Article

Logging, Metrics and Tracing

Considered the three pillars of observability, logging, metrics and tracing provide different ways of viewing how your application is working.

Overview

Apple has developed packages for each of the observability systems (swift-log, swift-metrics, swift-distributed-tracing). They provide a consistent API while not defining how the backend is implemented. With these it is possible to add observability to your own libraries without commiting to a certain implementation of each system.

Hummingbird has middleware for each of these systems. As these are provided as middleware you can add these to your application as and when you need them.

Logging

Logs provides a record of discrete events over time. Each event has a timestamp, description and an array of metadata. Hummingbird automatically does some logging of events. You can control the fidelity of your logging by providing your own `Logger` when creating your `Application` eg

```swift
var logger = Logger(label: "MyLogger")
logger.logLevel = .debug
let application = Application(
    router: router,
    logger: logger
)
```

If you want a record of every request to the server you can add the `LogRequestsMiddleware` middleware. You can control at what `logLevel` the request logging will occur and whether it includes information about each requests headers. eg

```
let router = Router()
router.middlewares.add(LogRequestsMiddleware(.debug, includeHeaders: false))
```

If you would like to add your own logging, or implement your own logging backend you can find out more here. A complete list of logging implementations can be found here.

Metrics

Metrics provides an overview of how your application is working over time. It allows you to create visualisations of the state of your application.

The middleware `MetricsMiddleware` will record how many requests are being made to each route, how long they took and how many failed. To add recording of these metrics to your Hummingbird application you need to add this middleware and bootstrap your chosen metrics backend. Below is an example setting up recording metrics with Prometheus, using the package SwiftPrometheus.

```
import Metrics
import Prometheus

// Bootstrap Prometheus
let prometheus = PrometheusClient()
MetricsSystem.bootstrap(PrometheusMetricsFactory(client: prometheus))

// Add metrics middleware to router
router.middlewares.add(MetricsMiddleware())
```

If you would like to record your own metrics, or implement your own metrics backed you can find out more here. A list of metrics backend implementations can be found here.

Tracing

Tracing is used to understand how data flows through an application’s various services.

The middleware `TracingMiddleware` will record spans for each request made to your application and attach the relevant metadata about request and responses. To add tracing to your Hummingbird application you need to add this middleware and bootstrap your chosen tracing backend. Below is an example setting up tracing using the Open Telemetry package swift-otel.

```
import OpenTelemetry
import Tracing

// Bootstrap Open Telemetry
let otel = OTel(serviceName: "example", eventLoopGroup: .singleton)
try otel.start().wait()
InstrumentationSystem.bootstrap(otel.tracer())

// Add tracing middleware
router.middlewares.add(TracingMiddleware(recordingHeaders: ["content-type",
```

If you would like to find out more about tracing, or implement your own tracing backend you can find out more here.

See Also

Related Documentation

`struct LogRequestsMiddleware`

Middleware outputting to log for every call to server.

`struct MetricsMiddleware`

Middleware recording metrics for each request

`struct TracingMiddleware`

Middleware creating Distributed Tracing spans for each request.

Hummingbird Server

Router

The router directs requests to their handlers based on the contents of their path.

Request Decoding

Decoding of Requests with JSON content and other formats.

Response Encoding

Writing Responses using JSON and other formats.

Request Contexts

Controlling contextual data provided to middleware and route handlers

Middleware

Processing requests and responses outside of request handlers.

Error Handling

How to build errors for the server to return.

Result Builder Router

Building your router using a result builder.

Server protocol

Support for TLS and HTTP2 upgrades

Service Lifecycle

Integration with Swift Service Lifecycle

Testing

Using the HummingbirdTesting framework to test your application

Persistent data

How to persist data between requests to your server.

Migrating to Hummingbird v2

Migration guide for converting Hummingbird v1 applications to Hummingbird v2