제목: 쿠버네티스 파드 클러스터링 방법과 활용 예

안녕하세요 넷마블 QA실 QA시스템팀 이동근입니다.

쿠버네티스

쿠버네티스는 대표적인 무상태(Stateless) 아키텍처를 갖는 시스템 입니다. 신규 생성된 파드와 그 전 파드 간 영향을 주지 않는 기본 구조를 가지고 있습니다. 그렇기 때문에 스케일아웃을 통한 급증하는 트래픽 대응에 큰 장점을 가지고 있습니다. 그로인해 Micro Service Architechure (MSA)로 구성된 시스템에서는 적극적으로 적용하고 있습니다. 제가 담당하는 크래시리포트 또한 쿠버네티스를 사용하고 있으며 트래픽에 기반한 Horizontal Pod Autoscaling(HPA) 을 사용하고 있습니다. 하지만, 이런 특징으로 파드들 중에 일부 파드에서만 기능을 활성화 시키거나, 주변 파드에서 생성된 데이터를 공유해야 하는 기능이 필요하다면 추가 설정이 필요합니다.

서비스 디스커버리*

최근 서비스 아키텍처가 모놀리식(Monolithic) 에서 MSA로 변화하는 과정에서 부하 분산을 위해 신규 인스턴스가 추가될 때 이를 전체 서비스 목록에 자동으로 추가하고 트래픽을 전달하기 위한 자동화 절차가 필요했습니다.

서비스 디스커버리 구현 패턴으로 클라이언트 기반 패턴과 서버 기반 패턴이 존재 합니다. 클라이언트 기반 패턴의 대표 Open Source Software(OSS) 는 Spring Cloud Netflix Eureka** 가 있습니다. 서버 기반 패턴은 쿠버네티스가 대표적입니다.

*https://catsbi.oopy.io/c95e714b-0d28-49d0-aff5-10acd5d0dce5

**https://spring.io/projects/spring-cloud-netflix

본 블로그에서는 쿠버네티스의 서비스에서 제공하는 서비스 디스커버리 절차를 사용한 파드 클러스터링을 구성하여 실시간 스트리밍 서비스 구성과 캐시 데이터 공유를 통한 메시지 처리 성능 개선을 해보려 합니다. 이를 적용한 두 개의 어플리케이션에 대해서 좀 더 자세히 알아보겠습니다.

크래시리포트 실시간 5분 통계 기능

첫번째로 크래시리포트는 최근 5분간 발생한 데이터를 스트리밍 데이터처리 기능을 활용해서 실시간으로 대시보드에 푸시로 사용자에게 제공하고 있습니다.

	게임코드	OS	빌드 코드	게임명	앱로드수 ↓	크래시수	크래시율(%)
★	skiagb		A	세븐나이츠 키우기 Global (Android)	5492	16	0.29
★	skiagb		A	세븐나이츠 키우기 Global (iOS)	2606	6	0.23
★	tog		A	신의 탑: 새로운 세계 - Android	2228	6	0.27
★	tog		A	신의 탑: 새로운 세계 - iOS	1792	2	0.11

Figure 1 서비스 예시

기존 서비스 구성의 한계

기존 구성은 실시간 데이터 생성을 위해 Google Cloud Platform(GCP) 의 Dataflow 제품을 사용합니다. Dataflow* 는 서버리스 제품으로 Redis 인스턴스에 연결 및 푸시 하는 절차가 복잡합니다. 또한 대시보드에서 Websocket 구독을 연결하기에도 적합하지 않습니다. 그렇다 보니, Redis 에 푸시 하는 기능과 Redis 를 구독하고 Websocket 서버를 구성하기 위해서 별도의 쿠버네티스 클러스터를 추가 구성해야 했습니다. Apache Beam 기반인 Dataflow 신규 데이터 항목 추가시 학습량이 높으며 Apache Beam 의 빈번한 버전 업데이트와 GCP 의 지원 만료** 등으로 유지보수의 어려움이 있습니다. Dataflow 의 기능을 대체 할 수 있는 적절한 라이브러리를 이용해서 분산된 인스턴스들을 단일 쿠버네티스 기반 실시간 푸시 서비스로 재 구성하려 합니다.

*https://cloud.google.com/dataflow?hl=ko

**https://cloud.google.com/dataflow/docs/support/sdk-version-support-status?hl=ko

실시간 대시보드 푸시 기능을 위해서 Google Cloud Platform에 다음과 같이 신규 구성하였습니다.

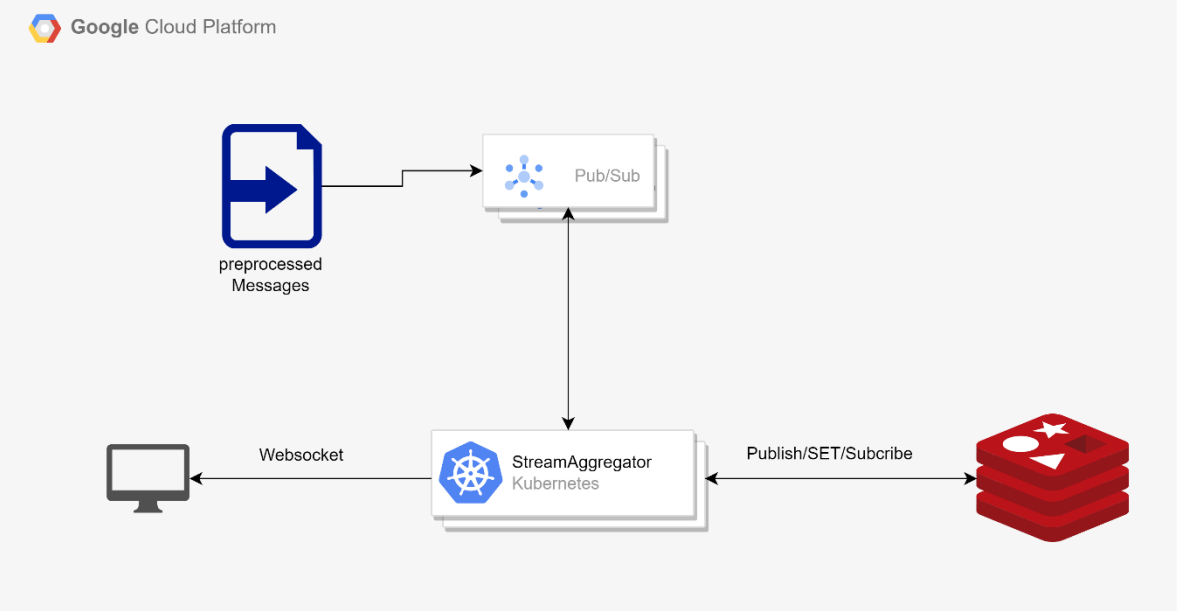

Figure 2 서비스 구성도

Figure2 의 쿠버네티스는 다수의 파드가 Pub/Sub 에서 수신하는 메시지를 일괄 전 처리한 후 마스터 파드로 전달되어 최종 값이 계산 될 수 있도록 구성했습니다. 계산된 마지막 값은 Redis 에 푸시하여 Websocket 으로 구독 중인 대시보드에 전달될 수 있도록 하였습니다.

신규 구성은 Apache Camel 과 Springboot 를 활용했습니다. Apache Camel 은 *라우터기반으로 업무를 생성할 수 있습니다. 파드 내에는 개별 메시지에 대한 사전 통계 계산을 하는 스트리밍 라우터와 사전 계산된 데이터를 기반으로 최종 통계를 계산 하는 스트리밍 라우터로 구성하였습니다. 결과적으로 Hadoop의 MapReduce 아키텍처**와 유사하게 구성(Figure 3)했습니다. 스트리밍 처리 방법은 Apache Camel 에서 제공하는 Camel Aggregation Strategy*** 를 활용하였습니다.

*https://camel.apache.org/camel-core/getting-started/index.html#BookGettingStarted-Routes

**https://velog.io/@kimdukbae/MapReduce

***https://camel.apache.org/components/3.20.x/eips/aggregate-eip.html

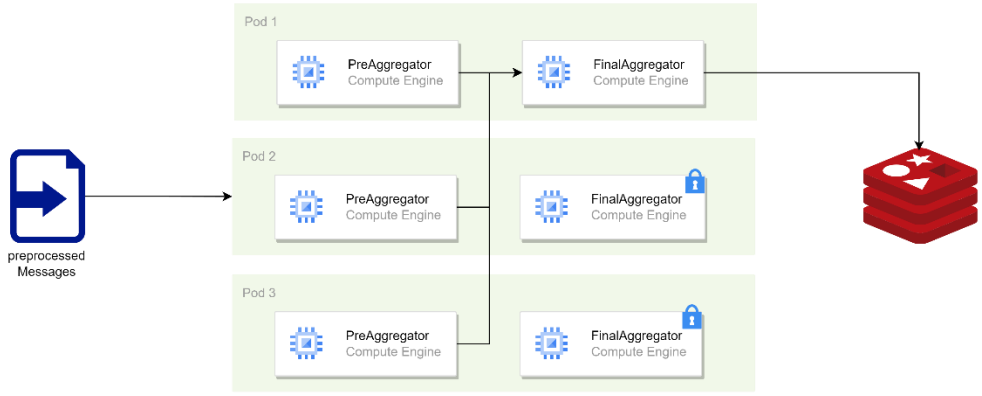

Figure 3 파드 내 라우터 구성 예시

Apache Camel 에서는 쿠버네티스 환경에서의 클러스터링을 지원합니다. Camel-Kubernetes*는 쿠버네티스의 Master 노드(API endpoint)에서 제공하는 파드 정보를 활용한 클러스터 내 서비스 디스커버리를 제공합니다. Camel-master는 쿠버네티스의 Lease 객체를 활용하여 리더(Master)를 선출**하여 Active – Standby 서비스를 구성할 수 있게 해줍니다.

*https://developer.broadleafcommerce.com/production/configuration/camel-cluster-service

**https://developers.redhat.com/articles/2021/09/23/leader-election-kubernetes-using-apache-camel#

리스(Lease)*

리스는 분산 시스템에서 공유 리소스를 잠그고 노드 간의 활동을 조정하는 메커니즘입니다. 쿠버네티스에서 "리스" 개념은 coordination.k8s.io API 그룹**에 있는 Lease 오브젝트로 표현되며, 노드 하트비트 및 컴포넌트 수준의 리더 선출과 같은 시스템 핵심 기능에서 사용됩니다.

*https://en.wikipedia.org/wiki/Lease_(computer_science)

**https://kubernetes.io/docs/concepts/architecture/leases/

Camel-kubernetes 클러스터링을 위해서 다음과 같이 설정 되어야 합니다.

```yaml
camel:
    cluster:
        kubernetes:
            enabled: true
            id: ${random.uuid}
            master-url: https://10.21.88.159
            lease-duration-millis: 120000
            renew-deadline-millis: 80000
            retry-period-millis: 10000
            kubernetes-namespace: default
            cluster-labels:
               app: stream-aggregator
        controller:
            enabled: true
            namespace: lock2
            routes[heartbeat]:
                clustered: false
    main:
        dev-console-enabled: true
    springboot:
        name: stream-aggregator
        main-run-controller: true
```

Figure 4 어플리케이션 설정 예시

쿠버네티스 내 서비스 디스커버리와 리스 객체 점유를 위해서 아래와 같이 계정 및 권한 설정을 합니다.

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: camel-leader-election
apiVersion: rbac.authorization.k8s.io/v1
---
kind: Role
metadata:
  name: camel-leader-election
rules:
  - apiGroups:
      - ""
    resources:
      - pods
    verbs:
      - get
      - list
  - apiGroups:
      - coordination.k8s.io
    resources:
      - leases
    verbs:
      - get
      - create
      - update
      - list
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: camel-leader-election
subjects:
  - kind: ServiceAccount
    name: camel-leader-election
roleRef:
  kind: Role
  name: camel-leader-election
  apiGroup: rbac.authorization.k8s.io
apiVersion: apps/v1
---
kind: Deployment
metadata:
  name: stream-server
  labels:
    app: stream-server
spec:
  replicas: 2
  selector:
    matchLabels:
      app: stream-server
  template:
    metadata:
      labels:
        app: stream-server
    spec:
      serviceAccountName: camel-leader-election
      containers:
      - name: stream-server-container
        image: us-west2-docker.pkg.dev/gke-subsystems/stream-server:v0.4.0-rc.0
```

Figure 5 ServiceAccount 설정 예시

위 설정을 적용하면 Figure 6 로그에서 보여지는 것처럼 클러스터 목록과 최초 선출된 리더를 확인할 수 있습니다. 최초 선정된 리더 파드를 종료하면서 기존 파드 목록 중에서 신규 리더가 선출되는 것을 확인할 수 있습니다. Figue 7 과 같인 kubectl 을 통해서도 리스 객체를 점유하고 있는 파드의 아이디를 확인할 수 있습니다. kubectl get leases leaders-lock2 -o yaml 명령어를 사용하면 좀더 상세한 내용을 확인 할 수 있습니다.

```
2023-10-30 16:03:27.868 TimedLeaderNotifier - L:166 The cluster has a new leader: Optional[stream-aggregator-5c6b74f548-p672z]
2023-10-30 16:03:27.962 TimedLeaderNotifier - L:178 The list of cluster members has changed: [stream-aggregator-5c6b74f548-p672z, stream-aggregator-5c6b74f548-pjvsj]
2023-10-30 16:09:56.947 TimedLeaderNotifier - L:166 The cluster has a new leader: Optional[stream-aggregator-5c6b74f548-pjvsj]
2023-10-30 16:09:56.953 TimedLeaderNotifier - L:178 The list of cluster members has changed: [stream-aggregator-5c6b74f548-b4b7b, stream-aggregator-5c6b74f548-pjvsj]
```

Figure 6 로그로 확인하는 리더 클러스터

Figure 7 kubectl 로 확인한 리더 변경 내역

적용 결과

신규 구성을 적용한 후 기존 구성 대비 미확인 메시지 수와 가장 오래된 미확인 메시지 기간이 감소된 것을 확인 할 수 있었습니다. 기존 구성만큼 안정적으로 서비스 제공 됨을 확인할 수 있었습니다.

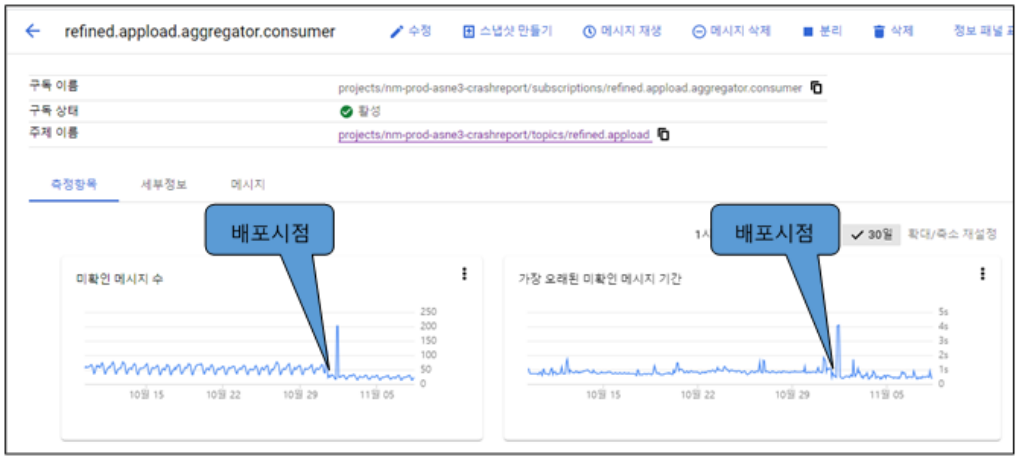

Figure 8 아키텍처 전환 후 PubSub 메시지 처리 성능 비교

심볼리케이팅 캐시 데이터 공유

두번째로 파드 클러스터링을 이용해서 개별 파드 내 캐시 데이터를 공유하도록 구성했습니다. 크래시리포트의 주요 기능 중 하나인 심볼리케이팅*은 크래시가 발생한 Thread에서 나오는 BackTrace에 포함된 Framework 의 내용을 사람이 읽을 수 있는 형태로 변환 하는 작업을 의미합니다. 아래 이미지는 심볼리케이팅 전/후에 대한 예시**입니다.

```
libsystem_platform.dylib            0x00000001826c130c _sigtramp + 36
mahya                               0x0000000100006e174 mahya + 156020
mahya                               0x0000000100006d060 mahya + 151648
mahya                               0x0000000100006ad34 mahya + 142644
UIKit                               0x0000000189925d74  + 184
UIKit                               0x00000001898eedb8  + 128
```

Figure 9 심볼리케이팅 전 변환 대상 mahya framework

```
libsystem_platform.dylib            0x00000001826c130c _sigtramp + 36
-[MHViewController countlyProductionTest] (in mahya) (MHViewController.m:620)
-[MHViewController transitionToMahya] (in mahya) (MHViewController.m:443)
-[MHViewController textFieldShouldReturn:] (in mahya) (MHViewController.m:210)
UIKit                               0x0000000189925d74  + 184
UIKit                               0x00000001898eedb8  + 128
```

Figure 10 심볼리케이팅 후 결과

*https://hcn1519.github.io/articles/2020-02/crash_report_symbolication

**https://support.count.ly/hc/en-us/articles/360037261472-Crash-Symbolication

기존 캐시 시스템의 한계

크래시를 수집하다 보면 동일한 크래시가 반복적으로 수집됨을 알 수 있습니다. 같은 크래시가 수집되기 때문에 심볼리케이팅 소요시간을 단축하기 위해 발생 프레임워크명과 메모리 주소 정보 등을 활용해서 개별 파드 내 캐시 데이터를 생성하고 재 사용하고 있습니다. 캐시 데이터가 각자 파드에 쌓이기 때문에 모든 파드에 캐시 데이터가 쌓이기 전까지는 캐시를 활용한 성능 개선에 한계를 갖고 있었습니다. 그래서 캐시 간 데이터가 공유되어 캐시 데이터 수집 시간 단축이 될 수 있다면 캐시의 효율성을 높이고 신규 배포시 캐시가 초기화 되는 문제도 일정부분 해소 될 것으로 판단됩니다. 이를 위해서 파드 간 클러스터링을 활용하려 합니다.

헤이즐캐스트(Hazelcast)*

기존 구성에서는 embedded H2DB 를 활용해서 JPA 기반 캐싱을 사용하였습니다. 하지만 이 경우 파드간 데이터를 공유하기 어려워 캐싱으로 얻을 수 있는 장점이 줄어 들었습니다. 신규 구성에서는 헤이즐캐스트를 이용한 Key-Value 캐싱을 사용하려 합니다. 헤이즐캐스트는 쿠버네티스 내 서비스 디스커버리를 간단한 설정만으로 지원하여 클러스터링을 통한 데이터 공유를 쉽게 구성할 수 있게 해줍니다. 구체적인 서비스 설정은 다음과 같습니다.

*https://docs.hazelcast.com/hazelcast/5.1/kubernetes/kubernetes-auto-discovery

```
hazelcast:
    cluster-name: hazelcast-stg-cluster
    map:
        products:
            max-idle-seconds: 3600
            eviction:
                eviction-policy: LRU
                max-size-policy: PER_NODE
                size: 200
    network:
        join:
            multicast:
                enabled: false
            kubernetes:
                enabled: true
                namespace: default
                service-name: symbolicator-service
                kubernetes-master: https://10.168.0.111
```

Figure 11 Spring Boot 내 쿠버네티스 클러스터링용 hazelcast 설정 예시

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: symbolicator-cluster-account
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: symbolicator-cluster-account
rules:
  -   apiGroups:
        - ""
        - apps
      resources:
        - endpoints
        - pods
        - nodes
        - services
        - statefulsets
      verbs:
        - get
        - list
        # Watching resources is only required to support automatic cluster state management
        - watch
  -   apiGroups:
        - "discovery.k8s.io"
      resources:
        - endpointslices
      verbs:
        - get
        - list
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: symbolicator-cluster-account
subjects:
  - kind: ServiceAccount
    name: symbolicator-cluster-account
roleRef:
  kind: Role
  name: symbolicator-cluster-account
  apiGroup: rbac.authorization.k8s.io
```

Figure 12 Hazelcast 클러스터를 위한 계정 및 권한 설정 예시

```yaml
apiVersion: v1
kind: Service
metadata:
  name: symbolicator-service
  annotations:
    beta.cloud.google.com/backend-config: '{"ports": {"8080":"symbolicator-backend-config"}}'
spec:
  type: NodePort
  selector:
    app: symbolicator
  ports:
    - name: tomcat
      port: 8090
      targetPort: 8090
      nodePort: 30001
    - name: hazelcast
      port: 5701
      targetPort: 5701
      nodePort: 30051
```

Figure 13 Hazelcast 간 통신을 위한 포트 설정

위 설정을 적용 후 배포를 하면 Figure 14 와 같은 로그가 출력됩니다. 로그 출력을 통해서 설정이 정상 작동하는 것을 확인 할 수 있습니다.

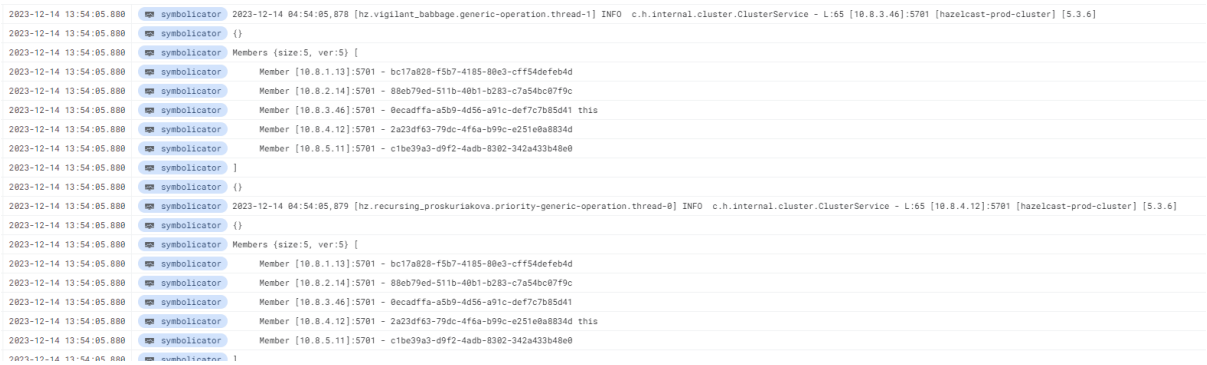

Figure 14 Hazelcast 클러스터링 성공 로그

적용 결과

파드에서 발생한 캐시 데이터가 주변 파드의 캐시에 공유됨으로써 크래시의 심볼리케이팅을 수행하는 평균 시간이 감소한 것을 확인할 수 있었습니다. Figure14 를 통해서 최대값 기준 기존 대

비 50% 정도 성능 개선이 있었습니다. 헤이즐케스트는 쿠버네티스의 배포 과정에서 기존 캐시 데이터를 신규 파드에 동기화하는 옵션*을 제공합니다. 이를 활용하면 신규 기능이 배포되어도 기존 캐시데이터를 유지하기 때문에 배포에 따른 초기 성능 지연 문제도 해소될 수 있습니다.

*https://docs.hazelcast.com/hazelcast/5.1/kubernetes/kubernetes-auto-discovery#preventing-data-loss-during-upgrades

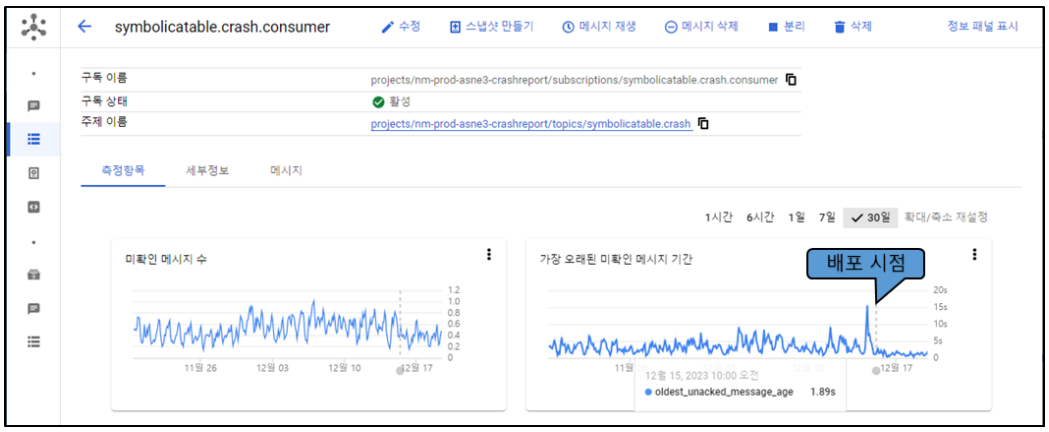

Figure 15 가장 오래된 미확인 메시지의 평균 시간이 감소됨

마치며

쿠버네티스는 개별 파드에서 발생 데이터가 주변 파드에 영향을 주지 않는 것이 기본 구성이지만 사용자가 필요하다면 공유할 수 있는 방법을 제공하고 있습니다. 아파치 카멜과 헤이즐캐스트와 같은 OSS 들은 이미 쿠버네티스에서 제공하는 클러스터링 절차에 따른 클러스터링 기능을 제공하고 있었습니다. 저는 이를 이용해 2가지 상황에 적절하게 사용하였습니다.

파드 클러스터링과 리더 선출을 통해서 MapReduce 아키텍처를 구성할 수 있었고 이를 통해 실시간 스트리밍 데이터 처리 구성이 가능했습니다. 쿠버네티스가 갖는 스케일 아웃의 장점을 유지하면서 실시간 통계 계산도 가능하게 되었습니다.

파드 클러스터링을 통해 캐시 데이터가 공유되도록 구성하였습니다. 모든 파드에 캐시 데이터가 쌓이는 시간을 단축되어 전체적으로 심볼리케이팅에 소요되는 시간이 감소하였습니다. 이는 메시지 처리량을 증대되는 개선으로 이어졌습니다.

저는 파드 클러스터링을 통해서 쿠버네티스에 대해서 좀더 자세히 알게 되었습니다. 그리고 크래시리포트 시스템이 갖고 있었던 작지만 작지 않았던 문제를 개선하였습니다. 위 2가지 사례 외에도 클러스터링을 활용한 개선 사례는 더 있을 것으로 생각합니다. 이 사례들을 통해서 여러분의 시스템에 있는 작은 문제들을 해결하는 또 다른 방법 중에 하나로 고려 되기를 희망합니다.

참고자료

1. https://catsbi.oopy.io/c95e714b-0d28-49d0-aff5-10acd5d0dce5
2. https://spring.io/projects/spring-cloud-netflix
3. https://cloud.google.com/dataflow?hl=ko
4. https://cloud.google.com/dataflow/docs/support/sdk-version-support-status?hl=ko
5. https://camel.apache.org/camel-core/getting-started/index.html#BookGettingStarted-Routes
6. https://velog.io/@kimdukbae/MapReduce
7. https://camel.apache.org/components/3.20.x/eips/aggregate-eip.html
8. https://developer.broadleafcommerce.com/production/configuration/camel-cluster-service
9. https://developers.redhat.com/articles/2021/09/23/leader-election-kubernetes-using-apache-camel#
10. https://en.wikipedia.org/wiki/Lease_(computer_science)
11. https://kubernetes.io/docs/concepts/architecture/leases/
12. https://hcn1519.github.io/articles/2020-02/crash_report_symbolication
13. https://support.count.ly/hc/en-us/articles/360037261472-Crash-Symbolication
14. https://docs.hazelcast.com/hazelcast/5.1/kubernetes/kubernetes-auto-discovery
15. https://docs.hazelcast.com/hazelcast/5.1/kubernetes/kubernetes-auto-discovery#preventing-data-loss-during-upgrades