AARHUS UNIVERSITET

Microservices and DevOps

Scalable Microservices

Monitoring

Henrik Bærbak Christensen

AARHUS UNIVERSITET

Monitorability

- According to Bass et al. (2012)

> **Monitorability**
>
> Monitorability deals with the ability of the operations staff to monitor the system while it is executing. Items such as queue lengths, average transaction processing time, and the health of various components should be visible to the operations staff so that they can take corrective action in case of potential problems. Scenarios will deal with a potential problem and its visibility to the operator, and potential corrective action.

- *Ability to monitor system while executing, to take corrective action in case of potential problems*

AARHUS UNIVERSITET

From my own backyard…

- Christmas lunch with the family
  - Ups…
- *What's the first thing you want to know? What the hell has gone wrong?*

- Easier in a monolith
  - One system, one log
- Microservices?
  - 30 systems, one big mess

AARHUS UNIVERSITET

Monitor System

- So – what do we want to monitor?
  - *Data that allows us corrective action*
- Which boils down to three things
  - That *we have data* – live and historic
    - That is, our services must *provide data*
  - That data provides the information that *allows corrective action*
    - That is, our services must provide the *right and relevant data*
  - And – that we can actually *find/overview that required data*
    - That is, our services' data is available, searchable, and meaningful

AARHUS UNIVERSITET

Monitor System

- So – what do we want to monitor?
  - *Data that allows us corrective action*
- And what data may that be?

- Three major classes
  - Hardware related data: CPU, Disk, IO, Memory, …
  - Domain related data: User behavior, Access patterns,
  - Architecture related data: Performance, Availability, …

  - Requires set of tools and techniques

AARHUS UNIVERSITET

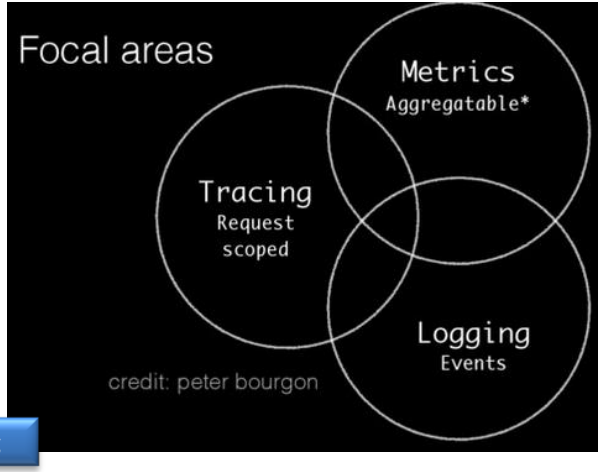
Metric Types

- Three major classes
  - Hardware related data: CPU, Disk, IO, Memory, …
  - Domain related data: User behavior, Access patterns,
  - Architecture related data: Performance, Availability, …
- Alternative, not completely orthogonal terms
  - **System** metrics Operating system/hardware
  - **Platform** metrics Framework/JVM
  - **Application** metrics Own logging at application level
- Splunk terminology
  - Logging for debug, Semantic Logging

AARHUS UNIVERSITET

Terminology

- Newman is in his usual ‘hand waiving’ mode
- Adrian Cole defines ***Observability***
  - Logs recording events
  - Metrics data combined from *measuring* events
  - Tracing recording events with *causal ordering*
- Ex
  - Log quote service failure
  - Measure # users each hour
  - Trace call seq for cmd ‘quote’ command



https://www.dotconferences.com/2017/04/adrian-cole-observability-3-ways-logging-metrics-tracing

AARHUS UNIVERSITET

Monitoring Anti-thesis

- Let us try to negate the statement
  - *Data that allows us corrective action*
- What can lead to situations in which *we cannot start corrective actions?*

- *Exercise – your system breaks and you cannot correct*
- Name the issues that may lead to this situation…

Single Service, Single Server

- The ‘simple case’
  - Monitor the machine
    - Nagios, ‘htop’, …
  - Access to log files
    - Ssh, and ‘more’ ☺
  - Application monitoring
    - E.g. response time

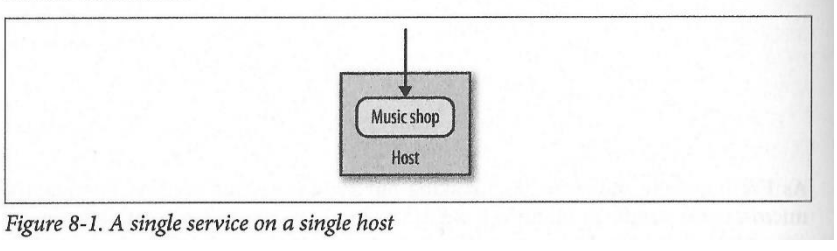
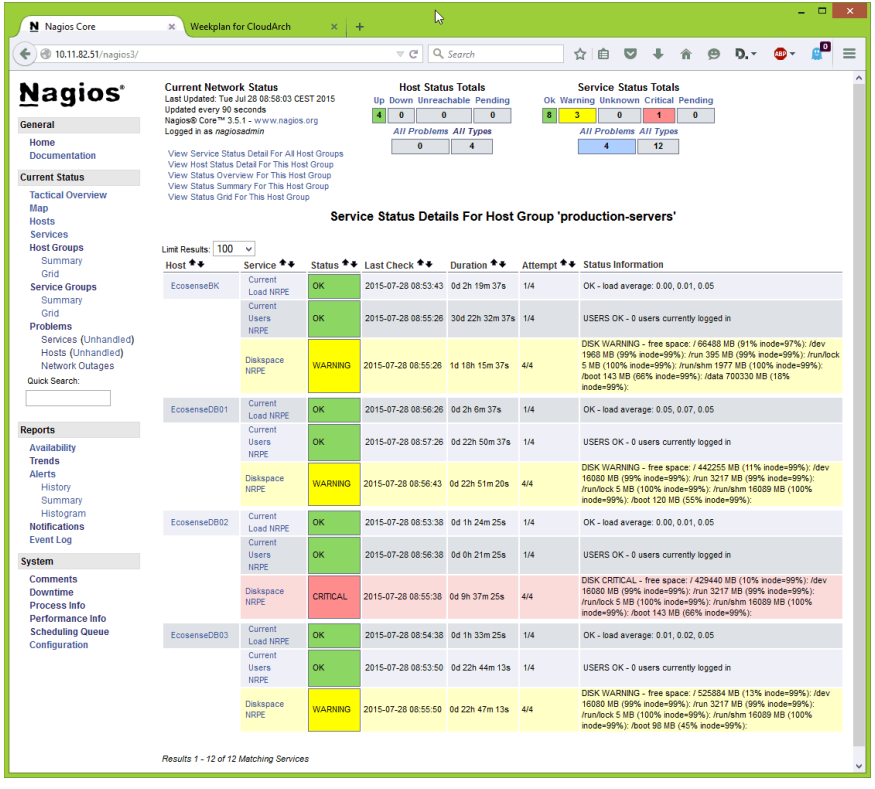


Figure 8-1. A single service on a single host

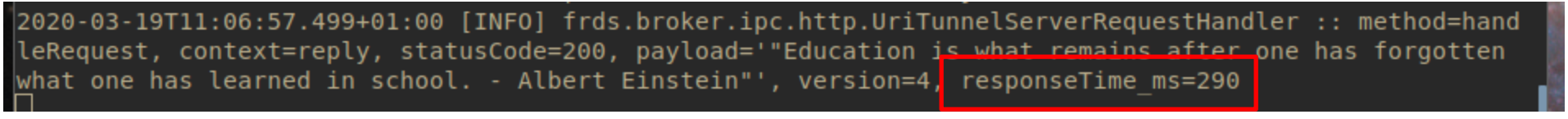



```
2020-03-19T08:47:54.290+01:00 [INFO] crunch.maestro.AssessExerciseDaemonCmd :: method=assess, context=end, groupName=Oscar, passed=true
2020-03-19T08:47:54.291+01:00 [INFO] crunch.maestro.StandardMaestro :: method=assessSubmittedExercisesByGroup, context=iterate-exercise, exercise=quote-service, submitted=true
2020-03-19T08:47:54.291+01:00 [INFO] crunch.maestro.AssessExerciseDaemonCmd :: method=assess, context=begin, groupName=Oscar, exercise=quote-service
2020-03-19T08:47:54.292+01:00 [INFO] crunch.maestro.AssessExerciseDaemonCmd :: method=assess, context=daemon-start, groupName=Oscar, exercise=quote-service
2020-03-19T08:47:54.292+01:00 [INFO] [mikravn/private:latest] :: Creating container for image: mikravn/private:latest
2020-03-19T08:47:54.506+01:00 [INFO] [mikravn/private:latest] :: Starting container with ID: 3b7341495cb3c6ab74fc6489d146dce39695ff21115d8dd93dab52954a21591c
2020-03-19T08:47:55.579+01:00 [INFO] [mikravn/private:latest] :: Container mikravn/private:latest is starting: 3b7341495cb3c6ab74fc6489d146dce39695ff21115d8dd93dab52954a21591c
```

AARHUS UNIVERSITET

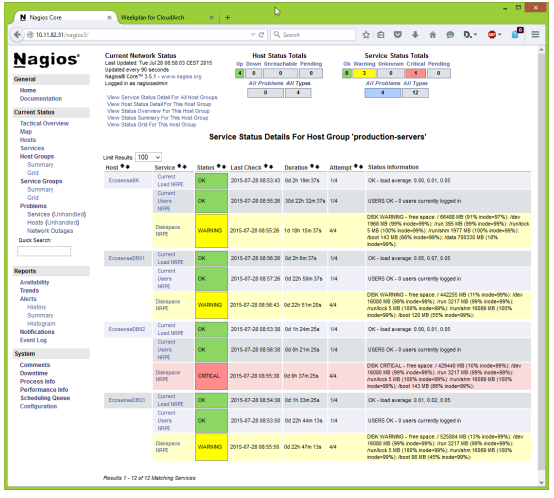
Metrics Collection

- Actually a bit of application metrics is collected by the URITunnelServerRequestHandler of the Frds.Broker library:



Single Service, Multiple Servers

- Horizontal Scaling
  - Monitor *all* machines
  - Aggregate logs!



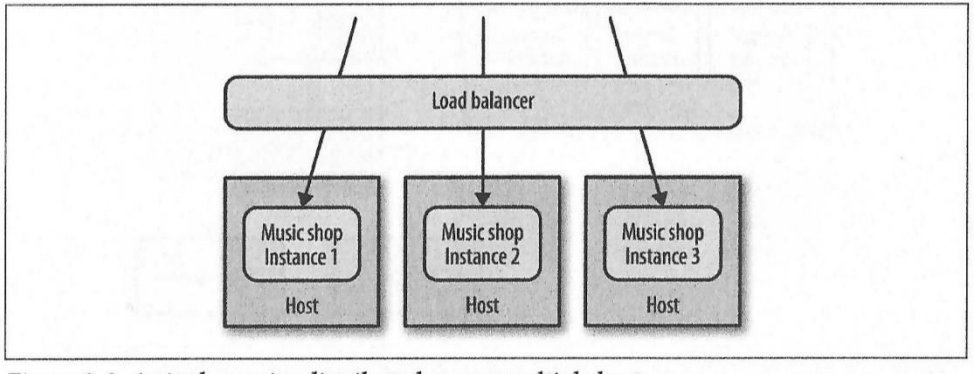


Figure 8-2. A single service distributed across multiple hosts

AARHUS UNIVERSITET

Swarm Aggregates Logs

- On a per-service level
  - Which makes sense…

```
csdev@m51:~/proj/cave$ docker service logs msdo_daemon
```

```
msdo_daemon.3.2rey2r54udqf@m51    | 2020-03-19T10:12:37.974Z [INFO] org.eclipse.jetty.server.session :: No SessionScavenger set, using defaults
msdo_daemon.2.krbul95y0odh@m51    | 2020-03-19T10:12:37.690Z [INFO] org.eclipse.jetty.server.Server :: Started @448ms
msdo_daemon.3.2rey2r54udqf@m51    | 2020-03-19T10:12:37.975Z [INFO] org.eclipse.jetty.server.session :: Scavenging every 660000ms
msdo_daemon.3.2rey2r54udqf@m51    | 2020-03-19T10:12:37.986Z [INFO] org.eclipse.jetty.server.AbstractConnector :: Started ServerConnector@71024f77{HTTP
.0:7777}
msdo_daemon.3.2rey2r54udqf@m51    | 2020-03-19T10:12:37.987Z [INFO] org.eclipse.jetty.server.Server :: Started @343ms
msdo_daemon.1.te9wukmf2jxx@m51    | 2020-03-19T10:12:38.840Z [INFO] org.mongodb.driver.cluster :: Cluster created with settings {hosts=[cavedb:27017],
lusterType=UNKNOWN, serverSelectionTimeout='30000 ms', maxWaitQueueSize=500}
msdo_daemon.1.te9wukmf2jxx@m51    | 2020-03-19T10:12:38.873Z [INFO] org.mongodb.driver.cluster :: Cluster description not yet available. Waiting for 30
ut
```

- However, to diagnose *one service* some filtering is required

Multiple Services, Multiple Servers

- Now it gets nasty ☹

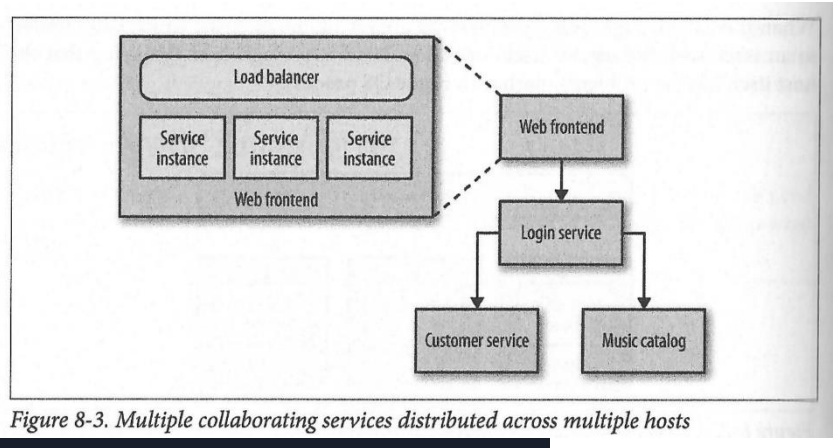
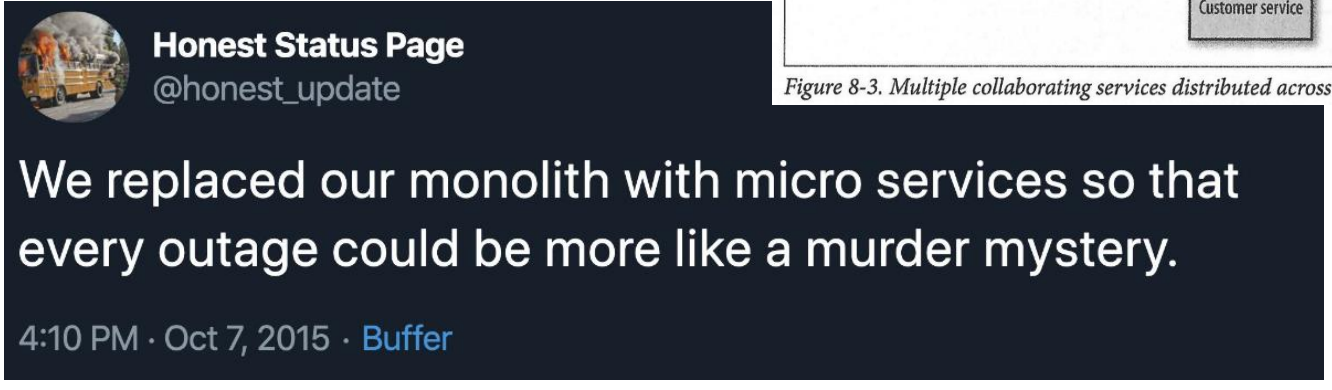


Figure 8-3. Multiple collaborating services distributed across multiple hosts



- *"Answer is collection and central aggregation"* [Newman, p 158]

Aggregation

- So – we need something to aggregate logs from many individual services…

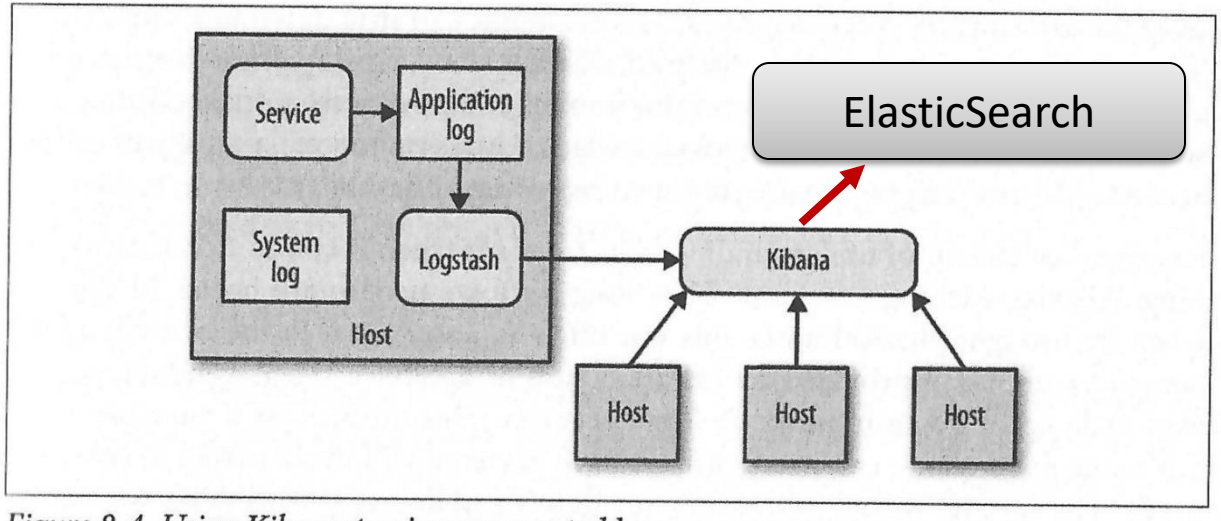
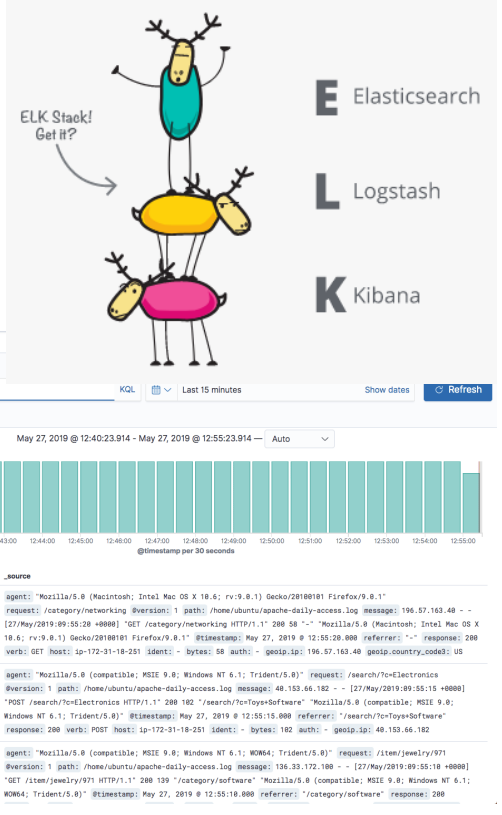


Figure 8-4. Using Kibana to view aggregated logs

ELK

- The classic solution to log aggregation is ELK
  - Logstash
    - Log ‘ingest pipeline’, collect logs from all services
  - Elasticsearch
    - JSON-based search engine
  - Kibana
    - Flexible visualization tool
- Tried it for MSDO
  - ☹ Really heavy machinery ☹
  - And hell to get going ☹



AARHUS UNIVERSITET

Service Metrics

- Section is about what I would call ‘domain metrics’
  - Or ‘semantic logging’ in the Splunk paper
- *Our service logs data related to the domain it handles*
  - *Expose number of times customers view past orders*
  - *How big is today’s sales*
  - *Which features are use, and how often, and how*

- Why
  - Improve service based upon real data
  - Humble: ‘Are we building the right thing’ / scientific experiments

Example

- Example (From the Splunk paper)

```
void submitPurchase(purchaseId)
{
    log.info("action=submitPurchaseStart, purchaseId=%d", purchaseId)
    // These calls throw an exception on error:

    submitToCreditCard(...)
    generateInvoice(...)
    generateFullfillmentOrder(...)
    log.info("action=submitPurchaseCompleted, purchaseId=%d", purchaseId)
}
```

- Graph your purchase volume by hour, by day, and by month.
- Find out how long purchases take during different times of the day and days of the week.
- Find out how long purchases are taking now compared to last month.
- Find out whether systems are getting slower.
- Find out how many purchases are failing, and graph these failures over time.
- Find out which specific purchases are failing.

AARHUS UNIVERSITET

Semantic Monitoring

- *Wrongly headlined as* 'Synthetic Monitoring' *in the book*
- Semantic Monitoring:
  - Insert synthetic transactions into the production environment
- A synthetic transaction is a 'fake event'
  - Can then be monitored for 'proper behavior' of system
    - *Does system handle event properly within timelimit?*
  - Of course, important to isolate from *real* events
- WarStory:
  - *Large number of washing machines arrived at head office ☺*
- *Preferably over monitoring hardware metrics, like CPU*
  - Blocked threads may halt a system even though CPU is fine…

AARHUS UNIVERSITET

Semantic Monitoring

- How to do it?
- Well, you already have the infrastructure ☺
- *It is a test journey! The test cases have (almost) been written.*
- Again, care must be taken to handle synthetic events in a way that does not influence real behaviors…

AARHUS UNIVERSITET

Similar Example

- I worked in the HealthCare domain for some years
  - Nancy
    - 251248-9996

Sundhedsvæsenet har behov for at anvende valide personnumre med fiktive data i produktionsmiljøer til brug for test. Anvendelsen af "rigtige" CPR-numre til testformål har gennem mange år været nødvendig for at sikre korrekt anvendelse, funktionalitet og til fejlfinding i sundhedssektorens it-systemer. Hidtil har der været brugt varierende testpersoner, hvor Nancy Ann Berggren nok er den mest kendte.

Source: MedCom website

- She has suffered *all diseases ever invented* ☹

Warstory: The letter to most wealthy customers of US bank.

AARHUS UNIVERSITET

Correlation ID

- One big issue in MS architectures
  - *Any given event/transaction may involve partial processing by a set of services*
    - That is, there is not five log messages from the ‘processItem()’ method execution…
    - There are two log messages in one service, three in another, two in a third one, etc.
      - *And all are interleaved with processing of **other** transactions!*
- Traceability is lost! No ‘stack-trace’ to diagnose ☹

Correlation ID

- Correlation ID:
  - A UUID generated by the *initiating* call, and then passed along to all subsequent calls

```
15-02-2014 16:01:01 Web-Frontend INFO [abc-123] Register
15-02-2014 16:01:02 RegisterService INFO [abc-123] RegisterCustomer ...
15-02-2014 16:01:03 PostalSystem INFO [abc-123] SendWelcomePack ...
15-02-2014 16:01:03 EmailSystem INFO [abc-123] SendWelcomeEmail ...
15-02-2014 16:01:03 PaymentGateway ERROR [abc-123] ValidatePayment ...
```

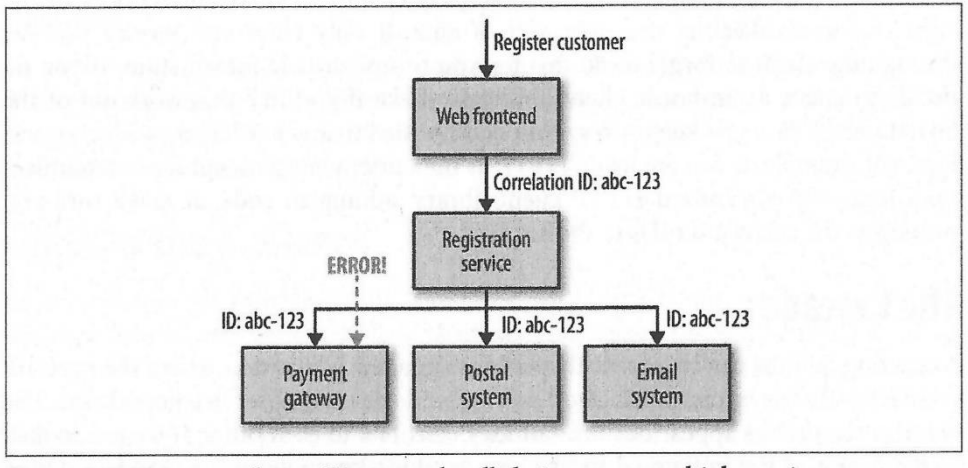


Figure 8-5. Using correlation IDs to track call chains across multiple services

AARHUS UNIVERSITET

Example

- RabbitMQ uses CorrelationId's to match replies with proper requests when it simulates RCP calls

Message properties

The AMQP 0-9-1 protocol predefines a set of 14 properties that go with a message. Most of the properties are rarely used, with the exception of the following:

- `deliveryMode`: Marks a message as persistent (with a value of `2`) or transient (any other value). You may remember this property from the second tutorial.
- `contentType`: Used to describe the mime-type of the encoding. For example for the often used JSON encoding it is a good practice to set this property to: `application/json`.
- `replyTo`: Commonly used to name a callback queue.
- `correlationId`: Useful to correlate RPC responses with requests.

Exercise

- SkyCave's *initiating call* hits … where?

- Is SkyCave prepared for using Correlation IDs?

Standardization

- An aggregated log of 30 services, each with 30 different approaches to how log messages are formatted…
- … leads to too much time spent on formulating very complex queries…

- Standardization is the name of the game.

- Return to that in a moment.

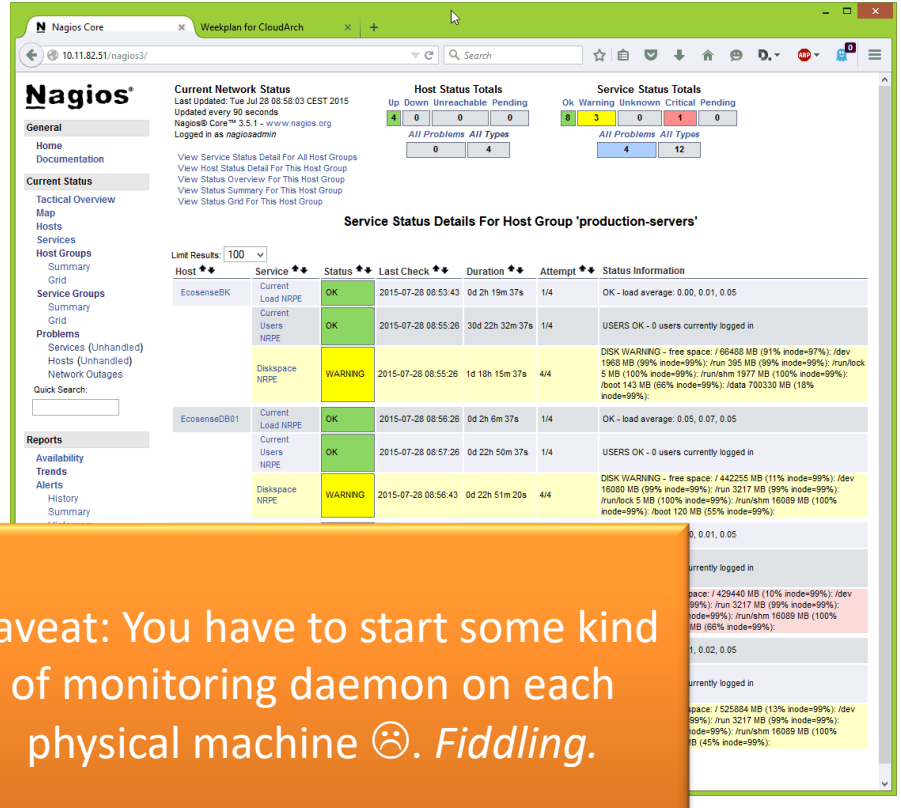
Tools and Libraries

Sigh – one zillion options

AARHUS UNIVERSITET

Hardware Data

- The Hardware side
  - Nagios
  - collectd
  - Windows: PerfMon
  - MetricBeat

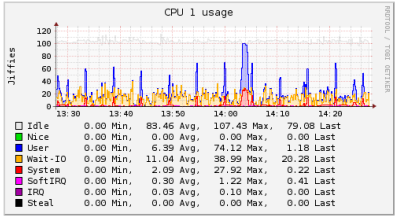


Caveat: You have to start some kind of monitoring daemon on each physical machine ☹. *Fiddling.*

What does collectd do?

collectd gathers metrics from various sources, e.g. the operating system, applications, logfiles and external devices, and stores this information or makes it available over the network. Those statistics can be used to monitor systems, find performance bottlenecks (i.e. *performance analysis*) and predict future system load (i.e. *capacity planning*). Or if you just want pretty graphs of your private server and are fed up with some homegrown solution you're at the right place, too ;).

A graph can say more than a thousand words, so here's a graph showing the CPU utilization of a system over the last 60 minutes:



AARHUS UNIVERSITET

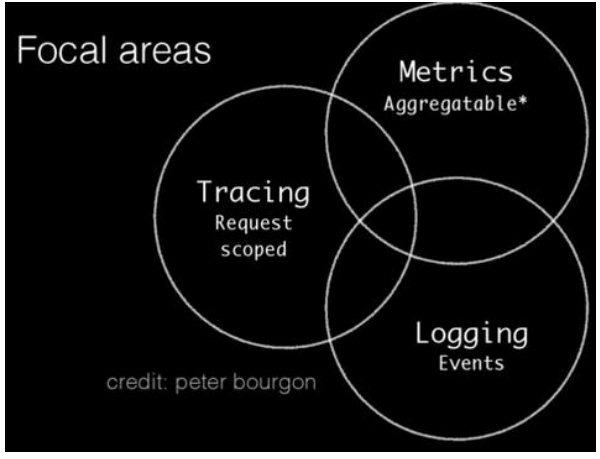
Architecture + Domain Data

- Require *logging* in applications + Log aggregation
  - Syslog (from the 1980'ies!)
  - ELK
  - And a zillion more, lots of vendors in this space…

AARHUS UNIVERSITET

Logging Versus Metrics

- Back to Adrian Cole
  - Logs: events emitted
    - Log.info(“method=foo(), param=7”);
  - Metrics: *periodic* aggregations
    - meter = new Meter(“getQuote-requests”)
    - meter.mark()
      - Counts all ‘mark()’ calls associated with that meter
      - Emits log message every 5 minutes with aggregated statistics
    - Return to that later ☺…



AARHUS UNIVERSITET

Our Choice in MSDO

Humio

Why Humio?

- Humio because
  - Community SaaS offering for free
  - Built-in tutorial ☺
    - Lower the learning curve
  - I made it run in one hour
    - Compared to the tears I shed doing ELK
  - It was a Danish (start-up) company
    - Now sold for an astronomical amount
  - And, I know the founder personally ☺

AARHUS UNIVERSITET

Metrics Libraries

Example: Dropwizard Metrics

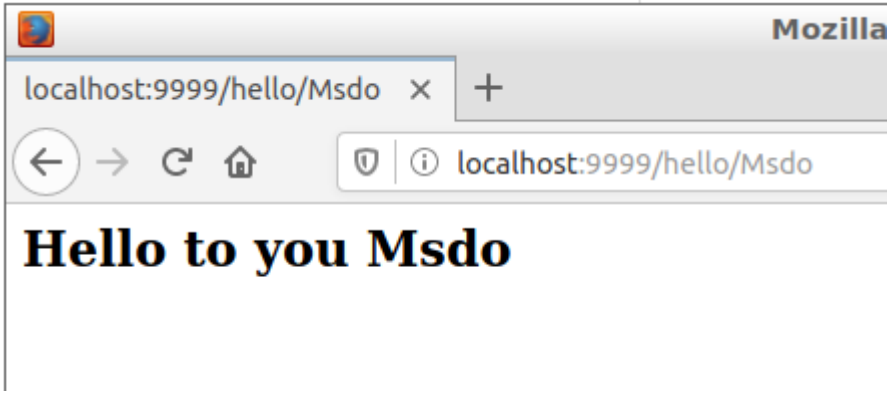
Just a simple Coding Kata

AARHUS UNIVERSITET

Hello World Web App

- The simplest ‘Hello World’ Spark-Java server:

```
private void registerRoutes() {
  port(9999);
  get( path: "/hello/:name",
          (req, res) -> {
            getRequests.mark();
            return "<H2>Hello to you " + req.params(":name") + "</H2>";
          }
  );

}
```



Monitoring it

- The Registry; the Meter; and the Slf4J Reporter

```
final MetricRegistry metrics;
final Meter getRequests;
final Slf4jReporter reporter;

public HelloSpark() {
  metrics = new MetricRegistry();
  getRequests = metrics.meter( name: "get-requests");

  reporter = Slf4jReporter.forRegistry(metrics)
          .outputTo(LoggerFactory.getLogger( name: "kata.sample"))
          .convertRatesTo(TimeUnit.SECONDS)
          .convertDurationsTo(TimeUnit.MILLISECONDS)
          .build();
  reporter.start( period: 1, TimeUnit.MINUTES);
  registerRoutes();
}
```

Meter = ‘rate’, events pr second

```
get( path: "/hello/:name",
        (req, res) -> {
          getRequests.mark();
          return "<H2>Hello to you " + req.params(":name") + "</H2>";
        }
);
```

AARHUS UNIVERSITET

Output to Log4J

- Total counts, 1 minute mean, 5 minute mean, etc.

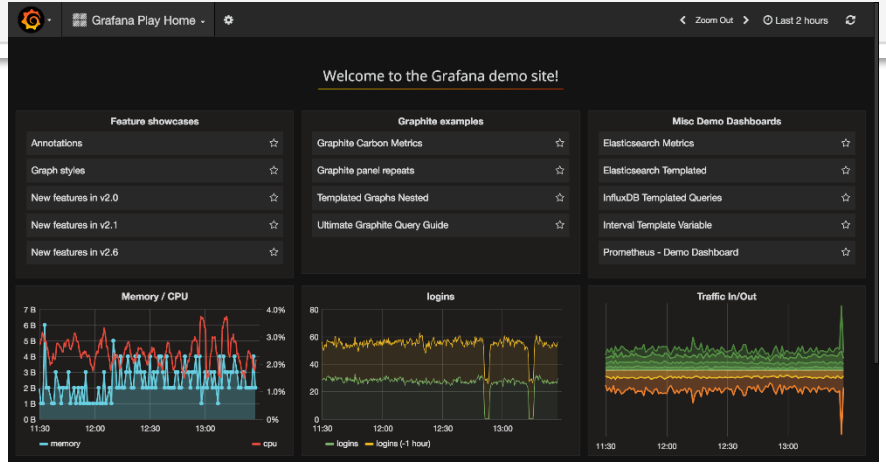
```
2020-03-18T10:03:13.274+01:00 [INFO] org.eclipse.jetty.server.AbstractConnector :: Started ServerConnector@3ce8fe25{HTTP/1.1,[http/1.1]}{0.0.0.0:9999}
2020-03-18T10:03:13.274+01:00 [INFO] org.eclipse.jetty.server.Server :: Started @251ms
2020-03-18T10:04:13.184+01:00 [INFO] kata.sample :: type=METER, name=get-requests, count=1, m1_rate=0.007553668846787294, m5_rate=0.0028452503068887463, m15_rate=0.001053991118789713, mean_rate=0.016652130306912764,
2020-03-18T10:05:13.167+01:00 [INFO] kata.sample :: type=METER, name=get-requests, count=4, m1_rate=0.05075219069655626, m5_rate=0.012246621633483888, m15_rate=0.004310107250851028, mean_rate=0.03332333000845199, rat
2020-03-18T10:06:13.167+01:00 [INFO] kata.sample :: type=METER, name=get-requests, count=4, m1_rate=0.0186706875516756, m5_rate=0.010026685752643365, m15_rate=0.004032135439406559, mean_rate=0.022217796626972857, rat
2020-03-18T10:07:13.167+01:00 [INFO] kata.sample :: type=METER, name=get-requests, count=4, m1_rate=0.0068685621027970295, m5_rate=0.008209155977137972, m15_rate=0.003772090868158367, mean_rate=0.016664179717678078,
2020-03-18T10:08:13.167+01:00 [INFO] kata.sample :: type=METER, name=get-requests, count=4, m1_rate=0.0025268027880283195, m5_rate=0.006721088455296787, m15_rate=0.003528817355336131 2, mean_rate=0.013331754886262039,
```

AARHUS UNIVERSITET

Alternative: Graphite

- Alternatively, output is sent to Graphite

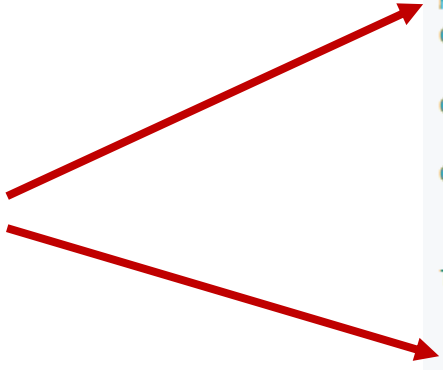
```
final Graphite graphite = new Graphite(new InetSocketAddress("graphite.example.com", 2003));
final GraphiteReporter reporter = GraphiteReporter.forRegistry(registry)
                                                  .prefixedWith("web1.example.com")
                                                  .convertRatesTo(TimeUnit.SECONDS)
                                                  .convertDurationsTo(TimeUnit.MILLISECONDS)
                                                  .filter(MetricFilter.ALL)
                                                  .build(graphite);

reporter.start(1, TimeUnit.MINUTES);
```



AARHUS UNIVERSITET

Other Metrics Library

- Micrometer
  - Integration with Resilience4J!



```
MeterRegistry meterRegistry = new SimpleMeterRegistry();
CircuitBreakerRegistry circuitBreakerRegistry =
  CircuitBreakerRegistry.ofDefaults();
CircuitBreaker foo = circuitBreakerRegistry
  .circuitBreaker("backendA");
CircuitBreaker boo = circuitBreakerRegistry
  .circuitBreaker("backendB");

TaggedCircuitBreakerMetrics
  .ofCircuitBreakerRegistry(circuitBreakerRegistry)
  .bindTo(meterRegistry)
```

  - So now your CB data is aggregated and logged periodically…
  - And Micrometer can send data to Humio…

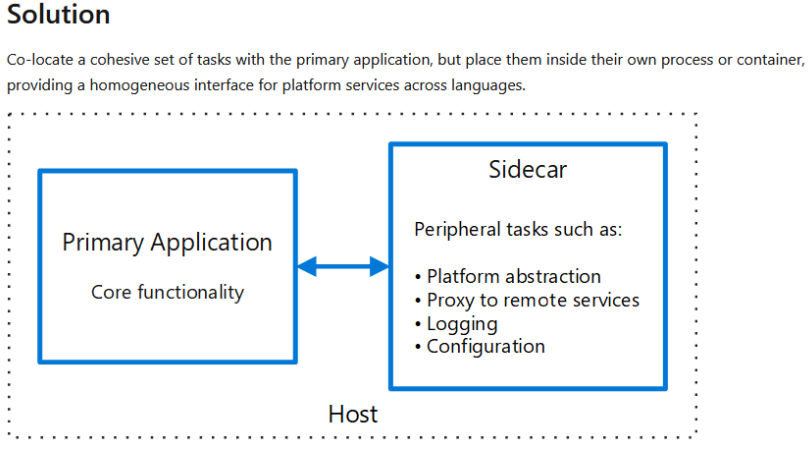
AARHUS UNIVERSITET

Non-invasive Logging

Sidecars

- ‘Decorator pattern’ on services

- https://docs.microsoft.com/en-us/azure/architecture/patterns/sidecar



AARHUS UNIVERSITET

Summary

- Phew…
- Vast and important topic
  - I looked *real hard* to find a terminology strong classification of the topic…
    - I am still searching
- But – the key concepts are…

AARHUS UNIVERSITET

Key Concepts

- Monitorability: *Ability to monitor system while executing, to take corrective action in case of potential problems*

- Classes of data
  - Hardware data, architectural data, domain data

- Classes of ‘events’
  - Logging (event), Metrics (aggregates), Traces (correlated events)