Query frontend example

- Before you begin
- Use case
  - Parallelization
- Kubernetes Deployment
  - ConfigMap
  - Frontend Service
  - Frontend Deployment
  - Grafana
  - gRPC mode (pull model)

Before you begin

This is a general-purpose example. It contains placeholder variables in the format `<variable_name>`. Replace them with values for your environment.

Use case

Many users start by running Grafana Loki as a single binary to simplify deployment. As usage grows and the deployment becomes more production-ready, performance bottlenecks can appear on the read path. Queries that ran quickly on small data sets may slow down or fail on larger ones. Adding more queriers can help in some cases. However, when a single query is too large for one querier to handle, you need the query frontend.

Parallelization

One of the most important functions of the query frontend is the ability to split larger queries into smaller ones, execute them in parallel, and combine the results. The `querier.split-queries-by-`

`interval` flag (or the YAML config `limits_config.split_queries_by_interval`) controls how the frontend splits queries. For example, with this value set to `1h`, the frontend splits a 24-hour query into 24 one-hour queries, distributes them to the queriers, and collects the results. This approach has two benefits in production environments: it lets you run larger queries through aggregation, and it distributes work evenly across queriers.

Kubernetes Deployment

ConfigMap

Use this ConfigMap to enable query parallelization and caching with the query-frontend component.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: loki-frontend
  namespace: <namespace>
data:
  config.yaml: |
    # Disable the requirement that every request to Loki has a
    # X-Scope-OrgID header. `fake` will be substituted in instead.
    auth_enabled: false

    server:
      http_listen_port: 3100

    query_range:
      # make queries more cache-able by aligning them with their step intervals
      align_queries_with_step: true
      max_retries: 5
      cache_results: true

      results_cache:
        cache:
          # Use the embedded cache
          embedded_cache:
            enabled: true
            max_size_mb: 100
```

```
            ttl: 24h

    limits_config:
      max_cache_freshness_per_query: '10m'
      # parallelize queries in 15min intervals
      split_queries_by_interval: 15m

    frontend:
      log_queries_longer_than: 5s
      downstream_url: http://querier.<namespace>.svc.cluster.local:3100
      compress_responses: true
```

Frontend Service

```
apiVersion: v1
kind: Service
metadata:
  labels:
    name: query-frontend
  name: query-frontend
  namespace: <namespace>
spec:
  ports:
    - name: query-frontend-http
      port: 3100
      protocol: TCP
      targetPort: 3100
  selector:
    name: query-frontend
  sessionAffinity: None
  clusterIP: None
  publishNotReadyAddresses: true
```

Frontend Deployment

```
apiVersion: apps/v1
kind: Deployment
```

```
metadata:
  labels:
    name: query-frontend
  name: query-frontend
  namespace: <namespace>
spec:
  minReadySeconds: 10
  replicas: 2
  selector:
    matchLabels:
      name: query-frontend
  template:
    metadata:
      labels:
        name: query-frontend
    spec:
      containers:
        - args:
            - -config.file=/etc/loki/config.yaml
            - -log.level=debug
            - -target=query-frontend
          image: grafana/loki:<VERSION>
          imagePullPolicy: Always
          name: query-frontend
          ports:
            - containerPort: 3100
              name: http
              protocol: TCP
          resources:
            limits:
              memory: 1200Mi
            requests:
              cpu: '2'
              memory: 600Mi
          volumeMounts:
            - mountPath: /etc/loki
              name: loki-frontend
      restartPolicy: Always
      terminationGracePeriodSeconds: 30
```

```
      volumes:
        - configMap:
            defaultMode: 420
            name: loki-frontend
          name: loki-frontend
```

Grafana

After you deploy these resources, point your Grafana data source to the new frontend service. The service is available within the cluster at `http://query-frontend.<namespace>.svc.cluster.local:3100`.

gRPC mode (pull model)

The query frontend operates in one of two modes:

- **Push mode:** Specify `--frontend.downstream-url` or its YAML equivalent `frontend.downstream_url`. The frontend proxies requests over HTTP to the specified URL.
- **Pull mode:** If you do not set `--frontend.downstream-url` or its YAML equivalent `frontend.downstream_url`, the query frontend operates as a pull service. The frontend creates per-tenant queues. Downstream queriers pull queries from these queues using gRPC. To use pull mode, configure queriers with `-querier.frontend-address` or its YAML equivalent `frontend_worker.frontend_address`.
  Set `clusterIP: None` on the query frontend pull service. This setting causes DNS to resolve to individual query frontend pod IP addresses instead of the service IP.
  Set `publishNotReadyAddresses: true` on the query frontend pull service. This setting prevents a race condition where a querier needs the query frontend address before the frontend is ready.

□□□ #2
□ □□□ □□□ 2026-10-31 13:44:32 UTC
□ □□□ □□□ 2026-10-31 13:44:33 UTC