Metrics

Document Level Classification

200

Introduction

This doc describes the high level details needed to understand the metrics systems in use at Epic. The intended audience is end-users, i.e. service owners and developers, of the metrics tools. If you are looking for a lower-level guide for observability/cloud engineers, see CTObs: OpenTelemetry Operations Guide.

tl;dr - what should I be using for what I'm building today?

These are **not hard rules** but general guidance that will save you time in the future as we reduce our New Relic footprint.

- Use OpenTelemetry for sending new metrics. Configure the OTEL SDK, or an OTEL-compatible framework (for example, micrometer), to talk to your cluster-local OTEL collector. (see below)
- Use the OTEL agent for auto-instrumentation, if auto-instrumentation it makes sense for your app.
- If your application has a choice between OTEL and Prometheus, use OTEL. Collection of Prometheus metrics is supported for off-the-shelf tools that only export Prometheus, or for internal applications where OTEL is not an option.
- Build Dashboards in Grafana
- Build Alerts in Grafana

Don't

- Instrument new apps using the proprietary New Relic Agent or SDK
- Build your dashboards in New Relic

- Configure new alerts in New Relic
- Use Epic Metrics for application or infrastructure metrics that could be collected with OTEL. Epic Metrics should be limited to client-side or analytics use cases.

Key Metric Systems for Substrate Environments

Grafana

Grafana is our preferred visualization and alerting tool. We treat it as a single pane of glass for metric data. It supports multiple data sources such as Prometheus, Epic Metrics, New Relic, CloudWatch, and allows you to build dashboards and configure alerts based on any combination of these data sources. We anticipate our metric storage systems will come and go every few years, but Grafana will remain our consistent view into these various systems and the go-to for viewing service health.

New Relic

New Relic is a third party SaaS tool for Metrics, Traces, and Logs. It has been our standard sink for metric data for over 3 years. For various reasons including cost, performance, and vendor lock-in risk, we do not want to continue to invest in New Relic and are working on a migration path away from this tool.

We do not use New Relic for logging or traces. Though some teams do send small amounts of both data types to the system, we do not view New Relic as a preferred solution for either and our pricing does not support these use cases at a large scale.

Chronosphere

Chronosphere is a new vendor that provides managed M3, a Prometheus-compatible metric storage system. For our purposes, it is essentially a large, scalable Prometheus server with useful governance tools that will help Epic manage cost and data access concerns.

Epic Metrics

Epic Metrics is our internally developed metrics system, designed primarily for generating metrics based off of the analytics event stream. It is used in particular for analyzing client data, and powers a number of critical dashboards including the Fortnite Operational Health board. Using Epic Metrics is not covered in this doc as it is not Substrate-specific, but if you think Epic Metrics might be right for your use case, see the documentation, or reach out to #data-support-ext.

How to emit metrics

We support two ways to emit metrics in Substrate: OpenTelemetry (**preferred)** and Prometheus. There is still extensive use of New Relic agents and SDKs at Epic, but new software development should use either OTEL or Prometheus and service teams should move away from proprietary NR metrics as time allows. OTEL and Prometheus are both open standards that are supported by multiple vendors and using these to instrument your service helps Epic to avoid expensive vendor lock-in.

Using OpenTelemetry

The approaches below are not mutually exclusive, you can use auto-instrumentation and custom instrumentation in tandem.

Using Auto-Instrumentation

OTEL supports agent-based instrumentation for some languages - including Java, Go, and NodeJS - with auto-instrumentation for common libraries. This removes a lot of toil when initially instrumenting a service as common things like HTTP requests, database calls, etc can be instrumented with reduced effort. We use the OTEL Operator to provision OpenTelemetry Collectors in Substrate Clusters. Another feature of the operator is its ability to inject auto-instrumentation agents directly into a running application without the need to rebuild the application. We currently support injected auto-instrumentation for Java. Other auto-instrumentation **is available but less tested**. To enable auto-instrumentation this way, simply annotate your pod deployment:

```
annotations:
  instrumentation.opentelemetry.io/inject-java: "observability/java-v2"
  instrumentation.opentelemetry.io/inject-go: "observability/golang" #
  instrumentation.opentelemetry.io/inject-nodejs: "observability/nodejs
  instrumentation.opentelemetry.io/inject-python: "observability/python
```

If you have multiple containers in your pod spec, the first container will be instrumented by default. To clarify to the OTEL operator which container(s) should be instrumented, add the following annotation:

```
instrumentation.opentelemetry.io/container-names: "myapp,myapp2"
```

You will need to re-deploy or restart your application to see this take effect.

Warning

Injected auto-instrumentation is not guaranteed to work perfectly as it makes assumptions about the environment and entrypoint script of the containers running the application. There is potential for this to cause your application to crash on startup, so validation in non-prod environments is strongly encouraged before attempting to use this in production.

If your service is built on Epic's shared uberjar-service-corretto image, you will need to at least build **360135c7566d3e1cbbbd48e5469dfa26ee4938e7.b177** (or an image built after June 14, 2023) to enable auto-instrumentation.

Custom Label Support for OTEL Auto-instrumentation

We currently have limited support to add custom labels to auto-instrumented metrics. The following labels can be set by defining an **OTEL_RESOURCE_ATTRIBUTES** environment variable in your app:

- epic.service
- epic.owner
- epic.euid
- epic.env
- epic.version

For example:

```
env:
  - name: OTEL_RESOURCE_ATTRIBUTES
    value: epic.service=myapp,epic.env=gamedev,epic.version=1.2.3
```

Manually with SDKs

Substrate

There are OpenTelemetry SDKs for many languages available. The specifics of each SDK is beyond the scope of this doc, but the general approach when using OTEL is:

1. Initialize the SDK with an OTLP exporter pointed directly at the cluster-local collector address. The address is consistent across all Substrate clusters. Whether you use the HTTP or GRPC endpoint may be language-specific, check with the SDK doc for the recommended exporter type. **If you are also using the auto-instrumentation agent**, **this is done automatically** and you can skip this step. Note, you can initialize this in your SDK or use the **OTEL_EXPORTER_OTLP_ENDPOINT** environment variable.
    1. http://otlp-collector.observability.svc.cluster.local:4317 (GRPC)
    2. http://otlp-collector.observability.svc.cluster.local:4318 (HTTP)
2. Use the initialized SDK objects to create any Metrics you need in your application.

You can do "easy mode" initialization with the **inject-sdk** annotation, which will pass the correct env vars to your app to interact with the OTEL collectors in your cluster.

```
annotations:
  instrumentation.opentelemetry.io/inject-sdk: "observability/inject-sd
```

Environment: A list of environment variables required when instrumenting an app without the help of an OTEL agent.

Environment Variable	Meaning

`OTEL_NODE_IP`	IP address of the node the po is running on.
`OTEL_POD_IP`	IP address of the current pod.
`OTEL_EXPORTER_OTLP_TRACES_ENDPOINT`	Where to send traces (OTLP endpoint).
`OTEL_LOGS_EXPORTER`	Defines which logs exporter to use (e.g., `otlp`).
`OTEL_INSTRUMENTATION_MICROMETER_ENABLED`	Enables Micrometer-based metrics (for Spring/Java apps)
`OTEL_INSTRUMENTATION_EXTERNAL_ANNOTATIONS_ENABLED`	Allows using external metadat (e.g., K8s annotations) in telemetry.
`OTEL_JMX_CONFIG`	Path to the JMX config file for JVM metrics.
`OTEL_METRICS_EXPORTER`	Defines the metrics exporter t use (e.g., `otlp`, `none`).

`OTEL_EXPORTER_OTLP_PROTOCOL`	Protocol for OTLP export (`grpc` or `http/protobuf`).
`OTEL_SERVICE_NAME`	Logical name of the service — must be unique per service.
`OTEL_EXPORTER_OTLP_ENDPOINT`	Base OTLP endpoint for all signal types.
`OTEL_RESOURCE_ATTRIBUTES_POD_NAME`	Sets the pod name as a resource attribute.
`OTEL_RESOURCE_ATTRIBUTES_NODE_NAME`	Sets the node name as a resource attribute.
`OTEL_PROPAGATORS`	Defines context propagation (e.g., `tracecontext`, `baggage`)
`OTEL_TRACES_SAMPLER`	Sets the trace sampling strategy.
`OTEL_TRACES_SAMPLER_ARG`	Sets sampling parameters (e.g., trace rate).

`OTEL_RESOURCE_ATTRIBUTES`	Custom resource attributes (comma-separated key=value pairs).
`OTEL_AGENT_ENABLED`	(Non-standard) Custom flag to toggle internal agent logic.

Over time we'll publish some language specific guidance/tutorials to help you navigate this setup. As we're still in early days, we're focusing on Java instrumentation. See the Using the OpenTelemetry Java SDK doc for some tips on getting started with Java.

Sending from other sources

We maintain a central collector for use cases outside of substrate and oldprod. Depending on on where your service lives, you should use the **internal** or **external** OTEL collectors. The internal collector sits on the service network, the external collector is reachable from the internet. In general, if you can reach the service network, you should just use the internal collector.

- interal collector: https://central-otel-collector.fbfb.live.use1a.on.epicgames.com(HTTP)
- external collector: https://external-metrics-collector.fbfb.live.use1a.on.epicgames.com (HTTP)

These OTEL collectors only accept traffic on 443 and not the default otlp/http port

Some notes about using these collectors:

- Only OTLP HTTP exporters are supported, GRPC is not

- You will need basic auth credentials. Reach out to #ct-obs-support-ext for help with this.
- You will need to supply a header, **X-Epic-Tenant-ID**, with all data coming
- Your data points must supply a **service.name** and **service.instance.id** resource attribute.
    - service.name - uniquely identifies a sending service, typically this is <yourservice>-environment, i.e. grafana-prod, grafana-gamedev, etc
    - service.instance.id - this is a unique identifier representing the metric producer. Typically, this can be pod name, host name, etc of your service, though OTEL SDKs can initialize this for you to something like a GUID.

Requests that do not supply all of the above will be dropped. See #ct-obs-support-ext if you need help with this.

Using Prometheus

Most languages and some frameworks (such as Springboot's Actuator) have ways to export telemetry with Prometheus. This is done by running an HTTP endpoint, usually **/metrics**, that gets discovered by a Prometheus scraper and periodically scraped (usually every 30s).

If your application is exporting Prometheus metrics, all you need to do to get them collected is to annotate your pods, and ensure that the port is listed as a container port. This can easily be done in epic-app by setting additional **podAnnotations.**

```
prometheus.io/scrape: "true"
prometheus.io/port: "8080" # change to the port your pod's exporter run
prometheus.io/path: /metrics # change to the path your pod's metrics ar
```

If the port specified in **promethus.io/port** is listed as a container port, you should be good to go. If your metrics exporter runs on a different port than your main application ports, make sure it's listed in your pod spec:

```
ports:
  - name: http-metrics
    containerPort: 8080
    protocol: TCP
```

Be aware that there are multiple ways to signal to Prometheus to scrape your metrics. Service resources can be annotated as well, and ServiceMonitor/PodMonitor Custom Resources are also supported in our environment. Configuring a combination of these can lead to metrics getting scraped more than once, which can be expensive to ingest and confusing when querying. If in doubt, just annotate your pods.

How your metrics get collected

We are currently in a state of migration, but the end goal is to have all service and infrastructure metrics in Substrate emitted via OpenTelemetry SDKs/auto-instrumentation or Prometheus and collected by OpenTelemetry collectors. Clusters sending metrics to New Relic will be slowly switched over to only Chronosphere, as sending to New Relic is the deprecated path (Reference the Observability Status for information regarding New Relic deprecation). Our collectors can be reconfigured on the fly to send to new destinations, which enables us to migrate your metrics between vendors without significant retooling as long as you are using OTEL or Prometheus.

We run three collectors in Substrate clusters that ship your metrics.

- **oltp-collector** - Receives telemetry from OTEL Auto-instrumentation agents and SDKs. OTEL uses the "push" method of sending metrics, where each service is responsible for connecting to the collector and sending metrics. These metrics go to either Chronosphere or New Relic, depending on the cluster configuration.

- **core-prometheus-collector -** Acts as a prometheus agent, discovering metrics that are exported over HTTP (usually via a **/metrics** endpoint on the individual pods) and scraping them on an interval. These metrics go to either Chronosphere or New Relic, depending on the cluster configuration.
- **keda-collector** - Receives the metric firehose from both the **otlp-collector** and **core-prometheus-collector** and filters down to a very small set of metrics that are used for KEDA autoscaling, sending them to a local Prometheus workspace. The exact list of metrics that can be used by KEDA is configurable.

How you can query your metrics

Generally, use Grafana. We have data sources for all of our metric systems in Grafana, and building dashboards in Grafana rather than in vendor-specific UIs (i.e. New Relic) will help you avoid losing them entirely in the future if vendors change. Creating new dashboards in New Relic should be avoided as this is the deprecated path for dashboarding. Teams should should work to start using Grafana for dashboarding and querying Chronosphere metrics (Reference the Observability Status for information regarding New Relic deprecation).

Chronosphere

If you have been onboarded to Chronosphere, you will have a "Chronosphere Tenant" in Grafana. All metrics from your EKS clusters will exist in this tenant. Queries are written in PromQL. You can use labels to filter metrics down to the account (**aws_account**), cluster (**k8s_cluster_name**), or service level (**service_name**).

- OTEL "Unified" Dashboard for Java - if you have enabled auto-instrumentation in Java, you can find some basic metrics for your application here.
- Example Chronosphere Query - trivial example of a rate query of HTTP requests by service.

FAQ

Prometheus or OTEL, what should I use?

Both OTEL and Prometheus are open standards we will continue to support indefinitely. OTEL is the **preferred** solution for all new development.

- If you're writing a completely new service, use OTEL. Auto-instrumentation will help you get started a lot faster.

- If you're migrating from New Relic agents and SDKs, use OTEL. The OTEL agent and SDKs will likely be a more familiar experience for you if you're already using NR.
- If you have existing patterns/libraries that use Prometheus, you are strongly encouraged to migrate to OTEL.
- Some services (especially off-the-shelf third party/OSS tools) only offer Prometheus, so in that case the choice is simple: just use Prometheus.

When should I use Epic Metrics?

Epic Metrics is an internally developed metrics platform that is primarily designed to handle client-side metrics and analytics (i.e. Data Warehouse) use cases. If you have server-side metrics you want to expose both as a metric for observability and as a table for analytics, use Epic Metrics.

For application performance monitoring, infrastructure, service health, etc, use OpenTelemetry and query your metrics with Prometheus (Chronosphere).

- Chronosphere User Guide
- Instrumentation for Spring Boot Applications
- Kubernetes Infrastructure Metrics
- Using KEDA with Service Metrics
- Using the OpenTelemetry Java SDK
