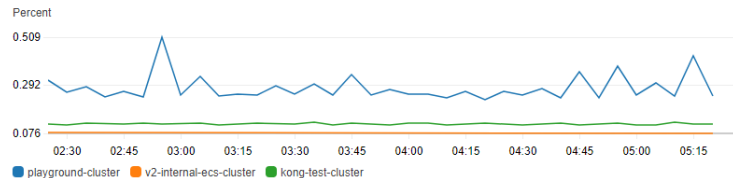


Memory Utilization Average

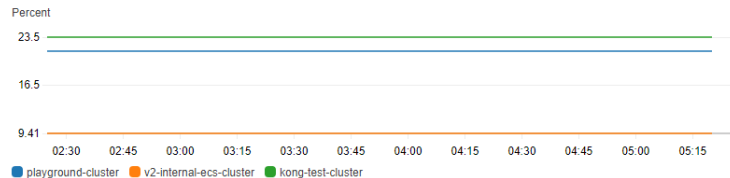


Cluster Metrics

CPU Utilization Average

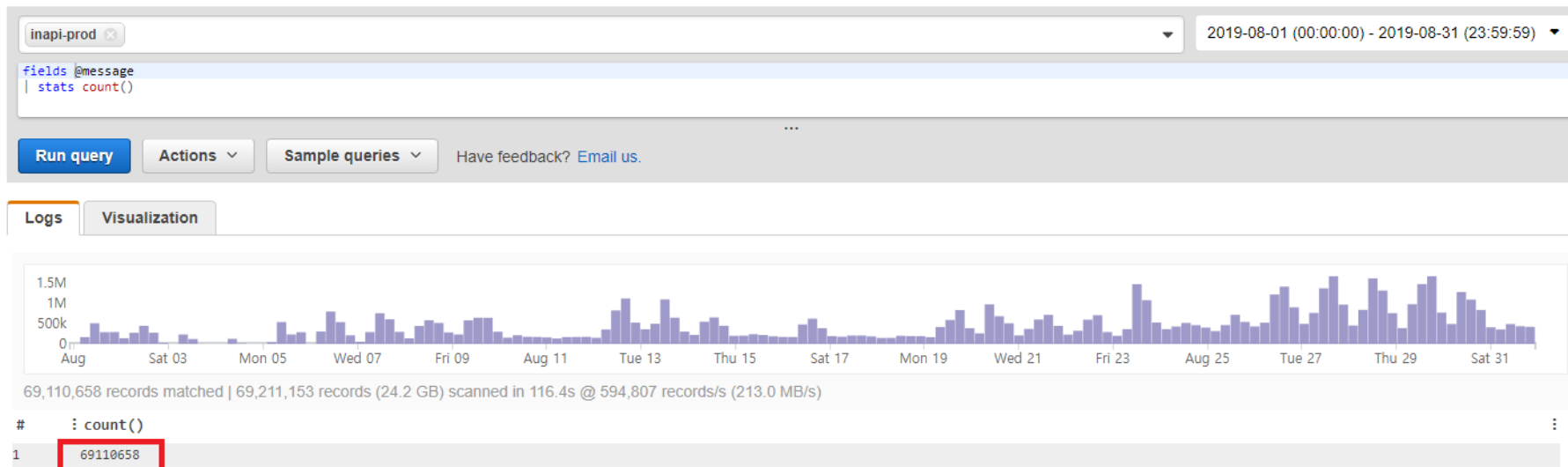


Memory Utilization Average



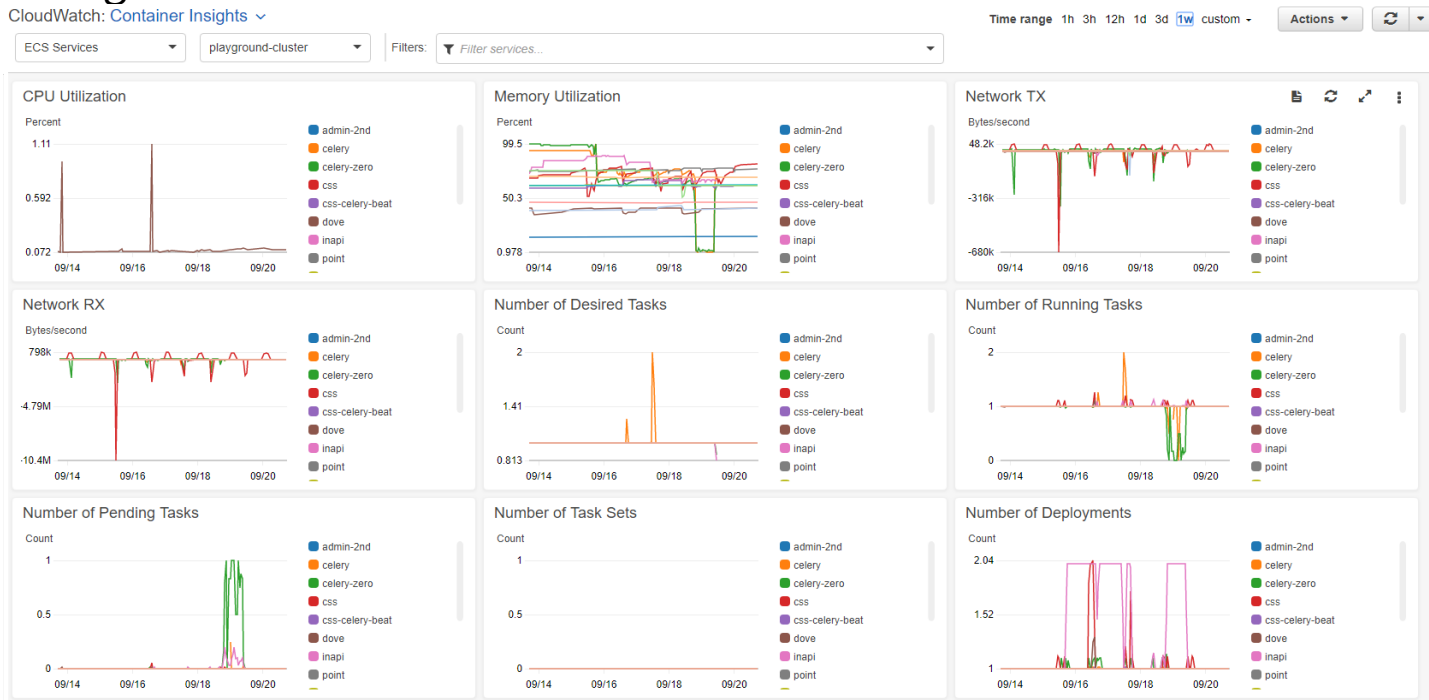
Cloudwatch Container insight

ECS 에서 Service별로 CPU, Memory, Network IO, Deploy, Task 등등 Metric을 로그 형태로 제공.
해당 로그를 AWS Cloudwatch logs insight 를 통해 custom Metric을 생성할 수 있음.



Cloudwatch

Container insight 를 활성화 한 cluster는 service별로 상세한 metric을 제공해줌.

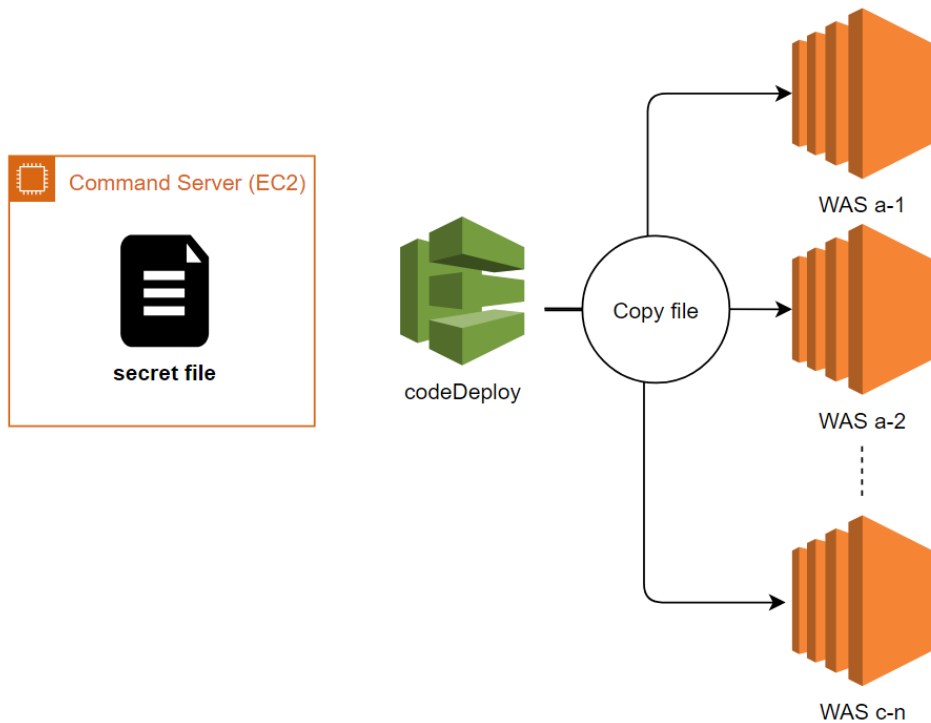


Secret

STARTUP TECH CHALLENGE

기존 Secret manage system

linux file system의 보안을 믿고 file에 암호화도 없이 secret이 저장되어있었음.

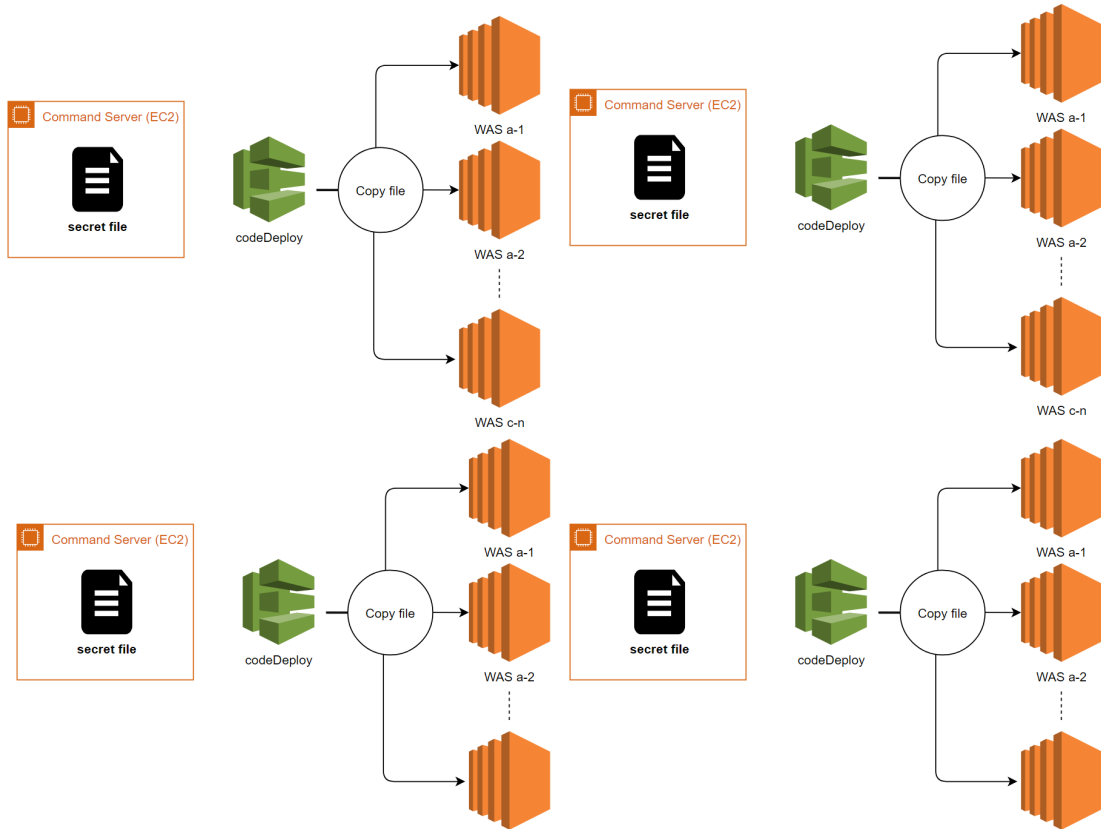


기존 Secret manage system

서비스가 점점 늘어나고,

dockerizing 하면서 더이상 파일로 관리 할 수 없어짐.

docker image 에 secret을 넣는것은 좋지 않기 때문.

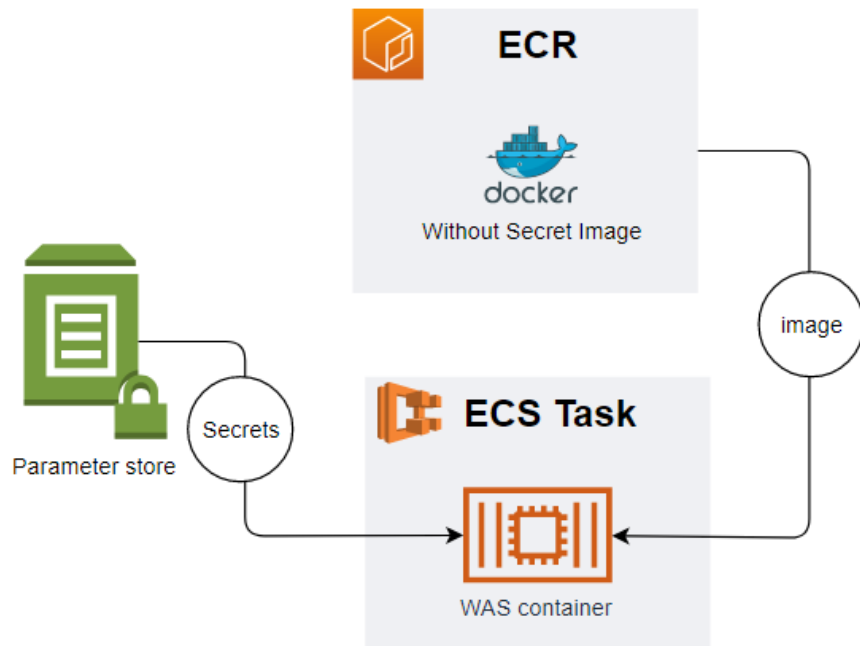


Secret manage - Parameter store

처음 사용했던것은 parameter store.

AWS managed service 이므로 관리 포인트가 없음.

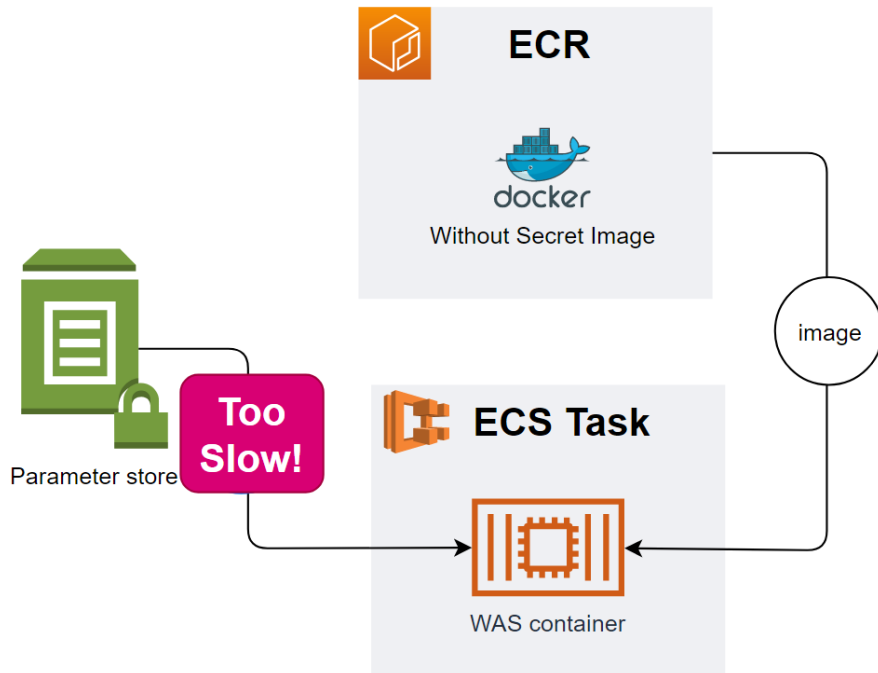
ECS task가 시작할때 parameter store에 연결하여 secret을 가져옴



Secret manage - Parameter store

하지만 단점이 너무 많음.

1. 속도 제한
2. 최대 secret 개수 제한
3. 모든 secret 을 같은 depth로 저장
4. 이미지 시작만 1분이 넘게 걸리게 하는 병목현상 발생



Secret manage - Parameter store

https://docs.aws.amazon.com/general/latest/gr/aws_service_limits.html#limits_ssm

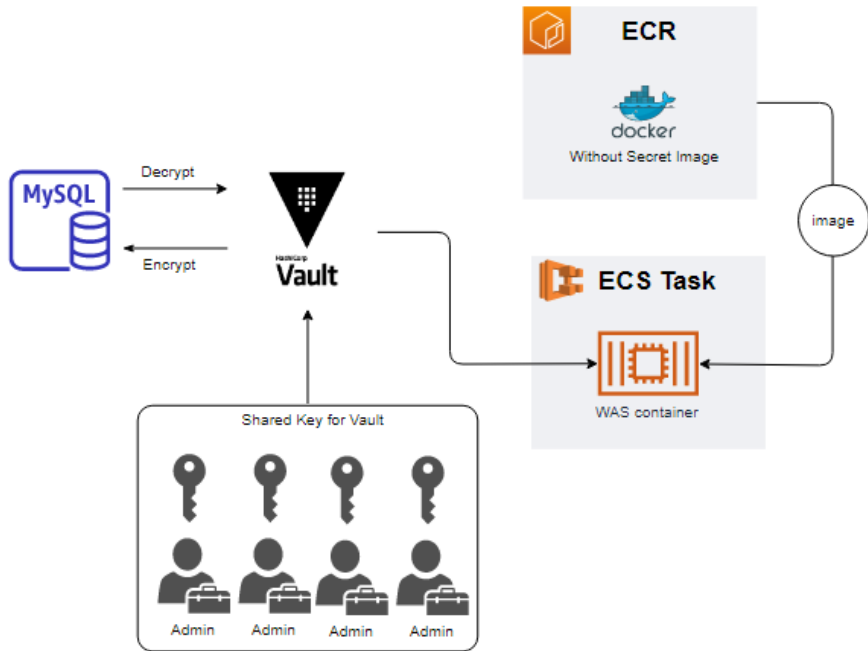
Parameter Store	Total number of parameters allowed (per AWS account and Region)	Standard parameters: 10,000 Advanced parameters: 100,000 For more information about advanced parameters, see About Systems Manager Advanced Parameters in the <i>AWS Systems Manager User Guide</i> .
Parameter Store	Max size for parameter value	Standard parameter: 4 KB Advanced parameter: 8 KB
Parameter Store	Max number of parameter policies per advanced parameter	10
Parameter Store	Max throughput (transactions per second)	Default throughput: 40 (Shared by the following API actions: GetParameter, GetParameters, GetParametersByPath) Higher throughput: 100 (GetParametersByPath) Higher throughput: 1000 (Shared by the following API actions: GetParameter and GetParameters) For more information about Parameter Store throughput, see Increasing Parameter Store Throughput in the <i>AWS Systems Manager User Guide</i> .
Parameter Store	Max history for a parameter	100 past values

Secret manage - Vault

Hashcorp 사의 Vault 를 Secret manag-er 로 사용

Vault에 service, env 별로 secret을 나눠 저장.

docker 에서 Vault python client인 hvc-c를 통해 secret을 불러옴.

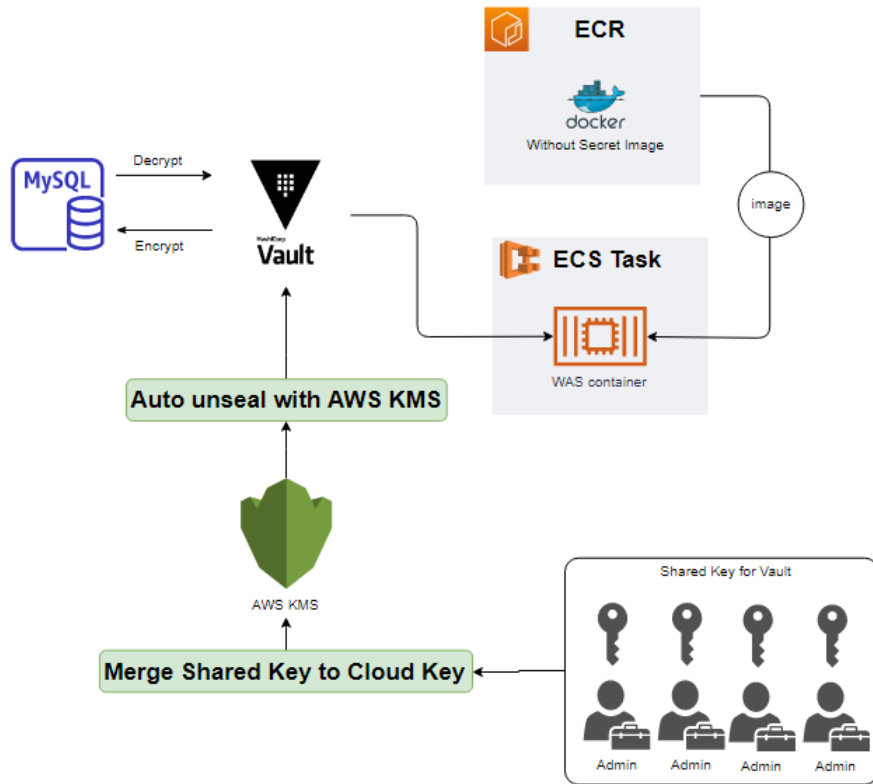


Vault - auto unseal

Vault는 재시동시 seal 상태로, admin N 명이 나눠 가진 key 를 입력하지 않으면 secret을 불러올 수 없음.

Vault service는 수시로 종료될 수 있으므로 auto unseal은 필수.

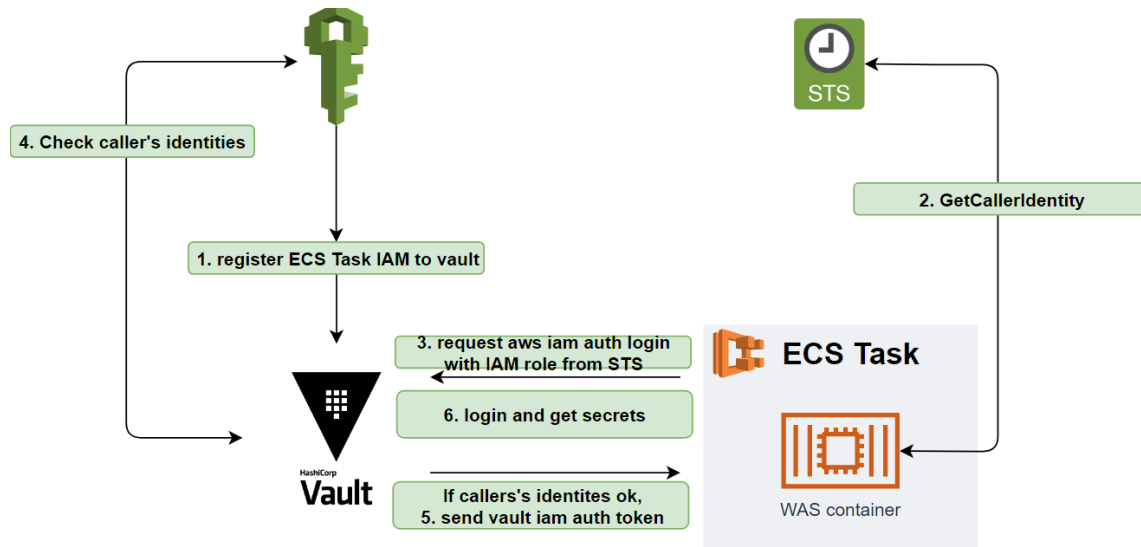
AWS KMS 로 key 를 migration 해두면 vault 재시동시 자동으로 unseal됨.



Vault - AWS IAM Auth

Vault Client 가 Vault에서 secret을 가져오기 위해서는 Token이 필요함.

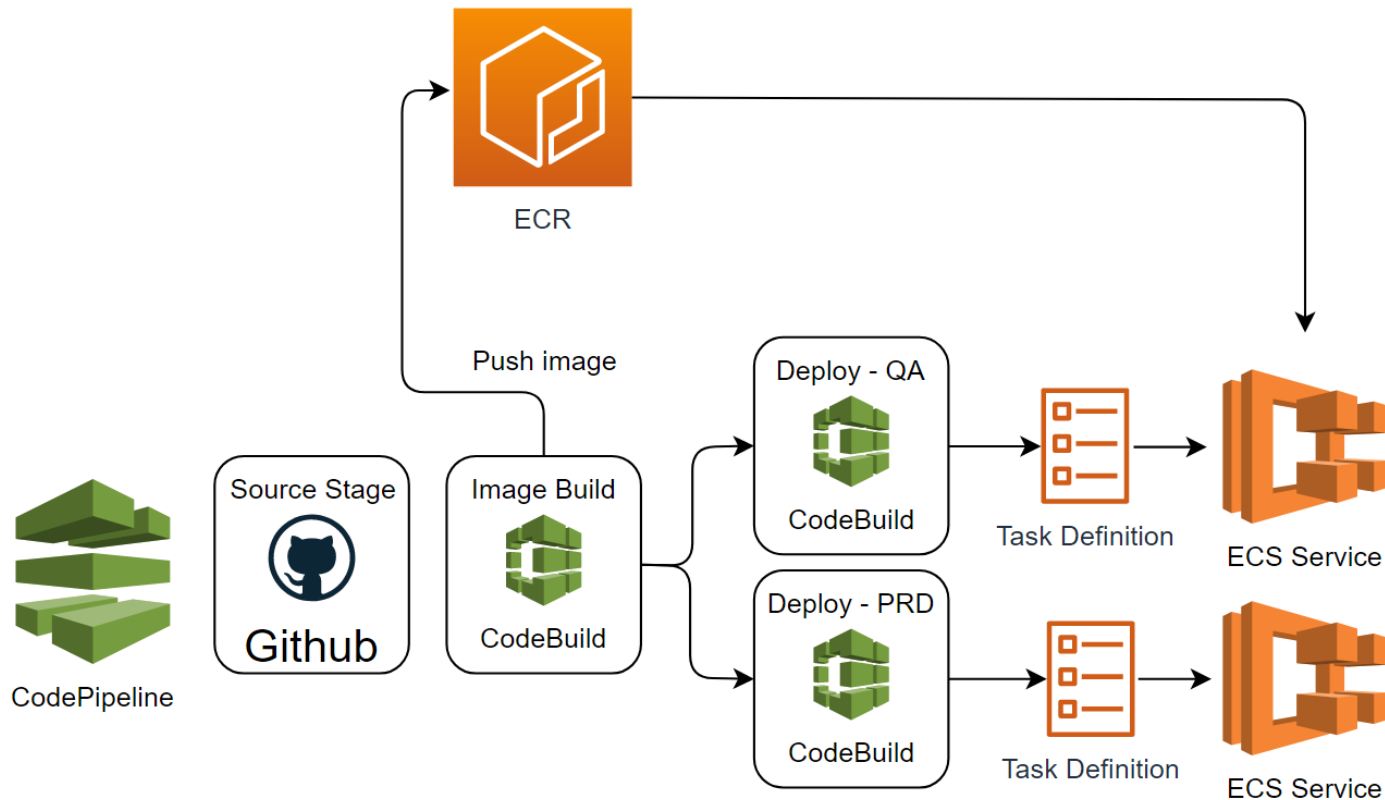
해당 Token을 AWS IAM을 통해 얻을 수 있음.



CD

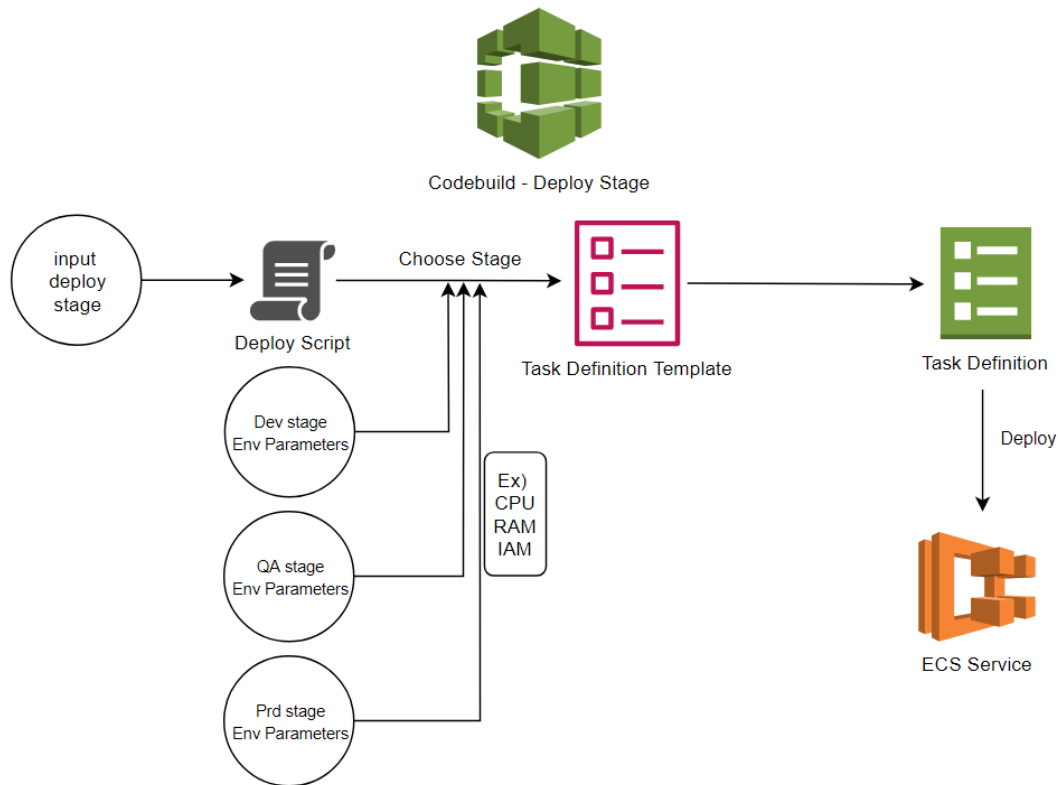
STARTUP TECH CHALLENGE

CD - Code pipeline, Code build



Multi env deploy with task definition template

- Task definition Version control with git.
- Change Resource(CPU, RAM, IAM ..) in source code



Multi env deploy with task definition template

```
elif [ "${inapi_env}" == "qa" ]; then
    TASK_ROLE_NAME=[REDACTED]
    TASK_MEMORY=512
    ECS_CLUSTER=[REDACTED]
    SERVICE_NAME="inapi-qa"
    TASK_FAMILY="inapi-qa"
elif [ "${inapi_env}" == "dev" ]; then
    TASK_ROLE_NAME=[REDACTED]
    TASK_MEMORY=512
    ECS_CLUSTER=[REDACTED]
    SERVICE_NAME="inapi"
    TASK_FAMILY="inapi"
else
    echo "입력받은 inapi env : " "${inapi_env}"
    echo "위의 환경은 지원하지 않는 배포 환경입니다."
fi

TASK_ROLE_ARN="arn:aws:iam:${ECR_ID}:role/${TASK_ROLE_NAME}"

sed -ie "s@%INAPI_ENVIRONMENT%@${inapi_env}@g" "${TASK_DEFINITION_FILE_PATH}"
sed -ie "s@%MEMORY%@${TASK_MEMORY}@g" "${TASK_DEFINITION_FILE_PATH}"
sed -ie "s@%taskRoleArn%@${TASK_ROLE_ARN}@g" "${TASK_DEFINITION_FILE_PATH}"
```

Multi env deploy with task definition template

```
aws ecs register-task-definition --family ${TASK_FAMILY} --cli-input-json file://"${TASK_DEFINITION_FILE_PATH}"

result=$(aws ecs describe-services --cluster ${ECS_CLUSTER} --service ${SERVICE_NAME} | jq '.failures[0].reason')

if [ "${result}" == "\"MISSING\"" ]; then
    echo result
    exit 1
fi

TASK_REVISION=$(aws ecs describe-task-definition --task-definition ${TASK_FAMILY} | jq '.taskDefinition.revision')
DESIRED_COUNT=$(aws ecs describe-services --service ${SERVICE_NAME} --cluster ${ECS_CLUSTER} | jq '.services[0].desiredCount')

aws ecs update-service --cluster ${ECS_CLUSTER} \
    --service ${SERVICE_NAME} \
    --task-definition ${TASK_FAMILY}:${TASK_REVISION} \
    --desired-count ${DESIRED_COUNT} \
    --deployment-configuration maximumPercent=200,minimumHealthyPercent=100
```

Alarm

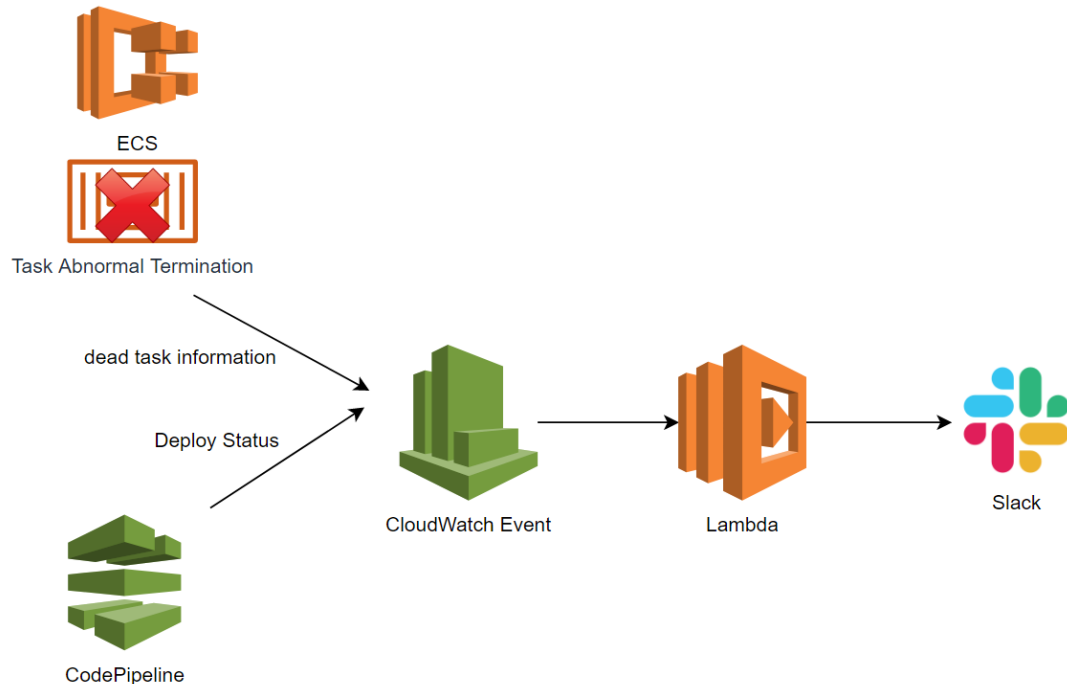
STARTUP TECH CHALLENGE

Alarm service

Task의 비정상 종료나,

배포시 비정상 종료 등의 로그 및 상태를 바로 확인하기위해



Lambda를 활용하여 알람을 제작하여 사용.



ECS Task abnormal termination alarm

Task 비정상 종료일 경우 해당 Task의 정보와 container info를 slack으로 전송.

AWS 에서 바로 log를 확인하기위해 해당 task의 link 바로가기를 제공.

 Task abnormal termination 

Reason: Essential container in task exited

Cluster name
playground-cluster

Group
service:celery


Task id
[REDACTED]

Task definition name
celery-docker:163

Container Info

name	reason
worker_default	OutOfMemoryError: Container killed due to memory usage
worker_segment	OutOfMemoryError: Container killed due to memory usage
worker_kakaopay	OutOfMemoryError: Container killed due to memory usage

Task details

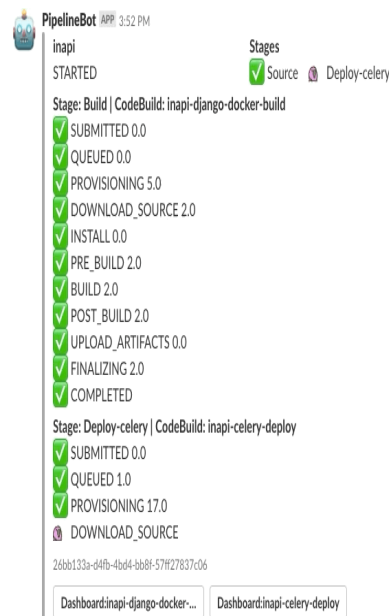
 2

Codepipeline slack integration

CodePipeline 배포 상태 및 비정상 배포를 파악하기 위함.

Cloudwatch Event + Lambda로 구현.

<https://github.com/peoplefund-tech/codepipeline-slack-integration>



Local

STARTUP TECH CHALLENGE

Task definition vs docker compose

Production에서는 Task definition으로 service를 운영하지만, 개발서버나 local에서는 사용할 수 없으므로 docker-compose를 사용해야함.

Production AWS



ECS



Task Definition

Local, development




ECS CLI

AWS 에서 만든 ECS Command line interface.

docker-compose 와 Task definition 간의 변환을 지원함.

아직 버그가 많음.

 [aws](#) / [amazon-ecs-cli](#)

 Watch ▾


89

 Star

1,301


 Fork

221

 Code


 Issues 84

 Pull requests 1

 Projects 1

 Wiki

 Security

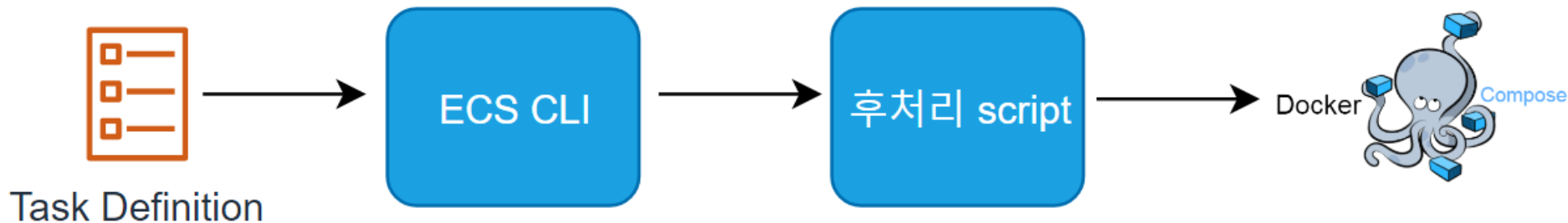
 Insights

<https://github.com/aws/amazon-ecs-cli#running-tasks-locally>

ECS CLI

Task definition을 기반으로 docker-compose를 생성.

local에서 사용하기 위해서 자동 생성된 compose에서 설정을 빼거나 추가해야하는 등의 후처리가 아
직은 필요함.



ECS CLI 후처리

1. task definition 생성
2. ecs-cli 로 docker-compose 생성
3. sed 로 ECS에서만 사용 가능한 옵션 제거
4. docker-compose up

```
docker-compose -f "${DEPLOY_HOME}/ops/dev/docker-compose.yml build
```

```
if [ "$1" != "exclude_celery" ]
```

```
then
```

```
# Restart Celery
```

```
sh ${DEPLOY_HOME}/ops/deploy/celery/celery_deploy.sh dev ${DEPLOY_HOME}
```

```
ecs-cli local create -f ${DEPLOY_HOME}/ops/deploy/celery/celery_zero_task_definition.json.copy.json -o ${DEPLOY_HOME}
```

```
ecs-cli local create -f ${DEPLOY_HOME}/ops/deploy/celery/celery_zero_task_definition.json.copy.json -o ${DEPLOY_HOME}
```

```
sed -ie "s@awslogs-stream-prefix@tag@g" "${DEPLOY_HOME}/ops/dev/docker_compose_celery.yml"
```

```
sed -ie "/AWS_CONTAINER_CREDENTIALS_RELATIVE_URI/d" "${DEPLOY_HOME}/ops/dev/docker_compose_celery.yml"
```

```
sed -ie "s@ECS_CONTAINER_METADATA_URI: http://169.254.170.2/v3@DJANGO_SETTINGS_MODULE: ${DJANGO_SETTINGS_MODULE}@g"
```

```
sed -ie "/networks:/d" "${DEPLOY_HOME}/ops/dev/docker_compose_celery.yml"
```

```
sed -ie "/ecs-local-network:/d" "${DEPLOY_HOME}/ops/dev/docker_compose_celery.yml"
```

```
sed -ie "/external: true/d" "${DEPLOY_HOME}/ops/dev/docker_compose_celery.yml"
```

```
sed -ie "s@awslogs-stream-prefix@tag@g" "${DEPLOY_HOME}/ops/dev/docker_compose_celery_zero.yml"
```

```
sed -ie "/AWS_CONTAINER_CREDENTIALS_RELATIVE_URI/d" "${DEPLOY_HOME}/ops/dev/docker_compose_celery_zero.yml"
```

```
sed -ie "s@ECS_CONTAINER_METADATA_URI: http://169.254.170.2/v3@DJANGO_SETTINGS_MODULE: ${DJANGO_SETTINGS_MODULE}@g"
```

```
sed -ie "/networks:/d" "${DEPLOY_HOME}/ops/dev/docker_compose_celery_zero.yml"
```

```
sed -ie "/ecs-local-network:/d" "${DEPLOY_HOME}/ops/dev/docker_compose_celery_zero.yml"
```

```
sed -ie "/external: true/d" "${DEPLOY_HOME}/ops/dev/docker_compose_celery_zero.yml"
```

```
docker-compose -f ${DEPLOY_HOME}/ops/dev/docker_compose_celery_zero.yml up -d
```

```
docker-compose -f ${DEPLOY_HOME}/ops/dev/docker_compose_celery.yml up -d
```

```
fi
```

```
docker-compose -f "${DEPLOY_HOME}/ops/dev/docker-compose.yml up -d
```


Q&A

STARTUP TECH CHALLENGE

Q. 물론 운영, 관리 차원에서 좋았겠지만 컨테이너로 바꾼 후 실무에서 가장 좋았던 점은 무엇인가요?

A. 배포시 테스트에서 사용했던 환경이 운영에서 깨지는 (내 로컬에서는 잘되는데...) 현상이 거의 없는것이 매우 좋았습니다. 또한 scale out, instance 변경 등 ECS에서 console 몇번 클릭으로 서버 이전 작업을 편하게 할 수 있다는 점도 큰 장점인거 같습니다.

Q. 쿠버네티스를 활용한 무중단 배포까지 구현할 계획이 있나요?

A. 아직은 ECS의 무중단 배포를 사용하고 있습니다. 서비스들이 많아지고 서비스 단위가 점점 작아져서 서비스 관리 자체가 힘들어진다면 쿠버네티스를 고려해볼 생각입니다.

Q. k3s나 k8s 대신에 도커를 선택한 이유가 있으신가요?

A. 회사에서 k8s 대신 ECS를 선택한것은 AWS managed service 의 장점때문이였습니다. EKS 정식 릴리즈가 나오기 전에 판단을 했었고, 당시 EC2에 k8s cluster를 구축하고 관리하는 cost가 현 피플펀드 개발팀 규모에 비해 너무 크다고 판단했습니다.

Q. 비용은 얼마나 차이가나나요?? 늘었나요? 줄었나요?

A. 늘었습니다. 기존에는 EC2에 메모리 제한 없이 하나하나 서비스들을 올렸지만, ECS는 각 서비스에 memory limit을 주어야하기 때문에 이전보다 널널하게 EC2 를 사용하면서 비용이 늘었습니다. 물론 다른 프로세스때문에 OOM으로 무작위 process가 죽는 문제는 없어 서버 운영의 측면에서는 매우 이득이라고 생각합니다.

Thank you 