Observability

Bacalhau supports the three main 'pillars' of observability - logging, metrics, and tracing. Bacalhau uses the OpenTelemetry Go SDK for metrics and tracing, which can be configured using the standard environment variables. Exporting metrics and traces can be as simple as setting the `OTEL_EXPORTER_OTLP_PROTOCOL` and `OTEL_EXPORTER_OTLP_ENDPOINT` environment variables. Custom code is used for logging as the OpenTelemetry Go SDK currently doesn't support logging.

Logging

Logging in Bacalhau outputs in human-friendly format to stderr at `INFO` level by default, but this can be changed by two environment variables:

- `LOG_LEVEL` - Can be one of `trace`, `debug`, `error`, `warn` or `fatal` to output more or fewer logging messages as required
- `LOG_TYPE` - Can be one of the following values:
  -

- `default` - output logs to stderr in a human-friendly format
- `json` - log messages outputted to stdout in JSON format
- `combined` - log JSON formatted messages to stdout and human-friendly format to stderr

Log statements should include the relevant trace, span and job ID so it can be tracked back to the work being performed.

Metrics

Bacalhau produces a number of different metrics including those around the libp2p resource manager (`rcmgr`), performance of the requester HTTP API and the number of jobs accepted/completed/received.

Tracing

Traces are produced for all major pieces of work when processing a job, although the naming of some spans is still being worked on. You can find relevant traces covering working on a job by searching for the `jobid` attribute.

Viewing

The metrics and traces can easily be forwarded to a variety of different services as we use OpenTelemetry, such as Honeycomb or Datadog.

To view the data locally, or simply to not use a SaaS offering, you can start up Jaeger and Prometheus placing these three files into a directory then running `docker compose start` while running Bacalhau with the `OTEL_EXPORTER_OTLP_PROTOCOL=grpc` and `OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4317` environment variables.

```yaml
version: "2"
services:

  jaeger-all-in-one:
    image: "jaegertracing/all-in-one:1.42"
    restart: "always"
    ports:
      - "16686:16686" # Jaeger UI
      - "14250:14250" # Jaeger gRPC endpoint

  otel-collector:
    image: "otel/opentelemetry-collector:0.70.0"
    restart: "always"
    command: ["--config=/etc/otel-collector-config.y
    volumes:
      - "./otel-collector-config.yaml:/etc/otel-coll
    ports:
      - "8888:8888"   # Prometheus metrics exposed b
      - "8889:8889"   # Prometheus exporter metrics
      - "13133:13133" # health_check extension
      - "4317:4317"   # OTLP gRPC receiver
    depends_on:
      - "jaeger-all-in-one"
      - "prometheus"

  prometheus:
    container_name: "prometheus"
    image: "prom/prometheus:v2.42.0"
    restart: "always"
    volumes:
      - "./prometheus.yaml:/etc/prometheus/prometheu
    ports:
      - "9090:9090" # Prometheus UI
```

```yaml
receivers:
  otlp:
    protocols:
      grpc:

exporters:
  prometheus:
    endpoint: "0.0.0.0:8889"

  jaeger:
    endpoint: jaeger-all-in-one:14250
    tls:
      insecure: true

processors:
  batch:

extensions:
  health_check:

service:
  extensions: [health_check]
  pipelines:
    traces:
      receivers: [otlp]
      processors: [batch]
      exporters: [jaeger]
    metrics:
      receivers: [otlp]
      processors: [batch]
      exporters: [prometheus]
```

```
scrape_configs:
  - job_name: 'otel-collector'
    scrape_interval: 10s
    static_configs:
      - targets: ['otel-collector:8889']
      - targets: ['otel-collector:8888']
```

Previous
Configuring Transport Level Security

Next
Limits and Timeouts