Ch08. Service Discovery

감시해야될 실행중인 모든 머신(또는 감시 대상인 node, app 들)에 대한 정보를 프로메테우스에 설정하는 방밥에 대해 살펴본다. 즉, 모니터링 대상이 무엇인지 알고, 어떤 정보를 수집할 수 있는지 파악하는 것이 프로메테우스 환경 구성이 첫번째일 것이다.

앞서 `prometheus.yml`에서 처럼, 모니터링 대상들을 직접 Static하게 정보를 저장하는 방식은 더 많은 감시 대상들이 있는 실제 환경에서도 확대해서 사용하는 것은 무리이다. 대신, 전통적인 Discovery(=Provisioning) 개념의 방법을 사용한다.

Discovery를 무작정 하는 것은 아니고, 이를 쉽게하기 위해서, 감시대상들이 있는 정보를 저장한 곳에서 그 정보를 얻는다. 보통 해당 정보가 저장된 일반 RDB, Consul, AWS(EC2), k8s 같은 서비스 소스(이미 provision정보를 가지고 있는)와 프로메테우스는 연동이 가능하므로 연동하여 정보를 얻는다. 또한, 해당 정보가 파일에 기록되어 있는 경우도 가능하다. (파일로 연동하는 방법은 Ansible, Chef등에 있는 provision정보를 가져올때 사용 한다.)

(별개이지만, 같은 맥락에서) Discovery된 모니터링 대상들에게도 Label을 사용할 수 있다. 이를 통해, 대상들을 이해하고, 관리하기 쉽도록 그룹화 하여 취급할 수 있다. 이러한 감시대상을 설정하고, 취급하기 위한 Label은 감시 대상인 Application이나 Exporter에서 구성하는 것이 아니고, 프로메테우스에서 구성하는 것이며, 이렇기 때문에 프로메테우스 Instance별로 각자 사용하는 용도(팀,프로메테우스 Instance)에 따라 별도의 Label 구성을 가질 수 있다.

- e.g. HW팀은 Rack, PDU, Server, OS에 관심이 있을 것이고, DB팀은 DB Instance들에 대해, 그리고, 특정 서비스 팀은 해당 서비스에만 관심이 있을 것이다.

8.1. Service Discovery Mechanism

“Service Dricovery(서비스검색) System”은 일반적인 Database이외에도 프로메테우스가 지원하는 서비스 검색 연동 가능 서비스들은 다음과 같다. (이 이외에도 많을 듯)

- Azure, AWS EC2, GCE, Consul, DNS, Openstack, k8s, Marathon, Nerve, Serverset, Triton, File 등

“Service Dricovery System”은 단순히 머신, 서비스의 목록만을 찾아내어 구성하는 것이 아니다. 대상 시스템 전반의 내용, Application 간의 의존성, Instance들의 운영상태 및 유지보수 상태, 운영 및 소유팀, 구조화된 태그 등의 Metadata를 확보하는 것을 포함한다.

- 만약 서비스 구성 관리정보나 서비스 정보 데이터베이스가 정형화 되지 않았거나, 정리가 되지 않았다면, Consul을 사용해 보기를 권장

Service Discovery List & Targets List

감시할 대상에 대한 정보를 얻어 내는 것을 Service Discovery라고 한다. 이 과정에서 다른 DB 또는 다른 구성관리도구를 정보를 얻어내는 소스로하여, 여기에 의존하여 얻어지는 Service Discovery의 결과물을 Targets List로 보자.

만약 FSD의 경우 라면, 해당 FSD에 정의되어 있는 List는 “Service Discovery List”이고, 이를 기반으로, 프로메테우스관점에서의 감사 대상인 “Targets List”를 얻는다. 이 과정에서 FSD는 그 Source 가 될 것이다.

Discovered Labels & Target Labels

감시대상 쪽, 즉 Exporter측에서는 Service Discovery로 얻어진 Metadata 정보들을 Label로 정리하게 되며 (=Discovered Labels), 이것을 기반으로 프로메테우스는 자신이 취급하는 정보들을 자신의 관점내에서 Label로 정리할 수 있다. (=Target Labels), 또한 프로메테우스는 단순히 Label을 가져오는 것이 아니라, 소스의 Label을 대체, 선택, Override등의 조작을 거쳐 받아 들일 수 있다.

Static Service Discovery

우리가 앞의 예제에서 보아온 `prometheus.yml`에 직접 필요한 내용을 설정하는 방법이다. 작은 규모의 서비스에 적합하며, 프로메테우스 자신, Pushgateway, Alertmanager등의 설정은 Static으로 구성하게 된다. 이때 DIscovery하여야 할 Metadata들이 Static하게 정의하는 방식이다.

만약, Ansible 같은 “구성관리도구”를 사용한다면, 아래와 같은 설정으로 `prometheus.yml`에 템플릿시스템(inline script방식)을 적용하여, Node Exporter가 있는 모든 머신의 목록을 생성할 수도 있다.

```
scrape_configs:
  - job_name: node
    static_configs:
    - targets:
{% for host in group["all"] %}
      - {{ host }}:9100
{% endfor %}
```

- 위의 예제는 여러개의 모니터링 대상 host들을 `scrape_configs:`에 “node”라는 하나의 `job_name`에 등록하고 있으며, `static_configs:` 아래에는 Inline script로 Target을 정의하고 있으며, 필요하다면 각 대상들을 위한 Label 필드들을 추가로 정의할 수도 있다.
- 다수의 `static_configs:`가 있다면, 이것은 하나의 리스트 형태로 인식하고 취급하며, 여러가지 방법으로 static한 구성정보를 정의해도 하나의 정의로 구성 할 수 있다. 이렇게 정의하는 것이 허용되는 이유는 여러 검색 서비스와 연동되어, 거기에서 소스를 얻어 대상을 정의해도, 하나의 `job_name`으로 구성할 수도 있도록 할 수있기 때문이다. (복잡성 문제는 당연히…)
- 하여간, `scrape_configs:`는 원하는 만큼 여러개의 수집 대상 구성을 하나의 리스트, 즉 하나의 `job_name`으로 반영할 수 있으며, 한가지 중요한 사실은 `scrape_configs`에서 존재하는 `job_name`은 unique해야 된다.

(유치하지만) 동일한 내용이지만, 이런 설정 구성도 가능하다.

```
- job_name: SimpleMetricExpositionWeb
    static_configs:
      - targets:
        - localhost:8000
      - targets:
        - localhost:8000
```

```
- job_name: SimpleMetricExpositionWeb
    static_configs:
      - targets:
        - 127.0.0.1:8000
      - targets:
        - localhost:8000
```

File based Service Discovery(FSD)

File based Service Discovery(FSD)를 사용하는 경우는 보통은 프로메테우스가 지원하지 않는 “Service Discovery System”과 통합에 사용하거나, 주어진 Metadata만으로는 부족하여, 추가로 설정이 필요할 경우, 또는 Static으로 정의할 때도 활용할 수 있다.

일반적으로 FSD방식에 의한 SD 방법은 지속적으로 활용하려는 목적과 형식이 강하므로, 보통은 Label 정의는 직접적으로 정의하여 사용하는 것이 일반적이며, 재지정(Conditional Filtering)하기 위한 메타데이터 형태로는 정의하지 않는 것이 일반적이다. (FSD를 사용하는 경우, 보통은 감시 대상에 대한 정리와 정의가 명시적인 경우에 많이 사용된다는 의미로…)

파일의 형식은 JSON, YAML 형태를 사용한다. 간단하게 실제 프로메테우스 환경에서 적용해 본다.

- 아래의 JSON으로 작성된 FSD파일 예제를 확인한다. job:fsd_node1에는 team:infra Label과 함께 2개의 node가 정의되어 있고, job:fsd_node2에는 team:monitoring과 함께 1개의 node가 정의되어 있다.

```
$ more 8-4-filesd.json
[
  {
    "targets": [ "host1:9100", "host2:9100" ],
    "labels": {
      "team": "infra",
      "job": "fsd_node1"
    }
  },
  {
    "targets": [ "host1:9090" ],
    "labels": {
      "team": "monitoring",
      "job": "fsd_node2"
    }
  }
]
$
```

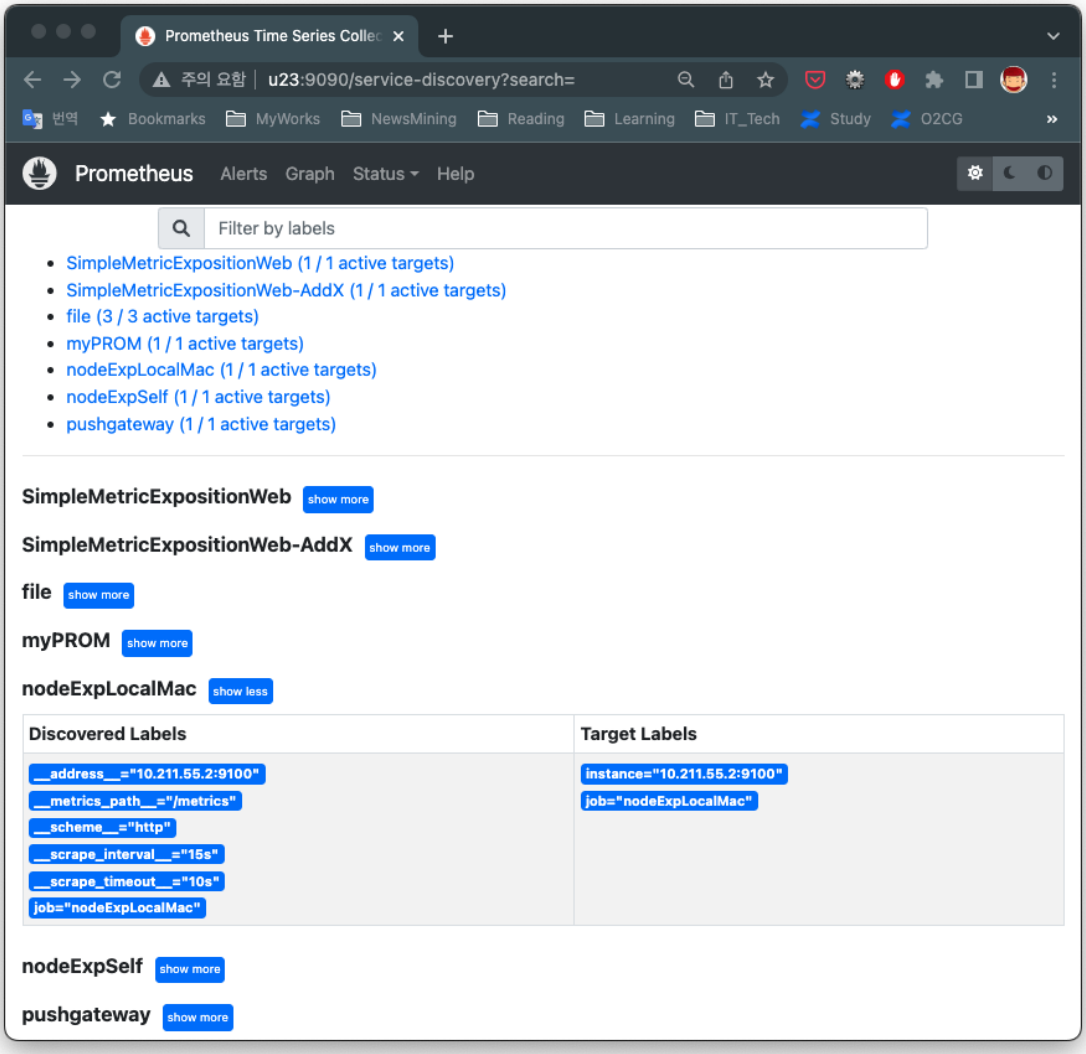
- prometheus.yml파일에 해당 FSD파일을 사용하도록 설정을 변경한다. 지시자는 `file_sd_config:`를 사용한다. 서비스검색구성에 사용할 파일명을 지정하게 되며, 파일명은 파일이 여러개 있는 경우에 대해 glob pattern으로 지정할 수 있다.

```
...
scrape_configs:
 - job_name: file
   file_sd_configs:
    - files:
       - '*.json'
...
```

  - 참고로 prometheus.yml에서는 FSD 파일을 지정하면서, `job_name`을 “file”로 하였고, json파일에서는 “`job`”이라는 key를 사용한 것에 주의 한다. 이 이름들은 이렇게만 사용해야 한다. (뒤에서 override에서 또 이야기 된다.)
    - prometheus.yml에 정의된 `job_name`의 내용은 실제 프로메테우스에서 “`job`” Label로 처리된다. (그렇다. `job_name`은 Discovered Label이고, `job`은 Target Label이 되는 것이다.)

- 위의 내용을 기준으로 프로메테우스를 다시 기동해 보면 다음과 같은 내용을 확인 할 수 있다. `host1:9100`, `host2:9100`에서 infra팀의 `fsd_node1` 각 1개씩 2개와 `host1:9090`에서 monitoring팀의 `fsd_node2` 1개, 총 3개의 instance가 추가된 것이 확인될 것이다. 그렇다. job_name은 그 자체로 unique 아 아닌, Label 단위의 unique이다.

- 이제 해당 프로메테우스 화면에서 `/service-discovery`로 접근해 본다. 현재 discovery하고 있는 내용을 확인 할 수 있다. 참고로 앞서 말한것 처럼, “Dicovered Labels”와 “Target Labels”를 잘 살펴보자.



  - 이후의 설명에서 더 이야기가 되겠지만, “Service Discovery”에 의해 검색(조사)된 결과를 살펴보면, “Discovered Labels”와 “Target Labels”의 2가지로 나누어진 Label들을 볼 수 있다.

- “Discoverd Labels”에 static으로 정의한 `job_name`이 “Target Label”에서는 `job`으로 정의되어 있다.
- 부가적으로 살펴보면, “Discoverd Labels”에는 우리가 직접 정의하지 않은 “__”로 시작되어 표현된 Label들이 있으며, “Target Label”에는 정의하지 않은 `instance`라는 Label이 추가되어 있다. 뒤에서 더 이야기 된다.

Consul (TBD)

(Added) What is Consul?

(Definition) Consul(콘술)은 런타임 플랫폼과 public 또는 private 클라우드에 걸쳐서 구성된 서비스 간 연결, 보안 및 구성에 대한 Distributed Service Mesh입니다.

- Consule는 서비스 디스커버리(Service discovery)와 설정을 관리하는 툴이다. Consule는 분산&클라우드 환경에 적응하기 위한 고가용성, 유연한 스케일링, 분산시스템의 특징을 가진다.
  - Service Discovery : DNS 나 HTTP 인터페이스를 통해서 서비스를 찾을 수 있게 한다. 외부의 SaaS 서비스업체도 등록할 수 있다.
  - Health Checking : 클러스터의 건강 상태를 모니터링하며 문제가 생길 경우 신속하게 전파한다. 헬스체크는 서비스 디스커버리와 함께 작동하며, 문제가 생긴 호스트로 서비스 요청이 흐르는 걸 막는다. 이를 이용해서 서비스레벨에서의 서킷 브레이커(circuit breaker)를 구현 할 수 있다.
  - KV(Key/Value) 저장소(Store) : Consul은 계층적으로 구성 할 수 있는 KV 저장소를 제공한다. 애플리케이션은 설정, 플래그, 리더 선출 등 다양한 목적을 위해서 이 저장소를 사용 할 수 있다. 이 저장소는 HTTP API를 이용해서 간단하게 사용 할 수 있다.
  - 멀티 데이터센터 대응 : 데이터센터 규모에서 사용 할 수 있으며, 복잡한 구성없이 여러 리전(region)을 지원 할 수 있다.
  - Security : Consule Connect는 TLS를 이용 서비스와 서비스사이에 안전한 통신이 가능하게 한다.

Consul이 현실의 SD를 수행하고 관리하다면, 프로메테우스는 Consul의 SD 정보를 사용하여, 자신의 SD정보로 사용할 수 있다. 이에 대한 연결과 사용에 대한 간단한 내용이다.

EC2

AWS의 EC2는 프로메테우스가 추가 설정등의 작업이 없이 AWS에 존재하는 EC2에 대한 Service Discovery를 할 수 있다. EC2의 정보를 Service Discovery를 위해 접근하여 사용하려면, EC2 API를 사용하기 위해 프로메테우스는 AWS의 Credential(자격증명)을 가지고 있고, 그것을 제공하여야지만 가능하다.

이렇게 하기 위해서는 AWS IAM에서 `AmazonEC2ReadOnlyAcess` Role(명확하게는 `Describe Instance`)을 가진 IAM 사용자로 설정하고, 여기에서 얻어지는 `AccessKey`와 `SecretKey`를 프로메테우스의 구성파일에 포함시켜, 프롬메테우스가 사용하도록 하여야 한다.

- EC2 서비스 검색을 사용하기 위한 `prometheus.yml` 구성 예시

```
...
scrape_configs:
 - job_name: ec2
   ec2_sd_configs:
    - region: <region>
      access_key: <access key>
      secret_key: <secret key>
...
```

- 해당 Region에서 가동중인 EC2가 한 개도 없다면, 하나 실행 시켜 본다. 프로메테우스의 `/service_discovery`에서 실행된 EC2를 확인 할 수 있을 것이다.
- 검색된 EC2는 EC2의 특성상 해당 VPC에서의 사설 IP를 가지고 있을 것이다. 외부에 노출되는 서비스용 EP IP는 별도인 것에 주의 한다.
- 다른 대부분의 Cloud서비스 제공자들도 유사한 방식이다.

8.2. Relabeling of Label

Consul, EC2 같은 것들에 대한 SF로 얻어지는 대상과 대상에 대한 메타데이터들은 좀 원시적인 날것이다. 다심 말해, 앞의 예에서 보았듯이, SD로 얻어지는 List에는 보통 SD측에서 관리하는 Label형태로 나름 메타데이터들을 포함하고 있지만, 프로메테우스에서 직접적으로 사용하기에 적합하지 않다. (FSD의 경우는, 앞서 이야기 한 것처럼, Target List를 위해 이미 정확하게 정리가 되어 있고, Targe Label로 정리되어 있을 것이다.)

이상적인 환경과 지향점은 새로운 Machine, Application들을 자동으로 SD를 진행하고 이것들을 모니터링 대상으로 Target LIst와 Target Label로 정리하고 등록 되는 과정에서, 사전의 정의를 통해 SD과정 이후, Relaeling으로 원하는 대상과 조건에만 Label을 지정하여 Target Label로 사용하게 할 수 있다.

수집할 대상 선택하기 (feat. regex)

가장 먼저 구성해야 하는 내용은 어떤 대상에 대한 정보를 실제 수집할 지 결정하는 것이다. EC2가 있는 Region의 모든 EC2를 대상으로 데이터를 수집하는 것은 아닐 것이다. 필요한 대상만 선태해야하는 것이다.

- 아래 예에서 처럼, `source_labels:`에 나열된 Labels값에 `regex`에 지정한 식과 일치하는 경우, 이후의 모니터링 `action`을 계속한다는 의미로 정의되어 있다. 결국 team=”infra”를 만족하는 대상만 걸러 낸다.

```
scrape_configs:
  - job_name: file
    file_sd_configs:
      - files:
        - '*.json'
    relabel_configs:
      - source_labels: [team]
        regex: infra
        action: keep
```

- `relabel_configs:`에는 여러개의 relabeling(`source_labels:`로 지정하는)을 정의할 수 있다. 아래와 같이 여러개의 정의는 relaeling은 순차적으로 적용(Filtering)하게 된다.
  - (그러나) 참고로 아래의 예는 team label을 확인하고 있는데, 앞의 정의에서 “infra”와 “monitoring”을 동시에 갖고 있는 대상이 없으므로 결과적으로 해당 되는 대상이 없는 경우가 된다.

```
scrape_configs:
  - job_name: file
    file_sd_configs:
      - files:
        - '*.json'
    relabel_configs:
      - source_labels: [team]
        regex: infra
        action: keep
      - source_labels: [team]
        regex: monitoring
        action: keep
```

- 하나의 Label에 대해, 여러개의 값이 해당되는 경우를 표현하려면, 예제의 경우, “infra”, “monitoring”이 논리식 OR로 구성되려면 regex의 |(pipe)를 사용한다.

```
scrape_configs:
  - job_name: file
    file_sd_configs:
      - files:
        - '*.json'
    relabel_configs:
      - source_labels: [team]
        regex: infra|monitoring
        action: keep
```

- 또 따른 예제로 `source_labels:`에 여러개의 label을 대상으로 하고, 각각의 regex를 적용하고자 하는 경우, 다음과 같이 할 수 있다. `source_labels:`에서는 ,(comma)로 구분하고, regex에서는 ;(semicolon)으로 구분한다. 아래 예의 의미는 job=”prometheus” AND team=”monitoring”에 대해 수집을 하지 않는다는 의미이다.

```
scrape_configs:
  - job_name: file
    file_sd_configs:
      - files:
        - '*.json'
    relabel_configs:
      - source_labels: [job, team]
        regex: prometheus;monitoring
        action: drop
```

프로메테우스에서 대상을 설정할때, vip 또는 load-balancer의 ip나 service ep를 지정하지 않는 것이 권장된다. 명확하게 수집할 Instance를 지정하여야 한다.
(그런데) 대상 자체가 Dynamic Instance이거나, 고정되지 않는 Instance이라면??? 아마도 내부적으로 해당 Instance를 지정해 낼 수 있는 개별 ep를 써야할 것 같은데…?

Target Label에 대한 고려사항

Target Label이란, 앞서서도 이미 설명되었지만, FSD처럼 Static으로 List에 지정되거나, Service Discovered된 List에 지정되어 있거나, 추가적으로는 `source_label:`로 Relabeling하여 정리된 List들에 대해 지정되어 있는 Label들이 수집되고, 이것들이 프로메테우스의 TSDB에 Label을 가진 List로 저장되며, 이 영역에서 취급되는 Label을 “Target Label”이라 한다.

Metric 이름과 마찬가지로 “Target Label”은 시간이 지나거나 상황에 따라 변경되어서는 안된다. 그러면 좋은 Target Label은 어떤 것인가? 절대적인 법칙은 아니지만, Target Label을 구성하고 사용할때의 고려사항 이나 권고사항 정도는 다음과 같다.

- 모든 대상(Target)들을 포함하지만, 의미적으로 구분하여 부여할 수 있는 Label Name을 고려한다.
  e.g. 개발단계, 생산단계, 지역, 건물, 관리주체팀, 큰의미의 서비스명 등
- 해당 대상이 가지는 구별되는, 또는 구분하여 의미가 있는 특징을 추가하는 것도 고려 할 수 있다.
  e.g. Master, Sharding, Replica, WebFront, WebFrontJS 등
- 기본 Label인 “instance”는 기본적으로 해당 host에 대한 거의 unique한(Cardinality가 높은)값일 텐데, 추가로 고유성이 있는 Label을 추가한다면, PromQL에서 Label을 집합연산으로 취급할때, 문제가 생길 수도 있다.

e.g. `without(instance)` 같은 집합연산에서…

- 경험상, 독립적인 고유성을 가지도록 Label들을 Layered 되게 구성하는 것이다.
  e.g. Instance를 포함하는 소작업, 소작업을 포함하는 대작업, 작업들을 포함하는 서비스, 서비스를 포함하는 환경(Center) 등
- 그외에 경우와 필요에 따라, `info` Metric이나, `external_labels:` 등을 추가적으로 사용하는 것을 고려해 볼 수 있다.

Replace for Relabeling

예를 들어, Metric과 수집쪽에서 “team” Label이 “monitoring”에서 “monitor”로 바뀌었다면 (FSD에서도 바꿀 수 없다면), 해당 Label에 대해 프로메테우스 측에서 replace로 relabel을 할 수 있다.

- 아래는 team=”monitoring”을 team=”monitor”로 변경하는 예시이다.

```
scrape_configs:
  - job_name: file
    file_sd_configs:
      - files:
        - '*.json'
    relabel_configs:
      - source_labels: [team]
        regex: monitoring
        replacement: monitor
        target_label: team
        action: replace
```

- 아래는 team Label에 대해, 마지막에 “ing”가 있는 모든 것을 replace하는 예시이다.

```
scrape_configs:
  - job_name: file
    file_sd_configs:
      - files:
        - '*.json'
    relabel_configs:
      - source_labels: [team]
        regex: '(.*)ing'
        replacement: '${1}'
        target_label: team
        action: replace
```

- 아래는 team Label를 삭제하도록 하는 replace하는 예시이다.

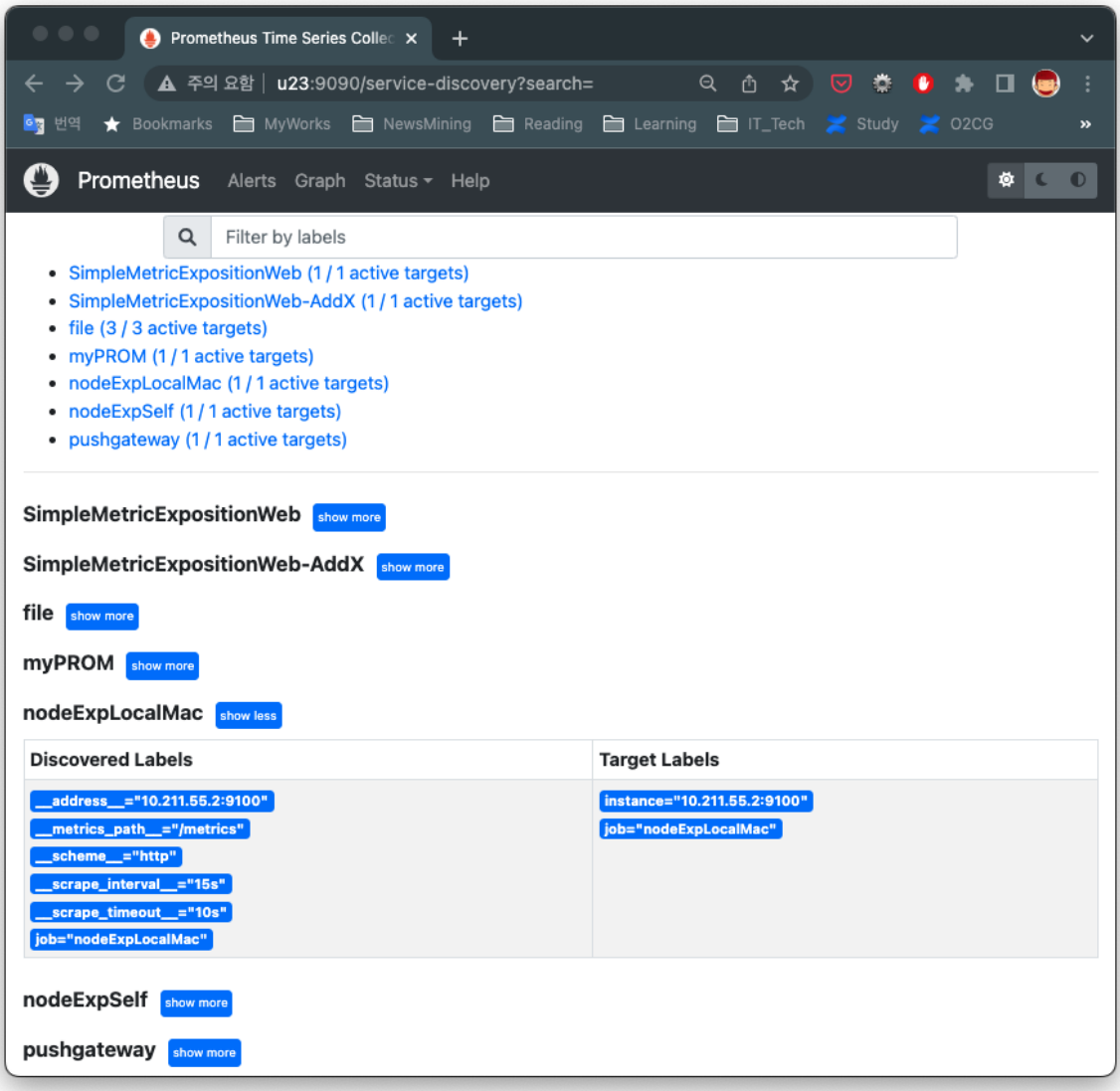
```
scrape_configs:
  - job_name: file
    file_sd_configs:
      - files:
        - '*.json'
    relabel_configs:
      - source_labels: []
        regex: '(.*)'
        replacement: '${1}'
        target_label: team
        action: replace
```

- 다른 예시로 Consul을 Service Discovery서비스로 사용하고 있는데, 모니터 대상을 지정한 ip:port 를 특정 port로 바꾸는 경우이다. 기존의 `localhost:8500` 을 `localhost:9100` 으로 replace하고 있다.

```
scrape_configs:
  - job_name: node
    consul_sd_configs:
      - server: 'localhost:8500'
    relabel_configs:
      - source_labels: [__meta_consul_address__]
        regex: '(.*)'
        replacement: '${1}:9100'
        target_label: __address__
```

Label: job, instance, __address__

이제까지 진행하면서 Label들 중에 몇몇 특수한 경우가 숨어 있음을 알 수 있다. 이것들에 대한 의미와 취급예를 살펴보자. (`job`, `instance`, `__address__`)



- `job` Label
  - `job` Label은 항상 존재하여야 하는 Label이며, Discovered Labels에 보여지는 `job`은 구성 옵션에서 Service Discovery를 구성한 설정정보에서 `job_name`에 정의된 내용이다.

    - `job` Label은 동일한 목적으로 동작하는 instance(Runnable Process or Task at OS)들의 집합을 나타내며, 일반적으로 동일한 Binary(Runnable program at OS)와 구성으로 동작하는 것으로 간주 한다.

- `instance` Label
    - 위의 Service-Discovery내용을 살펴 보면, `instance`에 대한 Target Labels는 있지만, 이것은 Discoverd Labels의 `__address__`를 사용한 것이다. (Discovered Labels에 `instance`가 명시적으로 정의되어 있다면, Target Lebels의 `instance`는 이것을 사용하게 된다.)
    - `instance` Label은 job내의 하나의 instance를 표시하는 것이다.
- 아래의 예제는 Consul을 Service Discovery소스로 사용하고, 여기서 얻어지는 IP 정보(__meta_consul_address__)에 Port 9100을 추가하여, `__address__` Target Label로 사용하고, Consul SD에서 얻어지는 node이름(__meta_consul_node__)을 `instance` Target Label로 replace하여 사용하는 예제이다.

```
scrape_configs:
  - job_name: consul
    consul_sd_configs:
      - server: 'localhost:8500'
    relabel_configs:
      - source_labels: [__meta_consul_address__]
        regex: '(.*)'
        replacement: '${1}:9100'
        target_label: __address__
      - source_labels: [__meta_consul_node__]
        regex: '(.*)'
        replacement: '${1}:9100'
        target_label: instance
```

    - 참고로, 프로메테우스에서는 ip:port형식 보다는 가독성이 더 높은 DNS Reverse Resolving된 hostname:port형식을 더 선호한다.
    - 위의 예제는 consul에서 SD로 확인되는 address, nodename이 여러개일 경우, 그것들을 모두, 프로메테우스의 scrape대상으로 등록(label은 relabel을 통해 각각의 이름이 가져가며) 하도록 하는 설정이다.

- (**labelmap**) 이것은 지금까지 보아온 drop, keep, replace 동작과는 다르며, Label의 값이 아닌, Label이름자체에 대한 처리에 대해 정의 할 수 있다.
    - 프로메테우스가 시스템 전체 아키텍쳐내에서 Metadata에 대해 독립적이거나, 유일한 사용처가 아니므로, 맹목적으로 Label을 복사 및 재활용하여 사용하는 것은 권장되지 않는다. 경우에 따라 엉뚱한 내용이 수집되거나, 구성된 그래프와 알람이 망가질 수도 있다.

- (**list**) 대부분의 SD(Service Discovery)의 메커니즘은 기본적으로 Key-Value 형태이다. 그러나, 일부는 Tag List만을 제공하기도 하는데, 이 경우, List를 Key-Value형태의 Metadata로 변경하여 사용할 때 사용한다. (별로 사용 안할 듯)
    - Target Label을 지정하고, Filter(feat. regex)된 Tag들을 그 값으로 사용하는 경우 등에 사용

8.3. Scrape Configuration

이제 까지 Target Label과 대표적인 Replace에 의한 Relabeling에 대한 내용을 보았으며, `__address__` 같은 수집대상도 있음을 알게 되었다.

좀 더 살펴벌 것은 /metric 이외의 경로를 사용하도록 하거나, Client Auth를 구성하는 등의 추가적인 수집(Scrape)에 대한 설정에 대해 알아 본다.

- 이러한 설정 옵션, 즉 다양한 설정 방법들은 변경 되는 경향이 있으므로, 가장 최신의 내용을 항상 확인하라. (Promethus Docs의 Configuration Section)

Example of scrap configuration

Target 정보를 얻어내어 구성하는 방법에 대한 설정, 즉 scrape에 대한 설정에는 앞서 이야기한 내용이외에 추가적인 부분들이 있으며, 그것들의 특징과 조건을 확인해 보자.

- 아래는 다양한 옵션을 가지는 `scrape_configs:`의 구성 예시이다.

```
scrape_configs:
 - job_name: example
   consul_sd_configs:
    - server: 'localhost:8500'
   scrape_timeout: 5s
   metrics_path: /admin/metrics
   params:
     foo: [bar]
   scheme: https
   tls_config:
     insecure_skip_verify: true
   basic_auth:
     username: brian
     password: hunter2
```

- URL scheme & TLS Authentication
  - scheme은 선택할 수 있는 옵션으로 `http`, `https`를 설정할 수 있으며, 부가적으로 https에서의 TLS인증을 사용하려면, `key_file`, `cert_file` 같은 추가적인 옵션으로 정보를 제공할 수 있어야 한다.
  - TLS 인증서의 유효성 검사를 비활성화 하려면, `insecure_skip_verify`를 사용할 수도 있다.
  - HTTP bearer Authentication을 사용하는 경우에는 `basic_auth`, `bearer_token`을 통해 제공되어 사용할 수 있으며, bearer 방식의 인증내용들은 프로메테우스 페이지들에서는 감추어 진다.
- `metrics_path` & (URL) `params`
  - matrics_path는 유일하게 URL 경로 형식으로 작성되며, /metrics?foo=bar 같은 경우는 /metrics%3Ffoo=bar 로 이스케이프 처리되고, 모든 parameter들은 `param`에 두어야 한다.
  - `params`는 SNMP Exporter, Exporter Class(as Blackbox Exporter), Exporter Federation에서만 필요하다. (일반적인 Exporter들에서는 URL에 `params`를 포함하여 검색을 하는 일이 거의 없다.)
  - 앞의 단순한 URL 설정형태 이상의 유연함이 필요하면, `proxy_url`을 사용하여 프록시 서버를 설정하여, 수집을 요청하는 방법도 있다. 개별적으로 임의의 Header를 추가 하는 등의 방법은 불가능하다.
- scrape interval overriding
  - `scrape_timeout`, `scrape_interval`을 override하여 설정할 수는 있지만, 가능하면, 단일한 일관된 수집주기를 가지는 설정값을 사용하는 것이 좋다.
- URL scheme, Path, URL parameter는 각각 Label Name으로 표현된 `__scheme__`, `__metrics_path__`, `__param_<name>`으로 각각 그 값들을 사용할 수 있으며, 필요시에 Relabeling을 통해 override하여 사용할 수있다.
  - 동일한 이름의 Label이 여러개 있다면, 첫 번째 만 유효하다. 또한, 보안을 이유로 앞의 3가지 이름 이외의 다른 설정에 대한 Relabeling은 허용되지 않는다.
- Service Discovery의 Metadata는 보안에 민감한 부분으로 취급되지 않고, 접근 가능한 누구나 그 내용을 확인 할 수 있다. 만약 보안이 필요한 내용이 있다면, 이것은 개별 수집 구성별로 지정할 수 있으며, 이때 필요한 Authentication Credential은 전체(모든) 서비스에 대해 표준화된 방법으로 사용되게 하는 것이 바람직 하다.

metrics_relabel_configs:

> 💡 “Relabel”은 SD Metadata를 Target Label로 매핑하는 본연의 목적으로 사용하는 것인데, 이 목적 이외에도 프로메테우스의 다양한 다른 영역에서도 “Relabel”이 적용된다.
>
> 어떤 Label이 어디에 적용되는 것인지 정리해 보면, `relabel_configs:` 는 수집 대상이 무엇인지 알아내어 정리하려는 경우에 사용되고, `metrics_relabel_configs:` 는 데이터 수집 이후에 사용된다.

Metrics 에 대한 relabel의 목적은 다음과 같은 두 가지 경우에 주로 사용한다. 이러한 Metrics relabel은 시계열상의 정보를 수집한 후 저장공간에 write하기 전에 접근하여 적용된다.

- 많은 비용이 소요되는 Metric을 삭제하는 경우
- 잘못된 Metric을 수정하거나 Override하고 싶은 Metric이 있는 경우

metrics_relabel_configs사용되는 기본적인 action은 `keep`, `drop`이 보통 사용된다.

- 추가로 metrics_relabel_configs의 설정을 통해 Metric Name, Label Name을 replace하거나, 해당 Name으로 부터 또 다른 Label Name을 추출하는 경우에도 사용될 수 있다.
- 또한, `labeldrop`, `labelkeep`을 action으로 label Name자체를 조작하는 경우에 사용 될 수도 있다. (잘 생각해보면, `labelkeep`은 사용하지 않는 것이 좋겠다.)

Label Collision과 honor_labels

앞의 `labeldrop` 은 Exporter가 Label을 잘못 알고 있거나 취급할때, 그것을 버리는 용도로 사용할 수 있다. 그러나 더 필요한 경우는 pushgateway이다. pushgateway에는 `instance` label이 없어야 한다. (Pushgateway로 Push되는 Metric은 `instance` label이 없고, 프로메테우스에서 관심있는 instance는 Pushgateway의 instance가 아니기 때문이다.)

Instrument(계측)된 Label을 우선하여, 그것으로 Target Label을 Override하기 원한다면, `scrape_configs:`에서 `honor_labels: true`를 설정한다. 이런 경우 만약 계측되어 수집된 Metirc이 명시적으로 instance=””인 Label이 있다면, 최종 결과에서 instance label은 존재하지 않는다.

metrics_relabel_config에서 Intrumentation label과 target label을 세밀하게 제어하도록 설정하여 조정할 수 있지만, `honor_labels`처리와 label collision은 그 전에 먼저 처리된다.

+==