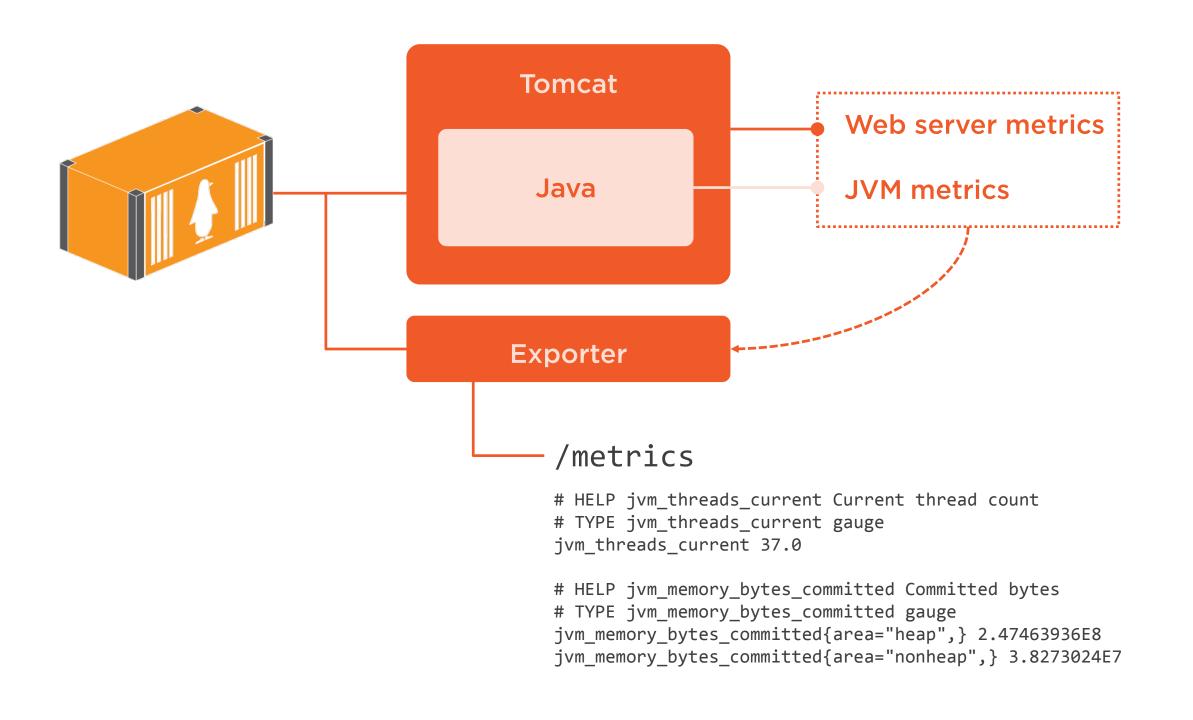
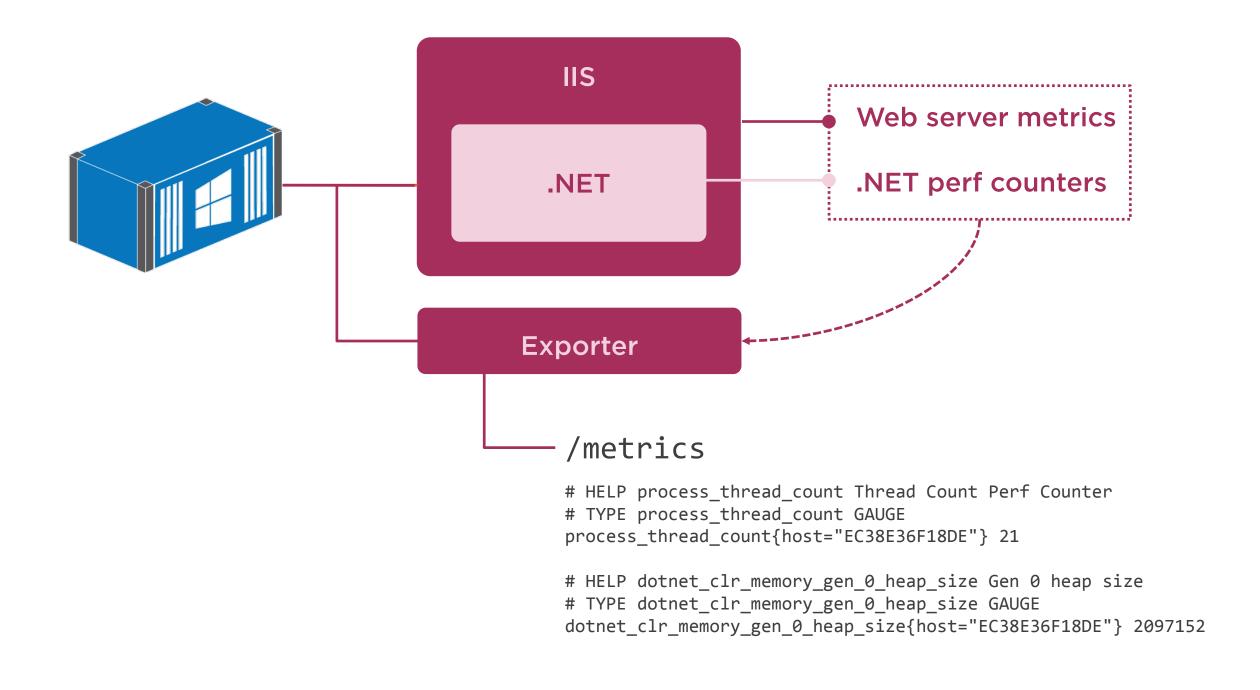
Exposing Application Metrics to Prometheus

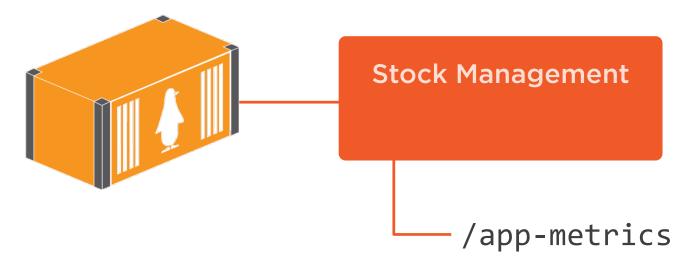


Elton Stoneman
DEVELOPER ADVOCATE

@EltonStoneman https://blog.sixeyed.com

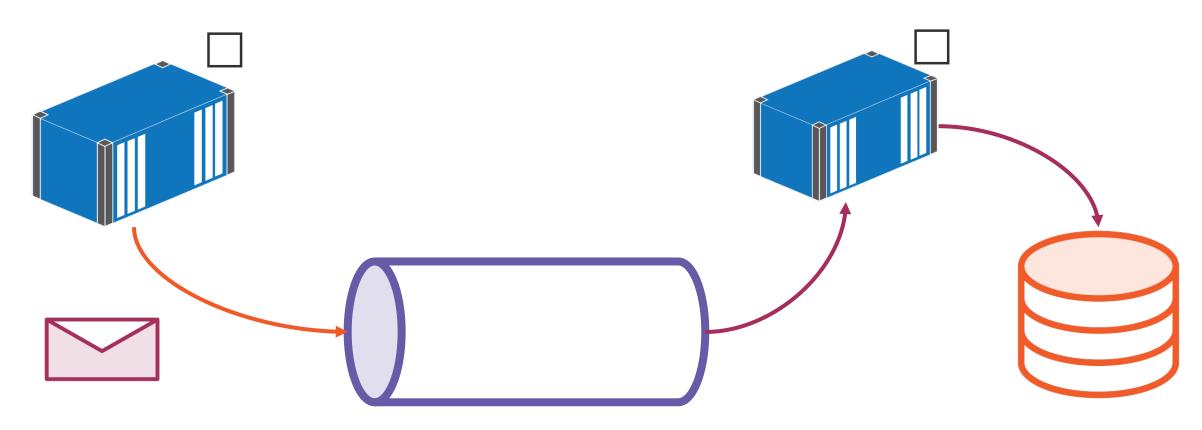




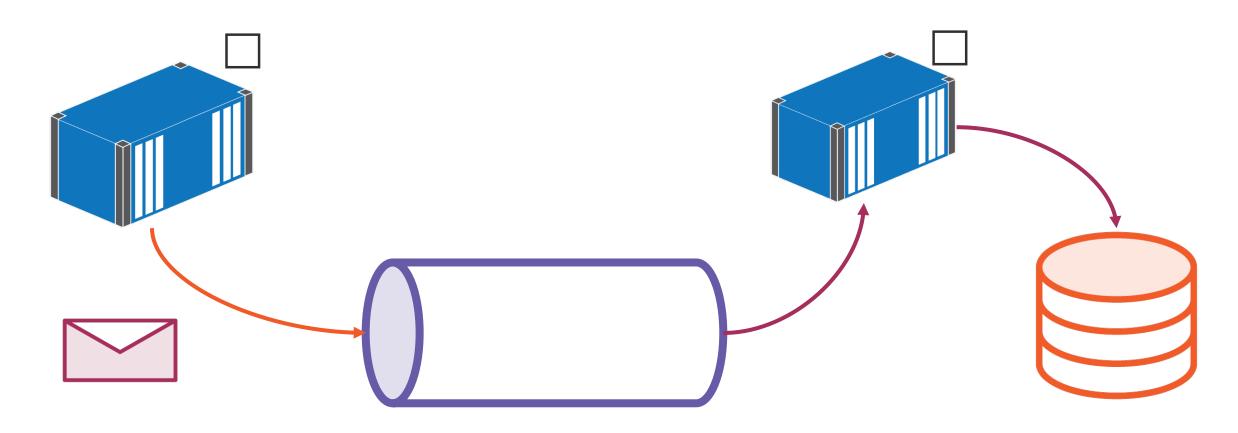


```
# HELP StockManagement_SessionStatus Count
# TYPE StockManagement_SessionStatus counter
StockManagement_SessionStatus{status="started"} 20
StockManagement_SessionStatus{status="order-submitted"} 12
# HELP StockManagement_ActiveSessions Count
# TYPE StockManagement_ActiveSessions gauge
StockManagement ActiveSessions 6
```

```
events_processed{status="published"} 1200
events_processed{status="handled"} 1172
events_processed{status="processed"} 1060
events_processed{status="failed"} 112
```



```
events_processed{type="user-login",status="failed"} 81
events_processed{type="pwd-reset",status="failed"} 20
events_processed{type="captcha",status="failed"} 11
```



service_calls{svc="address-lookup", status="processed"} 420 service_calls{svc="price-lookup", status="timed-out"} 32 service_calls{svc="sale-check", status="processed"} 160 service_calls{svc="stock-update", status="failed"} 15

new_customers 6000 sales_count 360 sales_value 86400

Module Overview



Exposing Application Metrics

- Using Prometheus client libraries
- Recording metrics
- Exposing metrics endpoints



JVM

Stock Management

Metrics Exporter

Application metrics

/app-metrics

Web server metrics

JVM metrics

/metrics



JVM

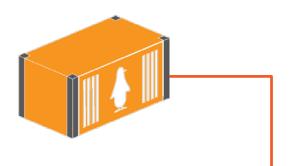
Stock Management

Application metrics

Web server metrics

JVM metrics

/metrics



JVM

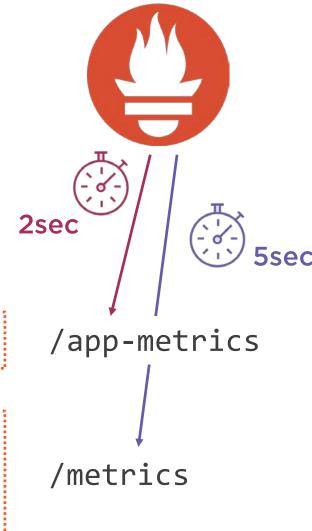
Stock Management

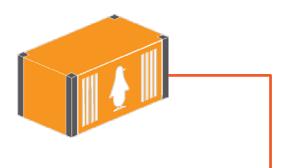
Metrics Exporter

Application metrics

Web server metrics

JVM metrics





JVM

Stock Management

Metrics Exporter

Prometheus client library

Host metrics endpoint Record custom metrics

Application metrics

/app-metrics

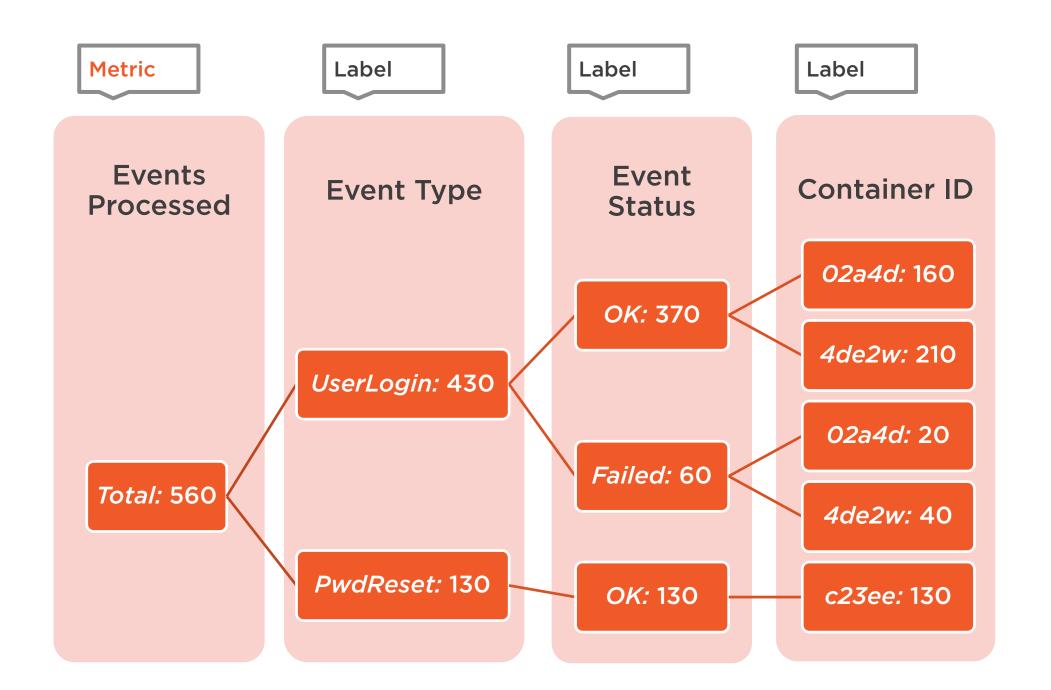
Web server metrics

JVM metrics

/metrics

Prometheus Client Libraries

Declaring and using metrics (pseudo-code)



```
counter.Labels
    "UserLogin",
    "OK",
    host_name
    .Inc()
```

Prometheus Client Libraries

Incrementing labelled metrics (pseudo-code)

Demo



Adding Application Metrics to Java Apps

- Maven Prometheus client library
- Create and register custom metrics
- Increment counters and gauges
- Servlet hosting metrics endpoint

Demo



Adding Application Metrics to .NET Apps

- NuGet Prometheus client library
- Create and register custom metrics
- Increment counters and gauges
- Custom hosting metrics endpoint



JVM

Stock Management

Prometheus client

Metrics Exporter

• Application metrics

Web server metrics

JVM metrics

```
private static final Counter _SessionCounter =
    Counter.build()
        .name("StockManagement_SessionStatus")
        .help("Count")
        .labelNames("host", "status")
        .register();
```

Prometheus Java Client

Creating and registering metrics

```
_SessionCounter.labels(_Host, "started").inc();
_SessionGauge.labels(_Host).inc();
_SessionGauge.labels(_Host).dec();
```

Prometheus Java Client Using labelled metrics

```
//Global.asax.cs
DefaultCollectorRegistry.Instance.Clear();
new MetricServer(50506).Start();
```

Prometheus .NET Client

Hosting metrics endpoint - custom

```
<!--web.xml-->
<servlet-mapping>

<servlet-name>metrics</servlet-name>

<url-pattern>/app-metrics/</url-pattern>
</servlet-mapping>
```

Prometheus Java Client

Hosting metrics endpoint - servlet

Prometheus Client Libraries

Official

Java/Scala

Go

Ruby

Python

```
counter = prometheus.NewCounter(prometheus.CounterOpts{
    Name: "new_users", Help: "New users", Labels: "source"})
prometheus.MustRegister(counter)
counter.WithLabelValues("Bing").Inc()
http.Handle("/metrics", promhttp.Handler())
```

Prometheus Go Client

Register metrics, increment counters, host endpoint

```
defaultRegistry = CollectorRegistry()
counter = Counter('requests', 'Req', 'path', defaultRegistry)
counter.labels('/').inc()
#...
start_http_server(8000)
```

Prometheus Python Client

Register metrics, increment counters, host endpoint

Prometheus Client Libraries

Official

Java/Scala

Go

Ruby

Python

Community

.NET

Node.js

Erlang

Haskell

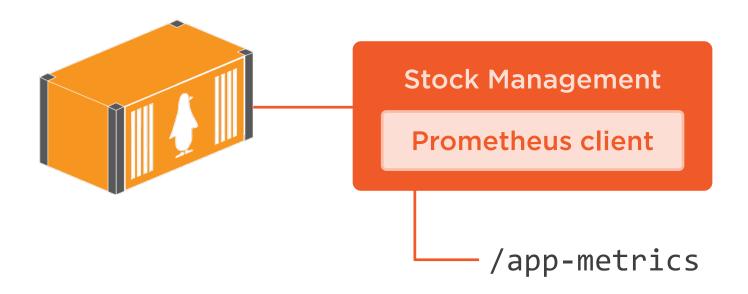
Rust

Demo

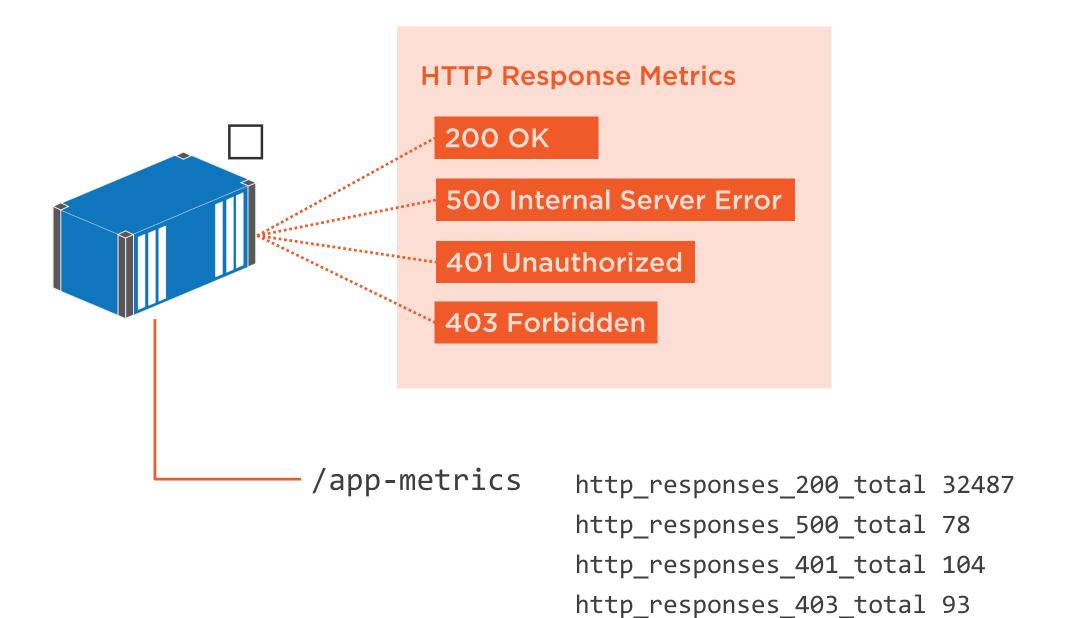


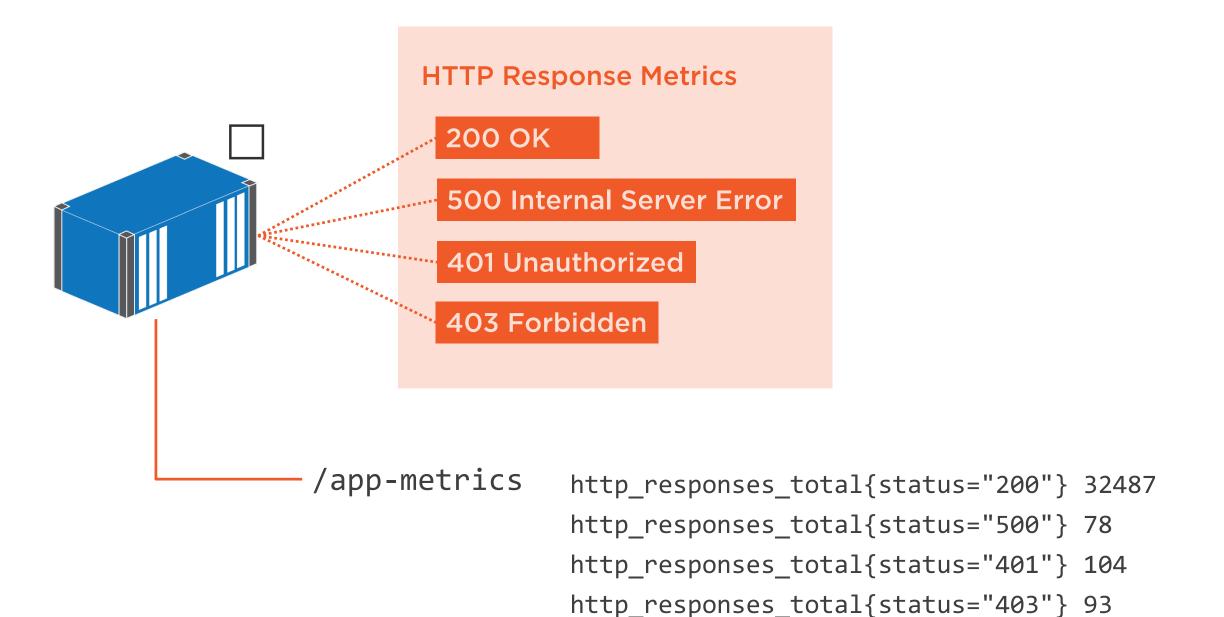
Scraping Application Metrics

- Upgrade apps to v2
- Generate traffic to record metrics
- Querying app metrics with PromQL

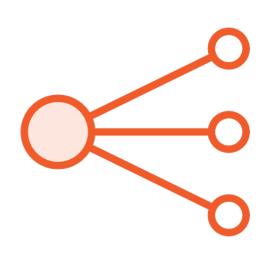


```
# HELP StockManagement_SessionStatus Count
# TYPE StockManagement_SessionStatus counter
StockManagement_SessionStatus{status="started"} 20
StockManagement_SessionStatus{status="order-submitted"} 12
# HELP StockManagement_ActiveSessions Count
# TYPE StockManagement_ActiveSessions gauge
StockManagement_ActiveSessions 6
```

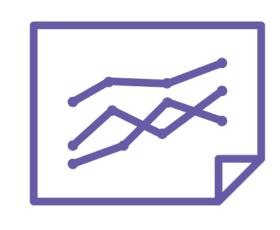




Instrumentation Guidelines









CommunicationOutcomes

Workflows
Correlation ID

Business Metrics Real-time

Logging Entry count

"...export the total number of info/error/warning lines that were logged by the application as a whole"

prometheus.io

DOCS



- (7 INTRODUCTION
- A CONCEPTS
- **PROMETHEUS**
- **✓** VISUALIZATION
- OPERATING
- </>
 INSTRUMENTING
- △ ALERTING
- ♠ BEST PRACTICES

Metric and label naming

Consoles and dashboards

Instrumentation

Histograms and summaries

Alerting

Recording rules

When to use the Pushgateway

METRIC AND LABEL NAMING

The metric and label conventions presented in this document are not required for using Prometheus, but can serve as both a styleguide and a collection of best practices. Individual organizations may want to approach some of these practices, e.g. naming conventions, differently.

- Metric names
- Labels
- Base units

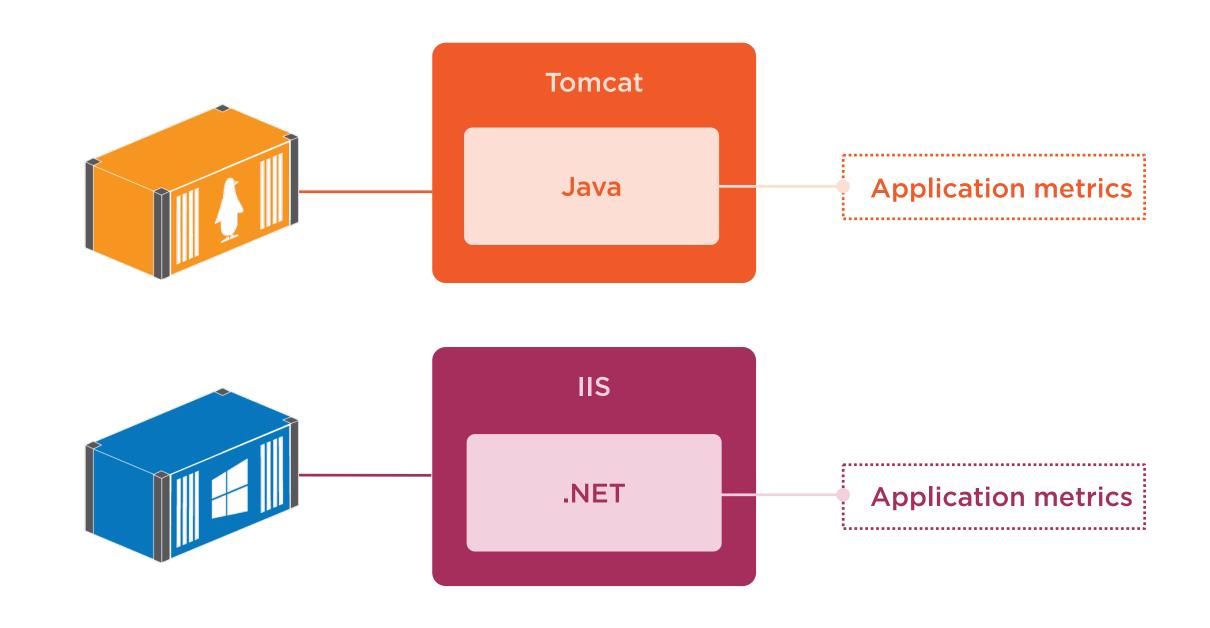


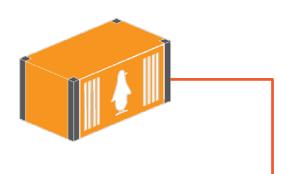
Metric names

A metric name...

https://is.gd/THNwiA

- ...should have a (single-word) application prefix relevant to the domain the metric belongs to. The prefix is sometimes referred to as namespace by client libraries. For metrics specific to an application, the prefix is usually the application name itself. Sometimes, however, metrics are more generic, like standardized metrics exported by client libraries. Examples:
 - o prometheus_notifications_total (specific to the Prometheus server)
 - o process cpu seconds total (exported by many client libraries)
 - http_request_duration_seconds (for all HTTP requests)
- ...must have a single unit (i.e. do not mix seconds with milliseconds, or seconds with bytes).
- ...should use base units (e.g. seconds, bytes, meters not milliseconds, megabytes, kilometers). See below for a list of base units.
- ...should have a suffix describing the unit, in plural form. Note that an accumulating count has total as a suffix, in addition to the unit if applicable.
 - o http request duration seconds
 - o node memory usage bytes
 - http requests total (for a unit-less accumulating count)
 - o process cpu seconds_total (for an accumulating count with unit)
- ...should represent the same logical thing-being-measured across all label dimensions.
 - request duration
 - o bytes of data transfer





JVM

Stock Management

Metrics Exporter

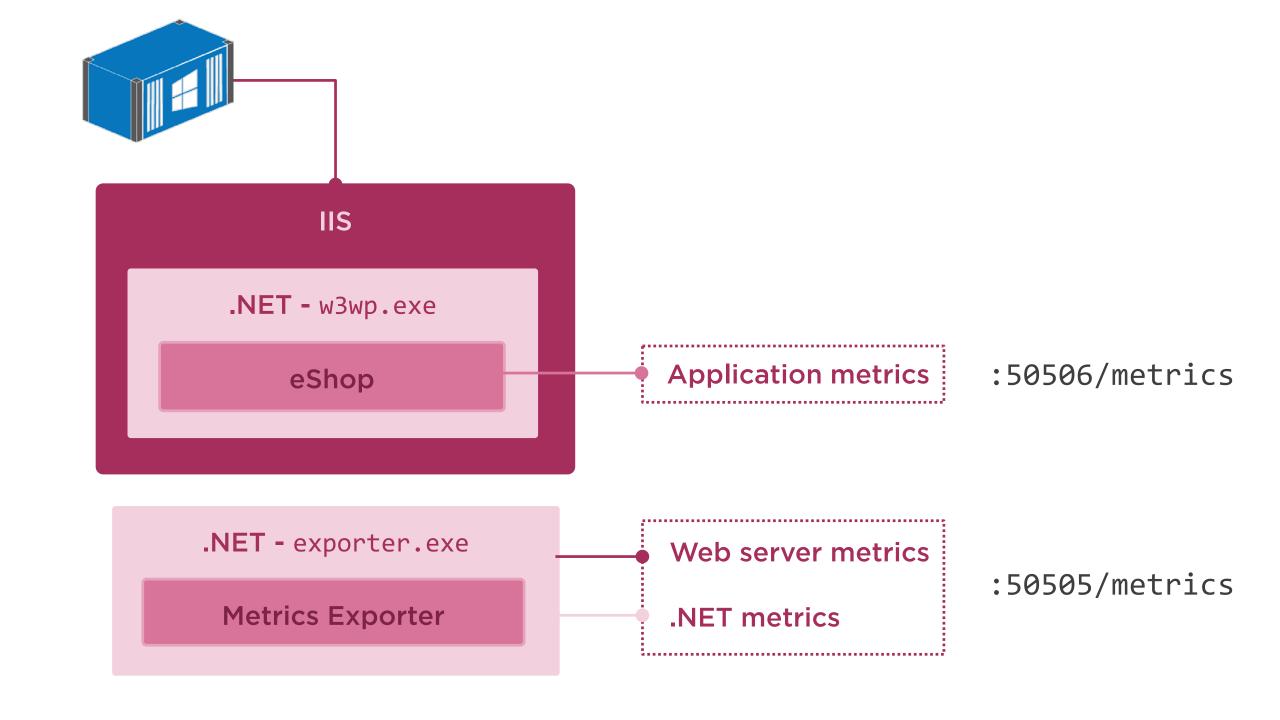
Application metrics

/app-metrics

Web server metrics

JVM metrics

/metrics



Module Summary



Exposing Application Metrics

- Client libraries Java & .NET
- Exposing metrics endpoints
- Instrumentation best practices
- Striping metrics with labels

Coming Next



Exposing Docker Metrics to Prometheus

- Enabling metrics
 - Docker Desktop Mac & Windows
 - Docker Engine Linux & Windows
- Docker engine metrics
- Docker swarm metrics
- Querying Docker metrics