Taking Control of Java Runtime at Enterprise Scale

GS.com/Engineering
April, 2018
Jessica Man
Who Am I?

Jessica Man

- 2001 – 2010
  - IBM Hursley Laboratory
- 2010 – 2016
  - Goldman Sachs Java Engineering
- 2016 –
  - Goldman Sachs Enterprise Technology Operations Engineering
- 2018 –
  - An alternate representative from GS on JCP EC
What is this talk about?

Consider an organization that has more than a million Java processes running on any given day, add the increasing pressure on security when any old library can be an attack vector, throw in the need to deprecate old versions of the platform, on top of capacity management, production hosts access prevention etc.. How can one centrally monitor the use of software to find out who is running what and where? How can it also seamlessly collect JMX metrics, with about 16 million data points per day? How can it enabled without effort from individual application teams?
Can’t we just use existing JMX tools?

• Fact: many apps simply have
  
  -Dcom.sun.management.jmxremote.authenticate=false
  -Dcom.sun.management.jmxremote.ssl=false

• Connect jConsole to a production running process? Forget it

• Performance impact of streaming JMX metrics remotely can be significant

• Doesn’t give you an aggregated view of Java processes that, say, belong to the same application

• We need data extracted from loaded classes (more later)
What about JFR and JMC?

• Flight Recorder and Mission Control give us more than JMX
  • Used widely in GS, different use cases

• Cannot aggregate multiple processes belong to the same application family on a single dashboard

• Integration with other tools to enrich the data or to configure the flight recorder can be challenging
  • But something we will look into

• We need data extracted from loaded classes (more later)
Requirements

- Continuous monitoring on application and JVM metrics
- Doesn’t require logon to production hosts
- Configuration decoupled from software code
- Linkage between processes to application owners
- Linkage between processes and libraries/components used
- Visualization on individual processes and application aggregated views
- Minimal impact to Java processes
- Centrally managed – no effort from app owners on deployment, upgrades and version control
- Extendable for future monitoring agents
- Scalable to support millions of processes
- And more…
The Solution – JRM (Java Runtime Monitor)!

Policy Configuration → Policy Distribution Server → Metrics Dashboard

Policy Configuration → Metrics Dashboard

Policy Distribution Server → OpenTSDB

Policy Distribution Server → Elasticsearch

Policy Distribution Server → Kafka Topics

Policy Distribution Server → JRM Data Consumer

Policy Distribution Server → Router/Load Balancer

Application Host

Application

JRM agents

JRM agent launcher

JRE

Host agent

Config Files

Data Collector/Forwarder
JRM – Data Collection

- In-process agent
  - `java.lang.instrumentation`
  - `java.lang.management`

- JMX metrics
  - An in-process agent polling from the MBean Server

- Beaconing data
  - Process: system properties, JVM arguments, code source path
  - Product: name and version data appended to classes during build time

```json
"brandUser": "jessica",
"brandHost": "host-12345.gs.com",
"brandTime":1409757033913,
"productIdent": "buildutility",
"productVersion":"1.0",
"classes":["com/gs/devteam/build/Utility"]
```
JRM – Data Consumer

- Router forwards the data based on type
- Multiple Kafka topics
  - 1 topic for time-series data
  - 1 topic for text based beaconsing data
  - 3 partitions for each topic
- 6 Kafka consumers
  - 1 consumer for each partition
- Consumers send metrics to time-series store and beaconsing data to Elasticsearch
JRM – Data Storage

- All data is sent in JSON format
- OpenTSDB – JMX metrics
  - 2 billion metrics received every day
  - 24k metrics/second
- Elasticsearch – Beaconing Data
  - 2 million records received per day
  - Record size varies – roughly 50GB per day
JRM – Visualization

- Grafana – JMX metrics
- Kibana + custom built dashboard
JRM – Configuration

- Decouple configuration from code
  - Separate release cycles
  - End users can provide custom configurations
  - Rapid way to disable/enable

- Configuration is pushed to dedicated hosts
  - Stored as a local file

- Integrates with the Enterprise standard tools for policy definition
  - Form based interface
Alerting and Dashboards

- Example derived metrics
  - Heap released over heap allocation
  - Accumulated GC counts
  - Accumulated GC time

- Example dashboards
  - Heap used vs max size
  - GC frequency
  - GC pause times
  - Heap allocation vs heap released

- Alert on combination of metrics
  - e.g. not on heap usage alone
Requirements

- Continuous monitoring on application and JVM metrics
- Doesn’t require logon to production hosts
- Configuration decoupled from software code
- Linkage between processes to application owners
- Linkage between x and libraries/components used
- Visualization on individual processes and application aggregated views
- Minimal impact to Java processes
- Centrally managed – no effort from app owners on deployment, upgrades and version control
- Extendable for future monitoring agents
- Scalable to support millions of processes
- And more…
Future

- Apply machine learning on gathered metrics
  - Anomaly detection

- Provide easy way to generate product usage reports
  - Simple button click for commonly used queries

- Beaconing Phase II
  - Enable product beacon data injection for all internal builds

- Integration with Flight Recorder/Mission Control

- Workshops/Blog Posts/Tutorial Videos
  - Show people how the data gather can solve their production products or improve their applications
Useful Links

- `java.lang.instrument`
- `java.lang.management`
- `OpenTSDB`
- `Elasticsearch`
- `Kafka`
- `Kibana`
Visit us online at:
goldmansachs.com/careers
GREAT INDIAN DEVELOPER SUMMIT 2019
Conference: April 23-26, Bangalore

Register early and get the best discounts!

www.developersummit.com @greatindiandev bit.ly/gidslinkedin