次世代のログ基盤 Grafana Loki を始めよう！

Loki　　: @uesyn
Promtail: @kameneko1004

Profile

上村 真也

- 所属: ゼットラボ株式会社
- Twitter: @uesyn
- 今回からこのMeetupの運営へ入りました

ゼットラボ株式会社 / Z Lab Corporation

- 2015年に設立されたヤフー株式会社の100%子会社
- ヤフーのインフラ課題に対して R&D でソリューション提供
- Kubernetes as a Service を開発・提供

Lokiとは？

- Log Aggregation System
  - Horizontally-Scalable
  - Highly-Available
  - Multi-tenant
- Grafana Labsが主体となって開発
  - Grafana連携
- OSS
  - https://github.com/grafana/loki
  - 最新はv1.2.0(2020/1/14時点)

デモ

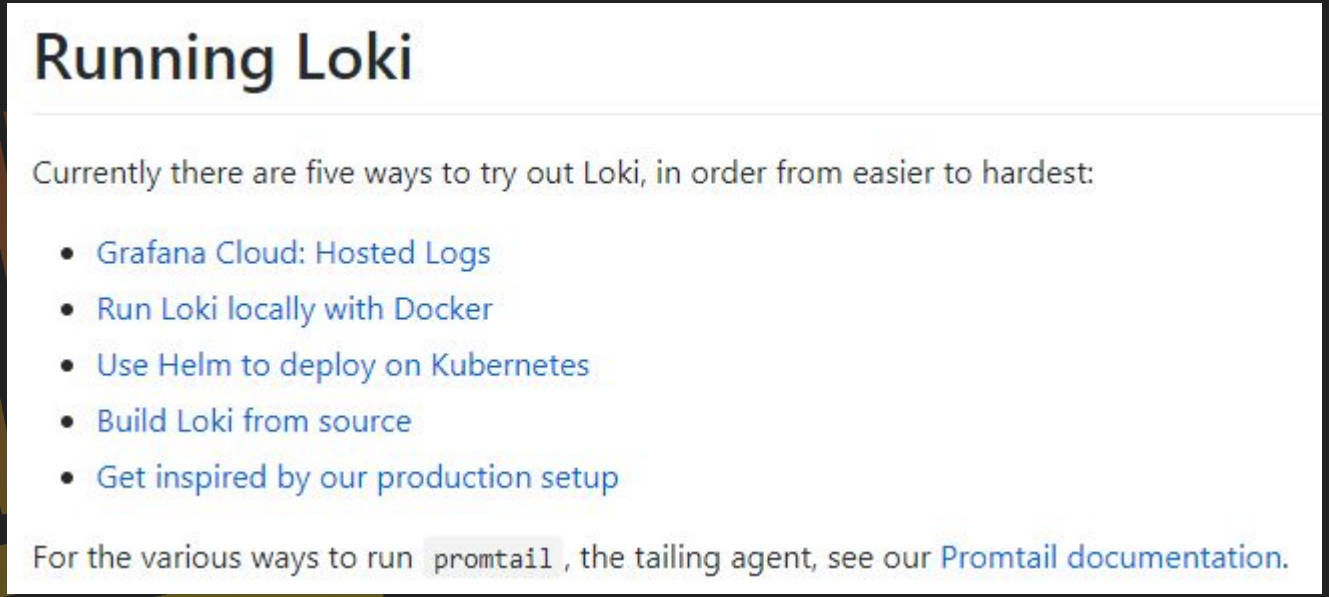
動かなかったら
この動画再生します

https://www.youtube.com/watch?time_continue=5&v=7n342UsAMo0

Lokiについて

- Like Prometheus, but for logs
  - ログ検索には以下のように行う
    - ログに付与されたラベルによるフィルタリング
    - さらに結果をgrepのように絞り込み
- ログをシンプルに扱う
  - ログを保存時する時、形態素解析のようなテキスト処理はしない
  - Prometheusのようにラベルを付与してログを保存

Lokiを動かす



Running Loki

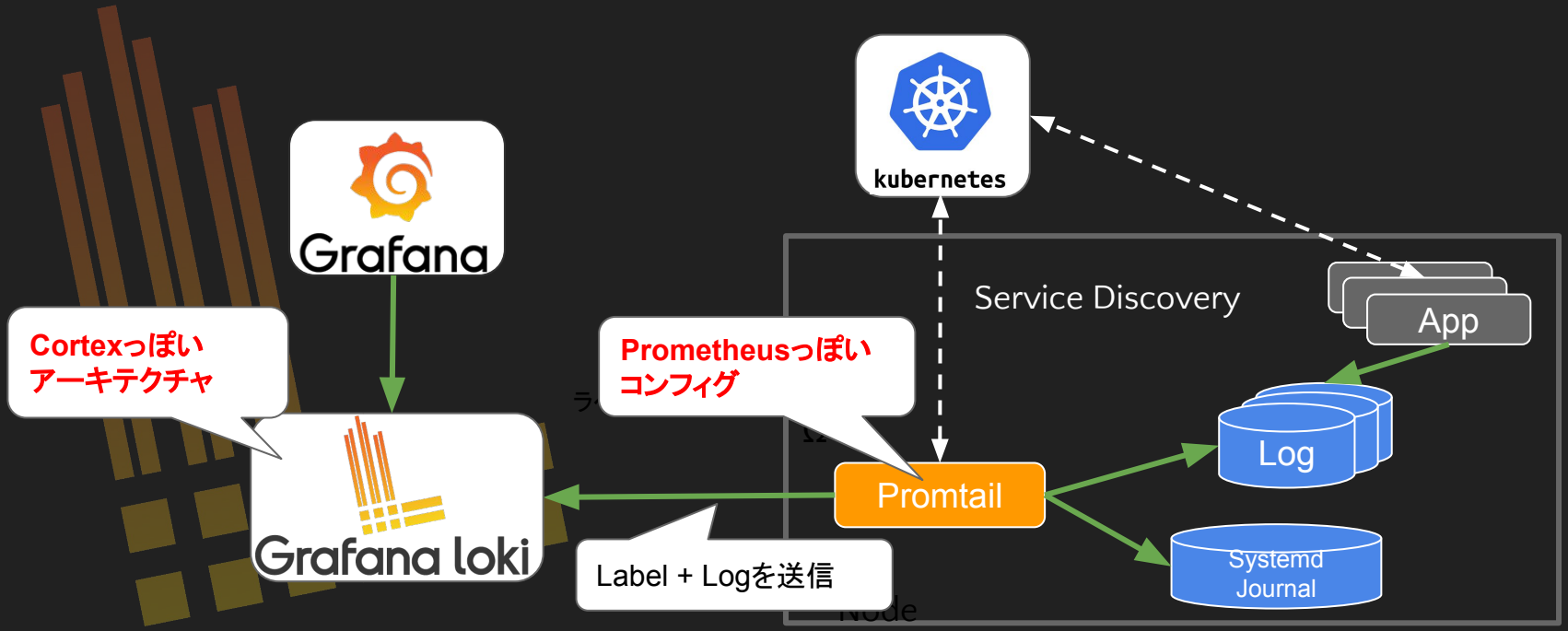
Currently there are five ways to try out Loki, in order from easier to hardest:

- Grafana Cloud: Hosted Logs
- Run Loki locally with Docker
- Use Helm to deploy on Kubernetes
- Build Loki from source
- Get inspired by our production setup

For the various ways to run `promtail`, the tailing agent, see our Promtail documentation.

https://github.com/grafana/loki/blob/master/production/README.md#running-loki

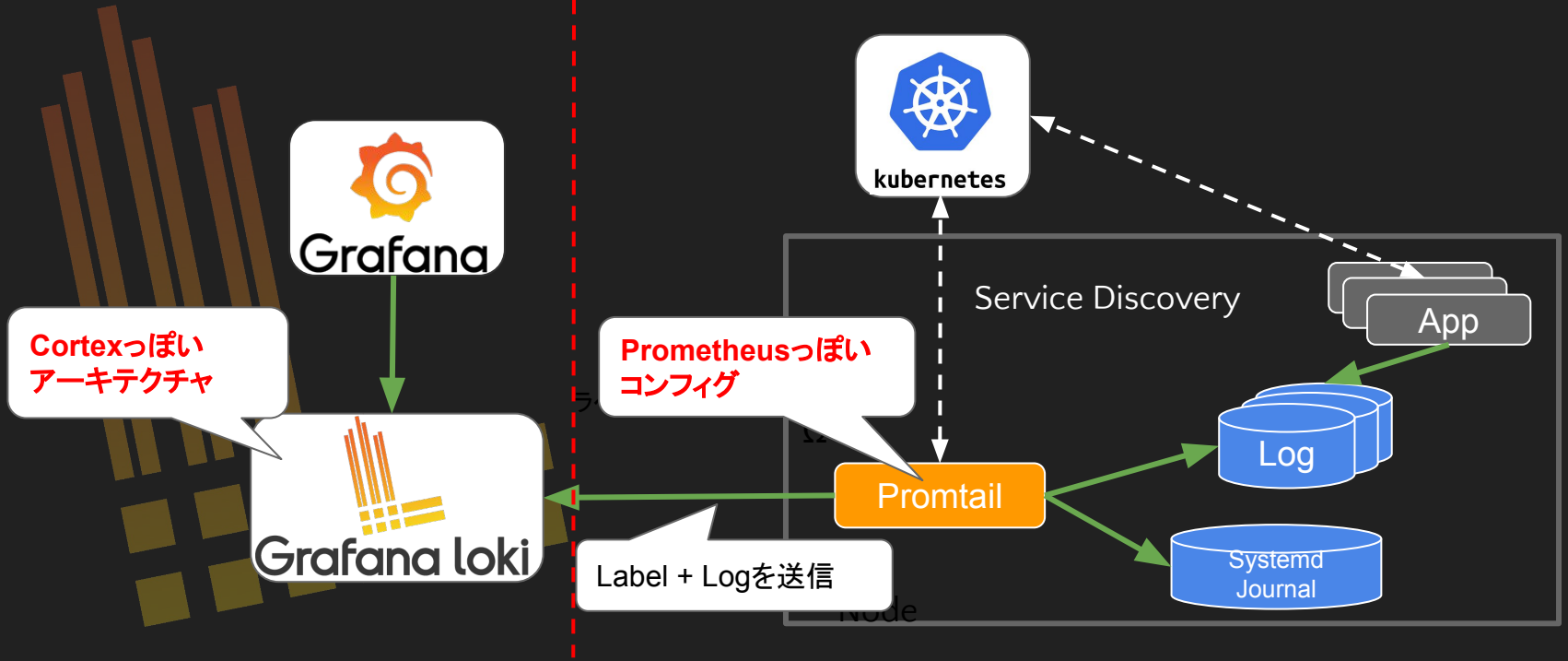
Lokiの概要

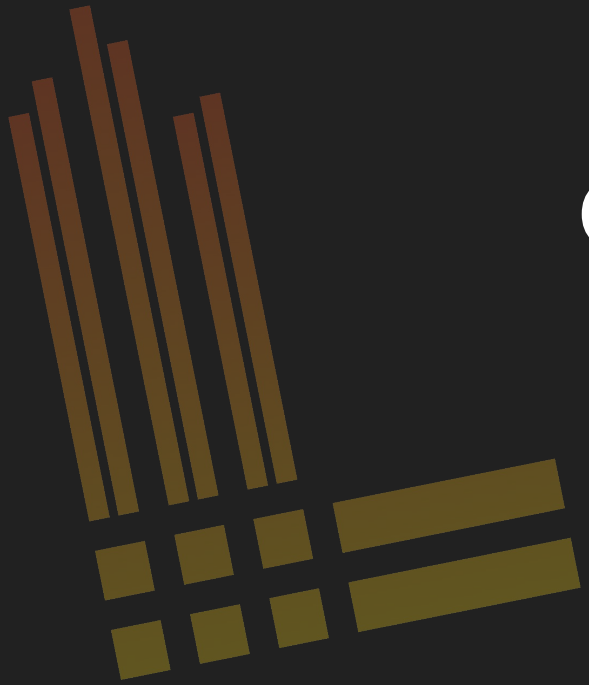

kubernetes
Grafana
Service Discovery
App
Cortexっぽい
アーキテクチャ
Prometheusっぽい
コンフィグ
Log
Promtail
Grafana loki
Label + Logを送信
Systemd
Journal

Lokiの概要

前半パート(@uesyn)

後半パート(@kameneko1004)

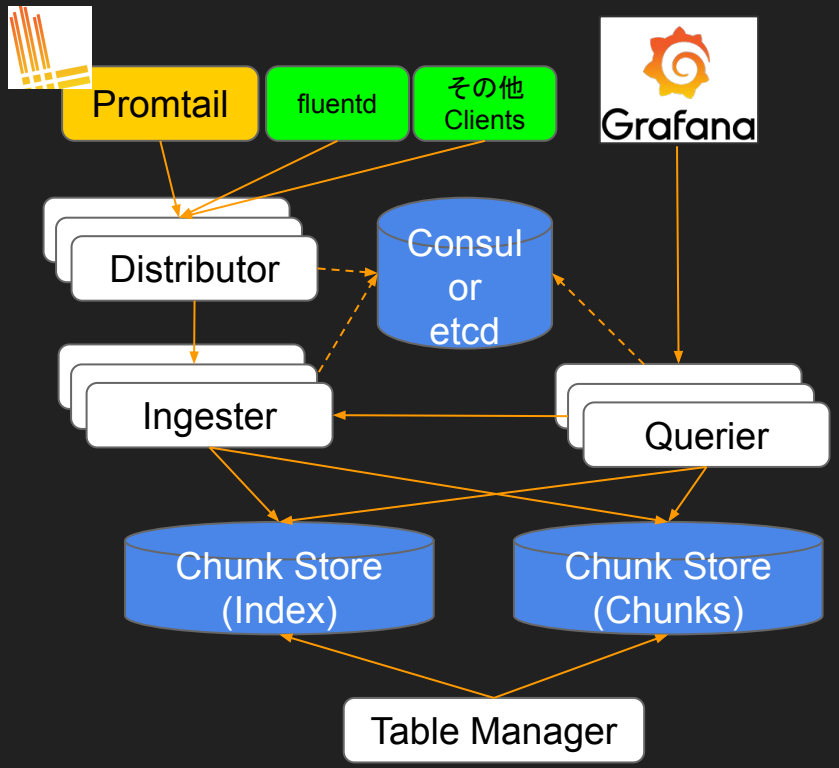




Grafana loki

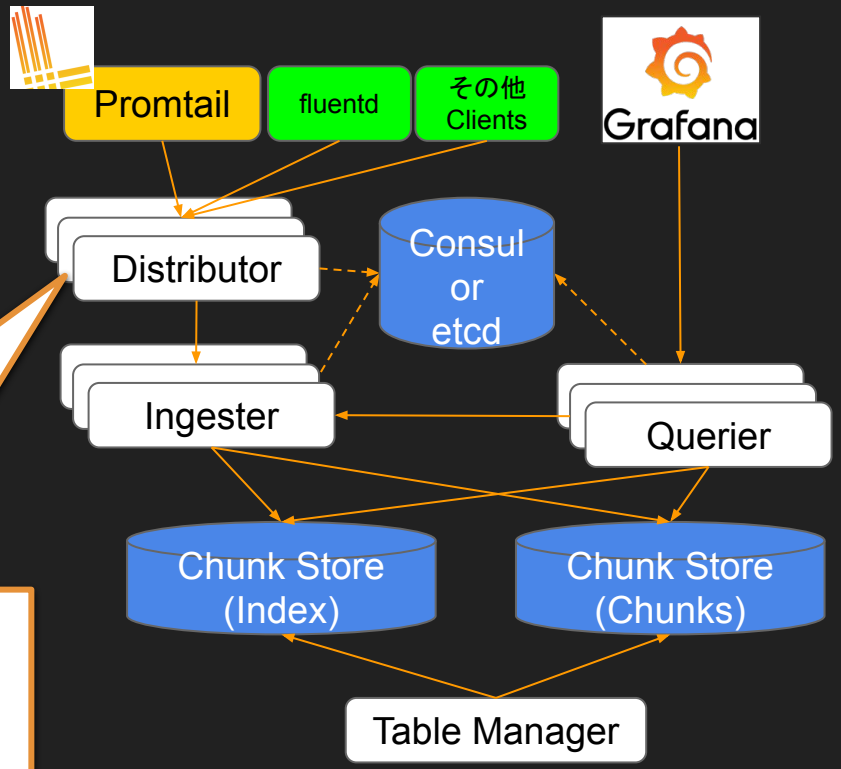
lokiのアーキテクチャ

- ほぼCortexのアーキテクチャ
  - https://github.com/cortexproject/cortex
- 主なコンポーネント
  - Distributor
  - Ingester
  - Chunk Store
  - Querier
  - Table Manager



lokiのアーキテクチャ

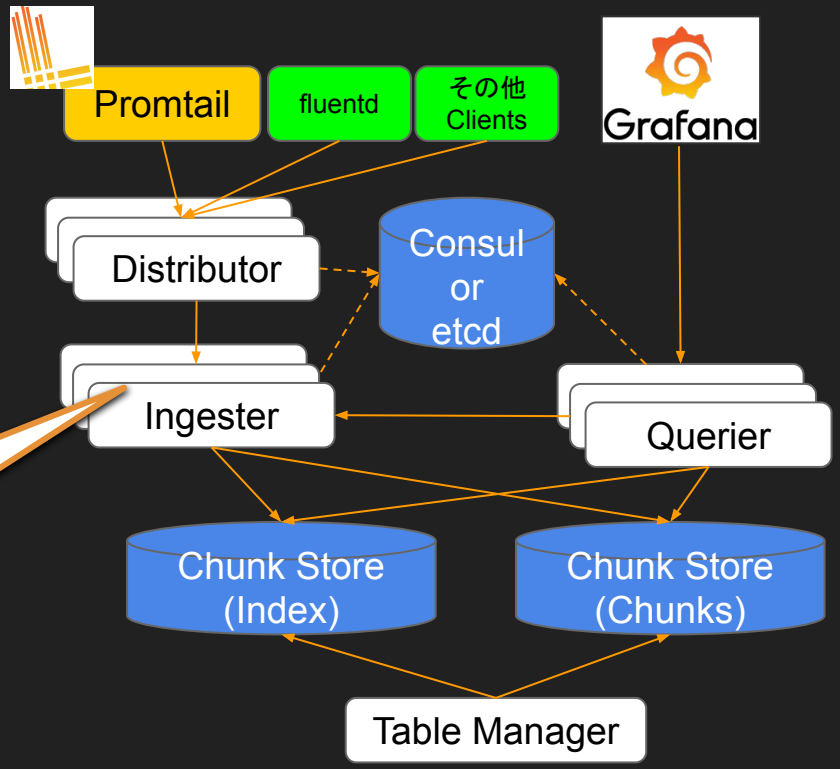
- ほぼCortexのアーキテクチャ
  - https://github.com/cortexproject/cortex
- 主なコンポーネント
  - **Distributor**
  - Ingester
  - Chunk Store
  - Querier
  - Table Manager



- 受信したログをIngesterへ送信
- 冗長化構成の時は複数のIngesterへ
- どのIngesterへ送信するかはConsul(or etcd)を参照

lokiのアーキテクチャ

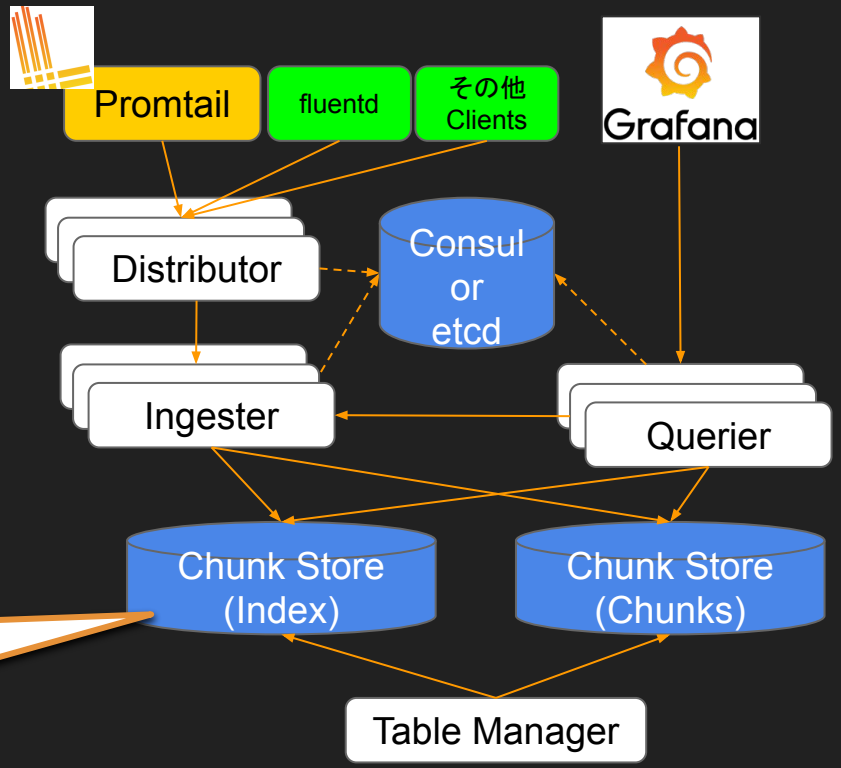
- ほぼCortexのアーキテクチャ
  - https://github.com/cortexproject/cortex
- 主なコンポーネント
  - Distributor
  - **Ingester**
  - Chunk Store
  - Querier
  - Table Manager



- ログを永続ストレージへ保存
- Ingester毎に受け付けるログが決まっている
  - Consul(or etcd)に情報を保持
- セミステートフルなコンポーネント
  - 直近のログを持つ

lokiのアーキテクチャ

- ほぼCortexのアーキテクチャ
  - https://github.com/cortexproject/cortex
- 主なコンポーネント
  - Distributor
  - Ingester
  - **Chunk Store**
  - Querier
  - Table Manager



- IndexとChunkの2種類のストレージ
  - 外部(または内部)のストレージを利用

Chunk Storeの補足

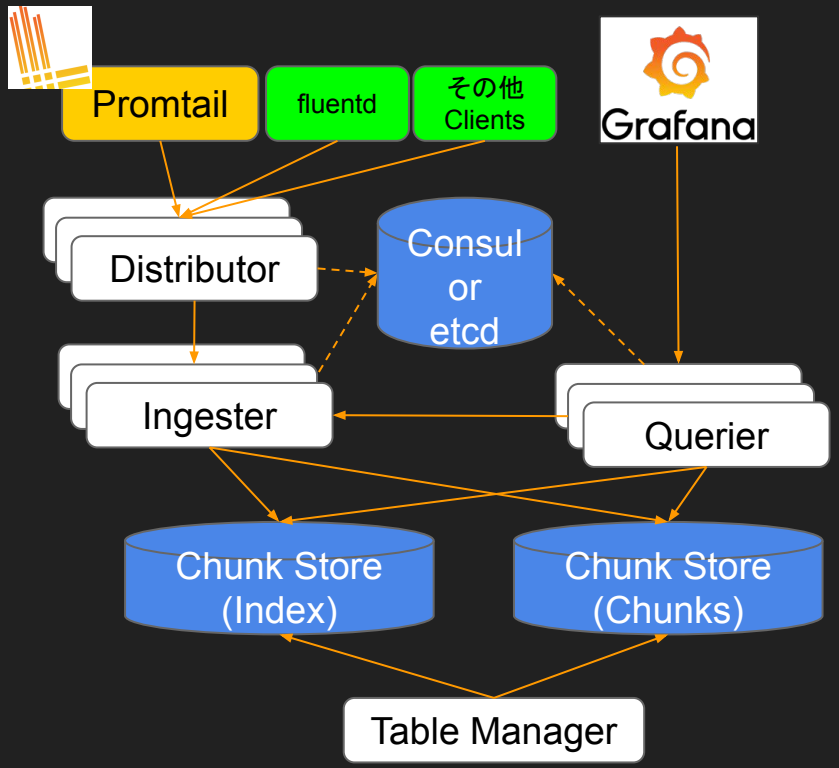
- lokiè‡ªä½“ã¯DBã§ã¯ãªã„
- 2種類のChunk Store
  - **Index**: ログ検索のための転置インデックス
  - **Chunk**: 実際のログを保持
- 利用可能なDB

Index

- Local
- DynamoDB
- Bigtable
- Cassandra

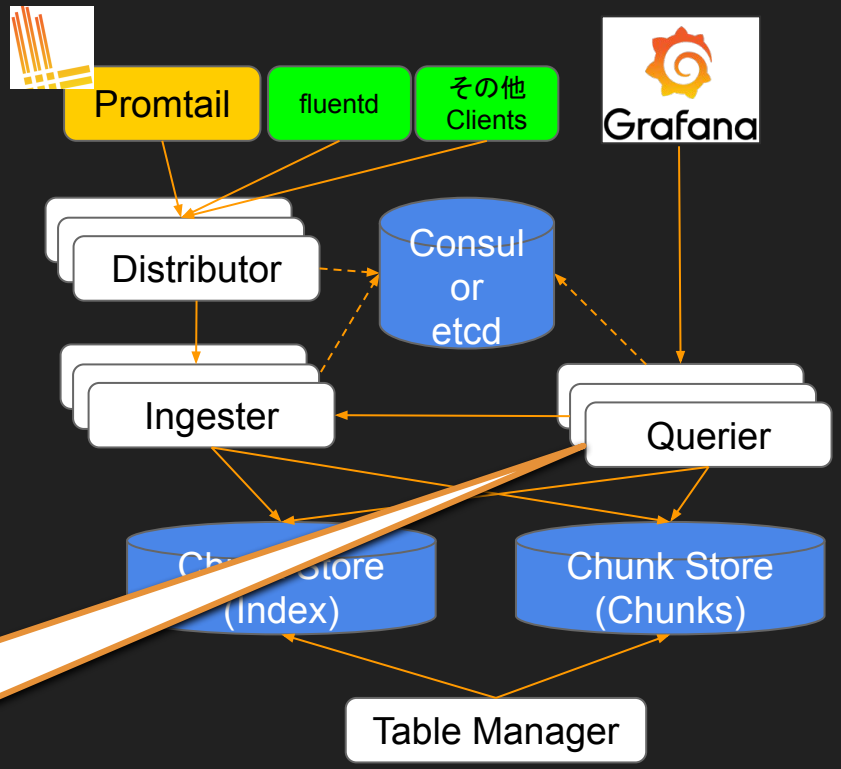
Chunk

- Local
- Cloud Storage
- S3



lokiのアーキテクチャ

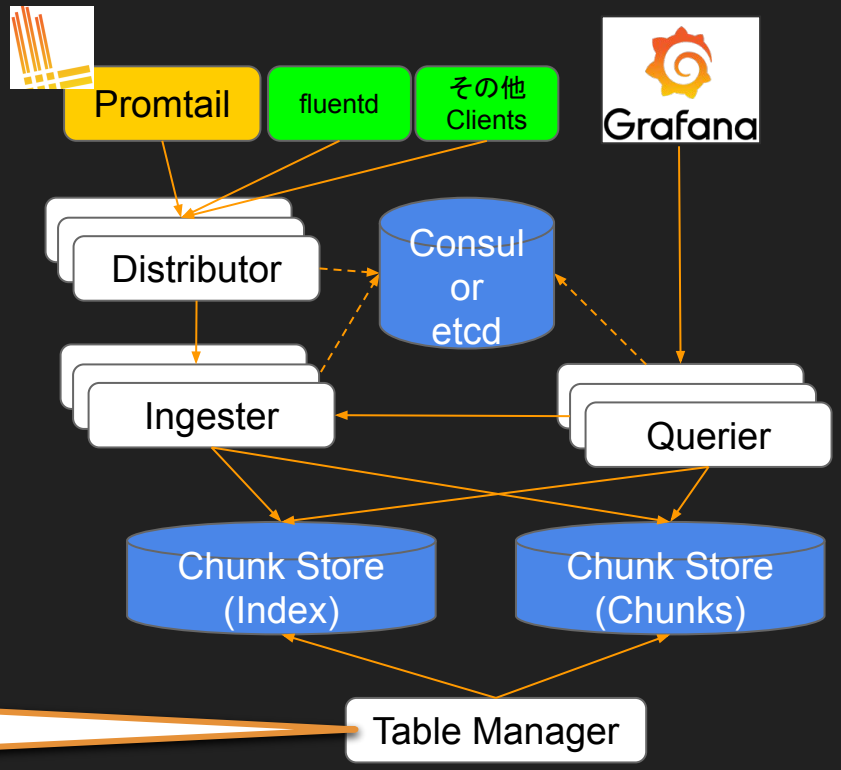
- ほぼCortexのアーキテクチャ
  - https://github.com/cortexproject/cortex
- 主なコンポーネント
  - Distributor
  - Ingester
  - Chunk Store
  - **Querier**
  - Table Manager



- ログの検索に応じる
- IngesterやChunk Storeからログを取得

lokiのアーキテクチャ

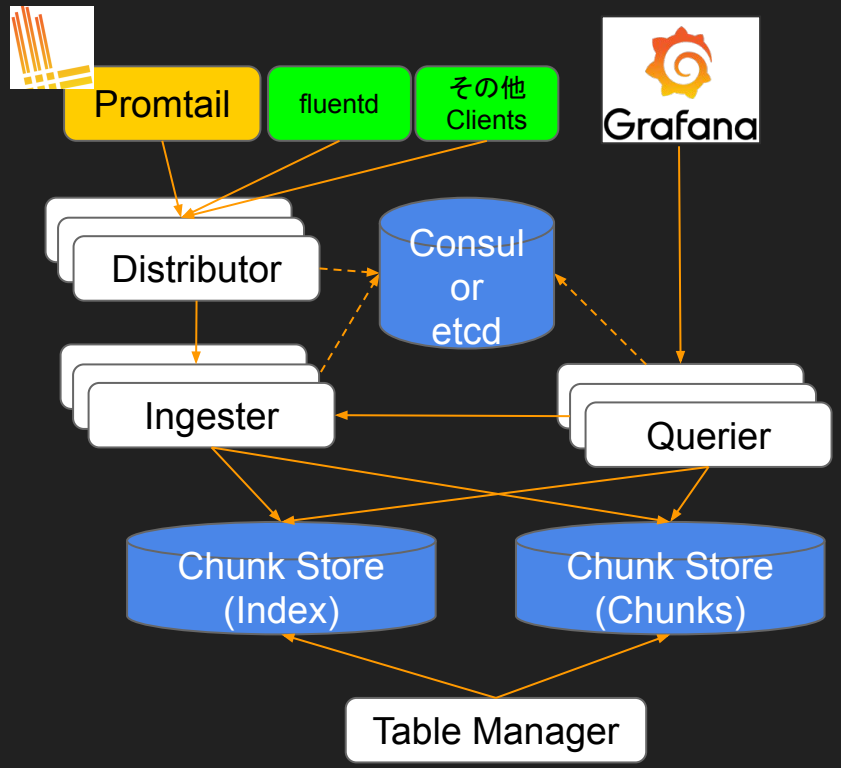
- ほぼCortexのアーキテクチャ
  - https://github.com/cortexproject/cortex
- 主なコンポーネント
  - Distributor
  - Ingester
  - Chunk Store
  - Querier
  - **Table Manager**



- 保存するログのretentionを管理
- 一部Chunk Storeには非対応

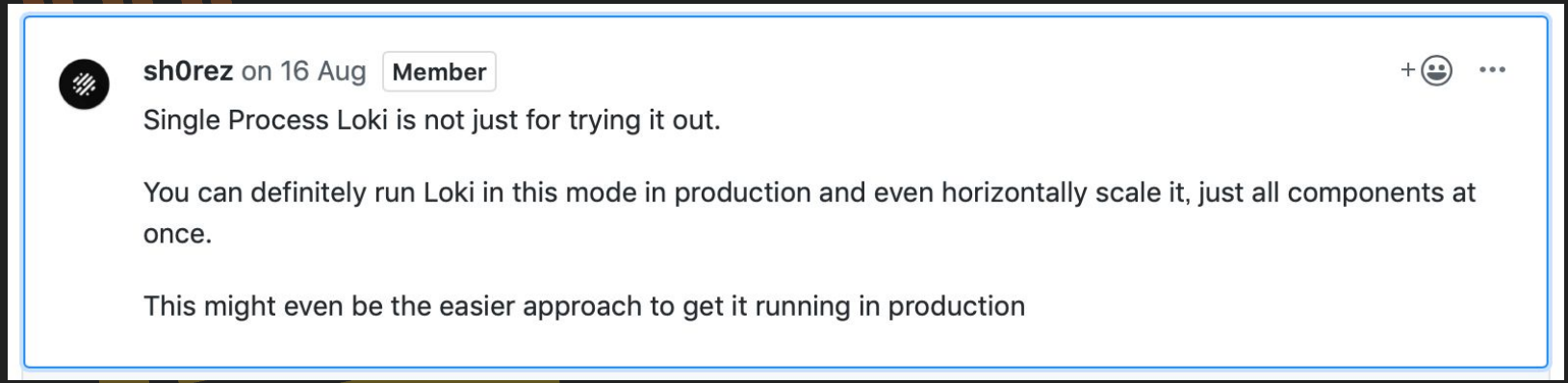
lokiのアーキテクチャ

- ほぼCortexのアーキテクチャ
  - https://github.com/cortexproject/cortex
- 主なコンポーネント
  - Distributor
  - Ingester
  - Chunk Store
  - Querier
  - Table Manager



lokiのコンポーネントについて

- 様々なコンポーネントがあるが、全て同一の実行バイナリ
  - シングルプロセスで全てのコンポーネントを動かすことも可能

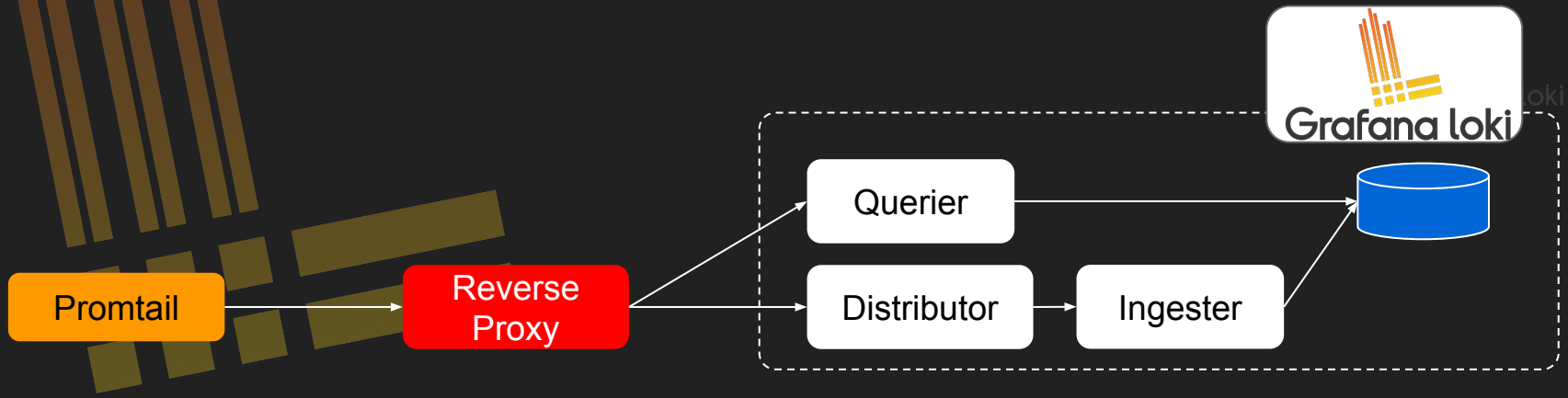


Lokiとは？ 再掲

- Log Aggregation System
  - Horizontally-Scalable
  - Highly-Available
  - **Multi-tenant**
- Grafana Labsが主体となって開発
  - Grafana連携
- OSS
  - https://github.com/grafana/loki
  - 最新はv1.2.0(2019/1/14時点)

マルチテナント

- テナントを識別するHTTPヘッダを付与することで実現
  - X-Scope-OrgID: <tenant ID>
- Loki自体にテナントの認証の仕組みは含まれていない
  - Nginx + Basic AuthやOAuth2 Proxyのようなリバースプロキシを用意する必要がある
  - 上記のヘッダは認証リバースプロキシでセットされるべき(と書いてありました)



Lokiとは？ 再掲

- Log Aggregation System
  - Horizontally-Scalable
  - Highly-Available
  - Multi-tenant
- Grafana Labsが主体となって開発
  - **Grafana連携**
- OSS
  - https://github.com/grafana/loki
  - 最新はv1.2.0(2019/1/14時点)

Grafanaからログの検索

- Exploreでデータソースを Lokiに指定すると利用可能

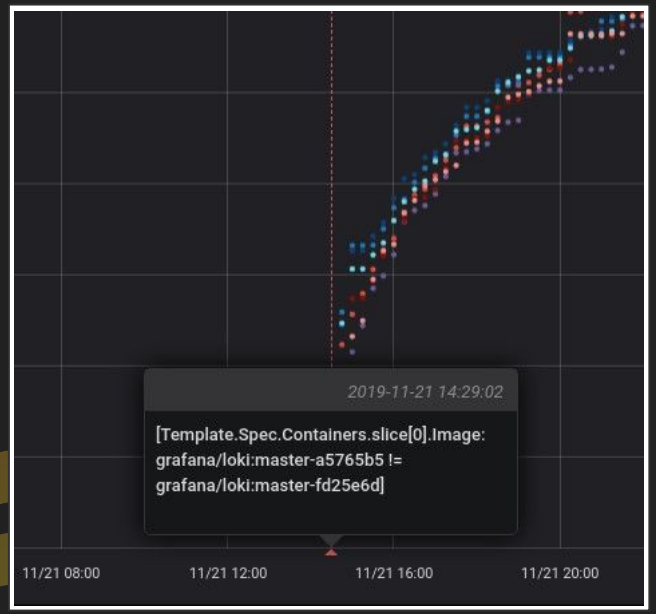
Grafana連携~ダッシュボードでログを表示~

- Log Panelが Grafana v6.4+ で利用可能
  - document: https://grafana.com/docs/grafana/latest/features/panels/logs/

```
2019-09-30 15:24:33  t=2019-09-30T13:24:33+0000 lvl=info msg="Alert Rule returned no data" logger=alerting.evalContext ruleId=
                     29 name="Data channel RTT latency alert" changing state to=ok
2019-09-30 15:24:33  t=2019-09-30T13:24:33+0000 lvl=info msg=Requesting logger=data-proxy-log url=https://management.azure.co
                     m/subscriptions
2019-09-30 15:24:33  t=2019-09-30T13:24:33+0000 lvl=info msg="Using token from cache" logger=data-proxy-log
2019-09-30 15:24:33  t=2019-09-30T13:24:33+0000 lvl=eror msg="Alert Rule Result Error" logger=alerting.evalContext ruleId=302
                     name="(NY) xo7-lbpp-01 - disk alert" error="Could not find datasource Data source not found" changing sta
                     te to=alerting
2019-09-30 15:24:33  t=2019-09-30T13:24:33+0000 lvl=eror msg="Alert Rule Result Error" logger=alerting.evalContext ruleId=299
                     name="(NY) xo7-lbpp-01 - CPU alert" error="Could not find datasource Data source not found" changing stat
                     e to=alerting
2019-09-30 15:24:33  t=2019-09-30T13:24:33+0000 lvl=eror msg="Alert Rule Result Error" logger=alerting.evalContext ruleId=296
                     name="(NY) xo7-lbpp-01 - conntrack alert" error="Could not find datasource Data source not found" changin
                     g state to=alerting
```

Grafana連携~Automatic Annotations~

- ログを動的にPanelへAnnotationとして付与可能



https://grafana.com/blog/2019/12/09/how-to-do-automatic-annotations-with-grafana-and-loki/

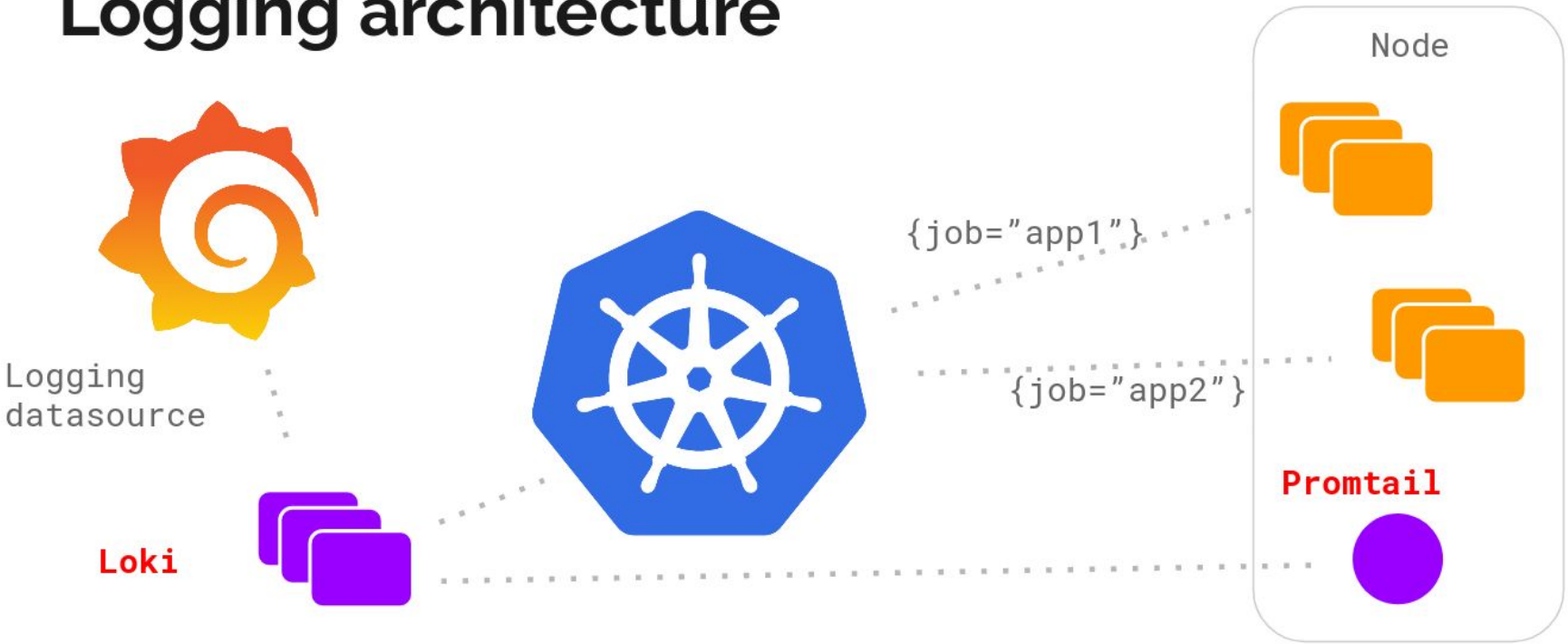
かめねこ

SAKURA internet Inc.
Evangelist, Infrastracture

@kameneko1004

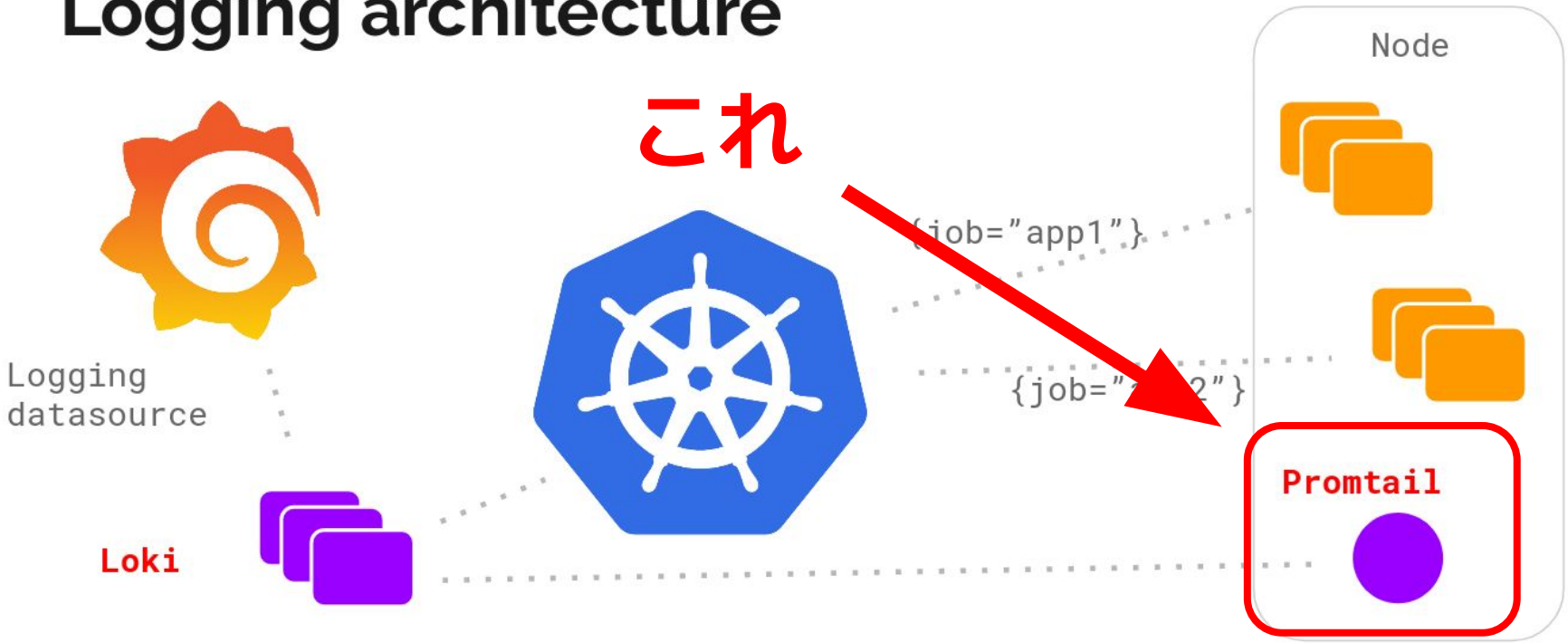
Promtailとは？

Logging architecture



https://grafana.com/blog/2018/12/12/loki-prometheus-inspired-open-source-logging-for-cloud-natives/

Logging architecture



https://grafana.com/blog/2018/12/12/loki-prometheus-inspired-open-source-logging-for-cloud-natives/

Promtailとは？

- 各Node上でPodなどのログを収集するエージェント
- Kubernetes以外やコンテナ以外もOK

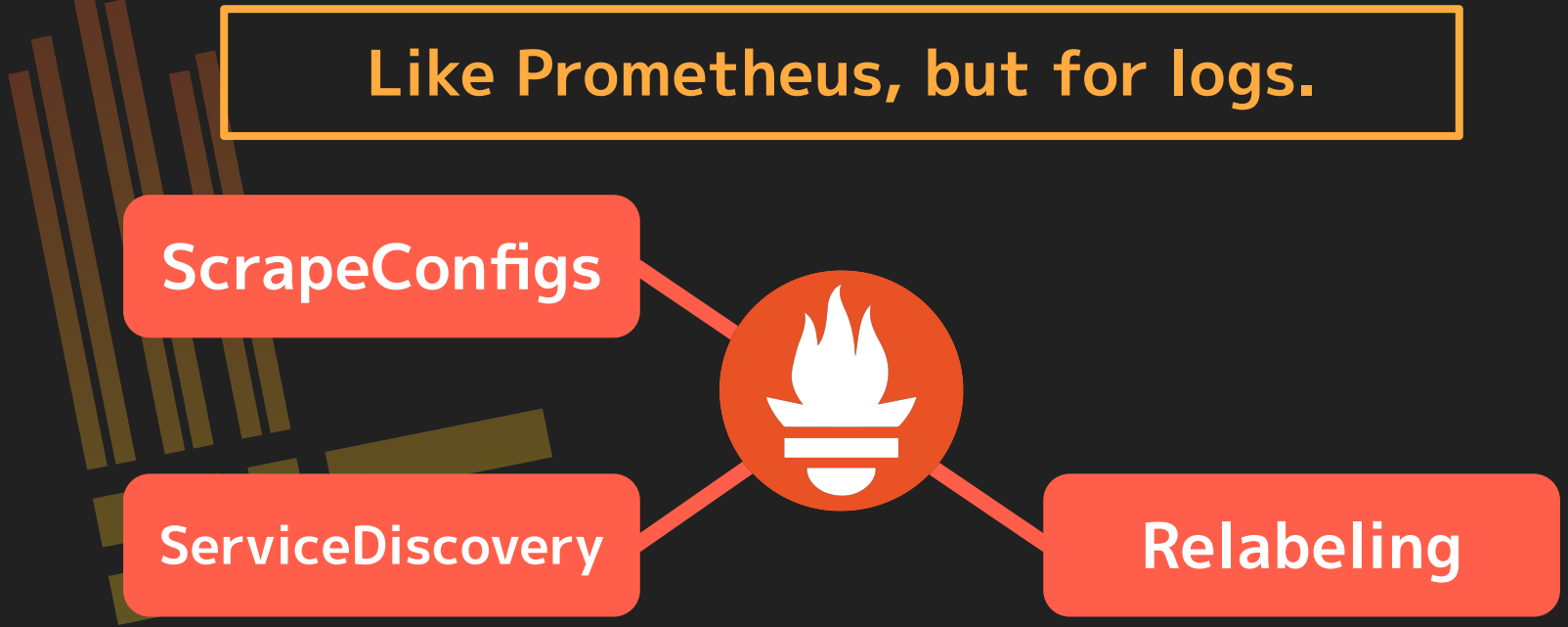
Promtailとは？

特徴

Like Prometheus, but for logs.

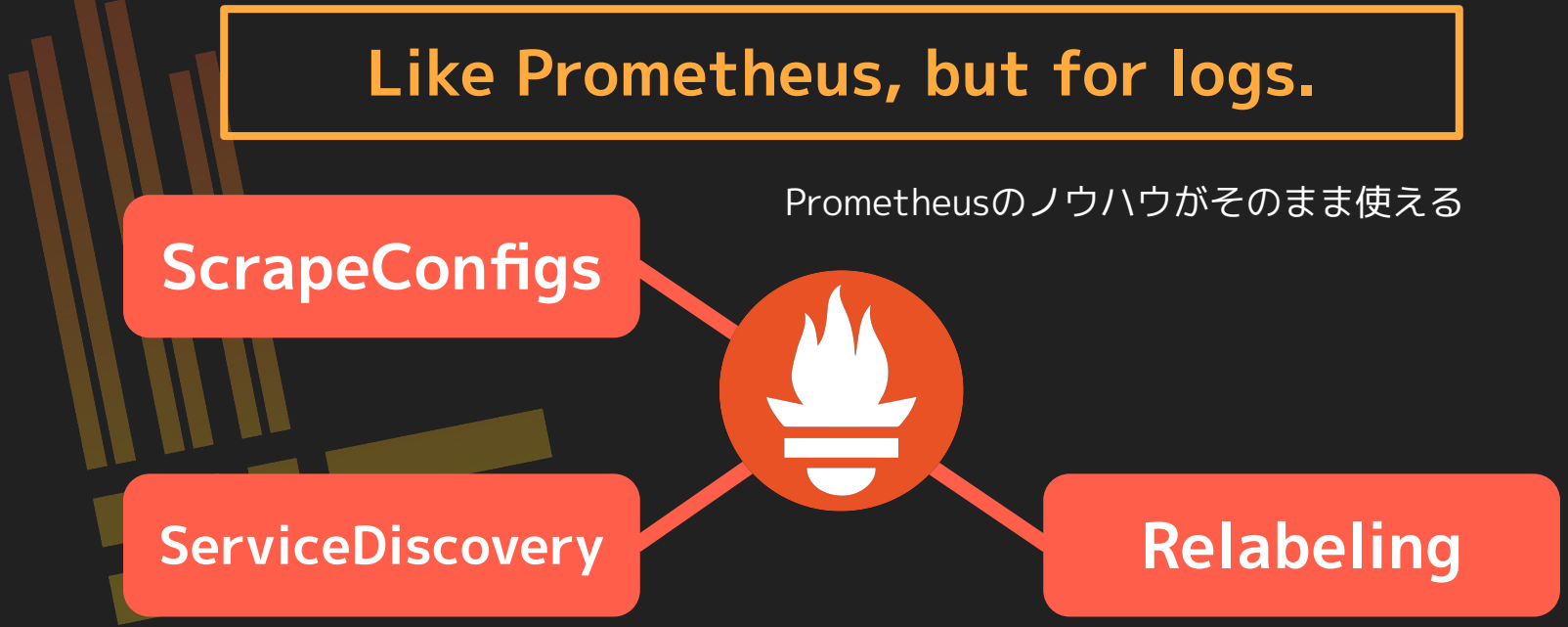
Promtailとは？

特徴



Promtailとは？

特徴

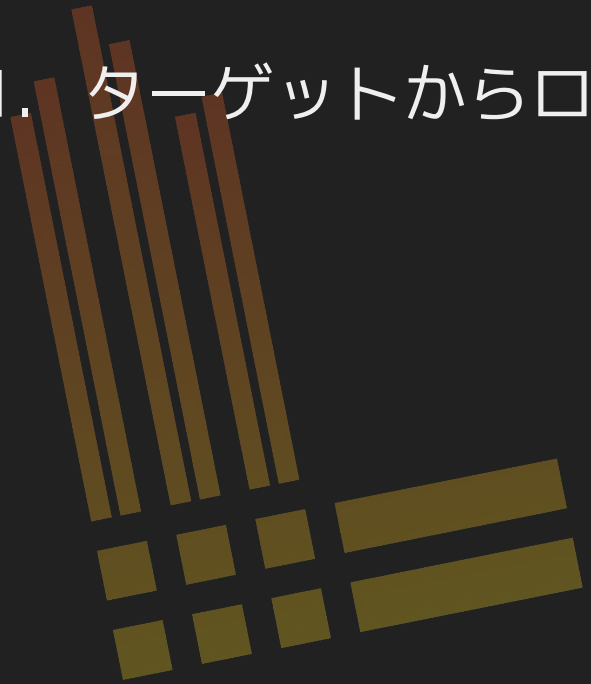


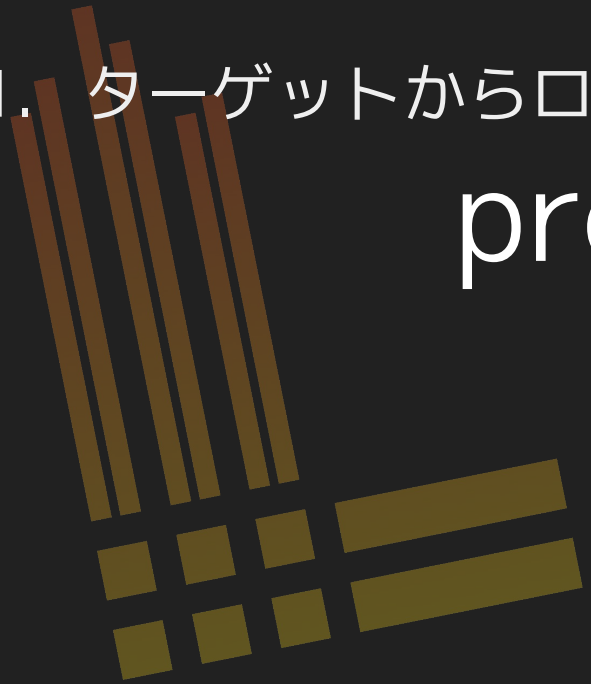
Promtailとは？

役割

1. ターゲットからログを収集してLokiに転送
2. ログからアラートを生成

1. ターゲットからログを収集してLokiに転送





1. ターゲットからログを収集してLokiに転送

promtail.yaml

promtail.yaml

- **ServiceDiscovery** で自動的にターゲットのログを追跡
  - **ScrapeConfigs** で定義
- **Relabeling** でログストリームにラベルを付与する

promtail.yaml

```
server:
  http_listen_port: 9080
  grpc_listen_port: 0

positions:
  filename: /tmp/positions.yaml

clients:
  - url: http://localhost:3100/loki
/api/v1/push

scrape_configs:
- job_name: system
  static_configs:
  - targets:
      - localhost
    labels:
      job: varlogs
      __path__: /var/log/*log
```

promtail.yaml

Promtail本体に関する設定

```
server:
  http_listen_port: 9080
  grpc_listen_port: 0

positions:
  filename: /tmp/positions.yaml

clients:
  - url: http://localhost:3100/loki
/api/v1/push

scrape_configs:
- job_name: system
  static_configs:
  - targets:
      - localhost
    labels:
      job: varlogs
      __path__: /var/log/*log
```

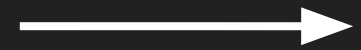
promtail.yaml

Positionファイルに関する設定

```
server:
  http_listen_port: 9080
  grpc_listen_port: 0

positions:
  filename: /tmp/positions.yaml

clients:
  - url: http://localhost:3100/loki
/api/v1/push

scrape_configs:
- job_name: system
  static_configs:
  - targets:
      - localhost
    labels:
      job: varlogs
      __path__: /var/log/*log
```

promtail.yaml

Lokiのインスタンスへの
接続に関する設定



```
server:
  http_listen_port: 9080
  grpc_listen_port: 0

positions:
  filename: /tmp/positions.yaml

clients:
  - url: http://localhost:3100/loki
/api/v1/push

scrape_configs:
- job_name: system
  static_configs:
  - targets:
      - localhost
    labels:
      job: varlogs
      __path__: /var/log/*log
```

promtail.yaml

ターゲットの検出に関する設定

```
server:
  http_listen_port: 9080
  grpc_listen_port: 0

positions:
  filename: /tmp/positions.yaml

clients:
  - url: http://localhost:3100/loki
/api/v1/push

scrape_configs:
- job_name: system
  static_configs:
  - targets:
      - localhost
    labels:
      job: varlogs
      __path__: /var/log/*log
```

scrape_configs:

もっともシンプルな構成

```
scrape_configs:
- job_name: system
  static_configs:
  - targets:
      - localhost
    labels:
      job: varlogs
      __path__: /var/log/*log
```

scrape_configs:

もっともシンプルな構成
__path__に指定したパスを参照する

localhost の /var/log/*log なログをスクレイプ

```
scrape_configs:
- job_name: system
  static_configs:
  - targets:
      - localhost
    labels:
      job: varlogs
      __path__: /var/log/*log
```

scrape_configs:

ServiceDiscovery も当然利用可能👍

```
scrape_configs:
- job_name: kubernetes-pods
  kubernetes_sd_configs:
  - role: pod
```

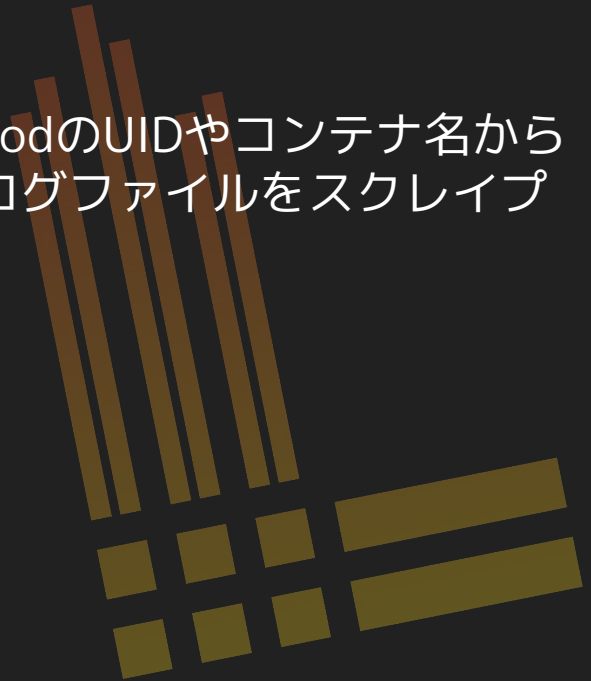
scrape_configs:

relabel_configs でラベルを加工

```
scrape_configs:
- job_name: kubernetes-pods
  kubernetes_sd_configs:
  - role: pod
  relabel_configs:
  - source_labels:
    - __meta_kubernetes_pod_label_app
    target_label: app
  - source_labels:
    - __meta_kubernetes_pod_node_name
    target_label: hostname
```

scrape_configs:

PodのUIDやコンテナ名から
ログファイルをスクレイプ



```
scrape_configs:
- job_name: kubernetes-pods
  kubernetes_sd_configs:
  - role: pod
  relabel_configs:
  - source_labels:
    - __meta_kubernetes_pod_uid
    - __meta_kubernetes_pod_container_name
    replacement:/var/log/pods/*$1/*.log
    separator: /
    target_label: __path__
```

ログからアラートを生成

ログからアラートを生成

- pipeline_stages
- 取得したログの値を加工・参照する
  - LogLevel をラベルのValueに
  - HTTPのステータスコードを
    ラベルのValueに

```
pipeline_stages
- match:
    selector: '{name="jaeger-agent"}'
    stages:
    - json:
        expressions:
        level: level
        components: components
    - labels:
        level:
        components:
```

例1

```
- job_name: kubernetes-pods-name
  kubernetes_sd_configs: ....
  pipeline_stages:
  - match:
      selector: '{name="promtail"}'
      stages:
      - regex:
          expression: '.*level=(?P<level>[a-zA-Z]+).*ts=(?P<timestamp>[T\d-:.Z]*)'
      - labels:
          level:
          component:
      - timestamp:
          format: RFC3339Nano
          source: timestamp
```

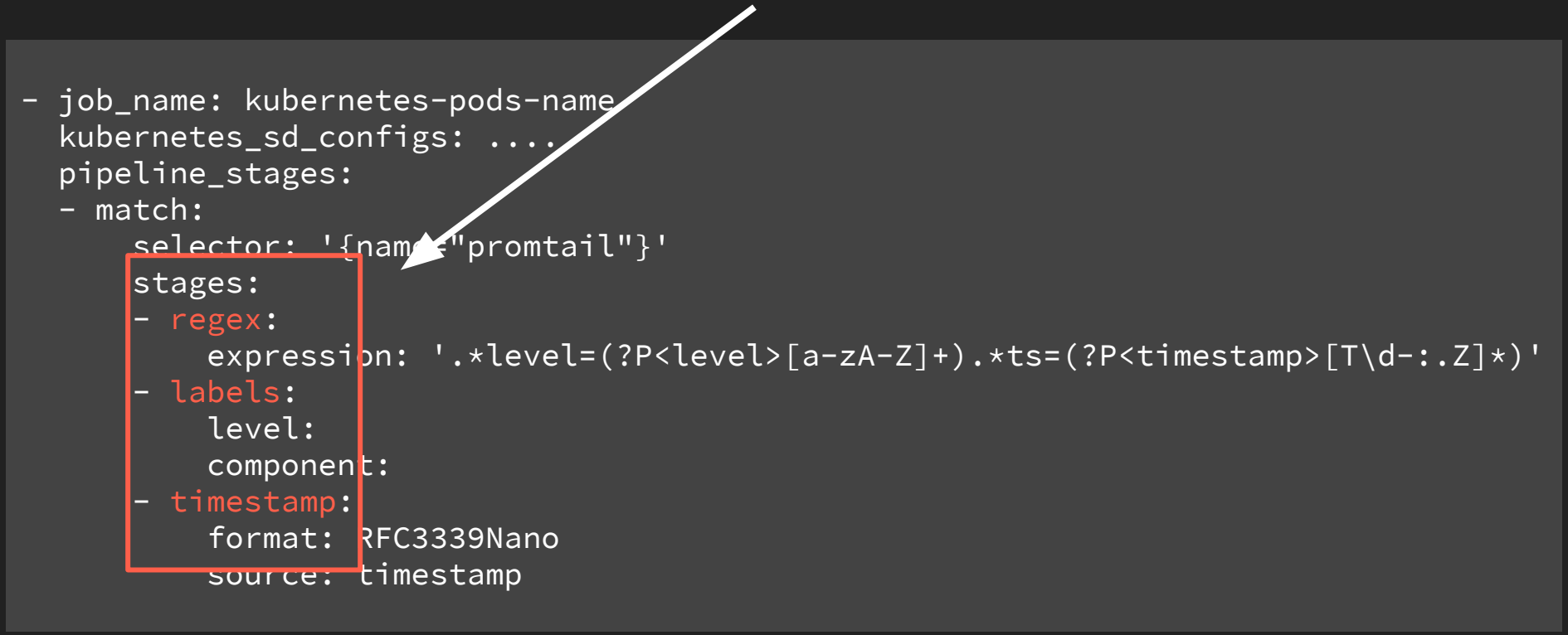
例1

selector で元となるログを指定
→ **LogQuery**

```
- job_name: kubernetes-pods-name
  kubernetes_sd_configs: ....
  pipeline_stages:
  - match:
      selector: '{name="promtail"}'
      stages:
      - regex:
          expression: '.*level=(?P<level>[a-zA-Z]+).*ts=(?P<timestamp>[T\d-:.Z]*)'
      - labels:
          level:
          component:
      - timestamp:
          format: RFC3339Nano
          source: timestamp
```

例1

各Stageごとに加工を行う



```
- job_name: kubernetes-pods-name
  kubernetes_sd_configs: ....
  pipeline_stages:
  - match:
      selector: '{name="promtail"}'
      stages:
      - regex:
          expression: '.*level=(?P<level>[a-zA-Z]+).*ts=(?P<timestamp>[T\d-:.Z]*)'
      - labels:
          level:
          component:
      - timestamp:
          format: RFC3339Nano
          source: timestamp
```

例1

regex:
ログストリームに対して正規表現で値を取得

```
- job_name: kubernetes-pods-name
  kubernetes_sd_configs: ....
  pipeline_stages:
  - match:
      selector: '{name="promtail"}'
      stages:
      - regex:
          expression: '.*level=(?P<level>[a-zA-Z]+).*ts=(?P<timestamp>[T\d-:.Z]*)'
      - labels:
          level:
          component:
      - timestamp:
          format: RFC3339Nano
          source: timestamp
```

例1

regex:
ログストリームに対して正規表現で値を取得

```
# 正規表現
expression: '.*level=(?P<level>[a-zA-Z]+).*ts=(?P<timestamp>[T\d-:.Z]*)'


# ログ
level=info ts=2020-01-14T23:44:05.12674019Z caller=tailer.go:77
component=tailer msg="start tailing file"
path=/var/log/pods/monitoring_prometheus-k8s-0_a993e0a4-e7a2-44bb-8716-c92d4
d24a1e5/prometheus-config-reloader/0.log
```

例1

level を参照して…

```
# 正規表現
expression: '.*level=(?P<level>[a-zA-Z]+).*ts=(?P<timestamp>[T\d-:.Z]*)'


# ログ
level=info ts=2020-01-14T23:44:05.12674019Z caller=tailer.go:77
component=tailer msg="start tailing file"
path=/var/log/pods/monitoring_prometheus-k8s-0_a993e0a4-e7a2-44bb-8716-c92d4
d24a1e5/prometheus-config-reloader/0.log
```

例1

かっこ内の正規表現に一致する値を
変数 <level> に格納

```
# 正規表現
expression: '.*level=(?P<level>[a-zA-Z]+).*ts=(?P<timestamp>[T\d-:.Z]*)'


# ログ
level=info ts=2020-01-14T23:44:05.12674019Z caller=tailer.go:77
component=tailer msg="start tailing file"
path=/var/log/pods/monitoring_prometheus-k8s-0_a993e0a4-e7a2-44bb-8716-c92d4
d24a1e5/prometheus-config-reloader/0.log
```

例1

ts を参照して…

```
# 正規表現
expression: '.*level=(?P<level>[a-zA-Z]+).*ts=(?P<timestamp>[T\d-:.Z]*)'


# ログ
level=info ts=2020-01-14T23:44:05.12674019Z caller=tailer.go:77
component=tailer msg="start tailing file"
path=/var/log/pods/monitoring_prometheus-k8s-0_a993e0a4-e7a2-44bb-8716-c92d4
d24a1e5/prometheus-config-reloader/0.log
```

例1

かっこ内の正規表現に一致する値を
変数 <timestamp> に格納

```
# 正規表現
expression: '.*level=(?P<level>[a-zA-Z]+).*ts=(?P<timestamp>[T\d-:.Z]*)'


# ログ
level=info ts=2020-01-14T23:44:05.12674019Z caller=tailer.go:77
component=tailer msg="start tailing file"
path=/var/log/pods/monitoring_prometheus-k8s-0_a993e0a4-e7a2-44bb-8716-c92d4
d24a1e5/prometheus-config-reloader/0.log
```

例1

label:
変数の値はラベルに置換可能

```
- job_name: kubernetes-pods-name
  kubernetes_sd_configs: ....
  pipeline_stages:
  - match:
      selector: '{name="promtail"}'
      stages:
      - regex:
          expression: '.*level=(?P<level>[a-zA-Z]+).*ts=(?P<timestamp>[T\d-:.Z]*)'
      - labels:
          level:
          component:
      - timestamp:
          format: RFC3339Nano
          source: timestamp
```

例1

label:
level=info
timestamp=2020-01-14T2…
が格納される

```
# 正規表現
expression: '.*level=(?P<level>[a-zA-Z]+).*ts=(?P<timestamp>[T\d-:.Z]*)'


# ログ
level=info ts=2020-01-14T23:44:05.12674019Z caller=tailer.go:77
component=tailer msg="start tailing file"
path=/var/log/pods/monitoring_prometheus-k8s-0_a993e0a4-e7a2-44bb-8716-c92d4
d24a1e5/prometheus-config-reloader/0.log
```

例1

timestamp:
指定の方法でタイムスタンプを指定

```
- job_name: kubernetes-pods-name
  kubernetes_sd_configs: ....
  pipeline_stages:
  - match:
      selector: '{name="promtail"}'
      stages:
      - regex:
          expression: '.*level=(?P<level>[a-zA-Z]+).*ts=(?P<timestamp>[T\d-:.Z]*)'
      - labels:
          level:
          component:
      - timestamp:
          format: RFC3339Nano
          source: timestamp
```

例2

```
- match:
    selector: '{name="sample-app"}'
    stages:
    - json:
        expressions:
          remote: remote_ip
          status: status
    - labels:
        remote:
        status:
```

例2

```
- match:
    selector: '{name="sample-app"}'
    stages:
    - json:
        expressions:
          remote: remote_ip
          status: status
    - labels:
        remote:
        status:
```

正規表現以外にも、
- json
- docker
- cri

などを指定可能

例2

```
- match:
    selector: '{name="sample-app"}'
    stages:
    - json:
        expressions:
          remote: remote_ip
          status: status
    - labels:
        remote:
        status:
```

json であれば、
ログ上のKeyを直接指定可能

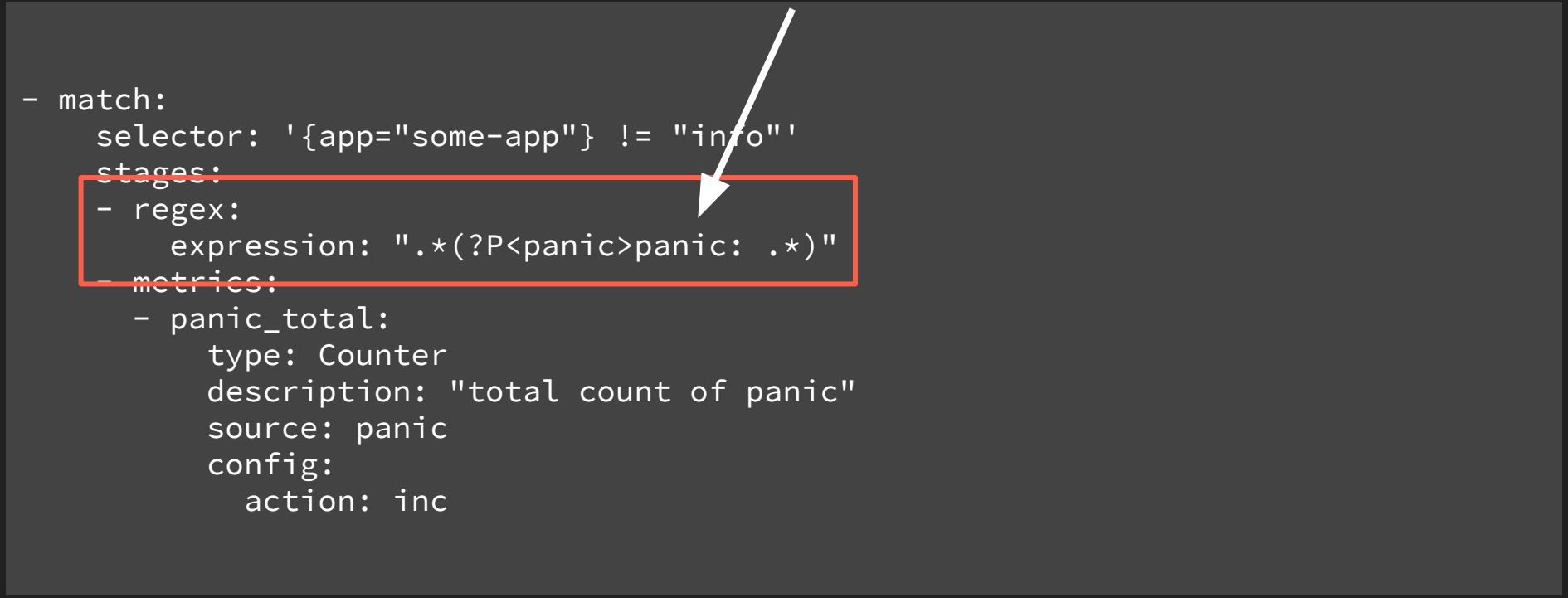

```
{"time":"2020-01-15T09:33:38.375111489Z","id":"","remote_ip":"10.42.29.199","host"
:"10.42.38.19:8080","method":"GET","uri":"/health","user_agent":"Go-http-client/1.
1","status":200,"error":"","latency":42187,"latency_human":"42.187
µs","bytes_in":0,"bytes_out":2}
```

例3

```
- match:
    selector: '{app="some-app"} != "info"'
    stages:
    - regex:
        expression: ".*(?P<panic>panic: .*)"
    - metrics:
      - panic_total:
          type: Counter
          description: "total count of panic"
          source: panic
          config:
            action: inc
```

例3

‘panic’ が含まれるログを抽出



```
- match:
    selector: '{app="some-app"} != "info"'
    stages:
    - regex:
        expression: ".*(?P<panic>panic: .*)"
    - metrics:
      - panic_total:
          type: Counter
          description: "total count of panic"
          source: panic
          config:
            action: inc
```

例3

metrics:
カスタムメトリクスを定義

```
- match:
    selector: '{app="some-app"} != "info"'
    stages:
    - regex:
        expression: ".*(?P<panic>panic: .*)"
    - metrics:
      - panic_total:
          type: Counter
          description: "total count of panic"
          source: panic
          config:
            action: inc
```

例3

```
- match:
    selector: '{app="some-app"} != "info"'
    stages:
    - regex:
        expression: ".*(?P<panic>panic: .*)"
    - metrics:
      - panic_total:
          type: Counter
          description: "total count of panic"
          source: panic
          config:
            action: inc
```

メトリクス名

例3

```
- match:
    selector: '{app="some-app"} != "info"'
    stages:
    - regex:
        expression: ".*(?P<panic>panic: .*)"
    - metrics:
      - panic_total:
          type: Counter
          description: "total count of panic"
          source: panic
          config:
            action: inc
```

メトリクスのタイプ

例3

```
- match:
    selector: '{app="some-app"} != "info"'
    stages:
    - regex:
        expression: ".*(?P<panic>panic: .*)"
    - metrics:
      - panic_total:
          type: Counter
          description: "total count of panic"
          source: panic
          config:
            action: inc
```

説明

例3

```
- match:
    selector: '{app="some-app"} != "info"'
    stages:
    - regex:
        expression: ".*(?P<panic>panic: .*)"
    - metrics:
      - panic_total:
          type: Counter
          description: "total count of panic"
          source: panic
          config:
            action: inc
```

メトリクスを変化させる
変数を指定

例3

```
- match:
    selector: '{app="some-app"} != "info"'
    stages:
    - regex:
        expression: ".*(?P<panic>panic: .*)"
    - metrics:
      - panic_total:
          type: Counter
          description: "total count of panic"
          source: panic
          config:
            action: inc
```

対象の変数にマッチしたら
行うアクションを指定
← inc: インクリメント

例3

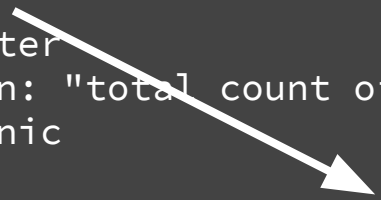
promtail/metricsで
‘promtail_custom_’から始まる
メトリクスが生成される

```
- match:
    selector: '{app="some-app"} != "info"'
    stages:
    - regex:
        expression: ".*(?P<panic>panic: .*)"
    - metrics:
      - panic_total:
          type: Counter
          description: "total count of panic"
          source: panic
          config:
            action: inc
```

例3

promtail/metricsで
‘promtail_custom_’から始まる
メトリクスが生成される

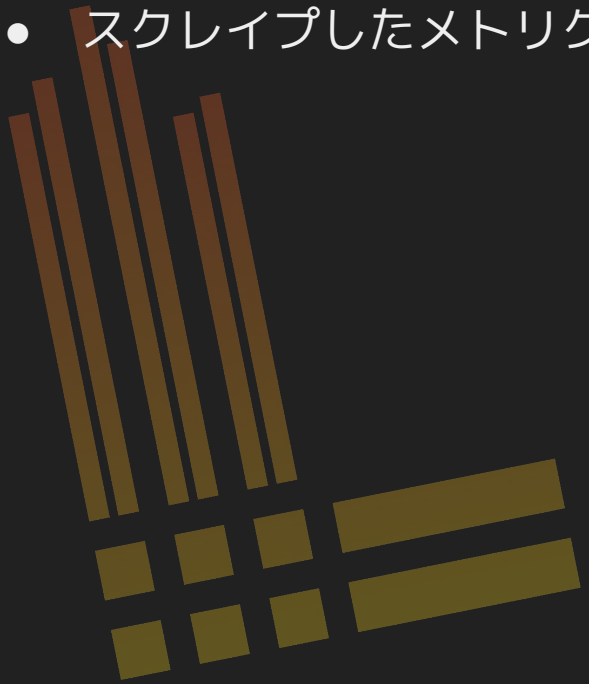
```
- match:
    selector: '{app="some-app"} != "info"'
    stages:
    - regex:
        expression: ".*(?P<panic>panic: .*)"
    - metrics:
      - panic_total:
          type: Counter
          description: "total count of panic"
          source: panic
          config:
            action: inc
```



promtail_custom_panic_total

カスタムメトリクスを元にアラート

- Prometheusから [promtail-host]:3101/metrics をスクレイプする
- スクレイプしたメトリクスを元に、アラートを生成する

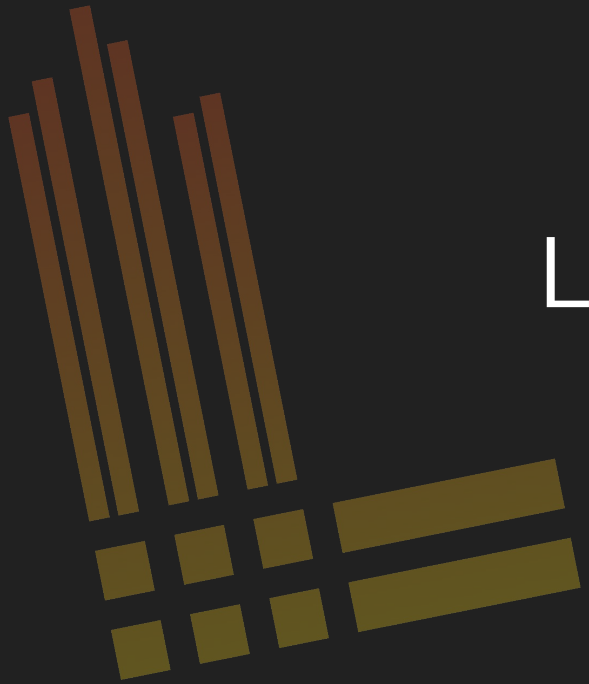


DEMO

可能であれば…(˘ω˘)

ドキュメント

https://github.com/grafana/loki/tree/master/docs/clients/promtail



Thank you!
Let’s Logging.