Getting familiar with the OpenTelemetry Collector

Alex Boten, Senior Staff Software Engineer @ Lightstep - May 2022

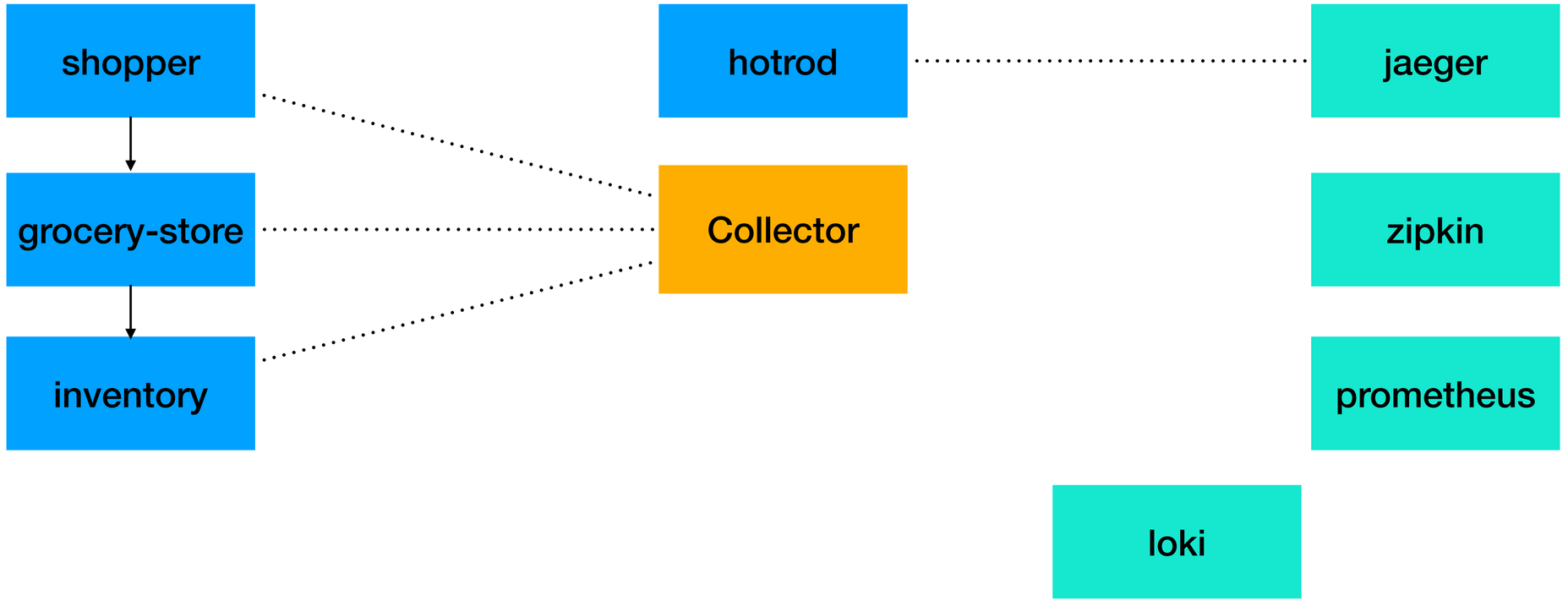
Agenda

- Setup
- Using the collector
  - Configure receivers and exporters
  - Process data
- Production-izing
  - Telemetry
  - Health checks
  - Sampling
  - Build custom collector

Launch demo application

- git clone https://github.com/codeboten/o11yfest
- docker compose up

Demo application



Consolidate telemetry

Configure exporters: logging

```
exporters:
  logging:
    loglevel: debug
```

```
$ docker logs -f opentelemetry-collector
```

Configure exporters: jaeger

```
exporters:
  jaeger:
    endpoint: jaeger:14250
    tls:
      insecure: true
service:
  pipelines:
    traces:
      exporters: [logging, jaeger]
```

http://localhost:16686

Configure exporters: zipkin

```
exporters:
  zipkin:
    endpoint: "http://zipkin:9411/api/v2/spans"
    tls:
      insecure: true
service:
  pipelines:
    traces:
      exporters: [logging, jaeger, zipkin]
```

http://localhost:9411

Configure exporters: prometheus

```
exporters:
  prometheus:
    endpoint: ":8889"
    namespace: o11yfest
    send_timestamps: true
    resource_to_telemetry_conversion:
      enabled: true
service:
  pipelines:
    metrics:
      exporters: [logging, prometheus]
```

http://localhost:9090

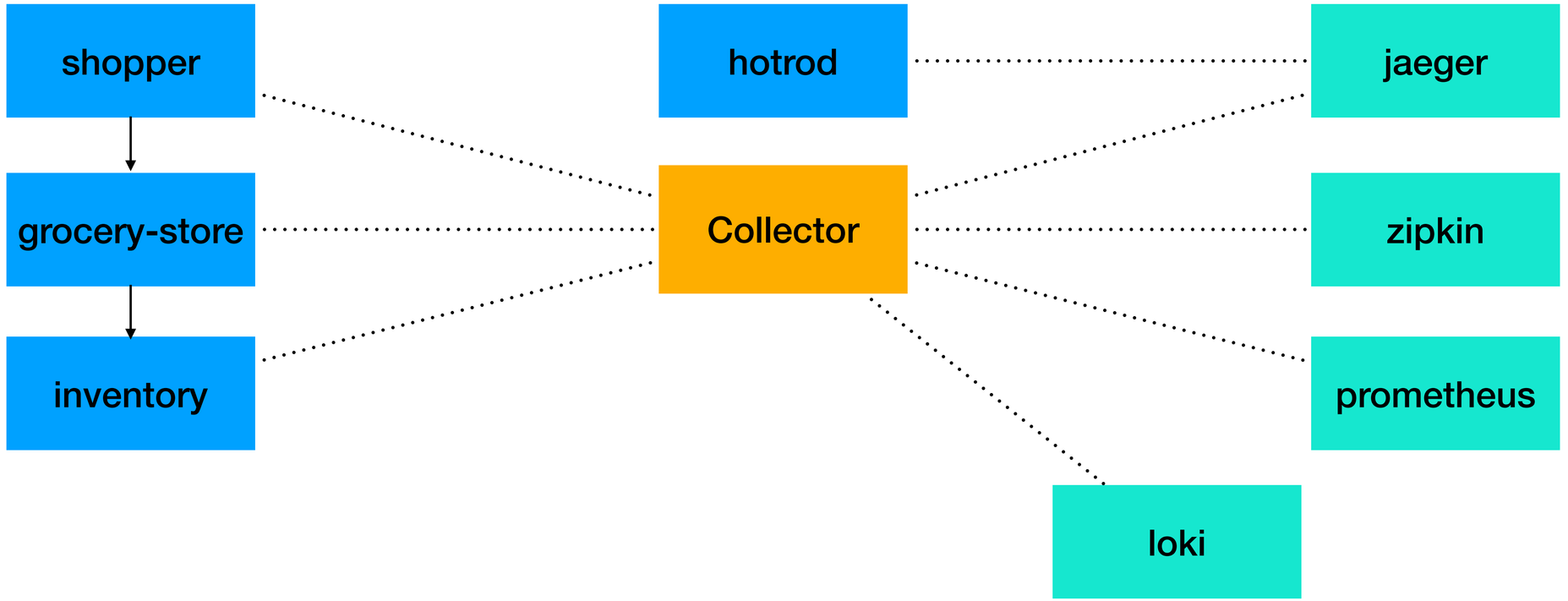
Configure exporters: loki

```
exporters:
  loki:
    endpoint: http://loki:3100/loki/api/v1/push
    labels:
      resource:
        service.name: "job"

service:
   logs:
     exporters: [logging, loki]
```

http://localhost:3000/explore

Demo application



Configure receivers: jaeger

```
receivers:
  jaeger:
    protocols:
      grpc:
service:
  pipelines:
    traces:
      receivers: [otlp, jaeger]
```

Configure receivers: jaeger

```
hotrod:
  image: jaegertracing/example-hotrod:latest
  container_name: hotrod
  environment:
    - JAEGER_AGENT_HOST=opentelemetry-collector
    - JAEGER_AGENT_PORT=6831
```

http://localhost:9411

Configure receivers: jaeger

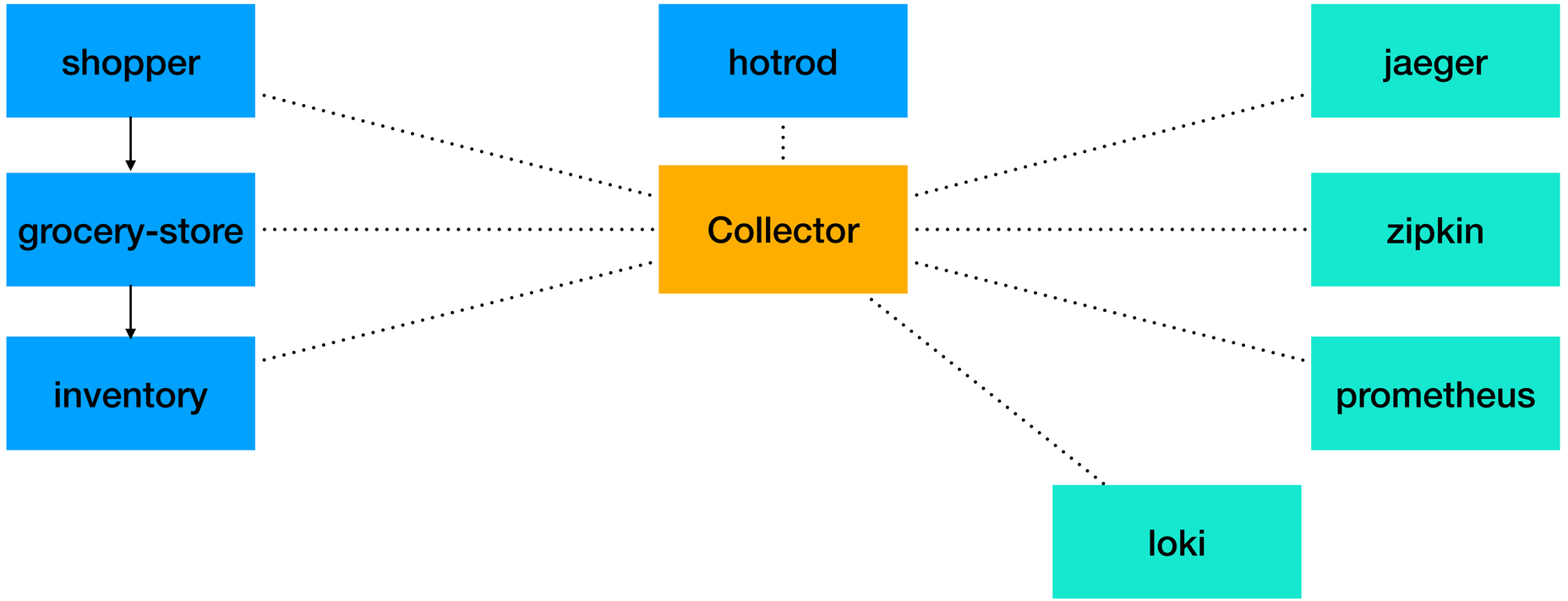
```
hotrod:
  image: jaegertracing/example-hotrod:latest
  container_name: hotrod
  environment:
    - JAEGER_AGENT_HOST=opentelemetry-collector
    - JAEGER_AGENT_PORT=6831
  command:
    - all
    - --metrics=prometheus
```

Configure receivers: prometheus

```
receivers:
  prometheus:
    config:
      scrape_configs:
        - job_name: 'hotrod'
          scrape_interval: 5s
          static_configs:
            - targets: ['hotrod:8083']
service:
  pipelines:
    metrics:
      receivers: [otlp, prometheus]
```

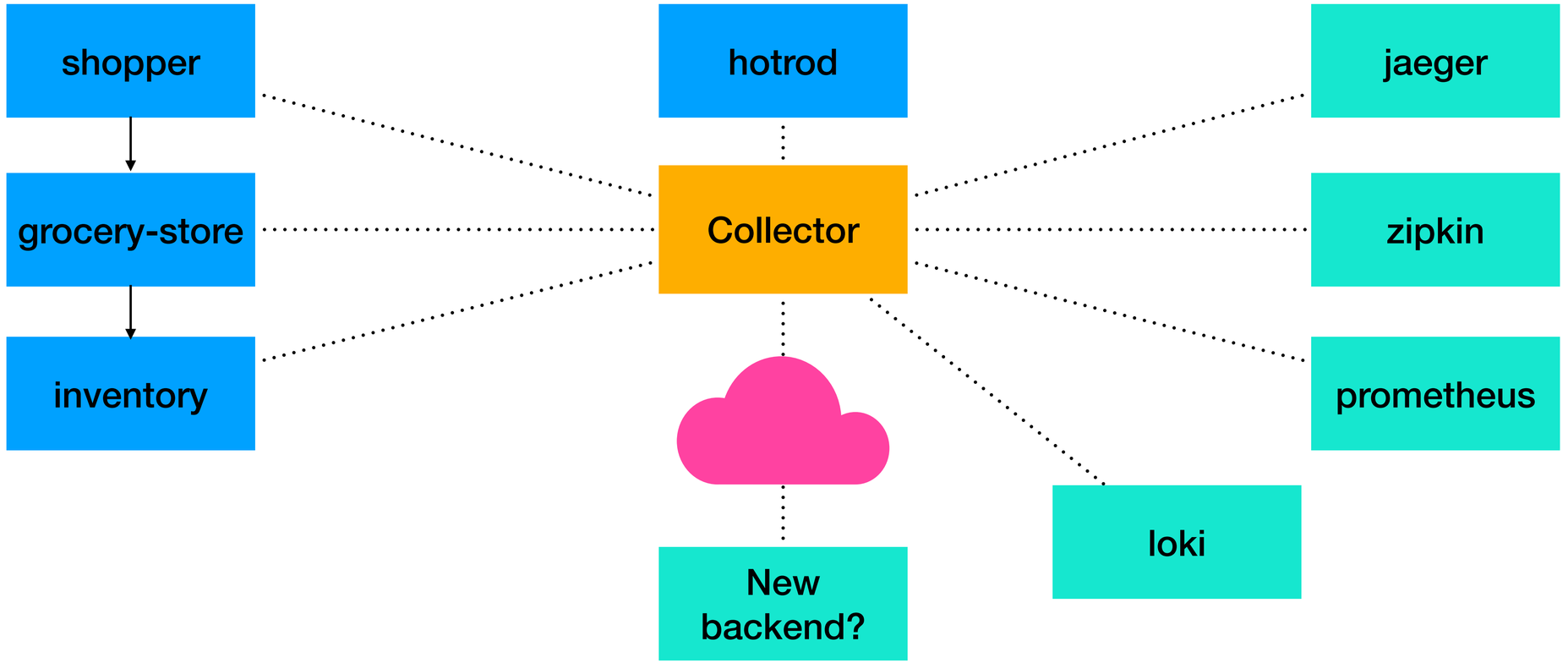
http://localhost:9090

Demo application



Ahhhhhhhh… what if… 🤔

Demo application



Configure exporters: otlp

```
exporters:
  otlp/lightstep:
    endpoint: ingest.lightstep.com:443
    headers:
      - lightstep-access-token: ${LS_ACCESS_TOKEN}

service:
   metrics:
     exporters: [logging, prometheus, otlp/lightstep]
   traces:
     exporters: [logging, jaeger, zipkin, otlp/lightstep]
```

Process the data

Configure processors: memory limiter

```
processors:
  memory_limiter:
    check_interval: 2s
    limit_mib: 2000
    spike_limit_mib: 800
service:
  pipelines:
    traces:
      processors: [memory_limiter]
    metrics:
      processors: [memory_limiter]
    logs:
      processors: [memory_limiter]
```

Configure processors: batch

```
processors:
  batch:
    timeout: 2s

service:
  pipelines:
    traces:
      processors: [memory_limiter, batch]
    metrics:
      processors: [memory_limiter, batch]
    logs:
      processors: [memory_limiter, batch]
```

Configure processors: resource detector

```
processors:
  resourcedetection/env:
    detectors: ["env"]
    timeout: 2s
    override: true

service:
  pipelines:
    traces:
      processors: [memory_limiter, batch, resourcedetection/env]
    metrics:
      processors: [memory_limiter, batch, resourcedetection/env]
    logs:
      processors: [memory_limiter, batch, resourcedetection/env]
```

Configure processors: resource detector

```
grocery-store:
  image: codeboten/grocery-store:chapter11
  container_name: grocery-store
  environment:
    - OTEL_EXPORTER_OTLP_ENDPOINT=opentelemetry-collector:4317
    - OTEL_EXPORTER_OTLP_INSECURE=true
    - OTEL_SERVICE_NAME=grocery-store
    - INVENTORY_URL=http://legacy-inventory:5001/inventory
    - OTEL_RESOURCE_ATTRIBUTES=env=o11yfest
```

Configure processors: metrics transform

```
processors:
  metricstransform:
    transforms:
      - include: request_count
        match_type: strict
        action: update
        new_name: http_request_count

service:
 pipelines:
   metrics:
     processors: [batch, resourcedetection/env, metricstransform]
```

Other processors coming soon

- Transform (https://github.com/open-telemetry/opentelemetry-collector-contrib/tree/main/processor/transformprocessor)
- Redaction (https://github.com/open-telemetry/opentelemetry-collector-contrib/tree/main/processor/redactionprocessor)

Component stability levels

- Stable: You’re good to go, breaking changes won’t happen without notice
- Beta: Configuration are deemed stable, breaking changes should be minimal
- Alpha: Ready to use for non-critical workloads
- In development: YMMV (your mileage may vary)
- Deprecated: This component’s going away

Break

Getting to production

Getting to production: telemetry

```
service:
  telemetry:
    metrics:
```

```
$ curl localhost:8888/metrics
```

Getting to production: health checks

```
extensions:
  health_check:

service:
  extensions: [health_check]
```

```
$ curl localhost:13133
```

More extensions for troubleshooting

- Performance Profiler
  - https://github.com/open-telemetry/opentelemetry-collector-contrib/blob/main/extension/pprofextension/README.md
- zPages
  - https://github.com/open-telemetry/opentelemetry-collector/tree/main/extension/zpagesextension

Getting to production: note about sampling

```
probabilistic_sampler:
    hash_seed: 22
    sampling_percentage: 15.3
```

```
tail_sampling:
  decision_wait: 10s
  num_traces: 100
  expected_new_traces_per_sec: 10
  policies:
    [
        {
          name: test-policy-1,
          type: always_sample
        },
        {
          name: test-policy-2,
          type: latency,
          latency: {threshold_ms: 5000}
        },
    ]
```

Getting to production: note about TLS

```
tls:
  insecure: false
  ca_file: server.crt
  cert_file: client.crt
  key_file: client.key
  min_version: "1.1"
  max_version: "1.2"
```

Getting to production: custom collector

https://github.com/open-telemetry/opentelemetry-collector/releases/tag/v0.51.0

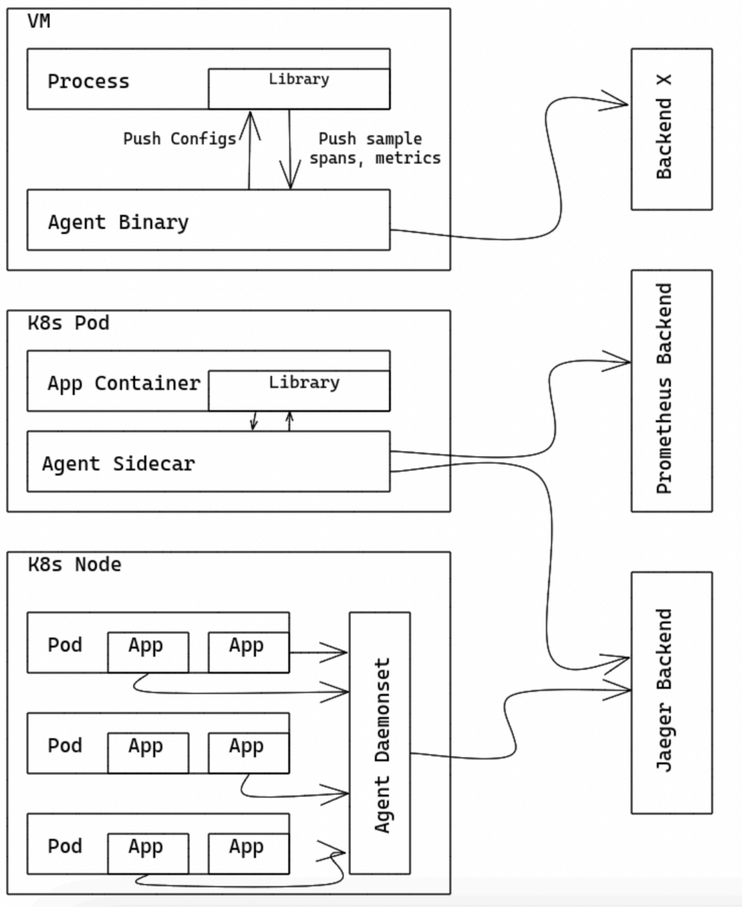
./ocb --config ./config/collector/core-manifest.yaml

./ocb --config ./config/collector/contrib-manifest.yaml

Deployment scenarios

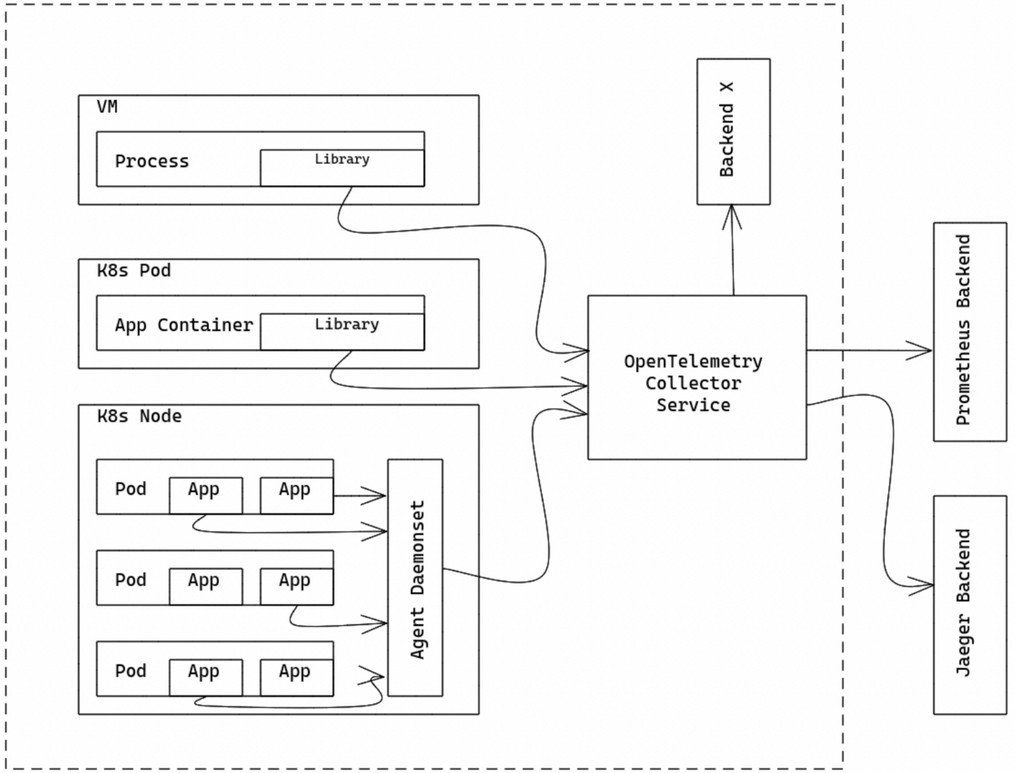
Deployment modes: agent

- Deploys a collector on each node
- Collects system level metrics for the node
- Acts as a destination for all telemetry from applications on the given node (OTLP defaults to localhost)



Deployment modes: gateway

- Deployed as a standalone service
- Destination for agents or individual applications
- Central location for processing all telemetry
- Horizontally scalable



Deployment tools

- Helm charts
  - https://github.com/open-telemetry/opentelemetry-helm-charts
- Kubernetes Operator
  - https://github.com/open-telemetry/opentelemetry-operator
- Nomad guide
  - https://github.com/hashicorp/nomad-open-telemetry-getting-started
- Kubecon Talk: OpenTelemetry Collector Deployment Patterns
  - https://www.youtube.com/watch?v=WhRrwSHDBFs
  - https://github.com/jpkrohling/opentelemetry-collector-deployment-patterns

So many more resources

- Project website
  - https://opentelemetry.io
- GitHub repos
  - https://github.com/open-telemetry/opentelemetry-collector
  - https://github.com/open-telemetry/opentelemetry-collector-contrib
  - https://github.com/open-telemetry/opentelemetry-collector-releases
- Slack
  - https://cloud-native.slack.com/archives/C01N6P7KR6W
- Curated list of OpenTelemetry resources
  - https://github.com/magsther/awesome-opentelemetry

Lightstep
from ServiceNow

Visit the Lightstep booth to download a free digital copy!

In this report you'll learn how to:

> Accurately assess upcoming shifts in technology
> Seamlessly roll out OpenTelemetry across your org
> Implement a robust observability pipeline based on OTel