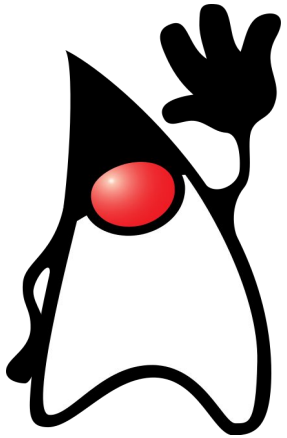
Prometheus for Java Developers



About Me

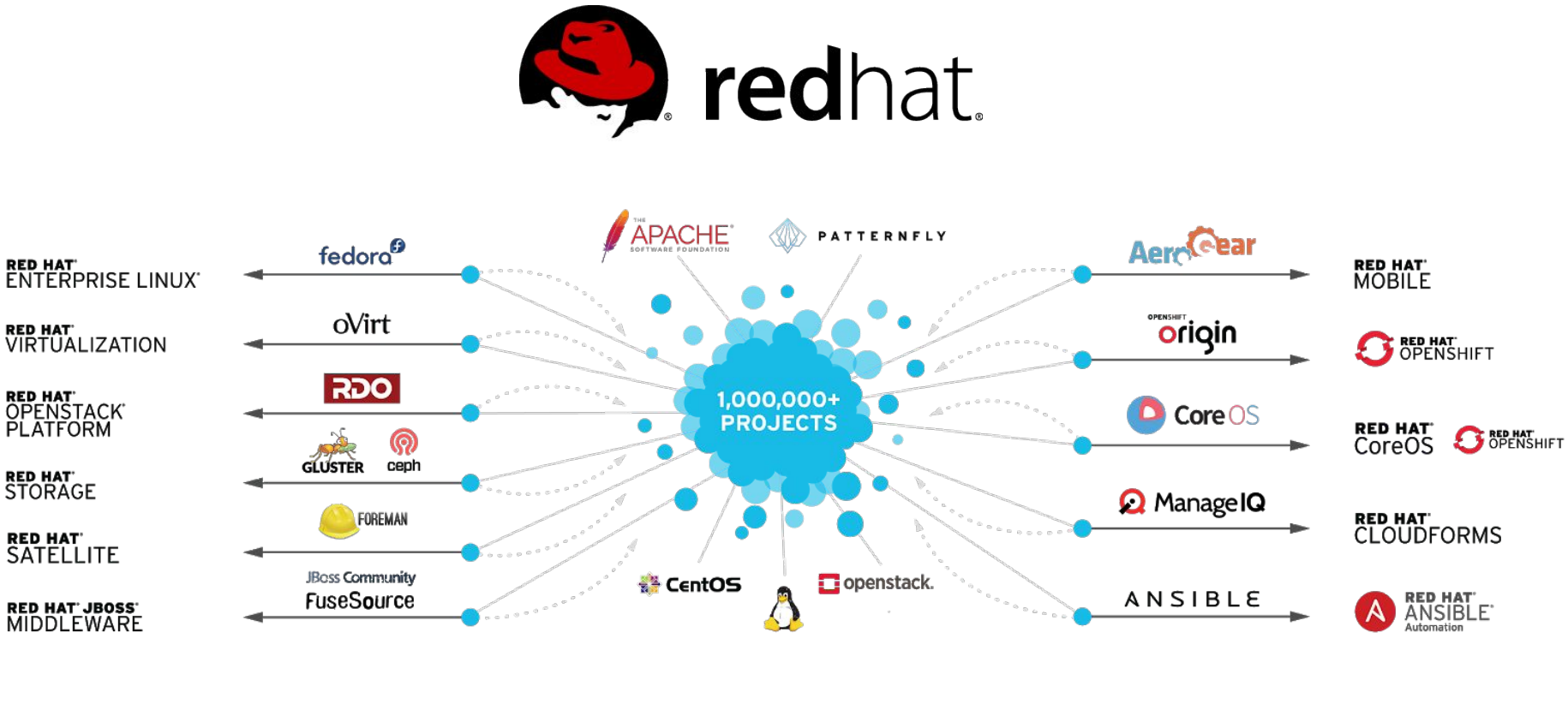
Paul Gier

Software Engineer at Red Hat

pgier@redhat.com

github.com/pgier

IRC: pgier


redhat.
RED HAT ENTERPRISE LINUX
RED HAT VIRTUALIZATION
RED HAT OPENSTACK PLATFORM
RED HAT STORAGE
RED HAT SATELLITE
RED HAT JBOSS MIDDLEWARE
fedora
oVirt
RDO
GLUSTER
ceph
FOREMAN
JBoss Community
FuseSource
THE APACHE SOFTWARE FOUNDATION
PATTERNFLY
1,000,000+ PROJECTS
CentOS
openstack.
AeroGear
OPENSHIFT origin
CoreOS
ManageIQ
ANSIBLE
RED HAT MOBILE
RED HAT OPENSHIFT
RED HAT CoreOS
RED HAT OPENSHIFT
RED HAT CLOUDFORMS
RED HAT ANSIBLE Automation

RH0064-4

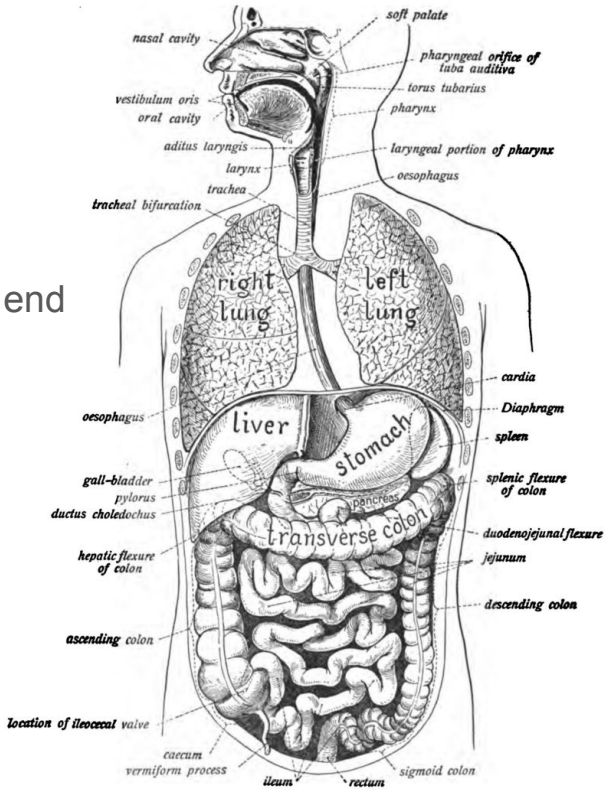
Overview

1. Observability
2. Prometheus
3. Using Exporters
4. Instrumenting Applications
5. Summary/Questions

Also demos and some JBoss stuff

Observability

- What is it?
  - The ability to see the guts of live systems
- Blackbox vs. Whitebox monitoring of a Human
  - Blackbox - food goes in one end and comes out the other end
    If the food stops coming out, there is a problem
  - Whitebox - see what's happening inside
- Which way is better?
  - 9 out of 10 doctors prefer whitebox




OFFICIAL WORLD
RECORD SETTER
RECORD HOLDER
© Caters News Agency

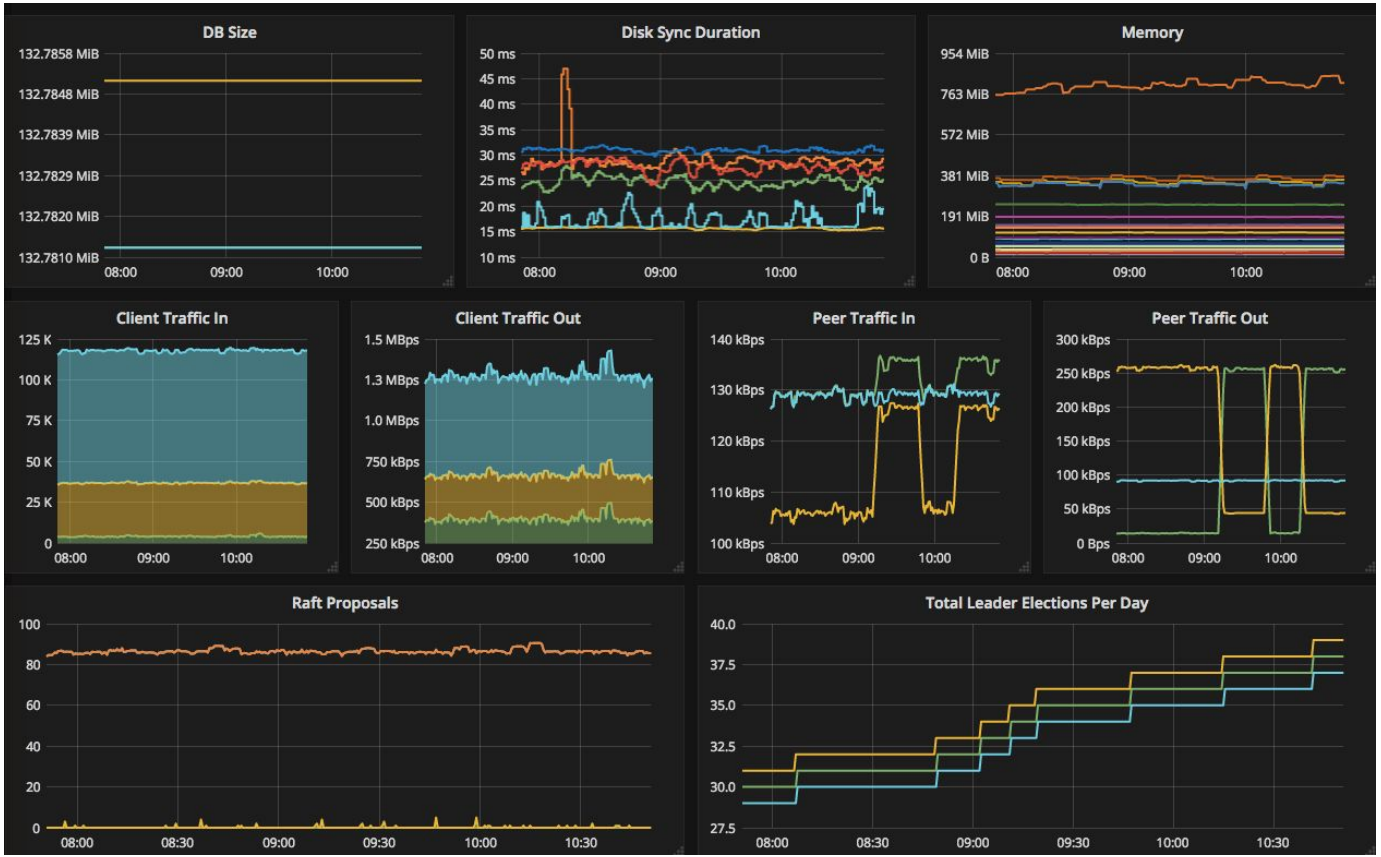
Blackbox Monitoring

```
#!/bin/bash

SERVER_URL=”www.google.com”
SERVER_DOWN=0

while [ “$SERVER_DOWN” -eq 0]; do
  sleep 2
  curl --silent --output /dev/null $SERVER_URL
  SERVER_DOWN=$?
done

echo "Alert!!!"
```

Blackbox Monitoring

Did you try turning it off and on again?

Whitebox Monitoring



Blackbox

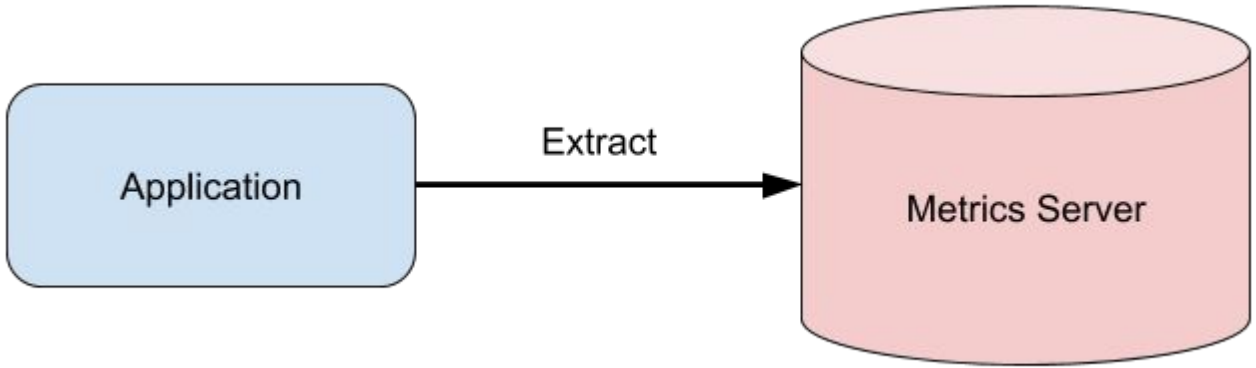
- Easy to implement
- Easy to understand
- Doesn’t require a lot of infrastructure (servers, storage, etc)

Whitebox

- Diagnose issues more quickly
- Gain understanding of a system, and potential problem areas
- **Proactive vs. Reactive** (fix problems before they are problems)

Observability

Allow information to be extracted from our system



Pull vs. Push

- Push
  - Each app opens a connection to the metrics server
  - Application is responsible for regularly sending metrics
  - Works well when app is behind a firewall
- Pull
  - Each app includes an HTTP server for metrics
  - Metrics server opens connection and downloads metrics
  - Configuration is centralized
  - Automatic up/down detection

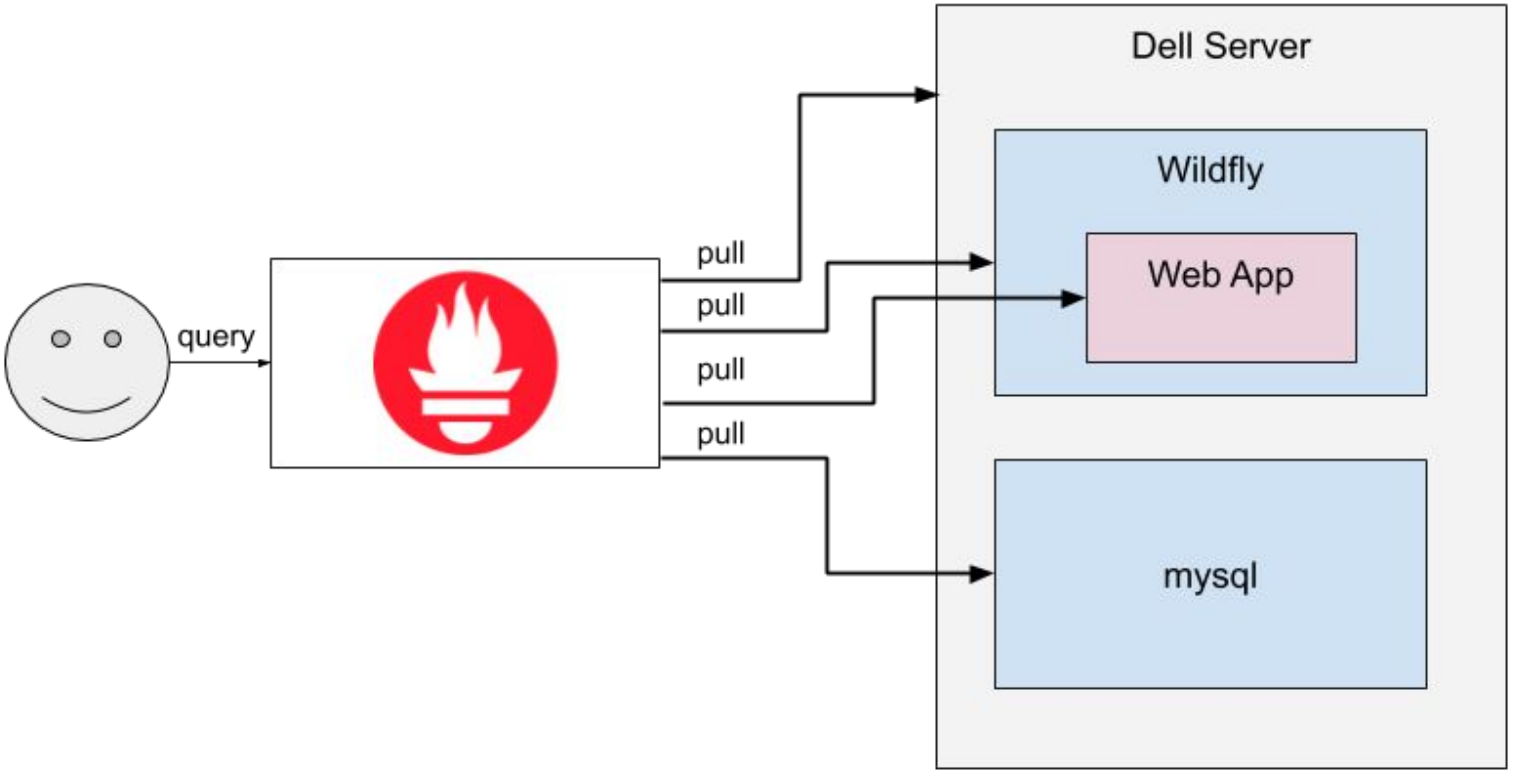
Choosing a Monitoring System

Amazon CloudWatch

Why Prometheus?

- Open Source
- Easy to Get Started
- Powerful Query Language
- Many Integrations
- CNCF - Second graduated project after Kubernetes
- Standard Metric Format (openmetrics)

Prometheus - 1000ft view



How to Use Prometheus

1. Download Prometheus Binary
2. `$./prometheus --config.file=prometheus.yml`
3. ????
4. Profit!

prometheus.yml

```
global:
  scrape_interval:  15s # Set the scrape interval to every 15 seconds.
                        # Default is every 1 minute.

# A scrape configuration containing exactly one endpoint to scrape: Promtheus itself
scrape_configs:

  # The job name is added as a label `job=<job_name>` to any
  # timeseries scraped from this config.

  - job_name: 'prometheus-itself'

     # metrics_path defaults to '/metrics'
     # scheme defaults to 'http'.

     static_configs:
     - targets: ['localhost:9090']
```

Kubernetes Service Discovery

```
# Example scrape config for pods
- job_name: 'kubernetes-pods'

  kubernetes_sd_configs:
  - role: pod

  relabel_configs:
  # Example relabel to scrape only pods that have
  # "example.io/should_be_scraped = true" annotation.
  - source_labels: [__meta_kubernetes_pod_annotation_example_io_should_be_scraped]
    action: keep
    regex: true
```

Demo - Installing Prometheus

Metrics

- Metrics
  - Measurement of something at a specific time
  - Thing, Value/Measurement, and a Time
  - Disk A, 80GB, Sept. 1 at 3:42PM
- Metric Types
  - Counter - Total HTTP Requests
  - Gauge - Current number of users
  - Histogram - Number of HTTP Requests that took less than 10ms, 100ms, 1000ms
  - Summary - Metrics organized by quantiles (0.50, 0.99, etc)
- Values organized by time series

Time Series Database

Thing	10:00 AM	10:15 AM	10:30 AM	10:45 AM
Disk A, GB Used	70.0	75.0	80.0	85.0
Current Users	98	121	115	107
Total HTTP Requests	100,103	100,152	100,211	100,275

- Look for patterns, Make estimates about future
- E.g. if disk usage is going up by 5GB every 15 min, we can estimate when we'll run out

Where does Prometheus Store Metric Data?

- `/data` directory
  - Files organized by time chunks
- For scalable long term storage see Thanos Project
  - Combines data from multiple prometheus instances
  - Downsamples data to reduce size requirements
  - Uses object storage such as amazon S3

Summary of Observability and Prometheus

- Observability
  - Allows more proactive approach to system administration
- Prometheus
  - Pulls data from targets
  - Each target is an HTTP server serving a plain text file
  - Stores metrics in a time series database
  - Allows querying through PromQL
  - Many applications already provide prometheus metrics
- How do we get more metrics?
  - Use existing application metrics
  - Exporters
  - Prometheus client libraries

Application Metrics

Requirements:

1. HTTP server serving plain text file (usually on /metrics)

```
# HELP hits_total Number of HTTP requests received.
# TYPE hits_total counter
hits_total{name=”myapp”} 26.0
```

Prometheus Metrics (Wildfly 18)

```
# HELP base_memory_committedNonHeap_bytes Displays the amount of memory that is committed for the Java virtual machine to use.
# TYPE base_memory_committedNonHeap_bytes gauge
base_memory_committedNonHeap_bytes 9.8828288E7

# HELP base_memory_maxHeap_bytes Displays the maximum amount of memory in bytes that can be used for memory management.
# TYPE base_memory_maxHeap_bytes gauge
base_memory_maxHeap_bytes 5.36870912E8

# HELP base_gc_time_total Displays the approximate accumulated collection elapsed time in milliseconds. This attribute displays -1 if the collection elapsed time is undefined for this collector. The Java virtual machine implementation may use a high resolution timer to measure the elapsed time. This attribute may display the same value even if the collection count has been incremented if the collection elapsed time is very short.
# TYPE base_gc_time_total counter
base_gc_time_total_seconds{name="G1 Young Generation1"} 0.11

# HELP base_gc_total Displays the total number of collections that have occurred. This attribute lists -1 if the collection count is undefined for this collector.
# TYPE base_gc_total counter
base_gc_total{name="G1 Young Generation1"} 16.0
```

Wildfly Metrics Web App

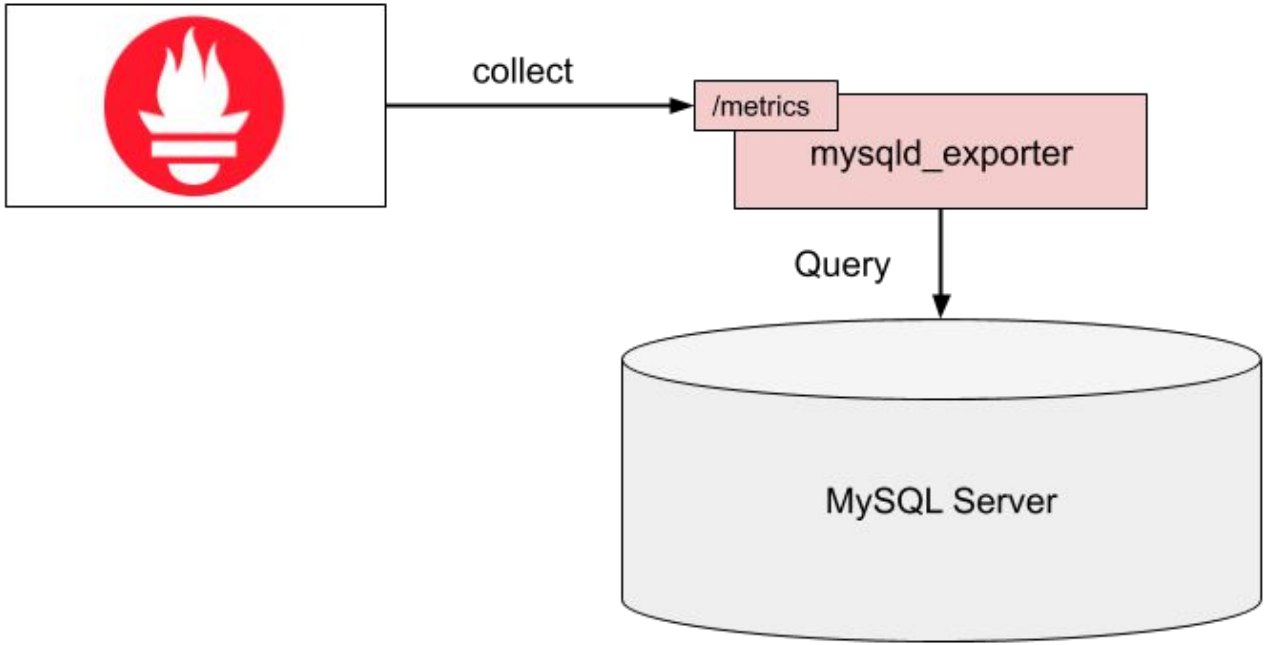
```
@WebServlet("/metrics")
public class MetricsServlet extends HttpServlet {

   private int hits;

   @Override
   protected void service(HttpServletRequest req, HttpServletResponse resp) throws IOException {
       hits++;
       PrintWriter out = resp.getWriter();
       resp.setContentType("text/plain");
       out.println("# HELP wildfly_metrics_hits_total The total number of requests received.");
       out.println("# TYPE wildfly_metrics_hits_total counter");
       out.printf("wildfly_metrics_hits_total %f\n", (double) hits);
   }
}
```

Demo - Wildfly Metrics

Prometheus Exporters

- Pre-made package for gathering prometheus metrics
- Available for monitoring many types of systems
  - Physical Servers - Node exporter
  - Databases - MySQL exporter
  - Messaging Systems - Kafka exporter
  - HTTP - Apache exporter
  - Java Applications - JMX exporter
  - Lot’s more (https://prometheus.io/docs/instrumenting/exporters/)

MySQL Exporter



JMX Exporter

- Java Management Extensions
  - JSR 003 - Approved in 1998
- Provides many JVM metrics
- Can be used for application metrics

Run as Java Agent

```
java -javaagent:./jmx_prometheus_javaagent.jar=8081:jmx_config.yml \
     -jar myapp.jar
```

JMX MBean

```
package com.example.jmx;

public interface HitCounterMBean {
   public int getHits();
   public void reset();
}
```

```
package com.example.jmx;

public class HitCounter implements HitCounterMBean {
   private int hits = 0;
   public int getHits() { return hits; }
   public void increment() { hits++; }
   public void reset() { hits = 0; }
}
```

Register JMX MBean

```
HitCounter hits = new HitCounter();
MBeanServer server = ManagementFactory.getPlatformMBeanServer();
ObjectName hitsObjectName1 =
       new ObjectName("com.example.jmx:type=counter,name=hits,app=jmx");
server.registerMBean(hits, hitsObjectName1);
```

Demo - JMX Exporter

Prometheus Client Libraries

- APIs for directly instrumenting applications
- Officially supported: Go, Java, Python, Ruby
- Unofficially supported: C++, Elixir, Node.js, Perl, and others
- https://prometheus.io/docs/instrumenting/clientlibs/

Prometheus client_java

- https://github.com/prometheus/client_java
- Maven dependencies for various components
- Metric Types: Counter, Gauge, Summary and Histogram
- Simple API to create metrics and make them available

Maven dependencies

```
<!-- The client -->
<dependency>
  <groupId>io.prometheus</groupId>
  <artifactId>simpleclient</artifactId>
  <version>0.6.0</version>
</dependency>
<!-- Hotspot JVM metrics-->
<dependency>
  <groupId>io.prometheus</groupId>
  <artifactId>simpleclient_hotspot</artifactId>
  <version>0.6.0</version>
</dependency>
<!-- Exposition HTTPServer-->
<dependency>
  <groupId>io.prometheus</groupId>
  <artifactId>simpleclient_httpserver</artifactId>
  <version>0.6.0</version>
</dependency>
```

Example Sources - Counter

```
import io.prometheus.client.Counter;
class YourClass {
  static final Counter requests = Counter.build()
     .name("requests_total").help("Total requests.").register();

  void processRequest() {
    requests.inc();
    // Your code here.
  }
}
```

Example Sources - Guage

```
class YourClass {
  static final Gauge inprogressRequests = Gauge.build()
     .name("inprogress_requests").help("Inprogress requests.").register();

  void processRequest() {
    inprogressRequests.inc();
    // Your code here.
    inprogressRequests.dec();
  }
}
```

Demo - Thorntail

Example - Histogram

```
class YourClass {
  static final Histogram requestLatency = Histogram.build()
     .name("requests_latency_seconds").help("Request latency in seconds.").register();

  void processRequest(Request req) {
    Histogram.Timer requestTimer = requestLatency.startTimer();
    try {
      // Your code here.
    } finally {
      requestTimer.observeDuration();
    }
  }
}
```

Example - Summary

```
class YourClass {
  static final Summary receivedBytes = Summary.build()
     .name("requests_size_bytes").help("Request size in bytes.").register();
  static final Summary requestLatency = Summary.build()
     .name("requests_latency_seconds").help("Request latency in seconds.").register();

  void processRequest(Request req) {
    Summary.Timer requestTimer = requestLatency.startTimer();
    try {
      // Your code here.
    } finally {
      receivedBytes.observe(req.size());
      requestTimer.observeDuration();
    }
  }
}
```

Histogram vs. Summary

Two rules of thumb:

1. If you need to aggregate, choose histograms.
2. Otherwise, choose a histogram if you have an idea of the range and distribution of values that will be observed. Choose a summary if you need an accurate quantile, no matter what the range and distribution of the values is.

For more information see documentation:

https://prometheus.io/docs/practices/histograms/

Quarkus

- Supersonic Subatomic Java
- Designed for Container Based Microservices
- Fast development iterations
- Native compilation
  - Fast startup times
  - Low memory footprint
  - JVM not required

Demo - Quarkus

What should I monitor?

- RED Method
  - https://www.weave.works/blog/the-red-method-key-metrics-for-microservices-architecture/
  - Rate - number of requests
  - Errors - how often are things failing
  - Duration - how long are requests taking
- Best practices provided on Prometheus web site
  - https://prometheus.io/docs/practices/instrumentation/

More Observability

- Metrics - Prometheus
- Alerting - Prometheus
- Logging - ELK/EFK
- Tracing - Jaeger

Also Check Out

- Kiali - observability for Istio microservices
- Micrometer - vendor neutral instrumentation
- Thanos - scalable long term storage for Prometheus

Summary

- Make your systems observable
- Metric collection is easy with Prometheus
- Use existing metrics where available
- Use exporters for common packages/frameworks
  - JMX exporter for gathering JVM metrics
- Instrument your applications using client libraries
  - Best practices provided on Prometheus web site
    https://prometheus.io/docs/practices/instrumentation/
- Good luck!

Thank You!