Monitoring Kafka

Lesson Objectives

- Learn how to monitor Kafka

Monitoring

Objectives

- Learn monitoring tool
- And best practices

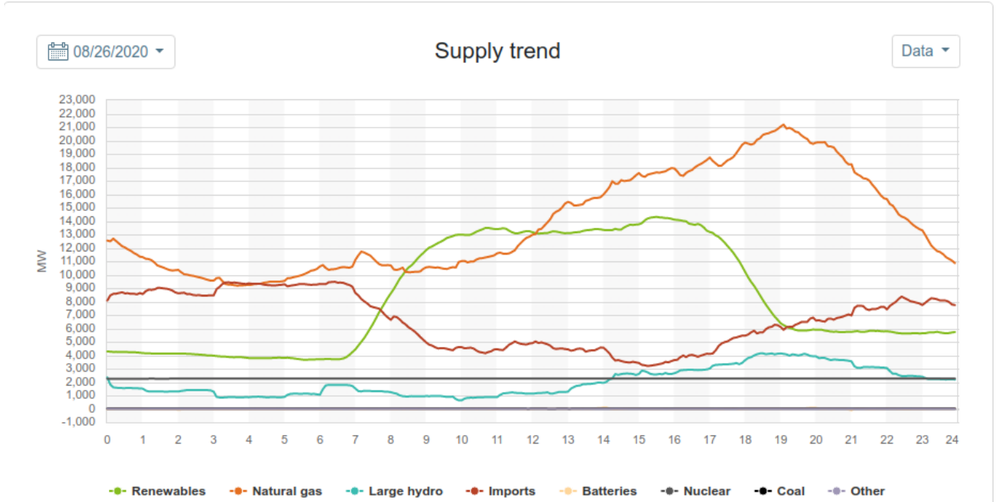
Why Monitoring

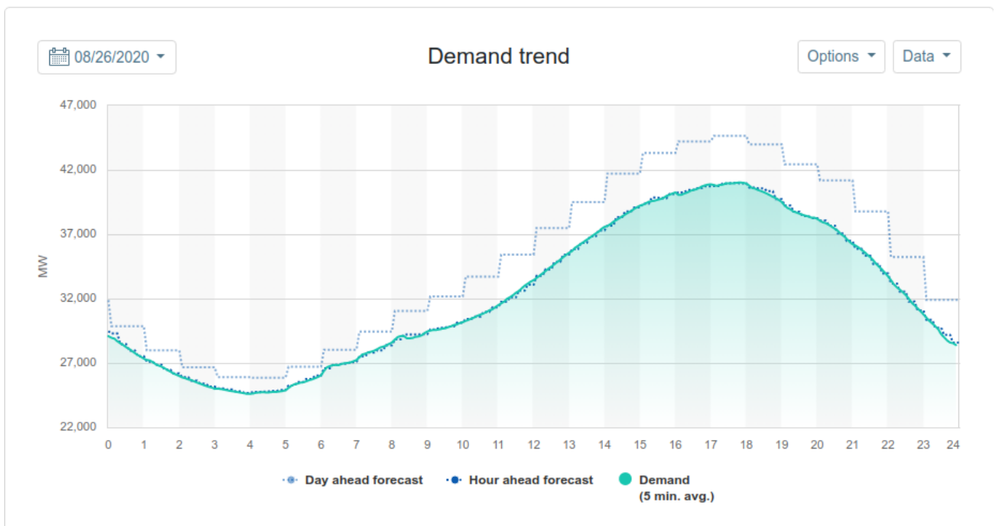
Monitoring

- Monitoring at SpaceX

Monitoring

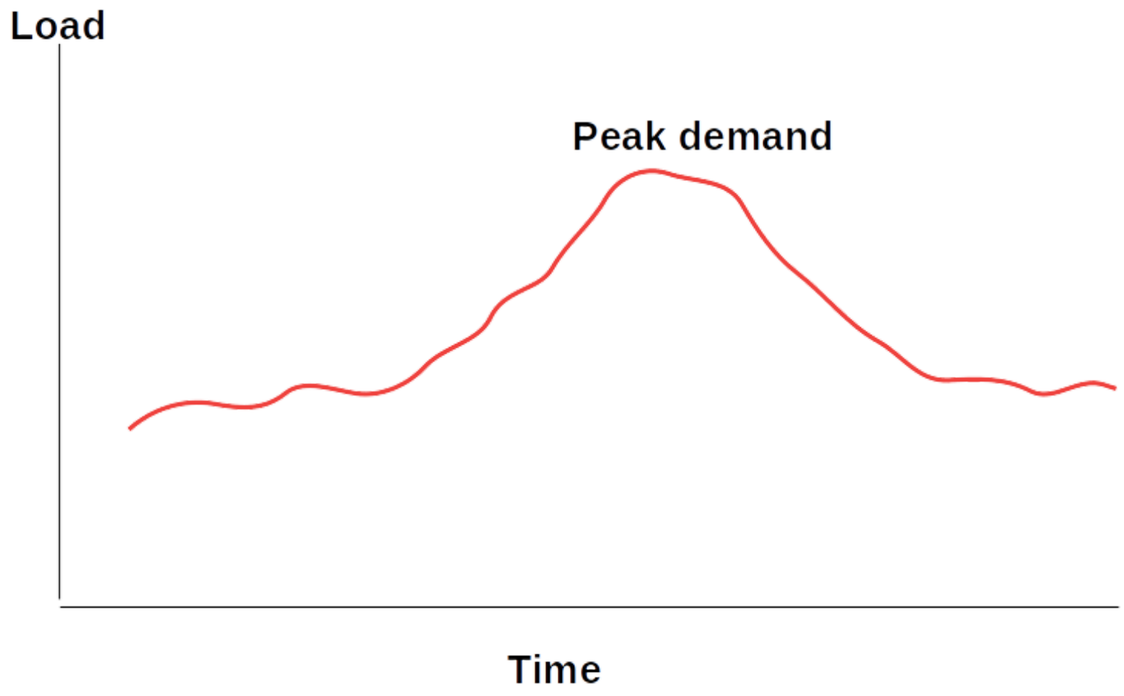
- California ISO (Independent System Operator) - that monitors California electrical grid





Why Monitoring?

- Monitoring helps to keep an eye on sytems and applications
- Helps us identify problem spots before they actually become problems
- Helps us spot trends and patterns

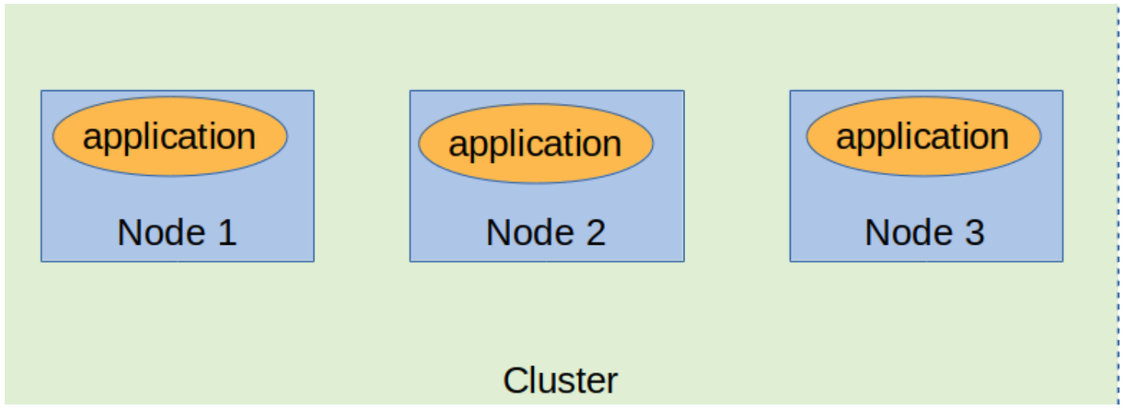


Monitoring Best Practices

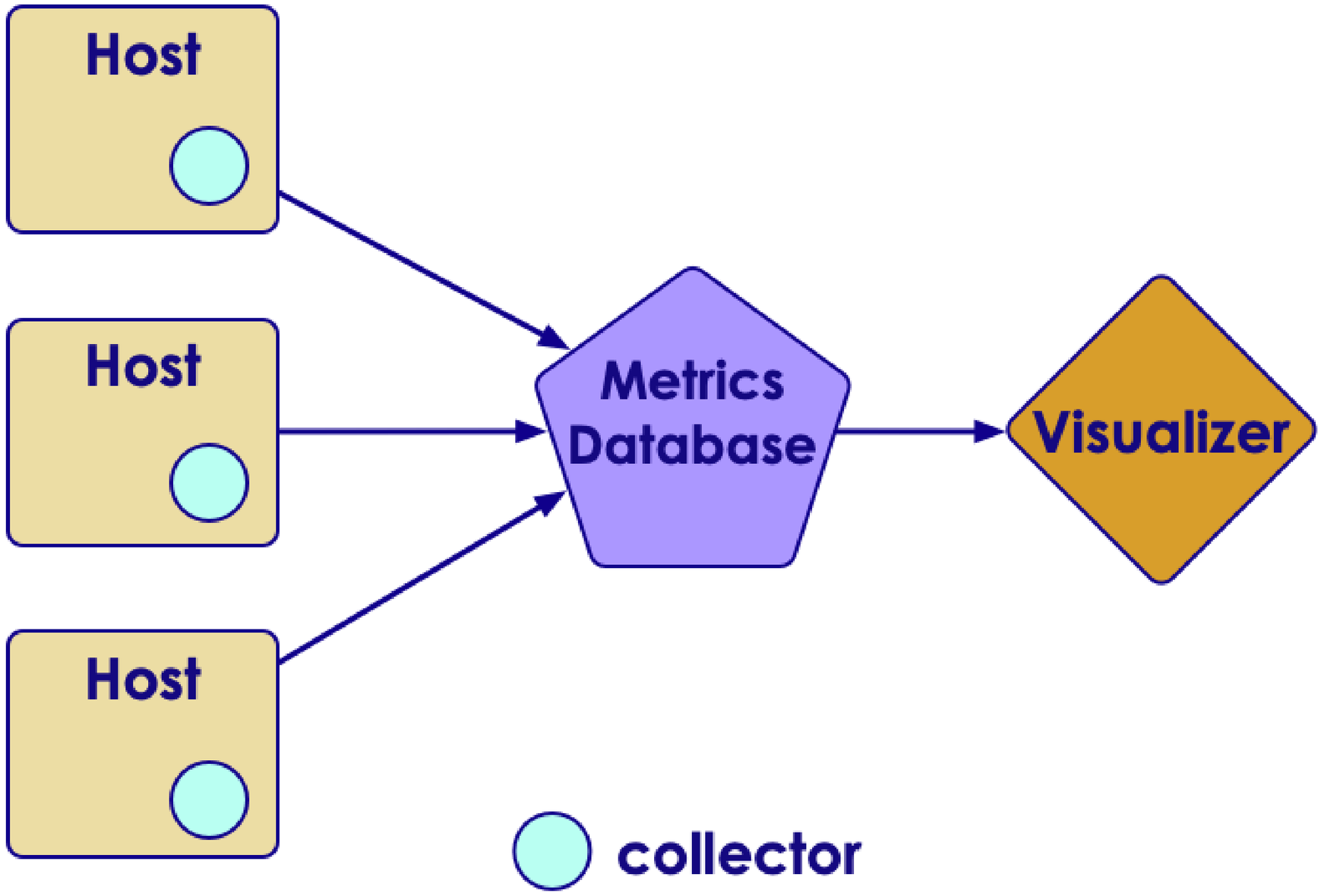
- Some monitoring is better than no monitoring
- Implement **actionable monitoring**
  Without any action, monitoring is just 'pretty graphs'. For example, when a problem is detected, an alert should be generated
- **Automate** as much as possible
  Monitoring can generate a lot of data. Going through all the data manually can be tedious. We want to implement tools to cut through the data and spot patterns.
- Use **good monitoring tools**
  These tools provide lot of automation and implement best practices

What to Monitor?

- We want to monitor the following:
  - Cluster, individual nodes and applications
- **Cluster:** Monitor overall cluster status
  - Overall utilization (e.g. 60%)
- **Individual Nodes:** Monitor each machine to identify issues
  - CPU, memory, disk, bandwidth
- **Applications:** Monitor user applications
  - latencies, requests per second ..etc.



Monitoring Architecture



Monitoring Architecture

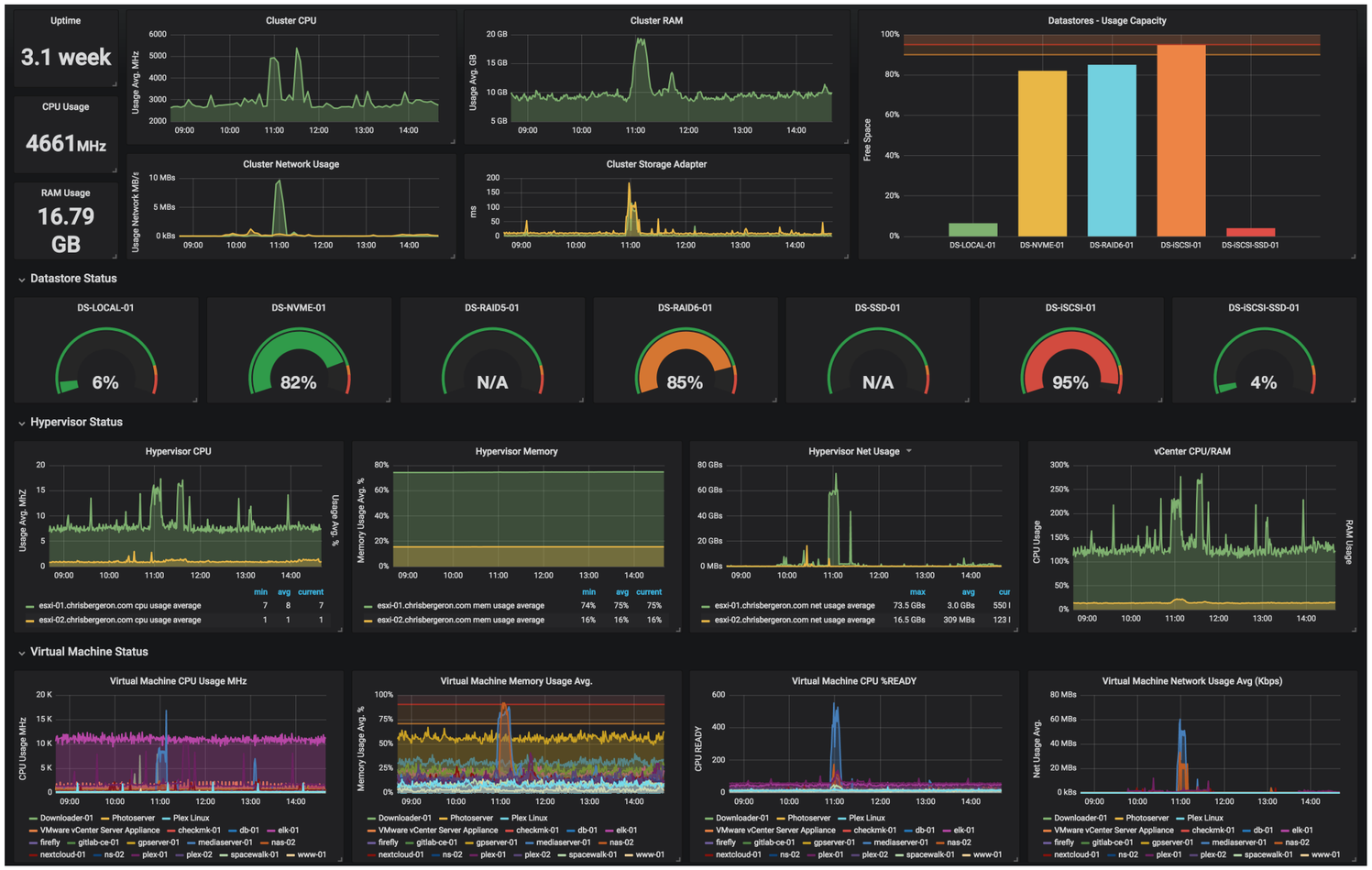
- 3 main components: collector, database, visualizer
- **Collector/agent**
  - Collects metrics from the host and pushes to database
- **Database**
  - Collects and stores metrics from various sources
  - Performs aggregations (current rate / last_1m rate / min / max)
  - A time-series database
- **Visualizer**
  - Create nice looking visualizations of metrics
  - Various graphs

Monitoring Database Choices

	Prometheus	Graphite	InfluxDB	OpenTSDB
Ease of use	Easy	Easy	Easy	Considerable effort required
Scale	Small / medium	Small/medium	large	Massively scalable
License	Open source	Open source	Open source + premium	Open source
Website	https://prometheus.io/	https://graphiteapp.org/	https://github.com/influxdata/influxdb	http://opentsdb.net/

Visualization - Grafana

- Modern, Open source
- Very attractive graphs
- Easy to setup and use
- Supports multiple databases: Graphite / Influx / OpenTSDB
- Grafana.com



Cluster Monitoring Tools

Cluster Monitoring Tools

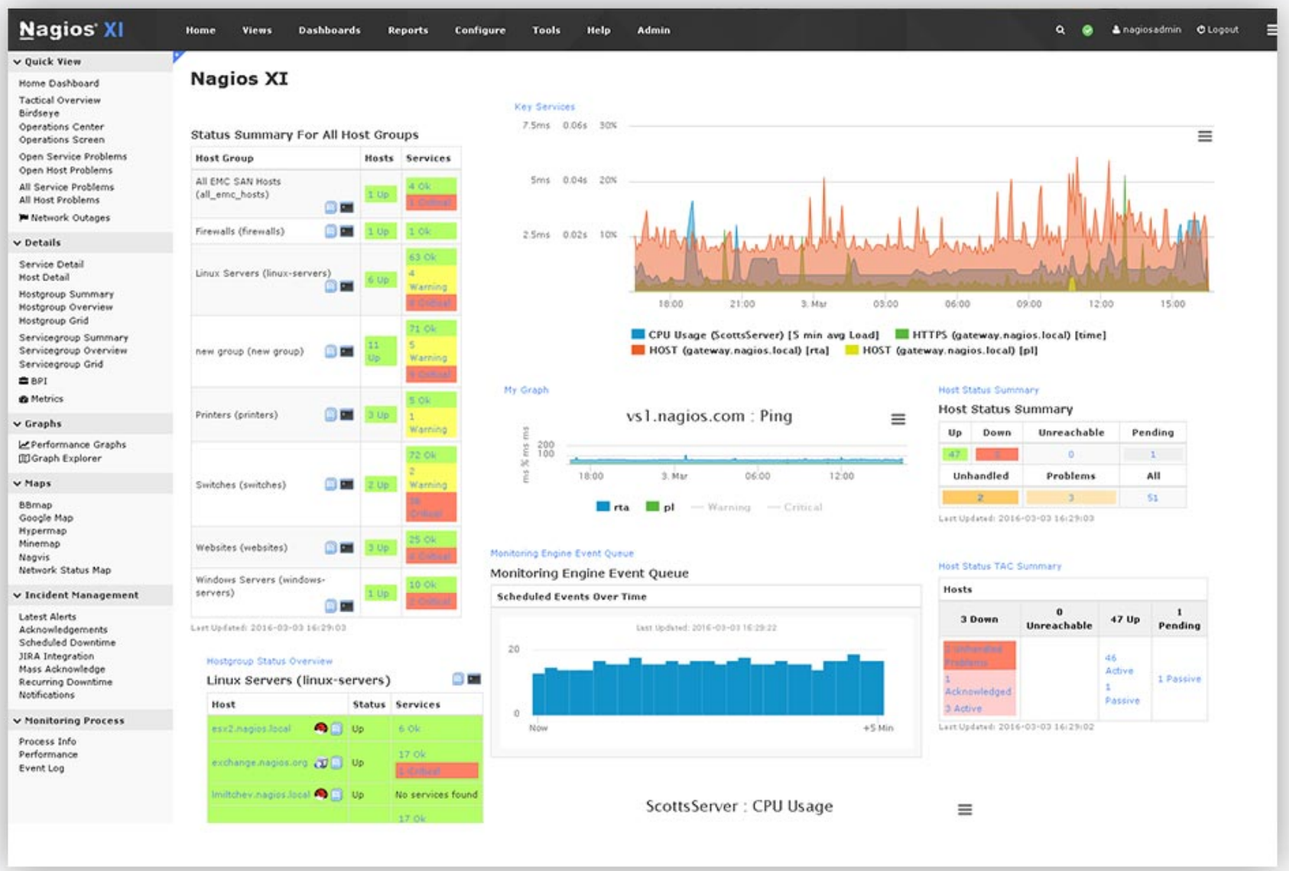
- There are tons of tools availble; most are open source and very capable.
  - Choose one that works with your environment
- Prometheus
- Riemann
- Sensu
- Zabbix
- Icninga
- Nagios
- Cacti
- M/Monit
- LibreNMS
- References
  - 1, 2, 3

Cluster Monitoring Tools

- **Instructor:** The following slides describe the tools in details. They are provided as reference. Cover as necessary.

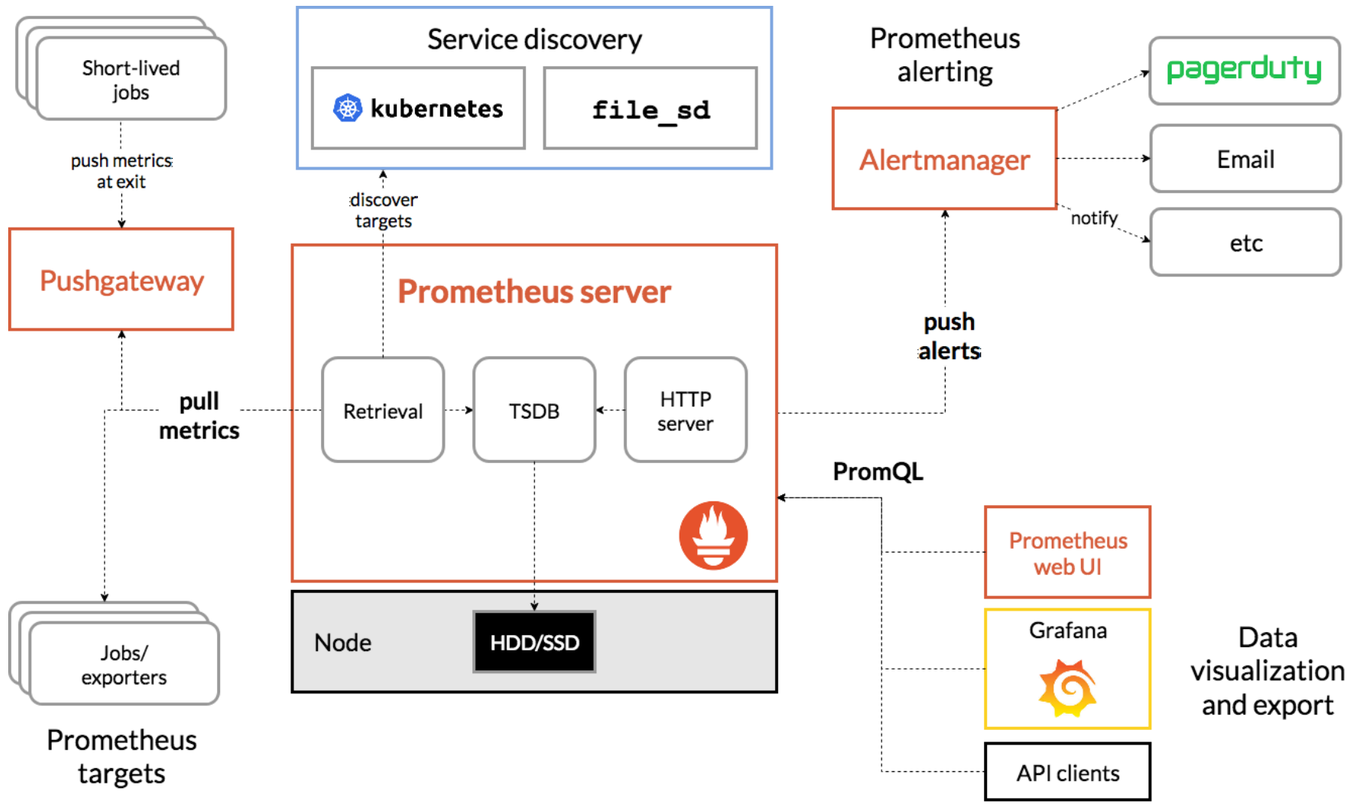
Nagios

- Scalable
- Very well field tested (since 1999!)
- Host level and application level monitoring
- Large plugin library
- https://www.nagios.org



Prometheus

- Open source and very popular
- **Prometheus** database is an excellent time series db
    - PromQL provides easy querying
- Works really well Kubenetes and container environments
- Built in Alertmanager helps you manage alerts
- Prometheus website
- References
    - Prometheus Monitoring : The Definitive Guide



System Monitoring Tools

Host/System Monitoring Tools

- All the tools mentioned above will monitor overall cluster
- We can also drill down into individual hosts as well
- Example metrics:
  - CPU / Memory / Disk usage
  - Network traffic
- Sometimes we need more detailed information than provided by the tools
- Here are some tools to help with that

Linux System Monitoring Tools

- System load
  - Top and vaiants
  - Atop
  - htop
  - glances
- System IO stats
  - vmstat
  - iostat
  - lsof
- Network
  - Tcp dump
  - Netstat
- References:
  - 4 open source tools for Linux system monitoring
  - 20 Command Line Tools to Monitor Linux Performance

TOP / ATOP / HTOP / GLANCES

- These will give you a snapshot of what is running on your machine

```
top - 20:38:40 up 2 days,  9:36,  1 user,  load average: 1.43, 1.32, 0.93
Tasks: 629 total,   1 running, 499 sleeping,   0 stopped,   1 zombie
%Cpu(s):  4.4 us,  1.8 sy,  0.0 ni, 92.7 id,  0.0 wa,  0.0 hi,  1.2 si,  0.0 st
KiB Mem : 32801244 total,   343168 free, 13676116 used, 18781960 buff/cache
KiB Swap:  2097148 total, 2042876 free,    54272 used. 17503552 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 2827 sujee     20   0 5291172 655224  93468 S  20.7  2.0  75:37.08 gnome-shell
 3874 sujee     20   0 5153240 301828 121740 S  14.1  0.9   1:18.04 chrome
 3590 sujee     20   0 1809116 456392 149020 S  13.2  1.4  54:17.94 chrome
 3553 sujee     20   0 2200488 668044 220260 S   9.5  2.0 137:17.98 chrome
 1471 sujee     20   0 5024496 201288 106676 S   6.9  0.6   4:49.23 chrome
 1480 root     -51   0       0      0      0 S   6.6  0.0  23:56.49 irq/131-nvidia
 4926 sujee     20   0 7000628 736400 126188 S   5.6  2.2 255:17.00 zoom
```

TOP

```
PRC | sys    0.23s | user   0.35s | #proc    630 | #tslpu     0 | #zombie    1 | #exit      0 |
CPU | sys      48% | user     53% | irq       32% | idle    1467% | wait       0% | curscal  20% |
CPL | avg1    0.87 | avg5    1.18 | avg15    0.91 | csw     22288 | intr   13824 | numcpu     16 |
MEM | tot    31.3G | free  314.3M | cache    4.8G | buff     8.8G | slab    4.7G | hptot    0.0M |
SWP | tot     2.0G | free    1.9G |               |                | vmcom  45.8G | vmlim  17.6G |
DSK |          sda | busy      1% | read        0 | write      20 | MBw/s    0.3 | avio 0.80 ms |
NET | transport    | tcpi      6 | tcpo        6 | udpi       17 | udpo      11 | tcpao      0 |
NET | network      | ipi      18 | ipo        17 | ipfrw       0 | deliv     18 | icmpo      0 |
NET | eno1     0% | pcki     11 | pcko        9 | sp   1000 Mbps | si   28 Kbps | so   36 Kbps |
NET | lo    ---- | pcki      8 | pcko        9 | sp      0 Mbps | si    4 Kbps | so    4 Kbps |

  PID CID          SYSCPU  USRCPU  VGROW  RGROW  RUID     ST  EXC  THR S  CPUNR  CPU CMD        1/9
 3874 host-------- 0.04s   0.07s     0K   264K  sujee    --    -   31 S     11   6% chrome
 7592 host-------- 0.06s   0.02s 13820K  6864K  sujee    --    -    1 R      2   5% atop
 2827 host-------- 0.02s   0.04s     0K     0K  sujee    --    -   26 S     10   3% gnome-shell
 2669 host-------- 0.03s   0.03s     0K     0K  sujee    --    -    2 S     12   3% Xorg
```

atop

```
  1 [|||        11.6%]   5 [||         7.0%]   9 [||         7.5%]   13 [|||       11.6%]
  2 [||         9.8%]   6 [||        10.3%]   10 [|||       10.8%]   14 [||         7.0%]
  3 [||         7.5%]   7 [||         6.4%]   11 [|||       14.7%]   15 [|||        9.3%]
  4 [|||        9.5%]   8 [||         8.3%]   12 [||        3.9%]   16 [||         5.1%]
  Mem[||||||||||||||||||||||||||14.2G/31.3G]   Tasks: 332, 2474 thr; 1 running
  Swp[||                          53.0M/2.00G]   Load average: 1.11 1.25 0.93
                                                 Uptime: 2 days, 09:36:49

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 2827 sujee      20   0 5167M  640M 93468 S 21.5  2.0  1h15:43 /usr/bin/gnome-shell
 3874 sujee      20   0 5000M  293M  118M S 13.3  0.9  1:22.82 /opt/google/chrome/chrome --type=rend
 3553 sujee      20   0 2145M  652M  214M S  7.6  2.0  2h17:20 /opt/google/chrome/chrome
 6369 sujee      20   0 8952M  170M 87196 S  7.6  0.5  2:18.69 /opt/google/chrome/chrome --type=rend
 2669 root       20   0  775M  277M 76252 S  6.9  0.9  1h37:50 /usr/lib/xorg/Xorg vt2 -displayfd 3 -
 4926 sujee      20   0 6836M  719M  123M S  5.1  2.2  4h15:18 /opt/zoom/zoom zoommtg://us02web.zoom
```

htop

```
melbourne - IP 192.168.86.21/24 Pub 2601:647:4100:109a:7d4b:f8ba:def9:11e9   Uptime: 2 days, 9:37:36

4.77/3.60GHz   CPU       8.9%  GPU GeForce RTX 2  MEM     46.8%  SWAP      2.6%  LOAD     16-core
CPU  [ 11.9%]  user:     7.7%  proc:          1%   total:  31.3G  total:  2.00G  1 min:     1.24
MEM  [ 46.8%]  system:   1.6%  mem:          21%   used:   14.6G  used:   53.0M  5 min:     1.25
SWAP [  2.6%]  idle:    89.4%                      free:   16.6G  free:   1.95G  15 min:    0.94

NETWORK       Rx/s   Tx/s   CONTAINERS 1 (served
docker0         0b   760b
eno1         21Kb    39Kb    Name                 Status  CPU%    MEM    /MAX   IOR/s   IOW/s   Rx/s
lo          976b    976b    eloquent_wescoff     running  0.0    179M  31.3G      0B      0B     0b /bi
veth4428e95     0b   760b

                             TASKS 617 (3110 thr), 2 run, 499 slp, 116 oth sorted automatically
DISK I/O       R/s    W/s
loop0            0      0   CPU%   MEM%    PID USER      THR  NI S Command
loop1            0      0   20.9   1.8    8679 sujee     27    0 S /opt/google/chrome/chrome --type
loop2            0      0   16.0   2.0    2827 sujee     26    0 S /usr/bin/gnome-shell
loop3            0      0   10.8   1.4    3590 sujee      8    0 S /opt/google/chrome/chrome --type
loop4            0      0
```

glances

IO Stats

- **vmstats** will display memory/disk/thread stats
- **iostats** will display IO stats
- Install using:
  `sudo apt install sysstats`
- References:
  - Linux Performance Monitoring with Vmstat and Iostat Commands

```
(base) sujee@melbourne:~$ vmstat -s
     32801244 K total memory
     14054936 K used memory
     23030404 K active memory
      3666852 K inactive memory
       513964 K free memory
      9201920 K buffer memory
      9030424 K swap cache
      2097148 K total swap
        55296 K used swap
      2041852 K free swap
```

```
(base) sujee@melbourne:~$ iostat -d | grep -v loop
Linux 5.4.0-42-generic (melbourne)      08/31/2020      _x86_64_        (16 CPU)

Device             tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda              10.11        65.78       179.81   13815344   37763312
sdc              15.29        39.88       401.54    8374685   84328564
sdb               0.00         0.14         0.00      28684        164
```

LSOF

- **lsof** displays files opened by processes
- It can be handy when diagnosing file IO errors
- For example, IO intensive apps like Kafka and Spark may run out of file handles; We can use this to see which files are being opened
- References:
  - 10 lsof Command Examples in Linux

```
(base) sujee@melbourne:~$ lsof | grep vscode
code      30010                  sujee  mem       REG                8,5   1735645    3675107 /us
r/share/code/resources/app/node_modules.asar.unpacked/vscode-sqlite3/build/Release/sqlite.node
code      30010                  sujee   68u      unix 0x0000000000000000       0t0    2814925 /ru
n/user/1000/vscode-3719e128-1.48.2-main.sock type=STREAM
ThreadPoo 30010 12225            sujee  mem       REG                8,5   1735645    3675107 /us
r/share/code/resources/app/node_modules.asar.unpacked/vscode-sqlite3/build/Release/sqlite.node
ThreadPoo 30010 12225            sujee   68u      unix 0x0000000000000000       0t0    2814925 /ru
n/user/1000/vscode-3719e128-1.48.2-main.sock type=STREAM
```

Lab: Using System Monitoring Tools

- **Overview:**
  - Learn Linux system monitoring tools
- **Approximate run time:**
  - ~15 mins
- **Instructions:**
  - Try the Linux tools we just learned.
  - Try various options for each tool

Java Monitoring Tools

Java Monitoring Tools

- Java is the language of choice for lot of big data systems (Kafa, Spark, Cassandra)
- So being able to monintor Java apps is important part of diagnosing issues
- Following are some tools of trade:
    - JMX
    - jolokia
    - Jconsole
    - visualVM
    - Java Mission Control - Commercial
    - Java flight recorder - Commercial

JMX

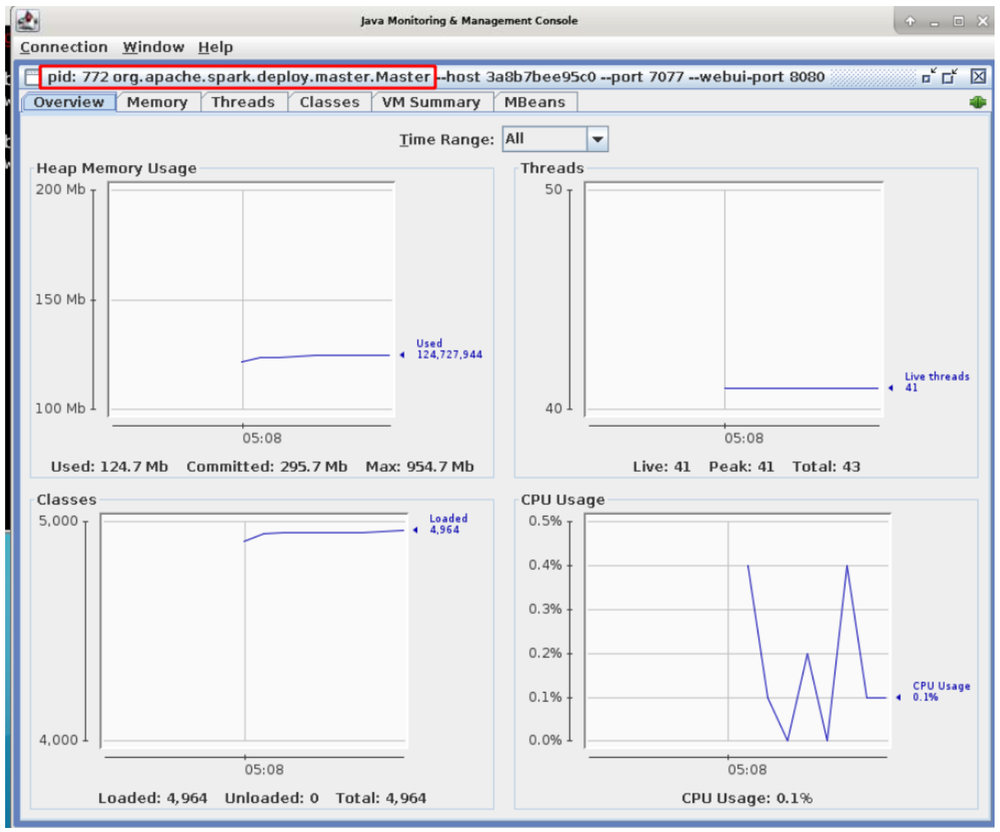
- **JMX** is tools and interfaces for monitoring Java applications
- Lot of apps can export metrics using JMX interface
- These metrics can be collected by apps and displayed
- References:
  - 10 mins Quick Start JMX Tutorial

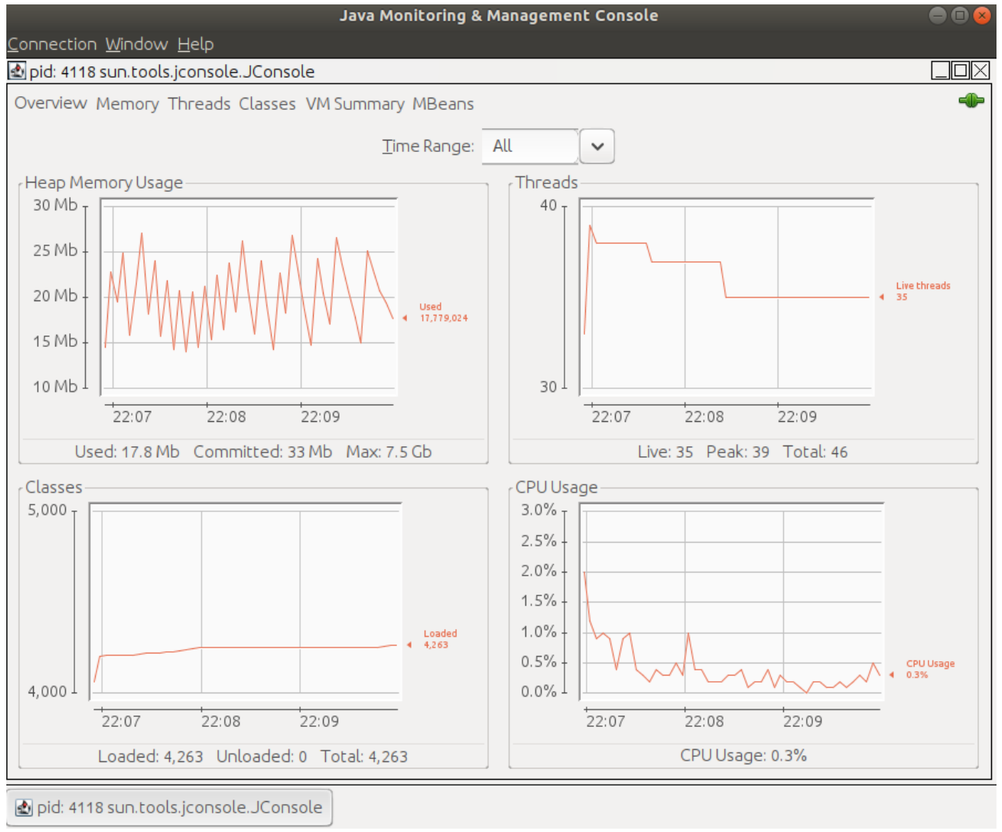
Combined Metrics

Rate	Mean	1 min	5 min	15 min
Messages in /sec	0.02	0.01	0.00	0.00
Bytes in /sec	9.33	5.69	1.64	0.58
Bytes out /sec	0.00	0.00	0.00	0.00
Bytes rejected /sec	0.00	0.00	0.00	0.00
Failed fetch request /sec	0.00	0.00	0.00	0.00
Failed produce request /sec	0.00	0.00	0.00	0.00

JConsole

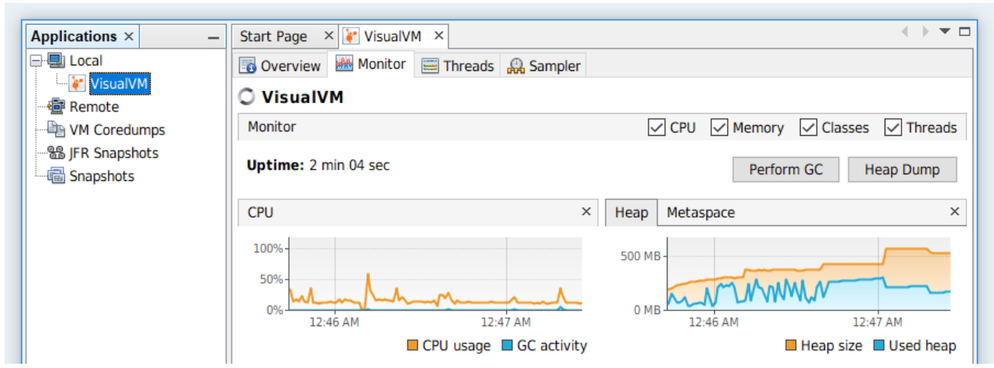
- **JConsole** is a GUI monitoring tool
- Uses JMX to collect metrics
- References:
  - JConsole
  - Jconsole example

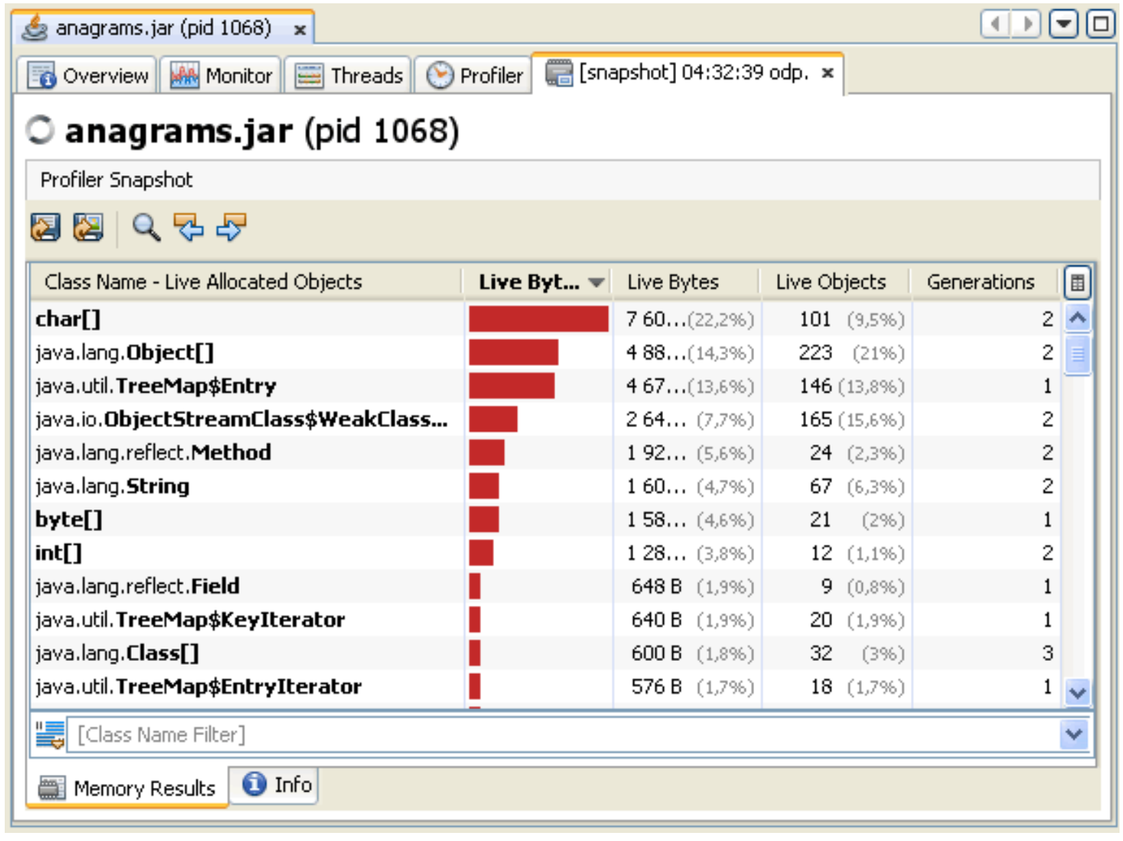




VisualVM

- **VisualVM** is a GUI Java profiler
- Can monitor: Heap memory, threads
- References:
  - VisualVM
  - Visual VM guide
  - Visual VM tutorial





Java Flight Recorder

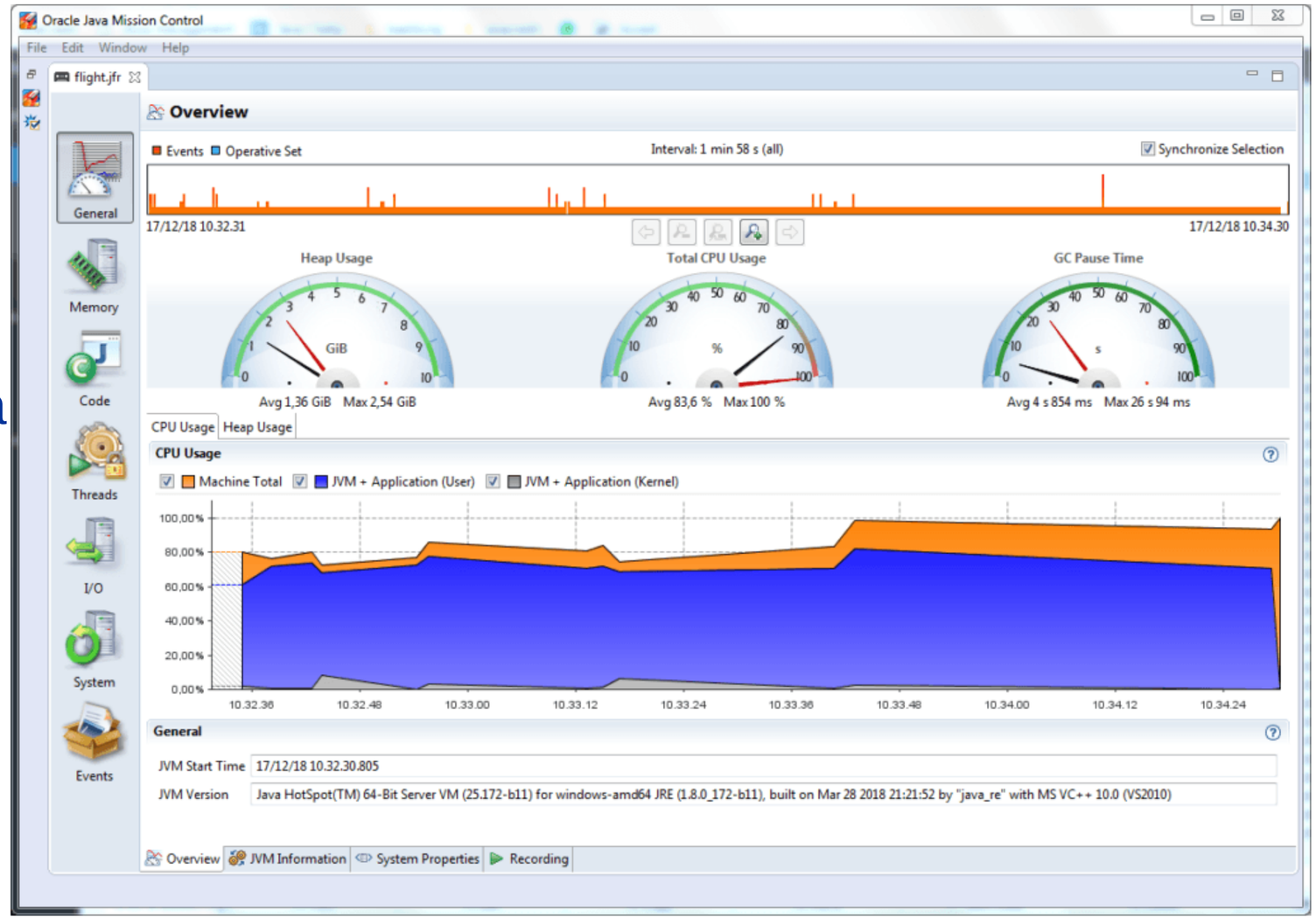
- **Java Flight Recorder (JFR)** is a tool for collecting diagnostic and profiling data of JVM
    - It is integrated into the Java Virtual Machine (JVM) and causes almost no performance overhead, so it can be used even in heavily loaded production environments
- Start as follows

```
java -XX:+UnlockCommercialFeatures -XX:+FlightRecorder
MyApp
```

- JFR will start instrumenting the app and collecting data
- References:
    - Java Flight Recorder
    - Using Java Flight Recorder tutorial

Java Mission Control

- **Java Mission Control** enables monitoring and managing Java applications without introducing the performance overhead
- For example, can collect data from `java flight recorder`
- Start as **`jmc`**
- References:
  - Java Mission Control



Lab: Using Java Monitoring Tools

- **Overview:**
  - Experiment with Java monitoring tools
- **Approximate run time:**
  - ~15 mins
- **Instructions:**
  - Try the JVM tools we just learned.

Application Monitoring Tools

Application Monitoring

- In previous sections we have learned the following:
    - System level monitoring (CPU, Memory, IO)
    - JVM monitoring (threads, heap size)
- Often times, we need to measure **application specific** metrics that we can not gather from the above
- We need to instrument/profile our application code
- For example, let's say we are saving data to a db, and want to measure the time taken.

```
long t1 = mark_time();
result = saveToDB(data);
long t2 = mark_time();
// time taken is : t2 - t1
```

Application Monitoring Best Practices

- Start early! Start as you are developing the application.
    - This encourages good monitoring practices; and spot bottlenecks early on
- Prioritize what to monitor; Profile critical application paths first
- Put in alerts in monitoring system; make sure they work!
- Have a process to monitor alerts. Figure out who is on 'pager duty'
- References:
    - []

Metrics Library

Metrics Library

- **Metrics** is a Java library, that is used to report metrics.Formerly known as **codahale metrics** (authored by Coda Hale)
- Light weight and fast
- Widely used by many projects (Hadoop / Spark / Cassandra)
- Supported various backends: Graphite, Ganglia
- Supported UIs: built-in UI, JMX
- Metrics page

Using Metrics Library

- Metrics is a Java library
- Import the package into project; Here is a fragment in `pom.xml`

```
<dependencies>
  <dependency>
    <groupId>io.dropwizard.metrics</groupId>
    <artifactId>metrics-core</artifactId>
    <version>4.1.2</version>
  </dependency>
</dependencies>
```

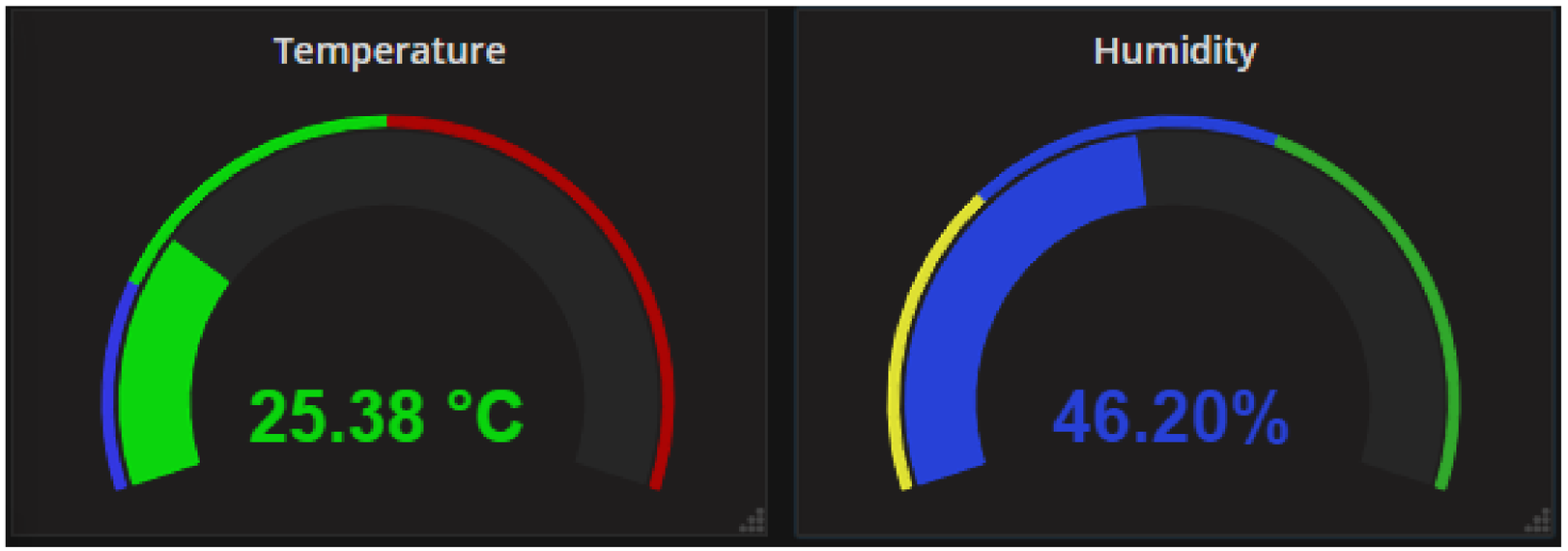
Using Metrics Library

```
import java.util.concurrent.TimeUnit;
import com.codahale.metrics.ConsoleReporter;
import com.codahale.metrics.JmxReporter;
import com.codahale.metrics.MetricFilter;
import com.codahale.metrics.MetricRegistry;
import com.codahale.metrics.graphite.Graphite;
import com.codahale.metrics.graphite.GraphiteReporter;

private final MetricRegistry metrics = new MetricRegistry();

// console reporter
ConsoleReporter consoleReporter =
ConsoleReporter.forRegistry(metrics).convertRatesTo(TimeUnit.SECONDS)
        .convertDurationsTo(TimeUnit.MILLISECONDS).build();
consoleReporter.start(30, TimeUnit.SECONDS);

// graphite
final Graphite graphite = new Graphite(new
InetSocketAddress("localhost", 2003));
final GraphiteReporter graphiteReporter
GraphiteReporter.forRegistry(metrics).prefixedWith("myapp")
  .convertRatesTo(TimeUnit.SECONDS)
  .convertDurationsTo(TimeUnit.MILLISECONDS)
  .filter(MetricFilter.ALL)
  .build(graphite);
graphiteReporter.start(30, TimeUnit.SECONDS);
```

Metrics Library: Meters

- A meter measures the rate of events over time.(e.g., "requests per second").
- In addition to the mean rate, meters also track 1-, 5-, and 15-minute moving averages.

```
private final Meter requests = metrics.meter("requests");

public void handleRequest(Request request, Response response) {
    requests.mark();
}
```



Metrics Library: Counters

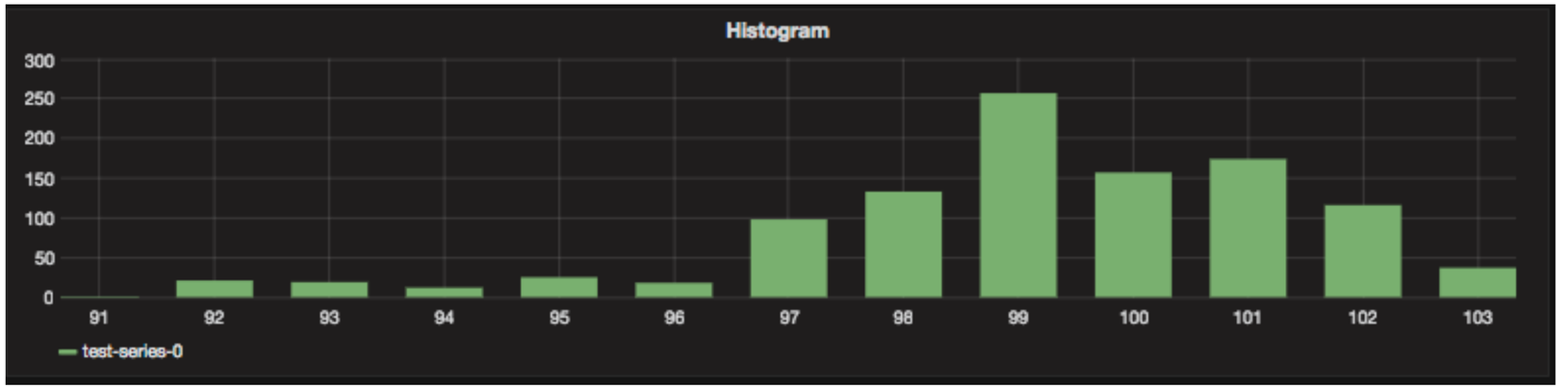
- A counter is used to 'count' things.Number of messages in queue, ...etc
- Counter is an AtomicLongCan be incremented or decremented

```
private final Counter msgCounter = metrics.counter("messages-in
q");

msgCounter.inc();
msgCounter.inc(10);

msgCounter.dec();
msgCounter.dec(5);
```

Metrics Library: Histograms

- A **histogram** measures the statistical distribution of values in a stream of data.
- In addition to minimum, maximum, mean, etc., it also measures median, 75th, 90th, 95th, 98th, 99th, and 99.9th percentiles.

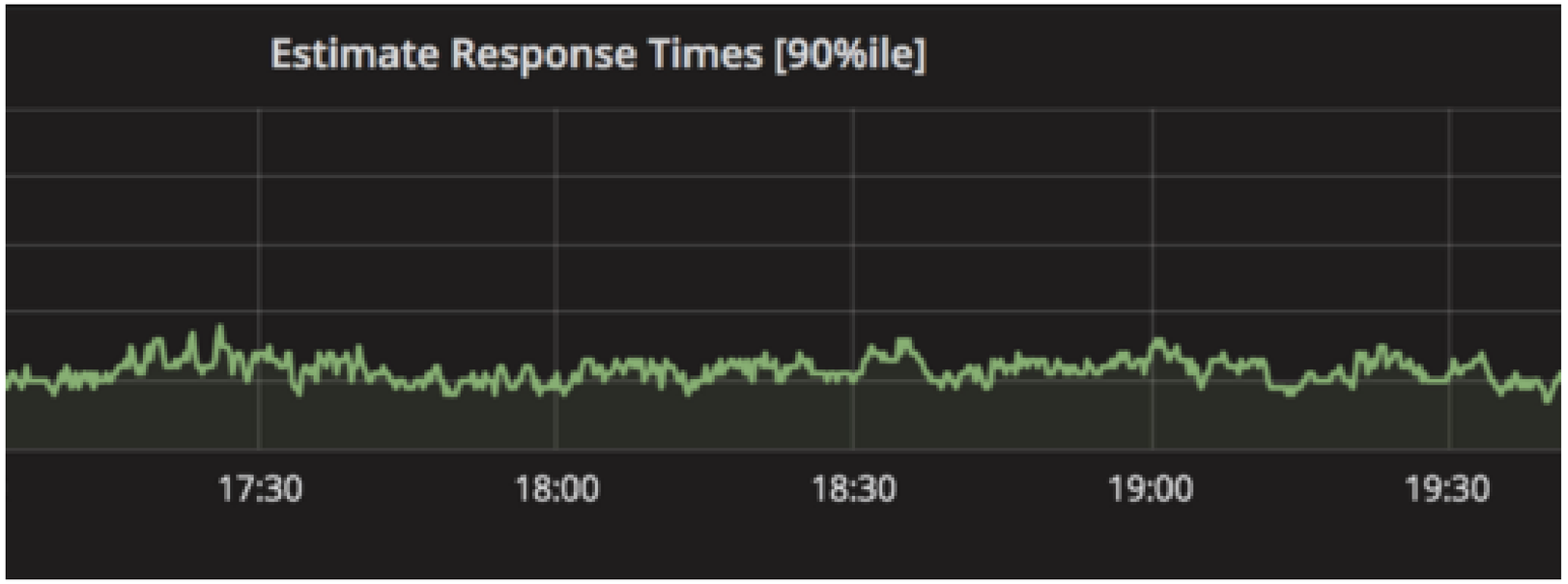
```
private final Histogram msgSizes = metrics.histogram( "message_sizes");

msgSizes.update(100);
msgSizes.update(50);
```



Metrics Library: Timers

- A **timer** measures the duration of piece of code
- Also measures the rate the code is called

```
private final Timer timerExec = metrics.timer("execTime"));

Timer.Context context = timerExec.time();
// do some work here
f();  // calling a function
context.stop();
```



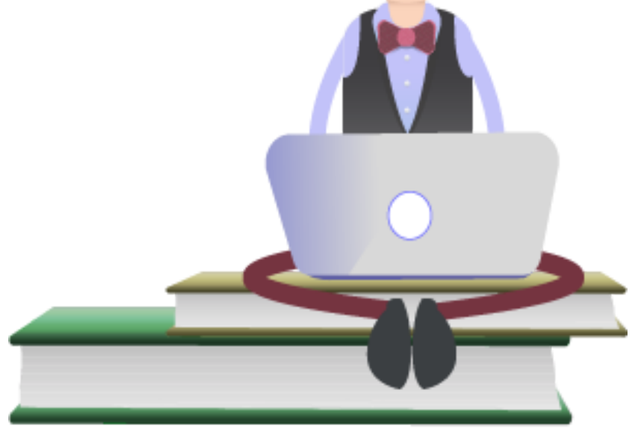
Optional Lab: Metrics Labs

- **Overview:**
  - Learn to use Metrics library
- **Run Time:**
  - ~30 mins
- **Instructions:**
  - Grab the lab from https://github.com/elephantscale/learning-metrics
  - Follow the instructions to get metrics demo running

- **To Instructor:**
  - Demo this if time permits

Optional Lab: Kafka Metrics Labs (Intermediate)

- **Overview:**
  - Instrument Kafka Producer and Consumer
- **Run Time:**
  - ~30 mins
- **Instructions:**
  - **Kafka Metrics (9.2)**
- **To Instructor:**
  - Demo this if time permits



References

- "Site Reliability Engineering" book

Review and Q&A

- Let's go over what we have covered so far
- Any questions?

Monitoring Kafka

Kafka Monitoring Vital Stats

- The following are vital stats to monitor:
  - Log flush latency (95th percentile)
  - Under Replicated Partitions
  - Messages in / sec per broker and per topic
  - Bytes in / sec per broker
  - Bytes in / sec per topicBytes / message
  - End-to-End time for a message

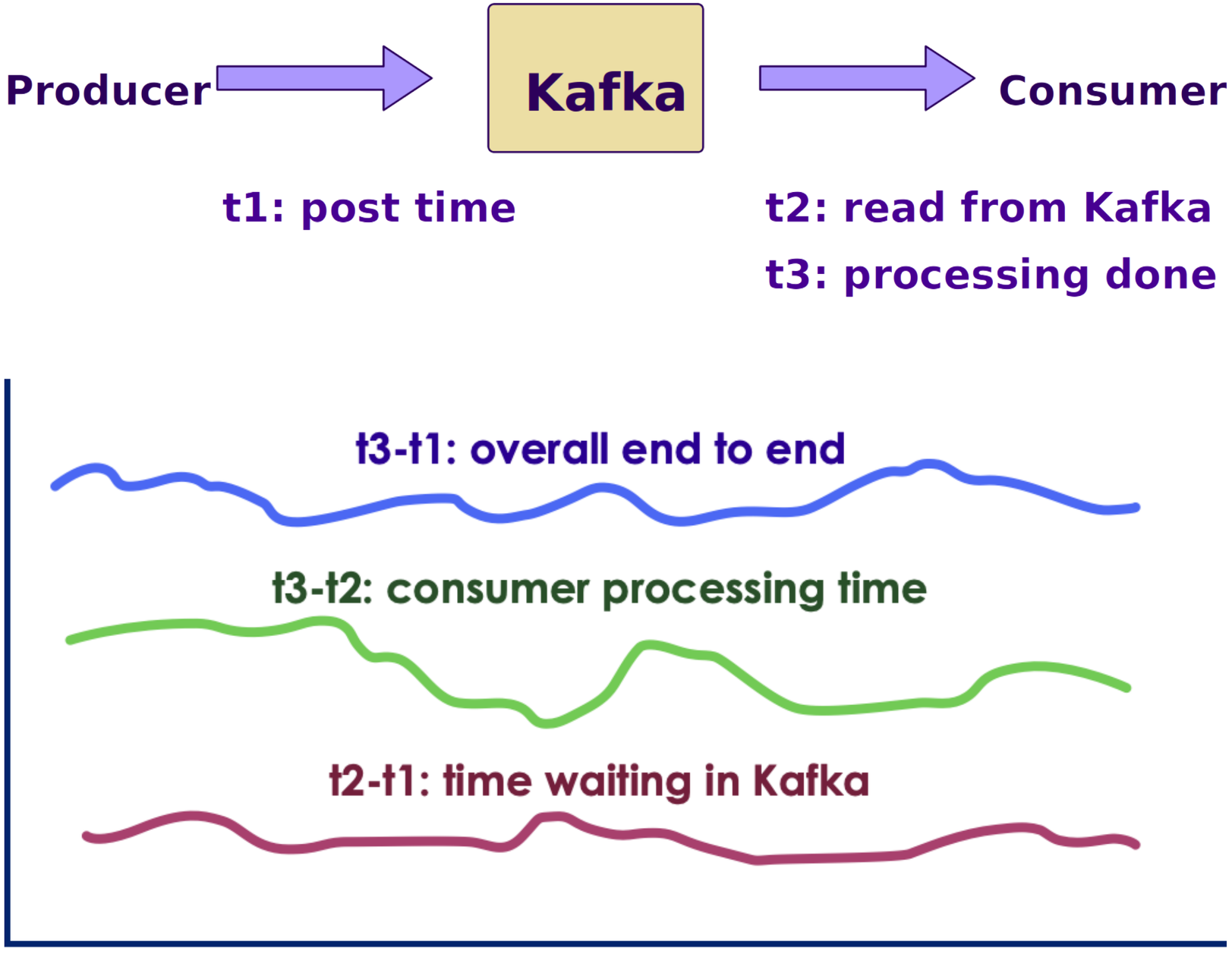
Monitoring Kafka

- **Log flush latency**
  - How long does it take to flush to disk
  - Longer it takes, longer the write pipeline backs up!
- **Under Replicated Partitions**
  - Replication is lagging behind
  - Messages are being written at very high speed
  - Consumers won't get data that isn't replicated
    - Consumers lag behind as well
  - Chance of data loss is high, when the lead broker fails

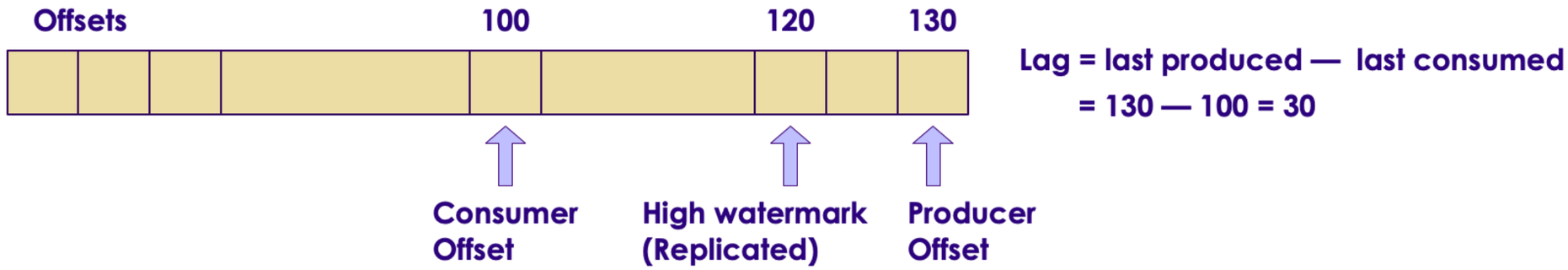
Kafka Monitoring: End to End Lag

- **End-to-End time**
  - How long does it take for a message to arrive from Producer to Consumer
  - Indicates overall speed / latency of kafka pipeline
- Below, is an example (see next slide for graph)
  - (t2 - t1): how long message was waiting in Kafka queue
  - (t3 - t2): consumer side processing time
  - (t3 - t1): overall processing

Best Practices: End to End Latency



Kafka Monitoring Consumer Lag



- Consumer Lag = Size of Partition (last offset) - Consumer offset (last committed)
- Large offsets means consumers can't keep up with data
- **Question for class:** What can cause consumer lag?
- Tools to monitor consumer lag:
  - JMX stats
  - Burrow
  - Confluent dashboard
  - Datadog

Kafka Streams + Metrics

```
import com.codahale.metrics.MetricRegistry;
import com.codahale.metrics.Meter;
import com.codahale.metrics.Timer;

private final MetricRegistry metrics = new MetricRegistry();
// register listener (Console & Graphite)

final Meter meterEvents = metrics.meter("events");
final Timer timerExec = metrics.Timer("time_to_process");
// ...snip...
final KStream<String, String> clickstream = // create stream

// process each record and report traffic
clickstream.foreach(new ForeachAction<String, String>() {
   public void apply(String key, String value) {
     meterEvents.mark(); // got the event!

     Timer.Context context = timerExec.time();
     // process the event
     context.stop();
  }
});
// start the stream
```

Lab 9: Kafka Metrics Labs

- **Overview:**
  - Use Metrics with Kafka
- **Approximate Time:**
  - ~30 - 40 mins
- **Instructions:**
  - Please follow: lab 9

Review and Q&A

- Let's go over what we have covered so far
- Any questions?