Ch02. 프로메테우스 시작하기

2.1. Download & Install Prometheus:

- Repository: https://prometheus.io/download/
- 작성기준 LTS버젼: 2.37.6, 2023-02-20
- Prometheus Tarball을 download받고, 압축을 푼 후, 실행할 위치로 옮겨 놓고, 설정파일인 `prometheus.yml`을 확인해 본다. 몇가지 내용을 확인해 보기 위해 수정해 보았다.

```
- job_name: "myPROM"
    static_configs:
      - targets: ["localhost:9090"]
```

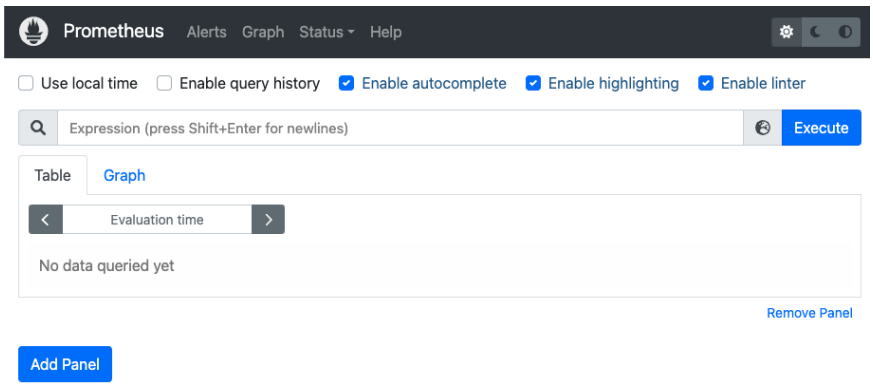
- 이후 `./prometheus` 명령으로 프로메테우스를 실행한다.

```
$ tar -zxvf prometheus-2.37.6.linux-amd64.tar.gz
$ ln -s prometheus-2.37.6.linux-amd64 prometheus
$ cd prometheus
$ ./prometheus
```

- Listen Port는 9090이다. 필요시 ufw, NAT등을 정리해 준다.

2.2. 1st Use: Web Access to prometheus

- `http://installhost:9090/` 로 접속한다.
- 기본적으로 PromQL 쿼리를 수행할 수 있는 페이지가 보인다. (여기가 Console View이며, 동시에 /graph 페이지 이다.)
  - 기본적으로 Panel단위로 PromQL표현식으로 adhoc쿼리를 수행하고 결과를 알 수 있으며, 데이터의 디버깅에도 사용된다.



- Status 메뉴 밑에서는 프로메테우스가 수행하는 작업을 파악할 수 있는 메뉴들이 있다.
  - Status>Targets를 보면, 최근 수집한 내역에 대한 정보를 확인 할 수 있다.

- URL에서 /metrics로 접근해 본다. 수집하고 있는 metric들에 대한 내용이다. 프로메테우스 자체도 프로메테우스가 Metric으로 계측(수집)하고 있다. 프로메테우스 코드가 아닌, Go-Runtime 같은 내용도 포함되어 있음에 주의한다.

```
prometheus_target_metadata_cache_bytes{scrape_job="myPROM"} 10281
```

  - 참고로 이것은 프로메테우스 자체에 대한 것이며, node_exporter로 확인되는 것과는 다른 것이다. http://localhost:9090/metrics 에서 다음과 같은 내용을 확인 할 수 있다.

2.3. 1st Use: PromQL, Graph

- Expression을 입력하고 실행할 수 있는 Console이 기본으로 보여진다.
  - Console의 Exresssion Browser에 쿼리 `up` 을 실행하면, 결과는 `up{instance="localhost:9090", job="prometheus"}, 1` 을 보여준다.
  - instance는 수집된 대상을 나타내는 LABEL이다. 해당 실행에 대한 결과는 성공이고, jobname LABEL은 “prometheus” 라고 되어 있는데, 이것은 사용자가 설정으로 생성해야 하는 규칙에 따른 것이다.
  - 보통 일반적으로 jobname이라는 LABEL은 Application의 형식을 나타낼 때 사용한다.
- 쿼리 `process_resident_memory_bytes` 실행하면, 현재 수집 내역 중, 실제 사용중인 메모리 크기를 조회한다.
  - 프로메테우스에서 사용하는 단위는 보통 byte, sec, UTC time 단위 같은 기본 단위를 사용한다. 정확한 집계와 표현을 위해서 이다. 축약된 수와 단위의 표현은 보통 Grafana나 다른 Front Tool에 맡긴다.
- 현 상태에서 “쿼리탭”이 아닌, “Graph탭”을 선택해 본다. 시간에 따른 조회된 결과가 그래프로 표햔될 것이다.

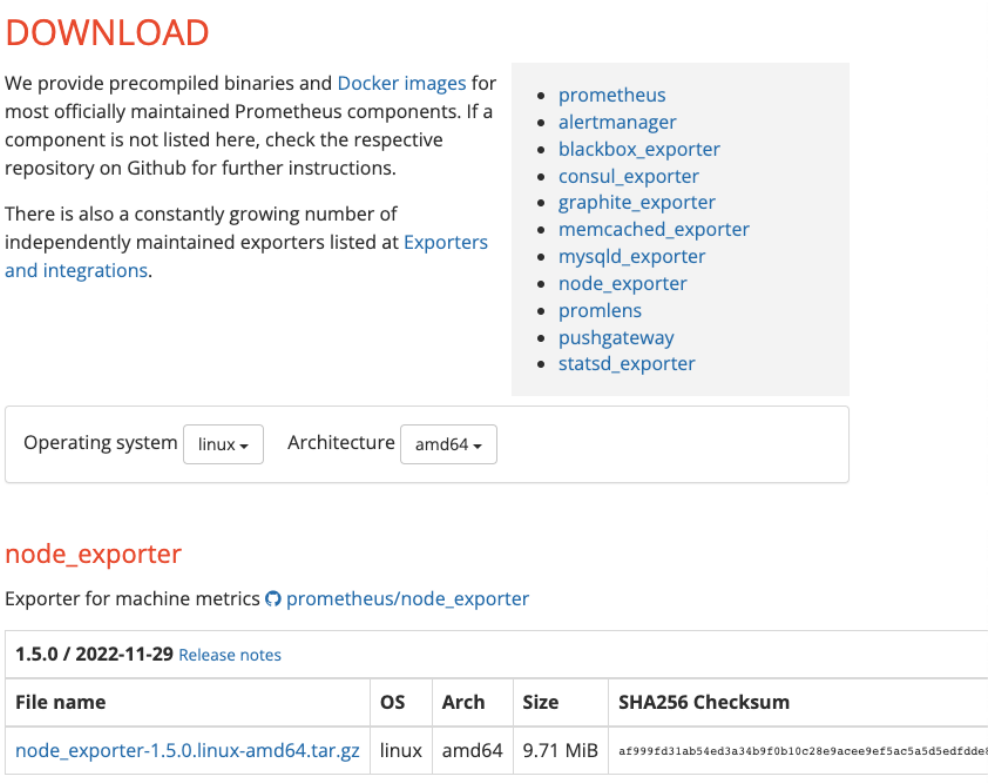
2.4. Metric의 주요 Type

- (1) Gauge: `process_resident_memory_bytes` 같은 Metric은 Gauge라고 부른다. 특정 시점의 절대시점의 상태값이 중요한 의미를 갖는다.
  - 특정시점의 snapshot된 상태값이 scrape되므로 scrape 실패는 데이터의 손실,
  - 보통 해당 시점의 특정 값 정보가 중요 (CPU사용율, HTTP동시접속수, 큐의 적재량, Throughput 개념의 것들)
- (2) Counter: 이벤트의 발생 횟수, 집합(이벤트 전체)의 수 같은 누적값을 의미한다. 카운터는 절대값보다는 증감의 변화도가 더 중요한 의미를 갖는다. 그래서 아래와 같이 단위 시간동안의 증감정도를 아는 것으로 사용된다.
  - scrape가 실패해도 누적되는 성격의 값이므로 다음 scrape로 진행, 인스턴스 재시작 등으로 카운터 초기화시 데이터 의미 손실, rate(), irate() 추이 분석에 주로 활용
  - 보통 단위(지정되는) 시간당 처리된 양에 대한 정보가 중요 (Request수, Function수행 횟수 등 누적 개념의 것들)
  - e.g.) Counter Type인, `prometheus_tsdb_head_samples_appended_total` 을 확인해 본다.
  - e.g.) `rate(prometheus_tsdb_head_samples_appended_total[1m])` 함수를 사용하여 확인을 해본다. 1분동안의 평균적 초당 처리(저장) 샘플 갯수이다. (rate함수는 프로세스가 재시작되거나, 정확하게 정렬되지 않는 경우, Counter를 재설정한다. 그래도 Counter절대값이 아니라, Counter 변화정도를 의미있게 보므로 무리가 없다.)
- (3) Summary Type:
  - 수집된 데이터의 분위수를 클라이언트측에서 계산한다. 주로 App에서 발생하는 정보나 GC정보등을 보여주기 위해 사용. 분위수 자체가 다른 수로 대체될 수 없는 특징이 있음
- (4) Histogram Type:
  - 그룹(Bucket)화된 데이터에 대해 분위수 형태로 계산하여 표현한다. 주로 분포에 대해 2차워, 3차원의 표현을 하기위해 사용함

- “1분동안 평균적으로 초당 처리량 = 1분동안 모든 처리량 평균”에 주의 한다.

2.5. Node Exporter실행해 보기

- Node Exporter는 유닉스계열 시스템의 커널, 머신 레벨의 기본적인 Metric들을 표시 한다. Node Exporter는 CPU, Mem, Disk, IO, Net, HW/Resource Status등 모든 표준 수준의 Metric을 제공한다.
  - Windows계열인 경우, (`https://github.com/martinlindhe/wmi_exporter/`)를 사용한다.
- Node Exporter(=Machine Metric)는 Host Node기준이며, 다음의 Metric은 지원하지 않는다.
  - 개별 프로세스(Custom Application Process, User Process)에 대한 Metric
  - (일반적인, 알려진) Application에 대한 Proxy-Metric
  - 다른 Exporter에서 제공하는 Metric
- 특정 Machine Metric을 지원하는 Node Exporter의 미리 빌드되어 있는 버젼은 아래에서 구할 수 있으며, 예제로 AMD64 아키텍쳐기반 Linux버젼을 다운로드 해본다.
  - `https://prometheus.io/download/` 에서 적절한 node_exporter을 다운로드 받는다.
  - 참고로 prometheus, alertmanager 이외에 여러 종류의 exporter와 promlens, pushgateway가 있는 것을 참고하자. prometheus가 어떤 구성 방식으로 무엇을 하려는지 짐작이 된다.



- 작성기준 버젼: 1.5.0/2022-11-29

```
$ tar -zxvf node_exporter-1.5.0.linux-amd64.tar.gz
$ ln -s node_exporter-1.5.0.linux-amd64 node_exporter
$ cd node_exporter
$ ./node_exporter
```

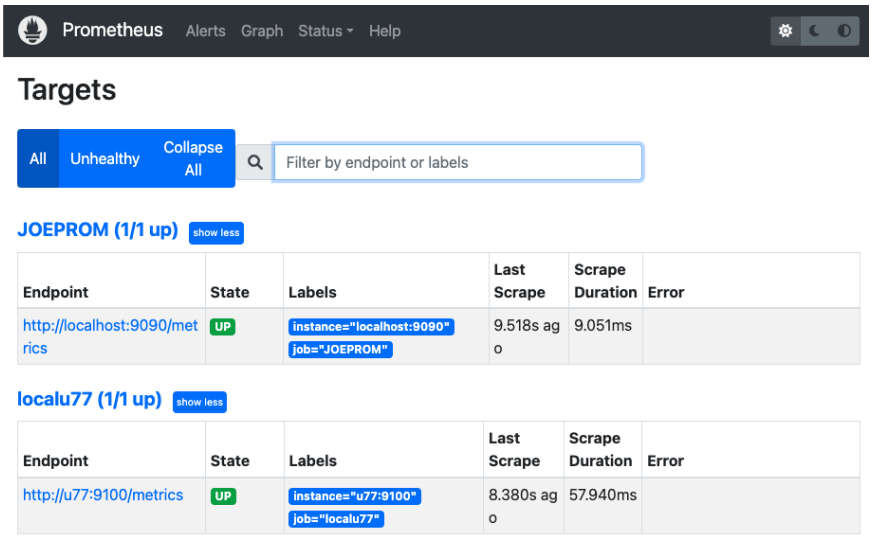
- Listen Port는 9100이다. 필요시 ufw, NAT등을 정리해 준다.
- 이제 `http://localhost:9100/metrics` 로 접근해 본다. node_exporter가 제공하는 해당 node에 대한 정보들이 담겨져 있다. 참고로 내용중에 node_uname_info에 해당 node가 누구인지에 대한 정보가 있다

2.6. Prometheus에서 Node Exporter확인해 보기

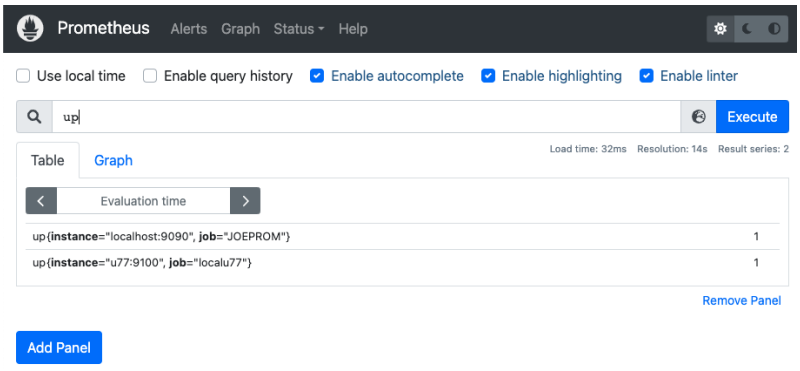
- 프로메테우스가 동작하고 있는 위의 node exporter를 모니터링 하려면, `prometheus.yml` 파일에 해당 node exporter에 대한 내용을 scrape_configs 항목으로 추가하여야 한다. 다음을 참고한다.

```
- job_name: "nodeExu77"
    static_configs:
      - targets: ["localhost:9100"]
```

- 앞에서 이미 Node exporter는 기동 시켜 놓았고, 위의 설정을 반영하기 위하여, 프로메테우스를 다시 기동하고, Targets페이지를 확인해 본다. `prometheus.yml` 에 설정한 `scrape_configs`에 있는 내용에 해당 하는 내용을 확인 할 수 있다.

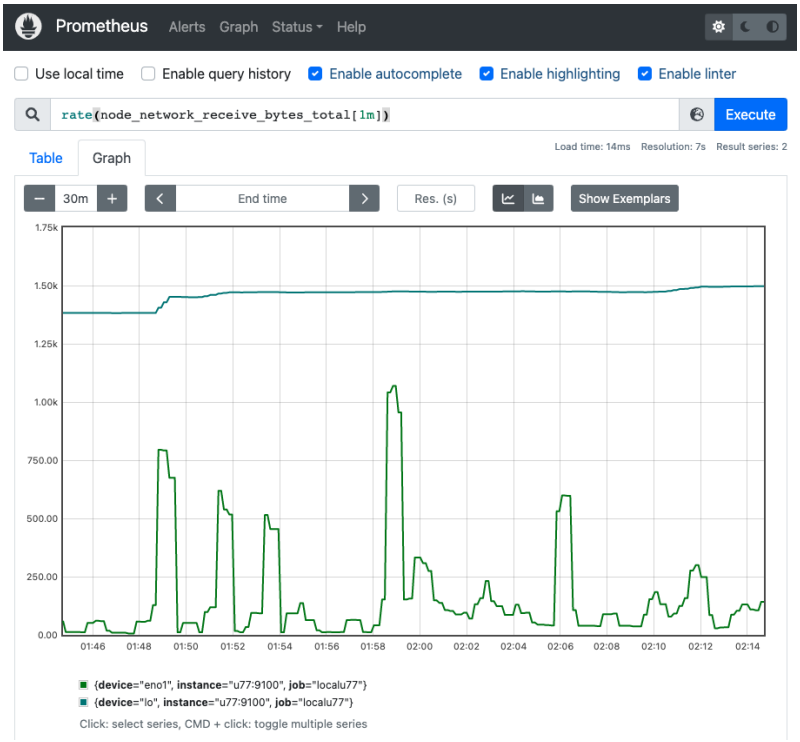


- 콘솔의 수식브라우져에서 'up' 명령을 수행해 보면, 다음과 같은 내용을 확인 할 수 있다.



- 수식 브라우져(Expression Browser)에서 `process_resident_memory_bytes`를 조회하면, 프로메테우스가 수집한 해당 되는 모든 내용이 나오며, `process_resident_memory_bytes{job="nodeExu77"}`를 조회하면, 지정된 LABLE에 대한 내용만 조회된다.

- 이번에는 Guage 특성의 값이 아닌, 변화도를 알아보는 것이 중요한 Counter 특성의 값을 조회해 본다. `rate(node_network_receive_bytes_total[1m])` 를 조회해 본다. 해당 node exporter가 사용할 수 있는 모든 NIC에 대한 단위 시간 사용량에 대한 정보를 확인 할 수 있다.

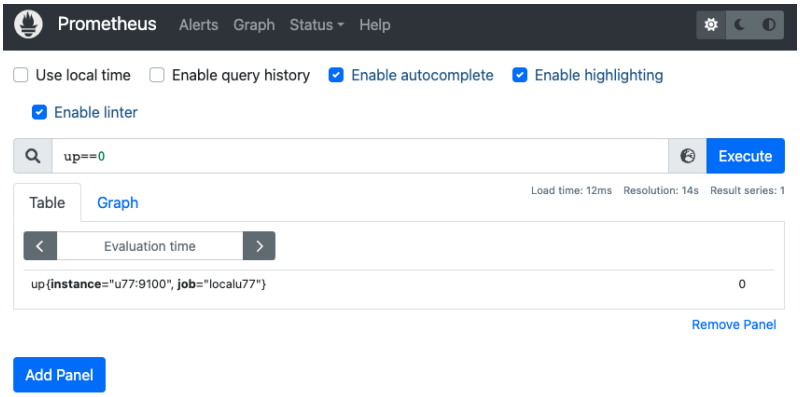


2.7. Alert

- Alert는 크게 두 부분으로 구성된다.
  - (1) 프로메테우스에 알림 규칙을 추가하고, 알림 규칙을 구성하는 로직을 정의해 주는 것
  - (2) 발생된 Alert들을 Alert Manager가 eMail, SMS, Chat 등의 메세지로 변경하여 전달하는 것
- 우선 Alertmanager를 설치해 본다. 앞의 Node Exporter처럼, `https://prometheus.io/download/` 에서 적절한 `alertmanager`를 다운로드 받아 설치한다.
  - 작성기준 버젼: 0.25.0/2022-12-22

```
$ tar zxvf alertmanager-0.25.0.linux-amd64.tar.gz
$ ln -s alertmanager-0.25.0.linux-amd64 alertmanager
$ cd alertmanager
```

- (0: Test Case) 가동중인 Node Exporter를 중지하면, 프로메테우스에서는 해당 Node Exporter가 수집되지 않으므로, 수집상태가 Down으로 표현된다. 프로메테우스의 Targets에서 확인해 볼 수도 있지만, 수식 입력창에서 up == 0 를 조회해 본다. (일반적으로 이런 단순 Bool수식으로 Expression을 사용하지는 않지만, 예제로서…)
  - 테스트는 Prometheus가 이 조건에 해당되는 경우를 감지하고, 이후 Alertmanager가 적절한 Alert처리를 하도록 구성하는 것으로 테스트를 진행할 것이다.



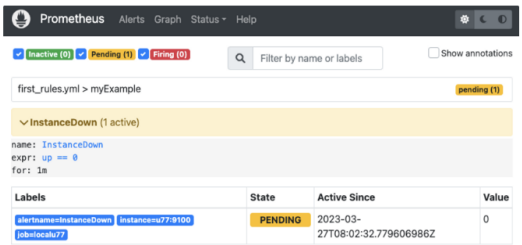
- (1) 이제 앞의 점검(확인) 수식을 프로메테우스의 알림규칙에 추가한다. 또한, 어떤 알림 매니저(Alert Manager)와 통신할 것인지 알려주어여 한다. prometheus.yml에 아래와 같은 해당 내용을 추가 한다. (해당 예제는 예시를 위한 간단한 내용임을 감안한다.)
  - (1-1) 우선은 prometheus.yml에 사용할 rule파일(`rule_files:`)과 사용할 alert manager(`alerting:`)정보를 추가한다. alertmanager는 다른 port를 사용하는 다른 process로 정리되고 있음에 주의 한다.

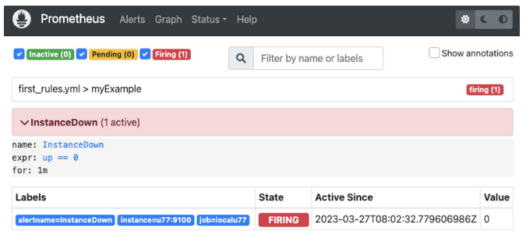
```
alerting:
  alertmanagers:
    - static_configs:
        - targets:
          - "localhost:9093"
rule_files:
  - "myFirstRule.yml"
```

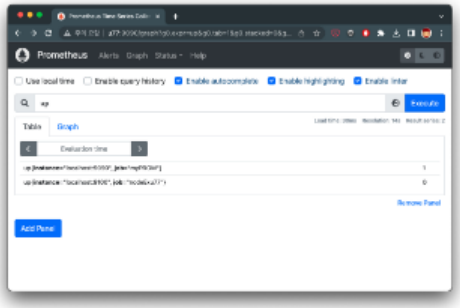
  - (1-2) 사용할 rule이 저장된 `myFirstRule.yml` 파일을 아래와 같이 만들어 준다. 아래 내용은 global로 설정된 evaluation_interval 시간 마다 확인은 하지만, 아래에 for로 지정된 시간동안은 해당 상태에 대해 pending하고 있다가 지정된 시간이 지나면, 해당 Alert를 발생시킨다.

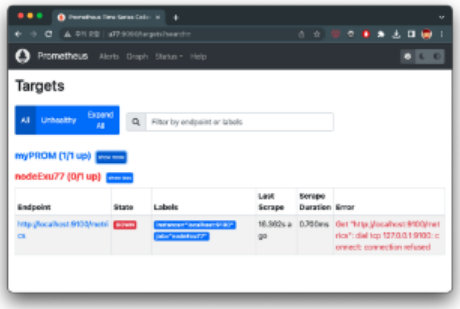
```
groups:
 - name: myRuleExample
   rules:
    - alert: InstanceDown
      expr: up == 0
      for: 1m
```

  - (1-3) Prometheus는 변경 내용 반영을 위해 재기동하고, 앞의 Node Exporter는 테스트를 위해 정지시킨다. Prometheus가 감시 대상중, Node Exporter의 정지(NOT up)된 것을 감지하도록 하려는 목적이다. 이제 Alert의 메뉴를 보면, 잠시 동안은 `"Pending"`된 매시지와 해당 내용이 있지만, 1분 후에는 `"Firing"`으로 해당 Alert가 표현되고 있다.





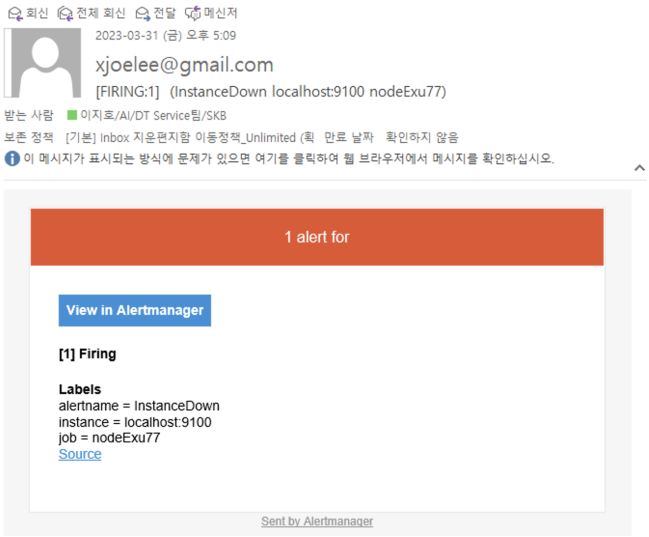




- (2) 이제 Rule에 의해 알림(Alert)이 반응할 수 있는 예제 상태를 앞에서 만들었으며, 이것을 Alert Manager를 이용해 다른 무엇인가, 다른 Action을 동작하도록 해볼 수 있다.
  - (2-1) 예제로 Alertmanager가 앞의 Event를 email을 사용하여 alert message로 누군가에게 전달하게 하고자 한다면, alertmanager.yml에 해당 내용을 설정하고, Alertmanager를 기동한다.

```
route:
  receiver: 'myLocalEmail'
receivers:
  - name: 'myLocalEmail'
    email_configs:
      - to: 'xjoelee@sk.com'
        from: 'xjoelee@gmail.com'
        smarthost: smtp.gmail.com:587
        auth_username: 'xjoelee@gmail.com'
        auth_identity: 'xjoelee@gmail.com'
        auth_password: 'GOOGLE_APPS_PASSWORD'
```

    - 위의 내용은 Alertmanager가 Gmail의 특정 사용자에 대해 가동중인 Linux(Ubuntu)에서 Smarthost방식으로 Gmail Agent로 Mail Sending을 할 수 있도록 하는 설정이다. eMail관련된 더 많은 방법의 eMail 구성방식과 방법들 그리고, 또는 다른 방식의 Alert에 대해서는 나중에...
    - 설정 내용중, `GOOGLE_APPS_PASSWORD`는 구글의 서비스를 사용자가 직접 사용하는 것이 아닌, App에 의해서 사용하고 싶을때, Google에 등록하여 만들어진 Password 대체용 Key값이다.
  - (2-2) Prometheus에서 Node exporter가 감시되지 않고, Down으로 간주되는 상태가 확인되면, Prometheus는 Alertmanager에게 정의된 Alert작업을 의뢰하고, Alertmanager는 정의된 작업대로 해당 건에 대한 Alert처리를 하게 된다. 위에서 정의된 대로 아래와 같은 메일이 도착할 것이다.



- : Google에서 App용 Password만들기

+==