Leased Logs:

Saving Money By Saving Fewer Logs

Carson Anderson

@carsonoid

Principal Engineer, Weave

@carsonoid@kind.social

getweave.com

- The problem with logs
- Logs @ Weave
  - Existing architecture
  - How we implemented leased logging
- GCP Primer
  - Cloud logging
  - Firestore
- Sample Code & CLI

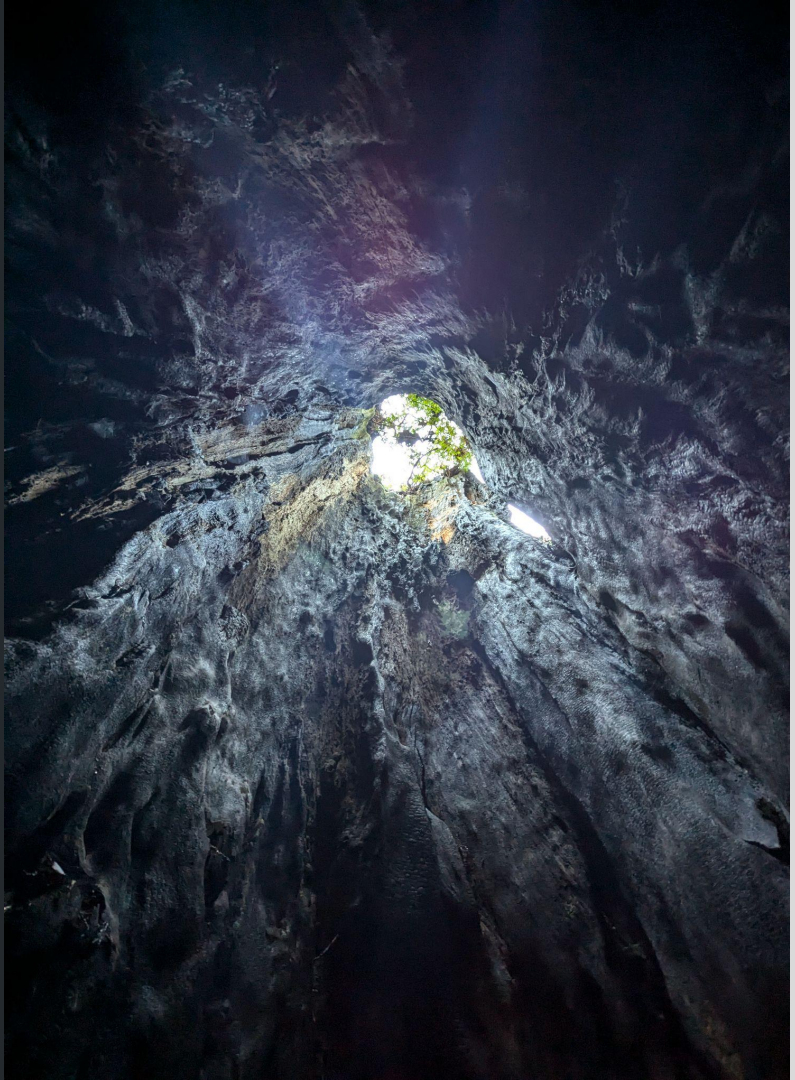

The problem with logs

The Problem: Scale

- Cost increases linearly
- Uses egress bandwidth

Solutions?

- Pay all the money
- Police log usage in code
- Don't log at all



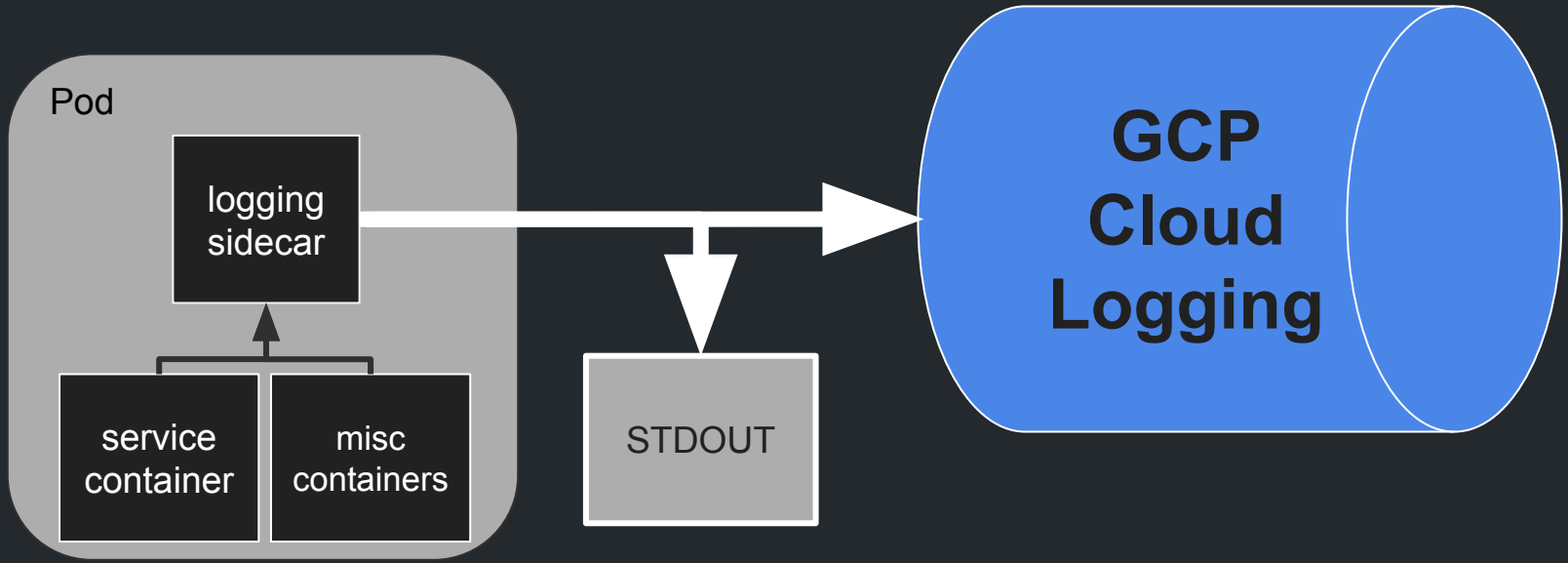
Leased Logging

Send Logs Conditionally

Only ship logs if:

- **It is the first 15 minutes of an app's lifespan**
- **OR "log lease" is active**
- **OR level is "ERROR"**
- **OR level is "ALWAYS"**

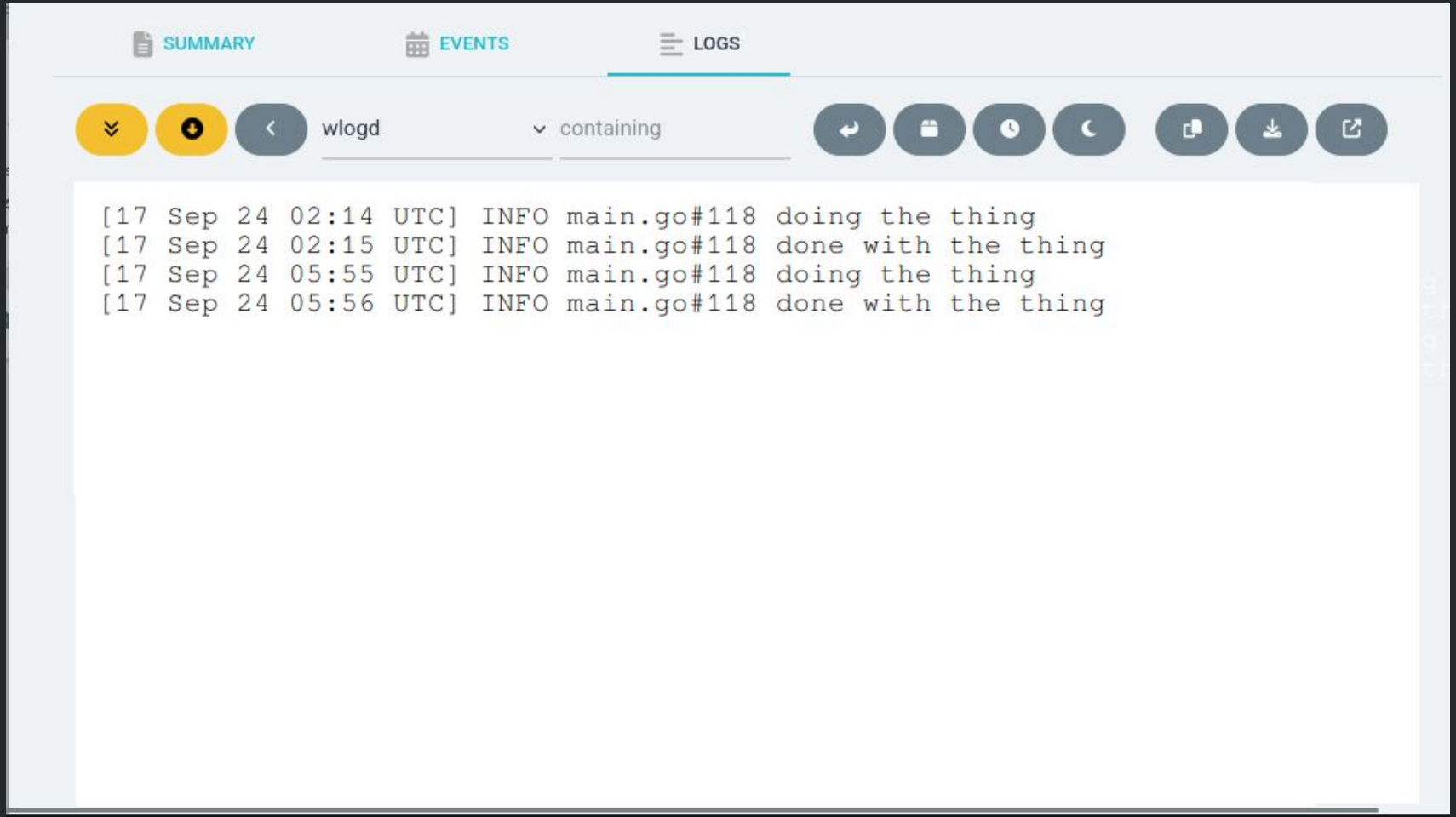

Logging @ Weave

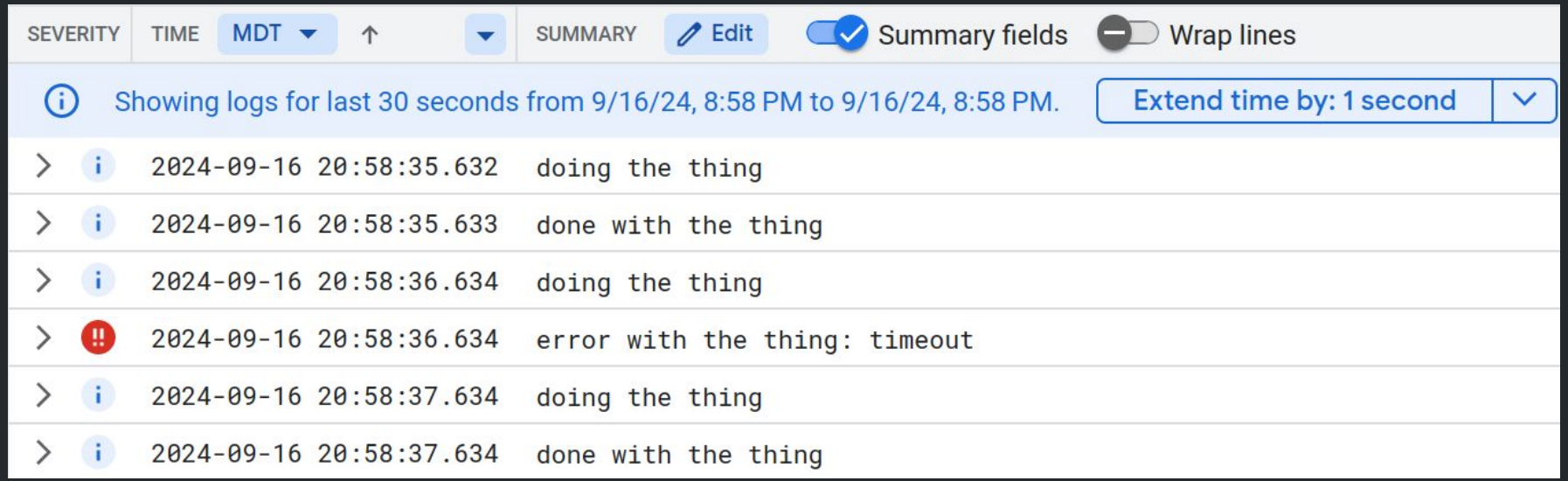

Pod
logging sidecar
service container
misc containers
STDOUT
GCP Cloud Logging

```go
func DoThing(ctx context.Context) {
    wlog.Info(ctx, "doing the thing")
    err := do()
    if err != nil {
        wlog.Error(ctx, err, "failed to do thing")
        return
    }
    wlog.InfoC(ctx, "done with the thing")
}
```

$ bart logs

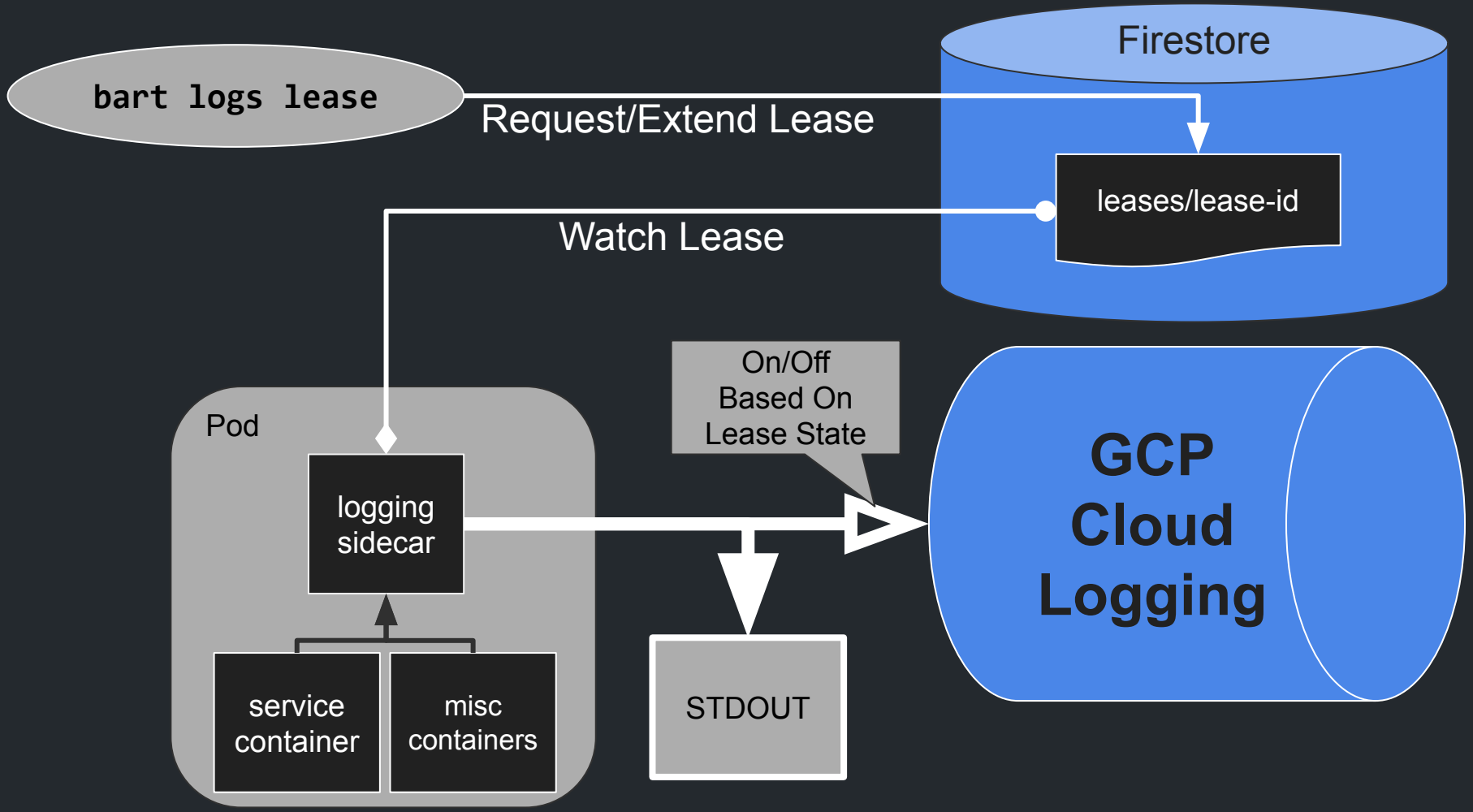
```
[pod-1 wlogd] [17 Sep 24 02:14 UTC] INFO main.go#118 doing the thing
[pod-1 wlogd] [17 Sep 24 02:15 UTC] INFO main.go#118 done with the thing
[pod-2 wlogd] [17 Sep 24 03:11 UTC] INFO main.go#118 doing the thing
[pod-2 wlogd] [17 Sep 24 03:11 UTC] ERROR main.go#118 error with the thing: timeout
[pod-1 wlogd] [17 Sep 24 05:55 UTC] INFO main.go#118 doing the thing
[pod-1 wlogd] [17 Sep 24 05:56 UTC] INFO main.go#118 done with the thing
```


SUMMARY
EVENTS
LOGS
wlogd
containing
[17 Sep 24 02:14 UTC] INFO main.go#118 doing the thing
[17 Sep 24 02:15 UTC] INFO main.go#118 done with the thing
[17 Sep 24 05:55 UTC] INFO main.go#118 doing the thing
[17 Sep 24 05:56 UTC] INFO main.go#118 done with the thing


SEVERITY
TIME
MDT
SUMMARY
Edit
Summary fields
Wrap lines
Showing logs for last 30 seconds from 9/16/24, 8:58 PM to 9/16/24, 8:58 PM.
Extend time by: 1 second
2024-09-16 20:58:35.632 doing the thing
2024-09-16 20:58:35.633 done with the thing
2024-09-16 20:58:36.634 doing the thing
2024-09-16 20:58:36.634 error with the thing: timeout
2024-09-16 20:58:37.634 doing the thing
2024-09-16 20:58:37.634 done with the thing

- Ephemeral
  - `bart logs` (via `kubetail`)
  - Argo-CD Web UI
- Durable
  - GCP Logging Console


Leased Logging @ Weave


bart logs lease
Request/Extend Lease
Firestore
leases/lease-id
Watch Lease
On/Off
Based On
Lease State
Pod
logging
sidecar
service
container
misc
containers
STDOUT
GCP
Cloud
Logging

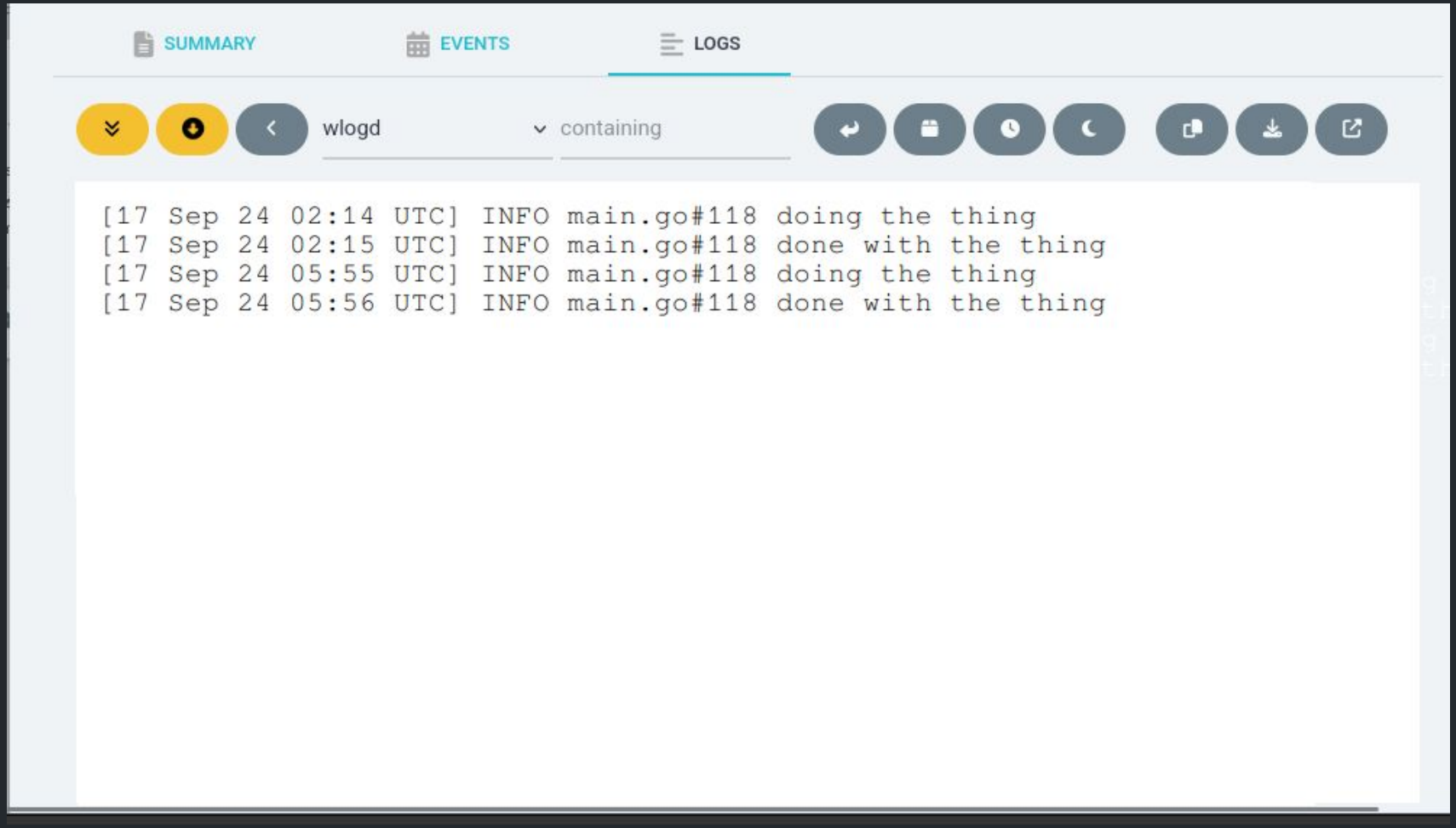
Code: Unchanged

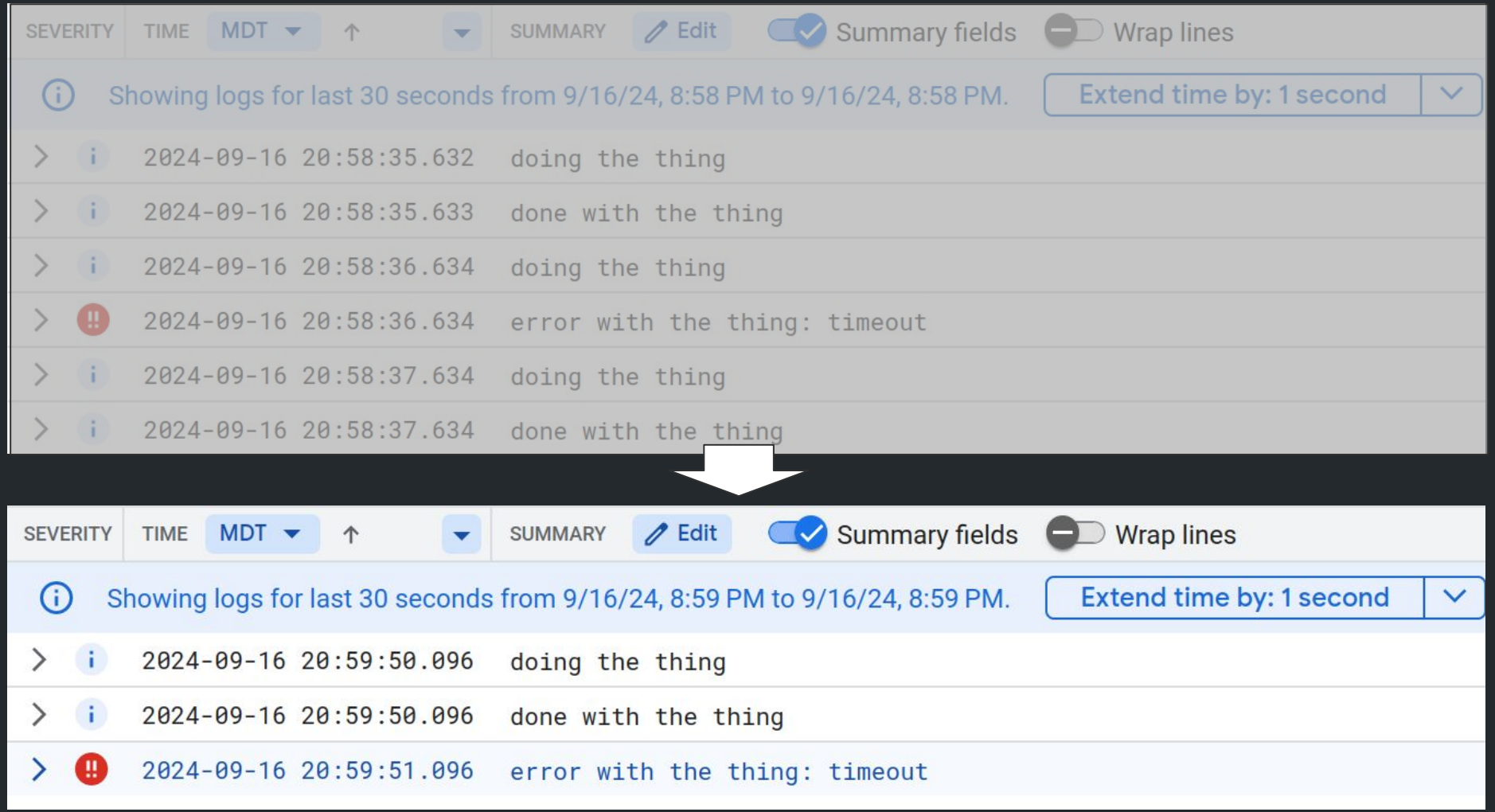
```
func DoThing(ctx context.Context) {
    wlog.Info(ctx, "doing the thing")
    err := do()
    if err != nil {
        wlog.Error(ctx, err, "failed to do thing")
        return
    }
    wlog.InfoC(ctx, "done with the thing")
}
```

bart (kubetail) logs: Unchanged

```
$ bart logs
[pod-1 wlogd] [17 Sep 24 02:14 UTC] INFO main.go#118 doing the thing
[pod-1 wlogd] [17 Sep 24 02:15 UTC] INFO main.go#118 done with the thing
[pod-2 wlogd] [17 Sep 24 03:11 UTC] INFO main.go#118 doing the thing
[pod-2 wlogd] [17 Sep 24 03:11 UTC] ERROR main.go#118 error with the thing: timeout
[pod-1 wlogd] [17 Sep 24 05:55 UTC] INFO main.go#118 doing the thing
[pod-1 wlogd] [17 Sep 24 05:56 UTC] INFO main.go#118 done with the thing
```

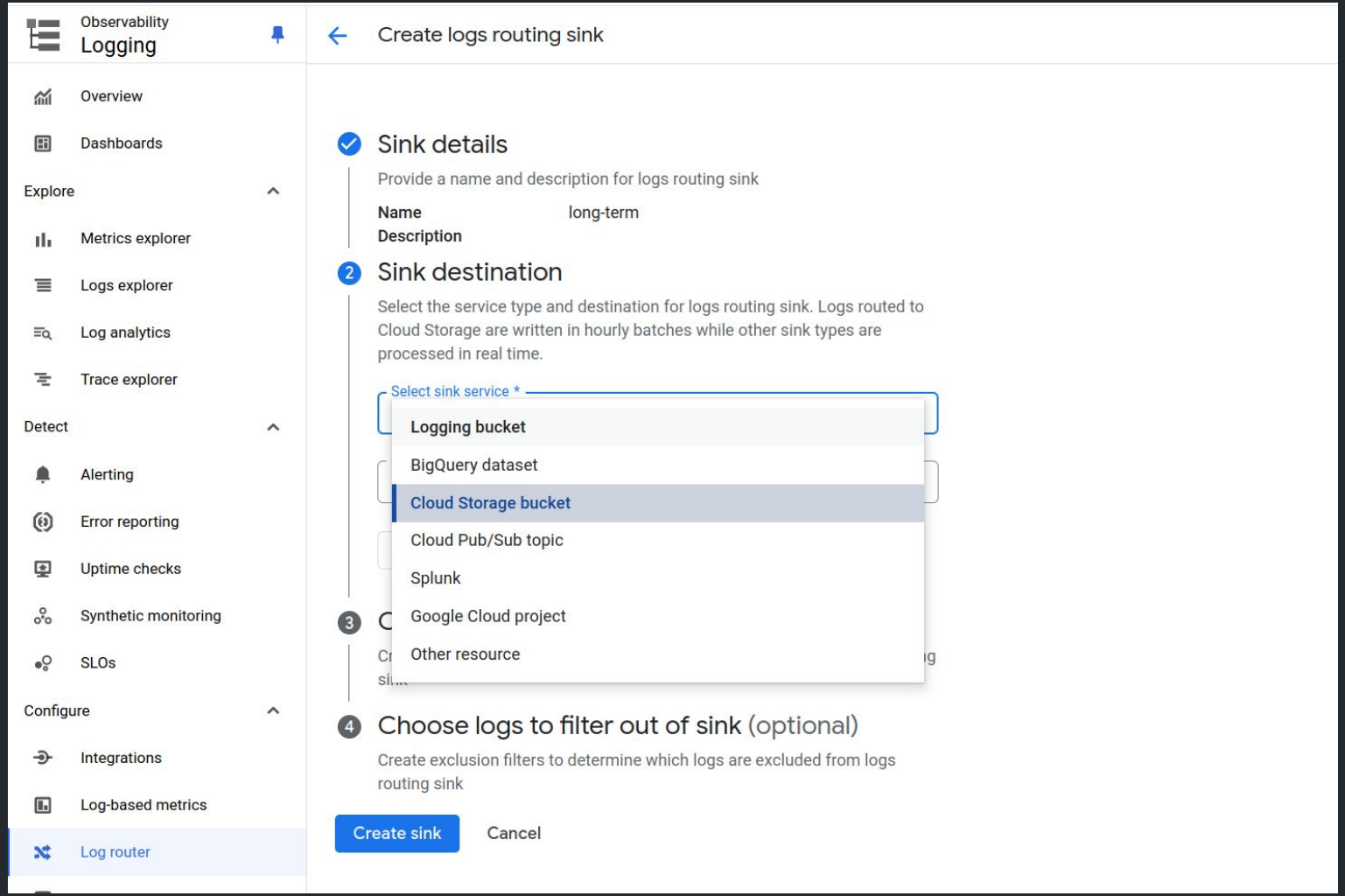
ArgoCD Logs: Unchanged




SEVERITY
TIME
MDT
SUMMARY
Edit
Summary fields
Wrap lines
Showing logs for last 30 seconds from 9/16/24, 8:58 PM to 9/16/24, 8:58 PM.
Extend time by: 1 second
2024-09-16 20:58:35.632 doing the thing
2024-09-16 20:58:35.633 done with the thing
2024-09-16 20:58:36.634 doing the thing
2024-09-16 20:58:36.634 error with the thing: timeout
2024-09-16 20:58:37.634 doing the thing
2024-09-16 20:58:37.634 done with the thing
SEVERITY
TIME
MDT
SUMMARY
Edit
Summary fields
Wrap lines
Showing logs for last 30 seconds from 9/16/24, 8:59 PM to 9/16/24, 8:59 PM.
Extend time by: 1 second
2024-09-16 20:59:50.096 doing the thing
2024-09-16 20:59:50.096 done with the thing
2024-09-16 20:59:51.096 error with the thing: timeout

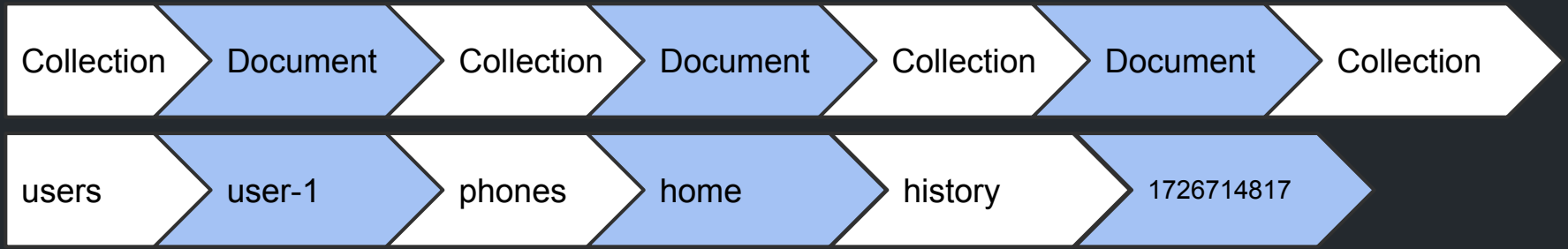

GCP Primer:
Cloud Logging

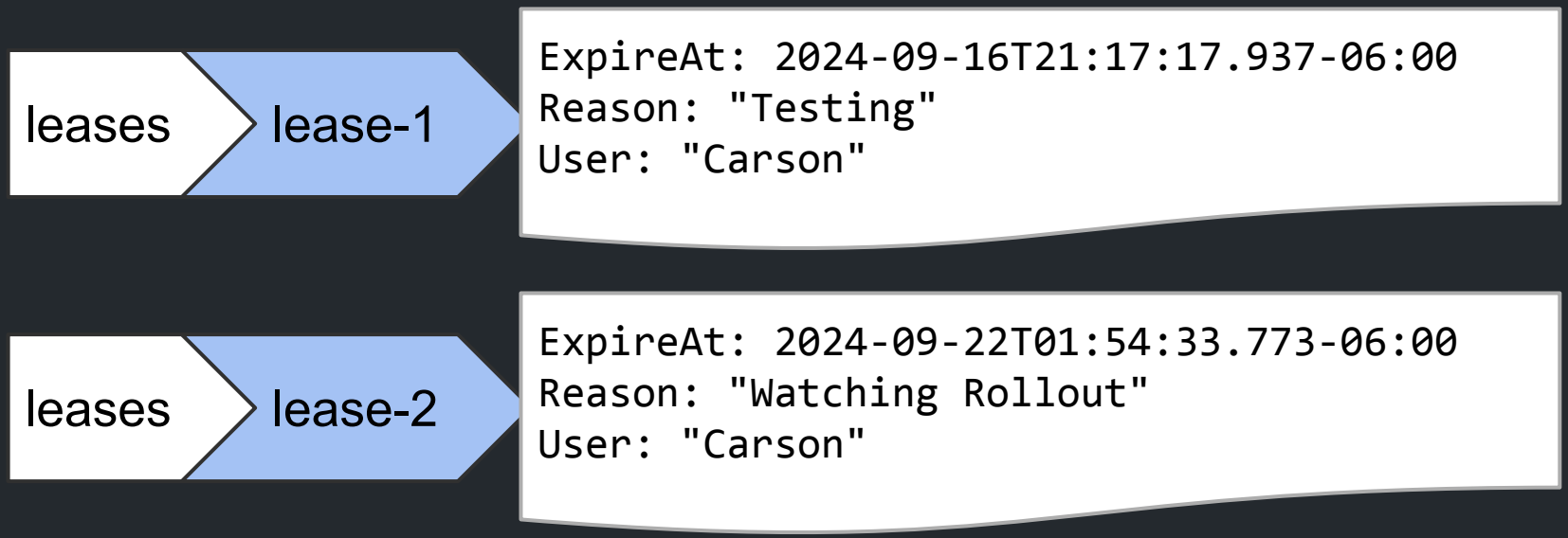
- Serverless
- Generous Free Tier
- Supports Structured Logs & Rich Queries
- **Log Routers** can change where logs are sent
  - May be an easier way to reduce spend in some cases
- REST API
- SDKs for multiple languages
  - Usable in Go as a simple "io.Writer"


Observability
Logging
Overview
Dashboards
Explore
Metrics explorer
Logs explorer
Log analytics
Trace explorer
Detect
Alerting
Error reporting
Uptime checks
Synthetic monitoring
SLOs
Configure
Integrations
Log-based metrics
Log router
Create logs routing sink
Sink details
Provide a name and description for logs routing sink
Name
long-term
Description
2
Sink destination
Select the service type and destination for logs routing sink. Logs routed to Cloud Storage are written in hourly batches while other sink types are processed in real time.
Select sink service *
Logging bucket
BigQuery dataset
Cloud Storage bucket
Cloud Pub/Sub topic
Splunk
Google Cloud project
Other resource
3
4
Choose logs to filter out of sink (optional)
Create exclusion filters to determine which logs are excluded from logs routing sink
Create sink
Cancel

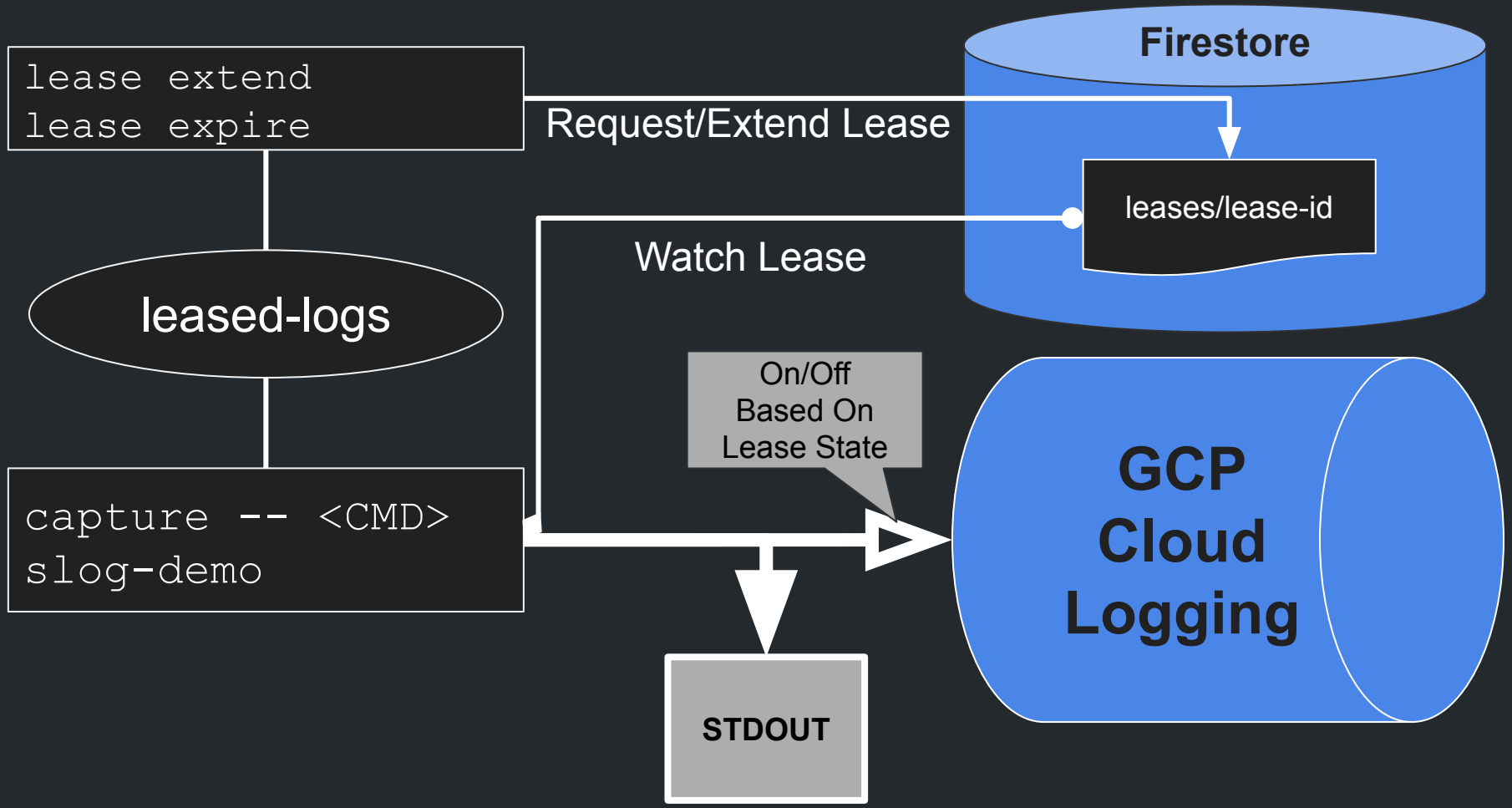

GCP Primer:
Firestore

- NoSQL Database
- Serverless
- Generous Free Tier
- Requires Structured data (not json)
  - Automatic with Golang SDK
- **Live Watch** 👁️
- Supports automatic cleanup (TTL Job)
- Uses a collection and document design pattern




leases
lease-1
ExpireAt: 2024-09-16T21:17:17.937-06:00
Reason: "Testing"
User: "Carson"
leases
lease-2
ExpireAt: 2024-09-22T01:54:33.773-06:00
Reason: "Watching Rollout"
User: "Carson"


Sample CLI
diagram • code tour • demo


lease extend
lease expire
Request/Extend Lease
Firestore
leases/lease-id
Watch Lease
leased-logs
On/Off
Based On
Lease State
capture -- <CMD>
slog-demo
GCP
Cloud
Logging
STDOUT

```go
// create a GCP cloud logging client using the project ID and default credentials
logClient, err := logging.NewClient(ctx, cli.ProjectID)
kctx.FatalIfErrorf(err, "Failed to create logging client")
defer logClient.Close()

// create a Firestore client using the project ID and default credentials
fsClient, err := firestore.NewClient(ctx, cli.ProjectID)
kctx.FatalIfErrorf(err, "Failed to create firestore client")

// make a document reference to the lease document
// this does not fetch the doc but can be used to interact with it later
docRef := fsClient.Collection("leases").Doc(cli.LeaseID)

// run sub-commands passing the firestore client, log client, and docRef for use
err = kctx.Run(fsClient, logClient, docRef)
```

```go
// Document represents a Firestore document representing a lease.
type Document struct {
    ExpireAt time.Time
    User     string
    Reason   string
}

// Manager handles a lease document and manages the lease state.
type Manager struct {
    logger          *logging.Logger
    guaranteedUntil time.Time

    enabled     atomic.Bool
    expireTimer *time.Timer
}
```

```go
// watchLease watches a lease document for changes and updates the lease state.
func (m *Manager) watchLease(ctx context.Context, docRef *firestore.DocumentRef) error
{
    fmt.Fprintln(os.Stderr, "===  WATCH LEASE", docRef.Path)
    iter := docRef.Snapshots(ctx)
    defer iter.Stop()
    for {
        snapshot, err := iter.Next()
        switch {
        case err == io.EOF,
            err == context.DeadlineExceeded,
            err == context.Canceled:
            return nil
        case err != nil:
            fmt.Fprintln(os.Stderr, "Failed to get snapshot:", err)
            return err
        }
```

```
// watchLease CONTINUED
        // if the snapshot does not yet exist, espire after the guaranteedUntil time
        // for leases that are deleted after the guaranteedUntil time, this will disable the lease
immediately
        if !snapshot.Exists() {
            m.expireAfter(m.guaranteedUntil)
            continue
        }

        var lease Document
        if err := snapshot.DataTo(&lease); err != nil {
            fmt.Fprintln(os.Stderr, "Failed to parse lease:", err)
            continue
        }

        m.expireAfter(lease.ExpireAt)
        if lease.ExpireAt.After(m.guaranteedUntil) {
            fmt.Fprintf(os.Stderr, "=== LEASE EXTENDED, expires in %s | user=%q reason=%q\n",
time.Until(lease.ExpireAt).Round(time.Millisecond*100), lease.User, lease.Reason)
        }
    }
}
```

```
// expireAfter sets a new lease expiration time, resetting the lease timer
//   - respects the guaranteedUntil time, even if the lease is shorter
func (m *Manager) expireAfter(expire time.Time) {
   // ensure guaranteedUntil is always respected, even if the lease is
shorter
   if expire.Before(m.guaranteedUntil) {
       expire = m.guaranteedUntil
   }

   // cancel the previous timer if it was unfired
   if m.expireTimer != nil && !m.expireTimer.Stop() {
       select {
       case <-m.expireTimer.C:
       default:
       }
   }
```

```
// expireAfter continued

    // already expired, disable immediately
    if expire.Before(time.Now().UTC()) {
        fmt.Fprintln(os.Stderr, "=== LEASE EXPIRED")
        m.disable()
        return
    }

    // enable and set a new timer
    m.enable()

    m.expireTimer = time.AfterFunc(time.Until(expire), func() {
        fmt.Fprintln(os.Stderr, "=== LEASE EXPIRED")
        m.disable()
    })
}
```

DEMO

Thank you!

https://github.com/carsonoid/talk-leased-logs

https://engineering.getweave.com

Carson Anderson

Principal Engineer, Weave

@carsonoid

@carsonoid@kind.social