Huge Codebases
Taming the Beasts

20 Jan. 2016

Roman Mohr
Red Hat

JBug
January 2016

About Me

Roman Mohr

Senior Software Engineer at Red Hat

Member of the SLA team in oVirt/RHEV

Twitter: @rfenkhuber

Mail: rmohr@redhat.com

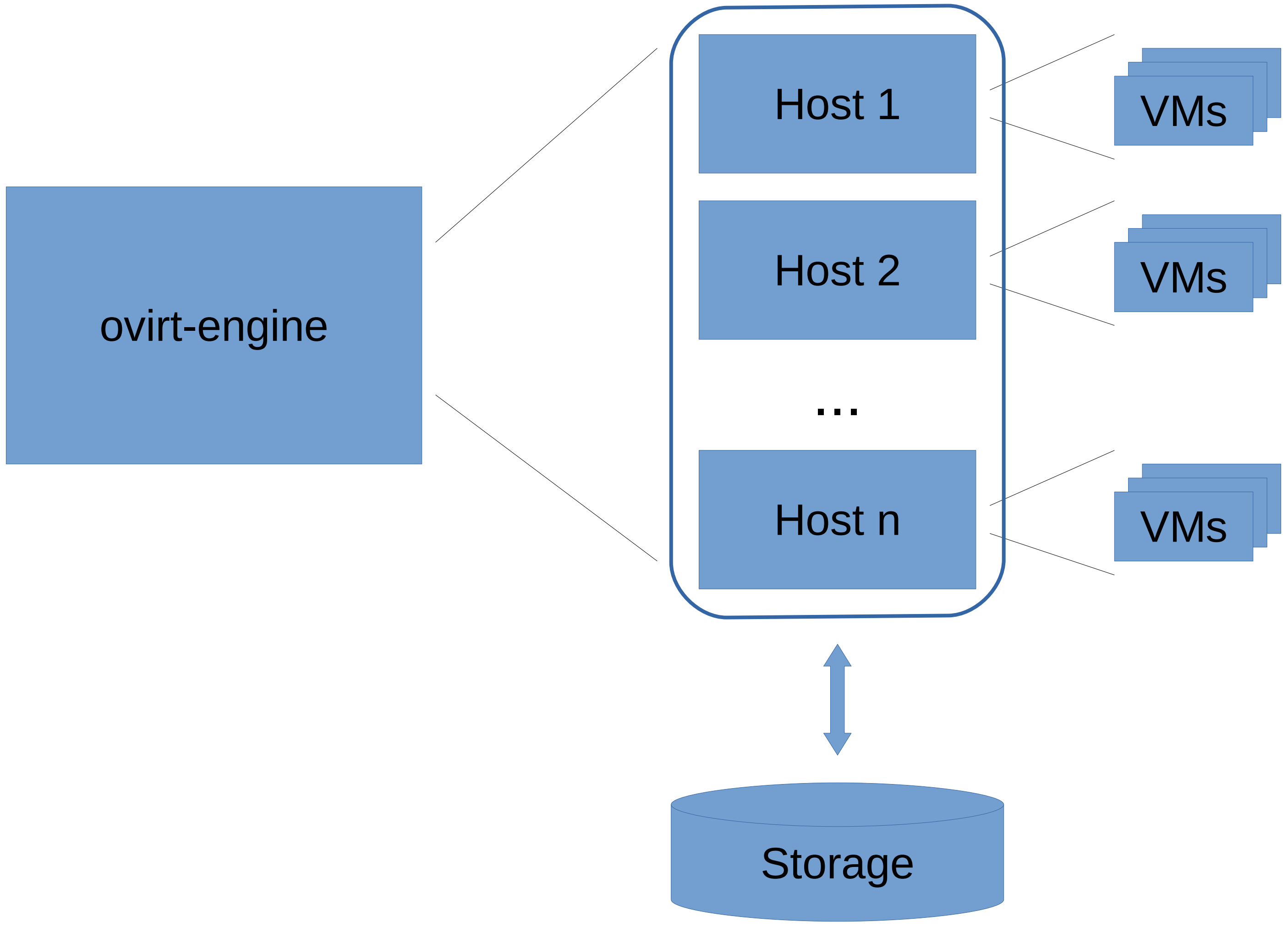
Github: https://github.com/rmohr

IRC: #ovirt irc.oftc.net

oVirt

“oVirt is a powerful virtual machine manager for up to datacenter-class deployments, and provides an awesome KVM management interface for multi-node virtualization.” – http://www.ovirt.org

oVirt Artitecture

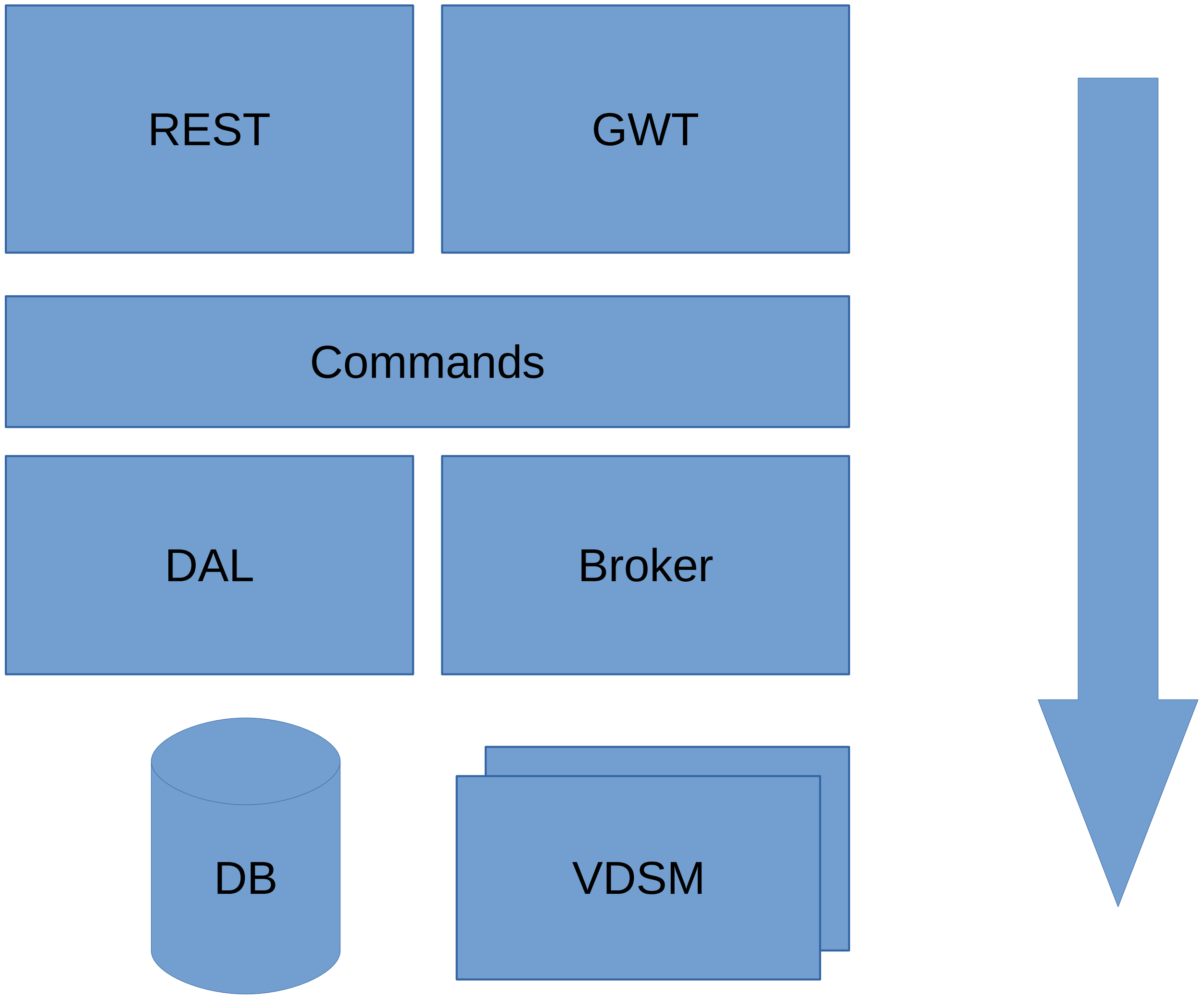


ovirt-engine

The Beast

`Take 1`

Architecture of ovirt-engine



Git Statistics of ovirt-engine

Branch: **master**

Generated: 2016-01-14 14:02:53 (in 370 seconds)

Generator: GitStats (version 2014-12-09), git version 2.4.3, gnuplot 5.0 patchlevel 0

Report Period: 2011-10-04 18:43:09 to 2025-03-31 23:18:53

Age: 4928 days, 1449 active days (29.40%)

Total Files: 10355

Total Lines of Code: **1123168** (**2557376 added**, **1434208 removed**)

Total Commits: **20166** (average 13.9 commits per active day)

Authors: 174 (average 115.9 commits per author)

Issues we have

- Many developers
- A lot of code
- No second level cache
- REST performance problems
- The product runs at the user/customer site
- Test coverage
- Hard to configure and run the application

Where to start?

- Try to get a high level overview of the architecture
- “Datamine your sourcecontrol” – Greg Young*
- Gather code metrics (JArchitect, Sonar)
- Monitor your application **before** you change something

* How to get productive in a project in 24h

https://www.youtube.com/watch?v=KaLROwp-VDY

Java and
Application Monitoring

Profiler

- Natural first Choice
- You can see where your application spends its time
- Easy to get started. Just connect to the JVM in question and browse the cpu profiling graph.

JProfiler:

- Can handle huge code bases
- Supports JDBC, JPA and NoSQL
- Can handle Application Servers

Profiler

- Provides no application or framework specific interpretation of data
- Can impact performance when used in production
- How do you get access to the system of the user?
- Many good profilers are closed source

XRebel for Monitoring?

- Easy to integrate. Just start an additional Java agent
- Every servlet now contains an additonal popup where you can access application metrics.

You can see:

- Basic profiling information
- Session size
- Database calls per page action
- Stack traces
- ...

Xrebel for Monitoring?

Development Only!

NewRelic for Monitoring?

- Excellent visualization
- Supports multi host applications
- Knows a lot about Java

Examples:

- JAX-RS
- Ehcache
- Transactions
- Databases
- Solr
- ...

NewRelic for Monitoring?

- Production only
- Closed Source
- How to get data from users? They would have to buy licenses

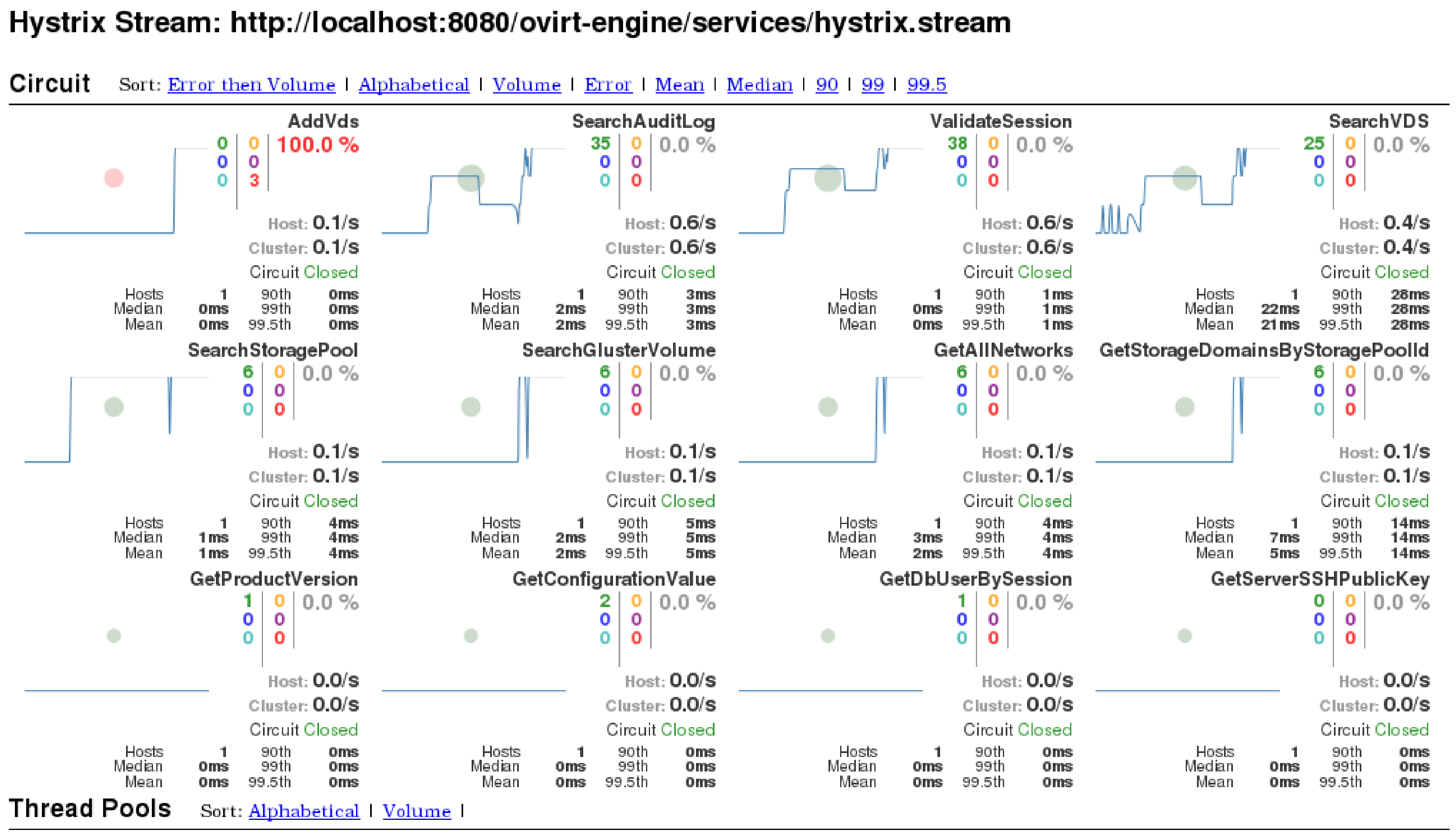
Hystrix

"Hystrix is a latency and fault tolerance library designed to isolate points of access to remote systems, services and 3rd party libraries, stop cascading failure and enable resilience in complex distributed systems where failure is inevitable." – https://github.com/Netflix/Hystrix

Hystrix

Hystrix also provides metrics!

Hystrix Dashboard



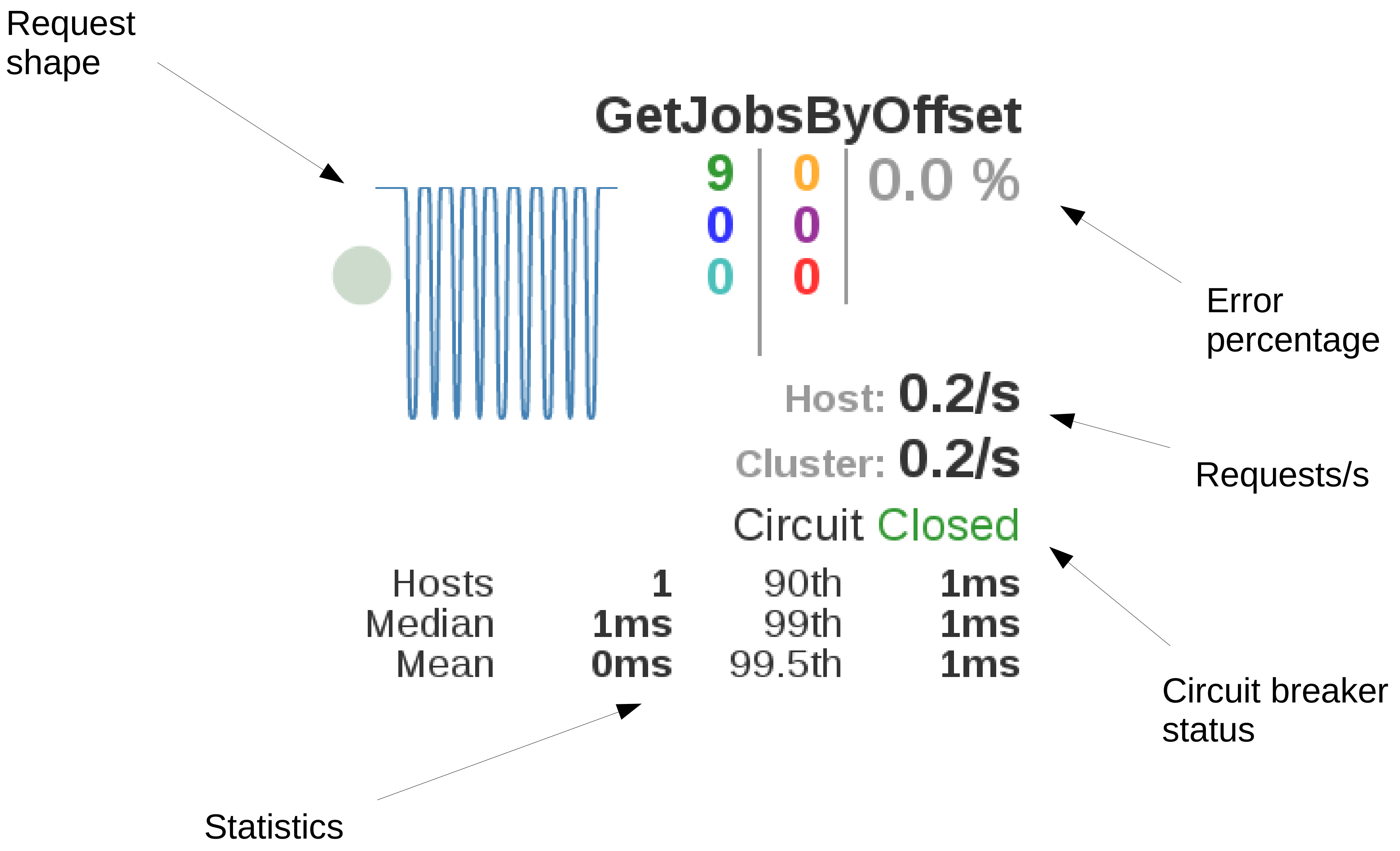
Hystrix Stream: http://localhost:8080/ovirt-engine/services/hystrix.stream

Circuit Sort: Error then Volume | Alphabetical | Volume | Error | Mean | Median | 90 | 99 | 99.5

```
AddVds
0  0  100.0 %
0  0
0  3
Host: 0.1/s
Cluster: 0.1/s
Circuit Closed
Hosts    1    90th    0ms
Median   0ms  99th    0ms
Mean     0ms  99.5th  0ms

SearchAuditLog
35  0  0.0 %
0   0
0   0
Host: 0.6/s
Cluster: 0.6/s
Circuit Closed
Hosts    1    90th    3ms
Median   2ms  99th    3ms
Mean     2ms  99.5th  3ms

ValidateSession
38  0  0.0 %
0   0
0   0
Host: 0.6/s
Cluster: 0.6/s
Circuit Closed
Hosts    1    90th    1ms
Median   0ms  99th    1ms
Mean     0ms  99.5th  1ms

SearchVDS
25  0  0.0 %
0   0
0   0
Host: 0.4/s
Cluster: 0.4/s
Circuit Closed
Hosts    1     90th    28ms
Median   22ms  99th    28ms
Mean     21ms  99.5th  28ms

SearchStoragePool
6  0  0.0 %
0  0
0  0
Host: 0.1/s
Cluster: 0.1/s
Circuit Closed
Hosts    1    90th    4ms
Median   1ms  99th    4ms
Mean     1ms  99.5th  4ms

SearchGlusterVolume
6  0  0.0 %
0  0
0  0
Host: 0.1/s
Cluster: 0.1/s
Circuit Closed
Hosts    1    90th    5ms
Median   2ms  99th    5ms
Mean     2ms  99.5th  5ms

GetAllNetworks
6  0  0.0 %
0  0
0  0
Host: 0.1/s
Cluster: 0.1/s
Circuit Closed
Hosts    1    90th    4ms
Median   3ms  99th    4ms
Mean     2ms  99.5th  4ms

GetStorageDomainsByStoragePoolId
6  0  0.0 %
0  0
0  0
Host: 0.1/s
Cluster: 0.1/s
Circuit Closed
Hosts    1    90th    14ms
Median   7ms  99th    14ms
Mean     5ms  99.5th  14ms

GetProductVersion
1  0  0.0 %
0  0
0  0
Host: 0.0/s
Cluster: 0.0/s
Circuit Closed
Hosts    1    90th    0ms
Median   0ms  99th    0ms
Mean     0ms  99.5th  0ms

GetConfigurationValue
2  0  0.0 %
0  0
0  0
Host: 0.0/s
Cluster: 0.0/s
Circuit Closed
Hosts    1    90th    0ms
Median   0ms  99th    0ms
Mean     0ms  99.5th  0ms

GetDbUserBySession
1  0  0.0 %
0  0
0  0
Host: 0.0/s
Cluster: 0.0/s
Circuit Closed
Hosts    1    90th    0ms
Median   0ms  99th    0ms
Mean     0ms  99.5th  0ms

GetServerSSHPublicKey
0  0  0.0 %
0  0
0  0
Host: 0.0/s
Cluster: 0.0/s
Circuit Closed
Hosts    1    90th    0ms
Median   0ms  99th    0ms
Mean     0ms  99.5th  0ms
```

Thread Pools Sort: Alphabetical | Volume |

Hystrix Dashboard



Hystrix Dashboard

Successful	9	0	Timeouts (thread isolation)
Rejected (Short circuit)	0	0	Rejected (max. concurrent invocations)
Bad request (exception)	0	0	Failed executions (exception)

Hystrix Dashboard

Easy to run

```
$ git clone https://github.com/Netflix/Hystrix.git
$ cd Hystrix/hystrix-dashboard
$ ../gradlew jettyRun
> Running at http://localhost:7979/hystrix-dashboard
```

Easy to integrate

- Drop the WAR from maven central in your container
- Add the WAR as dependency and serve the resources folder on an endpoint.

Add Hystrix to your maven project

```
<dependency>
  <groupId>com.netflix.hystrix</groupId>
  <artifactId>hystrix-core</artifactId>
  <version>${hystrix.version}</version>
</dependency>
<dependency>
  <groupId>com.netflix.hystrix</groupId>
  <artifactId>hystrix-metrics-event-stream</artifactId>
  <version>${hystrix.version}</version>
</dependency>
```

Add the Hystrix Metrics Servlet

```
<servlet>
 <display-name>HystrixMetricsStreamServlet</display-name>
 <servlet-name>HystrixMetricsStreamServlet</servlet-name>
 <servlet-class>
    com.netflix.hystrix.contrib.metrics.eventstream.HystrixMetricsStreamServlet
 </servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>HystrixMetricsStreamServlet</servlet-name>
  <url-pattern>/hystrix.stream</url-pattern>
</servlet-mapping>
```

Hello World Hystrix Command

```
Setter setter = Setter.withGroupKey(
        HystrixCommandGroupKey.Factory.asKey("helloWorld")
).andCommandKey(
        HystrixCommandKey.Factory.asKey("helloWorld")
);


HystrixCommand<String> helloWorldCommand =
new HystrixCommand<String>(setter) {
    @Override protected String run() throws Exception {
        return "Hello world!";
    }
};


return helloWorldCommand.execute();
```

Wrapping ovirt-engine commands

```java
final HystrixCommand<VdcReturnValueBase> hystrixCommand = new HystrixCommand(setter) {
  @Override
  protected VdcReturnValueBase run() throws Exception {
    final VdcReturnValueBase returnValue = command.executeAction();
    if (returnValue.getSucceeded()) {
      return returnValue;
    }
    // throw this so that hystrix can see that this command failed
    throw new ActionFailedException(returnValue);
  }
};

// execute the command
try {
  return hystrixCommand.execute();
} catch (HystrixRuntimeException e) {
  // only thrown for hystrix, so catch it and proceed normally
  if (e.getCause() instanceof ActionFailedException) {
    return ((ActionFailedException) e.getCause()).getReturnValue();
  }
  throw e;
}
```

Configuration for monitoring

```java
private HystrixCommand.Setter setter(final String key) {
 return HystrixCommand.Setter.withGroupKey(
     HystrixCommandGroupKey.Factory.asKey(key)
 ).andCommandKey(
     HystrixCommandKey.Factory.asKey(key)
 ).andCommandPropertiesDefaults(
     HystrixCommandProperties.Setter()
             .withExecutionIsolationStrategy(SEMAPHORE)
             .withExecutionTimeoutEnabled(false)
             .withCircuitBreakerEnabled(false)
             .withFallbackEnabled(false)
             .withMetricsRollingStatisticalWindowInMilliseconds(60000)
             .withMetricsRollingStatisticalWindowBuckets(60)
             .withExecutionIsolationSemaphoreMaxConcurrentRequests(100)
 );
}
```

Aspectj

```
@Around("execution(public * org.ovirt.engine.core.bll.commands.*.execute(..))")
public Object CircuitBreakerAdvice(final ProceedingJoinPoint joinPoint) {
  HystrixCommand.Setter setter = HystrixCommand.Setter.withGroupKey(
     HystrixCommandGroupKey.Factory.asKey(joinPoint.getSignature().toShortString())
  ).andCommandKey(
      HystrixCommandKey.Factory.asKey(joinPoint.getSignature().toShortString())
  );
  HystrixCommand<Object> command = new HystrixCommand(setter) {
    @Override protected Object run() throws Exception {
      try {
        return joinPoint.proceed();
      } catch (Throwable throwable) {
        throw new RuntimeException(throwable);
      }
    }
  };
  return command.execute();
}
```

Spring Cloud Netflix

```java
@Component
public class StoreIntegration {
  @HystrixCommand(fallbackMethod = "defaultStores")
  public Object getStores(Map<String, Object> parameters) {
    //do stuff that might fail
  }
  public Object defaultStores(Map<String, Object> parameters) {
    return /* something useful */;
  }
```

ovirt-engine

The Problem

```
Take 2
```

Problem description

- We have a datacenter with 1000 VMs.
- We query the /api/vms endpoint which returns all VMs.
- We need 2.5 seconds to fetch them with no additional load.

- We have a datacenter with 2000 VMs.
- We query the /api/vms endpoint which returns all VMs.
- We need 5 seconds to fetch them with no additional load.

Solution: Let's curl a little bit

1000 VMs, 1 request

```
$> time bash rest.sh vms > /dev/null
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 7043k    0 7043k    0     0  2774k      0 --:--:--  0:00:02 --:--:-- 2775k

real 0m2.547s
user0m0.008s
sys  0m0.011s
```

2000 VMs, 1 request

```
$> time bash rest.sh vms > /dev/null
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 13.7M    0 13.7M    0     0  2876k      0 --:--:--  0:00:04 --:--:-- 3815k

real 0m4.900s
user0m0.009s
sys  0m0.014s
```

Solution: Let's curl a little bit more

oVirt

2000 VMs, 10 parallel requests

```
$> time seq 1 10 | parallel -j 10 bash rest.sh vms > /dev/null

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 13.7M    0 13.7M    0     0   488k      0 --:--:--  0:00:28 --:--:-- 3911k
[...]
100 13.7M    0 13.7M    0     0   487k      0 --:--:--  0:00:28 --:--:-- 3848k

real 0m29.590s
user0m0.212s
sys  0m0.438s
```

Solution: We can guess

- “That's because our database is so slow.”
- “The database can cache everything, it is because our REST application code is so slow.”
- “That's because we are keeping the database busy with status updates of Hosts and VMs.”
- “That's because our architecture is not smart enough, it is just an ordinary monolith. That must be solved with streaming and eventbuses.”

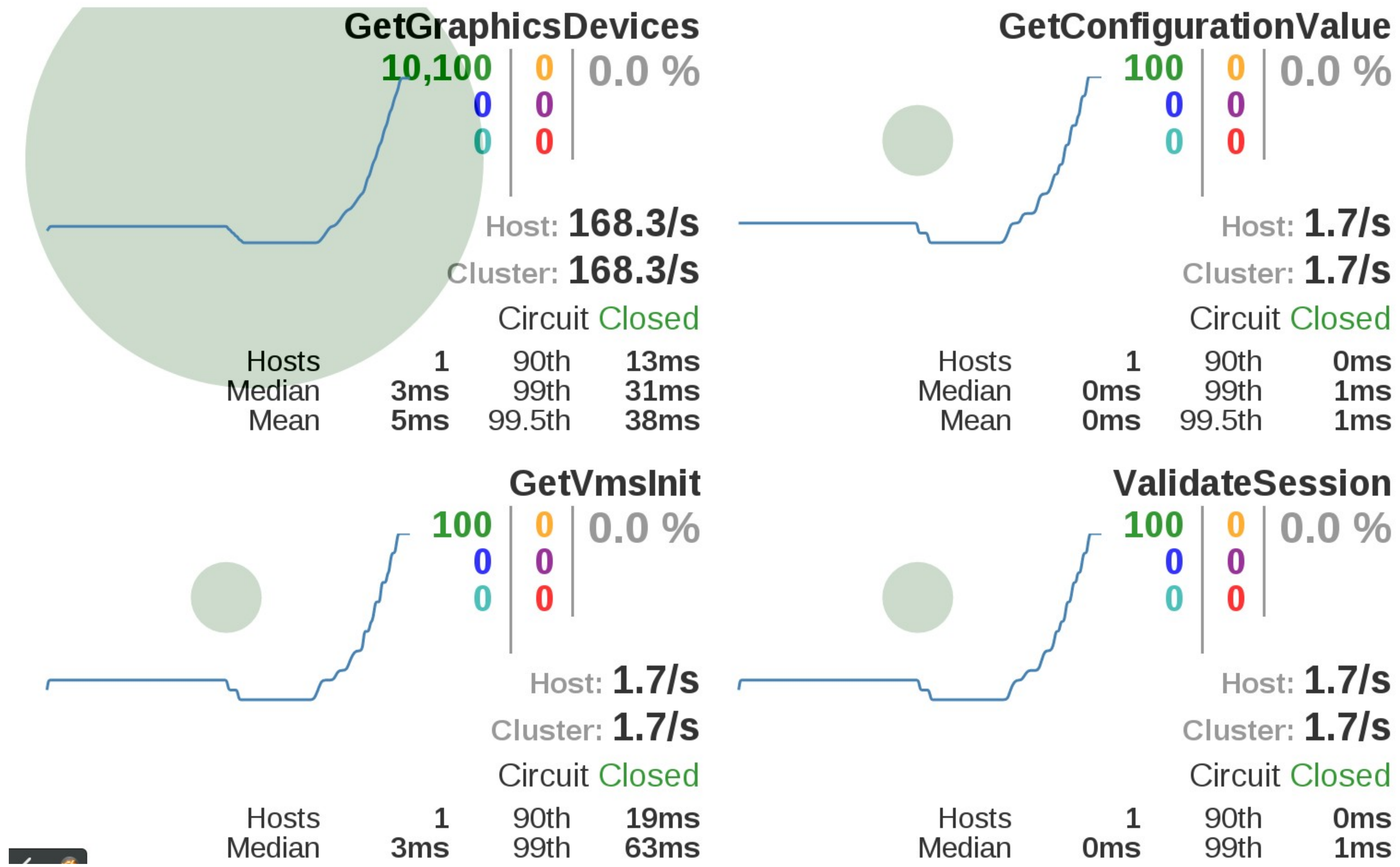
Solution: Let's curl a little bit more

Let us execute the following last scenario:

100 Vms, 10 parallel requests, 100 requests total

```
$> seq 1 100 | parallel -j 10 bash rest.sh vms > /dev/null
```

Find the Error



GetGraphicsDevices
10,100 | 0 | 0.0 %
0 | 0
0 | 0
Host: 168.3/s
Cluster: 168.3/s
Circuit Closed

```
Hosts        1     90th    13ms
Median     3ms     99th    31ms
Mean       5ms   99.5th    38ms
```

GetConfigurationValue
100 | 0 | 0.0 %
0 | 0
0 | 0
Host: 1.7/s
Cluster: 1.7/s
Circuit Closed

```
Hosts        1     90th     0ms
Median     0ms     99th     1ms
Mean       0ms   99.5th     1ms
```

GetVmsInit
100 | 0 | 0.0 %
0 | 0
0 | 0
Host: 1.7/s
Cluster: 1.7/s
Circuit Closed

```
Hosts        1     90th    19ms
Median     3ms     99th    63ms
```

ValidateSession
100 | 0 | 0.0 %
0 | 0
0 | 0
Host: 1.7/s
Cluster: 1.7/s
Circuit Closed

```
Hosts        1     90th     0ms
Median     0ms     99th     1ms
```

With the fix

1000 VMs, 1 request

```
$> time bash rest.sh vms > /dev/null
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 7051k    0 7051k    0     0  8521k      0 --:--:-- --:--:-- --:--:-- 8516k

real 0m1.008s
user0m0.091s
sys  0m0.042s
```

2000 VMs, 1 request

```
$> time bash rest.sh vms > /dev/null
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 13.7M    0 13.7M    0     0  7210k      0 --:--:--  0:00:01 --:--:-- 7210k

real 0m2.218s
user 0m0.079s
sys  0m0.062s
```

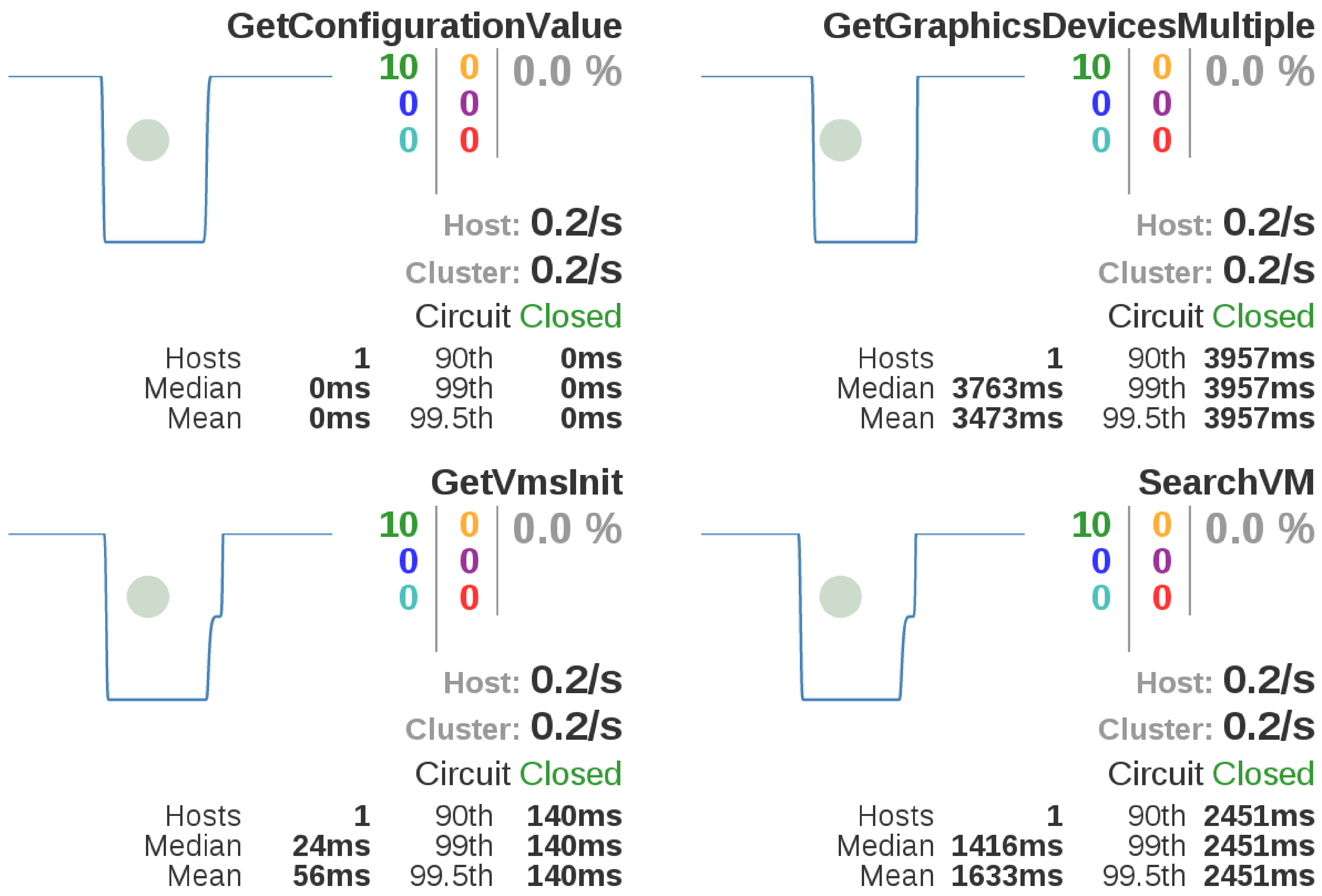
With the Fix

2000 VMs, 10 parallel requests

```
time seq 1 10 | parallel -j 10 bash rest.sh vms > /dev/null

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 13.7M    0 13.7M    0     0  1473k      0 --:--:--  0:00:09 --:--:-- 3249k
[...]
100 13.7M    0 13.7M    0     0  1534k      0 --:--:--  0:00:09 --:--:-- 3566k

real 0m10.228s
user0m0.211s
sys  0m0.436s
```

Much better but still too slow. We will see later how to avoid being overwhelmed by too much expensive calls.

With the Fix



```
GetConfigurationValue
10 | 0 | 0.0 %
 0 | 0 |
 0 | 0 |
Host: 0.2/s
Cluster: 0.2/s
Circuit Closed
 Hosts     1    90th  0ms
Median   0ms    99th  0ms
  Mean   0ms  99.5th  0ms
```

```
GetGraphicsDevicesMultiple
10 | 0 | 0.0 %
 0 | 0 |
 0 | 0 |
Host: 0.2/s
Cluster: 0.2/s
Circuit Closed
 Hosts        1    90th  3957ms
Median   3763ms    99th  3957ms
  Mean   3473ms  99.5th  3957ms
```

```
GetVmsInit
10 | 0 | 0.0 %
 0 | 0 |
 0 | 0 |
Host: 0.2/s
Cluster: 0.2/s
Circuit Closed
 Hosts      1    90th  140ms
Median   24ms    99th  140ms
  Mean   56ms  99.5th  140ms
```

```
SearchVM
10 | 0 | 0.0 %
 0 | 0 |
 0 | 0 |
Host: 0.2/s
Cluster: 0.2/s
Circuit Closed
 Hosts        1    90th  2451ms
Median   1416ms    99th  2451ms
  Mean   1633ms  99.5th  2451ms
```

Collect data from user systems

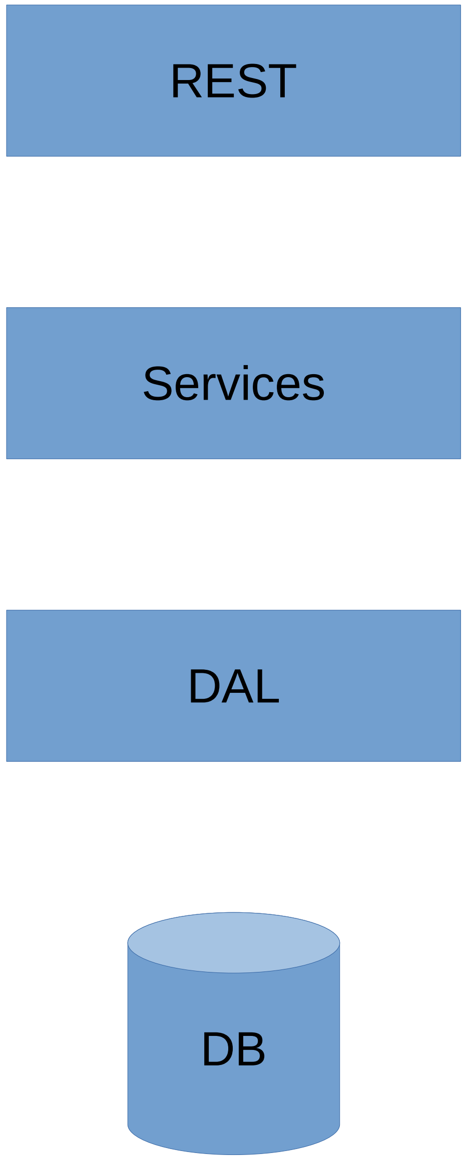
Collect as much streaming data as you want:

```
$> curl -H "Accept: application/json"
        -H "Content-type: application/json" -X GET
        --user admin@internal:engine
        http://localhost:8080/ovirt-engine/services/hystrix.stream

ping:

data: {"type":"HystrixCommand","name":"GetVmsInit","group":"GetVmsInit", [...] }
data: {"type":"HystrixCommand","name":"VdsHostDevListByCaps", [...] }
```

Import it later in your favourite analysis tool or send the data directly to it by using Hystrix plugins.

Defensive Programming

Simple Webshop



First try: ProductDTO

```
@Data
@NoArgsConstructor
public class ProductDTO {

  Long id;

  Integer amount;
}
```

Second try: ProductDTO

```
@Data
@NoArgsConstructor
public class ProductDTO {

●  @NotNull
●  @Min(0)
   Long id;

●  @NotNull
●  @Min(1)
   Integer amount;
}
```

First try: REST Resource

```
@Path("/cart")
public class ShoppingCartResourceBad extends BasicResource {
  @Inject ShoppingCartService cartService;
  @Inject ProductDao productDao;
  @Inject ShoppingCartDao cartDao;

  @POST
  @Path("/{id}/product")
  public Response addProduct(
      @PathParam("id") final Long id,
      final ProductDTO productDTO
  ) {
    User user = getUser();
    Product product = productDao.get(productDTO.getId());
    ShoppingCart cart = cartDao.get(id);
    Integer amount = productDTO.getAmount();
    if (cartService.addItem(cart, product, amount, user)) {
      return Response.status(OK).build();
    } else {
      return Response.status(OK).entity(new OutOfProductError()).build();
    }
  }
```

Second try: REST Resource

oVirt

```
@Path("/cart")
public class ShoppingCartResourceGood extends BasicResource {
    @Inject ShoppingCartService cartService;
    @Inject ProductDao productDao;
    @Inject ShoppingCartDao cartDao;

    @POST
  ● @Path("/{id:\\d+}/product")
  ● @UserRequired
  ● public Response addProduct(
  ●         @PathParam("id") @NotNull @Min(0) final Long id,
            @NotNull @Valid final ProductDTO productDTO)
  ● {
  ●     User user = checkExists(getUser());
  ●     Product product = checkExists(productDao.get(productDTO.getId()));
        ShoppingCart cart = checkExists(cartDao.get(id));
        Integer amount = productDTO.getAmount();
        if (cartService.addItem(cart, product, amount, user)) {
            return Response.status(OK).build();
        } else {
            return Response.status(OK).entity(new OutOfProductError()).build();
        }
    }
```

First try: ShoppingCartService

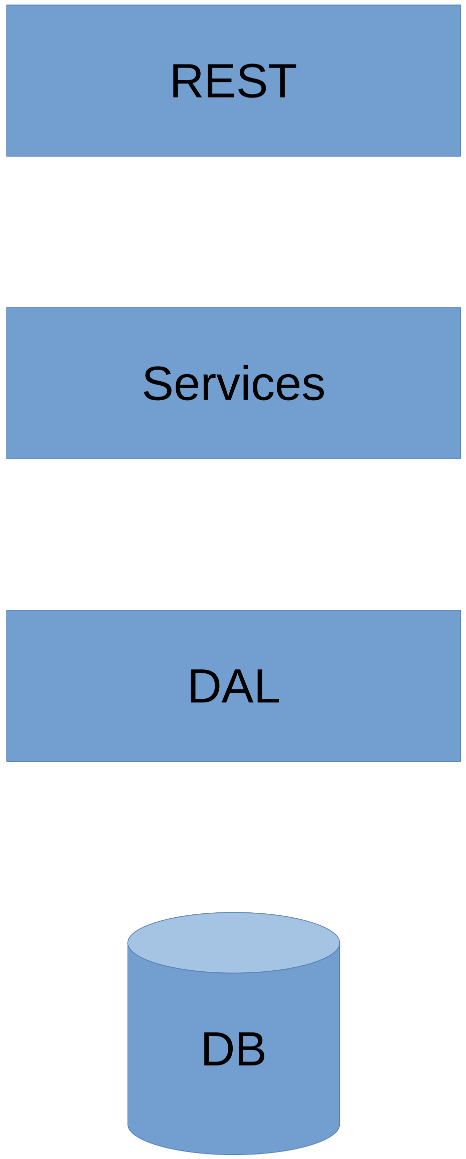
oVirt

```
public class ShoppingCartServiceBad implements ShoppingCartService {
  @Inject ShoppingCartDao cartDao;
  @Inject EventBus bus;
  @Inject AuditLog auditLog;
  @Inject Store store;

  @Transactional
  public boolean addItem(ShoppingCart cart,
      Product product,
      Integer amount,
      User responsible) {
    if (!store.hasItems(product, amount)) {
      return false;
    }
    ShopItem shopItem = new ShopItem(product, amount);
    cart.addItem(shopItem);
    cartDao.save(cart);
    bus.post(new ShoppItemAddedEvent(cart, product, amount, responsible));
    auditLog.log(Action.ITEM_ADDED, cart, shopItem, responsible);
    return true;
  }
}
```

Second try: SchoppingCartService

```
public class ShoppingCartServiceGood implements ShoppingCartService {
  @Inject ShoppingCartDao cartDao;
  @Inject EventBus bus;
  @Inject AuditLog auditLog;
  @Inject Store store;

  @Transactional
  public boolean addItem(ShoppingCart cart,
      Product product,
      Integer amount,
      User responsible) {
●   checkNotNull(cart);
●   checkArgument(amount > 0, "Amount is %s but must be greater than 0", amount);
●   checkNotNull(responsible);
    if (!store.hasItems(product, amount)) {
      return false;
    }
    ShopItem shopItem = new ShopItem(product, amount);
    cart.addItem(shopItem);
    cartDao.save(cart);
    bus.post(new ShoppItemAddedEvent(cart, product, amount, responsible));
    auditLog.log(Action.ITEM_ADDED, cart, shopItem, responsible);
    return true;
  }
```

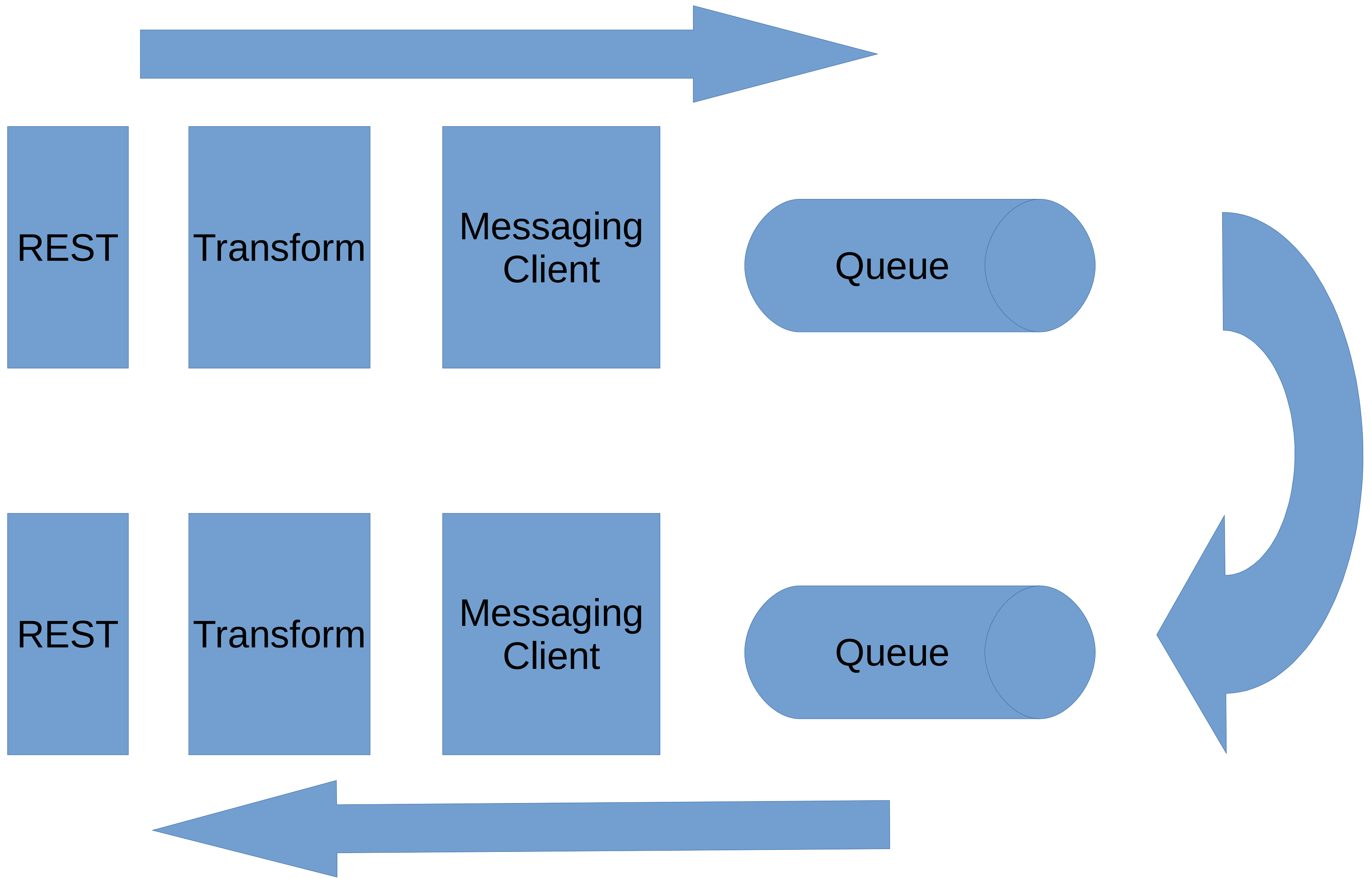
Simple Webshop Monolith again



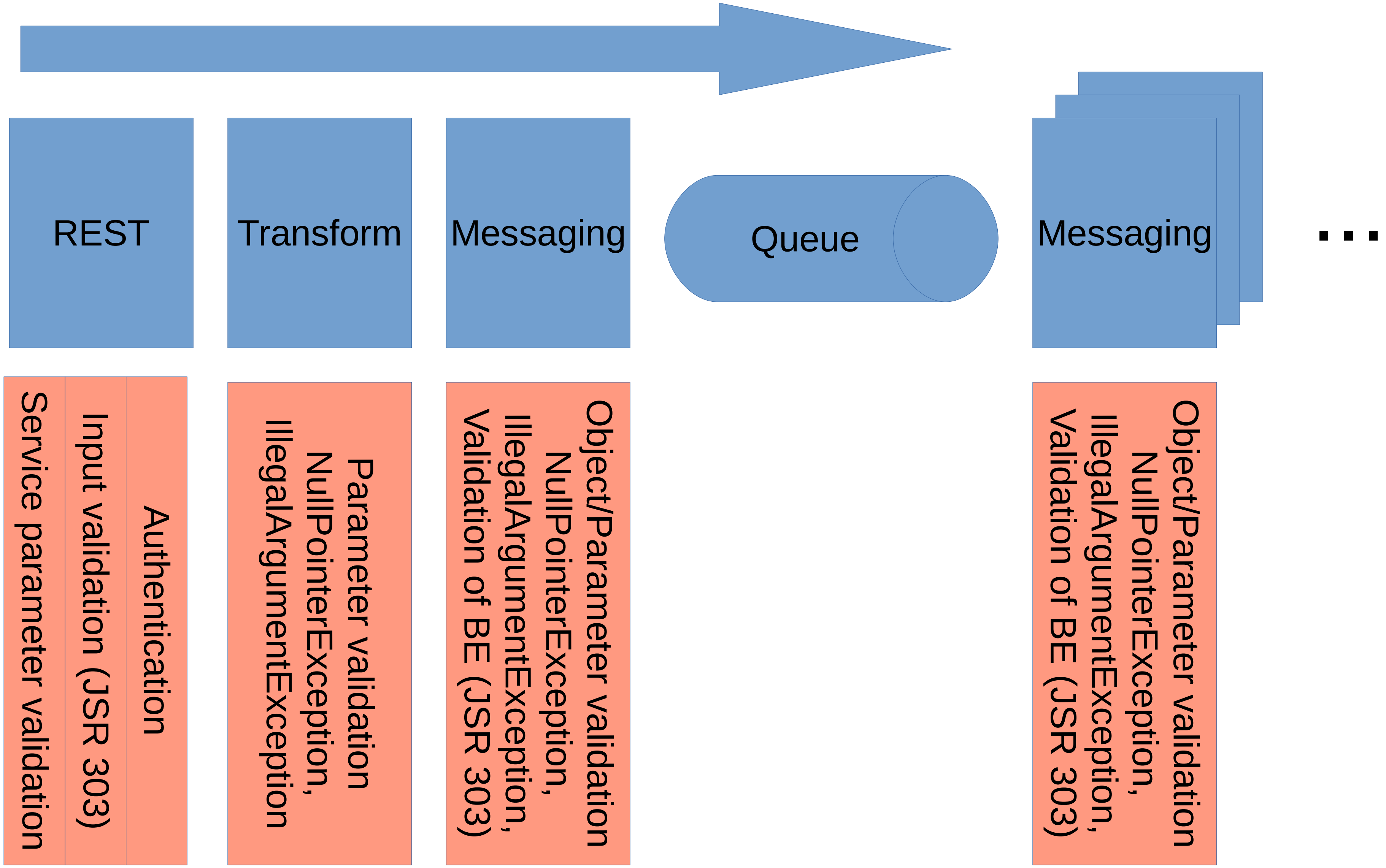
Defensive Monolith

REST	Authentication	401
	Input validation (JSR 303)	400
	Service parameter validation	400, 404, 403
Services	Service parameter validation NullPointerException, IllegalArgumentException	500
DAL	DAL parameter validation NullPointerException, IllegalArgumentException, Validation of BE (JSR 303)	500
DB	DB constraints	500

Microservice



Defensive Microservice


REST
Transform
Messaging
Queue
Messaging
...
Service parameter validation
Input validation (JSR 303)
Authentication
Parameter validation
NullPointerException,
IllegalArgumentException
Object/Parameter validation
NullPointerException,
IllegalArgumentException,
Validation of BE (JSR 303)
Object/Parameter validation
NullPointerException,
IllegalArgumentException,
Validation of BE (JSR 303)

Resiliency

Timeouts

oVirt

- Set all timeouts you can find
- Thread pools can be out of threads
- Test your services against something like Bane*
- Add automated tests (e.g. with Bane, Netty, Untertow)

* https://github.com/danielwellman/bane

Circuit Breakers

- Don't trust external services
- Don't trust SDKs
- Don't trust internal services
- Test your services with something like Bane
- Add automated tests (e.g. with Bane, Netty, Undertow)
- Throttle expensive calls

Further reading

Relase It!

Design and Deploy
Production-Ready Software

by Michael T.Nygard

ovirt-engine

Under Siege

`Take 3`

Making ovirt-engine more resilient

- Hystrix is a curcuit breaker
- We can configure Hystrix with Arachaius
- Arachaus accepts different configuration sources
  - System properties
  - Properties files
  - Zookeeper
  - Etcd
  - JMX
  - ...

Archaius: config.properties

oVirt

How to limit concurrent invocations of a command protected by Hystrix:

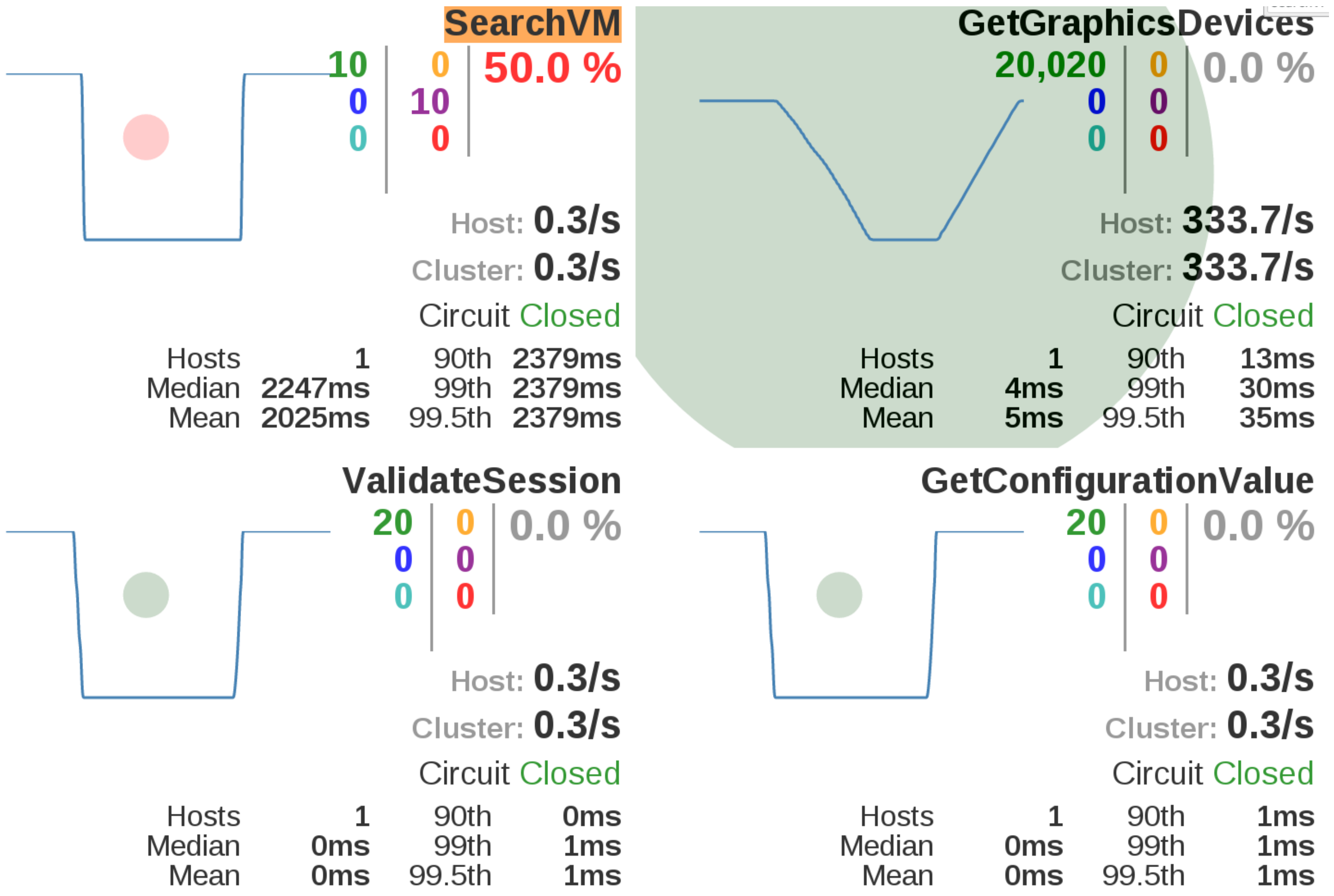
Default Value	`10`
Default Property	`hystrix.command.default.execution.isolation.semaphore .maxConcurrentRequests`
Instance Property	`hystrix.command.`*`HystrixCommandKey`*`.execution.isolation .semaphore.maxConcurrentRequests`
How to Set Instance Default	HystrixCommandProperties.Setter() .withExecutionIsolationSemaphoreMaxConcurrentRequests(int value)

This will override the default configuration of the **SearchVM** command:

```
hystrix.command.SearchVM.execution.isolation.semaphore.maxConcurrentRequests=10
```

Just add it to your config.properties file or set it as system property.

Protected VMs Endpoint



SearchVM
10 | 0 | 50.0 %
0 | 10
0 | 0

Host: 0.3/s
Cluster: 0.3/s
Circuit Closed

```
Hosts       1     90th  2379ms
Median 2247ms     99th  2379ms
Mean   2025ms   99.5th  2379ms
```

GetGraphicsDevices
20,020 | 0 | 0.0 %
0 | 0
0 | 0

Host: 333.7/s
Cluster: 333.7/s
Circuit Closed

```
Hosts       1     90th  13ms
Median    4ms     99th  30ms
Mean      5ms   99.5th  35ms
```

ValidateSession
20 | 0 | 0.0 %
0 | 0
0 | 0

Host: 0.3/s
Cluster: 0.3/s
Circuit Closed

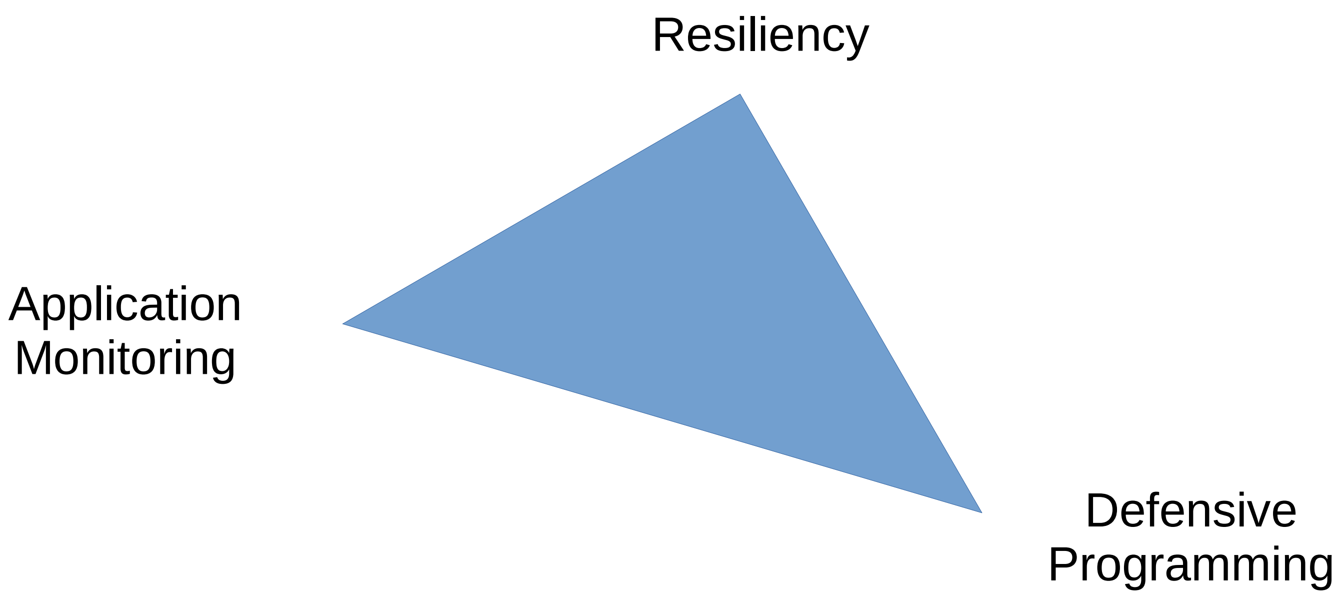
```
Hosts       1     90th  0ms
Median    0ms     99th  1ms
Mean      0ms   99.5th  1ms
```

GetConfigurationValue
20 | 0 | 0.0 %
0 | 0
0 | 0

Host: 0.3/s
Cluster: 0.3/s
Circuit Closed

```
Hosts       1     90th  1ms
Median    0ms     99th  1ms
Mean      0ms   99.5th  1ms
```

Conclusion


Resiliency
Application Monitoring
Defensive Programming