[Devocean 세미나] 클라우드 비용 최적화 - EKS 비용 최적화

23년 TANGO Public Cloud 전환

Exported on 05/25/2023

Table of Contents

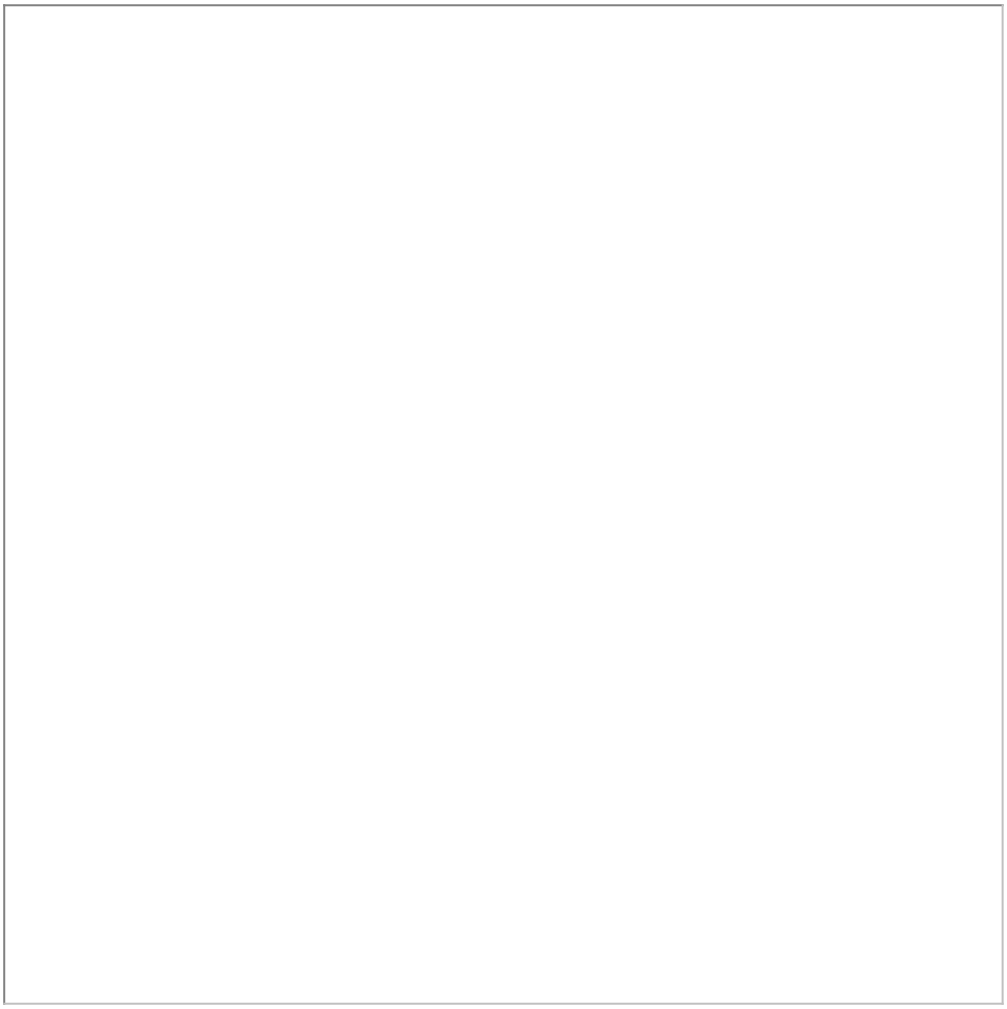
1 Intro. EKS 비용 최적화 (feat. App 변경 없이)

- 낭비 최소화 → 불필요한 리소스 최소화
- 가성비 향상 → 적절한(최신) 기술 및 가격정책 사용

2 1. 불필요한 EKS 리소스 최소화 하기

- **Auto Scaling (Node, Pod)**
- **EKS Auto Stop & Start (kube-downscaler)**
- **Right Sizing (Node, Pod**, Disk**)**
- 미사용 리소스 제거

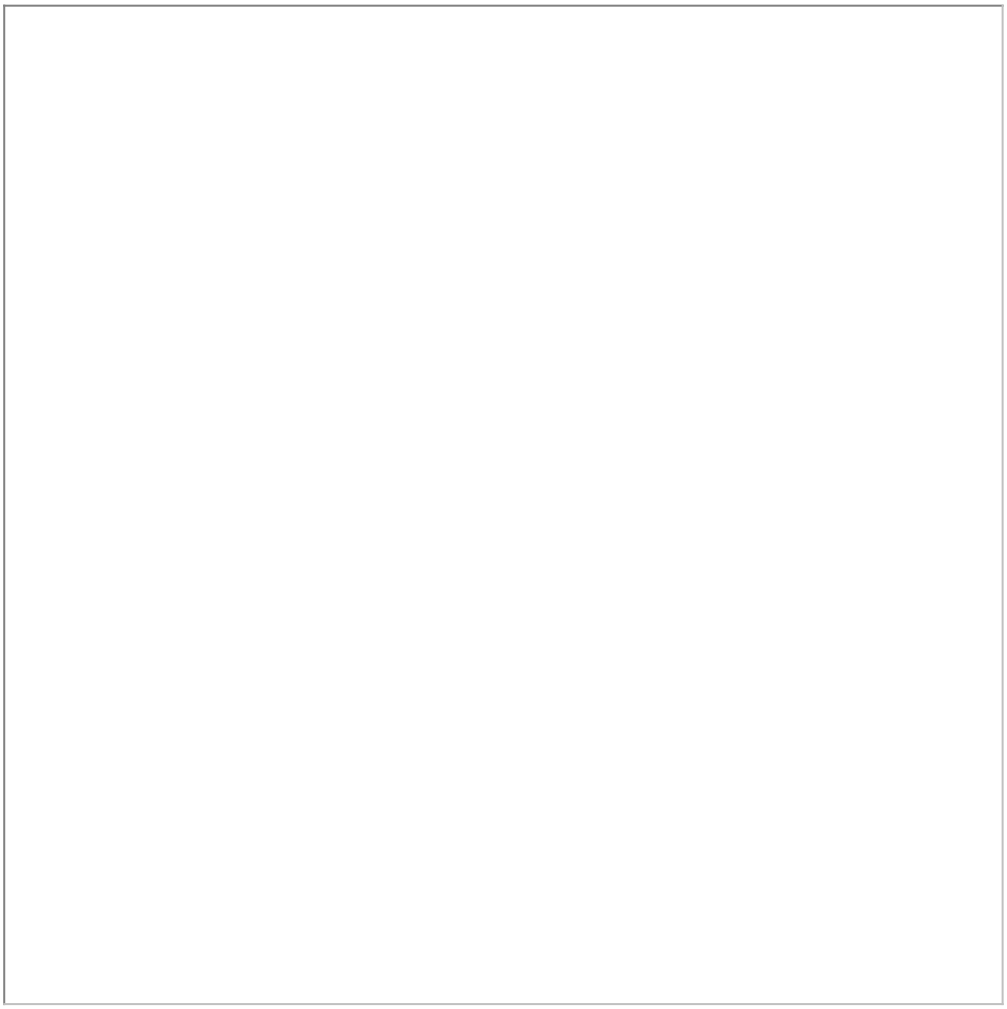
2.1 1-1) Auto Scaling



https://docs.aws.amazon.com/ko_kr/autoscaling/ec2/userguide/auto-scaling-benefits.html

2.1.1 1-1-1) Pod

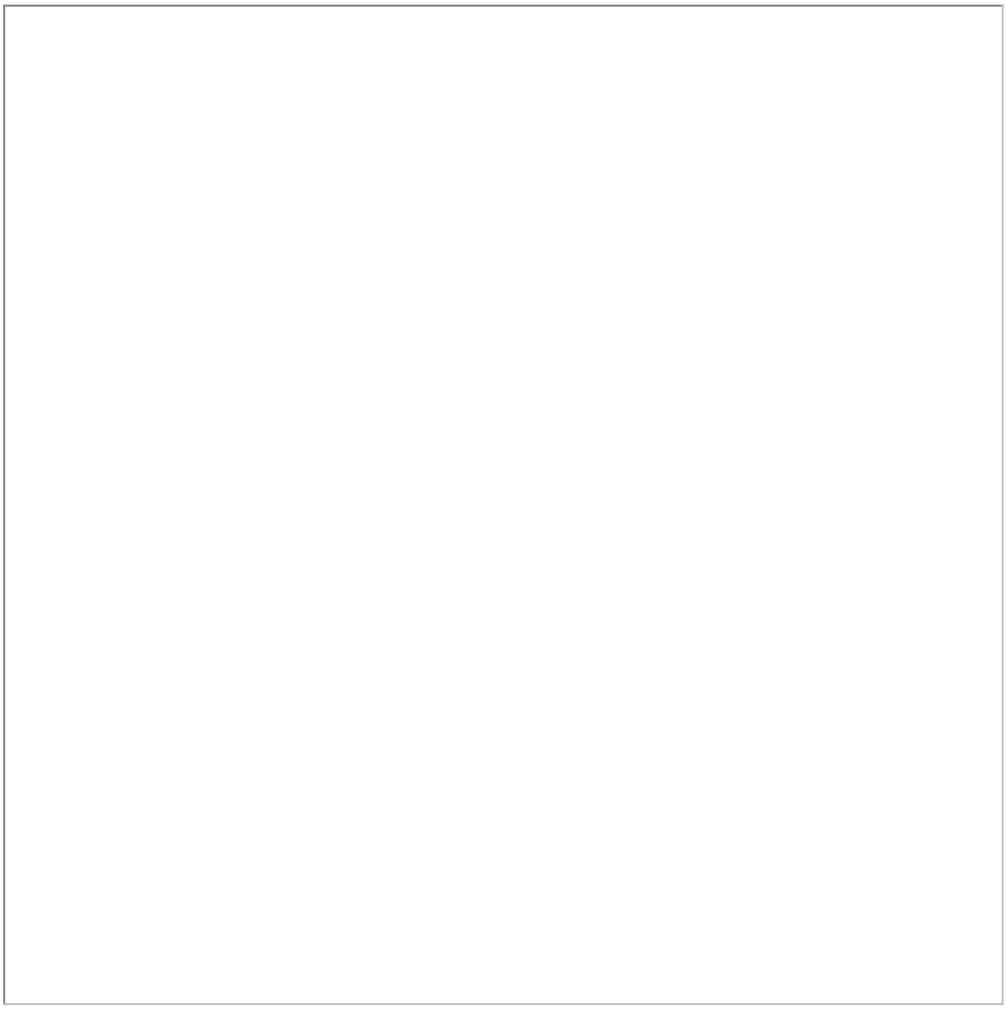
- HPA: Scale-out
- VPA: Scale-up



https://www.densify.com/kubernetes-autoscaling/kubernetes-hpa/

2.1.2 1-1-2) Node

- CA(Cluster Autoscaler)
  - scale-down-utilization-threshold 조정
    - Scale-down 기준, Default값은 0.5
  - 단독 실행 Pod내 Annotation 추가
    - cluster-autoscaler.kubernetes.io/safe-to-evict: true



https://www.kubecost.com/kubernetes-autoscaling/kubernetes-cluster-autoscaler

- Karpenter
    - CA 대비 다양한 기능과 운영 편의성 향상
    - Reference 가 상대적으로 부족

Karpenter is an open-source node provisioning project built for Kubernetes. Karpenter improves the efficiency and cost of running workloads on Kubernetes clusters by:

- **Watching** for pods that the Kubernetes scheduler has marked as unschedulable
- **Evaluating** scheduling constraints (resource requests, nodeselectors, affinities, tolerations, and topology spread constraints) requested by the pods
- **Provisioning** nodes that meet the requirements of the pods
- **Removing** the nodes when the nodes are no longer needed

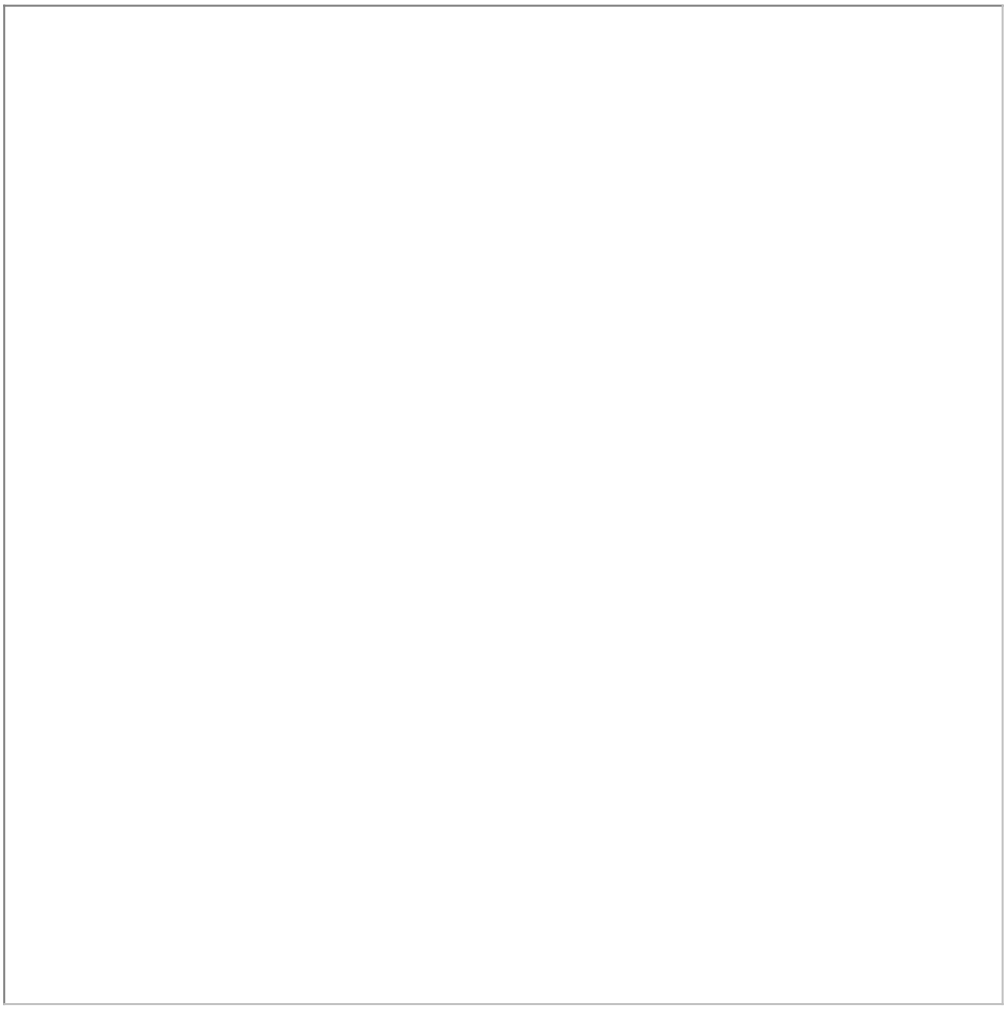
https://github.com/aws/karpenter

2.2 1-2) EKS Auto Stop & Start (kube-downscaler)

- 불필요한 시간대 개발/검증 환경 EKS 리소스 최소화
- 새벽 시간대 (22:00~07:00) EKS Pod 를 0로 만들기 → kube-downscaler

2.2.1 **1-2-1) kube-downscaler 란?**

- 원하는 시간대에 Kubernetes Workload Scale down 및 pause 할 수 있음
- Deployment, Statefulsets, HPA, CronJob 등 지원
- 비용절감은 Node Scale-in 에 의해서 발생하므로, 사전에 Node Auto Scaling 구성이 필수

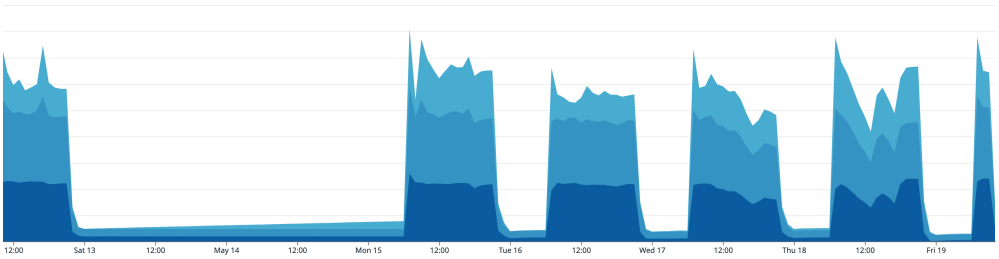


https://codeberg.org/hjacobs/kube-downscaler

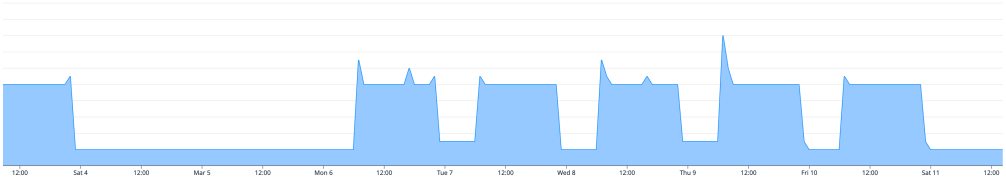
2.2.2 **1-2-2) 적용하기**

적용효과

- Pod 적용결과



- Node 적용결과



- 비용효과 (개발환경 EC2 비용 기준)
  - (5일*15시간) / (7일*24시간) = 35~44% 절감
  - 휴일이 많을수록, 워라벨이 좋을수록 더 많이 절감된다는 사실!!
- 부수적인 효과
  - 일종의 장애 테스트
  - 전체 서비스 다운 → 전체 서비스 재기동

설치하기

- Helm 을 통한 설치
  - https://artifacthub.io/packages/helm/deliveryhero/kube-downscaler
- 주요 Parameter
  - --downtime-replicas=0
    - Down Time 동안의 Pod 개수
    - 1로 설정할 경우 CronJob 은 중단되지 않음
    - PRD 환경의 경우 새벽시간 HPA의 minReplicas만 조정 가능
  - --include-resources=statefulsets,deployments,cronjobs
    - Scale-Down 할 리소스 종류
  - --exclude-namespaces=(서비스 namespace 외 모두)
    - Scale-Down 에서 예외처리할 namespace
    - 정규표현식 사용 가능
  - --default-uptime=Mon-Mon 07:00-22:00 Asia/Seoul,Tue-Tue 07:00-...
    - Default uptime
    - 요일별, 날짜별 설정 가능
  - --exclude-deployments
    - Scale-Down 대상에서 제외해야하는 Workload
    - 데이터 유지가 필요한 In Memory DB 및 기타

실행 로그

- Scale Down

```
[tangostg@ip-101-11-128-53 test]$ kubectl logs -f --tail=100 -n kube-downscaler deployment/kube-downscaler
2023-01-25 13:22:01,125 INFO: Downscaler v22.9.0 started with debug=False, default_downtime=Mon-Fri 19:00-24:00 Asia/Seoul,Mon-Fri 00:00-08:00 Asia/Seoul,Sat-Sun 00:00-24:00 Asia/Seoul, default_uptime=alwa
ys, deployment_time_annotation=None, downscale_period=never, downtime_replicas=0, dry_run=False, enable_events=True, exclude_deployments=kube-downscaler,downscaler, exclude_namespaces=kube-system, grace_pe
riod=900, include_resources=deployments,statefulsets,cronjobs, interval=60, namespace=test, once=False, upscale_period=never
2023-01-25 23:00:17,925 INFO: Scaling up Deployment test/deloy-busybox-hpa from 0 to 2 replicas (uptime: always, downtime: Mon-Fri 19:00-24:00 Asia/Seoul,Mon-Fri 00:00-08:00 Asia/Seoul,Sat-Sun 00:00-24:00
Asia/Seoul)
2023-01-25 23:00:18,032 INFO: Scaling up Deployment test/deloy-busybox-normal from 0 to 3 replicas (uptime: always, downtime: Mon-Fri 19:00-24:00 Asia/Seoul,Mon-Fri 00:00-08:00 Asia/Seoul,Sat-Sun 00:00-24:
00 Asia/Seoul)
2023-01-25 23:00:18,228 INFO: Scaling up StatefulSet test/sts-busybox-normal from 0 to 3 replicas (uptime: always, downtime: Mon-Fri 19:00-24:00 Asia/Seoul,Mon-Fri 00:00-08:00 Asia/Seoul,Sat-Sun 00:00-24:0
0 Asia/Seoul)
2023-01-25 23:00:18,429 INFO: Unsuspending CronJob test/cron-busybox (uptime: always, downtime: Mon-Fri 19:00-24:00 Asia/Seoul,Mon-Fri 00:00-08:00 Asia/Seoul,Sat-Sun 00:00-24:00 Asia/Seoul)
```

- Scale Up

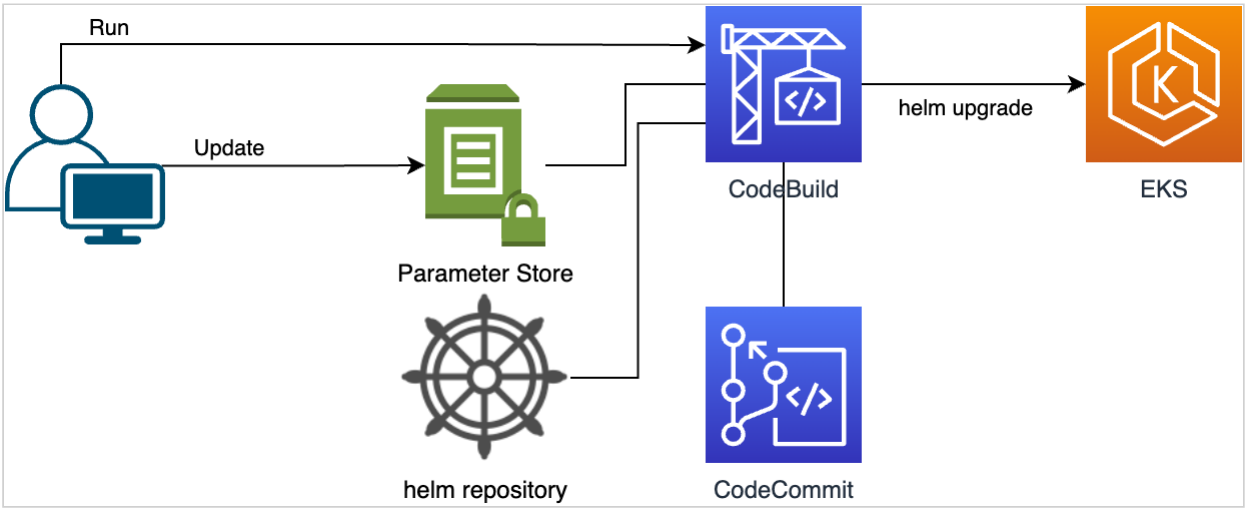
```
2023-03-07 13:23:25,255 INFO: Downscaler v22.9.0 started with debug=False, default_downtime=never, default_uptime=Mon-Mon 07:00-22:00 Asia/Seoul,Tue-Tue 07:00-22:00 Asia/Se
2023-03-07 22:00:47,061 INFO: Scaling up Deployment airflow/tgo-pf-airflow from 0 to 1 replicas (uptime: Mon-Mon 07:00-22:00 Asia/Seoul,Tue-Tue 07:00-22:00 Asia/Seoul,Thu-T
2023-03-07 22:00:47,254 INFO: Scaling up Deployment argocd/argocd-applicationset-controller from 0 to 1 replicas (uptime: Mon-Mon 07:00-22:00 Asia/Seoul,Tue-Tue 07:00-22:00
2023-03-07 22:00:47,369 INFO: Scaling up Deployment argocd/argocd-dex-server from 0 to 1 replicas (uptime: Mon-Mon 07:00-22:00 Asia/Seoul,Tue-Tue 07:00-22:00 Asia/Seoul,Thu
2023-03-07 22:00:47,475 INFO: Scaling up Deployment argocd/argocd-notifications-controller from 0 to 1 replicas (uptime: Mon-Mon 07:00-22:00 Asia/Seoul,Tue-Tue 07:00-22:00
2023-03-07 22:00:47,579 INFO: Scaling up Deployment argocd/argocd-redis-ha-haproxy from 0 to 3 replicas (uptime: Mon-Mon 07:00-22:00 Asia/Seoul,Tue-Tue 07:00-22:00 Asia/Seo
2023-03-07 22:00:47,675 INFO: Scaling up Deployment argocd/argocd-repo-server from 0 to 2 replicas (uptime: Mon-Mon 07:00-22:00 Asia/Seoul,Tue-Tue 07:00-22:00 Asia/Seoul,Th
2023-03-07 22:00:47,873 INFO: Scaling up Deployment argocd/argocd-server from 0 to 2 replicas (uptime: Mon-Mon 07:00-22:00 Asia/Seoul,Tue-Tue 07:00-22:00 Asia/Seoul,Thu-Thu
2023-03-07 22:00:48,056 INFO: Scaling up Deployment ns-tgo-fm/deploy-tgo-fm-afe-com-acccore-fm-srvfault-task from 0 to 2 replicas (uptime: Mon-Mon 07:00-22:00 Asia/Seoul,Tu
2023-03-07 22:00:48,155 INFO: Scaling up Deployment ns-tgo-fm/deploy-tgo-fm-afe-com-acccore-pm-srvgrptca-task from 0 to 2 replicas (uptime: Mon-Mon 07:00-22:00 Asia/Seoul,T
2023-03-07 22:00:48,255 INFO: Scaling up Deployment ns-tgo-fm/deploy-tgo-fm-cfrloader-5gnsa-duh-sse-task from 0 to 2 replicas (uptime: Mon-Mon 07:00-22:00 Asia/Seoul,Tue-Tu
2023-03-07 22:00:48,372 INFO: Scaling up Deployment ns-tgo-fm/deploy-tgo-fm-cfrloader-5gnsa-mme-sse-task from 0 to 1 replicas (uptime: Mon-Mon 07:00-22:00 Asia/Seoul,Tue-Tu
2023-03-07 22:00:48,569 INFO: Scaling up Deployment ns-tgo-fm/deploy-tgo-fm-ebceldr-3g-acc-alarm from 0 to 1 replicas (uptime: Mon-Mon 07:00-22:00 Asia/Seoul,Tue-Tue 07:00-
2023-03-07 22:00:48,670 INFO: Scaling up Deployment ns-tgo-fm/deploy-tgo-fm-ebceldr-3g-bk-alarm from 0 to 1 replicas (uptime: Mon-Mon 07:00-22:00 Asia/Seoul,Tue-Tue 07:00-2
2023-03-07 22:00:48,855 INFO: Scaling up Deployment ns-tgo-fm/deploy-tgo-fm-ebceldr-3g-core-alarm from 0 to 1 replicas (uptime: Mon-Mon 07:00-22:00 Asia/Seoul,Tue-Tue 07:00
```

도입시 고려사항

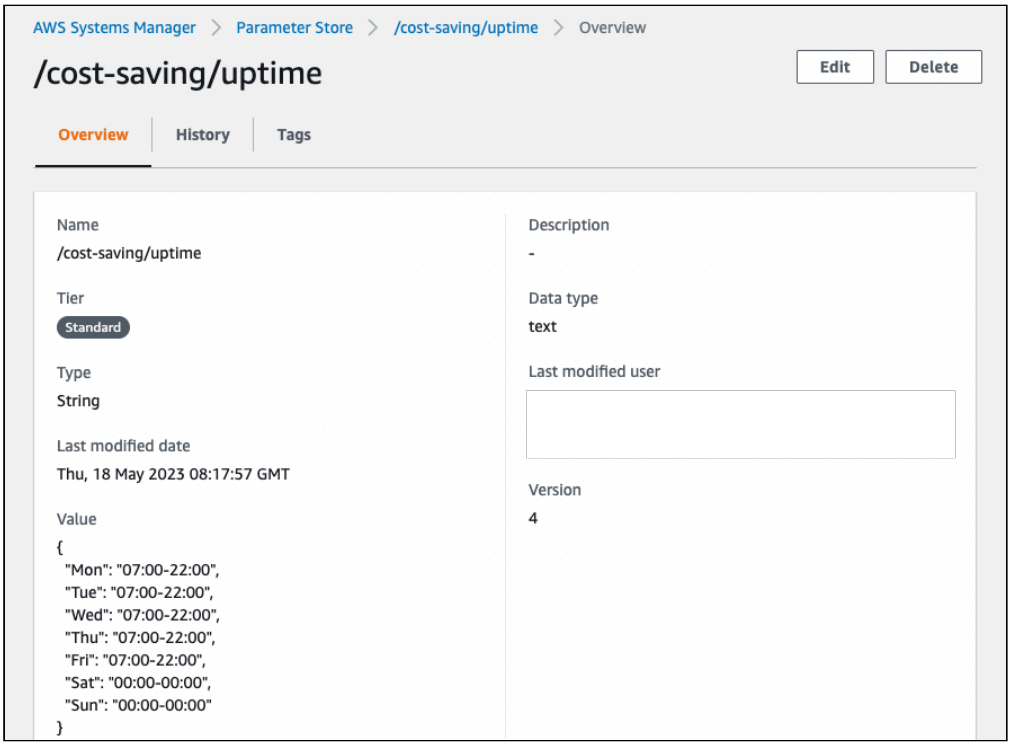
- 예외 Workload 검토
  - kube-system
  - kube-downscaler
  - cluster-autoscaler, metric-server
  - 모니터링 (datadog)
  - 기타
- 예외 Workload의 Affinity 확인
  - 최대의 효과를 위해서는 예외 Workload의 Pod가 1개의 Node에서 동작
  - NodeSelector, Anti-Affinity 등 Node간 분산배치가 필요할 경우 CA 효과 제한
- Cluster Autoscaler 정상동작 확인
  - 단독실행 Pod의 safe-to-evict 설정
  - 기타 CA Log 확인
- Application 복구동작 확인
  - App별 복구를 위한 추가 동작 구성
  - 예외처리 (데이터 유지가 필요한 In Memory DB 등)
- ArgoCD
  - HA 구성 필요 (데이터 etcd 내 저장)
- Job Workload Stop 불가
- (2023-05-19 기준) K8S 1.25 이상 버전 사용시 CronJob suspend 동작 불가

2.2.3 **1-2-3)** 지속 가능하게 kube-downscaler 사용하기

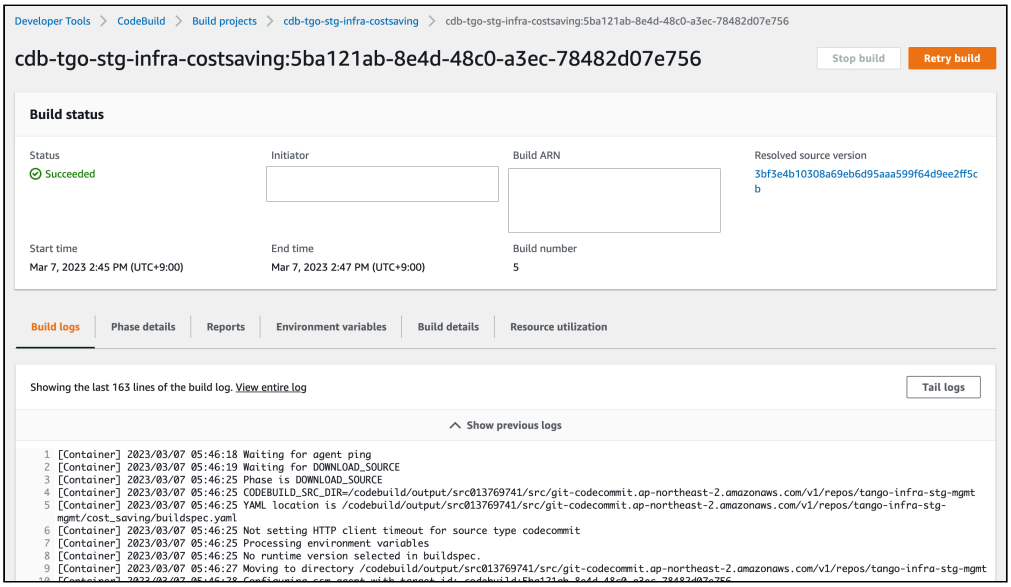
- 야간 작업/배포/테스트시 uptime 변경 필요
- 공휴일 정보 업데이트 필요
- Parameter Store 및 CodeBuild 를 활용한 사용자 편의 배포 구성



- Parameter Store



- CodeBuild 실행



2.3 1-3) Right Sizing

2.3.1 1-3-1) Pod

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: app
    image: images.my-company.example/app:v4
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

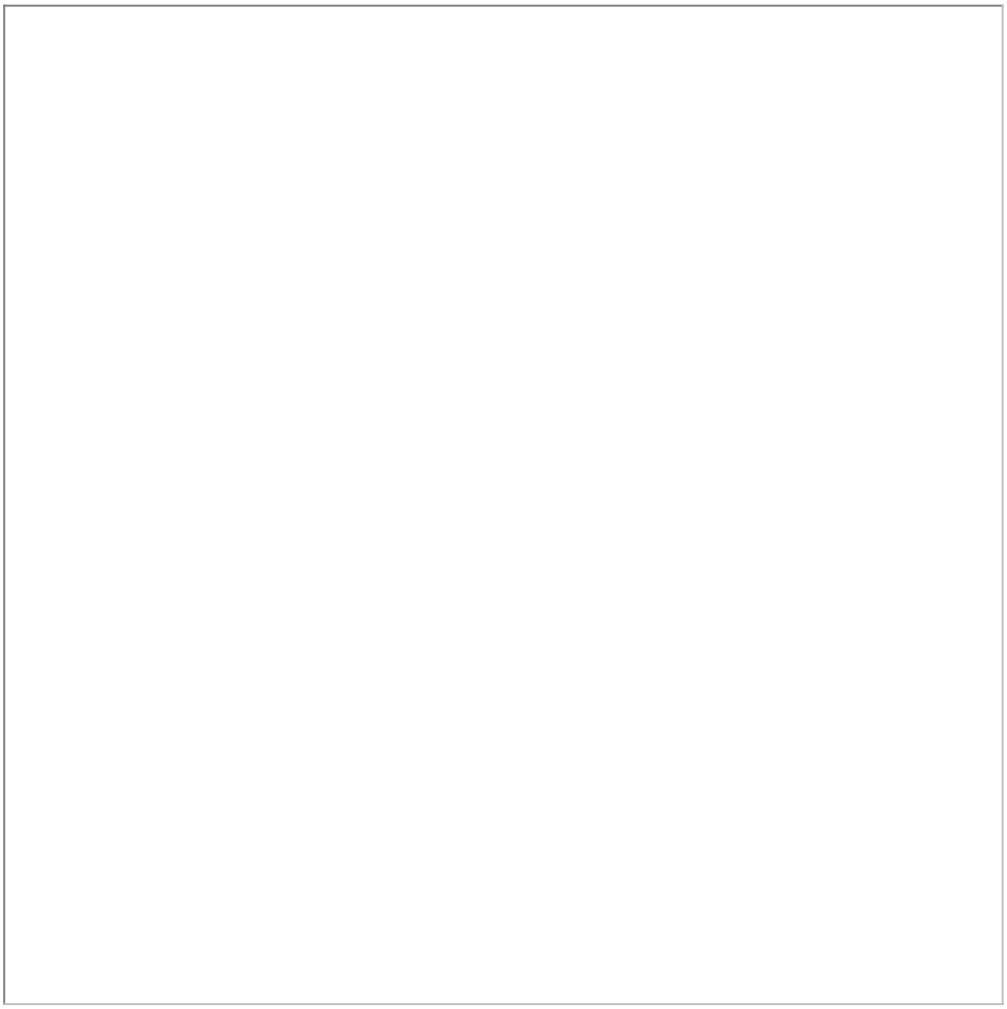
- requests: 컨테이너가 예약하는 최소 리소스. 파드가 배치될 노드를 결정
- limits: 컨테이너가 최대 사용 가능한 리소스

> **파드 및 컨테이너 리소스 관리**
>
> 파드를 지정할 때, 컨테이너에 필요한 각 리소스의 양을 선택적으로 지정할 수 있다. 지정할 가장 일반적인 리소스는 CPU와 메모리(RAM) 그리고 다른 것들이 있다.
>
> 파드에서 컨테이너에 대한 리소스 *요청(request)* 을 지정하면, kube-scheduler는 이 정보를 사용하여 파드가 배치될 노드를 결정한다. 컨테이너에 대한 리소스 *제한(limit)* 을 지정하면, kubelet은 실행 중인 컨테이너가 설정한 제한보다 많은 리소스를 사용할 수 없도록 해당 제한을 적용한다. 또한 kubelet은 컨테이너가 사용할 수 있도록 해당 시스템 리소스의 최소 *요청* 량을 예약한다.

Requests=(비용)

- 너무 클 경우에는 불필요한 Node로 인한 비용 낭비 발생
- 너무 적을 경우 빈번한 throttling(CPU) 및 OOMKilled(Memory) 발생
- Recommenader
    - VPA Recommender
    - Kubecost Request Sizing Recommendations

Kubecost Request Sizing Recommendations



https://docs.kubecost.com/architecture/architecture

사전정의 Profile

- Window: Duration of time over which to calculate usage
- algorithm: Calculate recommendations based on historical usage data
- q: The desired quantile to base recommendations
- targetUtilization: A ratio of headroom on the base recommended request

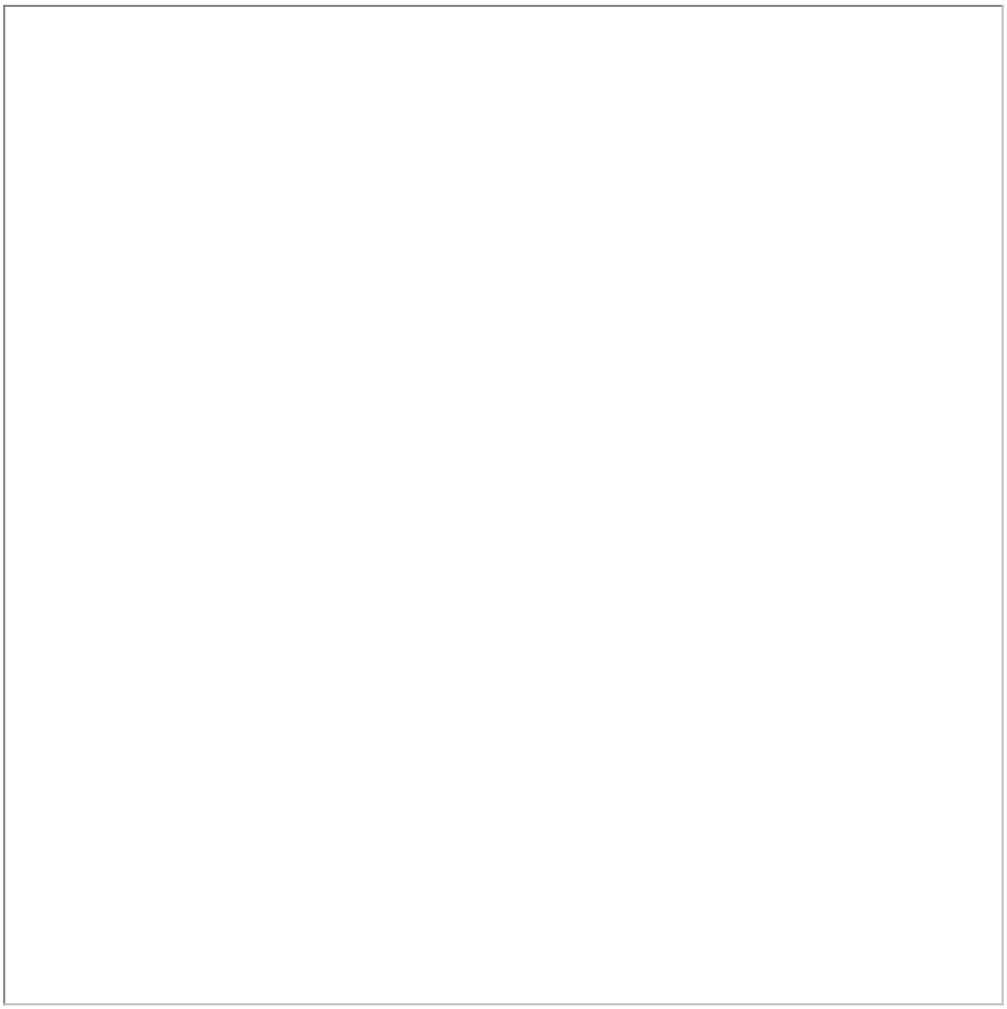
Profile	**Window**	**algorithm (CPU|RAM)**	**q(CPU|RAM)**	**target (CPU|RAM) Utilization**
Development	3d	Max	-	80%
Production	3d	Max	-	65%

Profile	Window	algorithm (CPU|RAM)	q(CPU|RAM)	target (CPU|RAM) Utilization
High Availability	3d	Max	-	50%

계산해보자

- Production 환경이고, 3일간의 CPU 사용량의 최대값이 650m 이라면,
- requests.cpu = 650 * (100/65) = 100m

VPA Recommender



https://www.kubecost.com/kubernetes-autoscaling/kubernetes-vpa/

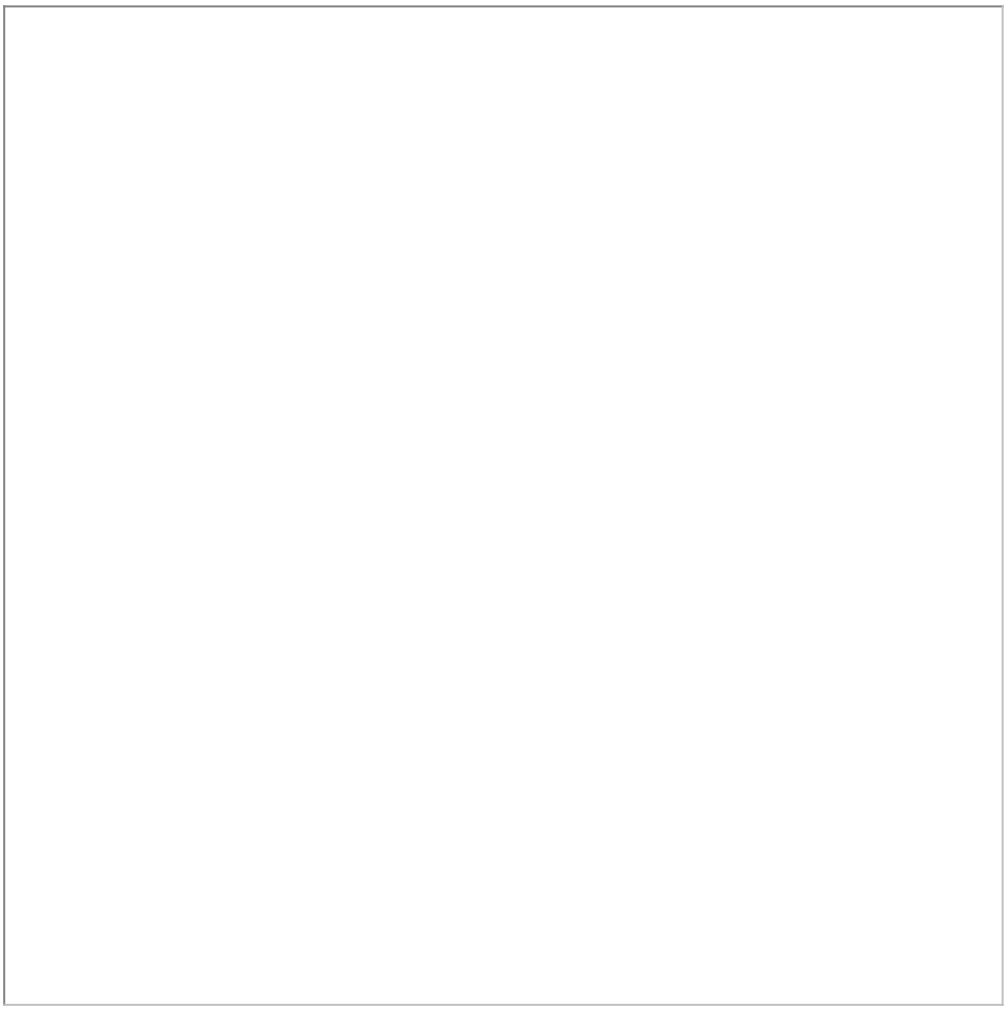
사전정의 Profile

- target-percentile: calculate their recommendations.
- recommendation-margin-fraction: Fraction of usage added as the safety margin
- histogram-decay-half-life: The amount of time it takes a historical usage sample to lose half of its weight.

Profile	Window	target-(cpu|memory)-percentile	recommendation-margin-fraction	(cpu|memory)-histogram-decay-half-life
frual	8d	50%	15%	24h
standard	8d	90%	15%	24h
performance	8d	95%	15%	24h

2.3.2 1-3-2) Node

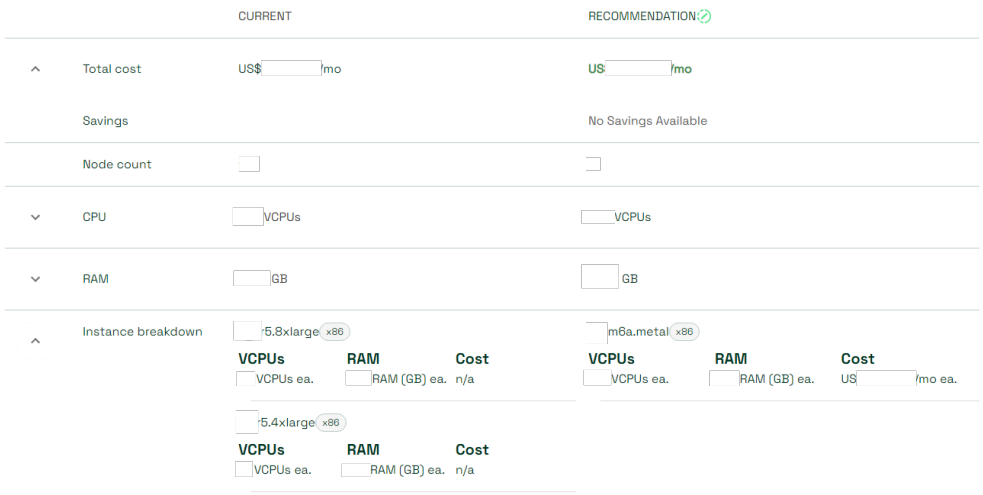
- Nodegroup 별 특성을 파악하고 적절한 Instance Type 선택하기



https://www.nakivo.com/blog/the-definitive-guide-to-aws-ec2-instance-types/

- Instance Family: Nodegroup내 Pod의 CPU / Memory 비율 고려
  - c → 1:2
  - m, t → 1:4
  - r → 1:8
- Instance generation: 가급적 최신버전
- Additional capability: 가급적 graviton
- Instance size: Nodegroup 상황에 맞춰 적절히 선택

kubecost - Cluster Sizing Recommendations



추가 고려사항

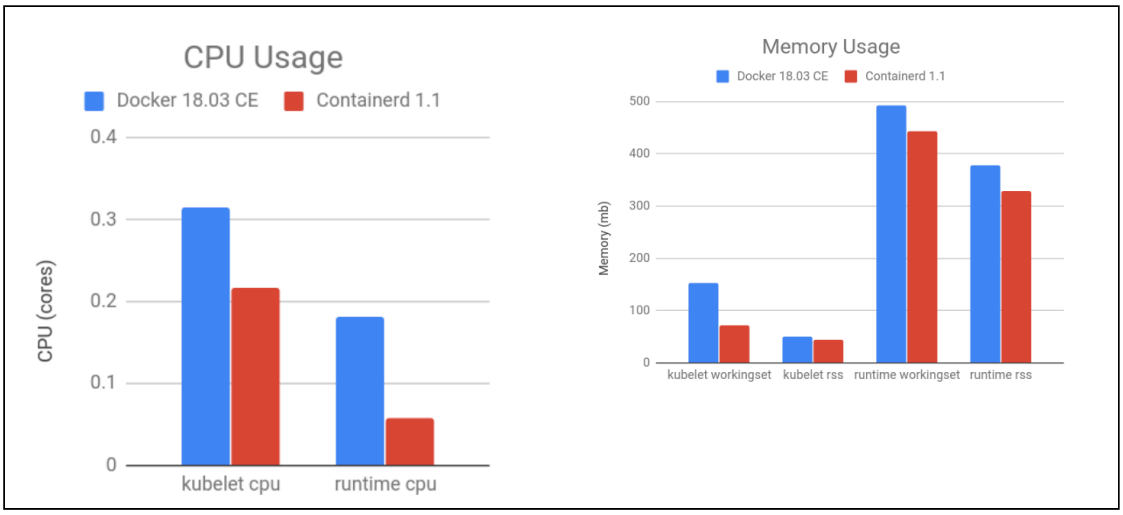
- 회사내 RI 또는 Saving Plan 정책 확인 필요
- 가격정책 적용이 가능한 Instance Family 및 Instance Size 확인
- Karpenter 적용 검토

3 2. EKS 가성비 향상시키기

3.1 2-1) 지속적으로 업데이트 하기

docker → containerd

- container runtime 변경
- EKS 1.24 이상버전은 docker 미지원
- 최대 60%. (실제 적용시 약 12% 정도의 효과)
- docker 소켓사용 등 전환시 영향도 검토



최신 Instance Family

- 최대 15%

> Amazon R6i 인스턴스는 3세대 인텔 제온 스케일러블 프로세서(코드명 아이스 레이크)로 구동되며 메모리 집약적 워크로드에 적합합니다.
>
> **기능:**
>
> - 최대 3.5GHz의 3세대 인텔 제온 스케일러블 프로세서(아이스 레이크 8375C)
> - 최대 15% 향상된 컴퓨팅 가격 대비 성능(R5 인스턴스와 비교)
> - 최대 20% 더 뛰어난 vCPU당 메모리 대역폭(R5 인스턴스와 비교)

Gravition

- 최대 40%
- arm Architecture

> Amazon EC2 R6g 인스턴스는 Arm 기반 AWS Graviton2 프로세서로 구동됩니다. 기존 세대 R5 인스턴스에 비해 메모리 집약적인 애플리케이션에 최대 40% 향상된 가격 대비 성능을 제공합니다.

3.2 2-2) 가격정책 활용하기

RI(Reserved Instance) 과 Saving Plan

- EC2 약정할인
- 세부 옵션에 따라서 31~72% 까지 절감
- 가격 vs 유연성

특성	표준	컨버터블
약정 기간(온디맨드 대비 평균 할인율)	1년(40%), 3년(60%)	1년(31%), 3년(54%)
가용 영역, 인스턴스 크기(Linux OS의 경우) 및 네트워킹 유형 변경	예(ModifyReservedInstances API와 콘솔 사용)	예(ExchangeReservedInstances API와 콘솔 사용)
인스턴스 패밀리, 운영 체제, 테넌시 및 결제 옵션 변경		예
가격 인하 혜택		예

- **컴퓨팅 Savings Plans**는 유연성이 가장 뛰어나며 비용을 최대 66%까지 절감할 수 있습니다(컨버터블 RI와 동일). 이 플랜은 EMR, ECS 또는 EKS 클러스터에 속하는 EC2 인스턴스 또는 Fargate에서 시작한 EC2 인스턴스를 포함하여 리전, 인스턴스 패밀리, 운영 체제 또는 테넌시에 관계없이 모든 EC2 인스턴스에 자동으로 적용됩니다. 예를 들어 아무것도 하지 않아도 C4에서 C5 인스턴스로 이동하고, 워크로드를 더블린에서 런던으로 이동하거나, EC2에서 Fargate로 마이그레이션할 때 Savings Plan 요금 혜택을 받을 수 있습니다.
- **EC2 인스턴스 Savings Plans**는 리전 내의 특정 인스턴스 패밀리에 적용되며 가장 큰 할인 혜택(최대 72%, 표준 RI와 동일)을 제공합니다. RI와 마찬가지로 Savings Plan은 한 리전 전체에서 동일 인스턴스 유형의 여러 다른 크기(예: **c5.4xlarge** 또는 **c5.large**)의 사용에 적용됩니다. Windows에서 Linux로 전환할 때 Savings Plan을 변경하지 않아도 할인 혜택을 계속 받을 수 있습니다.

3.3 2-3) 가성비 좋은 Instance Type 사용하기

Spot Instance

- 가격 vs 안정성
- 개발환경내 도입 검토

Amazon EC2 스팟 인스턴스를 사용하면 AWS 클라우드에서 미사용 EC2 용량을 활용할 수 있습니다. 스팟 인스턴스는 온디맨드 요금과 비교하여 최대 90% 할인된 금액으로 제공됩니다. 빅 데이터, 컨테이너식 워크로드, CI/CD, 웹 서버, 고성능 컴퓨팅(HPC), 테스트 및 개발 워크로드 등 다양한 상태 비저장, 내결함성 또는 유연한 애플리케이션에 스팟 인스턴스를 사용할 수 있습니다. 스팟 인스턴스는 Auto Scaling, EMR, ECS, CloudFormation, Data Pipeline 및 AWS Batch와 같은 AWS 서비스와 긴밀하게 통합되므로, 스팟 인스턴스에서 실행되는 애플리케이션을 시작하고 유지 관리하는 방법을 선택할 수 있습니다.

Burst (t3, t4, t4g) Instance Type

- 가격 vs 성능
- 개발환경내 도입 검토

4 3. 지속 가능한 EKS 비용 최적화 환경 구성하기 (kubecost)

- Application 기능 변경
- H/W(Instance Type, Disk 등) 변경
- 오픈소스, 클러스터 업데이트
- 데이터 변경
- 기타

4.1 3-1) 측정하기

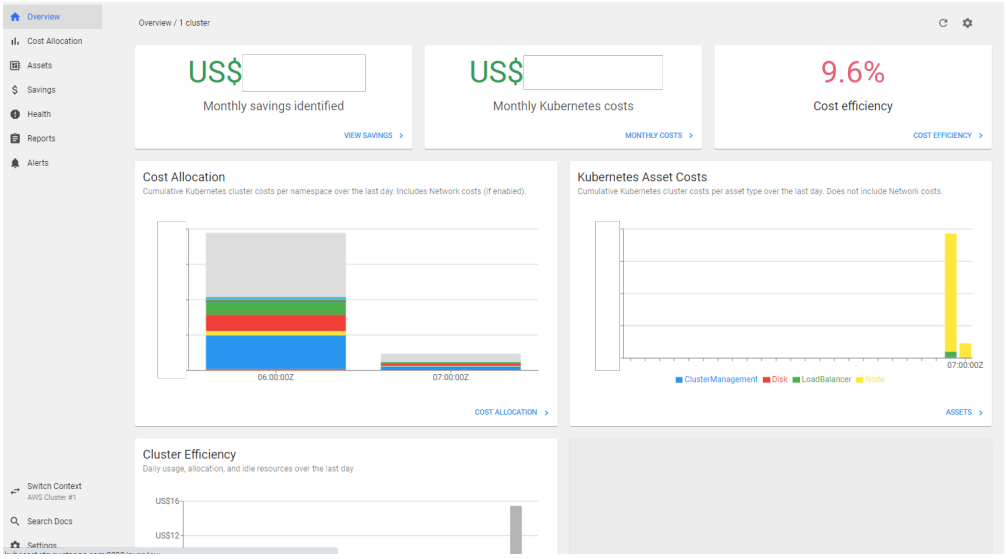
- Request 대비 Pod의 CPU, Memory가 얼마나 사용되고 있는가?
- Node의 예약되지 않은 CPU, Memory 가 얼마나 있는가?

kubecost Efficiency

*(((CPU Usage / CPU Requested) * CPU Cost) + ((RAM Usage / RAM Requested) * RAM Cost)) / (RAM Cost + CPU Cost)*

where
CPU Usage = rate(container_cpu_usage_seconds_total) over the time window
RAM Usage = avg(container_memory_working_set_bytes) over the time window

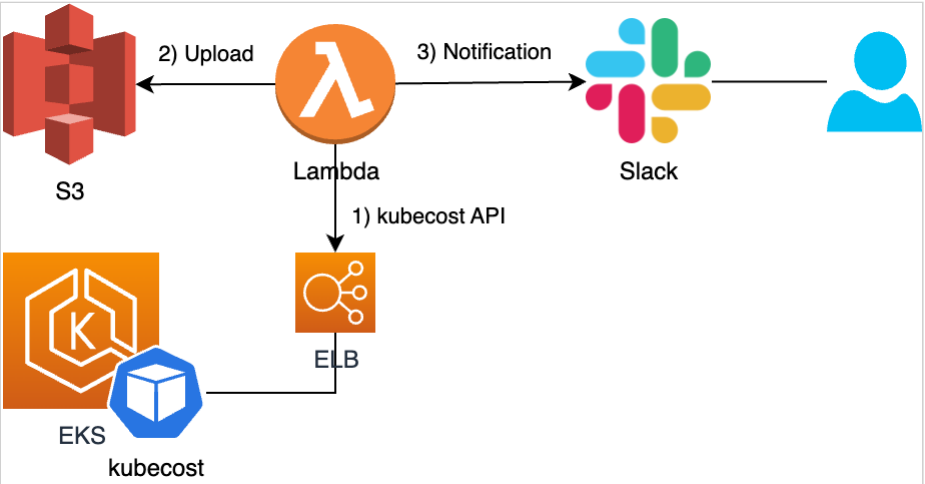
kubecost Dashbaord



4.2 3-2) 감시하기

- Lambda, S3, kubecost API 를 통한 데이터 수집/분석
- Slack 을 통한 주기적 Notification

Architecture



Slack Notification

- Scale Up, Scale Down

[prd, 20230515, Top20] CPU - Scale DOWN
- S3: s3://[redacted]kubecost/20230515/5_kubecost_container_scaling_rank.csv

Index	Controller Kind	Namespace	Controller Name	Container Name	Efficie (ratio)	Cur Reqst (m)	Rec Reqst (m)	Rec Limit (m)	Rec HpaCpu (%)
269.3	deployment	ns-	[redacted]	pa[redacted]	0.162	2679	953	1270	106
105.2	deployment	ns-	[redacted]	pa[redacted]	nan	720	491	655	106
88.5	statefulset	ns-	[redacted]	ap[redacted]	0.010	600	50	50	80
44.8	deployment	ns-	[redacted]	de[redacted]	0.011	700	50	50	80
43.3	deployment	ns-	[redacted]	ap[redacted]	0.098	910	532	709	106
41.6	deployment	ns-	[redacted]	pa[redacted]	0.014	2236	439	586	106
40.2	deployment	ns-	[redacted]	de[redacted]	z 0.020	400	50	50	80
40.0	statefulset	ns-	[redacted]	ad[redacted]	0.063	133	66	88	106
39.6	deployment	ns-	[redacted]	ap[redacted]	0.123	450	105	140	106
34.5	deployment	ns-	[redacted]	de[redacted]	0.008	800	50	50	80
31.0	deployment	ns-	[redacted]	fi[redacted]	nan	500	50	50	80
31.0	deployment	ns-	[redacted]	fi[redacted]	nan	500	50	50	80
29.9	deployment	ns-	[redacted]	ap[redacted]	0.012	700	50	50	80
29.9	deployment	ns-	[redacted]	ap[redacted]	0.009	700	50	50	80
29.9	deployment	ns-	[redacted]	ap[redacted]	0.009	700	50	50	80
29.8	deployment	ns-	[redacted]	de[redacted]	0.009	700	50	50	80
29.8	deployment	ns-	[redacted]	de[redacted]	0.018	700	50	50	80
29.8	deployment	ns-	[redacted]	ap[redacted]	0.005	700	50	50	80
29.3	deployment	ns-	[redacted]	ap[redacted]	0.224	428	172	229	106
23.7	deployment	ns-	[redacted]	aul[redacted]	0.135	300	94	125	106

[prd, 20230515, Top20] CPU - Scale UP
- S3: s3://[redacted]kubecost/20230515/5_kubecost_container_scaling_rank.csv

Index	Controller Kind	Namespace	Controller Name	Container Name	Efficie (ratio)	Cur Reqst (m)	Rec Reqst (m)	Rec Limit (m)	Rec HpaCpu (%)
-172.1	statefulset	ns-	[redacted]	ad[redacted]	1.101	171	396	528	106
-21.8	deployment	ns-	[redacted]	pa[redacted]	0.442	460	650	867	106
-11.5	statefulset	ns-	[redacted]	ad[redacted]	0.468	202	329	439	106
-5.7	statefulset	ns-	[redacted]	ad[redacted]	0.103	50	63	84	106
-3.8	deployment	ns-	[redacted]	ta[redacted]	0.577	140	223	297	106
-3.3	deployment	ns-	[redacted]	ap[redacted]	0.638	50	86	115	106