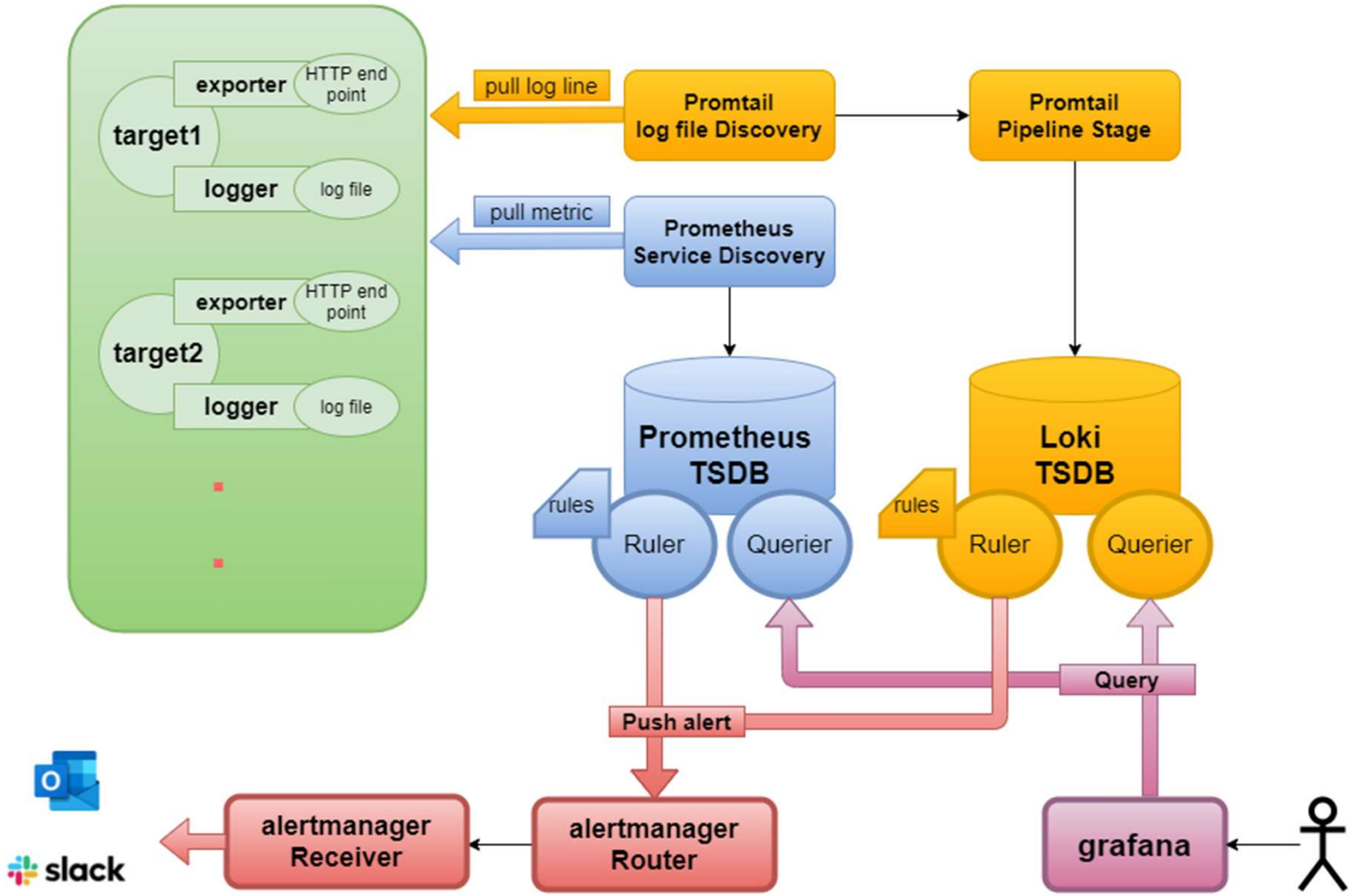
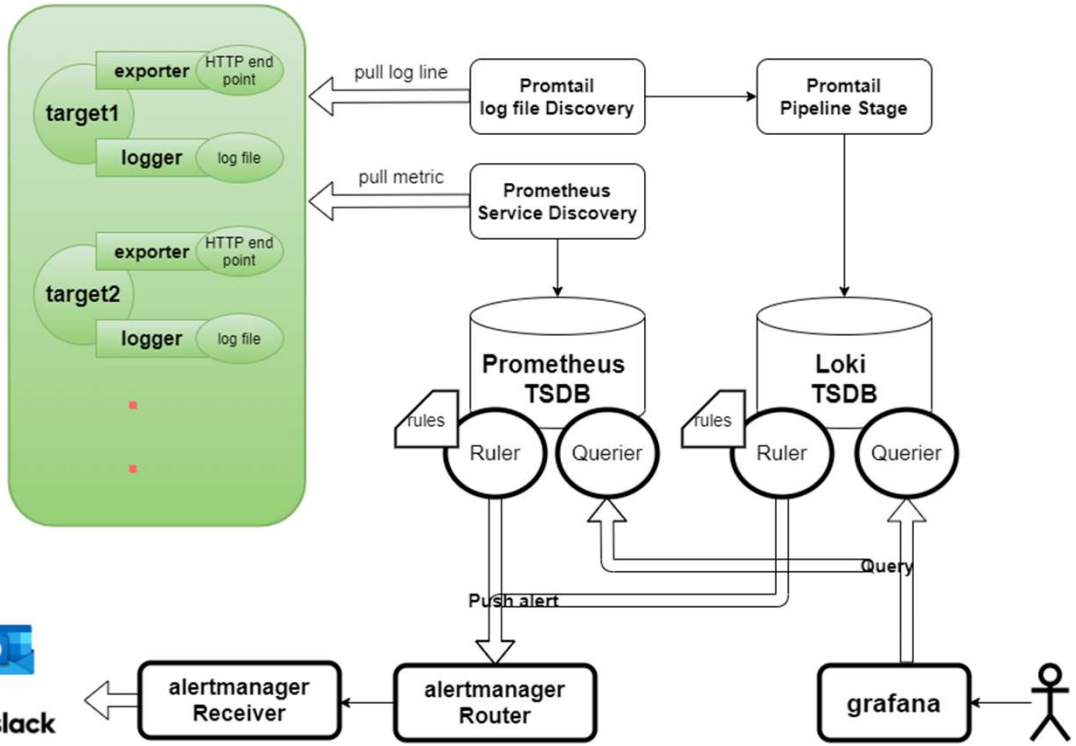
구조


exporter
HTTP end point
target1
logger
log file
exporter
HTTP end point
target2
logger
log file
pull log line
Promtail
log file Discovery
Promtail
Pipeline Stage
pull metric
Prometheus
Service Discovery
Prometheus
TSDB
Loki
TSDB
rules
Ruler
Querier
rules
Ruler
Querier
Query
Push alert
slack
alertmanager
Receiver
alertmanager
Router
grafana

모니터링 정보 노출



1) Metric(exporter)

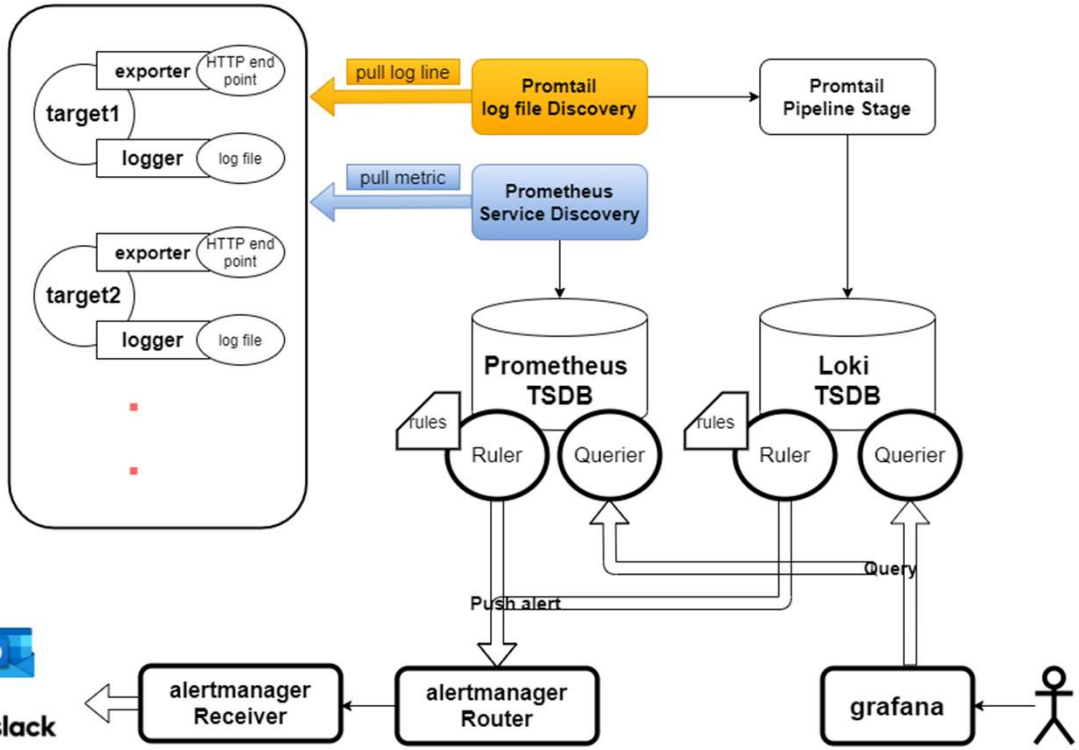
- Target metric 추출
- Prometheus가 원하는 형태로 가공
- http end point를 통해 metric 노출

⇒ **모니터링을 원하는 대상이 있으면 기본적으로 exporter가 설치되어 http end point로 모니터링을 원하는 정보를 노출해 줘야 함**

2) Log(logger)

⇒ **Target application 내부에 logging 기능을 활용해 file 형태로 log를 남겨 줘야 함**

수집



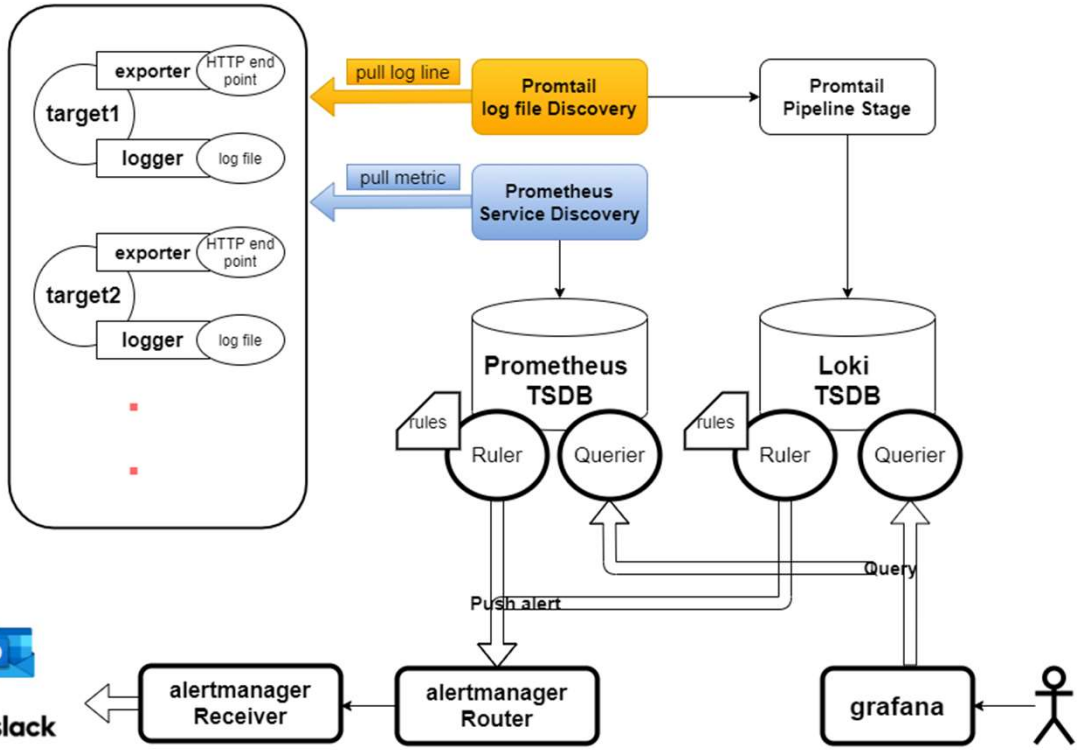
1) Metric(service discovery)

- http end point를 설정해 주면 이를 통해 메트릭 수집
- custom labeling 가능(promql에서 필터링 가능)

```
scrape_configs:
  - job_name: 'All Targets'
    scrape_interval: 10s
    file_sd_configs:
      - files:
        - ./prometheusTargets/prometheusTarget.json
```

```
{
    "targets": [
        "localhost:9183"
    ],
    "labels": {
        "job": "wmi exporter"
    }
},
{
    "targets": [
        "[redacted]"
    ],
    "labels": {
        "__metrics_path__": "/actuator/prometheus",
        "job": "spring boot"
    }
},
{
    "targets": [
        "1[redacted]0"
    ],
    "labels": {
        "job": "node exporter"
    }
},
```

수집



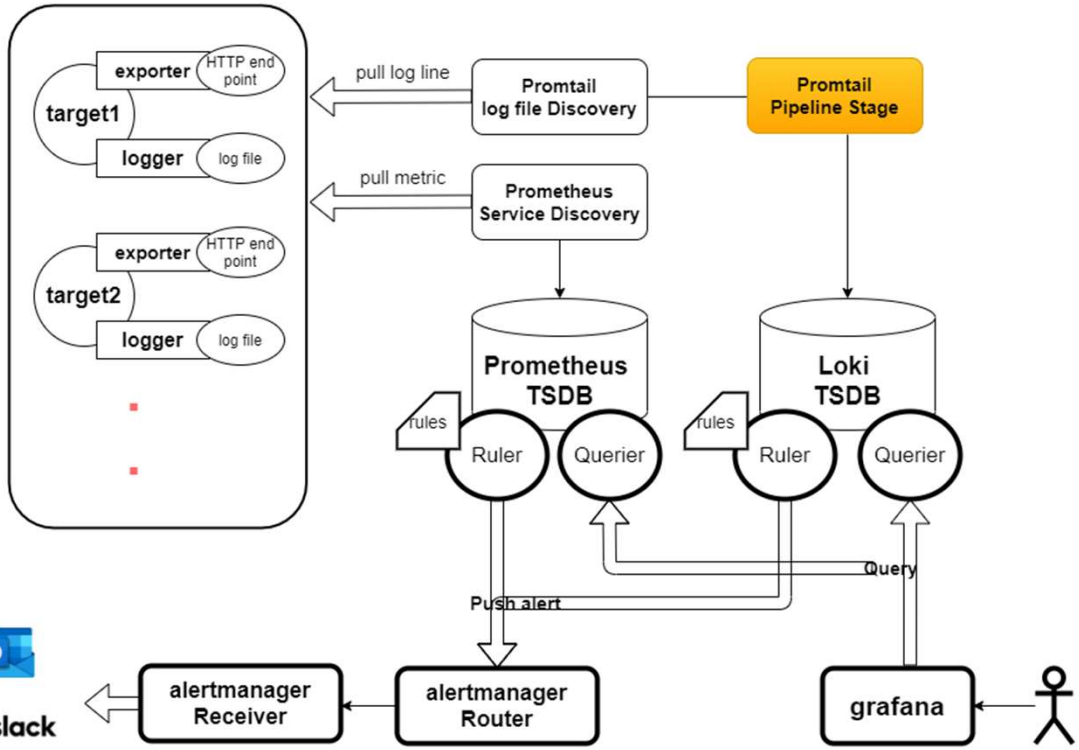
2) Log(log file discovery)

- Log file의 경로를 설정해 주면 promtail이 log를 수집

```
- job_name: 'Targets(Service discovery)'
  file_sd_configs:
    - files:
      - ./promtailConfig/promtailTargets/promtailTargets.json
```

```
{
    "targets": [
        "grafana"
    ],
    "labels": {
        "__path__": "[redacted]grafana-7.5.3/data/log/grafana.log",
        "job": "grafana"
    }
}
```

가공



1) Metric

- 별도 가공 없이 스냅샷을 그대로 push(metric 자체로 유의미한 데이터)

2) Log(Pipeline stage)

- log line은 단순한 string일 뿐임
- String을 parsing해 데이터를 추출하고 가공하는 단계

Parsing stages:

- docker : Extract data by parsing the log line using the standard Docker format.
- cri : Extract data by parsing the log line using the standard CRI format.
- regex : Extract data using a regular expression.
- json : Extract data by parsing the log line as JSON.

Transform stages:

- multiline : Merges multiple lines, e.g. stack traces, into multiline blocks.
- template : Use Go templates to modify extracted data.

Action stages:

- timestamp : Set the timestamp value for the log entry.
- output : Set the log line text.
- labels : Update the label set for the log entry.
- metrics : Calculate metrics based on extracted data.
- tenant : Set the tenant ID value to use for the log entry.

Filtering stages:

- match : Conditionally run stages based on the label set.
- drop : Conditionally drop log lines based on several options.

가공

2) Log(Pipeline stage)

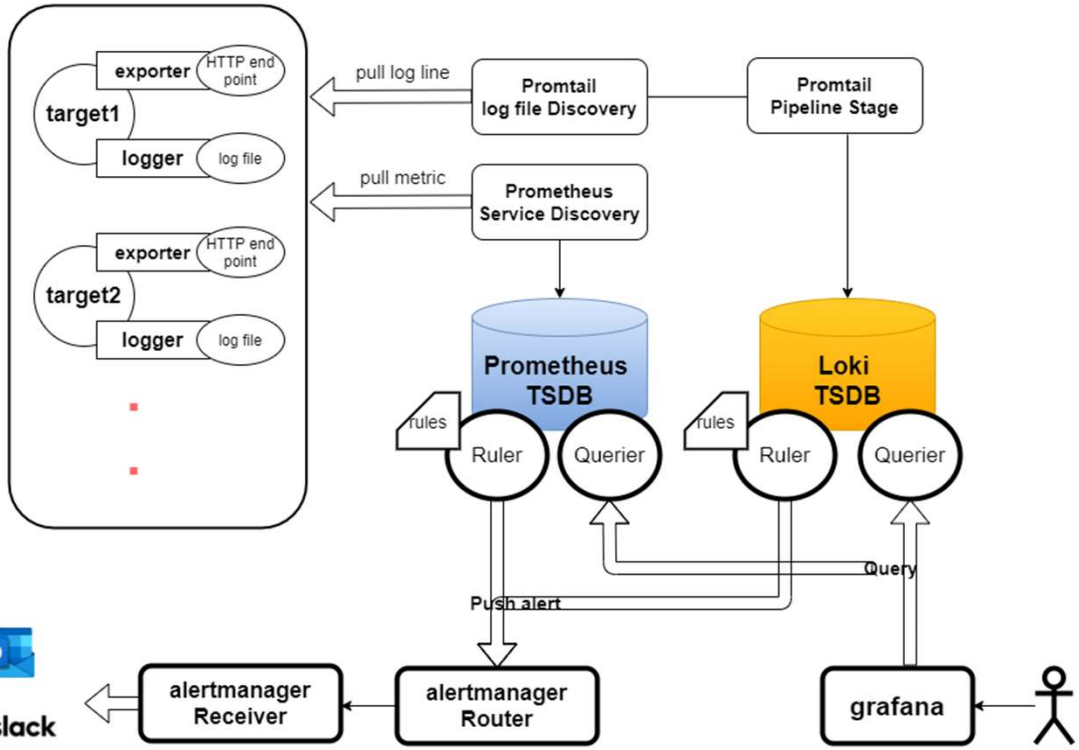
- log line은 단순한 string일 뿐임
- String을 parsing해 데이터를 추출하고 가공하는 단계

```
- job_name: 'Targets(Service discovery)'
  file_sd_configs:
    - files:
      - ./promtailConfig/promtailTargets/promtailTargets.json

  pipeline_stages:
  - match:
      selector: '{job="springboot"}'
      stages:
      - multiline:
          firstline: '^\d{4}-\d{2}-\d{2}\s\d{1,2}\:\d{2}\:\d{2}\.\d{3}'
          max_wait_time: 3s
      - regex:
          expression: '^(?s)(?P<timestamp>\d{4}-\d{2}-\d{2}\s\d{1,2}\:\d{2}\:\d{2}\.\d{3})\s+(?P<level>[A-Z]{4,5})\s(?P<pid>\d{1,10})\s---\s\[\s*(?P<thre
      - template:
          source: timestamp
          template: '{{ .timestamp }} KST'
      - labels:
          timestamp:
          level:
          pid:
          thread:
          logger:
          message:
      - timestamp:
          format: "2006-01-02 15:04:05.999 MST"
          source: timestamp
```

```
> 2021-05-13 15:27:41 2021-05-13 15:27:41.817 TRACE 10743 --- [http-nio-8080-exec-5] o.h.type.descriptor.sql.BasicExtrac
                      T]) - [2]
> 2021-05-13 15:27:41 2021-05-13 15:27:41.817 TRACE 10743 --- [http-nio-8080-exec-5] o.h.type.descriptor.sql.BasicExtrac
                      T]) - [3]
> 2021-05-13 15:27:41 2021-05-13 15:27:41.816 DEBUG 10743 --- [http-nio-8080-exec-5] org.hibernate.SQL
                          select
                              categoryen0_.id as id1_0_,
                              categoryen0_.category1 as category2_0_,
                              categoryen0_.category2 as category3_0_
                          from
                              category categoryen0_
```

저장



- 기본적으로 file형태로 local에 저장

https://prometheus.io/docs/prometheus/latest/storage/

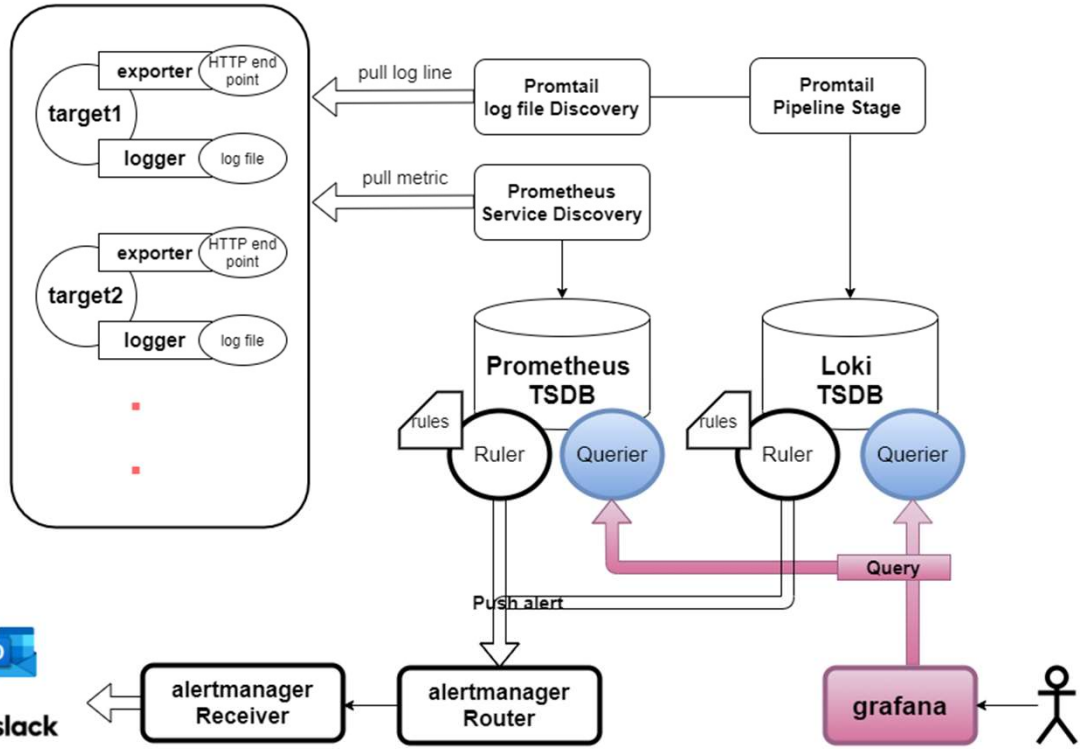
https://grafana.com/docs/loki/latest/configuration/

- 우선은 기본 설정을 유지(공부가 더 필요..)

- 추후 고려할 만한 것들

1) 저장 위치 설정
2) 저장 기간 설정
3) 저장소 max size 설정
4) Remote DB(long-term DB)
5) 등등...

시각화



1) Metric

- promql을 통해 원하는 메트릭을 가져와 시각화

```
sum by (instance)(irate(node_cpu_seconds_total{mode="system",instance="$node",job="$job"}[5m])) * 100
```

1) node_cpu_second_total
2) {mode="system",instance="$node",job="$job"}
3) [5m]
4) Irate
5) Sum by (instance)

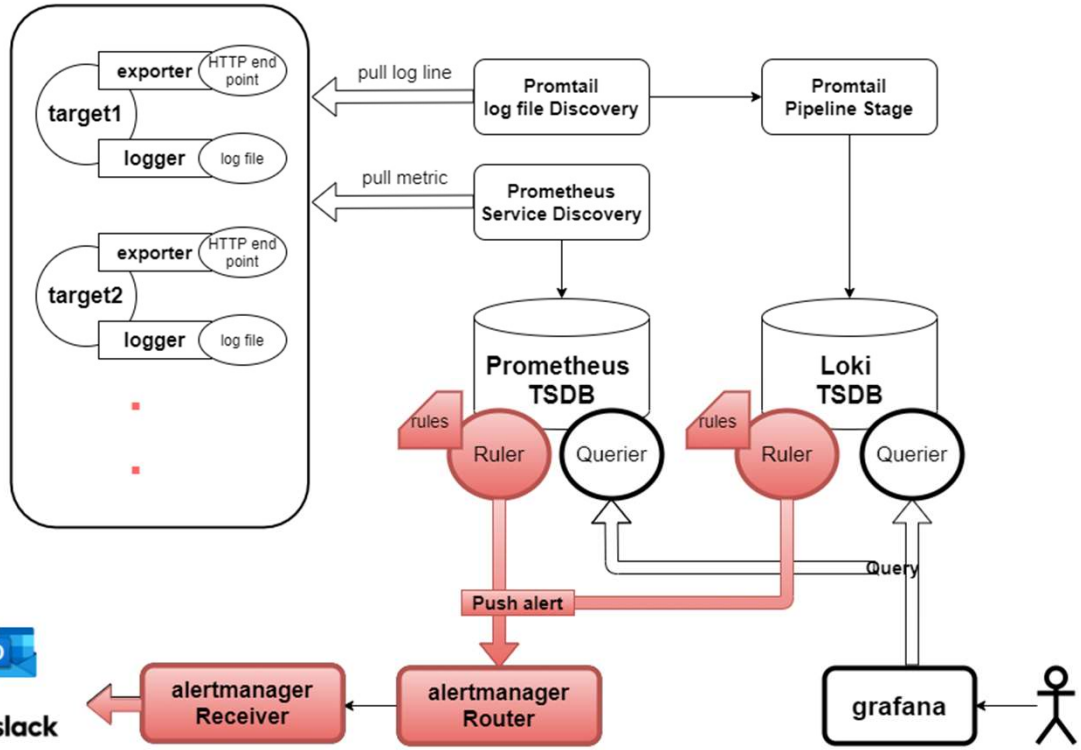
2) Log

- logql을 통해 원하는 메트릭을 가져와 시각화

```
sum by (status) (count_over_time({filename=~`.*json_access.log.*`, host="$host"} | json |  __error__="" [$__interval]))
```

1) {filename=~`.*json_access.log.*`, host="$host"}
2) json
3) __error__=""
4) [$__interval]
5) count_over_time
6) sum by (status)

경고



1) Rule 정의

- Expr & for: rule의 트리거 조건
- Labels & annotation: alertmanager로 전달되는 데이터

- prometheus rule

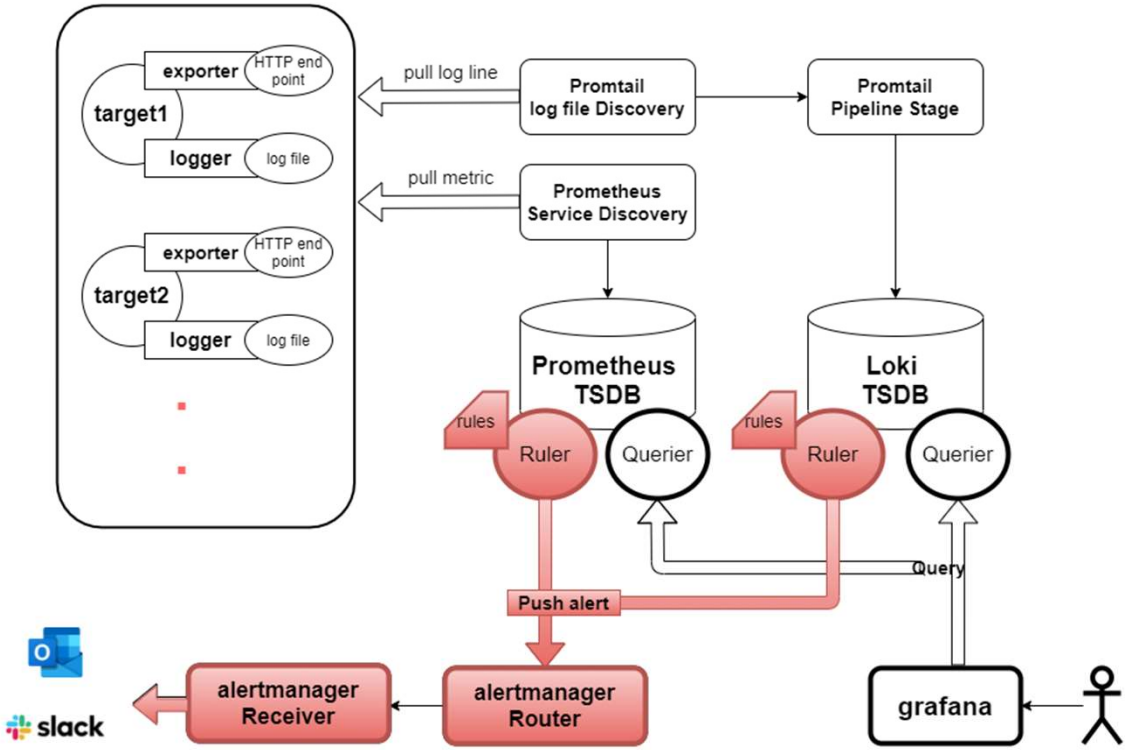
```
- alert: Node Exporter(Linux) Down Alert
  expr: up{instance="[redacted]0"} == 0
  for: 3m
  labels:
    severity: warning
    identifier: "{{$labels.instance}}"
  annotations:
    title: Node {{ $labels.instance }} is down

    description: Failed to scrape {{ $labels.job }} on {{ $labels.instance }} for more than 3 minutes. Node seems down.
```

- Loki rule

```
- alert: High log stream per second
  expr:  rate ({job="springboot"}[5m]) > 0.5
  for: 3s
  labels:
      severity: page
      datasource: loki
  annotations:
      title: High log stream per second
      description: springboot my Blog - High log stream!!
```

경고



2) Ruler 설정

- prometheus ruler

```
evaluation_interval: 15s
# scrape_timeout is set to the global default

# Alertmanager configuration
alerting:
  alertmanagers:
  - static_configs:
    - targets:
      - localhost:9093

# Load rules once and periodically evaluate
rule_files:
  - "./prometheusRules/nodeExporterRule.yml"
  - "./prometheusRules/springBootRule.yml"
  - "./prometheusRules/wmiExporterRule.yml"
  # - "second_rules.yml"
```

평가 주기

어디로 보낼지

어떤 rule을 평가할지

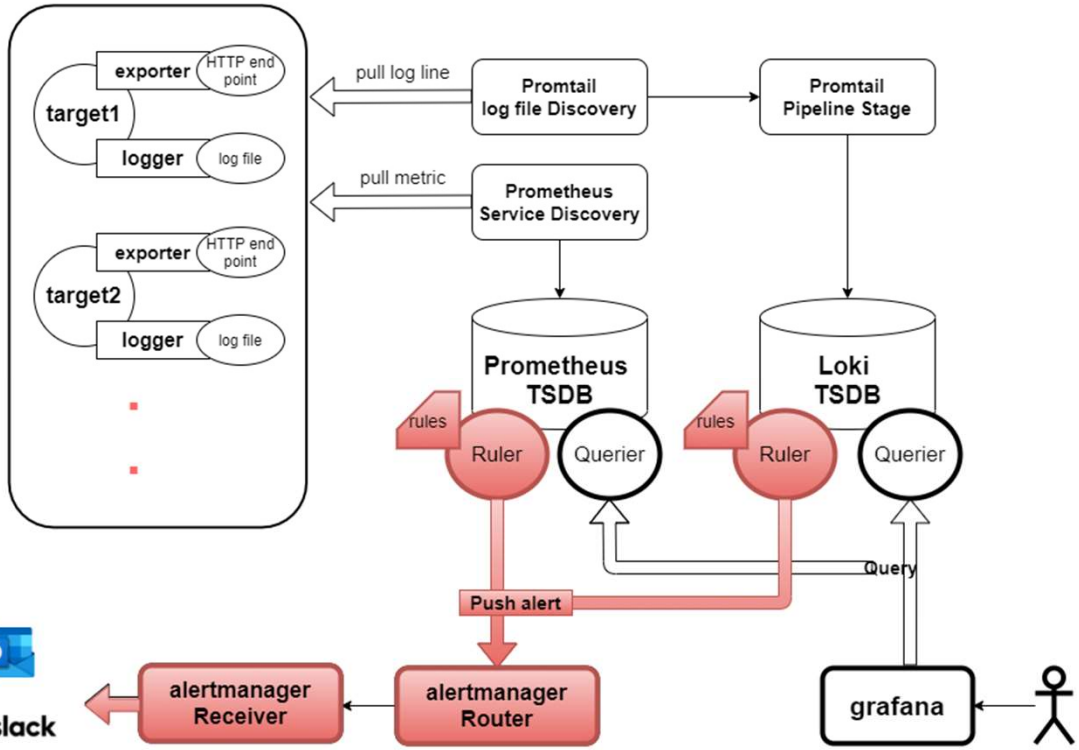
- Loki ruler

```
ruler:
  storage:
    type: local
    local:
      directory: ./lokiConfig/lokiRules
    rule_path: ./lokiConfig/lokiRules-temp
  alertmanager_url: http://localhost:9093
  ring:
    kvstore:
      store: inmemory
  enable_api: true
  enable_alertmanager_v2: true
```

어떤 rule을 평가할지

어디로 보낼지

경고



3) Alertmanager에서 수신 및 라우팅

- 기본적으로 모든 alert은 root로 들어옴
- Root로 들어온 alert을 적절한 receiver에게 전달하는 설정
- Match 조건에서 alert을 통해 전달된 label정보로 필터링

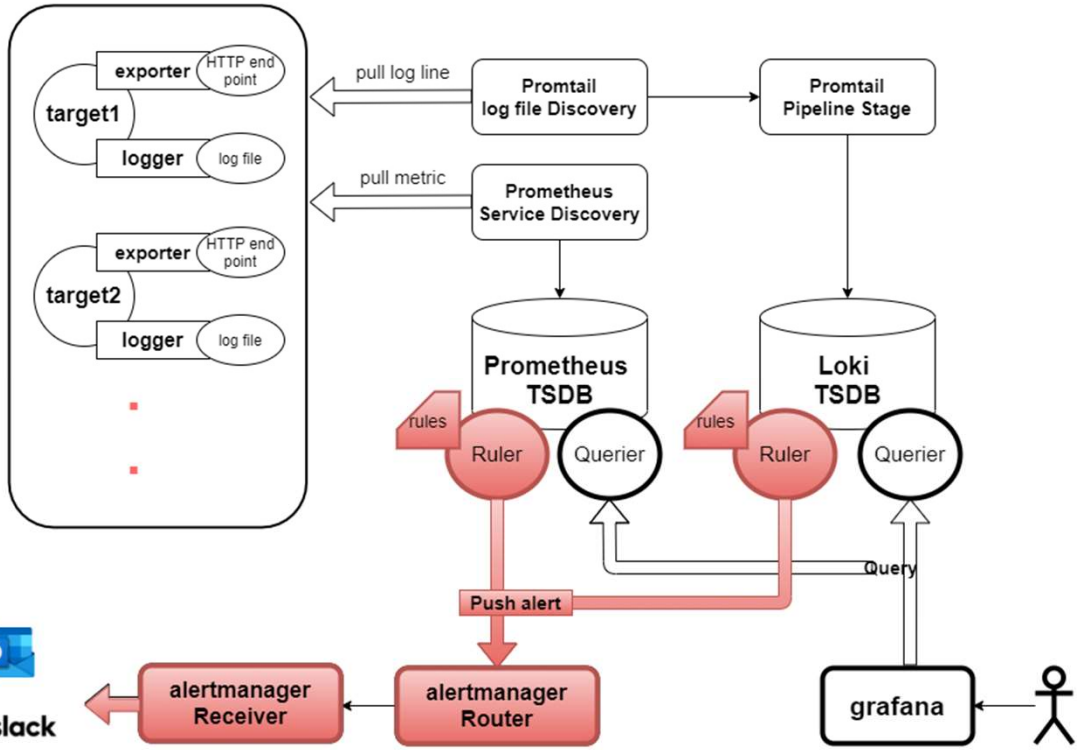
```
# prometheus가 보내는 모든 알람은 그냥 다 이 alertmanager로 보내질 것임
# 보내는 alert의 종류에 따라 다른 receiver를 지정하고 싶을 수 있음
# 기본적으로 root로 들어오는 alert을 다른 reciever로 전달하는 부분이 route 설정임
route:
  group_by: ['alertname']
  group_wait: 30s
  repeat_interval: 1s
  receiver: 'nodeExporter'
  routes:
  - receiver: 'nodeExporter'
    group_wait: 10s
    match:
      identifier: 1[redacted]04:9100
  - receiver: 'wmiExporter'
    group_wait: 10s
    match:
      identifier: localhost:9183

  - receiver: 'loki'
    group_wait: 10s
    match:
      datasource: loki
```

2. 어느 receiver로 보낼지

1. 어떤 알람을

4) Receiver에서 특정 template에 맞게 notify



모니터링 타겟 별 디테일

모니터링 타겟 별 디테일

1) Nginx

- 주요 기능

주요기능	지원여부
Backend-service 장애 대응 처리	서비스로 연결되어 있는 서버들의 max fails, fail timeout 의 옵션으로 해당 서버들의 상태를 체크하여 특정 서버의 장애 발생 시 백업 서버로 연결해주는 기능이다.
Load Balancing	Nginx 의 로드 밸런싱 기능을 사용하여 여러 대의 웹 서버를 연결하여 분산 처리해주는 기능이다.
Keep alive 제어	Socket에 IN/OUT Access가 마지막으로 종료된 시점부터 지정된 시간까지 Access 가 없더라도 대기하는 구조. 정의된 시간내에 Access 가 이뤄진다면 연결상태를 유지하도록 설정해주는 기능이다.
Sub-domain (Nginx) 관리	여러 도메인을 가상 호스트를 설정하여 사용 할 수 있는 기능이다.
caching 처리	image 및 기타 콘텐츠 데이터에 대해 캐싱해 주는 기능이다.

1) Nginx

- 메트릭 노출

(1) Nginx nginx.conf: stub_status 설정을 통해 기본 상태 정보 노출

http://nginx.org/en/docs/http/ngx_http_stub_status_module.html

```
location /metrics {
    stub_status on;
}
```

(2) 해당 end point를 바라보도록 nginx exporter 실행

```
./nginx-prometheus-exporter -nginx.scrape-uri=http://localhost:80/metrics
```

1) Nginx

- 메트릭 노출

(3) Prometheus http end point 제공

```
./nginx-prometheus-exporter -nginx.scrape-uri=http://localhost:80/metrics
```

주의 요함 | [redacted]4:9113/metrics

```
# HELP nginx_connections_accepted Accepted client connections
# TYPE nginx_connections_accepted counter
nginx_connections_accepted 140
# HELP nginx_connections_active Active client connections
# TYPE nginx_connections_active gauge
nginx_connections_active 2
# HELP nginx_connections_handled Handled client connections
# TYPE nginx_connections_handled counter
nginx_connections_handled 140
# HELP nginx_connections_reading Connections where NGINX is reading the request header
# TYPE nginx_connections_reading gauge
nginx_connections_reading 0
# HELP nginx_connections_waiting Idle client connections
# TYPE nginx_connections_waiting gauge
nginx_connections_waiting 1
# HELP nginx_connections_writing Connections where NGINX is writing the response back to the client
# TYPE nginx_connections_writing gauge
nginx_connections_writing 1
# HELP nginx_http_requests_total Total http requests
# TYPE nginx_http_requests_total counter
nginx_http_requests_total 25342
# HELP nginx_up Status of the last metric scrape
# TYPE nginx_up gauge
nginx_up 1
# HELP nginxexporter_build_info Exporter build information
# TYPE nginxexporter_build_info gauge
nginxexporter_build_info{commit="5f88afbd906baae02edfbab4f5715e06d88538a0",date="2021-03-22T20:16:09Z",version="0.9.0"} 1
```

1) Nginx

- 로그 파일 노출

(1) Nginx log 설정을 통해 log파일 작성

- Nginx variable 사용
- https://nginx.org/libxslt/en/docs/varindex.html?_ga=2.176189797.1315513953.1620778269-1229558557.1620348426

- Log line parsing을 통해 로그에 남은 유용한 정보를 얻어 메트릭으로 사용

```
log_format json_analytics escape=json '{'
                        '"msec": "$msec", ' # request unixtime in seconds with a milliseconds resolution
                        '"connection": "$connection", ' # connection serial number
                        '"connection_requests": "$connection_requests", ' # number of requests made in connection
                '"pid": "$pid", ' # process pid
                '"request_id": "$request_id", ' # the unique request id
                '"request_length": "$request_length", ' # request length (including headers and body)
                '"remote_addr": "$remote_addr", ' # client IP
                '"remote_user": "$remote_user", ' # client HTTP username
                '"remote_port": "$remote_port", ' # client port
                '"time_local": "$time_local", '
                '"time_iso8601": "$time_iso8601", ' # local time in the ISO 8601 standard format
                '"request": "$request", ' # full path no arguments if the request
                '"request_uri": "$request_uri", ' # full path and arguments if the request
                '"args": "$args", ' # args
                '"status": "$status", ' # response status code
                '"body_bytes_sent": "$body_bytes_sent", ' # the number of body bytes exclude headers sent to a client
                '"bytes_sent": "$bytes_sent", ' # the number of bytes sent to a client
                '"http_referer": "$http_referer", ' # HTTP referer
                '"http_user_agent": "$http_user_agent", ' # user agent
                '"http_x_forwarded_for": "$http_x_forwarded_for", ' # http_x_forwarded_for
                '"http_host": "$http_host", ' # the request Host: header
                '"server_name": "$server_name", ' # the name of the vhost serving the request
                '"request_time": "$request_time", ' # request processing time in seconds with msec resolution
                '"upstream": "$upstream_addr", ' # upstream backend server for proxied requests
                '"upstream_connect_time": "$upstream_connect_time", ' # upstream handshake time incl. TLS
                '"upstream_header_time": "$upstream_header_time", ' # time spent receiving upstream headers
                '"upstream_response_time": "$upstream_response_time", ' # time spend receiving upstream body
                '"upstream_response_length": "$upstream_response_length", ' # upstream response length
                '"upstream_cache_status": "$upstream_cache_status", ' # cache HIT/MISS where applicable
                '"ssl_protocol": "$ssl_protocol", ' # TLS protocol
                '"ssl_cipher": "$ssl_cipher", ' # TLS cipher
                '"scheme": "$scheme", ' # http or https
                '"request_method": "$request_method", ' # request method
                '"server_protocol": "$server_protocol", ' # request protocol, like HTTP/1.1 or HTTP/2.0
                '"pipe": "$pipe", ' # "p" if request was pipelined, "." otherwise
                '"gzip_ratio": "$gzip_ratio", '
                '"http_cf_ray": "$http_cf_ray",'
                '"geoip_country_code": "ko"'
                '}';

access_log  /var/log/nginx/json_access.log json_analytics;
```

2) Spring boot

- 메트릭 노출

Spring Boot Actuator + Micrometer Registry

- spring boot actuator와 micrometer-registry-prometheus 추가

pom.xml

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>
```

application.yml

- spring boot actuator의 prometheus 엔드포인트 활성화
- 서비스를 구분하기 위한 common tag 추가(서비스는 한 개의 이상이 인스턴스로 이루어짐)
  - 일반적으로 application이라는 이름으로 `spring.application.name` 속성이 많이 사용됨
  - 각 인스턴스를 구분하기 위한 instance 태그는 Prometheus에 의해서 추가됨으로 설정 불필요

```
spring:
  application:
    name: my_spring_boot_app
management:
  endpoints:
    web:
      exposure:
        include: "prometheus"
  metrics:
    tags:
      application: ${spring.application.name}    # 서비스 단위의 식별자. Prometheus label에 추가됨.
```

2) Spring boot

- 로그 파일 노출

- 기본 옵션은 log를 콘솔에만 찍고 파일로 안 남김
- Application.properties(또는 yml) 설정

```
#로그파일 노출
logging.file.path=logs
```

3) Postgres

- 메트릭 노출

Postgres exporter

- Postgres는 자체적으로 자신의 상태 정보를 저장하는 테이블을 관리하고 있음
- 필요 시, postgres db에 원하는 정보를 쿼리해서 Prometheus end point로 노출
- 일종의 데이터베이스 클라이언트

```
sudo -u postgres DATA_SOURCE_NAME=postgresql://postgres:[redacted]@localhost:5432?sslmode=disable nohup ./postgres_exporter
```

3) Postgres

- 로그 파일 노출

postgresql.conf의 설정을 통해 logging 관련 설정 가능

- https://postgresql.kr/docs/9.6/runtime-config-logging.html