



Ch06. Dashboard by Grafana

Grafana는 InfluxDB, Graphite 같은 TSDB 또는 ELK, PostgreSQL 같은 DB들 또는 다른 시스템을 사용하는 Dashboard를 제작할 수 있는 도구이다.

- (참고) 프로메테우스의 TSDB도 유명하지만, 프로메테우스는 이제 계측 툴로서의 의미나 성격이 더 크다. TSDB에도 여러 종류가 있다. Time Series DB로 유명한 것들: InfluxDB, Kdb+, Graphite, TimescaleDB, Apache Druid 들이 있다.

원래 프로메테우스에는 PromDash가 있지만, 프로메테우스 개발자들도 Dashboard는 Grafana를 사용하기로 결정했다. 프로메테우스는 Grafana의 최우선 등급의 Plugin으로 구성되어 있다.

- 프로메테우스에는 Console Template가 내장되어 있다. Grafana, PromDash는 Dashboard 자체에 대한 정보를 RDB에 저장하는데, 내장된 ConsoleTemplate는 내부 Filesystem에 저장된다.
- 내장 ConsoleTemplate는 Go언어로 작성되고, 특수한 예외적 사용이나, 고급 사용자용으로 사용 되는 편이다.



도서의 예제가 시간이 좀 되어, 현재 최신의 Grafana와 맞지 않는 부분이 좀 있으나, 의미적인 내용으로 찾아 보면, 더 나아진 방법으로 Grafana 내부에서 필요한 내용들은 모두 존재하는 것에 주의 한다.

6.1. Install Grafana

<http://grafana.com/grafana/download>를 참고하여, 필요한 내용을 다운로드하거나, 다른 설치 방법들을 참고할 수 있다.

- 설치 패키지 Download 받아 설치 하기: Grafana를 패키지로 풀어서, /bin/grafana를 실행시키면 Daemon으로 구동되고, 3000 port에서 HTTP Listen하고 있다.

```
$ wget https://dl.grafana.com/enterprise/release/grafana-enterprise-9.4.7.linux-amd64.tar.gz  
$ tar -zvxf grafana-enterprise-9.4.7.linux-amd64.tar.gz
```

- Ubuntu에서 apt로 설치 해 보기

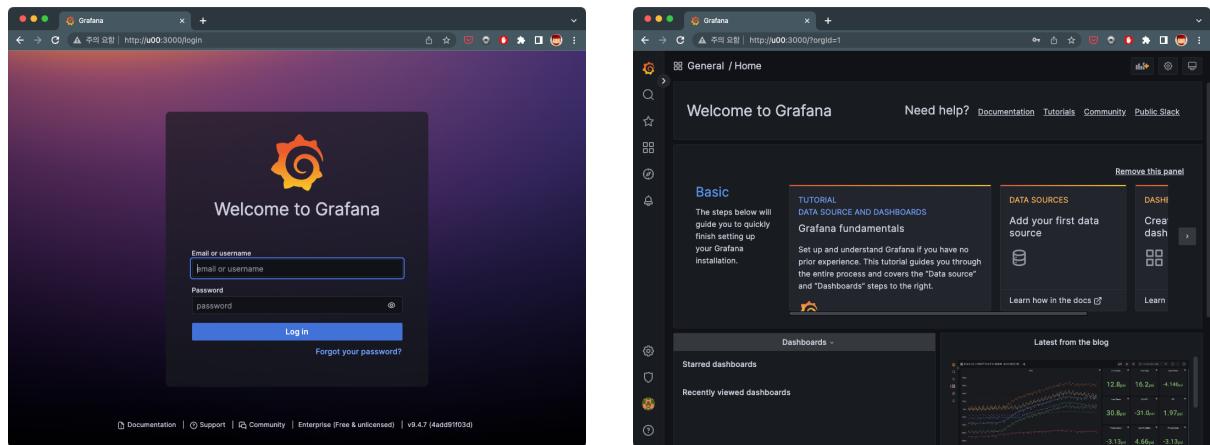
```
$ sudo apt-get install -y adduser libfontconfig1  
$ wget https://dl.grafana.com/enterprise/release/grafana-enterprise_9.4.7_amd64.deb  
$ sudo dpkg -i grafana-enterprise_9.4.7_amd64.deb
```

- 책에서는 5.0.0을 기준으로 진행하였으나, 현재 최신은 9.5.7이다. 또한, 최근에 Grafana는 Cloud로 서비스를 제공하고 있기도 하다.
- Docker를 사용하는 경우, 설치와 사용에 아래를 참고한다. Docker이므로 Grafana는 Docker 내부에만 데이터가 저장되는 것에 주의 한다.

```
$ docker run -d --name=mygrafana --net=host grafana/grafana:5.0.0
```

- (참고) Grafana Configuration Options: [가이드 페이지](#)
 - Grafana를 설치하면, 보통 grafana를 설치한 디렉토리의 ~/conf에 내용들이 위치한다.
 - 이 구성 파일에서 기본 관리자 암호, http 포트, grafana 데이터베이스(sqlite3, mysql, postgres), 인증 옵션 (google, github, ldap, auth 프록시)과 같은 항목을 다른 많은 옵션과 함께 변경할 수 있습니다.
 - 그라파나 서버를 시작하고, 관리 사용자(기본 admin/admin)로 로그인합니다. 사이드 메뉴 열기(상단 메뉴에서 Grafana 아이콘 클릭) 데이터 소스로 이동하여 데이터 소스를 추가합니다.

처음 접속해 보면 다음과 같은 화면이 보이고, admin/admin을 ID와 Password로 입력하면 첫 화면을 만나게 된다.

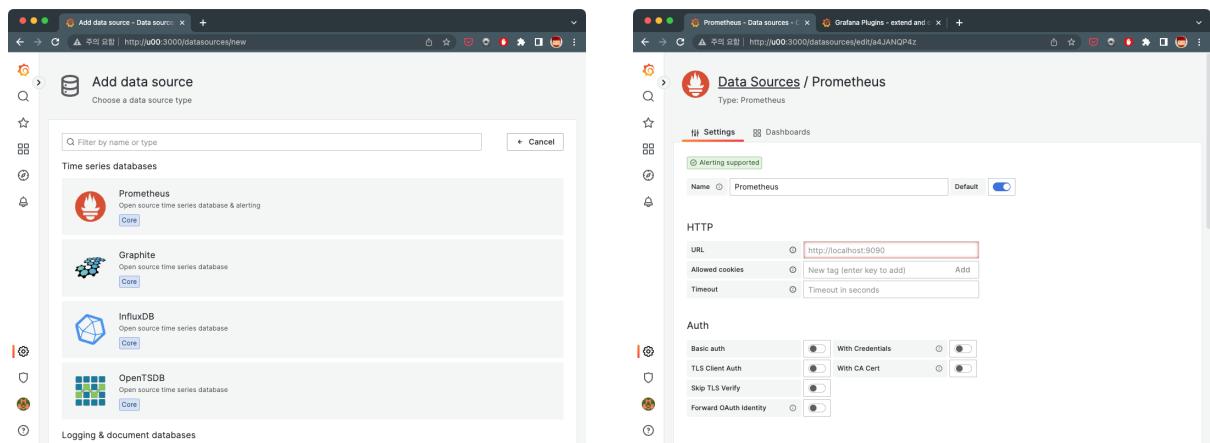


6.2. Datasource

Grafana는 그래프를 그릴 정보를 추출할 대상 Data Source가 있어야 하며, Data Source로는 프로메테우스 이외에도 OpenTSDB, PostgreSQL 같은 다양한 유형의 Data Source를 추가 설치나 설정 없이, 선택하기만 하면, 사용할 수 있다.

- Grafana는 보통 1개의 Data Source를 사용하지만, 여러 Data Source를 혼합하여 사용할 수도 있다.

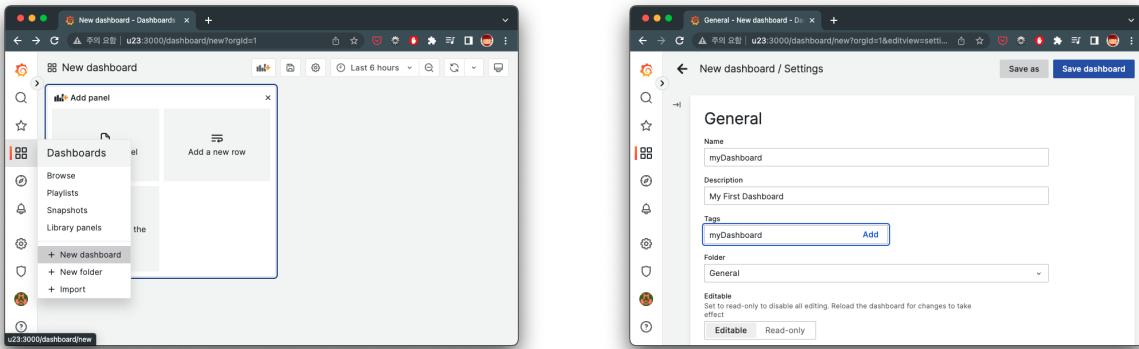
첫번째 기본적인 설정 단계는 Data Source를 우리가 사용하고 있는 local의 프로메테우스인 localhost:9090을 아래의 그림을 참고하여 등록한다. 기본적으로 프로메테우스 관련되어, 사용할 Name, 접근 HTTP URL, Type & Version을 지정하고, “Save&Test”를 선택하여 저장하고 테스트 한다.



- Grafana에 준비된 기본적인 Data Source들은 정말 많은 것 같다. 아래 기본 나열된 것 외에도 Plugin 설치를 통해 더 많은 것들이 준비되어 있다. (158+)
 - TSDB: Prometheus, [Graphite](#), InfluxDB, OpenTSDB, kdb+
 - Logging & Document DB: Loki, [Elasticsearch](#)
 - Distributed Tracing: Jaeger, Tempo, Zipkin
 - SQL(RDB): [MySQL](#), [PostgreSQL](#), MS SQL Server
 - Cloud: MS Azure Monitor, [AWS Cloud Watch](#), Google Cloud Monitoring, Grafana Cloud, [Amazon Athena](#)
 - Enterprise Plugins: AppDynamics, Azure Devops, DataDog, Dynatrace, GitLab, Honeycomb, Jira, [MongoDB](#), NewRelic, [Oracle DB](#), SAP HANA, Salesforce, ServiceNow, [Snowflake](#), Splunk, Wavefront
 - Others: [Alertmanager](#), [TestData](#), [Ambari Metrics](#), [Kafka](#), MQTT

6.3. Dashboard and Panel

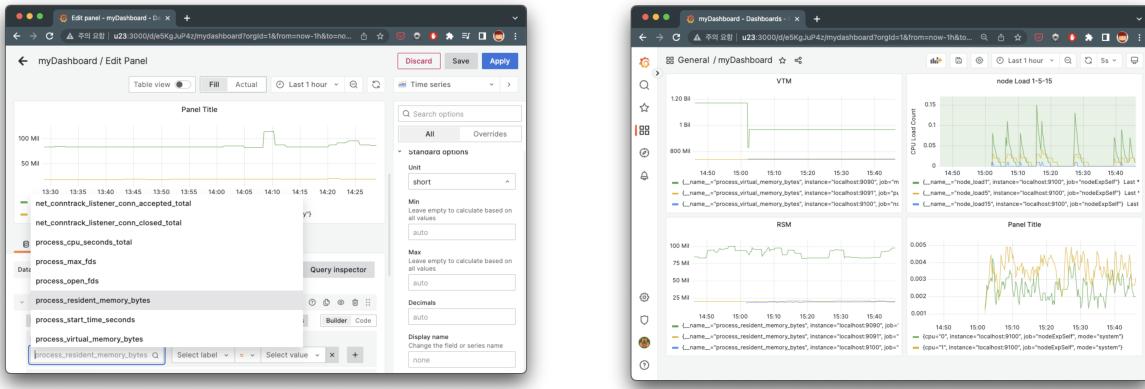
두번째 기본적인 설정 단계로는 앞서 등록된 Data Source를 사용하여, Panel을 구성하는 것이다. 다음과 같이, “Add Dashboard”를 선택하여 Panel을 포함할 수 있는 Dashboard를 생성하고, 필요시 해당 Dashboard의 Property들을 수정 할 수 있다.



- Dashboard는 보통, 관심 있는 서비스, 팀의 업무, 감시 목적으로 맞도록 구성하는 것이 원칙이며, 빠르고 정확한 상황의 판단을 위한 필요한 내용들로만 적절한 양으로 구성하는 것이 권장된다. Dashboard가 많거나 크다고 좋은 것이 아니란 이야기이다.
- 참고로 위 화면의 전체가 Dashboard 하나이며, 처음 등록할 Panel을 추가하기 위한 준비화면이 표현된 것임에 주의 한다.

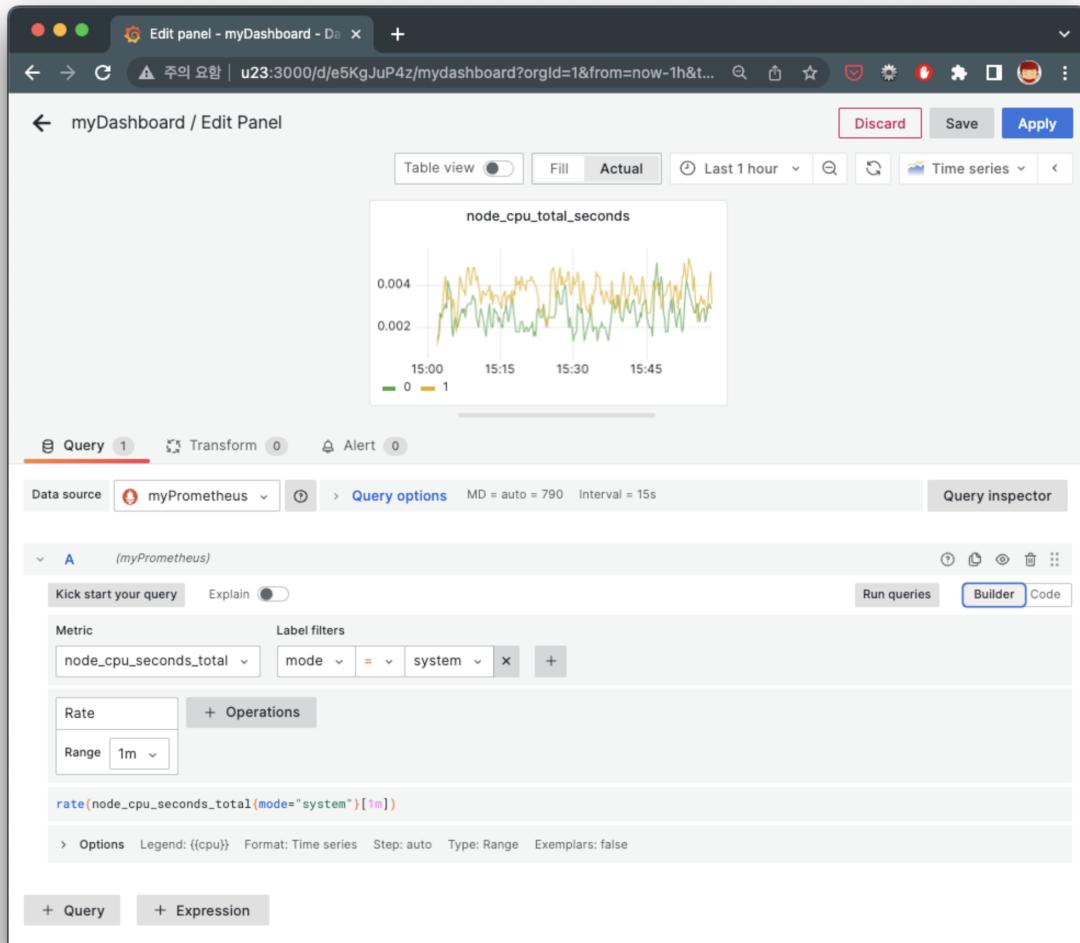
6.4. Graph Panel

세번째 기본적인 설정 단계로는 Dashboard에 Panel을 추가해 보는 것이다. 아래의 예시처럼, 브라우저 내부 좌측에서는 추가하고자 하는 Panel에서 표현 하고자 하는 Metric(Label, 조건등 포함)(에 대한 설정을 추가 할 수 있고, 우측에서는 Panel에서 보여지는 Look Feel에 대한 설정을 조절할 수 있다.



- 좌측의 예시에서는 `process_resident_memory_bytes` 를 등록하는 과정이고, 우측의 예시는 `process_virtual_memory_bytes`, `node_cpu_seconds_total`, `node_load` 를 등록한 예제이다.

Grafana Panel은 프로메테우스의 Expression(수식 브라우저)보다 더 많은 기능을 제공한다. 우측 아래쪽의 Panel에 등록한 `node_cpu_seconds_total`의 경우, Query 설정에서 “Builder”에 의한 설정이 아닌, “Code”기반 입력 모드로 설정하고, `rate(node_cpu_seconds_total{mode="system"}[1m])` 를 입력하여 설정하였다.



- 화면상에 있는 모든 버튼과 조건들을 조작하여 익숙해 져야 한다. 오른쪽에 보이던 부분은 Panel에서 보여지는 데이터에 대한 부분인 “Data Option”이 아닌, 데이터의 표시 형태, 즉, “Style Option”에 대한 부분이므로 숨겨 두었다.
 - 우측 상단의 Panel을 표시하는 형식에 대한 버튼들
 - 하단 “Query”탭에서 보여질 그래프에 대한 PromQL기반의 Query들 (여러개를 추가할 수 있다.)
 - 하단 Transform탭에서 보여질 Source Data에 대한 변환 내역들
 - 하단 “Alert”탭에서 해당 Panel을 기준으로 Alert를 처리하는 규칙들

(opt) Transformation at Panel

Transformation 기능을 사용하면 시각화단계를 마무리 하기 전에 쿼리 결과를 결합, 계산, 재정렬, 숨기기 및 이름 바꾸기 할 수 있습니다. 현재 그래프 시각화(Graph Type)를 사용하고 있다면 시계열 데이터(time series data)만 지원하므로 많은 Transformation 기능이 적절한 것이 아닐 수 있습니다. 그래서, Transformation 기능이 수행하는 작업을 이해하려면, 테이블 시각화(Table Type)로 전환해 놓고 살펴보는 것이 도움이 될 수 있습니다.

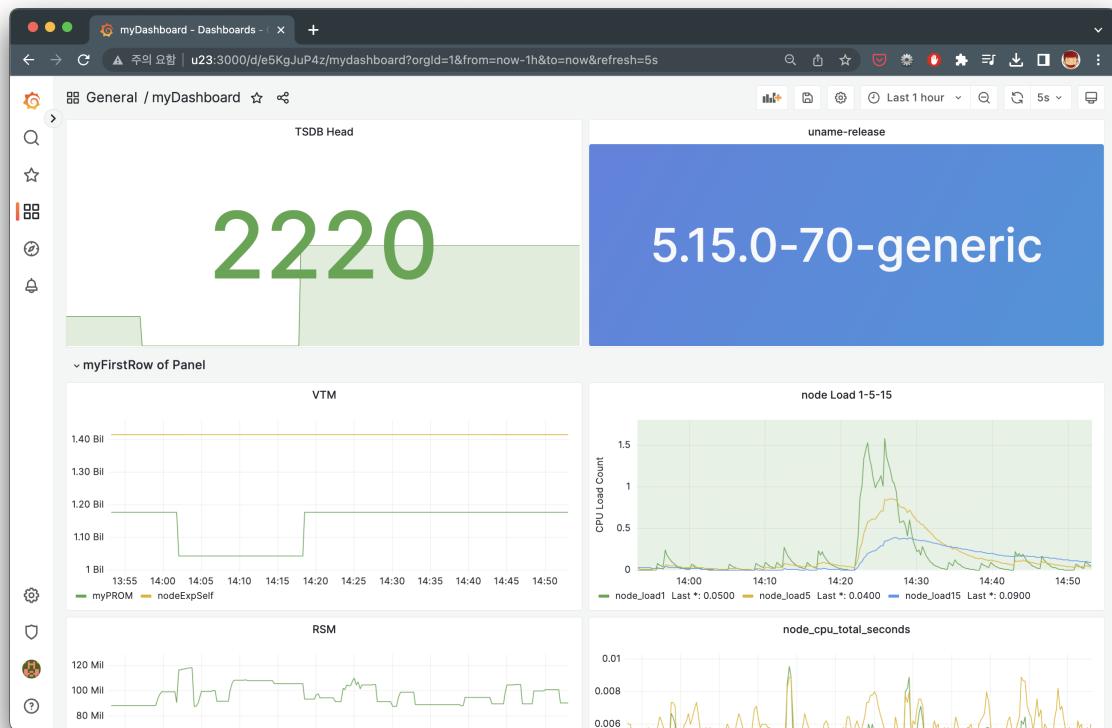
동일한 데이터 세트에 기반하지만, 여러 뷰에 의존하는, 즉 여러 뷰를 사용하고자 하는 사용자를 위해, Transformation은 다양한 대시 보드를 생성하고 유지하는 효율적인 방법을 제공합니다. 또한, 하나의 Transformation된 출력을 다른 Transformation의 입력으로 사용할 수도 있으며, 이는 성능 향상을 가져올 수 있습니다.

시스템이 데이터 변환을 그래프로 표시할 수 없는 경우가 있습니다. 그럴 때는 시각화 위에 있는 **Table View** 툴을 클릭하여 데이터의 테이블 뷰로 전환하세요. 이렇게 하면 변환의 최종 결과를 이해하는 데 도움이 됩니다.

Data를 Transformation으로 잘 다루는 것은 TBD ^^

6.5. Single-State Panel

Single-State Panel은 시계열 데이터에서 하나의 값만을 표시한다. 아래 처럼 하나의 값 (또는 하나의 값과 그를 보조하기 위한 것들)



- (좌측 상단 예) Panel을 하나 추가하고, 표시할 Metric을 `prometheus_tsdb_head_series`를 지정하고, 표시 형식을 (기본 형인) "Time-Series" type에서 "State" Type으로 바꾸어 본다.

- Single-State 값은 기본적으로 지정된 시간범위 시계열에서의 Last(Not Null)값이다. 평균값을 표시하고 싶다면, 우측의 "Style Option>Value Option>Calculation"에서 "Mean"으로 설정하면 평균값을 표시하게 된다. (평균 값은 그 특성상, 가끔 우리가 원하는 정확하게 기대한 값이 아닐 수 있음에 주의한다.)

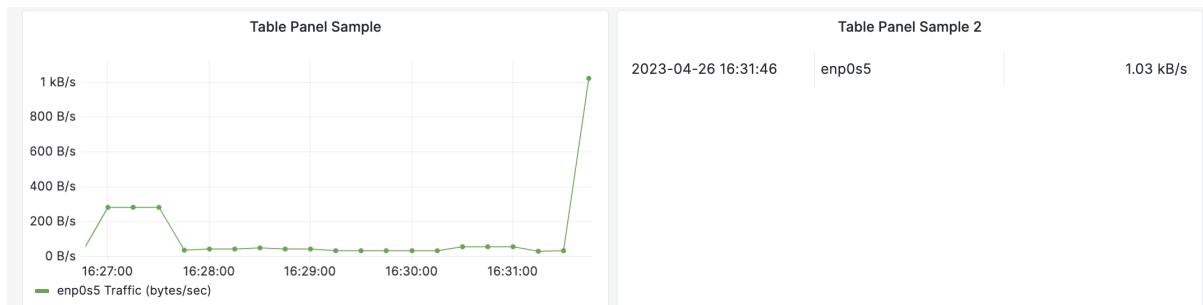


참고로 우측 Style Option의 Calculation에서 상당히 많은 종류의 계산에 따른 표현방식을 선택할 수 있다. 기본값은 Last이다: Last, First, Min, Max, Mean, Variance, Standard Deviation, Total, Count, Range, Delta, Diffrence, Distinct, All Zero, All Null등

- 우측의 "Style Option>Stat Styles>Graph Mode"를 선택하면, 저장소에 기록된 값을 Graph 형태로 background에 보여줄 수도 있다.
- (우측 상단 예) Panel을 하나 추가하고, 표시할 Metric을 `node_uname_info`를 지정하고, 표시 형식을 "State" Type으로 바꾸어 본다.
 - 해당 Metric은 "`_info`"이므로, 우리가 사용할 값은 측정된 값이 아니라, `_info`에 포함되어 있는 Label의 값이다. 따라서, 하단의 "Query Option"에서 사용할 Label 이름 "release"를 측(Legend)에 "`{release}`"로 지정한다.
 - 우측의 "Style Option>Value Options>Fields"에서 사용할 Field 이름인 Label `release`에 기록된 내용을 선택하고, "Style Option>Stat Styles>Text Mode"에서 `Name`을 지정한다.
 - 우측의 "Style Option>Stat Styles>Graph Mode"는 `None`으로 지우고, "Style Option>Stat Styles>Color Mode"는 `Background`로 하면, 지정된 Color가 Background에 적용된다.

6.6. Table Panel

Table Panel이 어떤 내용을 보여줄 수 있는지는 상상이 될 것이다. 아래처럼 Panel을 표현하기 위한 설정을 진행해 본다. 아래 2개의 Panel은 Node Exporter의 `rate(node_network_receive_bytes_total[1m])`를 다르게 표현한 동일한 Metric이다. (참고로, 해당 Node Exporter에 준비된 Network Interface가 "enp0s5" 1개만 존재하여, 1개만 표현되었다.)

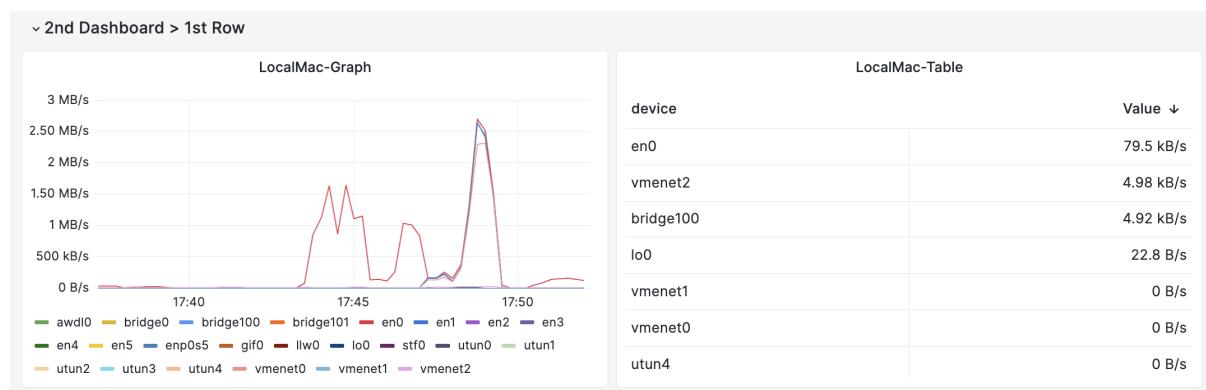


- (좌측의 Time Series형) Panel을 하나 추가하고, `rate(node_network_receive_bytes_total[1m])`을 표시할 Metric으로 지정하고, Panel의 기본 표시형식은 기본값인 "Time Series"로 한다.
 - Query Option에서 Format은 "Time Series", Type은 "Range"로 지정한다.
 - Style Option > Standard Options > Unit에서 "bytes/sec(SI)"를 입력/선택한다.
- (우측의 Table형) 좌측과 같은 내용의 Panel을 하나 더 추가하고, Panel의 기본 표시형식은 "Table"로 지정한 다음, 아래의 몇몇 설정을 추가해 본다.
 - Query Option에서 Format은 "Table", Type은 "Instant"로 지정한다.
 - Style Option > Table > Show table header를 Off한다.

- Style Option > Standard Options > Unit에서 “bytes/sec(SI)”를 입력/선택한다.
- Style Option 맨 아래에서 “`+Add Field Override`”를 선택하면, 새로운 Override조건이 생긴다. 여기에서 Override 할 “Field Name” 중에서 “Instance” Label을 선택하고, 그 아래의 “`+Add override Property`”를 선택하면, Override 방식을 선택할 수 있는데, 여기에서 “Hide in Table”을 선택한다. 그러면, Panel에서 해당 Field가 보여지지 않게 된다.
- 동일한 방법으로 “`+Add Field Override`”를 선택하여, 이번에는 “Field Name” 중에서 “Job” Label이 보여지지 않도록 설정 한다.

(opt) 다른 OS에서 Node Exporter를 설치, 가동해 본다.

Network Interface가 1개 밖에 없어, Local의 macOS에 Node Exporter를 설치하고, Parallels VM에 있는 프로메테우스에서 해당 Node Exporter를 감시해 보았다. 설정방법은 앞의 내용과 동일하게 진행하였다. macOS의 많은 NIC가 보인다.



맥에서 macOS를 위한 Node Exporter는 간단히 설치 가능하며, 아래와 같이 Background로 실행 시켜 주면 된다.

```
$ brew install node_exporter  
$ node_exporter &
```

참고로 Ubuntu의 Node Exporter(Prometheus제공)도 몇몇 문제가 있다. Ubuntu에서 제공하는 Node Exporter를 사용하는 것이 나을 듯 하다.

```
$ sudo apt install prometheus-node-exporter  
$ sudo systemctl enable prometheus-node-exporter  
$ sudo systemctl start prometheus-node-exporter
```

6.7. Variables

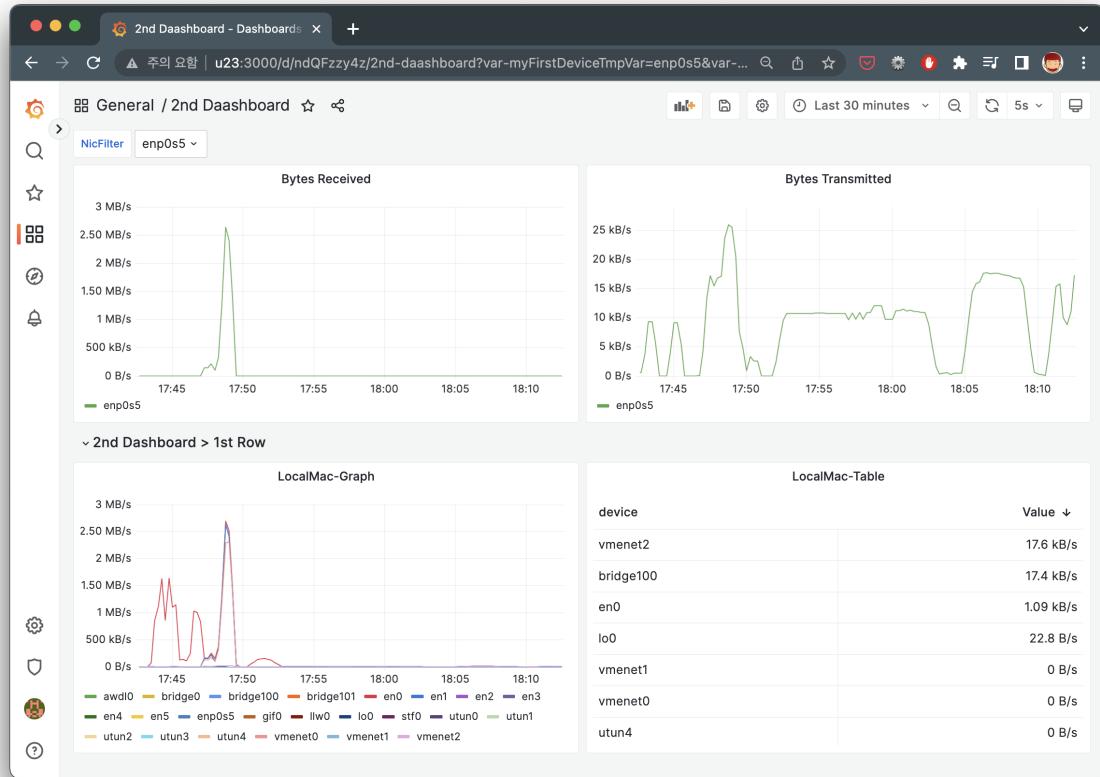
이제까지 살펴본 예제들은 단일 프로메테우스와 단일 Node Exporter를 사용하는 형태의 내용이었다. 수백개의 Node Exporter를 감시하고자 할 때, 일일히 이렇게 설정하는 것은 무리이므로, Grafana의 Template Variables기능을 사용한다.

보통의 경우, Variables 방식은 Node(Instance)를 선택할 때, 주로 많이 사용하지만, 아래 예제에서는 Node(Instance) 대신에 NIC를 선택하는 방법을 예제로 들어 본다. 상단에 보면, “NicFilter”라는 직접 만들어 준 Variable이 선택사항으로 나타나 있으며, Dropdown으로 보고 싶은 NIC를 선택할 수 있게 되어 있다. 이런 방식으로 (Template) Variable을 선택하면, 해당 선택된 내용이 Panel을 구성하는 Metric Query에 반영하여, 해당 결과만 볼 수 있도록 하는 것이다.



Variables 참고사항:

- Template Variable이 적용되는 Scope는 해당 Dashboard에 속한다.
- Variables라는 이름은 과거에 Template, Template Variables로 불리워 졌었다. 살펴보면 알겠지만, Variable 수준에서 Variable처럼 사용하므로 이름이 바뀐것으로 보인다. Template라 하면, 최소한 Panel 개념 수준에서의 Template 방식으로 처리되는 경우를 말할 것이다.



(1) Variable 만들기

처음 할 일은 사용할 Template Variable을 만들어 보는 것이다. Dashboard에서 우측 상단의 아이콘을 선택하면 해당 Dashboard에 관련된 많은 설정을 진행 할 수 있다. 여기서 우리가 사용하는 것은 Dashboard Setting > “Variables” 메뉴에서 “”를 선택하여 Variable을 추가한다. 아래와 같이 설정하면, 화면의 아래쪽 “Preview of values”에서 선택 가능한 대상 Metric의 구별 할 수 있는 값들이 나열된다.

The 'Variables' configuration screen shows two tabs:

- General Tab:**
 - Name: `NicFilter`
 - Label: `NicFilter`
 - Description: `mySecondTempVar`
 - Show on dashboard: `Label and value`
 - Query options:
 - Data source: `myPrometheus`
 - Query: `node_network_receive_bytes_total`
- Query Tab:**
 - User: `node_network_receive_bytes_total`
 - Regex: `Optional, if you want to extract part of a series name or metric node segment. Individual group can be used to separate the display name and value like example: ^deviceName[.]*(?!)^$`
 - Sort: `Optional, if you want to sort the values of this variable`
 - Refresh: `Optional, how often to update the values of this variable`
 - Selection options:
 - Multi-value: Enables multiple values to be selected at the same time
 - Help All option: Enables an option to include all variables
 - Preview of values: Shows a list of interface names: `awd0 bridge0 bridge100 bridge101 en0 en1 en2 en3 en5 enp0s5 gif0 llw0 lo0 stf0 utn0 utn1 utn2 utn3 utn4 vmenet0 vmenet1 vmenet2`

- General > Name: `NicFilter` (이름을 잘 정한다. 이 이름은 Panel 설정시 참조될 Variable의 이름이다.)
- General > Label: 프로메테우스의 Label이 아니다. 말그대로 Display Name으로 사용할 Label이다.
- Query options > Data Source: `프로메테우스` (현재 사용중인 프로메테우스를 지정한다.)
- Query options > Query: `node_network_receive_bytes_total` (대상 Metric 지정)

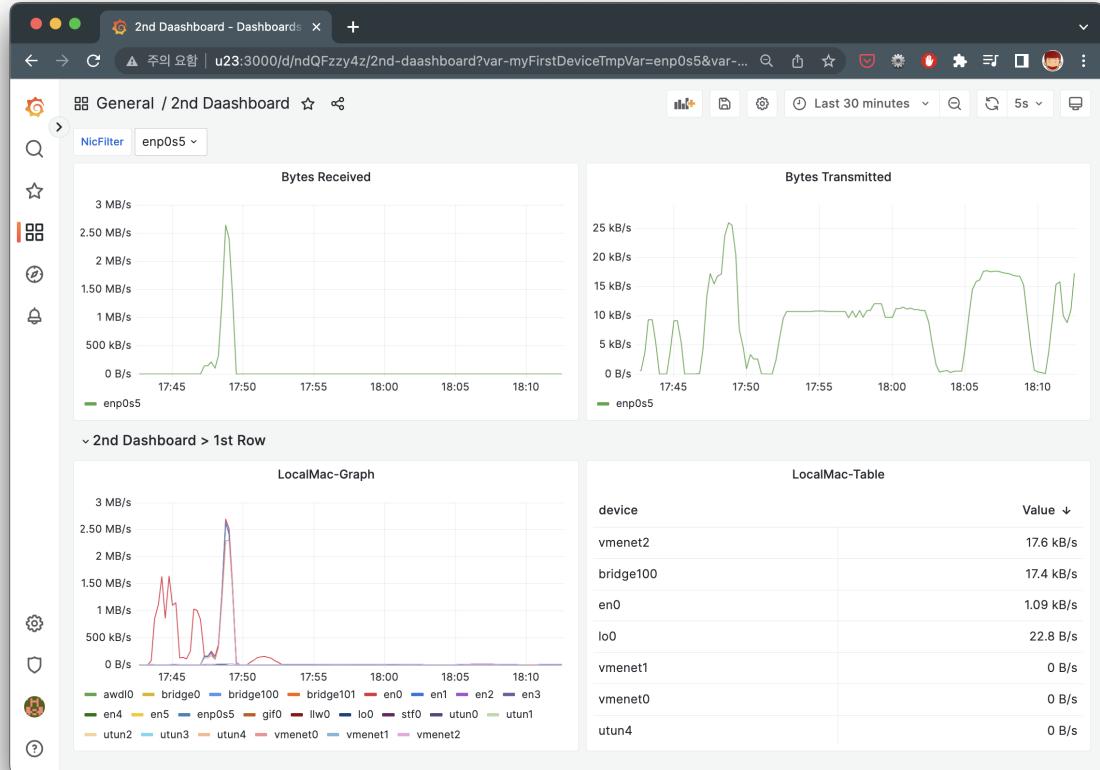
- Query options > Regex: `.device="(.?)".*` (Regex방식으로 Metric에서 추출할 Variable 지정)
- Query options > Refresh: `On time range change`
- 마지막으로 “Apply” 및 “Save Dashboard”를 선택하여 저장한다.
- Dashboard로 돌아와 보면, 상단에 지정된 이름의 Variables가 있을 것이다.

(2) Variable을 사용하는 Panel만들기

다음 할 일은 Dashboard로 돌아와서 Panel을 추가하는 것이다. Graph Panel 2개를 만들어 볼 것이며, 하나는 `node_network_receive_bytes_total` 을 다른 하나는 `node_network_transmit_bytes_total` 을 보여 주도록 한다. 각각의 Panel 설정에서 Query 표현식을 `rate(node_network_receive_bytes_total{device=\"$NicFilter\"}[1m])` 으로 설정한다. 여기서 `NicFilter` 는 앞에서 만든 Variable의 이름이다.

먼저 만들어진 Panel을 복사하여, `node_network_transmit_bytes_total` 내용만 수정하면 간단히 Graph Panel하나를 더 추가 할 수 있다.

이제 만들어진 Panel 2개에 대해, Dashboard상단의 Variable을 선택하는 것으로, Graph Panel에 보여지는 내용을 선택 할 수 있다.



Variables로 가능한 것들

다시 말하지만, 예제에서 NIC를 선택한 것과는 다르게, 보통은 Variable 방식은 Node(Instance)를 선택할 때, 주로 많이 사용한다 는것에 주의 한다.

경우에 따라, 하나의 Dashboard에서 여러개의 Variable을 사용할 수 있다. 예를 들어 java의 gc관련 내용을 감시하고자 하는 Dashboard를 구성하는 경우, 다음과 같이 여러개의 Variables를 구성하여, 입체적으로 사용할 수 도 있을 것이다.

- e.g. job Label기준(Daemon App 이름), instance Label기준 (실행중인 Node 이름) 등

그 외 몇가지들

- 참고로 우측상단에 시간에 대한 부분이 있다. 보통은 Dashboard전체에 적용되지만, 필요시 재정의를 통해 각 Panel 별로 다르게 할 수도 있다.
- 상단에 Share 아이콘이 있는데, 현재의 Dashboard를 다른 누군가에게 공유할 때 사용할 수도 있고, 현 상태의 snapshot을 저장할 수도 있으며, 현재의 Dashboard를 JSON등으로 Export하여, Dashboard 구성 내역을 Share할 수도 있다.



Aliasing

그래프가 갱신될 때, 데이터가 변경되지 않았는데도, 그래프의 모양이 변경되는 경우는 Data의 Sampling과 Data처리간의 차이에 의해서 발생한다. 수집빈도와 평가(처리)빈도를 높여 Aliasing현상을 완화 할 수는 있지만, Sampling기법의 태생적 한계임을 감안하여야 한다. (어차피 Metric에 나타난 숫자도 Sampling되었을 수 있다.) 더 정확한 Event에 대한 기록이 필요하면 Log방식을 사용하라.



Prometheus 전체를 간단히 둘러 보고...

Exporter들에서 얻어 낼 수 있는 시계열 정보들의 명확한 의미와 PromQL로 보고자 하는 정보를 잘 추출 하는 것이 프로메테우스를 사용하는 입장에서의 핵심이다. 책의 나머지 부분도 보면, 계측을 포함한, “Exporter”영역의 상세 내용, “PromQL”관련 상세내용, AlertManager사용관련내용, 그리고 마지막으로 프로메테우스 유지보수 관련 내용이지만, 결국 Label을 포함한 PromQL영역이 가장 중요한 부분이다.

한편으로는 Grafana입장에서 보면, 다양한 Data Source의 활용, 특히 프로메테우스 Data에 대한 직관적 표현을 통한 감시 방법의 다양성이 프로메테우스로 데이터를 잘 수집하는 것 보다, 운영감시의 더 중요한 핵심일 수도 있다. Grafana 관련되어서는, 자세한 지침 대신에 Grafana의 팬찮은 Dashboard와 Panel을 공유한 사람들이 많으므로 이 정보를 참고하여 재사용 하는 것도 팬찮으리라 본다.

+==