

K8s LMA

# venti-stack 개발기: 파트 2

김정민 - 삼성SDS



# Speaker



## 김정민

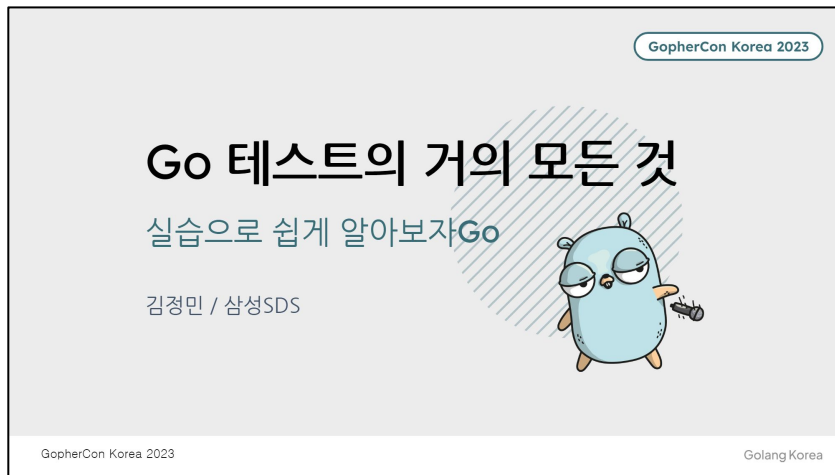
삼성SDS에서 Samsung Kubernetes Engine을 개발하는 업무를 하고 있습니다. K8s와 Go 테스트 관련 주제에 관심이 많습니다. 오픈소스 활동으로 Lethe (Log DB), Venti (Visualizer)를 개발하였습니다.

발표자료는? 질문은?

<https://github.com/jmnote/slides>



# 지난 발표

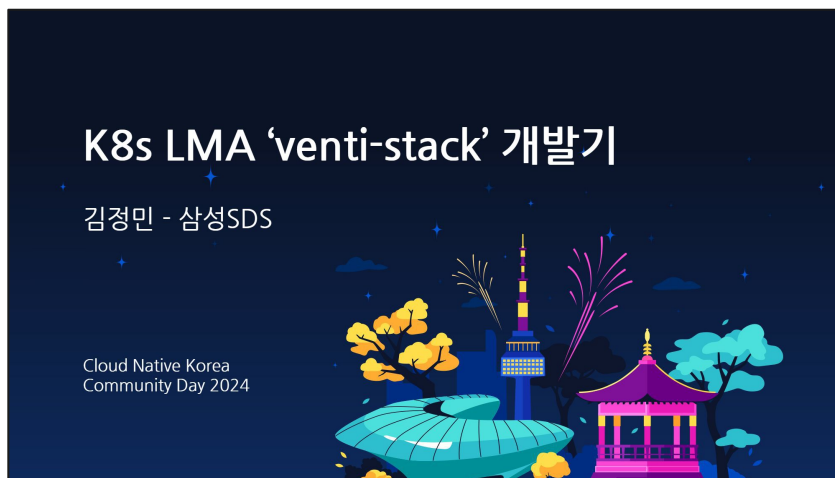


2023-08-05

## GopherCon Korea 2023

제목: Go 테스트의 거의 모든 것

Go 테스트에 관한 문서들을 모아서 목차를 만들고 정리하여 발표했습니다.



2024-09-24

## Cloud Native Korea Community Day 2024

제목: K8s LMA 'venti-stack' 개발기

같은 주제로 발표하였습니다(20분).

이번 Gophercon 발표에서는 Go 개발 관련 내용을 상세히 말씀드리겠습니다(60분).

발표자료: <https://github.com/imnote/slides> 질문 환영합니다

# 목차

I . Venti-Stack

II . Lethe (Log DB)

III . Lethe 리뷰

IV . Venti (Visualizer)

V . 기타

# Venti-Stack

# K8s LMA란?

K8s 환경에서 Logging/Monitoring/Alerting(로깅/모니터링/알림)을 통합관리하는 시스템/소프트웨어 스택

Logging	클러스터/애플리케이션의 로그 데이터 수집/저장/조회
Monitoring	클러스터/애플리케이션의 메트릭 데이터 수집/저장/조회
Alerting	수집된 데이터에서 특정 조건 발생시 운영자에게 알림 발송

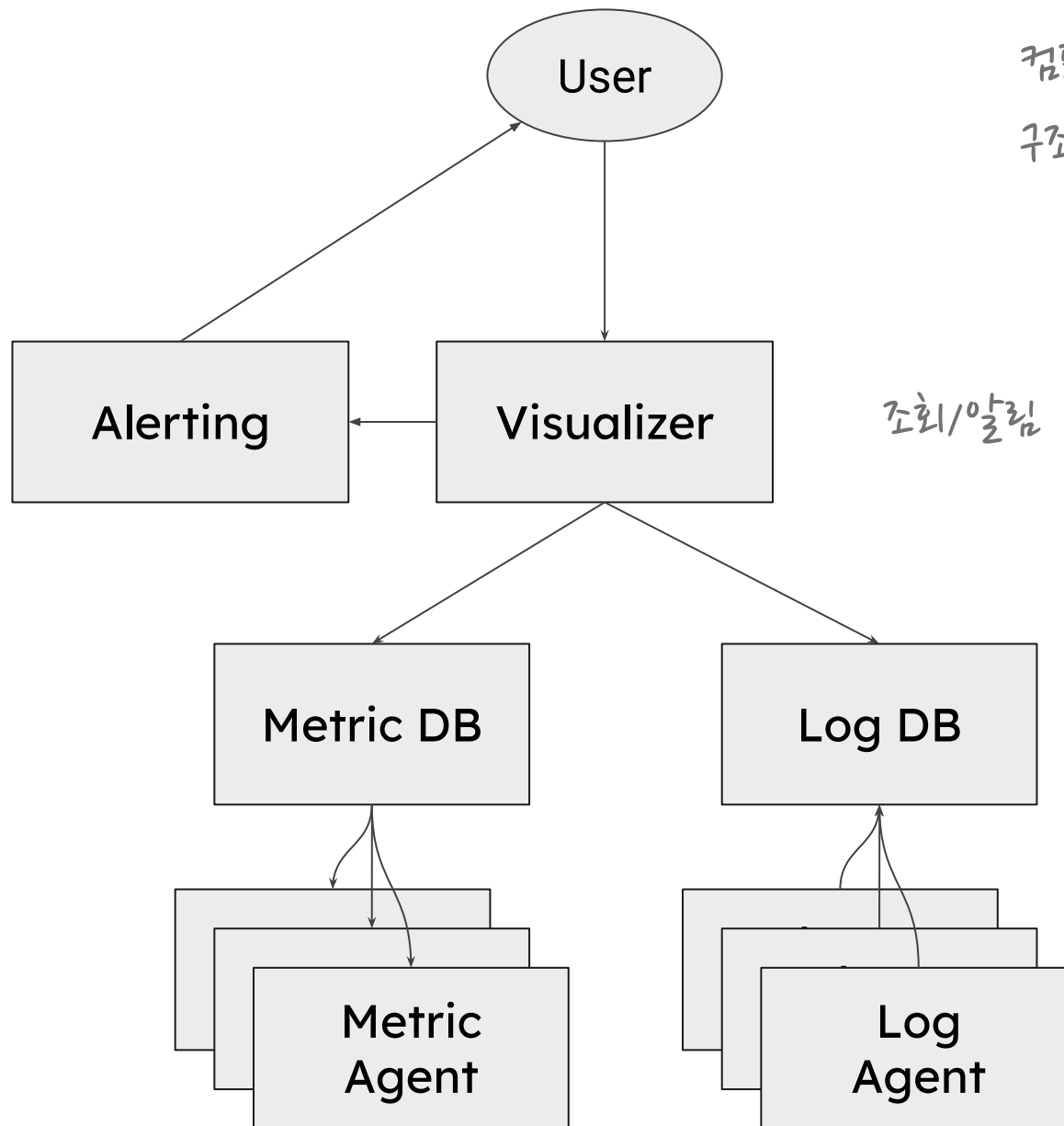
## 로그 예시

```
$ kubectl logs pod1
[INFO] Starting the application...
[INFO] Successfully connected to the DB.
[ERROR] Failed to connect to the API.
```

## 메트릭 예시

```
$ kubectl top pod
NAME      CPU(cores)   MEMORY(bytes)
pod1      2m           15Mi
pod2      3500m        16384Mi
```

# k8s LMA 구성도 (개념)



컴포넌트 추가될 수 있으나  
구조는 대략 이렇다...

조리/이탈

저장

수집

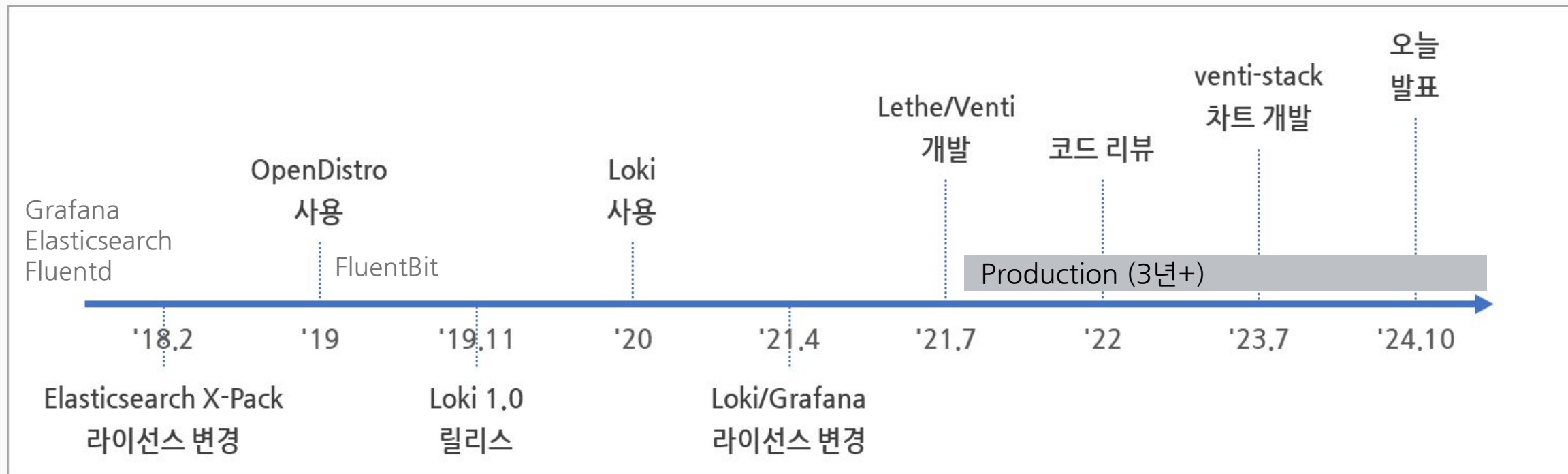
# venti-stack이란?

## K8s LMA 스택

- Helm Chart 제공 LMA 한번에 설치
- Production 사용 중 3년+
- Apache-2.0 라이선스
- 자체 개발 컴포넌트 포함
  - Lethe LogDB
  - Venti visualizer

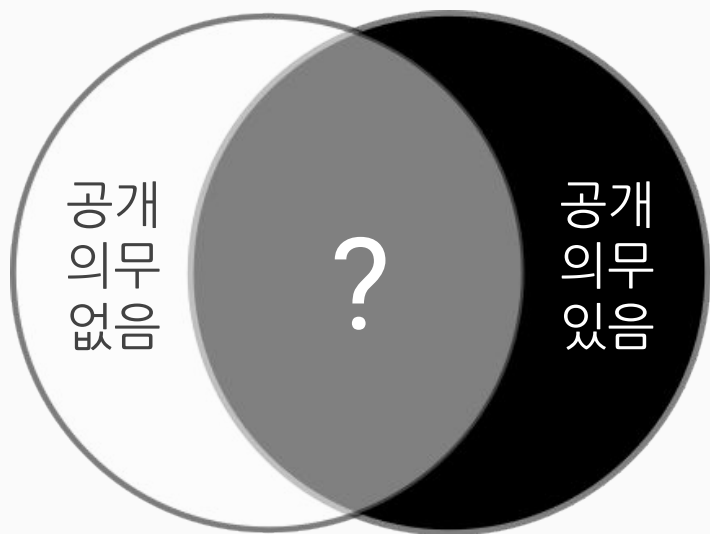


# 타임라인



- 2018-02-28 <https://www.elastic.co/kr/blog/doubling-down-on-open>
- OpenDistro: OpenSearch의 전신
- 2021-04-20 <https://grafana.com/blog/2021/04/20/grafana-loki-tempo-relicensing-to-agplv3/>

# AGPL에 대한 의견들



회색 영역

- 바이너리를 그대로 수정 없이 사용하면 괜찮다.<sup>1)</sup>
- 대기업이 AGPL을 피하는 이유 ... 소스코드를 직접 수정하지 않더라도 전체 시스템이 AGPL의 적용을 받을 수 있다.<sup>2)</sup>
- AGPL 소프트웨어를 사용하려면 링크하는 모든 것도 AGPL에 따라 라이선스를 받아야 한다. ... 위험이 이점보다 훨씬 크다.<sup>3)</sup>

결론? 잘 모르겠습니다(회색영역)  
OSS.kr 문의 사례 참고 4) 5) 6) 7)

※ 법적 분쟁 발생시, 본 자료는 법률적 해석이나 논리로 활용될 수 없습니다.

1. <https://medium.com/swlh/understanding-the-agpl-the-most-misunderstood-license-86fd1fe91275>
2. <https://www.signority.com/2024/05/03/is-agpl-a-scam-how-small-companies-can-maximize-benefits-while-remaining-compliant/>
3. <https://opensource.google/documentation/reference/using/agpl-policy/>
4. 2021-09-10 [https://www.oss.kr/oss\\_license\\_qna/show/6c691ee5-bdb3-4f91-ae2a-0e9325f89b8f](https://www.oss.kr/oss_license_qna/show/6c691ee5-bdb3-4f91-ae2a-0e9325f89b8f)
5. 2022-11-17 [https://www.oss.kr/oss\\_license\\_qna/show/cf42621c-2fef-461f-a245-e28ad3361866](https://www.oss.kr/oss_license_qna/show/cf42621c-2fef-461f-a245-e28ad3361866)
6. 2023-06-26 [https://www.oss.kr/oss\\_license\\_qna/show/07adc90a-ab99-4a24-9c75-92df081a058e](https://www.oss.kr/oss_license_qna/show/07adc90a-ab99-4a24-9c75-92df081a058e)
7. 2023-07-19 [https://www.oss.kr/oss\\_license\\_qna/show/d6847e91-a21c-4188-8914-7aba2a0a316c](https://www.oss.kr/oss_license_qna/show/d6847e91-a21c-4188-8914-7aba2a0a316c)

# venti-stack의 라이선스

## Apache-2.0

- Permissive 라이선스 자유로운 사용
- Kubernetes, Prometheus와 같음

## 라이선스 검증 방법

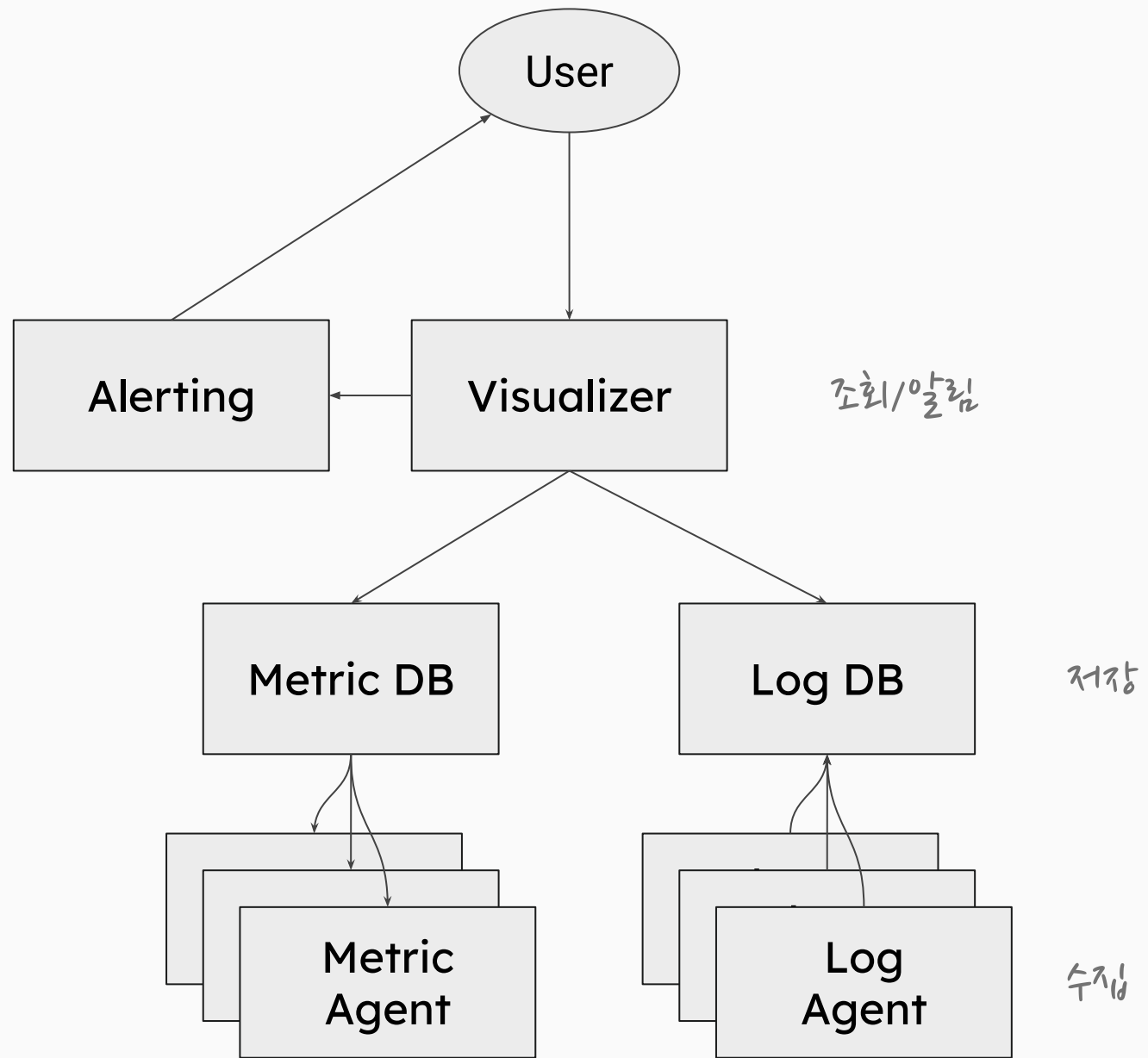
- go-licenses 활용<sup>1)</sup>
  - Google에서 만든 Go용 라이선스 검증 도구
  - 패키지 트리과 라이선스 문서를 분석/보고(report)
  - Google 정책에 따라 금지된 라이선스 확인(check)
- GitHub Actions 라이선스 검증
  - Pull Request 등록시 go-licenses 실행<sup>2) 3)</sup>

1. <https://github.com/google/go-licenses>

2. <https://github.com/kuoss/lethe/blob/v0.2.5/.github/workflows/pull-request.yml>

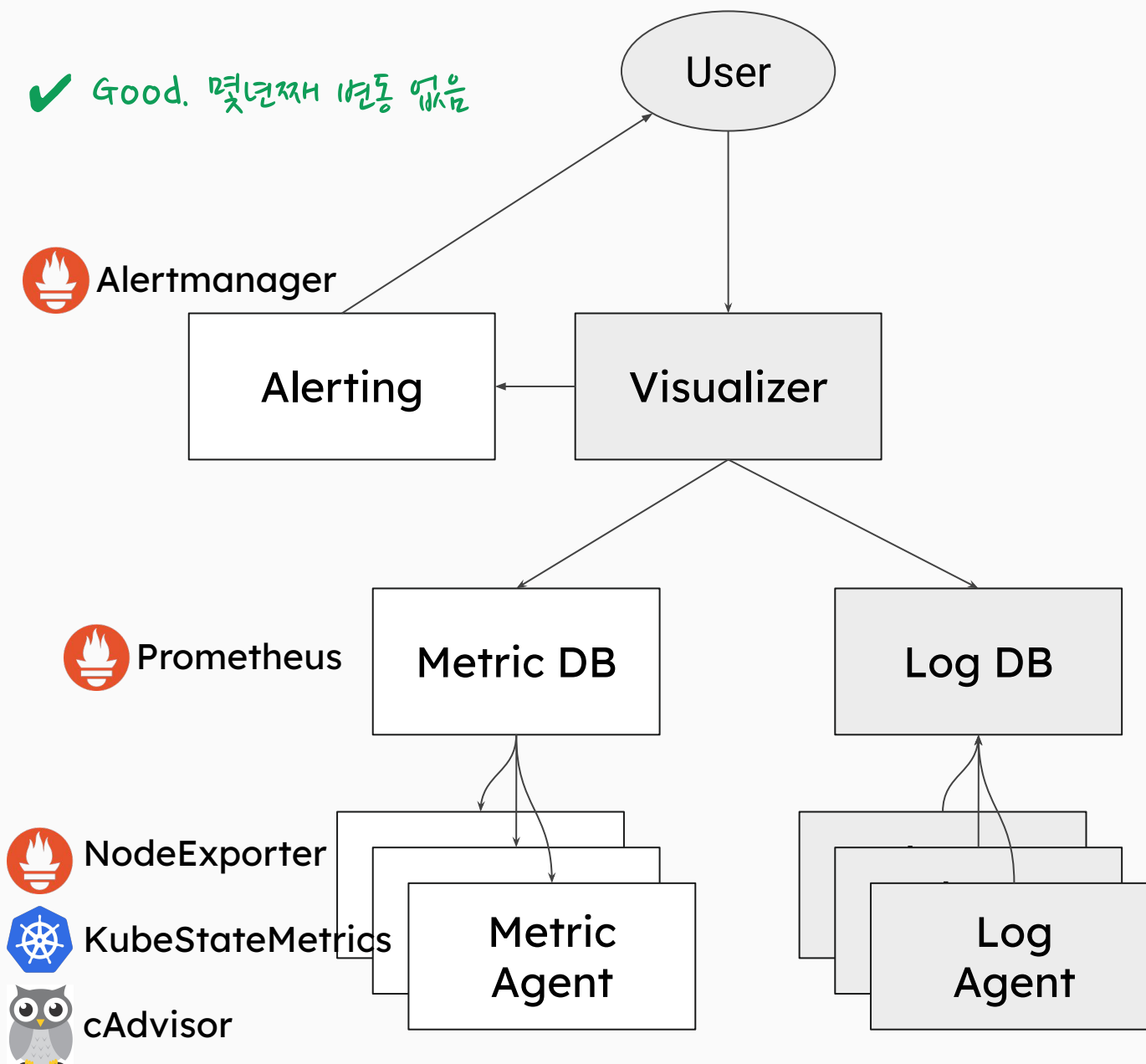
3. <https://github.com/kuoss/venti/blob/v0.2.20/.github/workflows/pull-request.yml>

# k8s LMA 구성도 (개념)

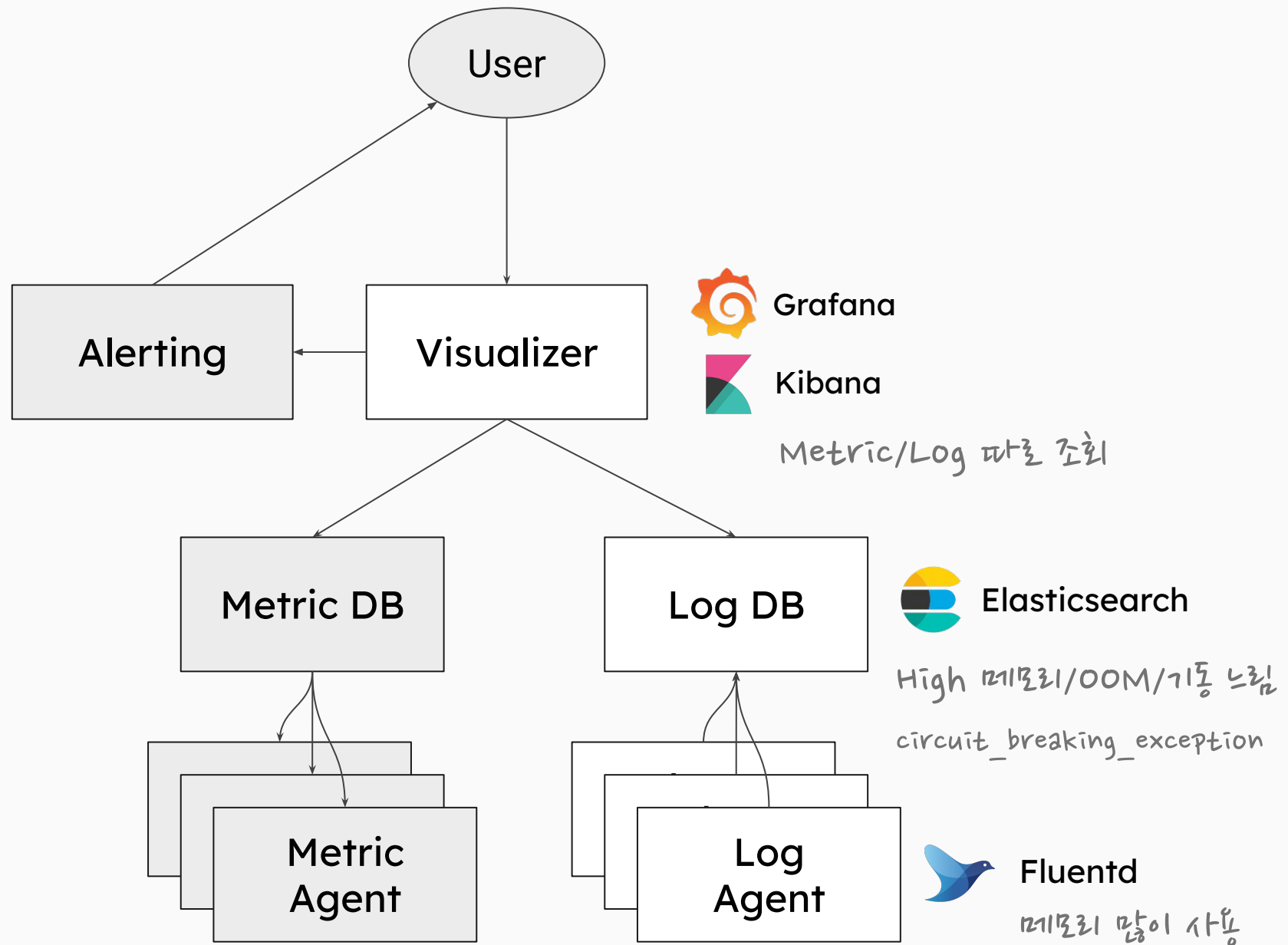


# k8s LMA 구성도 (Metric과 Alerting)

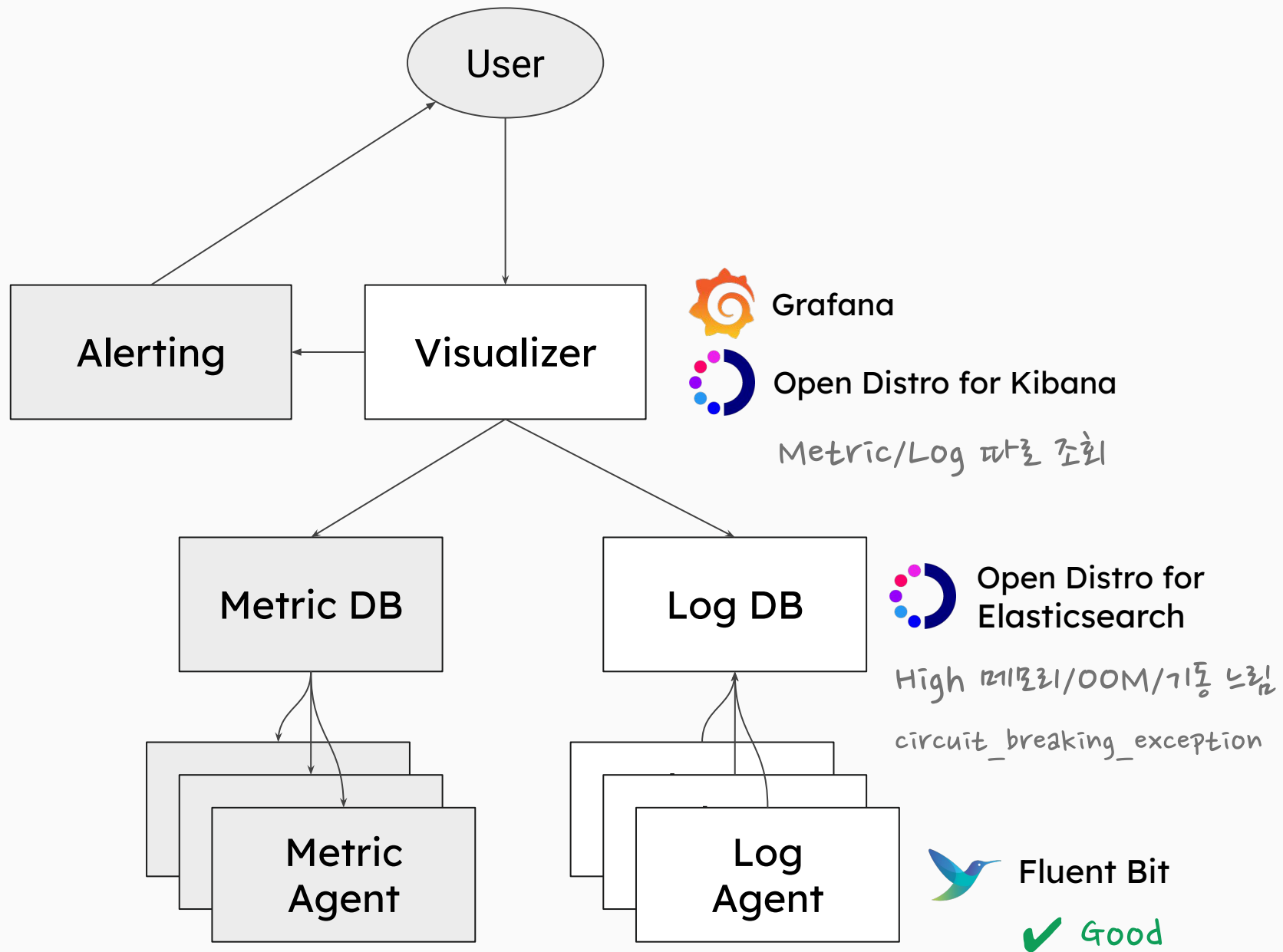
✓ Good. 몇년째 변동 없음



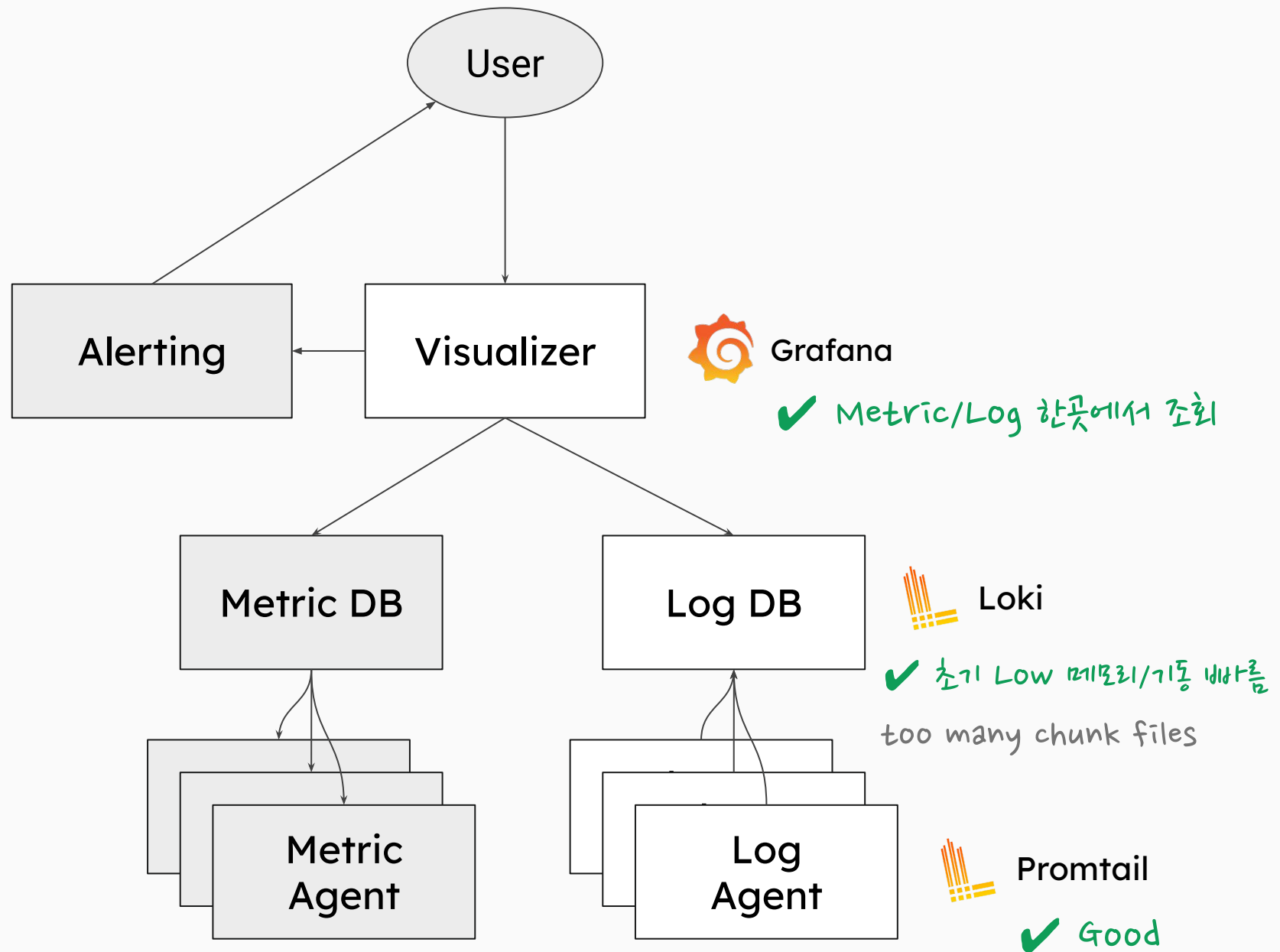
# k8s LMA 구성도 (2018)



# k8s LMA 구성도 (2019)



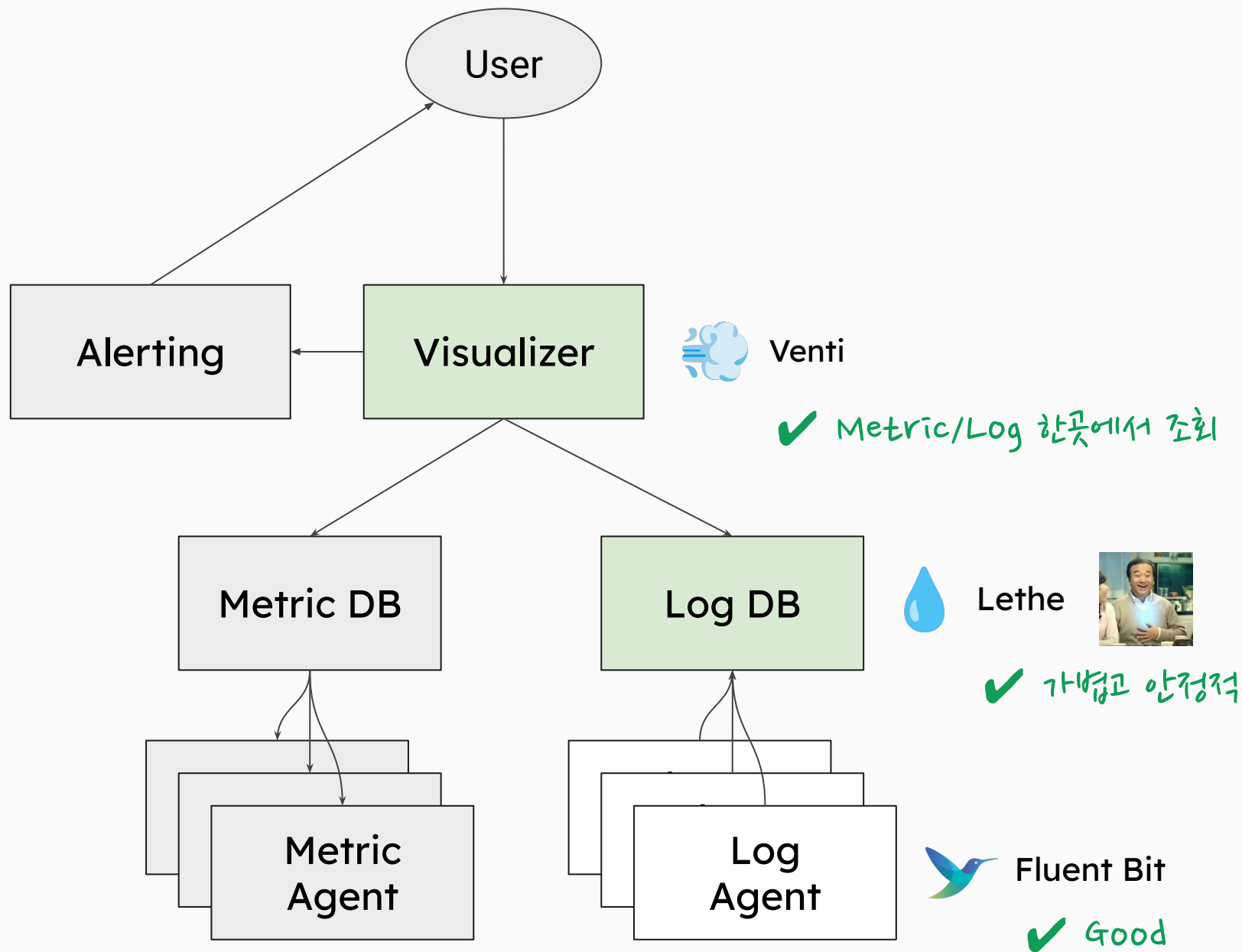
# k8s LMA 구성도 (2020)



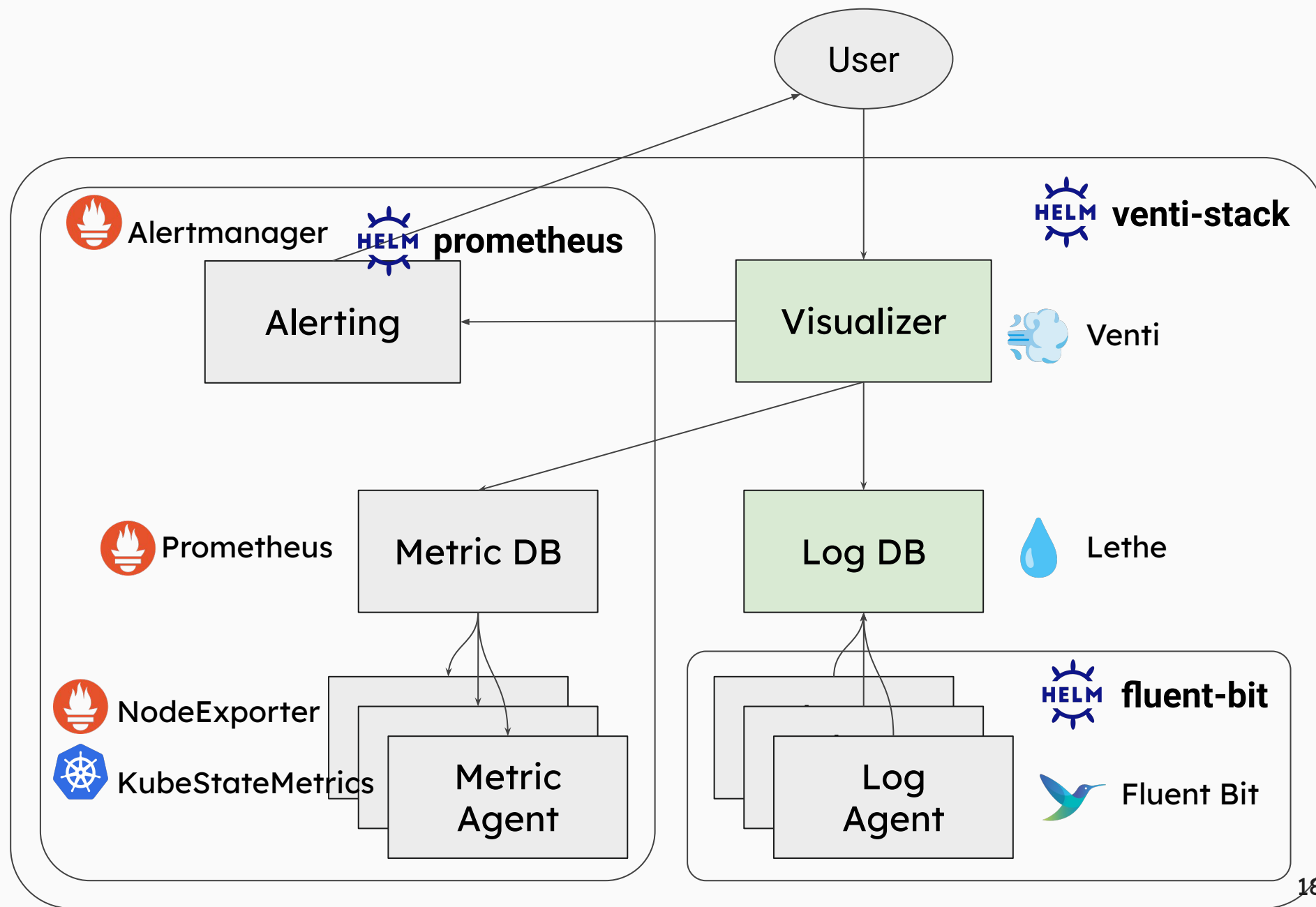


# k8s LMA 구성도 (2021~)

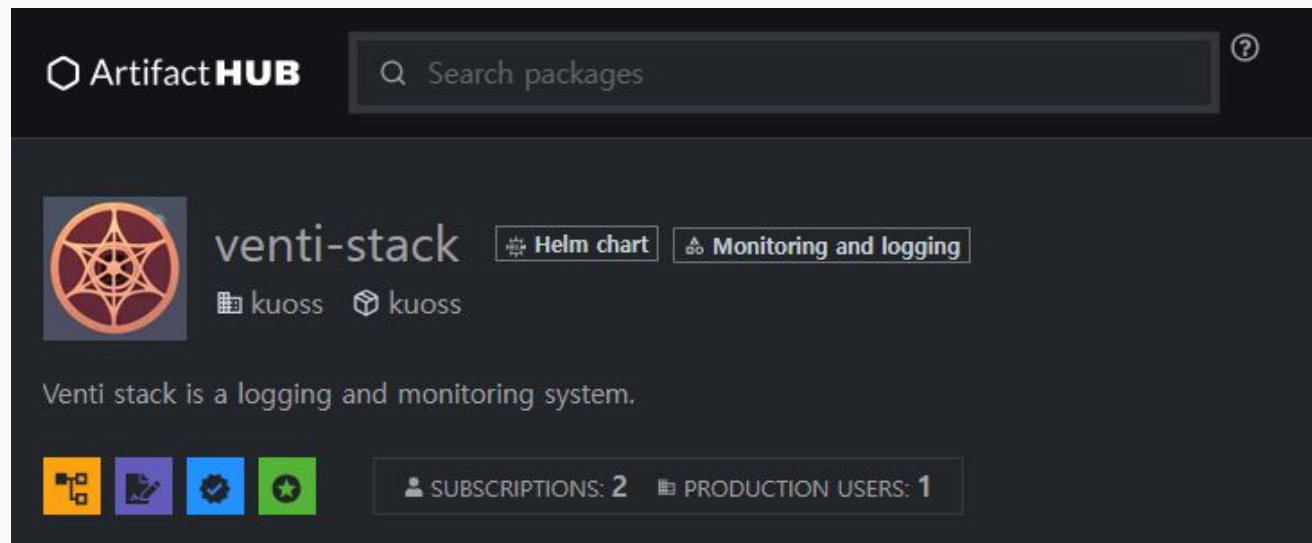
✓ Production 3년+



# venti stack 차트 (2023~)



# 설치 방법



→ 검색엔진(또는 ArtifactHUB)에서 'venti-stack' 검색

```
helm repo add kuoss https://kuoss.github.io/helm-charts
helm repo update
helm install vs --namespace=venti-stack kuoss/venti-stack
```

→ values.yaml에 prometheus.ingress와 venti.ingress 설정하면 좋겠지?

※ kuoss = kubernetes open source software

# 이름은 어떻게 지었나?

다음 기준에 따라 아이디어 내고 투표

- 1) 신화에 나오는 이름
- 2) 기억하기 쉽게 LogDB는 L로, visualizer는 V로 시작
- 3) Prometheus 상징이 불이니까, LogDB는 물, visualizer는 바람
- 4) 개발할 작업은 거. ~~인간적으로 prometheus는 너무 길지 않음?~~




※ MVP 개발기간은?

4주 정도 ('21.07)

(lethe 2주, venti 2주)

그 결과... (이런 명령어 있습니다...)

```
$ kubectl get VentiStackComponents
```

NAME	TYPE	MEANING	SYMBOL
Prometheus	MetricDB	불의 신	 불(햇불)
Lethe	LogDB	망각의 강	 물
Venti	Visualizer	바람의 신	 바람

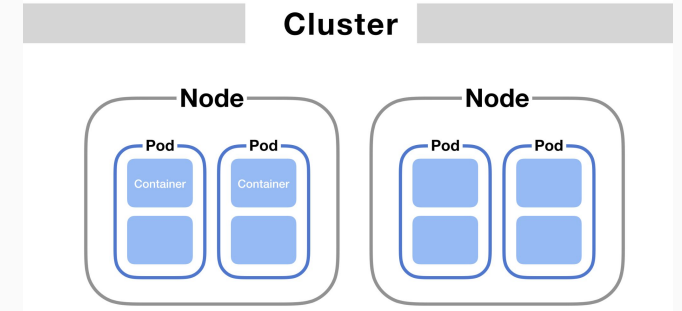
~~2개만 더 붙으면 되겠다~~



# II Lethe (Log DB)

# K8s 로그

- Cluster 로그: API 서버/컨트롤러매니저/스케줄러, 이벤트 로그
- Node 로그: Journal 로그(kubelet, containerd) 등 `journalctl`로 조회
- Pod 로그: 컨테이너 애플리케이션 로그 `kubectl`로 조회



<https://matthewpalmer.net/kubernetes-app-developer/articles/kubernetes-networking-guide-beginners.html>

```
$ journalctl -fu kubelet
```

```
...
```

```
Oct 09 15:59:37 controlplane kubelet[1568]: E1009 15:59:37.519108 1568 kuberuntime_container.go:896...
Oct 09 15:59:38 controlplane kubelet[1568]: I1009 15:59:38.494444 1568 scope.go:117] "RemoveContain...
```

```
$ kubectl logs pod1
```

```
[INFO] Starting the application...
```

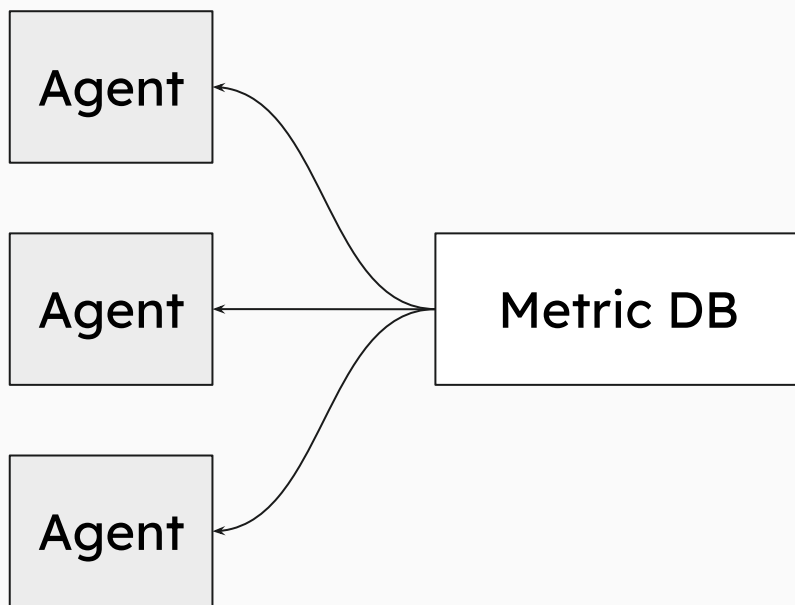
```
[INFO] Successfully connected to the DB.
```

```
[ERROR] Failed to connect to the API.
```

# 메트릭DB vs 로그DB

메트릭 수집

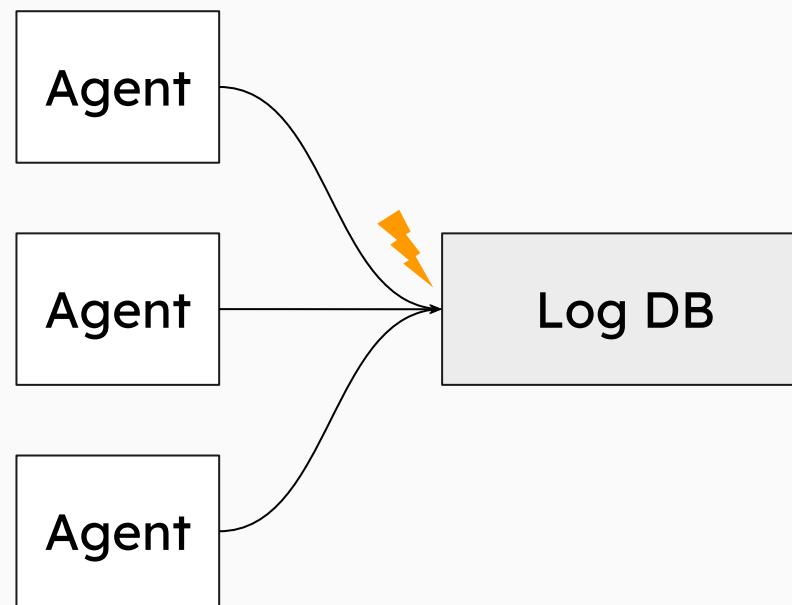
메트릭 저장



MetricDB가 수집주기 조절

로그 수집

로그 저장



과부하 발생 가능성

# Lethe

- 경량 Log DB
- DB이지만 DB가 아니다?
  - 단순히 파일로 저장
  - Fluent Bit 기반
- 기능: 저장, 조회, 로테이션
- LetheQL
  - 필터 연산자
  - 파이프라인 연결





# Fluentd vs Fluent Bit

	Fluentd	Fluent Bit
Scope	Containers / Servers	Embedded Linux / Containers / Servers
Language	C & Ruby	C
Memory	> 60MB	~1MB
Performance	Medium Performance	High Performance
Dependencies	Built as a Ruby Gem, it requires a certain number of gems.	Zero dependencies, unless some special plugin requires them.
Plugins	More than 1000 external plugins are available	More than 100 built-in plugins are available
License	<a href="#">Apache License v2.0</a>	<a href="#">Apache License v2.0</a>

로그DB를 경량으로 만든 비결입니다?

Fluent Bit을 로그수집기로 쓰면 로그DB가  
경량화된다는 말인가? (X)

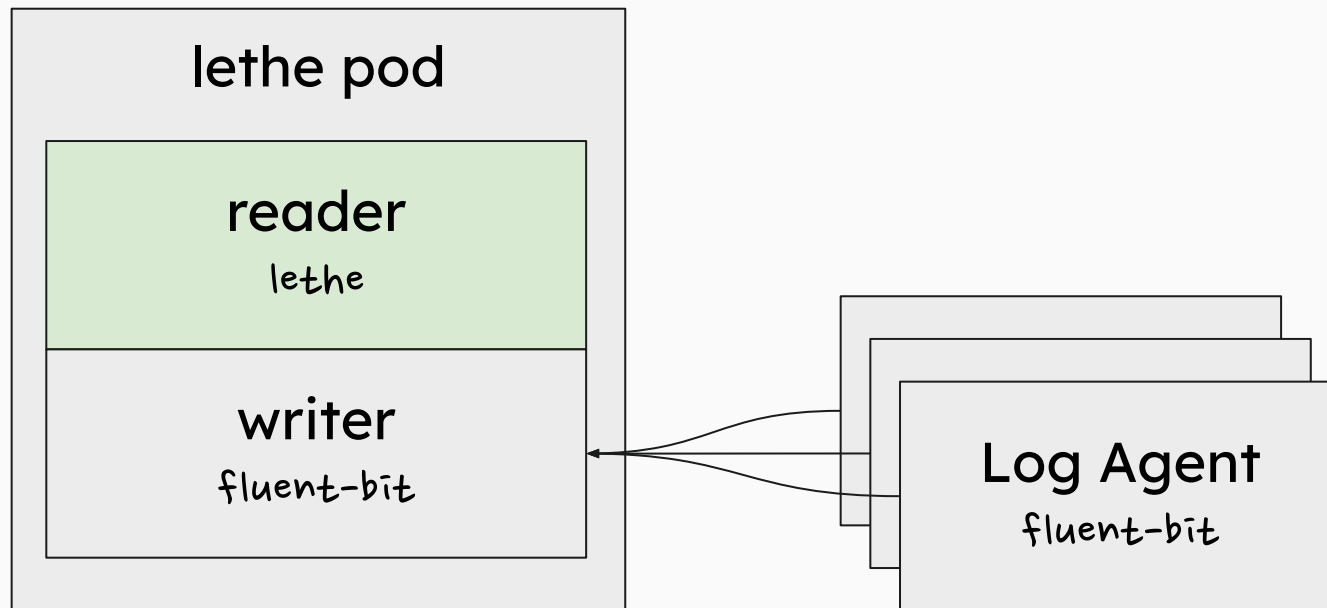
Fluent Bit을 잘 설정해서 로그를  
수집단기에서 경량화해서 로그DB의 부담을  
줄였다는 말인가? (△)

<https://docs.fluentbit.io/manual/about/fluentd-and-fluent-bit>

# Lethe 구성도 (컨테이너 수준)

사이드카 구조  
 $\text{lethe}(\text{pod}) = \text{lethe} + \text{fluent-bit}$

lethe는 쉬운 일(읽기)만 하고,  
fluent-bit이 힘든 일(쓰기)을 한다.



‘수집-저장’ 과정만 보면  
fluent-bit끼리 데이터를 주고 받는 것

# 로그 저장 방식

## 파일경로

node/	node01/	2024-09-24_15.log
	node02/	2024-09-24_15.log
pod/	namespace01/	2024-09-24_15.log
	namespace02/	2024-09-24_15.log

노드/네임스페이스 디렉토리 + 한시간.log

## 파일내용

2024-09-24T15:00:01Z[node01|kubelet] hello world  
2024-09-24T15:00:02Z[node01|kubelet] lorem ipsum  
2024-09-24T15:00:03Z[node01|containerd] hello world

날짜시간[노드|서비스] 로그내용

2024-09-24T15:00:01Z[namespace01|pod1|container1] hello  
2024-09-24T15:00:02Z[namespace01|pod1|container2] lorem  
2024-09-24T15:00:03Z[namespace01|pod2|container1] hello

날짜시간[네임스페이스|파드|컨테이너] 로그내용

이런 것도 DB라고 할 수 있나?

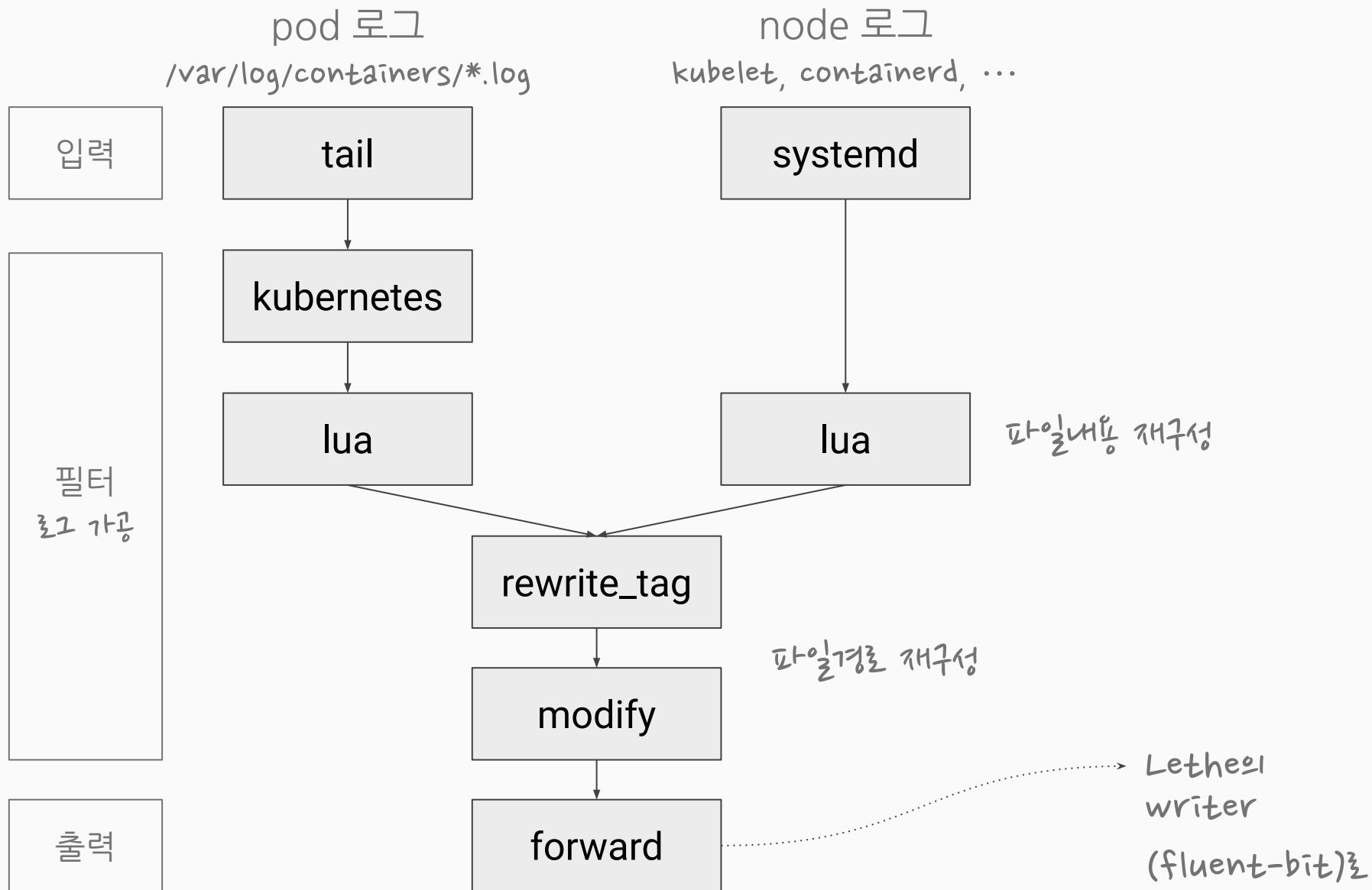
- 1) DB 기능을 하면 DB
- 2) 일반적 DB는 아니고 flat-file DB

다른 특징은?

- label 정보 없음 (대부분 pod 이름으로 로그 조회)
- 시스템 의존성 없음 (그냥 파일)

# 로그 수집

로그수집기  
(fluent-bit)  
내부 파이프라인



<https://github.com/kuoss/helm-charts/blob/venti-stack-0.2.10/charts/venti-stack/values.yaml#L16-L91>

<https://github.com/kuoss/helm-charts/blob/venti-stack-0.2.10/charts/venti-stack/templates/lethe/cm.yaml#L28-L38>



# LetheQL #1 기본 쿼리

Lethe에서 로그를 조회하기 위한 쿼리 언어  
PromQL(Prometheus Query Language)의 확장판 (소스코드 재사용)

```
$ kubectl logs nginx-66b6c48dd5-abcde  
127.0.0.1 - - [08/Sep/2024:12:34:56 +0000] "GET / HTTP/1.1" 200 612 "-" ...
```

➔ `pod{namespace="default",pod="nginx-66b6c48dd5-abcde"}`

```
$ kubectl logs deploy/nginx  
127.0.0.1 - - [08/Sep/2024:12:34:56 +0000] "GET / HTTP/1.1" 200 612 "-" ...
```

➔ `pod{namespace="default",pod=~"nginx-.*"}`

# LetheQL #2 필터 연산자

|= 문자열 포함  
!= 문자열 미포함  
|~ 정규식 포함  
!~ 정규식 미포함

필터 연산자 4종 도입  
(Loki의 LogQL에서 참고)  
파이프라인 연결 가능

```
$ kubectl logs deploy/nginx | grep -v POST | grep favicon  
127.0.0.1 - - [08/Sep/2024:12:34:57 +0000] "GET /favicon.ico ..."
```

→ `pod{namespace="default",pod=~"nginx-.*"} != "POST" != "favicon"`

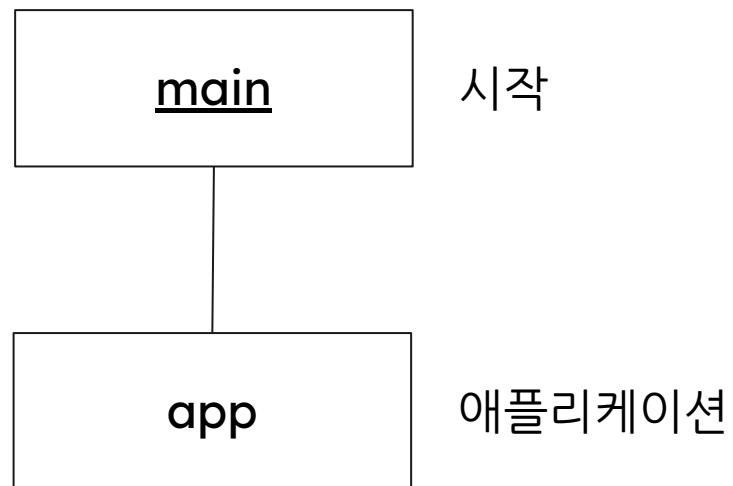
```
$ kubectl logs deploy/nginx | grep -v -E '(GET|POST)' | grep -E 'user/[0-9]+'  
127.0.0.1 - - [08/Sep/2024:12:34:56 +0000] "DELETE /user/456 ..."
```

→ `pod{namespace="default",pod=~"nginx-.*"} !~ "(GET|POST)" !~ "user/[0-9]+"`

# III Lethe 리뷰

Go로 개발된 부분을 살펴보자.

# Lethe 패키지 구성도



개발을 시작하자...



# main

```
[main.go]
```

```
var Version = "development"
```

```
func main() {  
    if err := app.Run(Version); err != nil {  
        log.Fatalf("Failed to run app: %v", err)  
    }  
}
```

애플리케이션 실행

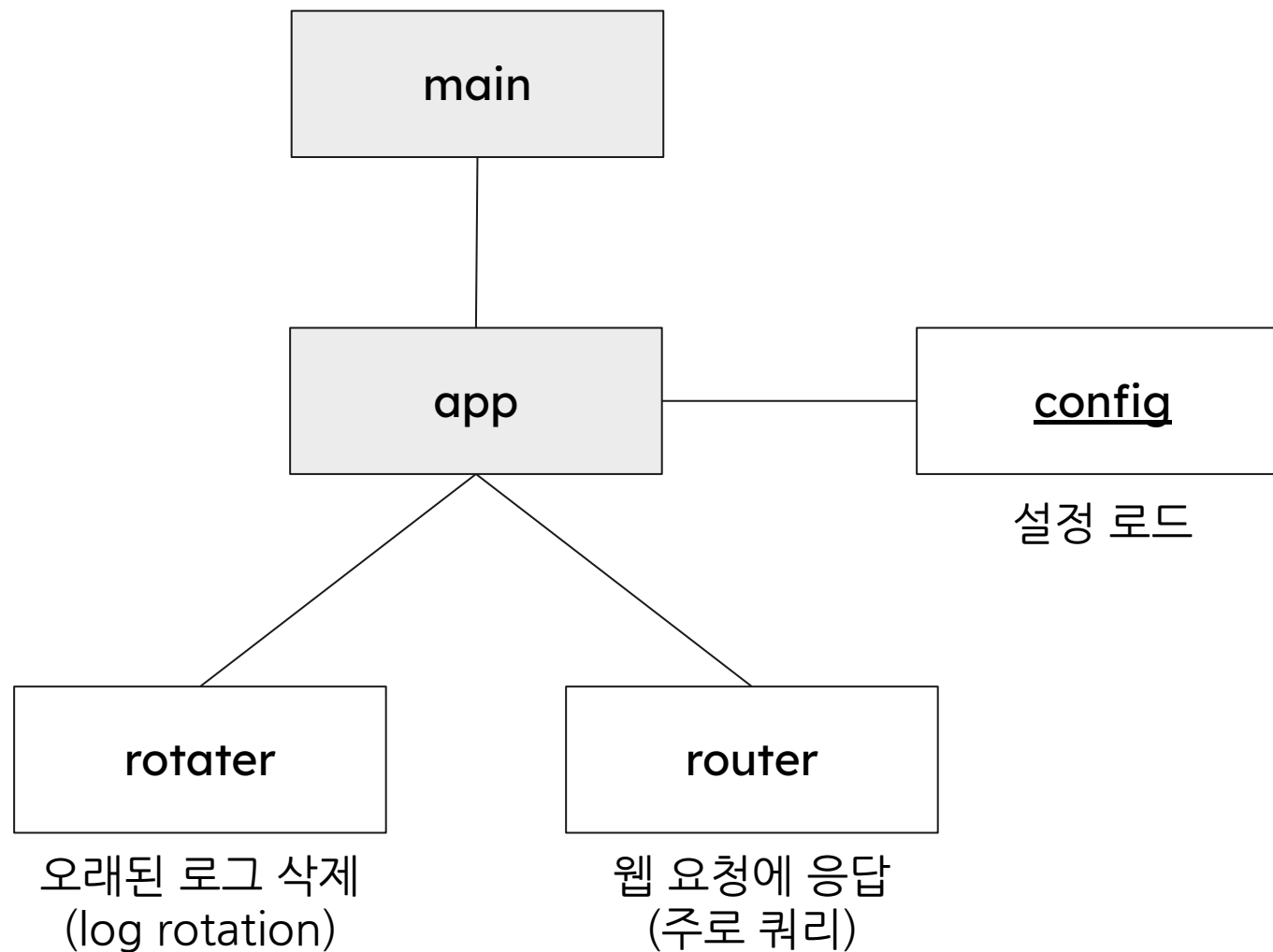
- version 빌드시 주입

- Exit는 여기에서만,

나머지 패키지는 err 처리

```
go build -ldflags "-X main.Version=v1.0.0"
```

# Lethe 패키지 구성도



## config

[etc/lethe.example.yaml] 설정파일 예시

retention:

time: 15d 보관기간

size: 1000GB 보관용량

storage:

log\_data\_path: /data/log 로그 데이터 경로

cf. prometheus flags

--storage.tsdb.path

--storage.tsdb.retention.time

--storage.tsdb.retention.size

※ case 선택

logDataPath: camel case (k8s)

log\_data\_path: snake case (prom) ✓

※ 실제 설정파일 경로 (venti-stack 설치시)

/app/etc/lethe.yaml

# config

```
[config_test.go]
    want := &Config{
        Version: "test",
        Query:    Query{
            Limit:    1000,
            Timeout: 20 * time.Second,
        },
        Retention: Retention{
            Time:          15 * 24 * time.Hour,
            Size:           1000 * 1024 * 1024 * 1024,
            SizeStrategy:   "file", // file | disk
            RotationInterval: 20 * time.Second,
        },
        Storage: Storage{LogDataPath: "/data/log"},
        Web: Web{
            ListenAddress: ":6060",
            GinMode:       "release",
        },
    }
}
```

버전 (빌드시 주입)

쿼리결과 최대개수

쿼리 타임아웃

## ※ sizeStrategy

file: 파일의 총합 (=du)

- 느림, 실제 데이터용량 ✓

disk: 디스크 용량 (=df)

- 빠름, 실제 데이터용량과 다름

보관기간 15일

보관용량 1000GiB

용량측정방법

로테이션 간격

로그데이터경로 /data/log

## ※ port

🔥 9090

💧 6060 ✓

☁️ 3030

## config

```
[config.go]
func New(version string) (*Config, error) {
    v := viper.New()
    v.SetConfigName("lethe")
    v.SetConfigType("yaml")
    v.AddConfigPath(filepath.Join(".", "etc"))
    v.AddConfigPath(filepath.Join("../", "etc"))

    // Set default values
    v.SetDefault("query.limit", 1000)
    ...

    // units
    retentionSizeString := v.GetString("retention.size")
    retentionSize, err := parseSize(retentionSizeString)
    if err != nil { ... }

    retentionTimeString := v.GetString("retention.time")
    retentionTime, err := model.ParseDuration(retentionTimeString)
    if err != nil { ... }
```

viper ✓  
- Pflags 지원  
- k8s 계열 프로젝트에서 사용

```
type Retention struct {
    Size int64   인사이트
    Time time.Duration
    ...
}
```

## config

```
import (  
    ...  
    "github.com/alecthomas/units"  
    "github.com/prometheus/common/model"  
    "k8s.io/apimachinery/pkg/api/resource"  
)  
  
func TestParseSize(t *testing.T) {  
    size1, _ := resource.ParseQuantity("1Gi") // 1GiB  
    size2, _ := units.ParseBase2Bytes("1GB") // 1GiB  
    assert.Equal(t, int64(1073741824), size1.Value())  
    assert.Equal(t, int64(1073741824), int64(size2))  
}  
  
func TestParseTime(t *testing.T) {  
    time1, _ := model.ParseDuration("15d") // 15일  
    assert.Equal(t, 15*24*time.Hour, time.Duration(time1))  
}
```

```
type Retention struct {  
    Size int64 비이트  
    Time time.Duration  
    ...  
}
```

resource 패키지

- kubernetes
- 단위: Ki, Mi, Gi, ...

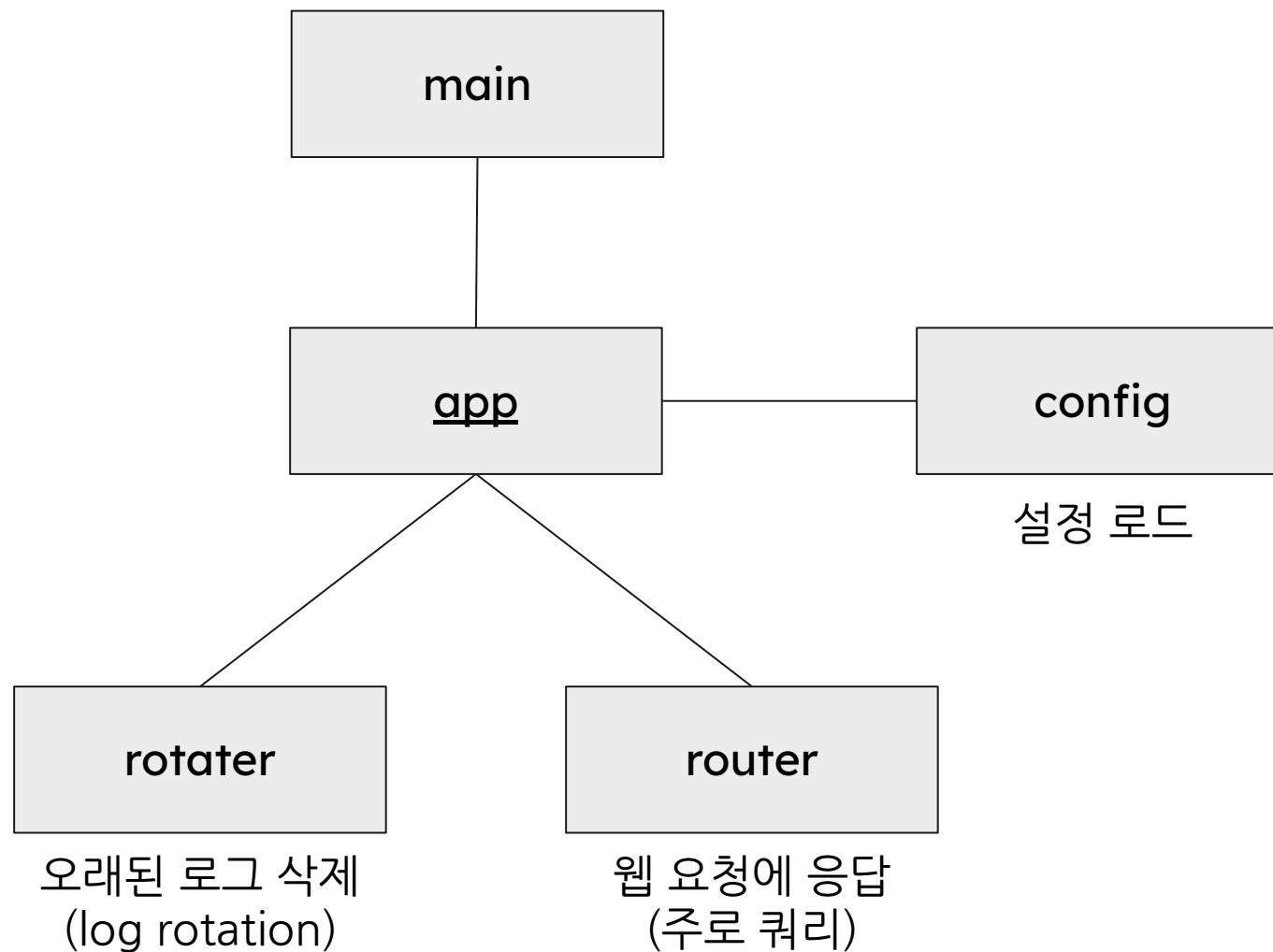
units 패키지 ✓

- prometheus (v3.0.0)
- 단위: KB, MB, GB, ...

model 패키지 ✓

- d, w, y 처리 가능

# Lethe 패키지 구성도



app

```
[app.go]
```

```
func Run(version string) error {  
    cfg, err := config.New(version)  
    if err != nil { return ... }  
    logger.Infof("Loaded configuration: %v", cfg)  
  
    fileService, err := fileservice.New(cfg)  
    if err != nil { return ... }  
    logService := logservice.New(cfg, fileService)  
    queryService := queryservice.New(cfg, logService)  
  
    myRotator := rotator.New(cfg, fileService)  
    myRotator.Start()  
  
    myRouter := router.New(cfg, fileService, queryService)  
    return myRouter.Run()  
}
```

설정 로드

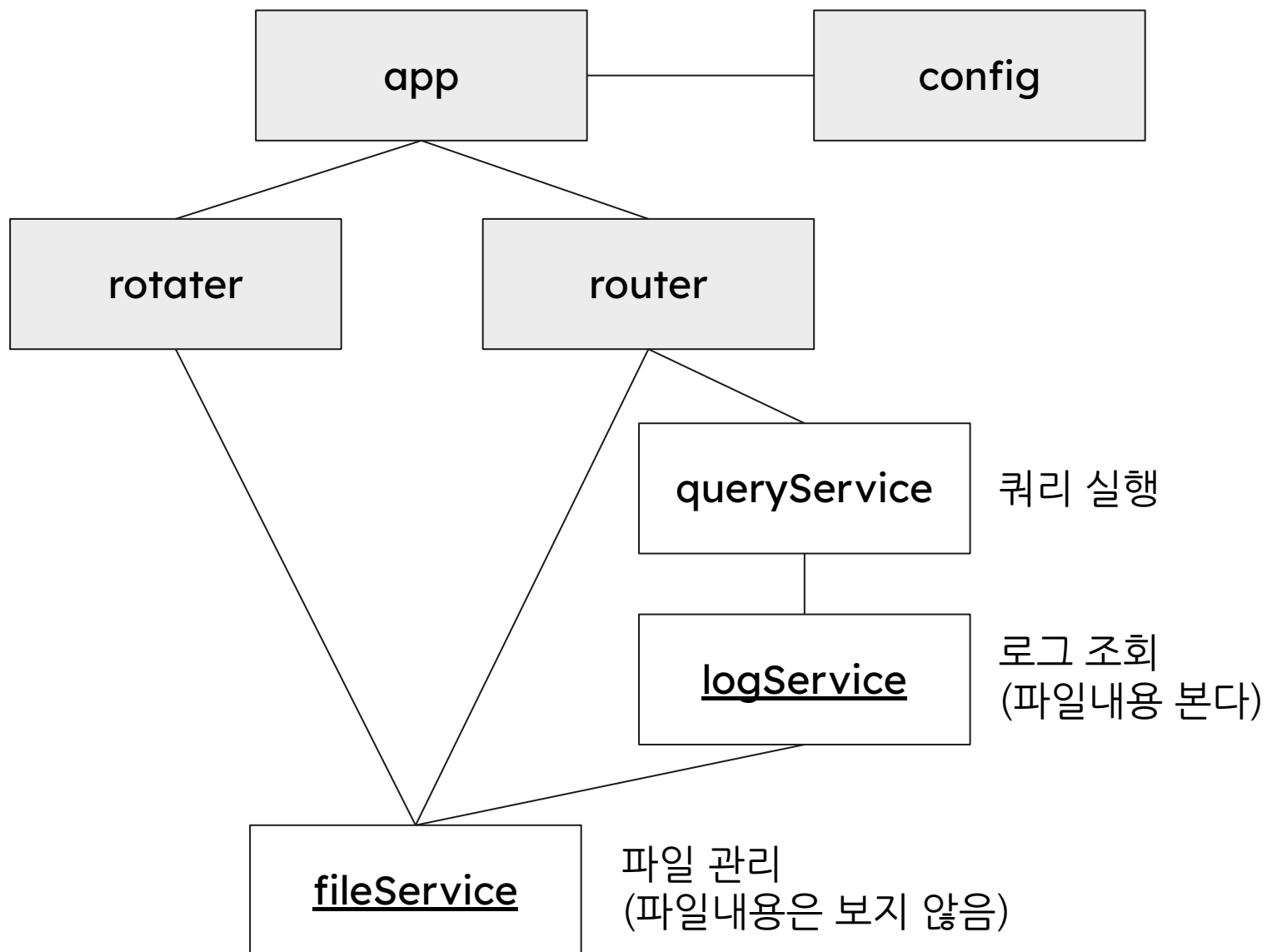
서비스 생성  
(파일, 로그, 쿼리)

rotator 시작

router 실행



# Lethe 패키지 구성도



# FileService vs LogService

파일경로 → FileService (파일 관리)

node/	node01/	2024-09-24_15.log
	node02/	2024-09-24_15.log
pod/	namespace01/	2024-09-24_15.log
	namespace02/	2024-09-24_15.log

노드/네임스페이스 디렉토리 + 한시간.log

파일내용 → LogService (로그 조회)

```
2024-09-24T15:00:01Z[node01|kubelet] hello world
2024-09-24T15:00:02Z[node01|kubelet] lorem ipsum
2024-09-24T15:00:03Z[node01|containerd] hello world
```

날짜시간[노드|서비스] 로그내용

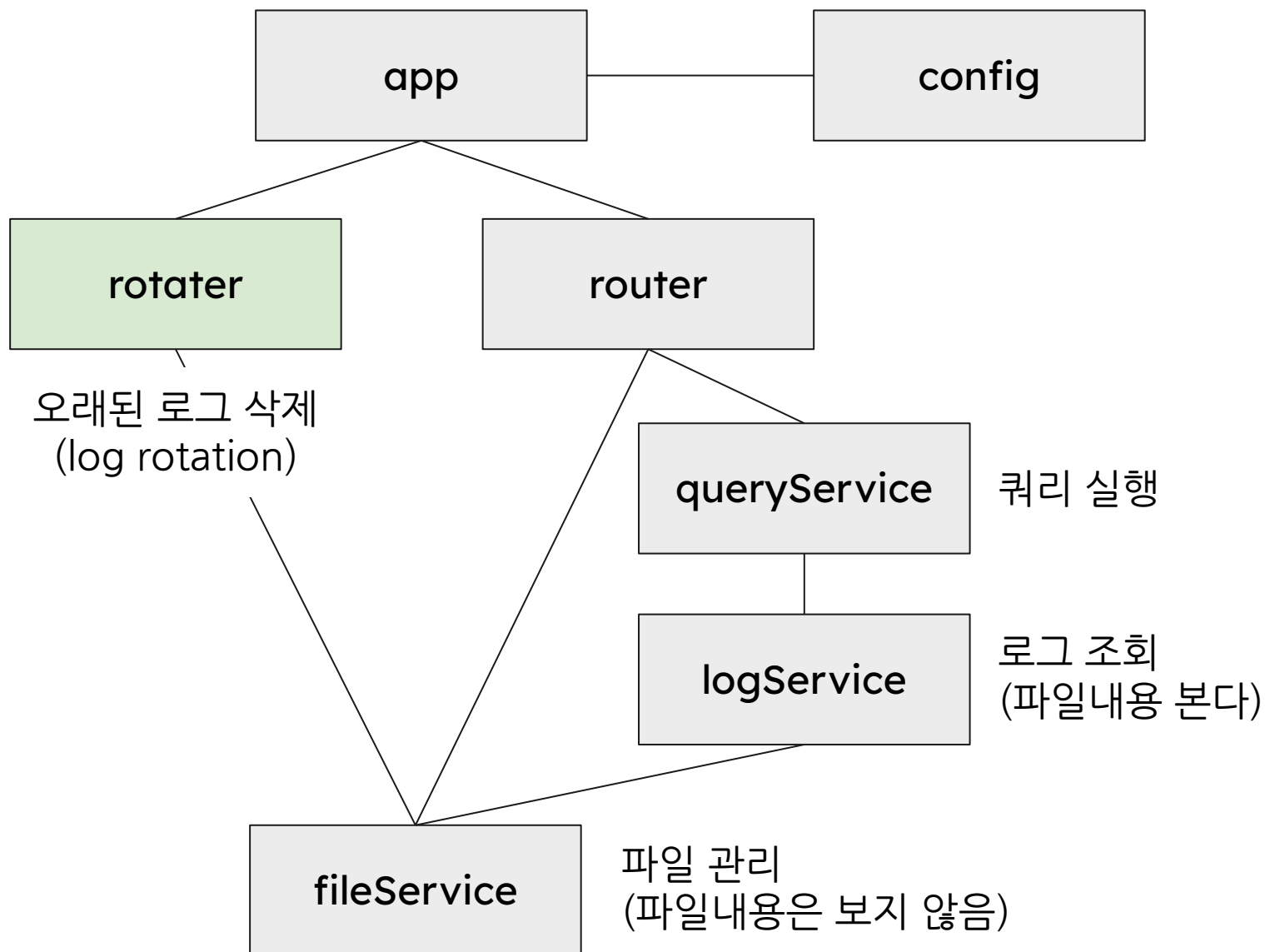
```
2024-09-24T15:00:01Z[namespace01|pod1|container1] hello
2024-09-24T15:00:02Z[namespace01|pod1|container2] lorem
2024-09-24T15:00:03Z[namespace01|pod2|container1] hello
```

날짜시간[네임스페이스|포드|컨테이너] 로그내용

※ 백엔드(Lethe)에서 날짜시간은 모두 UTC 기준 (k8s, prometheus) ✓

cf. 지역별 시간대(예: KST)는 최종 단계(프론트엔드, venti)에서 처리 (브라우저/시스템 시간)

# Lethe 패키지 구성도



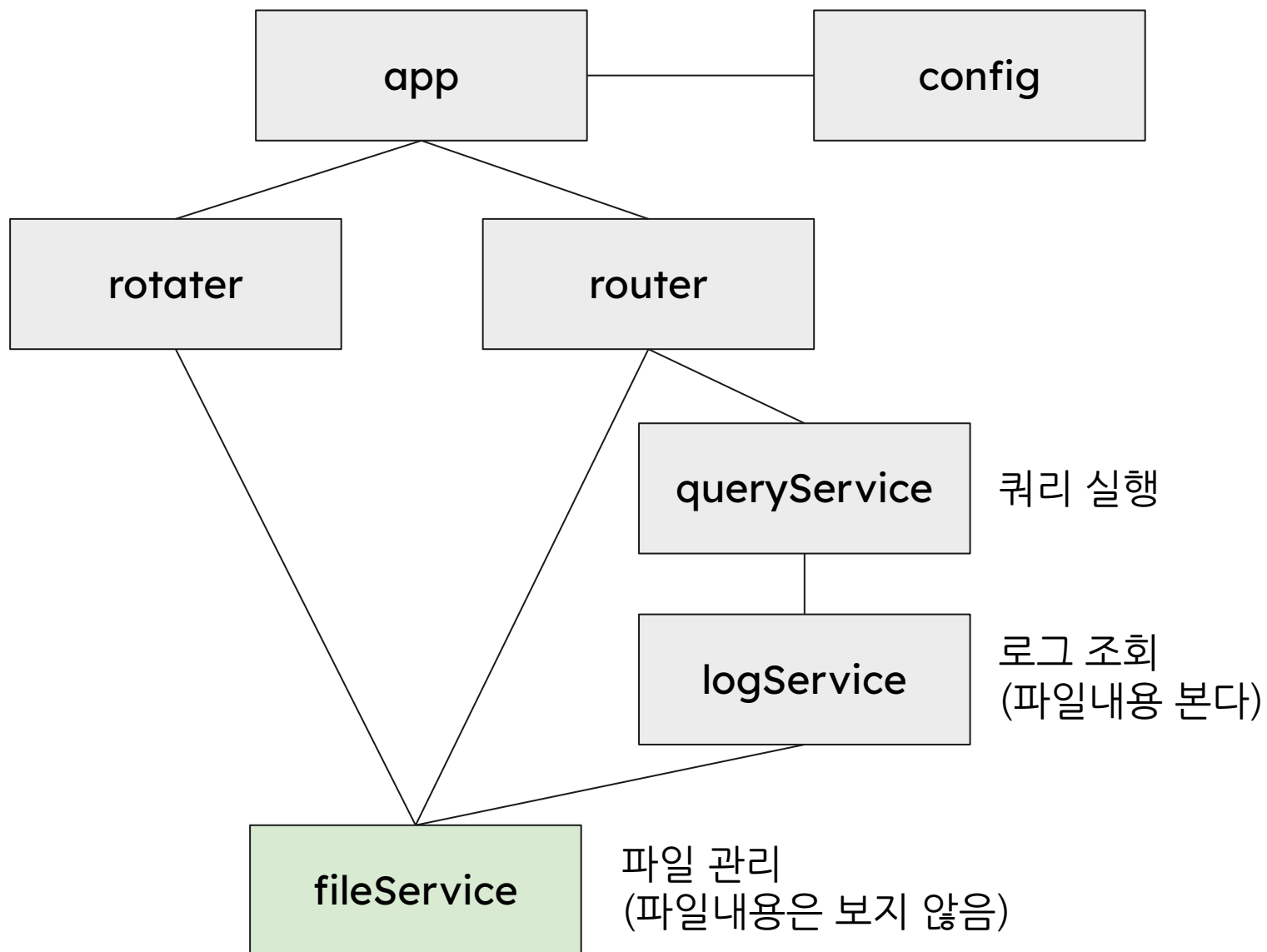
## rotator

```
[rotator.go]
func (r *Rotator) Start() {           rotator 시작
    go r.rotatePeriodically()
}

func (r *Rotator) rotatePeriodically() {   정기적으로 rotate
    for {
        r.Rotate()
        logger.Infof("sleep %s", r.interval)
        time.Sleep(r.interval)
    }
}

func (r *Rotator) Rotate() {           한번 rotate
    if err := r.fileService.DeleteByAge(); err != nil {   보관기간 지난 것 삭제
        logger.Errorf("DeleteByAge err: %v", err)
    }
    if err := r.fileService.DeleteBySize(); err != nil {   보관용량 넘는 것 삭제
        logger.Errorf("DeleteBySize err: %v", err)
    }
}
```

# Lethe 패키지 구성도



# File Service

[fileservice 패키지 함수]

```
func New(cfg *config.Config) (*FileService, error)
```

```
func (s *FileService) DeleteByAge() error
```

보관기간 지난 것 삭제

```
func (s *FileService) DeleteBySize() error
```

보관용량 넘는 것 삭제

```
func (s *FileService) List(subpath string) ([]string, error)
```

파일 목록

```
func (s *FileService) ListFiles() ([]LogFile, error)
```

파일 목록 (+정보)

```
func (s *FileService) ListLogDirs() []LogDir
```

디렉토리 목록

```
func (s *FileService) ListLogDirsWithSize() []LogDir
```

디렉토리 목록 (+용량)

```
func (s *FileService) ListTargets() []LogDir
```

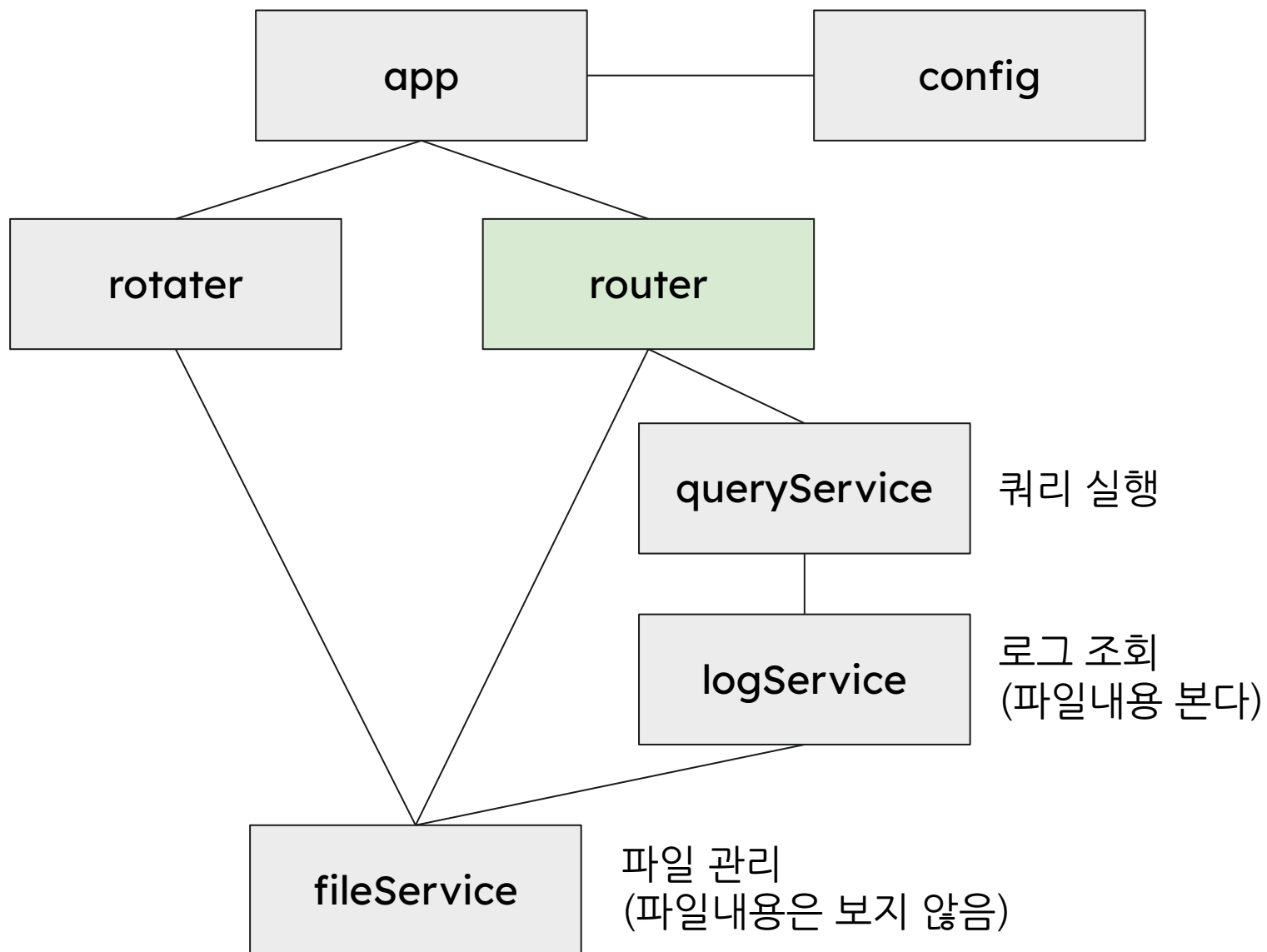
로그 수집 대상 목록

```
func (s *FileService) Scanner(subpath string) (*bufio.Scanner, error)
```

파일 읽기용 스캐너 제공

(LogService에 제공)

# Lethe 패키지 구성도



## router

```
[router.go]
func New(cfg *config.Config, fileService *fileservice.FileService,
queryService *queryservice.QueryService) *Router {
    router := &Router{cfg, fileService, queryService}
    router.setupRouter()
    return router
}
```

```
func (r *Router) setupRouter() {
    gin.SetMode(r.config.Web.GinMode)
    ginRouter := gin.New()
    ginRouter.GET("/-/healthy", r.Healthy)
    ginRouter.GET("/-/ready", r.Ready)
    ginRouter.GET("/api/v1/metadata", r.Metadata)
    ginRouter.GET("/api/v1/query", r.Query)
    ginRouter.GET("/api/v1/query_range", r.QueryRange)
    ginRouter.GET("/api/v1/targets", r.Target)
    r.ginRouter = ginRouter
}
```

```
func (r *Router) Run() error {
    return r.ginRouter.Run(r.config.Web.ListenAddress)
}
```

상태 체크

메타데이터 조회

최근 로그 조회

시간범위 로그 쿼리

타겟(로그수집대상) 조회

cf. prometheus

- instant query (vector)
- range query (matrix)





## router

```
[router/query.go]
func (r *Router) Query(c *gin.Context) {
    qs := c.Query("query")
    r.query(c, qs, model.TimeRange{})
}
```

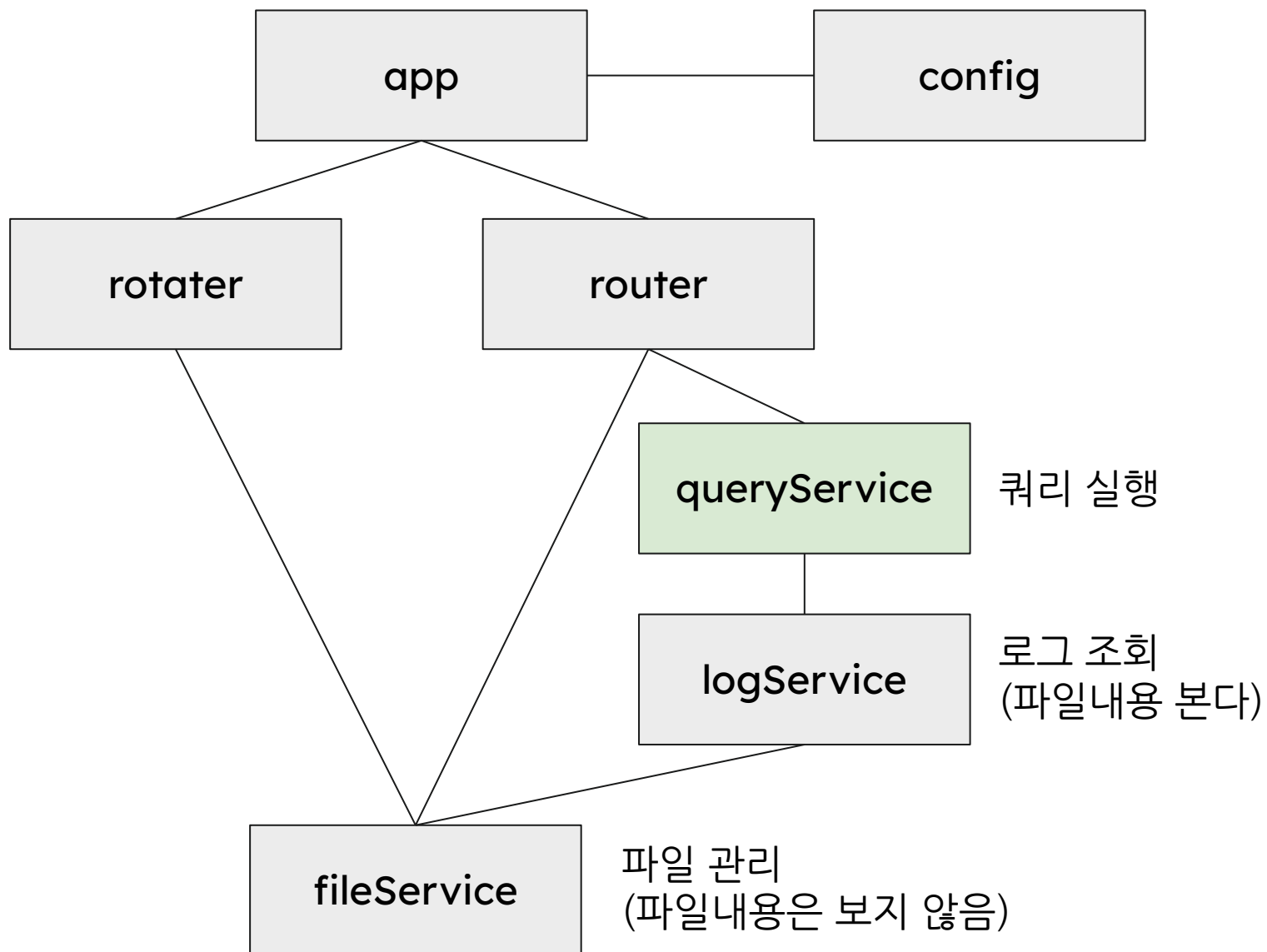
최근 로그 조회

```
func (r *Router) QueryRange(c *gin.Context) {
    qs := c.Query("query")
    start := c.Query("start")
    end := c.Query("end")
    r.query(c, qs, model.TimeRange{Start: toTime(start), End: toTime(end)})
}
```

시간범위 로그 조회

```
func (r *Router) query(c *gin.Context, qs string, tr model.TimeRange) {
    result := r.queryService.Query(c.Request.Context(), qs, tr)
    if result.Err != nil { ... }
    if result.Value.Type() == model.ValueTypeLog { ... }
    ...
}
```

# Lethe 패키지 구성도



## Query Service

```
[queryservice.go]
func New(cfg *config.Config, logService *logservice.LogService) *QueryService {
    return &QueryService{
        engine: letheql.NewEngine(cfg, logService), 엔진 생성·보유
    }
}

func (s *QueryService) Query(ctx context.Context, qs string, tr model.TimeRange)
*letheql.Result {
    if reflect.ValueOf(tr).IsZero() {
        now := clock.Now()
        tr = model.TimeRange{now.Add(-1 * time.Minute), now}
    }
    qry, err := s.engine.NewRangeQuery(qs, tr.Start, tr.End)
    if err != nil { ... }
    res := qry.Exec(ctx)
    if res.Err != nil { ... }
    return res
}
```

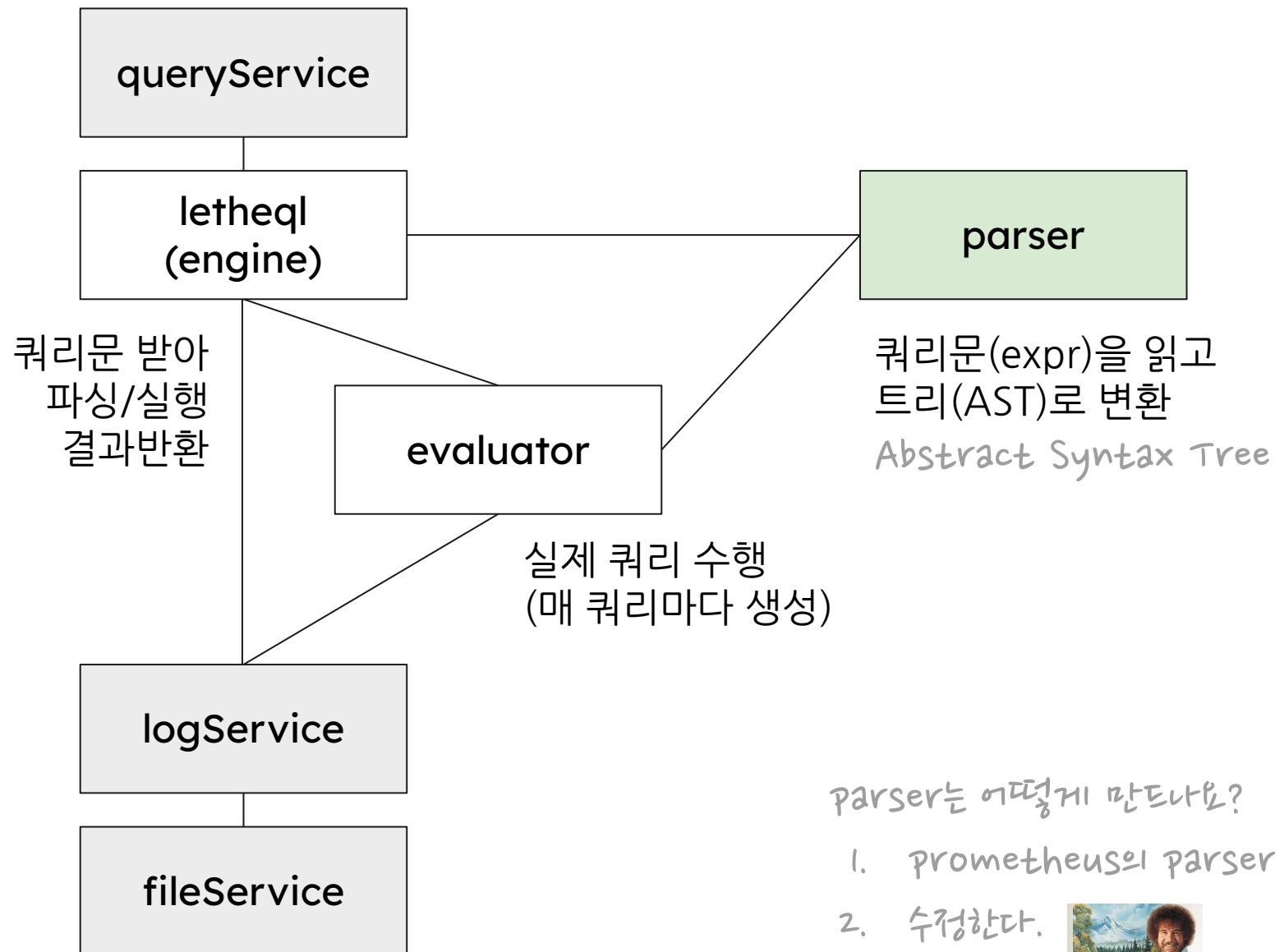
1. 쿼리 생성

2. 쿼리 실행

3. 결과 반환

# LetheQL

# LetheQL 구성도



LetheQL 시스템

- parser
- engine
- evaluator

parser는 어떻게 만드나요?

1. prometheus의 parser를 가져온다.
2. 수정한다.



참 실조?

# goyacc

Go의 Yacc

(Yet Another compiler-compiler)

- Go 언어를 위한 파서 생성기
- Go로 작성되었다.
- Go로 작성된 파서를 생성한다.

## 사용방법

### 1. goyacc 설치

```
$ go install golang.org/x/tools/cmd/goyacc@latest
```

### 2. .y (yacc) 파일 작성

### 3. goyacc 실행 (파서 생성)

```
$ goyacc -o expr.go expr.y
```

공식 문서 및 예시

<https://pkg.go.dev/golang.org/x/tools/cmd/goyacc>

<https://cs.opensource.google/go/x/tools/+refs/tags/v0.26.0:cmd/goyacc/testdata/expr/expr.y>

[expr.y]

```
%type    <num>    expr expr1 expr2 expr3
%token    '+' '-' '*' '/' '(' ')'
%token    <num>    NUM
%%
```

```
top:
    expr
    {
        if $1.IsInt() {
            fmt.Println($1.Num().String())
        } else {
            fmt.Println($1.String())
        }
    }
}
```

```
expr:
    expr1
|    '+' expr
    {
        $$ = $2
    }
|    '-' expr
    {
        $$ = $2.Neg($2)
    }
}
```



# Parser 만들기

```
[Makefile]
```

```
#### parser
```

```
.PHONY: parser-clone
```

```
parser-clone:
```

```
git clone -b $(PROMETHEUS_VERSION) --depth=1 https://github.com/prometheus/pro...
```

```
rm -rf letheql/parser
```

prometheus의 parser 코드를 가져온다.

```
mv prometheus/promql/parser letheql/
```

promql parser → letheql parser

```
rm -rf prometheus
```

```
.PHONY: parser-generate
```

```
parser-generate: goyacc
```

```
$(GOYACC) -o letheql/parser/generated_parser.y.go letheql/parser/generated_par...
```

```
rm -f letheql/parser/y.output
```

y 파일을 → go 코드 생성

( generated\_parser.y → generated\_parser.y.go )

```
.PHONY: parser-test
```

```
parser-test:
```

```
go test -failfast github.com/kuoss/lethe/letheql/parser
```

```
go test -failfast github.com/kuoss/lethe/letheql/parser_test
```

parser 테스트 코드 실행

# LetheQL 필터연산자 4종

**PromQL** `http_requests_total{method!= "GET", status!~ "4.."}`

**LetheQL** `pod{namespace="default", pod="nginx"} |= "error" |~ "4.."`

연산자	기능	이름	PromQL에 있는가?
!=	문자열 미포함	NEQ (같지않음)	O
!~	정규식 미포함	NEQ_REGEX (같지않음_정규식)	O
=	문자열 포함	PIPE_EQL (파이프_같음)	X
~	정규식 포함	PIPE_REGEX (파이프_정규식)	X



# Parser 만들기

[generated\_parser.y] yacc 파일

```
// Operators.
```

```
MUL
```

```
NEQ
```

```
NEQ_REGEX
```

```
PIPE_EQL
```

```
PIPE_REGEX
```

연산자 이름 추가

```
POW
```

```
SUB
```

```
AT
```

```
ATAN2
```

# Parser 만들기

[generated\_parser.y] yacc 파일

```
// Operators.
```

```
MUL
```

```
NEQ
```

```
NEQ_REGEX
```

```
PIPE_EQL
```

```
PIPE_REGEX
```

```
POW
```

```
SUB
```

```
AT
```

```
ATAN2
```

연산자 이름 추가

goyacc

```
[generated_parser.y.go]  
const MUL = 57380  
const NEQ = 57381  
const NEQ_REGEX = 57382  
const PIPE_EQL = 57383  
const PIPE_REGEX = 57384  
const POW = 57385  
const SUB = 57386  
const AT = 57387  
const ATAN2 = 57388
```

# Parser 만들기

[generated\_parser.y]

```
// Operator precedence only works if each of those is listed separately.
```

```
...
```

```
| expr NEQ      bin_modifier expr { $$ = yylex.(*parser).newBinaryExpression...
```

```
| expr NEQ_REGEX bin_modifier expr { $$ = yylex.(*parser).newBinaryExpression...
```

```
| expr PIPE_EQL  bin_modifier expr { $$ = yylex.(*parser).newBinaryExpression...
```

```
| expr PIPE_REGEX bin_modifier expr { $$ = yylex.(*parser).newBinaryExpression...
```

이항연산자 3종 추가

이항표현식

(이항연산자를 포함한 표현식)

pod{} != "error"

왜 2개가 아니고 3개를 추가해야 하나?

왜 NEQ(!=)만 있고, NEQ\_REGEX(!~)는 없었을까?

- PromQL: {} 내부에서만 사용(레이블 매치) metric{foo!~"bar-.\*"}

- LetheQL: {} 외부에서도 사용(값 매치) pod{} !~ "bar-.\*"

[generated\_parser.y]

// Operators are listed with increasing precedence.

%left LOR

%left LAND LUNLESS

%left EQLC GTE GTR LSS LTE NEQ NEQ\_REGEX PIPE\_EQL PIPE\_REGEX

%left ADD SUB

%left MUL DIV MOD ATAN2

%right POW

연산자 우선순위

[lex.go]

```
var ItemTypeStr = map[ItemType]string{
    ...
    EQLC:      "=",
    NEQ:       "!=",
    LTE:       "<=",
    LSS:       "<",
    GTE:       ">=",
    GTR:       ">",
    EQL_REGEX: "=~",
    NEQ_REGEX: "!~",
    PIPE_EQL:  "|=",
    PIPE_REGEX: "|~",
    POW:       "^",
}
```

## Lexing (Lexical Analysis, 어휘 분석)

- 컴파일의 첫 번째 단계
- 소스 코드의 문자 단위 입력을 받아서 토큰(token) 단위로 변환하는 과정
- 소스 코드를 토큰으로 나누어 파서(Parser)에 넣기 위한 준비 작업

# Parser 만들기

```
[lex.go]
// lexStatements is the top-level state for lexing.
func lexStatements(l *Lexer) stateFn {
    ...
    case r == '!':
        if t := l.next(); t == '=' {
            l.emit(NEQ)
        } else if t == '~' {
            l.emit(NEQ_REGEX)
        } else {
            return l.errorf("unexpected character after '!': %q", t)
        }
    case r == '|':
        if t := l.next(); t == '=' {
            l.emit(PIPE_EQL)
        } else if t == '~' {
            l.emit(PIPE_REGEX)
        } else {
            return l.errorf("unexpected character after '|': %q", t)
        }
}
```

원래는 != 만 가능

!~ 추가

= 추가

~ 추가

# Parser 만들기

```
[lex.go]
func (i ItemType) IsComparisonOperator() bool {
    switch i {
    case EQLC, NEQ, LTE, LSS, GTE, GTR:
        return true
    default:
        return false
    }
}
```

```
func (i ItemType) IsFilterOperator() bool {
    switch i {
    case NEQ, NEQ_REGEX, PIPE_EQL, PIPE_REGEX:
        return true
    default:
        return false
    }
}
```

evaluator에서  
사용할 함수 작성

# Parser 만들기

[parse.go]

## AST 유효성 점검

```
func (p *parser) checkAST(node Node) (typ ValueType) {  
    ...  
    case *BinaryExpr:    이항표현식인 경우...  
        lt := p.checkAST(n.LHS) 좌항 점검  
        rt := p.checkAST(n.RHS) 우항 점검  
        ...  
  
        if rt != ValueTypeScalar && rt != ValueTypeVector && rt != ValueTypeString {  
            p.addParseErrf(n.RHS.PositionRange(), "binary expression must contain  
only scalar and instant vector and string types")  
        }  
        우항이 문자열인 경우에도 통과시키자.
```

Parser 구현 끝. 테스트는?



# Parser 테스트 #1 원래 PromQL에 있던 테스트 코드 수정

[parse\_test.go]

```
{  
    input: "1 !~ 1",  
    fail: true,  
    errMsg: `unexpected character after '!': '~`,  
},
```

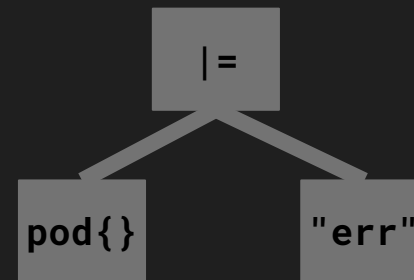
!~ 연산자는 checkAST에 의해 error였으나...

```
{  
    input: "1 !~ 1",  
    expected: &BinaryExpr{  
        Op: NEQ_REGEX,  
        LHS: &NumberLiteral{Val: 1, PosRange: PositionRange{Start: 0, End: 1}},  
        RHS: &NumberLiteral{Val: 1, PosRange: PositionRange{Start: 5, End: 6}},  
    },  
},
```

이제는 이항연산자로 인식됨

## Parser 테스트 #2 LetheQL을 위해 새로 작성한 테스트 코드

```
[parser_test.go]
{
    input: `pod{namespace="ns1"} |= "err"`,
    want: &parser.BinaryExpr{
        Op: parser.PIPE_EQL,
        LHS: &parser.VectorSelector{
            Name: "pod",
            LabelMatchers: []*labels.Matcher{
                parser.MustLabelMatcher(labels.MatchEqual, "namespace", "ns1"),
                parser.MustLabelMatcher(labels.MatchEqual, commonModel.MetricNameLabel, "pod")},
            PosRange: parser.PositionRange{Start: 0, End: 20}},
        RHS: &parser.StringLiteral{
            Val: "err",
            PosRange: parser.PositionRange{Start: 24, End: 29}},
    },
},
```



## Parser 테스트 #3 조금 더 복잡한 케이스

[parser\_test.go]

input: `pod{namespace="default",pod="nginx"} |= "err" |~ "4.."`,

want: &parser.BinaryExpr{

Op: parser.PIPE\_REGEX,

LHS: &parser.BinaryExpr{

Op: parser.PIPE\_EQL,

LHS: &parser.VectorSelector{

Name: "pod",

LabelMatchers: []\*labels.Matcher{

parser.MustLabelMatcher(labels.MatchEqual, "namespace", "default"),

parser.MustLabelMatcher(labels.MatchEqual, "pod", "nginx"),

parser.MustLabelMatcher(labels.MatchEqual, commonModel.MetricNameLabel, "pod")},

PosRange: parser.PositionRange{Start: 0, End: 36},

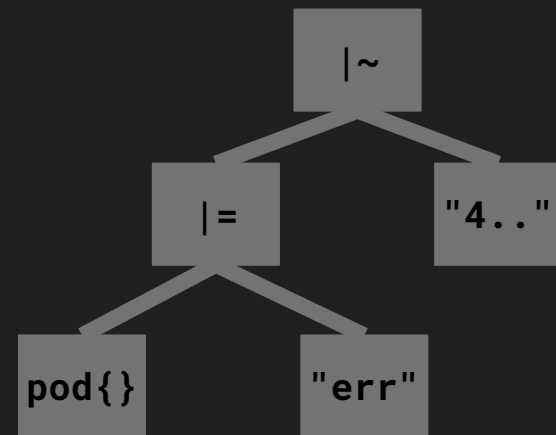
},

RHS: &parser.StringLiteral{Val: "err", PosRange: parser.PositionRange{Start: 40, End: 45}},

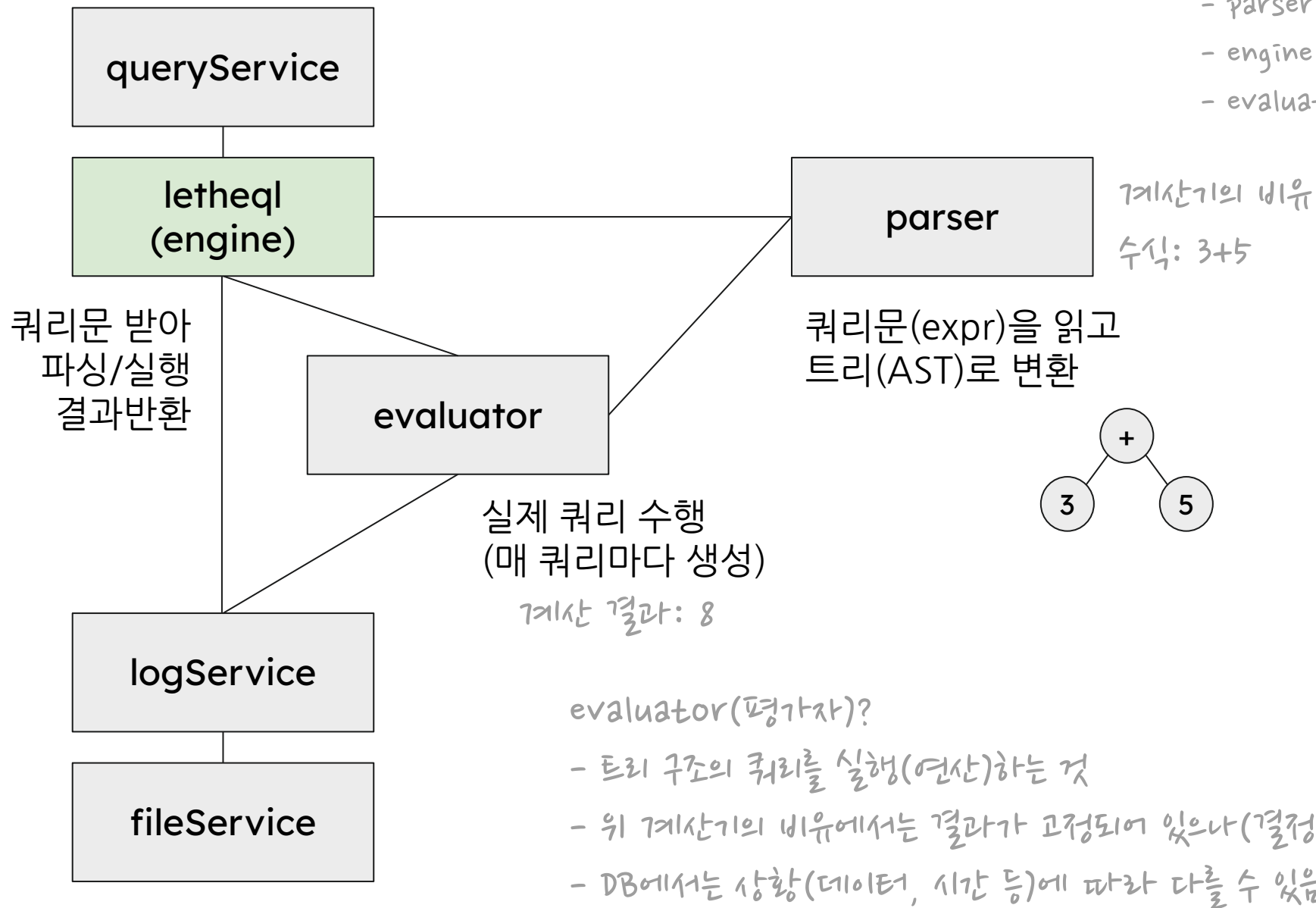
},

RHS: &parser.StringLiteral{Val: "4..", PosRange: parser.PositionRange{Start: 49, End: 54}},

},



# LetheQL 구성도



# LetheQL Engine

```
[engine.go]
func NewEngine(cfg *config.Config, logService *logservice.LogService) *Engine {
    return &Engine{cfg, logService}
}

func (ng *Engine) NewRangeQuery(qs string, start, end time.Time) (Query, error) {
    expr, err := parser.ParseExpr(qs)
    if err != nil { ... }
    qry := &query{
        q:    qs,
        stmt: &parser.EvalStmt{Expr: expr, Start: start, End: end},
        ng:    ng,
    }
    return qry, nil
}
...
```

1. 쿼리 생성

쿼리문자열  
평가문  
엔진

# [LetheQL] Query란 무엇인가?

[query.go]

```
type Query interface {  
    Cancel()  
    Exec(ctx context.Context) *Result  
    Statement() parser.Statement  
    String() string  
}
```

```
type query struct {  
    q      string      쿼리문자열  
    stmt   parser.Statement 평가문  
    cancel func()     취소함수  
    ng     *Engine     엔진  
}
```

```
func (q *query) Cancel() {  
    if q.cancel != nil {  
        q.cancel()  
    }  
}
```

```
func (q *query) Exec(ctx context.Context) *Result {  
    value, warnings, err := q.ng.exec(ctx, q)  
    return &Result{err, value, warnings}  
}
```

```
func (q *query) Statement() parser.Statement {  
    return q.stmt  
}
```

```
func (q *query) String() string {  
    return q.q  
}
```

2. 쿼리 실행

3. 결과 반환

# [LetheQL] Result, Value

[letheql/value.go]

```
type Result struct {  
    Err      error  
    Value     parser.Value  
    Warnings model.Warnings  
}
```

쿼리결과 { 값, 에러, 경고 }

[letheql/parser/value.go]

```
type Value interface {  
    Type() ValueType  
    String() string  
}
```

```
type ValueType string
```

```
const (  
    ValueTypeNone    ValueType = "none"  
    ValueTypeVector  ValueType = "vector"  
    ValueTypeScalar  ValueType = "scalar"  
    ValueTypeMatrix  ValueType = "matrix"  
    ValueTypeString  ValueType = "string"  
)
```

# [LetheQL] Value에 해당하는 것

```
root@wsl:~/go/src/lethe/letheql# grep -rn 'func.*Type().*ValueType.\+{' | grep ^parser
parser/ast.go:224:func (e *AggregateExpr)      Type() ValueType { return ValueTypeVector }
parser/ast.go:225:func (e *Call)                Type() ValueType { return e.Func.ReturnType }
parser/ast.go:226:func (e *MatrixSelector)      Type() ValueType { return ValueTypeMatrix }
parser/ast.go:227:func (e *SubqueryExpr)        Type() ValueType { return ValueTypeMatrix }
parser/ast.go:228:func (e *NumberLiteral)       Type() ValueType { return ValueTypeScalar }
parser/ast.go:229:func (e *ParenExpr)           Type() ValueType { return e.Expr.Type() }
parser/ast.go:230:func (e *StringLiteral)       Type() ValueType { return ValueTypeString }
parser/ast.go:231:func (e *UnaryExpr)           Type() ValueType { return e.Expr.Type() }
parser/ast.go:232:func (e *VectorSelector)      Type() ValueType { return ValueTypeVector }
parser/ast.go:233:func (e *BinaryExpr)          Type() ValueType {
parser/ast.go:239:func (e *StepInvariantExpr) Type() ValueType { return e.Expr.Type() }
```

문자열리터럴  
벡터-selector  
이항표현식

```
root@wsl:~/go/src/lethe/letheql# grep -rn 'func.*Type().*ValueType.\+{' | grep -v ^parser
model/log.go:16:      func (l Log)                Type() parser.ValueType { return ValueTypeLog }
model/selector.go:37:func (e *LogSelector) Type() parser.ValueType { return ValueTypeLogSelector }
value.go:29:      func (String)          Type() parser.ValueType { return parser.ValueTypeString }
```

로그, 로그-selector



```
[engine.go]
```

쿼리 실행

```
func (ng *Engine) exec(ctx context.Context, q *query) (v parser.Value, ws model.Warnings,  
err error) {
```

```
    ctx, cancel := context.WithTimeout(ctx, ng.cfg.Query.Timeout) 타임아웃 처리
```

```
    q.cancel = cancel
```

```
    defer q.cancel()
```

```
    switch s := q.Statement().(type) {
```

```
    case *parser.EvalStmt:
```

평가문이면

```
        return ng.execEvalStmt(ctx, s) 평가문을 실행
```

```
    case parser.TestStmt:
```

```
        return nil, nil, s(ctx)
```

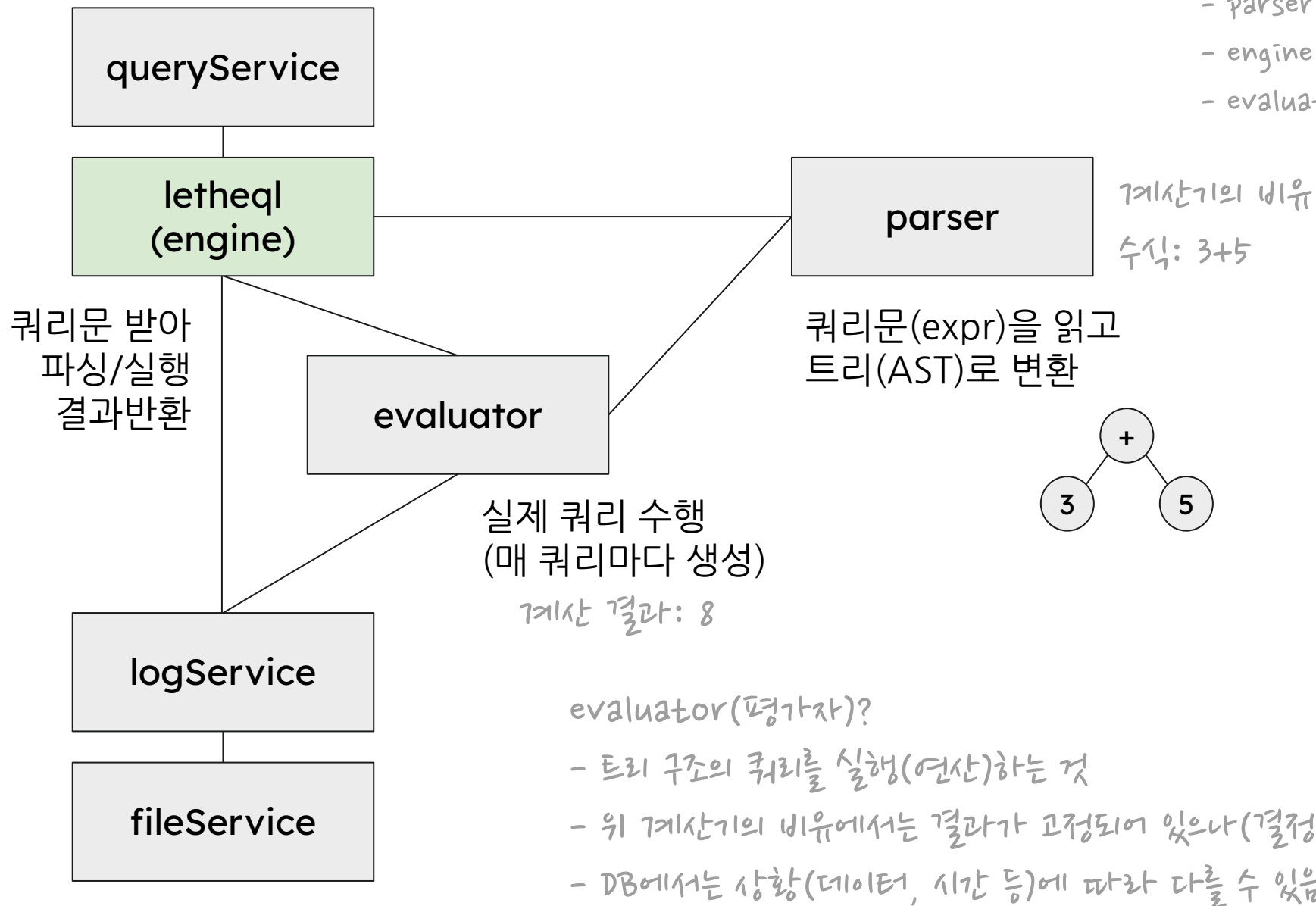
```
    }
```

```
    panic(fmt.Errorf("letheql.exec: unhandled statement of type %T", q.Statement()))
```

```
}
```

```
...
```

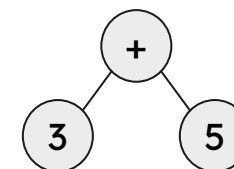
# LetheQL 구성도



LetheQL 시스템

- parser
- engine
- evaluator

계산기의 비유  
수식: 3+5



evaluator(평가자)?

- 트리 구조의 쿼리를 실행(연산)하는 것
- 위 계산기의 비유에서는 결과가 고정되어 있으나(결정적)
- DB에서는 상황(데이터, 시간 등)에 따라 다를 수 있음

```
[engine.go]      평가문을 실행하자
func (ng *Engine) execEvalStmt(ctx context.Context, stmt *parser.EvalStmt)
(parser.Value, model.Warnings, error) {
    evaluator := &evaluator{
        logService: ng.logService,
        start: stmt.Start,
        end: stmt.End,
        startTimestamp: timeMilliseconds(stmt.Start),
        endTimestamp: timeMilliseconds(stmt.End),
        ctx: ctx,
    }
    val, warnings, err := evaluator.Eval(stmt.Expr)
    if err != nil { ... }
    switch result := val.(type) {
    case model.Log:
        return result, warnings, nil
    ...
}
```

평가자 생성

- 로그서비스

- 시간범위

- 컨텍스트

표현식 평가

결과값이 로그 타입이면

로그 타입의 결과 반환

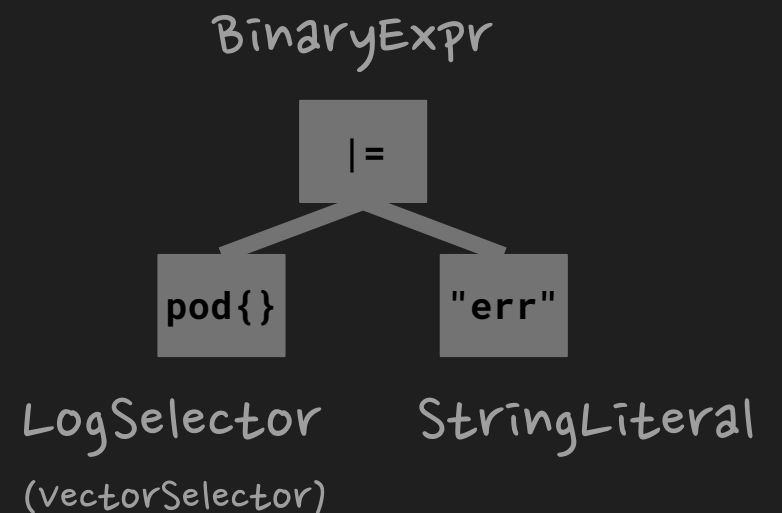
평가는 어떻게 하지?

[evaluator.go]      표현식을 평가하자

```
func (ev *evaluator) Eval(expr parser.Expr) (v parser.Value, ws model.Warnings, err error) {  
    defer ev.recover(expr, &ws, &err)      [panic-recover 패턴]  
    val, ws := ev.eval(expr)      err 래핑 대신 panic을 사용  
    ...      평가는 재귀(트리 깊이우선탐색) 처리  
    return val, ws, nil      어디서든 panic으로 즉시 귀환가능  
}
```

표현식을 평가하자

```
func (ev *evaluator) eval(expr parser.Expr) (parser.Value, model.Warnings) {  
    ...  
    switch e := expr.(type) {  
    case *parser.BinaryExpr:  
        return ev.evalBinaryExpr(e)  
    case *parser.StringLiteral:  
        return String{V: e.Val, T: ev.startTimestamp}, nil  
    case *model.LogSelector:  
        return ev.logSelector(e)  
    }  
    panic(fmt.Errorf("eval: unhandled expr: %#v", expr))  
}
```





[evaluator.go]

이항표현식을 평가하자

```
func (ev *evaluator) evalBinaryExpr(expr *parser.BinaryExpr) (parser.Value, model.Warnings) {
    if !expr.Op.IsFilterOperator() { ev.error(...) } 연산자가 필터연산자(lex.go)가 아니면 예러
    switch lhs := expr.LHS.(type) {
    case *parser.BinaryExpr:
        newLHS, warnings := ev.eval(lhs)
        switch n1 := newLHS.(type) {
        case *model.LogSelector:
            expr.LHS = n1
            return ev.evalWithWarnings(expr, &warnings)
        }
    case *parser.VectorSelector:
        newLHS, warnings := ev.evalVectorSelector(lhs)
        expr.LHS = newLHS
        return ev.evalWithWarnings(expr, &warnings)
    case *model.LogSelector:
        switch rhs := expr.RHS.(type) {
        case *parser.StringLiteral:
            lhs.LineMatchers = append(lhs.LineMatchers, &model.LineMatcher{expr.Op, rhs.Val})
            return lhs, nil
        }
    }
}
```

좌항이 또 이항표현식이면...

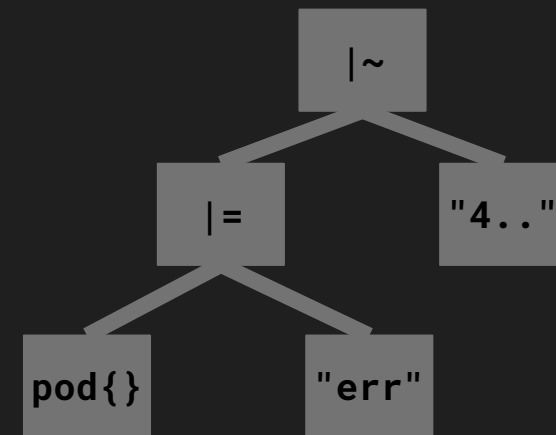
재평가(LogSelector 나올 때까지)

좌항이 vectorSelector이면...

LogSelector로 단순 변환

좌항이 LogSelector이면...

우항의 문자열을 LineMatcher로 추가



LogSelector는 무엇이고, Log는 언제 나오나?



GopherCon Korea 2024

# [LetheQL] LogSelector, Log

```
[logselector.go]
```

```
type LogSelector struct {  
    Name          string  
    LabelMatchers []*labels.Matcher  
    LineMatchers  []*LineMatcher  
    TimeRange      TimeRange  
}
```

쿼리(AST+시간범위)를 하나로 요약한 것

pod

{namespace="default", pod="nginx"}

!= "err" |~ "4.."

```
type LineMatcher struct {  
    Op      parser.ItemType  
    Value string  
}
```

```
type TimeRange struct {  
    Start time.Time  
    End   time.Time  
}
```

```
[log.go]
```

```
type Log struct {  
    Name string  
    Lines []LogLine  
}
```

평가의 최종산출물

(LogSelector 평가의 결과물)

```
type LogLine interface {  
    String() string  
}
```

[evaluator.go]

LogSelector를 평가하자

```
func (ev *evaluator) evalLogSelector(ls *model.LogSelector) (parser.Value, model.Warnings) {  
    val, ws, err := ev.logService.SelectLog(ls)  
    if err != nil {  
        ev.error(err)  
    }  
    return val, ws  
}
```

실제 Log 조회는 logService에 맡긴다.  
( letheql 패키지의 역할들은 여기까지 )

# [LetheQL] 테스트 케이스

```
[evaluator_test.go]
input: `pod{namespace="ns1",pod=~"nginx-.*",container="sidecar"}`,
want: model.Log{Name: "pod", Lines: []model.LogLine{
    logmodel.PodLog{Time: "2009-11-10T22:58:00.000Z", Namespace: "ns1", Pod: "nginx-756...", Container: "sidecar",
Log: "hello from sidecar"},
    logmodel.PodLog{Time: "2009-11-10T22:58:00.000Z", Namespace: "ns1", Pod: "nginx-756...", Container: "sidecar",
Log: "lorem from sidecar"},
    logmodel.PodLog{Time: "2009-11-10T22:58:00.000Z", Namespace: "ns1", Pod: "nginx-756...", Container: "sidecar",
Log: "hello from sidecar"}}}},

input: `node{node="node01",process!="kubelet"} |= "hello" != "sidecar"`,
want: model.Log{Name: "node", Lines: []model.LogLine{
    logmodel.NodeLog{Time: "2009-11-10T22:59:00.000Z", Node: "node01", Process: "containerd", Log: "hello"},
    logmodel.NodeLog{Time: "2009-11-10T23:00:00.000Z", Node: "node01", Process: "containerd", Log: "hello"}}}},
```

PodLog, NodeLog는? LogService에 있음

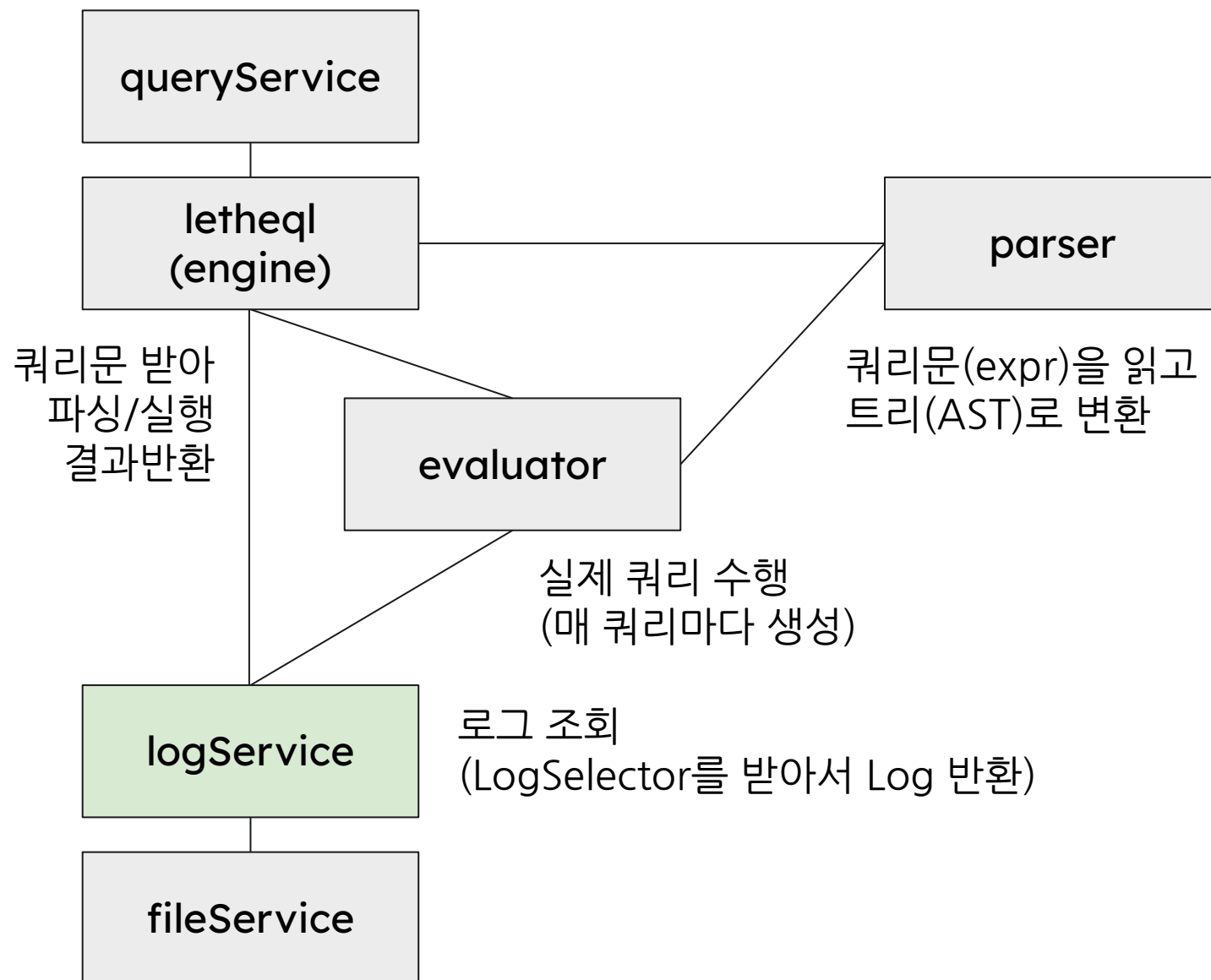
테스트데이터(로그 파일) 있는 곳

/testdata/log/pod/namespace01/

/testdata/log/node/node01/



# Lethe 패키지 구성도



LetheQL 시스템 구성

- parser
- engine
- evaluator

# Log Service

```
[logservice.go]      로그를 선택하자
func (s *LogService) SelectLog(sel *model.LogSelector) (log model.Log, warnings
model.Warnings, err error) {
    if sel.Name != "node" && sel.Name != "pod" {      타입 ( node | pod )
        return log, warnings, fmt.Errorf("unknown logType: %s", sel.Name)
    }
    targets, err := s.getTargets(sel, &warnings)      타겟 ( node | namespace )
    if err != nil { ... }
    return log, warnings, fmt.Errorf("getTargets err: %w", err)
}
files := s.GetFiles(sel, targets, &warnings)      파일을 목록
mfs, err := match.GetMatchFuncSet(sel)              매칭함수세트
if err != nil {
    return log, warnings, fmt.Errorf("getMatchFuncSet err: %w", err)
}
log = s.getLogFromFiles(sel, files, mfs, &warnings) 로그를 가져온다
return log, warnings, nil
}
```

# LogService

[logservice.go]

```
func (s *LogService) getFiles(sel *model.LogSelector, targets []string, ws *model.Warnings) []string {  
    var files []string  
    for _, target := range targets {  
        list, err := s.fileService.List(target)           타겟에서 대한 로그 파일 목록에서...  
        if err != nil { ... }  
        for _, item := range list {  
            if isFileInTimeRange(item, &sel.TimeRange) {  시간 범위가 맞는 것만 선택  
                files = append(files, item)  
            }  
        }  
    }  
    return files  
}
```

```
func isFileInTimeRange(file string, tr *model.TimeRange) bool {  파일이 시간범위에 맞는가?  
    name := filepath.Base(file)  
    fileStart, err := time.Parse(time.RFC3339, strings.Replace(name[0:13], "_", "T", 1)+":00:00Z")  
    if err != nil { ... }  
    fileEnd := fileStart.Add(time.Hour) // per hour for one logs  파일은 한시간 단위  
    return tr.Start.Before(fileEnd) && tr.End.After(fileStart)  일부만 겹쳐도 true  
}
```



# Log Service

```
[logservice.go]
```

```
type MatchFunc func(s string) bool
```

매치 함수 (문자열 매치 여부)

```
type MatchFuncSet struct {
```

매치 함수 세트

```
    LabelMatchFuncs []MatchFunc
```

- 레이블 매치 함수들

```
    LineMatchFuncs []MatchFunc
```

- 라인 매치 함수들

```
}
```

```
func GetMatchFuncSet(sel *model.LogSelector) (*MatchFuncSet, error) {
```

```
    labelMatchFuncs, err := getLabelMatchFuncs(sel)
```

```
    if err != nil { ... }
```

```
    lineMatchFuncs, err := getLineMatchFuncs(sel)
```

```
    if err != nil { ... }
```

```
    return &MatchFuncSet{
```

```
        LabelMatchFuncs: labelMatchFuncs,
```

```
        LineMatchFuncs: lineMatchFuncs,
```

```
    }, nil
```

```
}
```

로그셀렉터에 따라

MatchFuncSet 생성

# Log Service

```
[logservice.go]
func getLineMatchFuncs(sel *model.LogSelector) ([]MatchFunc, error) {
    var funcs []MatchFunc
    for _, m := range sel.LineMatchers {
        f, err := getLineMatchFunc(m)
        if err != nil { ... }
        funcs = append(funcs, f)
    }
    return funcs, nil
}
```

LineMatcher 목록에 따라

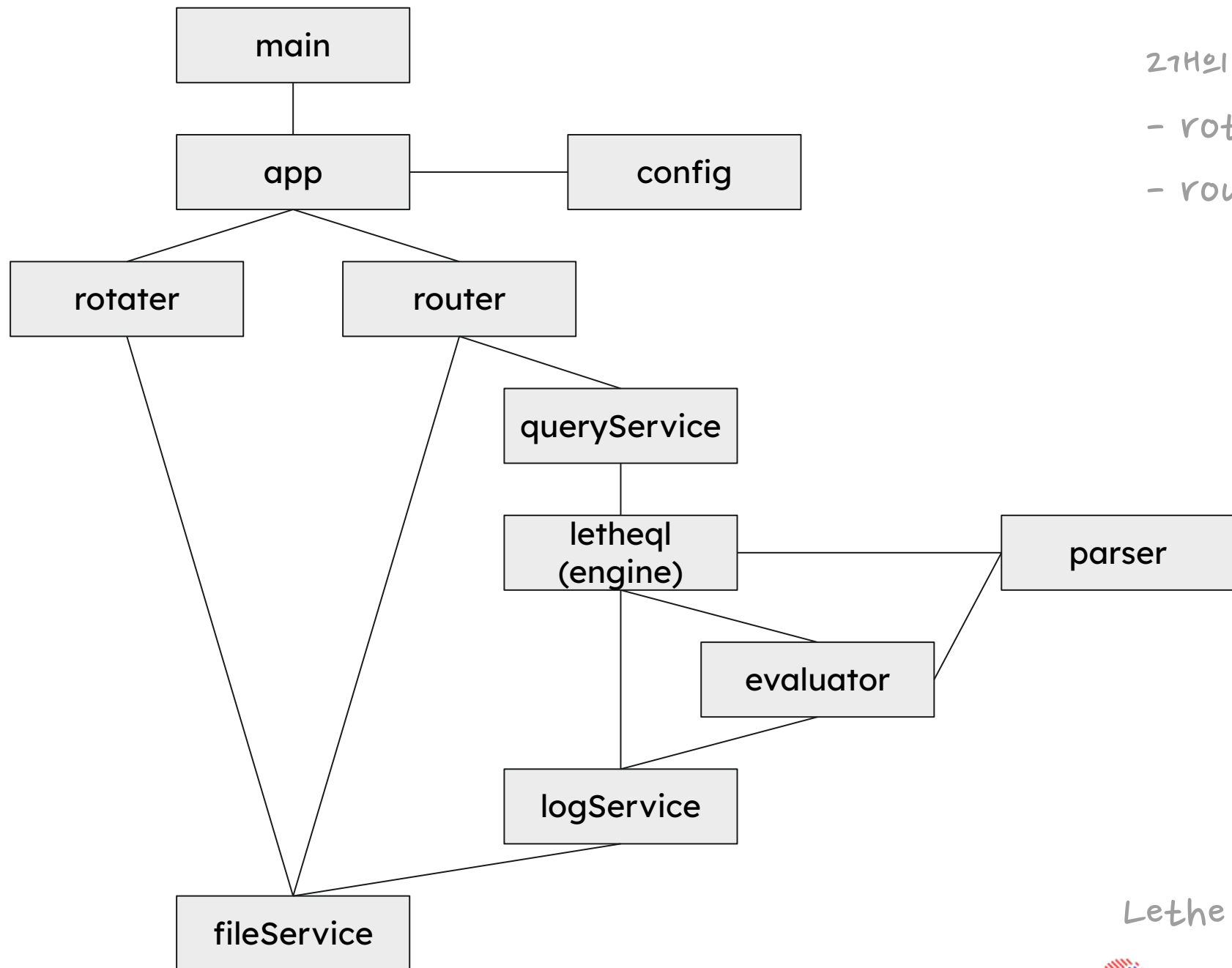
매치함수 목록을 만든다 (다건)

```
func getLineMatchFunc(m *model.LineMatcher) (MatchFunc, error) {
    switch m.Op {
    case parser.PIPE_EQ: // |=
        return func(s string) bool { return strings.Contains(s, m.Value) }, nil
    case parser.NEQ: // !=
        return func(s string) bool { return !strings.Contains(s, m.Value) }, nil
    case parser.PIPE_REGEX: // |~
        re, err := regexp.Compile(m.Value)
        if err != nil { ... }
        return func(s string) bool { return re.MatchString(s) }, nil
    case parser.NEQ_REGEX: // !~
        re, err := regexp.Compile(m.Value)
        if err != nil { ... }
        return func(s string) bool { return !re.MatchString(s) }, nil
    }
    return nil, fmt.Errorf("unknown match op: %s", m.Op)
}
```

LineMatcher에 따라

매치함수를 만든다 (단건)

# Lethe 패키지 구성도

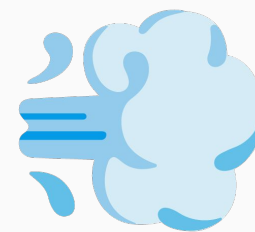


2개의  $\frac{n}{2}$   
- rotator  
- router

# **IV** Venti (Visualizer)

# Venti

- Visualizer
  - YAML 대시보드 설정
  - 패널: 스탯, 파이차트, 시계열, 로그
  - 마우스오버 데이터테이블
  - 시간 선택기, Auto Refresh
- 메트릭/로그 기반 Alerting
- 데이터 소스
  - 유형: Prometheus, Lethe
  - 멀티 데이터소스 지원
  - 데이터소스 발견  $\doteq$  service discovery







Metrics

Logs

Dashboards (4)

Cluster

Control Plane

Ingress

Node

Alert

Datasource

Status

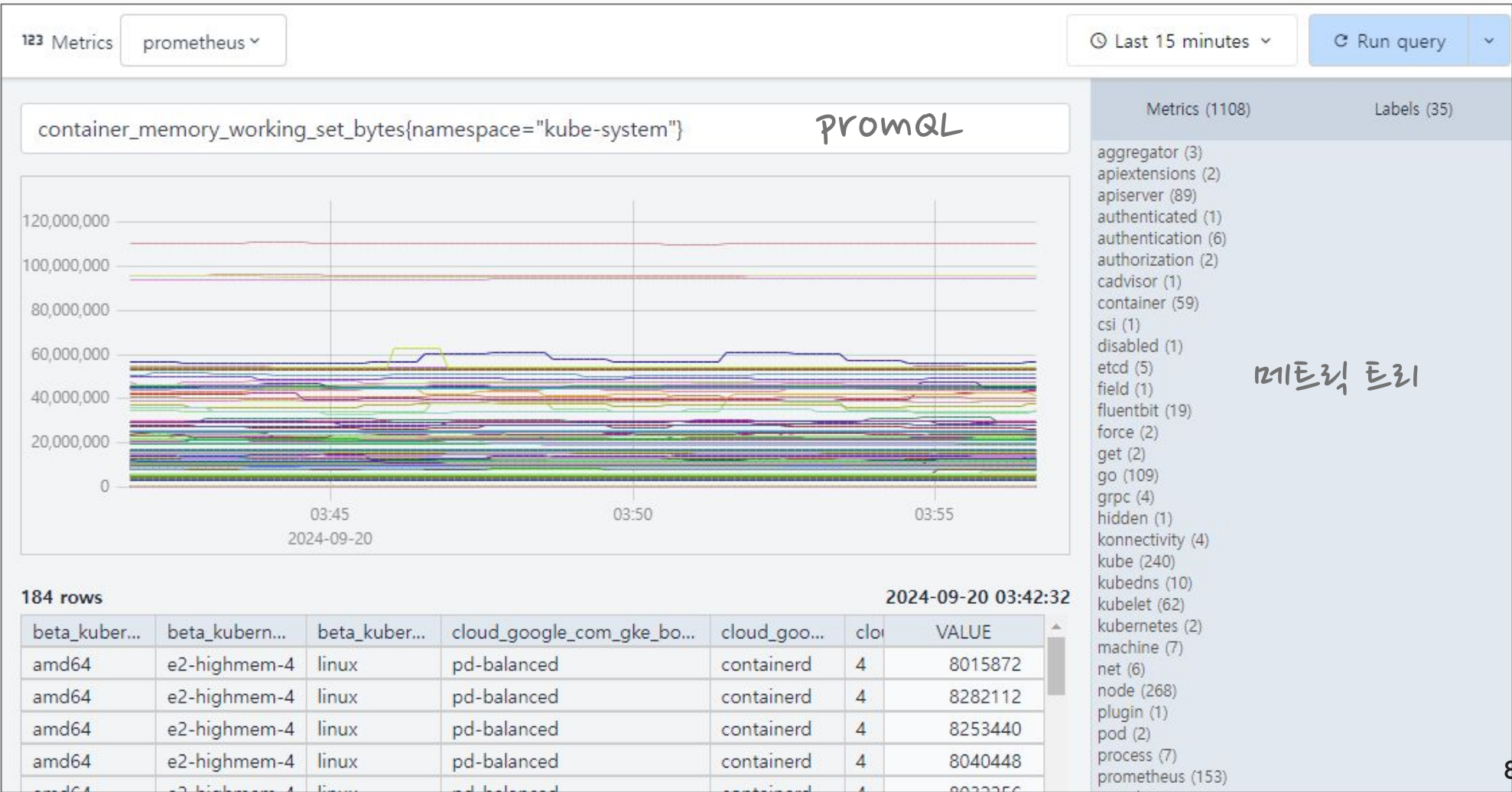
light

dark

Logout

데이터소스 선택기

시간 선택기



메트릭 트리



Metrics

Logs

Dashboards (4)

Cluster

Control Plane

Ingress

Node

Alert

Datasource

Status

light

dark

Logout

데이터소스 선택기

시간 선택기

≡ Logs

lethe ▾

🕒 now-60m to now-30m ▾

🏃 Run c

pod{namespace="kube-system", pod="calico-node-tfml4"}

LetheQL

2024-09-20 02:34:17 - 03:04:17

pod{namespace="kube-system", pod="calico-node-tfml4"}

209 rows

2024-09-20 02:34:19 kube-system calico-node-tfml4 calico-node

2024-09-19 17:34:19.837 [INFO][62] felix/summary.go 10

0: Summarising 9 dataplane reconciliation loops over 1m7.4s: avg=18ms longest=126ms ()

2024-09-20 02:35:23 kube-system calico-node-tfml4 calico-node

2024-09-19 17:35:23.231 [INFO][62] felix/summary.go 10

0: Summarising 12 dataplane reconciliation loops over 1m3.4s: avg=15ms longest=120ms ()

2024-09-20 02:36:26 kube-system calico-node-tfml4 calico-node

2024-09-19 17:36:26.169 [INFO][62] felix/summary.go 10

0: Summarising 11 dataplane reconciliation loops over 1m2.9s: avg=6ms longest=16ms (resync-filter-v4)

2024-09-20 02:37:26 kube-system calico-node-tfml4 calico-node

2024-09-19 17:37:26.229 [INFO][62] felix/summary.go 10

0: Summarising 8 dataplane reconciliation loops over 1m0.1s: avg=20ms longest=127ms ()

2024-09-20 02:38:31 kube-system calico-node-tfml4 calico-node

2024-09-19 17:38:31.186 [INFO][62] felix/summary.go 10

0: Summarising 11 dataplane reconciliation loops over 1m5s: avg=17ms longest=137ms ()

2024-09-20 02:39:34 kube-system calico-node-tfml4 calico-node

2024-09-19 17:39:34.639 [INFO][62] felix/summary.go 10

0: Summarising 11 dataplane reconciliation loops over 1m3.5s: avg=6ms longest=16ms (resync-filter-v4)

2024-09-20 02:40:00 kube-system calico-node-tfml4 calico-node

2024-09-19 17:40:00.617 [INFO][62] felix/calc\_graph.go

467: Local endpoint updated id=WorkloadEndpoint(node=gke-cluster1-worker5-9fdbf910-ibtd, orchestrator=k8s, workload=cron/analytics-28779460-xg7kn, name=eth0)

2024-09-20 02:40:00 kube-system calico-node-tfml4 calico-node

2024-09-19 17:40:00.617 [INFO][62] felix/int\_dataplan

e.go 1836: Received \*proto.ActiveProfileUpdate update from calculation graph msg=id:<name:"kns.cron" > profile:<inb

ound\_rules:<action:"allow" rule\_id:"W9dpr0z8lvoikY4p" > outbound\_rules:<action:"allow" rule\_id:"72zes54Cur1QHK\_n" >

>

2024-09-20 02:40:00 kube-system calico-node-tfml4 calico-node

2024-09-19 17:40:00.617 [INFO][62] felix/table.go 508:

Queueing update of chain chainName="calico-iptables" inVersion=0x4-iptables="filter"

Targets

node

gke-cluster1-pool-1-b04eaa76-2tqc

gke-cluster1-pool-1-b04eaa76-jsp1

gke-cluster1-pool-1-b04eaa76-z59d

gke-cluster1-worker5-9fdbf910-ibtd

pod

> cert-manager

> cron

> default

> dev

> dev2

> devpod

> docker

> gke-managed-system

> ingress-annotator

> ingress-nginx

> kube-node-lease

> kube-public

> kube-system

> deployment (10)

> daemonset (29)

> pod (51)

calico-node-tfml4

calico-node-vertical-autoscaler-774ff66

워크로드 트리

90

GopherCon Korea 2024



## 데시보드 설정 YAML 파일

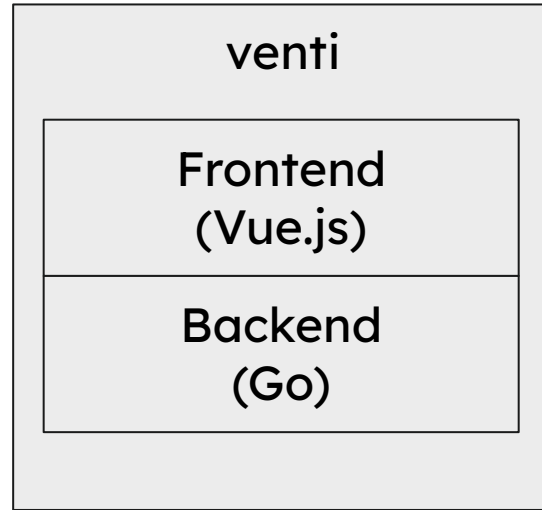
```
es/venti/cm-dashboards.yaml
```



Logout



# Venti 구성도



백엔드와 프론트엔드의 repo?

별도의 repo

- 백엔드와 프론트엔드의 버전간 호환성 고려
- kubernetes 스타일

하나의 repo ✓

- 하나의 버전으로 관리
- 하나의 도커 이미지
- prometheus 스타일

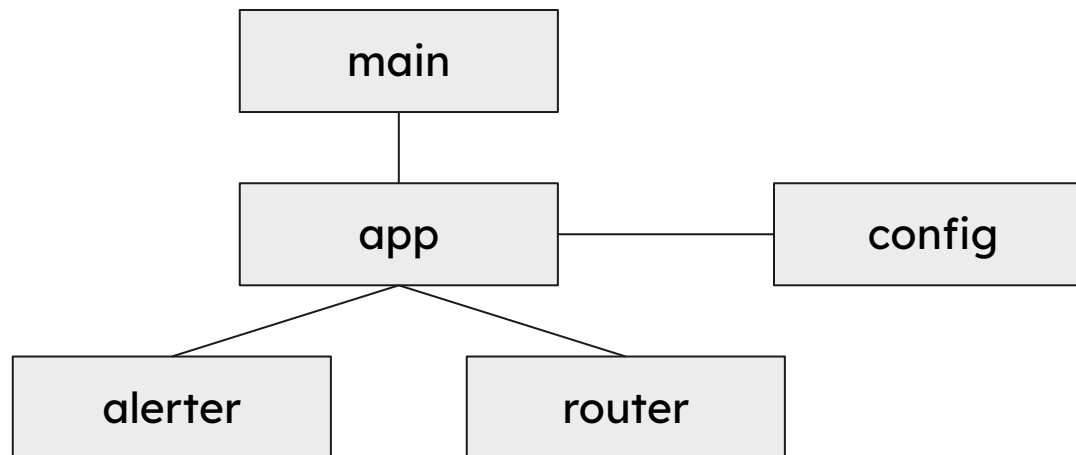
repo의 디렉토리 구조는?

/ Go 프로젝트 루트 (go.mod)

/pkg Go 패키지들은 여기에

/web vue.js 프로젝트 루트

# Venti 백엔드 구성도



설정

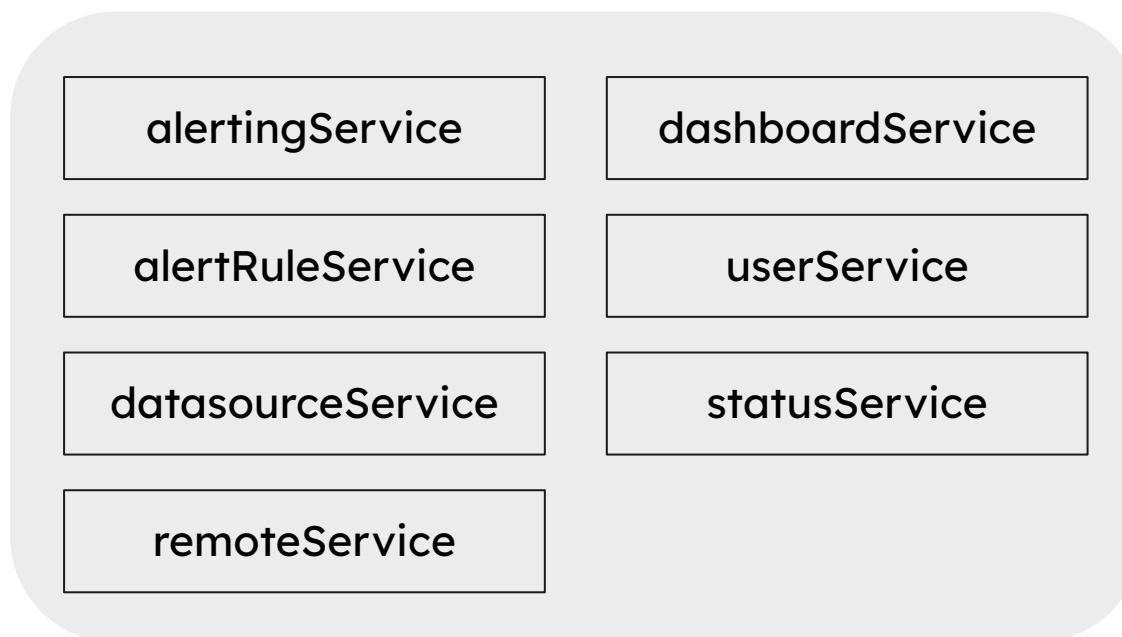
- 데이터소스
- 알람
- 대시보드

알림 서비스

알림규칙 서비스

데이터소스 서비스

원격조치 서비스



대시보드 서비스

사용자 서비스

상태 서비스

# Venti alert rule

[example]

- alert: NewNamespace

expr: time() - kube\_namespace\_created < 120

for: 0m

최근(120초 이내) 생성된 네임스페이스가 있나?

labels:

severity: info

annotations:

summary: 'A new namespace "{{ \$labels.namespace }}" was created'

- alert: NginxError

expr: pod{namespace="ns1",pod=~"nginx-.\*"} |= "error"

for: 0m

nginx 파드 로그에 error가 찍혔나?

labels:

severity: critical

annotations:

summary: 'nginx pod reports an error'

# Venti alerter

```
[alerter.go]
func New(cfg *config.Config, alertingService alerting.IAlertingService) *Alerter {
    return &Alerter{
        alertingService:    alertingService,
        evaluationInterval: cfg.AlertingConfig.EvaluationInterval,  평가 간격
    }
}

func (a *Alerter) Start() error {
    if a.isRunning {
        return fmt.Errorf("already running")
    }
    a.isRunning = true
    logger.Infof("starting alerter...")
    a.quitCh = make(chan bool)
    go a.loop(a.quitCh)  종료할 수 있는 채널을 전달
    return nil
}
```



# Venti alerter

```
[alerter.go]
func (a *Alerter) Once() {
    if err := a.alertingService.DoAlert(); err != nil {
        logger.Errorf("DoAlert err: %s", err)    alertingService에 요청
    }
}
```

```
[alerting.go]
func (s *AlertingService) DoAlert() error {
    ...
    fires := []Fire{}
    s.evalAlertingRuleGroups(&fires)    알림규칙 평가
    if err := s.sendFires(fires); err != nil {    발송
        return fmt.Errorf("sendFires err: %w", err)
    }
    return nil
}
```

# Venti router

```
[router.go]
/api/v1      api := router.Group("/api/v1")
              {
                alert 관련 기능 api.GET("/alerts", handlers.alertHandler.Alerts)
                                api.GET("/alerts/test", handlers.alertHandler.SendTestAlert)
                                api.GET("/alertmanagers", handlers.alertHandler.Alertmanagers)

                대시보드 조회 api.GET("/dashboards", handlers.dashboardHandler.Dashboards)

                                api.GET("/datasources", handlers.datasourceHandler.Datasource...)
                데이터 소스 조회 api.GET("/datasources/targets", handlers.datasourceHandler.Targets...)
                                api.GET("/datasources/targets/:name", handlers.datasourceHandler.Targets...)

                                api.GET("/remote/healthy", handlers.remoteHandler.Healthy)
                원격 조회      api.GET("/remote/metadata", handlers.remoteHandler.Metadata)
                (prom 또는 lenth) api.GET("/remote/query", handlers.remoteHandler.Query)
                                api.GET("/remote/query_range", handlers.remoteHandler.QueryRange)

                                api.GET("/status/buildinfo", handlers.statusHandler.BuildInfo)
                상태 조회      api.GET("/status/runtimeinfo", handlers.statusHandler.RuntimeInfo)

/auth
              }
              로그인/로그아웃 router.POST("/auth/login", handlers.authHandler.Login)
                              router.POST("/auth/logout", handlers.authHandler.Logout)

/
프론트엔드 연결 router.Use(handleSPA())
```

# [Venti] router

[handle\_spa.go]

```
func handleSPA() gin.HandlerFunc {  
    directory := static.LocalFile("./web/dist", true)    vue의 build 결과물 디렉토리  
    fileserver := http.StripPrefix("/", http.FileServer(directory))  
    return func(c *gin.Context) {  
        if !directory.Exists("/", c.Request.URL.Path) {  
            c.Request.URL.Path = "/"  
        }  
        fileserver.ServeHTTP(c.Writer, c.Request)  
        c.Abort()  
    }  
}
```

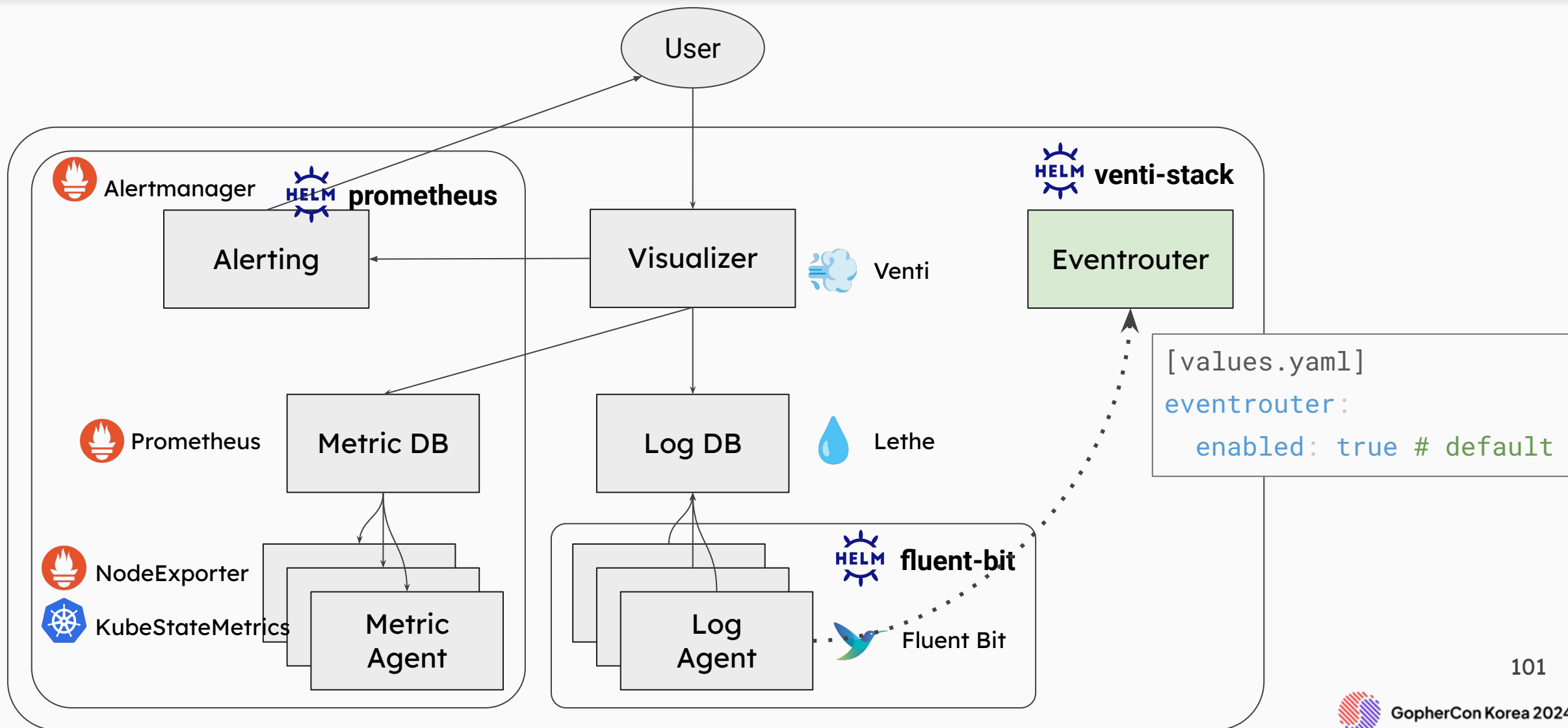
디렉토리를 파일서버로서  
"/"에 연결하여  
서비스 제공

(이후 url 경로는 vue에서 처리)

# V 기타



# Eventrouter #1



# Eventrouter #2

## 기능

- k8s Event를 감시하여 Log로서 전달

k8s Event를 watch하다가, 생성/변경 시 원하는 곳(sink)으로 전달(route)

- venti-stack의 컴포넌트(기본 enabled), sink는 stdout event로그 → pod로그

- ## repo
- <https://github.com/heptiolabs/eventrouter> '17년 시작(아파치-2.0), '18년 VMware 인수
  - <https://github.com/vmware-archive/eventrouter> '19년 개발중단, '22년 아카이브
  - <https://github.com/kuoss/eventrouter> '22년 포크

```
$ kubectl get event
```

LAST SEEN	TYPE	REASON	OBJECT	MESSAGE
30s	Normal	ScalingReplicaSet	deployment/nginx	Scaled up replica set nginx-6d4c... to 1
2m	Warning	FailedScheduling	pod/test-pod	0/3 nodes are available: 3 Insufficient...
4m	Normal	Pulled	pod/nginx-6d4cf...	Successfully pulled image "nginx"



# yaml.v3에서 UnmarshalStrict 구현

```
import "gopkg.in/yaml.v2"

func Foo() {
    err := yaml.UnmarshalStrict([]byte(data), &out)
```

없는 필드/중복된 매핑 키 발생시 예러  
yaml.v3에서 제거됨

```
import "gopkg.in/yaml.v3"

func UnmarshalStrict(data []byte, out any) error {
    decoder := yaml.NewDecoder(bytes.NewReader(data))
    decoder.KnownFields(true)
    return decoder.Decode(out)
```

yaml.v3에서 같은 기능을 하도록 구현

```
- alert: NewNamespace
  expr: time() - kube_namespace_created < 120
  # for: 0
  for: 0m
```

yaml.v3 변경 영향

time.Duration에 단위 필수 (h, m, s)  
(v2에서는 단위 없으면 나노초로 인식)

# tester.SetupDir

```
import "github.com/kuoss/common/tester"

func TestNew_ok1(t *testing.T) {
    _, cleanup := tester.SetupDir(t, map[string]string{
        "@/testdata/etc/lethe.ok1.yaml": "etc/lethe.yaml",
        "@/testdata/log":                "data/log",
    })
    defer cleanup()

    want := &Config{}
    got, err := New("ok1")
    require.NoError(t, err)
    require.Equal(t, want, got)
}
```

testing 패키지의  
t.TempDir() 활용  
테스트용 임시 디렉토리

테스트용 설정 파일  
테스트용 로그 디렉토리



# EOF

## venti-stack 요약

- Helm Chart 하나로 LMA 설치
- Production 사용 중 3년+, 가볍고 안정적
- Permissive 라이선스 사용/문의/참여 환영합니다.

## What's next

- LogDB 성능 비교 검증 이론적·경험적으로는 경량. 정량적 검증 필요
- venti-stack Helm Chart 개발기 기회가 되면
- docs 작성, venti TypeScript 전환 진행중