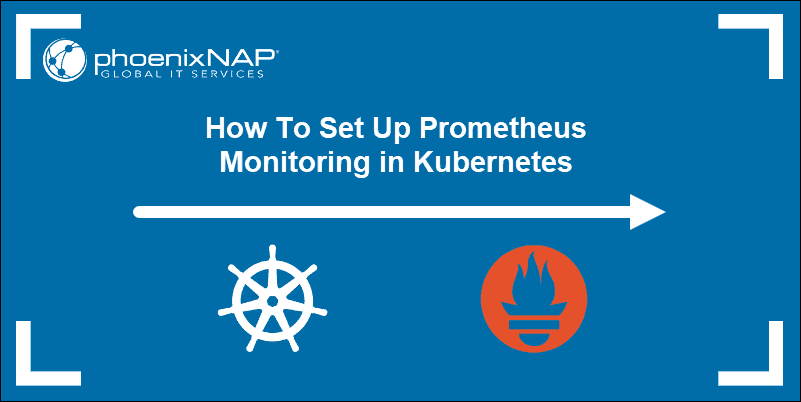
**소개**

Prometheus는 오픈 소스 계측 프레임 워크입니다. Prometheus는 매초 엄청난 양의 데이터를 흡수 할 수 있으므로 복잡한 워크로드에 적합합니다.

Prometheus를 사용하여 서버, VM, 데이터베이스를 모니터링하고 해당 데이터를 활용하여 애플리케이션 및 인프라의 성능을 분석합니다.

**이 문서에서는 Kubernetes 클러스터에서 Prometheus 모니터링을 설정하는 방법을 설명합니다.**



**전제 조건**

* Kubernetes 클러스터
* **kubectl**로컬 컴퓨터에서 완전히 구성된 명령 줄 인터페이스

**Prometheus로 Kubernetes 클러스터 모니터링**

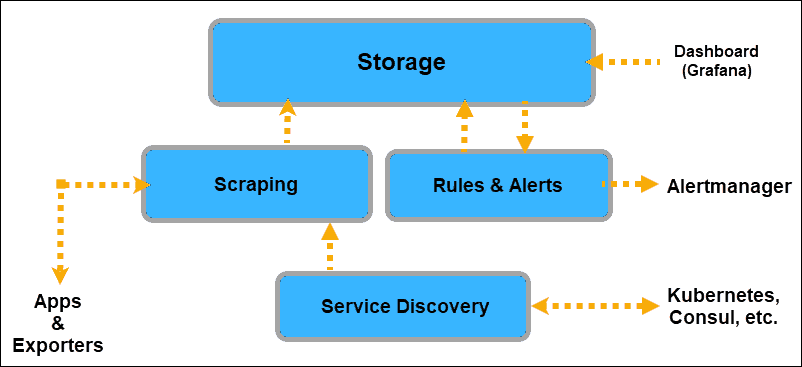
Prometheus는 풀 기반 시스템입니다. ***배포 파일에*scrape** 정의 된 구성에 따라 소위라는 HTTP 요청을 보냅니다 . 이 요청에 대한 응답 은 스크랩 자체에 대한 메트릭과 함께 스토리지에 저장되고 구문 분석됩니다.**scrape**

스토리지는 Prometheus 서버의 사용자 지정 데이터베이스이며 엄청난 양의 데이터를 처리 할 수 ​​있습니다. 단일 서버로 수천 대의 컴퓨터를 동시에 모니터링 할 수 있습니다.

**참고 :** 들어오는 데이터가 너무 많으면 디스크 공간이 빠르게 문제가 될 수 있습니다. 수집 된 데이터는 단기적으로 큰 가치가 있습니다. 광범위한 장기 기록을 보관할 계획이라면 추가 [영구 스토리지 볼륨](https://phoenixnap.com/kb/kubernetes-persistent-volumes) 을 프로비저닝하는 것이 좋습니다 .

데이터는 Prometheus가 수집 할 수 있도록 적절하게 노출되고 형식화되어야합니다. Prometheus는 앱의 클라이언트 라이브러리에서 직접 또는 내보내기를 사용하여 데이터에 액세스 할 수 있습니다.

***내보내기*** 는 전체 제어 권한이없는 데이터 (예 : 커널 메트릭)에 사용됩니다. 내보내기는 애플리케이션 옆에 배치 된 소프트웨어입니다. 그 목적은 Prometheus의 HTTP 요청을 수락하고 데이터가 지원되는 형식인지 확인한 다음 요청 된 데이터를 Prometheus 서버에 제공하는 것입니다.



이제 모든 애플리케이션이 Prometheus에 데이터를 제공 할 수 있습니다. Prometheus에 해당 데이터를 찾을 위치를 알려야합니다. Prometheus는 ***Service Discovery*** 를 사용하여 스크랩 할 대상을 ***검색*** 합니다.

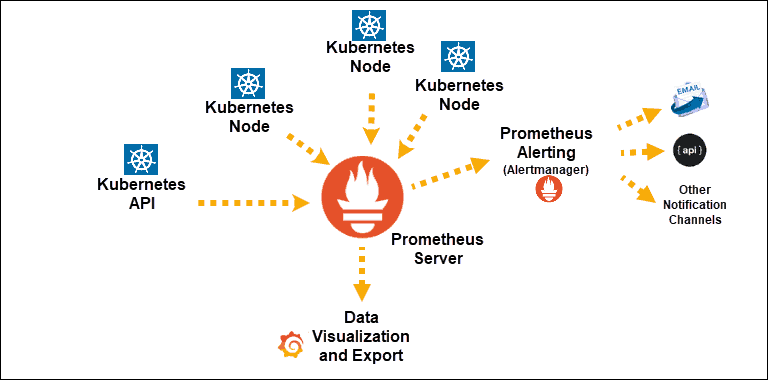
Kubernetes 클러스터에는 이미 레이블과 주석이 있으며 변경 사항과 해당 요소의 상태를 추적하기위한 훌륭한 메커니즘이 있습니다. 따라서 Prometheus는 Kubernetes API를 사용하여 대상을 검색합니다.

Prometheus에 노출 할 수있는 Kubernetes 서비스 검색은 다음과 같습니다.

* ***마디***
* ***끝점***
* ***서비스***
* ***현물 상환 지불***
* ***입구***

Prometheus는 애플리케이션 정보와 별도로 머신 레벨 메트릭을 검색합니다. 메모리, 디스크 공간, CPU 사용량 및 대역폭 측정 항목을 노출하는 유일한 방법은 ***노드 내보내기*** 를 사용하는 것 입니다. 또한 cgroup에 대한 메트릭도 노출되어야합니다.

다행히 cAdvisor 내보내기는 이미 Kubernetes 노드 수준에 포함되어 있으며 쉽게 노출 될 수 있습니다.



시스템이 데이터를 수집하면, 당신은 PromQL 쿼리 언어를 사용하여 액세스와 같은 그래픽 인터페이스로 내보낼 수 있습니다  [Grafana](https://grafana.com/" \t "_blank) , 또는으로 경고를 보내도록 사용  [Alertmanager](https://github.com/prometheus/alertmanager" \t "_blank) .

**Kubernetes에 Prometheus Monitoring 설치**

Prometheus 모니터링은 YAML (Yet Another Markup Language) 파일 세트를 사용하여 Kubernetes 클러스터에 설치할 수 있습니다. 이러한 파일에는 Prometheus가 클러스터의 요소를 스크랩하여 리소스에 액세스하고 정보를 가져올 수 있도록하는 구성, 권한 및 서비스가 포함되어 있습니다.

YAML 파일은 쉽게 추적, 편집 할 수 있으며 무기한 재사용 할 수 있습니다. 이 자습서에 제시된 파일은 GitHub와 같은 온라인 저장소에서 쉽고 무료로 사용할 수 있습니다.

**참고 : .yml** 아래의 파일은 현재의 형태로, 프로덕션 환경에서 사용하기에 의미가 없습니다. 대신 시스템 요구 사항에 맞게 이러한 파일을 적절하게 편집해야합니다.

**모니터링 네임 스페이스 생성**

Kubernetes의 모든 리소스는 네임 스페이스에서 시작됩니다. 하나를 지정하지 않으면 시스템은 기본 네임 스페이스를 사용합니다. 클러스터 모니터링 프로세스를 더 잘 제어하기 위해 모니터링 네임 스페이스를 지정합니다.

네임 스페이스의 이름은 DNS와 호환되는 레이블이어야합니다. 쉽게 참조 할 수 있도록 네임 스페이스의 이름을 ***monitoring*** 으로 지정하겠습니다 .

Kubernetes API에서 측정 항목을 검색하기위한 모니터링 네임 스페이스를 만드는 방법에는 두 가지가 있습니다.

**옵션 1:**

명령 줄 인터페이스에 다음과 같은 간단한 명령을 입력하고 호스트에 ***모니터링*** 네임 스페이스를 만듭니다 .

kubectl create namespace monitoring

**옵션 2 :**

.yml 파일을 만들고 적용합니다.

apiVersion: v1

kind: Namespace

metadata:

name: monitoring

이 방법은 향후 인스턴스에서 동일한 파일을 배포 할 수 있으므로 편리합니다. 명령 터미널에 다음 명령을 입력하여 파일을 클러스터에 적용합니다.

kubectl -f apply namespace monitoring.yml

사용 된 방법에 관계없이 다음 명령을 사용하여 기존 네임 스페이스를 나열합니다.

kubectl get namespaces

**Prometheus 배포 파일 구성**

다음 섹션에는 Kubernetes 클러스터 및 해당 요소에서 Prometheus 스크래핑을 성공적으로 설정하는 데 필요한 요소가 포함되어 있습니다.

섹션은 순서대로 실행되는 개별 **.yml** 파일 로 구현 될 수 있습니다 . 각 파일을 생성 한 후 다음 명령을 입력하여 적용 할 수 있습니다.

kubectl -f apply [name\_of\_file].yml

이 예에서 모든 요소는 단일 **.yml** 파일에 배치되고 동시에 적용됩니다.

이 예제 의 *prometheus.yml* 파일은 **kubectl** 이 Kubernetes API 서버에 요청을 제출하도록 지시 합니다. 파일에는 다음이 포함됩니다.

1. Prometheus가 모든 포드 및 노드에 액세스 할 수 있도록 허용하는 권한입니다.
2. 스크랩해야하는 요소를 정의하는 Prometheus **configMap** 입니다.
3. Prometheus 배포 지침.
4. Prometheus 사용자 인터페이스에 대한 액세스를 제공하는 서비스입니다.

**클러스터 역할, 서비스 계정 및 클러스터 역할 바인딩**

네임 스페이스는 Prometheus에게 해당 클러스터의 모든 리소스에 대한 액세스 권한을 부여해야하는 클러스터 전체 데이터를 검색하려는 경우 기본 역할의 권한을 제한하도록 설계되었습니다. 클러스터 전체 액세스를 제공하는 기본 Prometheus .yml 파일에는 다음 요소가 있습니다.

**1. 클러스터 역할 정의**

각 규칙의 동사는 역할이 apiGroups에서 수행 할 수있는 작업을 정의합니다.

apiVersion: rbac.authorization.k8s.io/v1beta1

kind: ClusterRole

metadata:

name: prometheus

rules:

- apiGroups: [""]

resources:

- nodes

- services

- endpoints

- pods

verbs: ["get", "list", "watch"]

- apiGroups:

- extensions

resources:

- ingresses

verbs: ["get", "list", "watch"]

**2. 서비스 계정 생성**

또한이 역할을 다음에 적용하려면 서비스 계정을 만들어야합니다.

apiVersion: v1

kind: ServiceAccount

metadata:

name: prometheus

namespace: monitoring

**3. ClusterRoleBinding 적용**

마지막으로 ***ClusterRoleBinding*** 을 적용해야합니다 . 이 작업은 이전에 만든 클러스터 역할에 서비스 계정을 바인딩합니다.

apiVersion: rbac.authorization.k8s.io/v1beta1

kind: ClusterRoleBinding

metadata:

name: prometheus

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: prometheus

subjects:

- kind: ServiceAccount

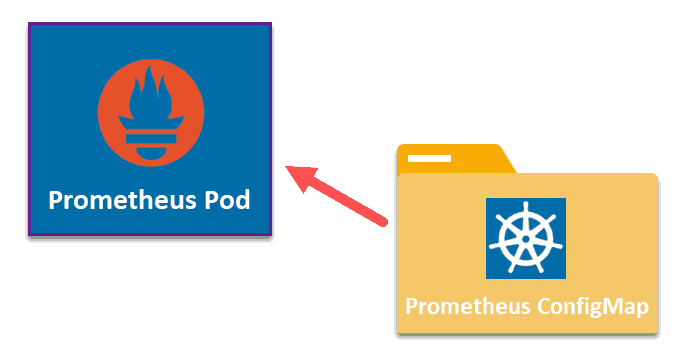
name: prometheus

namespace: monitoring

이러한 리소스를 파일에 추가하여 ***모니터링*** 네임 스페이스 에서 Prometheus 클러스터 전체 액세스 권한을 부여했습니다 .

**Prometheus ConfigMap**

파일의이 섹션에서는 스크래핑 프로세스에 대한 지침을 제공합니다. Kubernetes 클러스터의 각 요소에 대한 특정 지침은 모니터링 요구 사항 및 클러스터 설정에 맞게 사용자 지정해야합니다.



**1. 글로벌 스크랩 규칙**

apiVersion: v1

data:

prometheus.yml: |

global:

scrape\_interval: 10s

**2. 스크랩 노드**

이 서비스 검색은 Kubernetes 클러스터를 구성하는 노드를 노출합니다. kubelet은 모든 단일 노드에서 실행되며 귀중한 정보의 소스입니다.

**2.1 kubelet 스크랩**

scrape\_configs:

- job\_name: 'kubelet'

kubernetes\_sd\_configs:

- role: node

scheme: https

tls\_config:

ca\_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt

insecure\_skip\_verify: true # Required with Minikube.

**2.2 스크랩 cAdvisor (컨테이너 레벨 정보)**

**kubelet는** 단지 용기 자체에 대한 아니라 정보를 제공합니다. 컨테이너 수준에서 정보를 받으려면 내보내기를 사용해야합니다. **cAdvisor는** 이미 내장 만은 필요로한다 **metrics\_path** : **/ 측정 / cadvisor** 프로 메테우스에 대한 수집 컨테이너 데이터를 :

- job\_name: 'cadvisor'

kubernetes\_sd\_configs:

- role: node

scheme: https

tls\_config:

ca\_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt

insecure\_skip\_verify: true # Required with Minikube.

metrics\_path: /metrics/cadvisor

**3. 스크랩 APIServer**

엔드 포인트 역할을 사용하여 각 애플리케이션 인스턴스를 대상으로 지정하십시오. 이 파일 섹션에서는 Kubernetes 클러스터에서 API 서버를 스크 레이 핑 할 수 있습니다.

- job\_name: 'k8apiserver'

kubernetes\_sd\_configs:

- role: endpoints

scheme: https

tls\_config:

ca\_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt

insecure\_skip\_verify: true # Required if using Minikube.

bearer\_token\_file: /var/run/secrets/kubernetes.io/serviceaccount/token

relabel\_configs:

- source\_labels: [\_\_meta\_kubernetes\_namespace, \_\_meta\_kubernetes\_service\_name, \_\_meta\_kubernetes\_endpoint\_port\_name]

action: keep

regex: default;kubernetes;https

**4. Kubernetes 서비스 용 스크랩 포드 (API 서버 제외)**

모든 Kubernetes 서비스를 지원하는 팟 (Pod)을 스크랩하고 API 서버 메트릭을 무시하십시오.

- job\_name: 'k8services'

kubernetes\_sd\_configs:

- role: endpoints

relabel\_configs:

- source\_labels:

- \_\_meta\_kubernetes\_namespace

- \_\_meta\_kubernetes\_service\_name

action: drop

regex: default;kubernetes

- source\_labels:

- \_\_meta\_kubernetes\_namespace

regex: default

action: keep

- source\_labels: [\_\_meta\_kubernetes\_service\_name]

target\_label: job

**5. 포드 역할**

작업 레이블로 컨테이너 이름을 사용하여 메트릭 이름이있는 모든 포드 포트를 검색합니다.

- job\_name: 'k8pods'

kubernetes\_sd\_configs:

- role: pod

relabel\_configs:

- source\_labels: [\_\_meta\_kubernetes\_pod\_container\_port\_name]

regex: metrics

action: keep

- source\_labels: [\_\_meta\_kubernetes\_pod\_container\_name]

target\_label: job

kind: ConfigMap

metadata:

name: prometheus-config

**6. ReplicaSet 구성**

필요한 복제본 수와 정의 된 포드 집합에 적용 할 템플릿을 정의합니다.

apiVersion: apps/v1beta2

kind: Deployment

metadata:

name: prometheus

spec:

selector:

matchLabels:

app: prometheus

replicas: 1

template:

metadata:

labels:

app: prometheus

spec:

serviceAccountName: prometheus

containers:

- name: prometheus

image: prom/prometheus:v2.1.0

ports:

- containerPort: 9090

name: default

volumeMounts:

- name: config-volume

mountPath: /etc/prometheus

volumes:

- name: config-volume

configMap:

name: prometheus-config

**7. nodePort 정의**

Prometheus는 현재 클러스터에서 실행 중입니다. *prometheus.yml* 파일에 다음 섹션을 추가하면 Prometheus가 수집 한 데이터에 액세스 할 수 있습니다.

kind: Service

apiVersion: v1

metadata:

name: prometheus

spec:

selector:

app: prometheus

type: LoadBalancer

ports:

- protocol: TCP

port: 9090

targetPort: 9090

nodePort: 30909

**prometheus.yml 파일 적용**

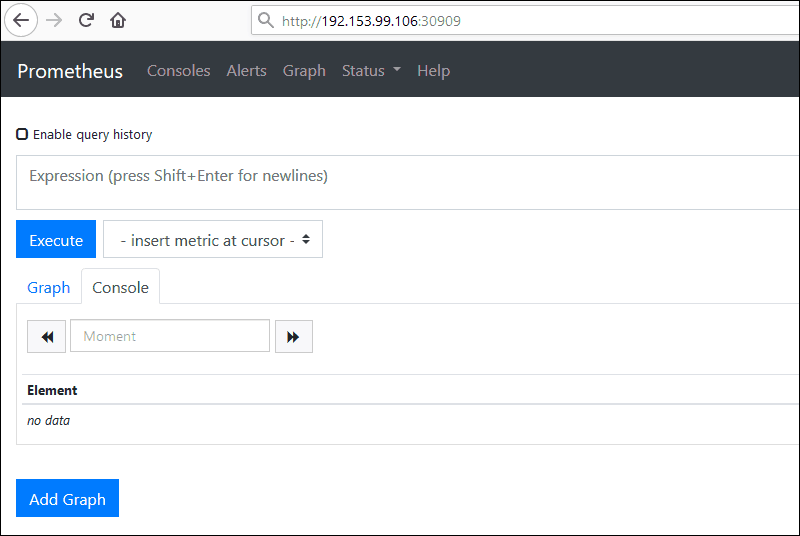
파일에 정의 된 구성 맵은 배포의 모든 포드에 구성 데이터를 제공합니다.

kubectl apply -f prometheus.yml

브라우저에서 Prometheus에 액세스 하려면 *prometheus.yml* 파일에 정의 된 개별 노드 URL 및 nodePort 를 사용하십시오. 예를 들면 :

http://192.153.99.106:30909

노드의 URL 또는 IP를 입력하고 yml 파일에서 포트를 지정하면 Prometheus Monitoring에 성공적으로 액세스 할 수 있습니다.



**참고** : Prometheus에서 수집 한 메트릭을 그래프로 표시하기 위해 포괄적 인 대시 보드 시스템이 필요한 경우 사용 가능한 옵션 중 하나는 Grafana입니다. 데이터 소스를 사용하여 그래프를 만드는 데 사용되는 정보를 검색합니다.

**kube-state-metrics를 모니터링하는 방법은 무엇입니까? (선택 과목)**

이제 [Kubernetes 인프라](https://phoenixnap.com/blog/kubernetes-monitoring-best-practices) 와 애플리케이션 인스턴스 를 완전히 [모니터링](https://phoenixnap.com/blog/kubernetes-monitoring-best-practices) 할 수 있습니다 . 그러나 여기에는 Kubernetes가 클러스터의 리소스에 대해 가지고있는 정보에 대한 메트릭이 포함되지 않습니다.

kube-state-metrics는 Prometheus가 해당 정보를 스크랩 할 수 있도록하는 내보내기 도구입니다. kube-state-metrics 내보내기를위한 YAML 파일을 만듭니다.

---

apiVersion: apps/v1beta2

kind: Deployment

metadata:

name: kube-state-metrics

spec:

selector:

matchLabels:

app: kube-state-metrics

replicas: 1

template:

metadata:

labels:

app: kube-state-metrics

spec:

serviceAccountName: prometheus

containers:

- name: kube-state-metrics

image: quay.io/coreos/kube-state-metrics:v1.2.0

ports:

- containerPort: 8080

name: monitoring

---

kind: Service

apiVersion: v1

metadata:

name: kube-state-metrics

spec:

selector:

app: kube-state-metrics

type: LoadBalancer

ports:

- protocol: TCP

port: 8080

targetPort: 8080

다음 명령을 입력하여 파일을 적용하십시오.

kubectl apply -f kube-state-metrics.yml

파일을 적용한 후 노드 IP / URL을 입력하고 이전에 정의한대로 정의 된 nodePort를 입력하여 Prometheus에 액세스하십시오.

**결론**

이제 Kubernetes 클러스터에 Prometheus Monitoring을 성공적으로 설치 했으므로 시스템의 전반적인 상태, 성능 및 동작을 추적 할 수 있습니다. 운영이 아무리 크고 복잡하더라도 Prometheus와 같은 메트릭 기반 모니터링 시스템은 분산 [마이크로 서비스 기반 아키텍처](https://phoenixnap.com/kb/introduction-to-microservices-architecture) 를 유지하기위한 필수 도구입니다 .