Resource Monitoring

1. Resource Monitoring

메트릭 vs 로그

- 메트릭은 특정 기준에 대한 수치
- 로그는 어떤 오류인지 파악하기 위해 사용하는 데이터
- 예를 들면 어플리케이션의 레이턴시가 높아지는 상황을 메트릭을 통해 파악한다면 이로 인해 발생하는 오류에 대한 내용을 파악하기 위해서는 로그를 사용합니다.

모니터링

먼저 모니터링의 방법에 따라 블랙박스와 화이트박스 모니터링으로 나누어집니다.

- 블랙박스 모니터링
  - 어플리케이션의 외부를 모니터링하는 방법
  - 일반적으로 CPU, 메모리와 같은 인프라 수준의 모니터링에 사용
- 화이트박스 모니터링
  - 어플리케이션의 내부 상태를 모니터링하는 방법
  - 요청, 에러율, 레이턴시와 같은 어플리케이션 수준의 모니터링에 사용

- 인프라 수준의 모니터링을 설계할 때에는 사용률, 포화도, 오류율(USE)을 파악하는 것이 중요합니다. 이를 통해 시스템의 리소스에 대한 제한과 오류를 파악할 수 있습니다.
- 어플리케이션 수준의 모니터링을 설계할 때에는 요청, 오류율, 소요 시간(RED)을 파악하는 것이 중요합니다. 이를 통해 어플리케이션을 사용하는 유저의 입장에서 서비스에 대한 경험을 파악할 수 있습니다.

쿠버네티스 메트릭

쿠버네티스 클러스터 및 구동 중인 어플리케이션의 상태를 확인하기 위해서 일반적으로 다음과 같은 요소들을 통해 메트릭을 수집합니다.

- **cAdvisor**

  cAdvisor는 컨테이너 리소스와 관련된 메트릭을 수집하는 오픈소스입니다. 쿠버네티스의 워커노드에 존재하는 kubelet에 내장되어 있으며 내부적으로는 Linux cgroups에 의해 분리된 CPU, Memory, 네트워크와 같은 리소스 메트릭을 커널에 내장된 statfs를 통해 수집합니다. cAdvisor Github을 살펴보면 리눅스 컨테이너 기술을 사용할 때 실행 옵션을 볼 수 있습니다.

- **Metrics-Server**

  이전에 메트릭 수집에 사용되던 `Heapster`는 1.11 버전부터 Deprecated 되기 시작해 1.13 버전 이후에는 내부 컴포넌트에서 완전히 삭제되었습니다. 현재는 Heapster 대신 쿠버네티스의 메트릭을 수집하기 위해 다음과 같은 방법을 사용합니다. (리소스 모니터링 도구 Docs)

  - 리소스 메트릭 HPA, VPA, 스케줄링에 사용되는 리소스 메트릭을 수집하기 위해 표준 API 구현체인 메트릭 서버를 사용합니다. Metric Server는 각 워커 노드에 존재하는 kubelet이 노출하는 API를 통해 메트릭을 수집하여 메모리에 저장하고 이를 통해 클러스터의 스케일링 및 스케줄링에 사용하게 됩니다.
  - 사용자 정의 메트릭표준 구현체인 Metric Server에서 제공하는 메트릭 이외에도 추가로 원하는 메트릭을 수집하기 위해서는 추가로 메트릭을 수집해야 합니다. 이를 위해서는 외부 모니터링 솔루션을 통해 데이터를 수집 및 보관하고 이를 Custom Metric API로 노출시키는 API Adapter를 구현함으로써 HPA 등에 사용할 수 있습니다.
    ex) Prometheus Adapter
  - top

```
kubectl top nodes kubectl top pods kubectl top pods POD_NAME
--sort-by=cpu # List PersistentVolumes sorted by capacity
kubectl get pv --sort-by=.spec.capacity.storage
```

어떤 메트릭을 모니터링 해야 할까?

- 모니터링 솔루션을 구축할 때 생각해야 할 점은 어떤 메트릭을 수집해야할 지에 대한 부분입니다. 가장 좋은 방법은 시스템의 상태를 알 수 있는 모든 메트릭을 수집하는 것이지만, 너무 많은 메트릭으로 인해 어떤 메트릭에 집중해야할 지 결정하기 어려워지는 문제가 존재합니다. 이를 위해서는 모니터링에 대한 정확한 Layer를 나누어 메트릭을 분류하고 실제 문제가 발생했을 때 원인을 파악하기 위해 Layer의 계층에 따라 분석함으로써 모니터링의 효율을 높일 수 있습니다.

  일반적으로 나눌 수 있는 Layer는 다음과 같습니다.

  - 노드(CPU / Memory / Disk)
  - 클러스터 컴포넌트 (etcd)
  - 클러스터 Addon (HPA, Ingress)
  - 사용자 어플리케이션 (Container Resource, Application Metric)

로깅

클러스터의 전체 상태를 파악하기 위해서는 여러 곳에서 발생하는 로그도 수집하여 중앙 집중화 해야할 필요가 있습니다. 로그도 메트릭과 마찬가지로 너무 많은 로그를 수집하는 경우 어떤 로그를 지표로 삼아야하는지 결정하기 어려워지며, 로그 보관에 의한 비용 문제도 발생합니다. 이를 해결하기 위해서는 어떤 로그가 필요한지, 수집된 로그를 얼마나 보관할 지에 대해서 정책을 정할 필요가 있습니다.

수집해야할 로그는 일반적으로 다음과 같습니다.

- **노드 로그**
  - 노드 레벨의 로그
- **컨트롤 플레인 (API Server, Scheduler, Controller Manager) 로그**
  - 쿠버네티스의 오브젝트에 대한 이벤트로는 원인을 파악하기 힘들 경우 컨트롤 플레인에 대한 로그를 통해 보다 쉽게 원인을 파악할 수도 있습니다. 이러한 로그는 호스트의 **/var/log/** 디렉토리 아래에 존재합니다.
- **Audit 로그**
  - 시스템 내부에 누가 무엇을 했는지에 대한 로그로 보안에 관련된 모니터링을 위해 사용합니다. 하지만 대규모의 클러스터 환경에서는 로그 양에 의해 시스템 부하가 발생할 수 있으므로 가이드를 참고하여 구성해야합니다.
- **어플리케이션 컨테이너 로그**
  - 어플리케이션의 상태에 대한 Observability를 제공합니다. 어플리케이션을 수집하는 방식은 직접 STDOUT을 통해 내보내거나 Sidecar를 사용하는 방법과 같이 여러 방법이 존재합니다. 자세한 내용은 관련 문서에서 확인하실 수 있습니다.

```
kubectl logs pod_name
```

로깅 아키텍쳐

쿠버네티스 클러스터에서 로깅 아키텍쳐를 구축하기 위해 다양한 도구가 존재합니다. 일반적으로는 Fluentd를 DaemonSet으로 배포하여 로그를 수집하고 이를 Elasticsearch에 저장 및 Kibana를 통해 시각화하는 EFK 스택을 주로 사용합니다.

알림

모니터링 및 로깅의 최종 목적은 시스템의 이상 징후를 파악하고 이를 개발자에게 알림에 있습니다. 즉, 데이터를 잘 수집하는 것만큼 중요한 것이 이를 통해 적절하게 알림을 보내고 있는지 입니다.

네이버 검색 SRE팀의 발표 자료나 D2글을 보면 너무 많은 알림은 알림 피로를 발생시켜 정말 집중해야할 문제를 놓치게 될 수도 있다고 말합니다. SLO(*SRE 엔지니어링에서 서비스를 운영에 필수적인 기준을 SLO라고 정의합니다.*)를 설정할 때 개발자가 직접 개입해야 하는 상황 이외에 자동으로 해결될 수 있는 플랜이 있다면 알람을 보내지 않음으로 알람 피로를 줄일 수 있는 방법이 될 수 있습니다. 또한, 알람에 대한 임계치를 설정할 때 너무 짧은 주기로 알람을 발생시킨다면 중복이 발생할 수 있기 때문에 임계치에 대한 고려도 필요합니다.

2. 기출 문제 풀이

참고 : kubectl 치트 시트

참고 : Kubectl Reference Docs

❓ Record the extracted log lines

- Cluster: **hk8s**

  `kubectl config use-context hk8s`

- Monitor the logs of pod `custom-app` and: Extract log lines corresponding to error `file not found` Write them to `/tmp/podlog`.

```
kubectl logs -n customera custom-app | grep -i 'file not found'
> /tmp/podlog
```

❓ Find a pod that consumes a lot of cpu

- Cluster: **k8s**

  `kubectl config use-context k8s`

- From the pod label `name=order`, find pods running high CPU workloads and write the **name of the pod** consuming most CPU to the file `/var/CKA2022/cpu_load_pod.txt`.

```
kubectl top pod -A -l name=order --sort-by cpu echo "eshop-order-
5fb8974476-dz8x8" > /var/CKA2022/cpu_load_pod.txt
```

❓ Resource Monitoring

Context