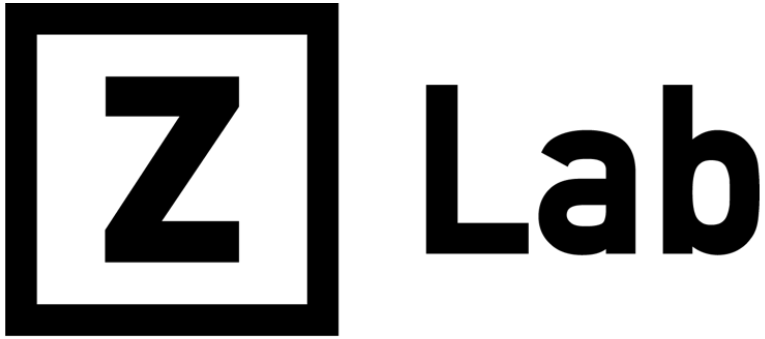
Loki入門

Profile

上村　真也

- 所属: Z Lab
- Twitter: @uesyn
- このMeetupの運営やってます



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

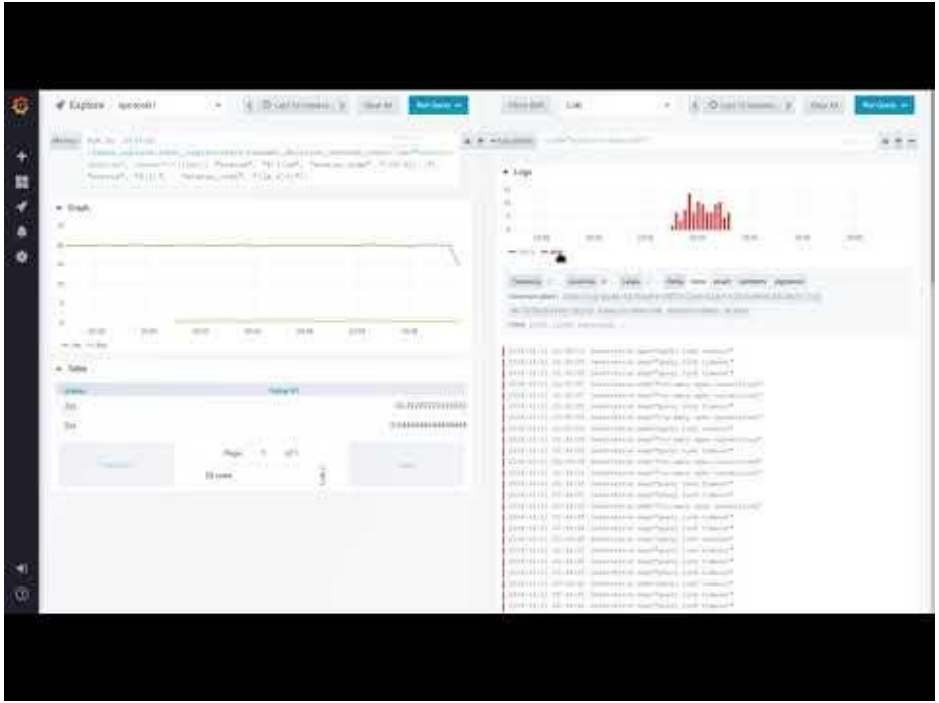
このスライドは個人の見解であり、
所属する組織の公式見解ではありません。

Lokiとは？

- Log Aggregation System
  - Horizontally-scalable
  - Highly-available
  - Multi-tenant
- Grafana Labs
  - Grafana連携(データソースの一つとして選択可能)
- OSS
  - https://github.com/grafana/loki

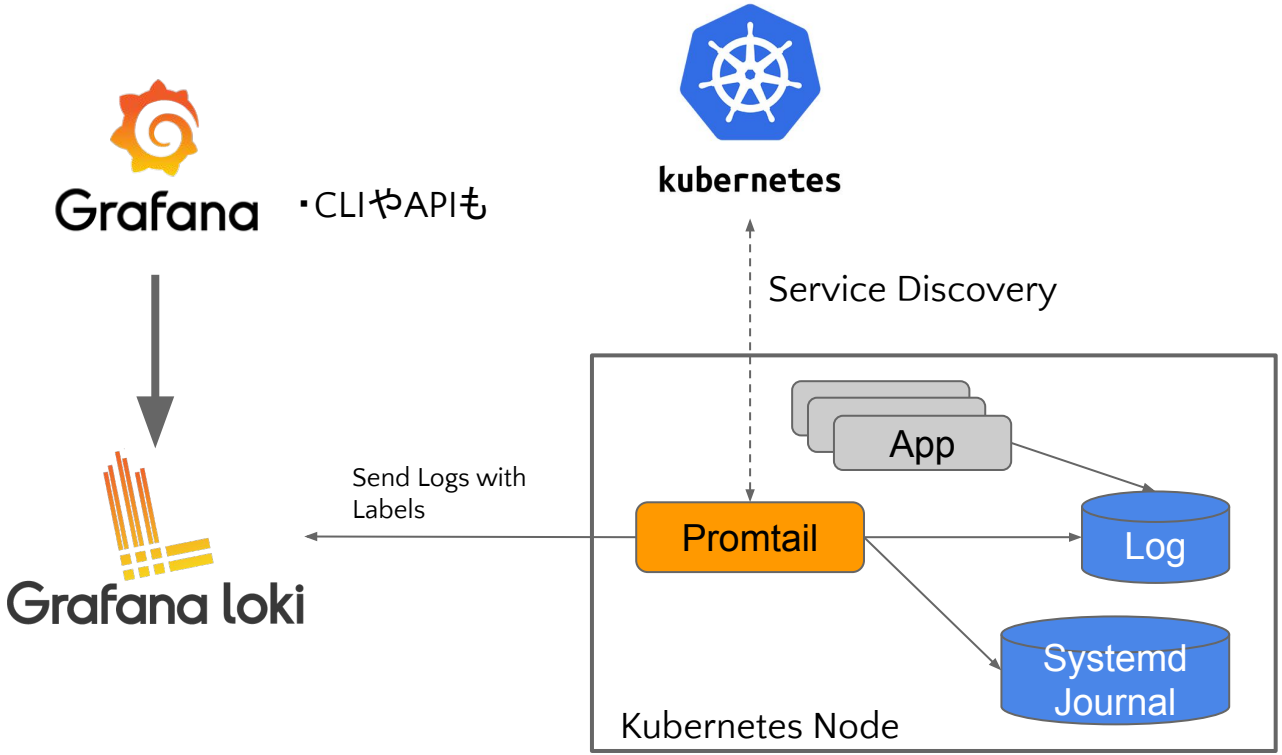
https://grafana.com/oss/loki

実施の画面



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

KubernetesとLoki



Lokiを動かす

Running Loki

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

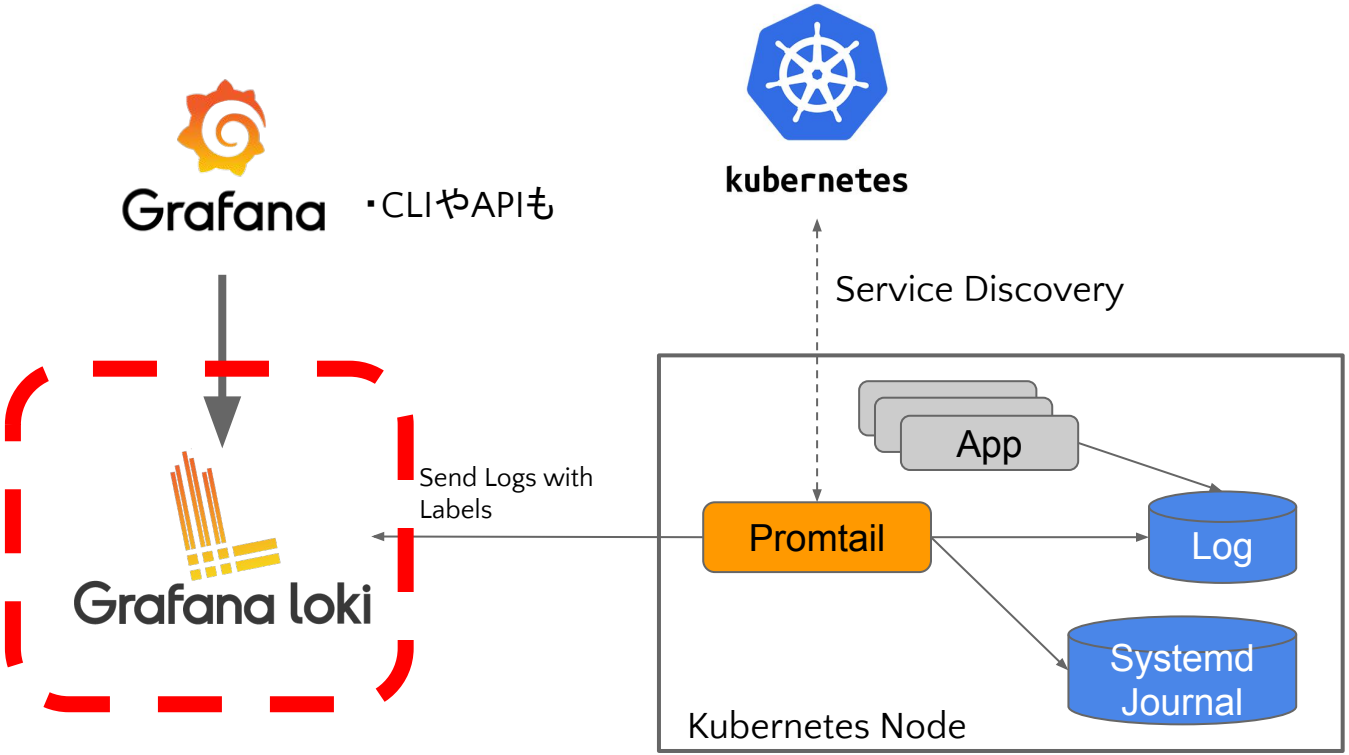
- Get a free hosted demo of Grafana Cloud: Logs
- Run Loki locally with Docker
- Use Helm to deploy on Kubernetes
- Build Loki from source
- Get inspired by our production setup

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

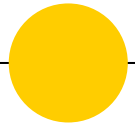
https://github.com/grafana/loki/blob/master/production/README.md#running-loki

再掲

KubernetesとLoki



loki



lokiについて

- Like Prometheus, but for logs
  - **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のように絞り込む

例: {app="nginx", env="dev"}

- 全文検索で用いられるようなテキスト処理はしない
  - ログに対して形態素解析して～など

logの送り側がラベルをつけて送信

lokiとCortex

- ほぼCNCFのCortexのアーキテクチャ
  - Cortexから派生しているため
- Cortexについては
  - https://speakerdeck.com/uesyn/cortexfalsehua-wokubecondewen-kitakatutatuteiuhua
  - 上記の情報は古いかも

Architecture

Loki mainly consists of three and a half individual services that achieve this in common. The high level architecture is based on Cortex, most of their documentation usually applies to Loki as well.

Distributor

The distributor can be considered the "first stop" for the log lines ingested by the agents (e.g. Promtail).

It performs validation tasks on the data, splits it into batches and sends it to multiple Ingesters in parallel.

Distributors communicate with ingesters via gRPC. They are *stateless* and can be scaled up and down as needed.

Refer to the Cortex docs for details on the internals.

Ingester

The ingester service is responsible for de-duplicating and persisting the data to long-term storage backends (DynamoDB, S3, Cassandra, etc.).

Ingesters are semi-*stateful*, the maintain the last 12 hours worth of logs before flushing to the Chunk store. When restarting ingesters, care must be taken no to lose this data.

More details can be found in the Cortex docs.

Chunk store

Loki is not database, so it needs some place to persist the ingested log lines to, for a longer period of time.

The chunk store is not really a service of Loki in the traditional way, but rather some storage backend Loki uses.

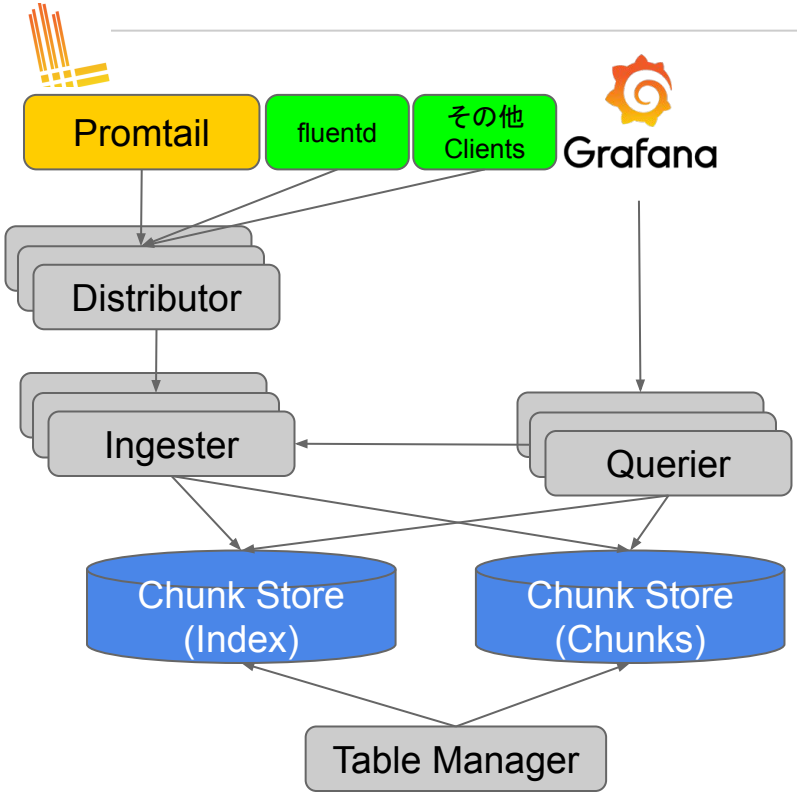
It consists of a key-value (KV) store for the actual **chunk data** and an **index store** to keep track of them. Refer to Storage for details.

The Cortex docs also have good information about this.

https://github.com/grafana/loki/blob/master/docs/loki/README.md

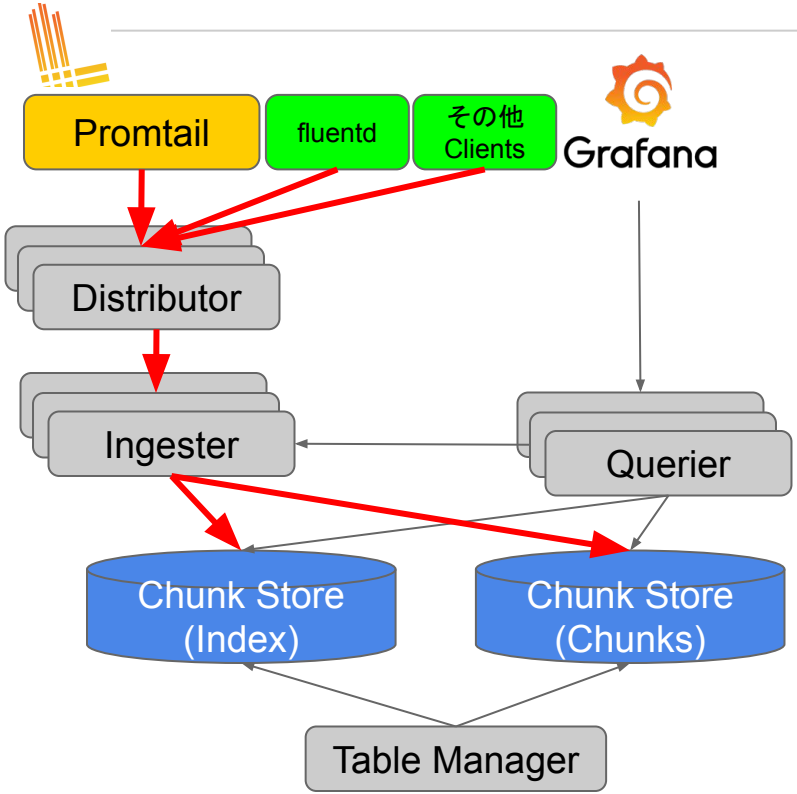
lokiのアーキテクチャ

- 主なコンポーネント
  - Distributor
  - Ingester
  - Querier
  - Chunk Store
  - Table Manager



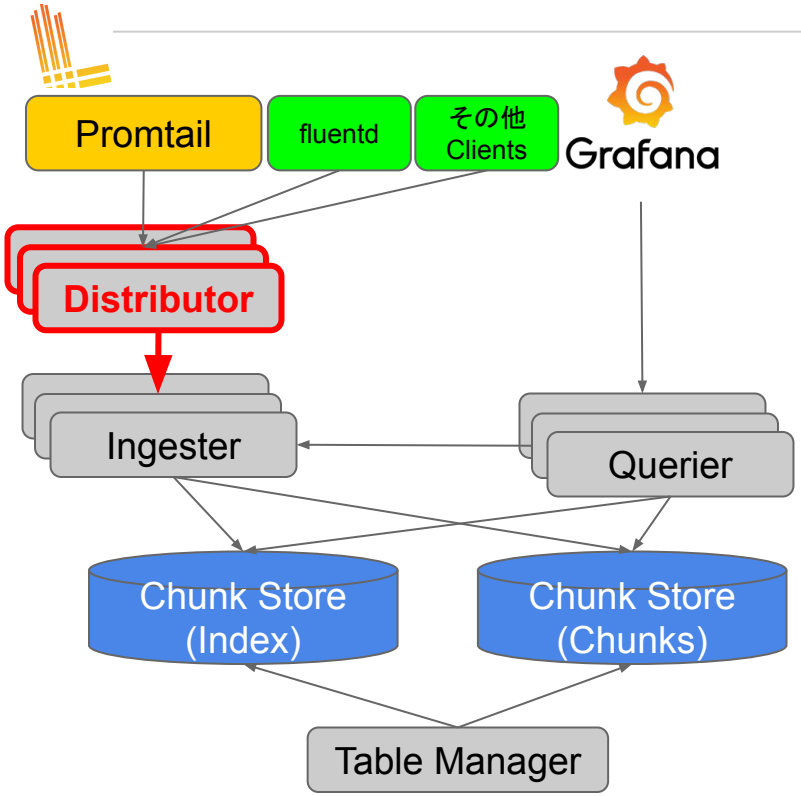
ログの書き込みの流れ

- 赤矢印の流れで書き込みされる



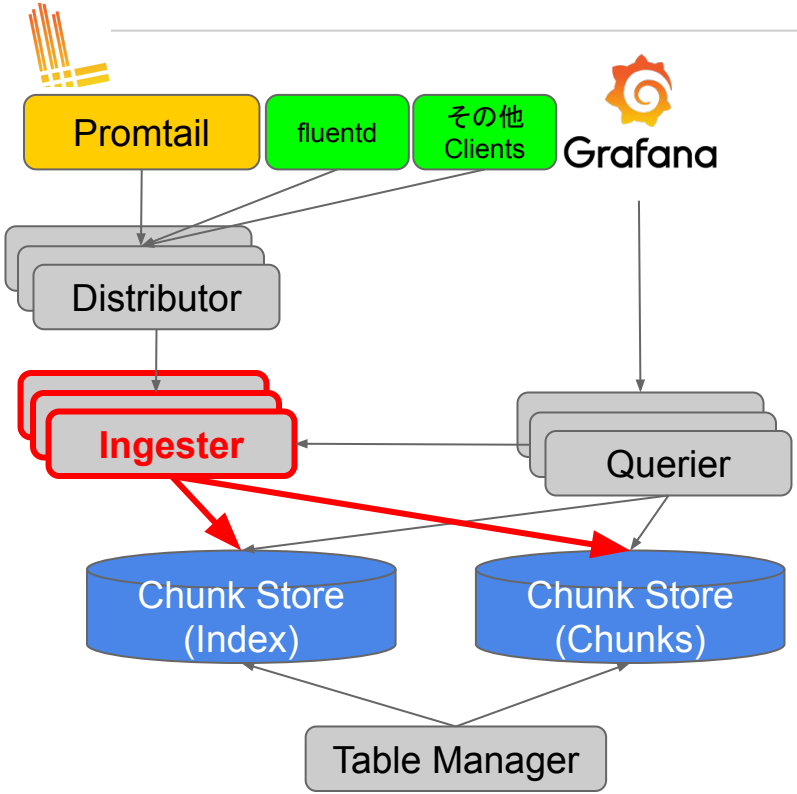
Distributor

- Clientから受けたログをIngesterへ
- ヘッダに含まれるIDでユーザを識別
  - マルチテナントで利用する場合は別途 Frontend の実装が必要
- ステートレスなコンポーネント



Ingester(1/2)

- Distributorから受けたログをChunk Storeへ
- セミステートフルなコンポーネント
  - 直近のログデータを保持



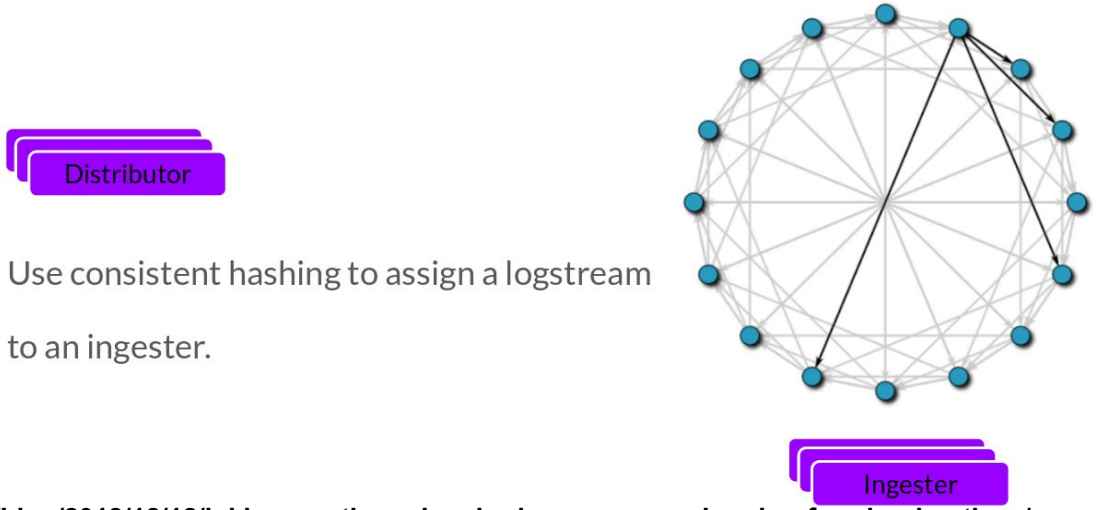
Ingester(2/2)

- Logをまとめて(Chunk)をChunk Storeへ
  - Chunk Storeへの頻繁な書き込みの抑制
  - Chunkがいっぱいになったら書き込む (もしかしたら一定時間経過後も？)

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

DistributorとIngester

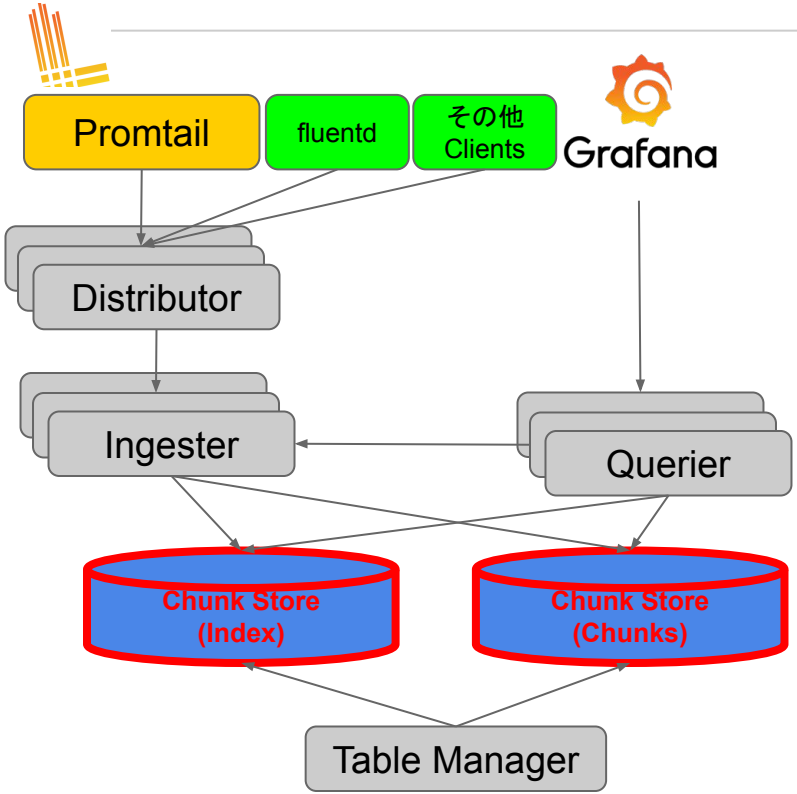
- Distributorはコンシステントハッシングにより Ingesterを決定



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

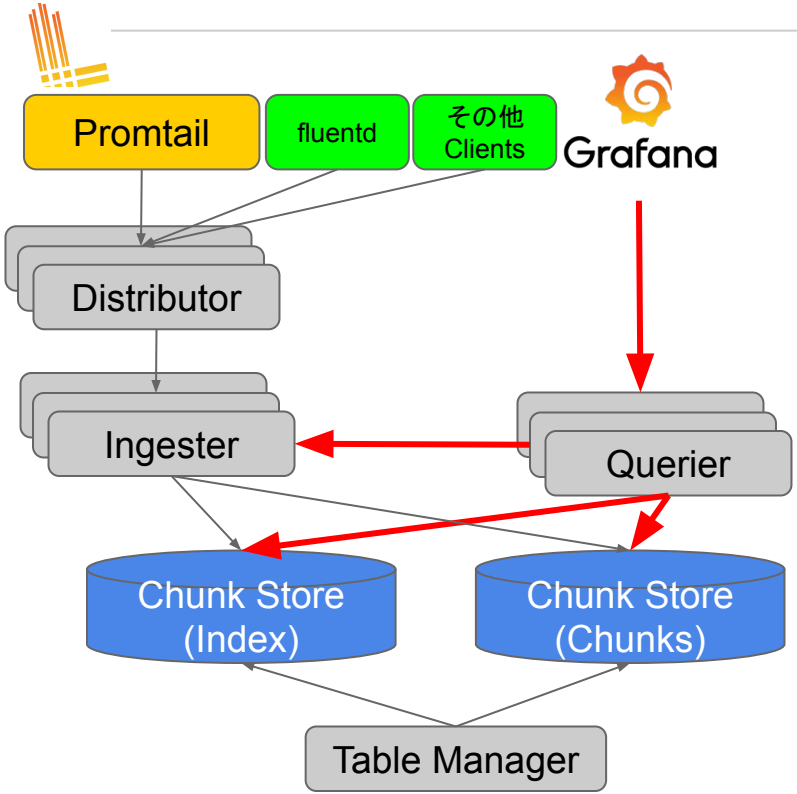
Chunk Store

- 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
    - ラベルなどのメタデータを保持
  - **Chunks**
    - 実際のログを保持
- 利用可能なDB
  - **Index**
    - Local
    - DynamoDB
    - Bigtable
    - Cassandra
  - **Chunks**
    - Local
    - Cloud Storage
    - S3



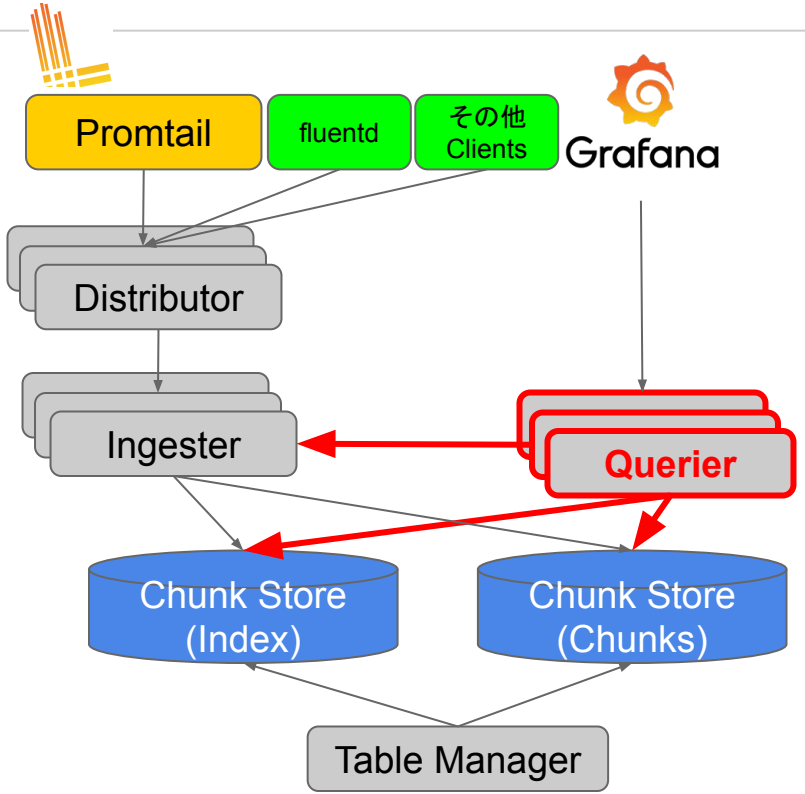
lokiのRead Path

- 赤矢印の流れ



Querier

- ログのクエリを受ける
- 直近のログはIngester, それ以外はChunk Store



LogのRetention

- Table Managerにより制御
- Table Manager
  - Retentionを管理
  - 基本的にindexを対象

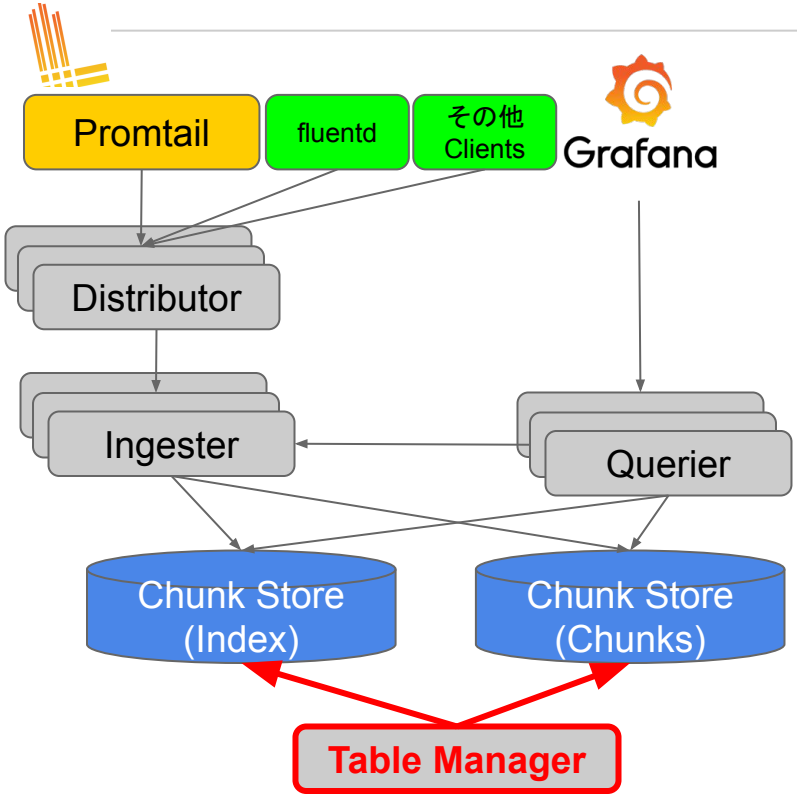


Table ManagerによるRetention

- Table level retention
- Retentionに関する設定
  - indexの下のperiod
    - 1 table の期間
  - retention_deletes_enabled
    - Retentionを有効にするか
  - retention_period
    - Retentionの期間
- 以下のテーブルが保持される
  - 現在アクティブに使われているテーブル
  - 過去のtable数(retention_period ÷ period)

```
...

schema_config:
  configs:
  - from: "2018-04-15"
    index:
      period: 12h
      prefix: index_
...

table_manager:
  retention_deletes_enabled: true
  retention_period: 24h
```

24h(retention_period) ÷ 12h(period) = **2 table**
次のページでこの設定の例を説明

Table ManagerによるRetention

Table ManagerによるRetention

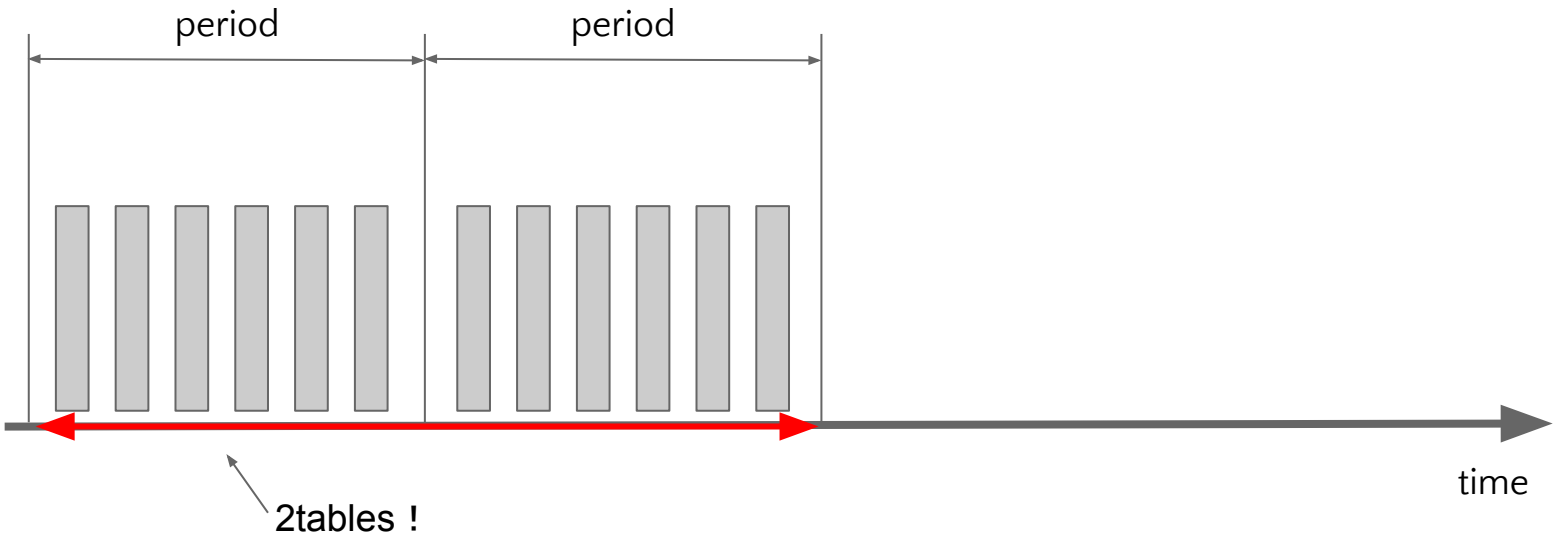


Table ManagerによるRetention

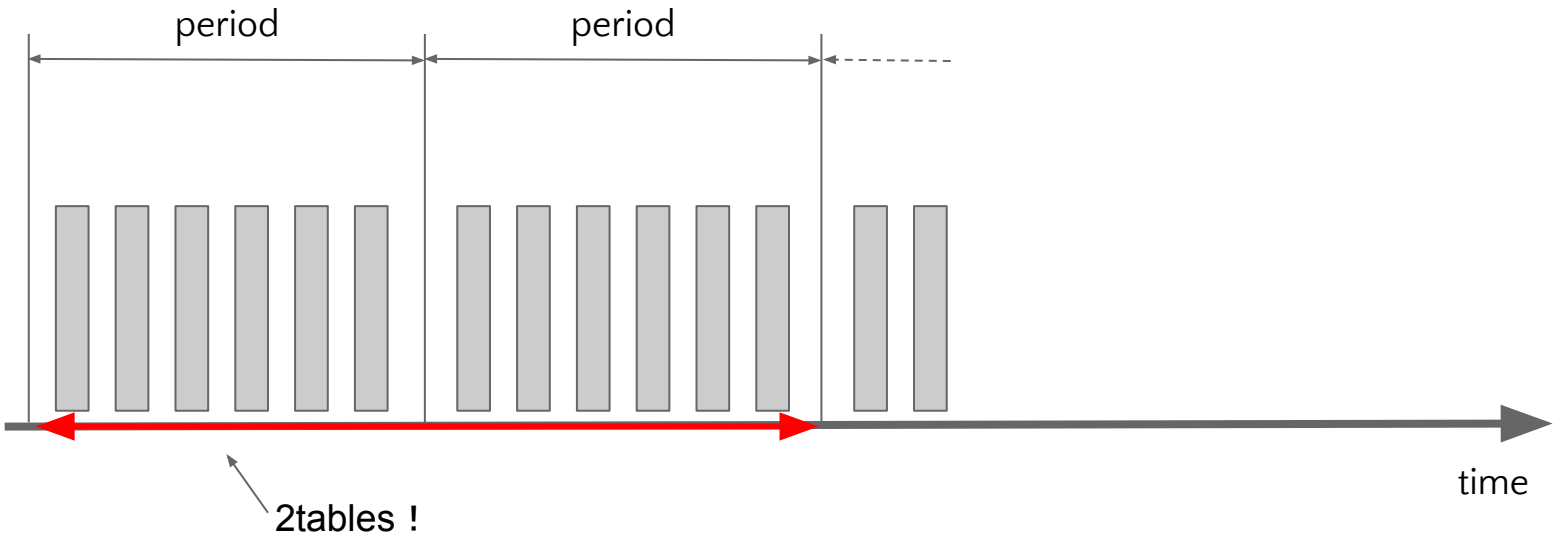


Table ManagerによるRetention

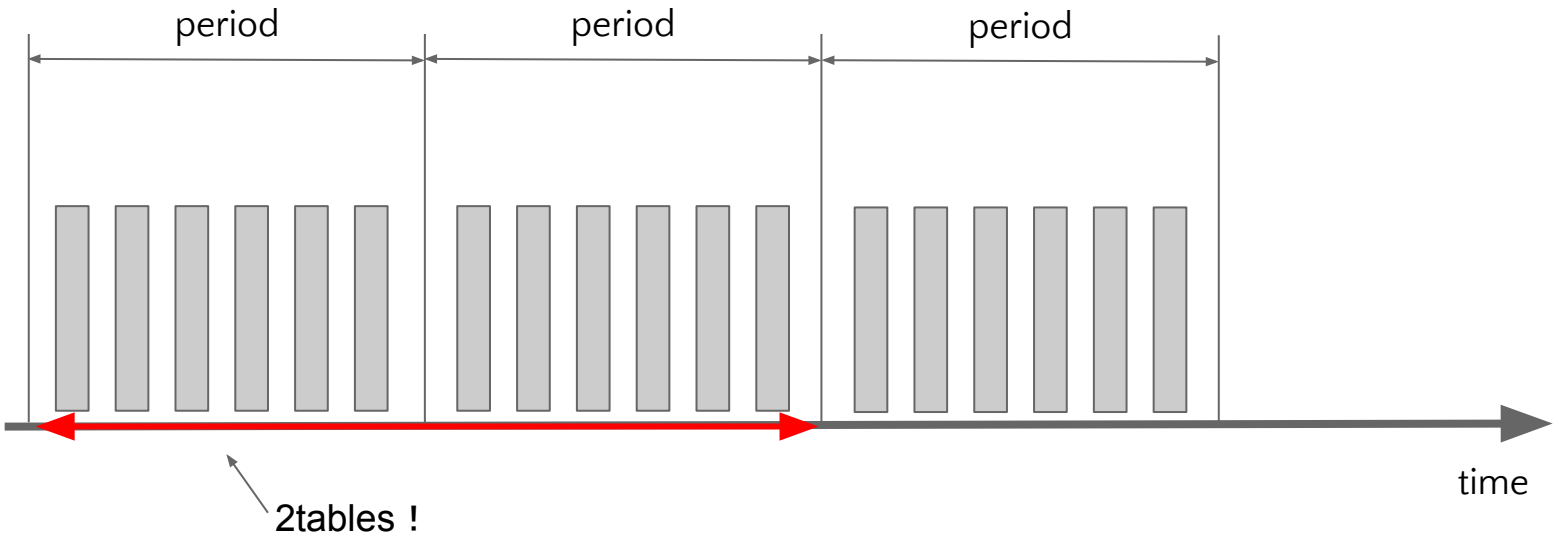


Table ManagerによるRetention

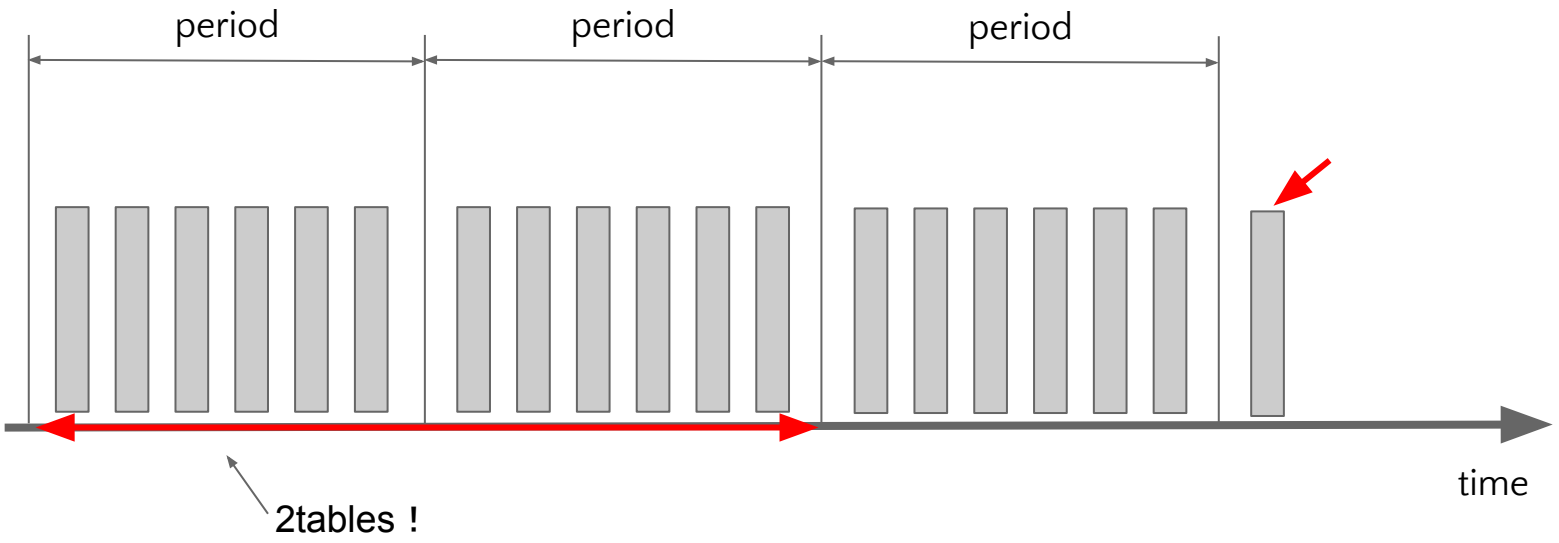
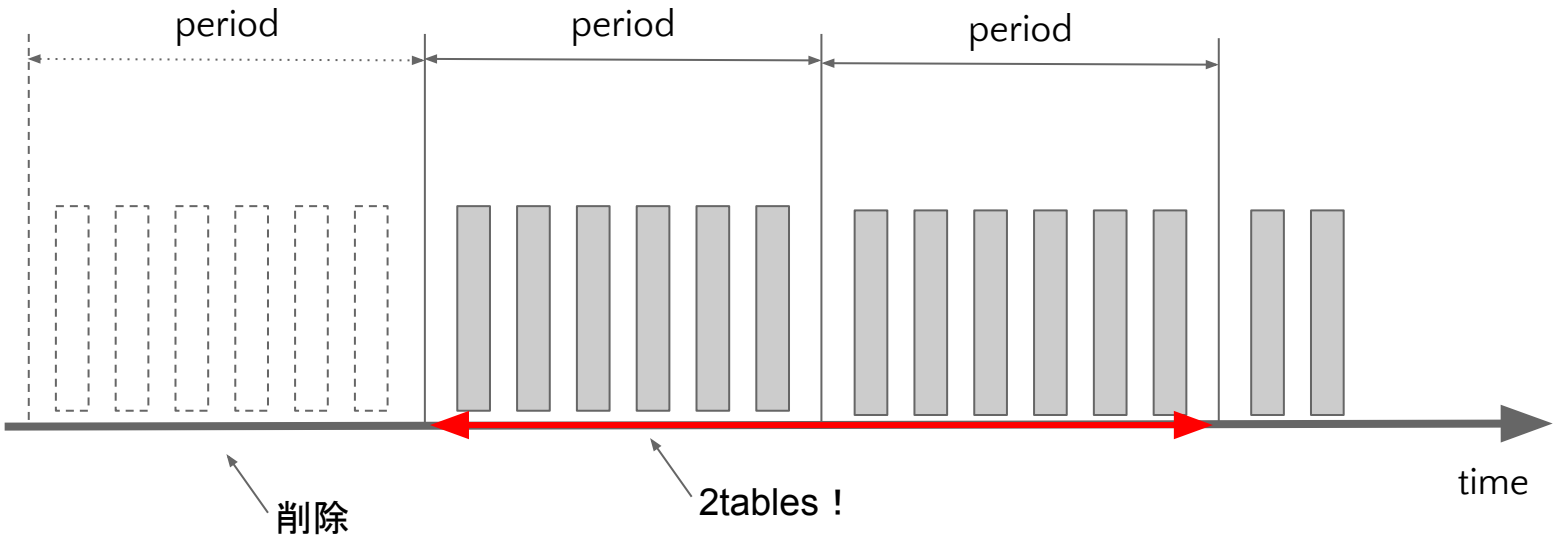
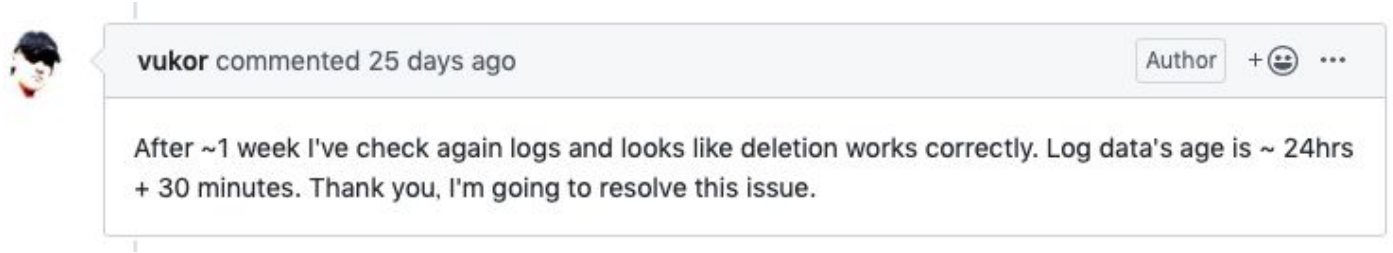


Table ManagerによるRetention



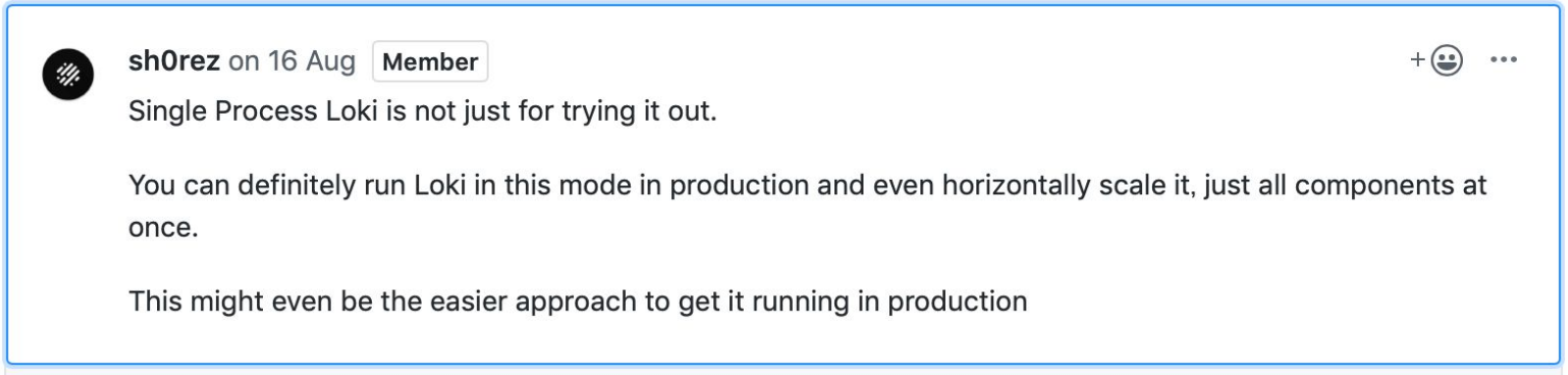
LogのRetentionのバグ？仕様？

- IndexやChunkが削除されても削除部分のログが参照できる仕様？バグ？が存在
  - Lokiのキャッシュの仕組みがまだ理解できておらず説明できません …
  - Issue : Retention/Deleting old data doesn't work #881(この人は解決しているらしい)



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、全て一つの実行ファイルから起動
  - 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の設定でどのコンポーネントを動かすかを指定



sh0rez on 16 Aug **Member**

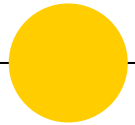
Single Process Loki is not just for trying it out.

You can definitely run Loki in this mode in production and even horizontally scale it, just all components at once.

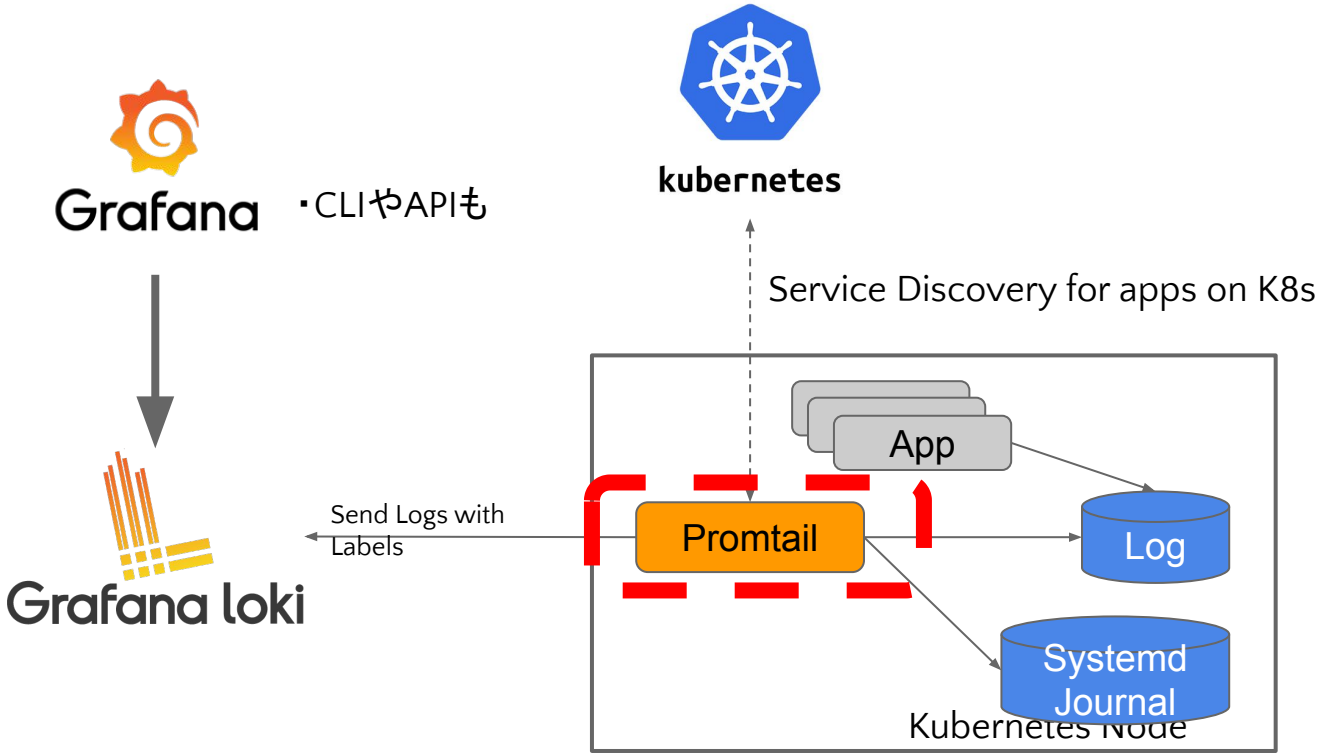
This might even be the easier approach to get it running in production

- コンポーネントを分離して動かすドキュメントはない(2019/9/17時点)

Promtail



KubernetesとLoki(再掲)



Promtail

- lokiへラベル付きのログを送信するためのエージェント
  - Prometheusと似たような設定項目
  - Kubernetesのアプリのログを対象する場合 Service Discoveryが利用可能
- Kubernetes上でPromtailをデプロイする例
  - DaemonSetとして起動しノードで動くコンテナのログや Systemd Journalを収集
  - 特定のアプリのログだけを対象とする場合は Sidecarとして起動

Promtailの設定について

- Prometheusの設定とよく似た項目
  - lokiへの接続方法
  - ログファイルのTarget Discovery
  - ログの処理に関する設定 等

```
- client:
  ...
  positions:
    filename: /run/promtail/positions.yaml
  server:
    http_listen_port: 3101
  target_config:
    sync_period: 10s
scrape_configs:
-  job_name: job1
  static_configs:
  ...
- job_name: job2
  kubernetes_sd_configs:
  ...
  relabel_configs:
  ...
  pipeline_stages:
  ...
```

Promtailの設定について

- Prometheusの設定とよく似た項目
  - lokiへの接続方法
  - ログファイルのTarget Discovery
  - ログの処理に関する設定 等

```
- client:
  ...
  positions:
    filename: /run/promtail/positions.yaml
  server:
    http_listen_port: 3101
  target_config:
    sync_period: 10s
scrape_configs:
-  job_name: job1
  static_configs:
  ...
- job_name: job2
  kubernetes_sd_configs:
  ...
  relabel_configs:
  ...
  pipeline_stages:
  ...
```

Promtailの設定について

- Prometheusの設定とよく似た項目
  - lokiへの接続方法
  - ログファイルのTarget Discovery
  - ログの処理に関する設定 等

```
- client:
  ...
  positions:
    filename: /run/promtail/positions.yaml
  server:
    http_listen_port: 3101
  target_config:
    sync_period: 10s
scrape_configs:
-  job_name: job1
  static_configs:
  ...
- job_name: job2
  kubernetes_sd_configs:
  ...
  relabel_configs:
  ...
  pipeline_stages:
  ...
```

static_configs

- Service Discoveryを利用しない設定
- labelsでログにつけるラベルを指定
- __path__ラベルでログのパスを指定
  - ワイルドカードを使った指定も可能

```
static_configs:
- targets:
    - localhost
  labels:
    job: someone_service
    host: yourhost
    __path__: /var/log/someone_service/*.log
```

再掲

Promtailの設定について

- Prometheusの設定とよく似た項目
  - lokiへの接続方法
  - ログファイルのTarget Discovery
  - ログの処理に関する設定 等

```
- client:
  ...
  positions:
    filename: /run/promtail/positions.yaml
  server:
    http_listen_port: 3101
  target_config:
    sync_period: 10s
scrape_configs:
-  job_name: job1
  static_configs:
  ...
- job_name: job2
  kubernetes_sd_configs:
  ...
  relabel_configs:
  ...
  pipeline_stages:
  ...
```

kubernetes_sd_configsとrelabel_configs

- kubernetes_sd_configs
  - Prometheusと同じ
- relabel_configs
  - ログを"読む前"の処理
  - relabelをすることで
    - ラベル付与
    - ログ収集対象のコンテナの選択
    - ログのPath指定 etc...

```
kubernetes_sd_configs:
  - role: pod
relabel_configs:
- action: relabelmap
  regex: __meta_kubernetes_pod_label_(.+)
- replacement: /var/log/pods/*$1/*.log
  separator: /
  source_labels:
  - __meta_kubernetes_pod_uid
  - __meta_kubernetes_pod_container_name
  target_label: __path__
```

再掲

Promtailの設定について

- Prometheusの設定とよく似た項目
  - lokiへの接続方法
  - ログファイルのTarget Discovery
  - ログの処理に関する設定 等

```
- client:
  ...
  positions:
    filename: /run/promtail/positions.yaml
  server:
    http_listen_port: 3101
  target_config:
    sync_period: 10s
scrape_configs:
-  job_name: job1
  static_configs:
  ...
- job_name: job2
  kubernetes_sd_configs:
  relabel_configs:
  ...
  pipeline_stages:
  ...
```

pipeline_stagesのstage

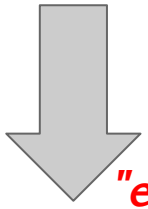
- ログを"読んだ後"の内容に対して処理
  - Prometheusのmetrics_relabel_configsに相当
- 各"stage"でログに対して様々な処理
  - ログをパースして ***extracted map*** へ
  - **extracted map** からtimestampをパース
  - **extracted map** からラベルの選択　etc…

```
pipeline_stages:
- json:
    expressions:
      log:
- json:
    expressions:
      output: msg
      level: level
      timestamp: time
      hostname: hostname
- output:
    source: output
- labels:
    hostname:
    level:
- timestamp:
    format: UnixNs
    source: timestamp
```

extracted mapについて

- ログの中身をパースした結果をmapへ
  - Mapの中身をさらにパースすることも可能
- stage間でextracted mapは共有

{time: "xxxx", message: "yyyy", host: "zzzz"}



"extracted map"

key	value
time	xxxx
message	yyyy
host	zzzz

ログをextracted mapへ

- **json stage**
  - ログがjson形式の場合に利用
- **regex stage**
  - 正規表現を用いる場合に利用

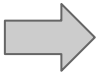
```
- json:
    expressions:
      time:
- regex:
    expression: "^(?P<year>\\d+)"
    source: "time"
```

extracted dataからラベル付与

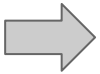
- **labels stage**
  - extracted mapから付与するラベルを選択

"extracted map"

key	value
message	yyyy
stream_type	stderr



```
- labels:
    stream: stream_type
```



付与されるラベル

stream: stderr

付与したいラベル名

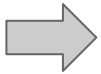
付与したい値を持つ extracted mapのkey

保存するログの選択

- **output stage**
  - lokiがログとして保存するコンテンツを選択する

"extracted map"

key	value
content	yyyy
stream_type	stderr



```
- output:
    source: content
```

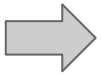
contentのvalueがログとして保存される

タイムスタンプのパース

- **timestamp stage**
  - extracted map中のタイムスタンプをパース

"extracted map"

key	value
timestamp	1562708916414



```
- timestamp:
    source: timestamp
    format: UnixMs
```

特定ラベルを持つものだけ処理

- **match stage**
  - 条件に一致したログのみ処理する stageを設定
    - logql stream selectorで指定
    - https://github.com/grafana/loki/blob/master/docs/querying.md#log-stream-selector
  - アプリごとに出力形式が異なる場合に便利

```
- match:
    selector: "{app=\"loki\"}"
    stages:
    ...
```

ログからメトリクスを生成

- **metrics stage**
  - extracted mapの値をPrometheusのメトリクスとして出力
    - Counter
    - Gauge
    - Histogram

Histogramの例

```
- metrics:
    http_response_time_seconds:
      type: Histogram
      description: "length of each log line"
      source: response_time
      config:
        buckets: [0.001, 0.0025, 0.005, 0.010, 0.025, 0.050]
```

設定済みstage

- デフォルトで設定されている便利な設定済みstage
  - docker
  - cri
- 上記はコンテナランタイムから吐き出されるログから
  - タイムスタンプ
  - メッセージの抽出
  - 出力がstdout, stderrかをstreamラベルとして付与　等
- 細かなstageの設定をしなくても、これだけでとりあえず使える

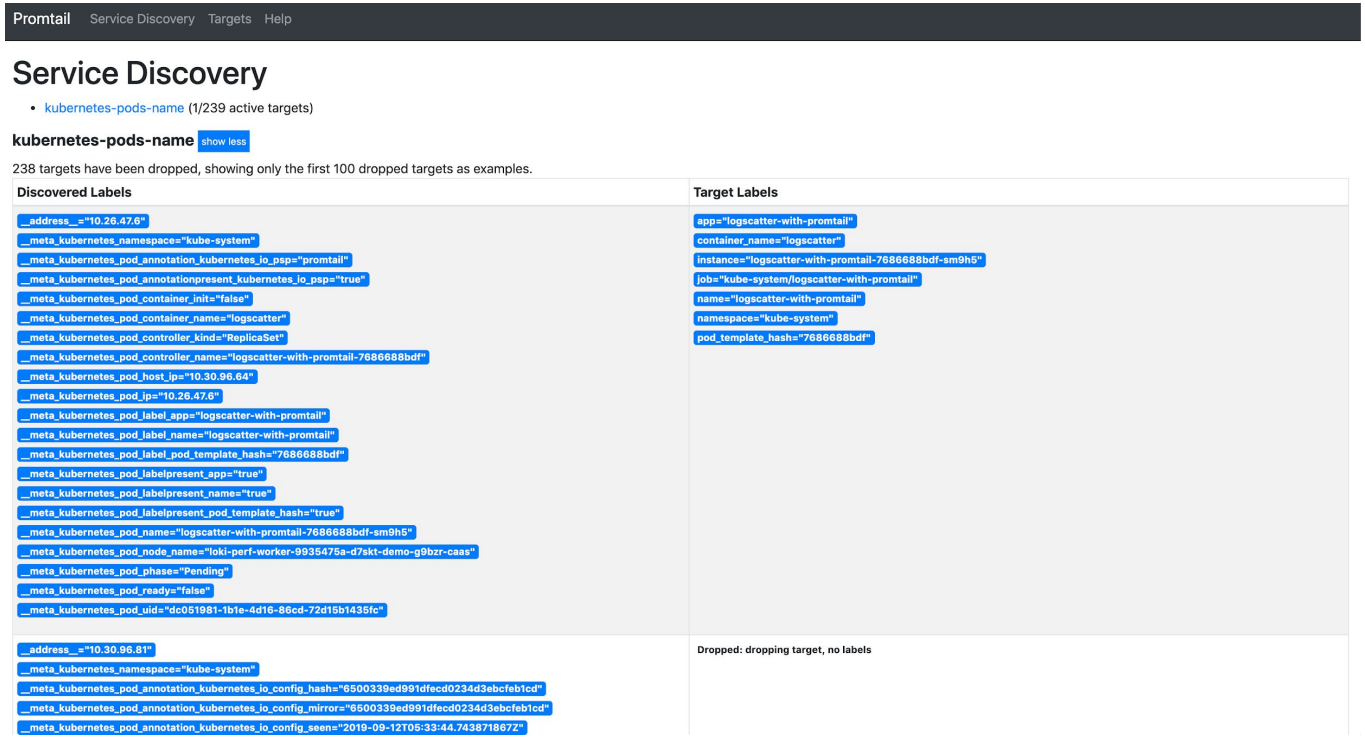
```
pipeline_stages:
- docker: {}
```

Promtailの設定確認(1/3)

- 設定確認のためのwebserverを起動

```
- client:
  ...
  positions:
    filename: /run/promtail/positions.yaml
  server:
    http_listen_port: 3101
  target_config:
    sync_period: 10s
scrape_configs:
-  job_name: job1
  static_configs:
  ...
- job_name: job2
  kubernetes_sd_configs:
  ...
  relabel_configs:
  ...
  pipeline_stages:
  ...
```

Promtailの設定確認(2/3)



Promtailの設定確認(3/3)

Promtailのより細かな情報

- 以下のURLを参照
  - https://grafana.com/blog/2019/07/25/lokis-path-to-ga-adding-structure-to-unstructured-logs/
  - https://github.com/grafana/loki/blob/master/docs/logentry/processing-log-lines.md

まとめ

- lokiのアーキテクチャ
  - Distributor
  - Ingester
  - Querier
  - Chunk Store
  - Table Manager

- Promtailの役割と設定について紹介
  - labelをつけてログをlokiへ送信
  - Prometheusの設定と似ているが大きな違いとして pipeline_stages