Performance Challenges along the Continuous Delivery Pipeline

Wolfgang Gottesheim
@gottesheim
Example of a “Bad” Web Deployment

<table>
<thead>
<tr>
<th>Browser</th>
<th>First Impression Time</th>
<th>OnLoad Time</th>
<th>Total Load Time</th>
<th>Client Time</th>
<th>Rendering Time</th>
<th>Javascript Time</th>
<th>Total Bytes</th>
<th>Caching Ratio</th>
<th>HTTP Request Count</th>
<th>Page Size</th>
<th>Connect Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF 32</td>
<td>1.4 sec</td>
<td>8.7 sec</td>
<td>8.8 sec</td>
<td>1.15 sec</td>
<td>23</td>
<td>690</td>
<td>9.66 MB</td>
<td>99%</td>
<td>45</td>
<td>282!</td>
<td>1.8s</td>
</tr>
<tr>
<td>FF 32</td>
<td>351</td>
<td>2816</td>
<td>2820</td>
<td>376</td>
<td>22</td>
<td>2016</td>
<td>2.07 MB</td>
<td>43%</td>
<td>58</td>
<td>282!</td>
<td></td>
</tr>
<tr>
<td>FF 32</td>
<td>448</td>
<td>1190</td>
<td>1203</td>
<td>360</td>
<td>5</td>
<td>511</td>
<td>1.07 MB</td>
<td>21%</td>
<td>45</td>
<td>282!</td>
<td></td>
</tr>
<tr>
<td>FF 32</td>
<td>380</td>
<td>2458</td>
<td>2438</td>
<td>304</td>
<td>33</td>
<td>324</td>
<td>7.02 MB</td>
<td>45%</td>
<td>171</td>
<td>282!</td>
<td></td>
</tr>
</tbody>
</table>

Most objects are images delivered from your main domain.

Very long Connect time (1.8s) to your CDN.
Application problems in the wild

526s to render a financial transaction report

1 SQL running 210s!

Debug Logging with log4j on outdated log4j library (sync issue)
But now we’re doing Continuous Delivery
700 Deployments / Year
It’s not about the pipeline...
Commit Stage → Acceptance Testing → Capacity Testing → UAT → Release

Developer(s)
Metrics to the rescue!
We want to replace magic with data!
4 examples
Mobile Landing Page of Super Bowl Ad

Find all Key Performance Indicators (KPI) for the selected page

These values help you compare with other versions of the same page to identify problems or regressions.

Learn more on Key Performance Indicators and how they get calculated.

<table>
<thead>
<tr>
<th>First Request</th>
<th>On Server</th>
<th>DNS</th>
<th>Network</th>
<th>Total Size</th>
<th>Total Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>6ms</td>
<td>5859ms</td>
<td>0ms</td>
<td>16307ms</td>
<td>~20MB</td>
<td>20814kb</td>
</tr>
<tr>
<td>First Impression Time</td>
<td>277ms</td>
<td>On Client</td>
<td>Connect</td>
<td>3268ms</td>
<td># of Requests</td>
</tr>
<tr>
<td>Onload Time</td>
<td>0ms</td>
<td>Transfer</td>
<td>Rendering</td>
<td>3402ms</td>
<td># of XHR</td>
</tr>
<tr>
<td>2018ms</td>
<td>Ø Interactive</td>
<td>Ø Wait</td>
<td>Ø Wait</td>
<td>1973ms</td>
<td>15065ms</td>
</tr>
</tbody>
</table>

Total size of ~ 20MB

434 Resources in total on that page:
- 230 JPEGs
- 75 PNGs
- 50 GIFs
- ...
8MB background image for STPCon
Key Metrics

# Resources
Size of Resources
Page Size
#2
#1: Loading too much data

24889! Calls to the Database API!

High CPU and High Memory Usage to keep all data in Memory
#2: On individual connections

12444! individual connections (and no pooling...)

Individual SQL really <1ms

Classical N+1 Query Problem
#3: Homegrown Caching Framework

Lots of time spent in `Hashtable.get`

Called from their Entity Objects
Key Metrics

# Functional Errors
# of SQL Calls
# of same SQL Execs (1+N)
# of Connections
Rows/Data Transferred
#3

Simply splitting web service calls is not a migration to a microservice architecture.
3136! Calls to H2 mostly executed on async background threads

40! internal Web Service Calls that do all these DB Updates

21671! Calls to Oracle

33! Different connections used

DB Exceptions on both Databases
Key Metrics

# Service Calls
# of Threads
Timing (execution, wait)
# SQL executions
# of SAME SQLs
Is this a successful new Build?

Response Time pattern similar between two builds

Deploy New Version at 12:00
Jump in Count caused by queued requests during restart

Application is able to handle the same load pattern
Look at Resource Usage: CPU, Memory, ...
Memory? Look at Heap Generations

- # Gen 0
- # Gen 1
- % spent in GC
Root Cause: Dependency Injection

<table>
<thead>
<tr>
<th>Controller Object</th>
<th>41</th>
<th>827.52 KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>PlanningController@156367</td>
<td>-</td>
<td>20.79 KB</td>
</tr>
<tr>
<td>PlanningController@156634</td>
<td>-</td>
<td>20.79 KB</td>
</tr>
<tr>
<td>PlanningController@141107</td>
<td>-</td>
<td>20.79 KB</td>
</tr>
<tr>
<td>PlanningController@141292</td>
<td>-</td>
<td>20.79 KB</td>
</tr>
<tr>
<td>PlanningController@139872</td>
<td>-</td>
<td>20.79 KB</td>
</tr>
<tr>
<td>PlanningController@149049</td>
<td>-</td>
<td>20.79 KB</td>
</tr>
<tr>
<td>PlanningController@165246</td>
<td>-</td>
<td>20.79 KB</td>
</tr>
<tr>
<td>PlanningController@165033</td>
<td>-</td>
<td>20.79 KB</td>
</tr>
<tr>
<td>PlanningController@150008</td>
<td>-</td>
<td>20.79 KB</td>
</tr>
</tbody>
</table>

**Before Deploy:** 41 Controller Objects with Average size of 20.79kB

<table>
<thead>
<tr>
<th>Controller Object</th>
<th>78</th>
<th>8.34 MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>PlanningController@237226</td>
<td>-</td>
<td>116.62 KB</td>
</tr>
<tr>
<td>PlanningController@126343</td>
<td>-</td>
<td>116.62 KB</td>
</tr>
<tr>
<td>PlanningController@420545</td>
<td>-</td>
<td>116.62 KB</td>
</tr>
<tr>
<td>PlanningController@415810</td>
<td>-</td>
<td>116.62 KB</td>
</tr>
<tr>
<td>PlanningController@368681</td>
<td>-</td>
<td>116.62 KB</td>
</tr>
<tr>
<td>PlanningController@316898</td>
<td>-</td>
<td>116.62 KB</td>
</tr>
<tr>
<td>PlanningController@105447</td>
<td>-</td>
<td>116.62 KB</td>
</tr>
<tr>
<td>PlanningController@277077</td>
<td>-</td>
<td>116.62 KB</td>
</tr>
</tbody>
</table>

**After Deploy:** ~6x larger Object size and ~2x as many objects on the heap
Prevent: Monitor Memory Metrics for every Build
Key Metrics

# of Objects per Generation
# of GC Runs
Total Impact of GC
Where to start?

1. # Resources
2. Size of Resources
3. Page Size
4. # Functional Errors
5. # SQL Executions
Allow me to introduce…

Pick
Agree
Measure
Integrate
Available Tools

Mission Control

VisualVM

New Relic

JProbe

YourKit

AppDynamics

JProfiler

Solaris Studio

NetBeans Profiler

Honest Profiler

Dynatrace

XRebel

Kieker

AND MANY MORE
We successfully replaced magic with data!
Just one more thing...
It’s also about the pipeline...

→ Define pipeline metrics!
## Test Framework Results

<table>
<thead>
<tr>
<th>Build #</th>
<th>Test Case</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build 17</td>
<td>testPurchase</td>
<td>OK</td>
</tr>
<tr>
<td></td>
<td>testSearch</td>
<td>OK</td>
</tr>
<tr>
<td>Build 18</td>
<td>testPurchase</td>
<td>FAILED</td>
</tr>
<tr>
<td></td>
<td>testSearch</td>
<td>OK</td>
</tr>
<tr>
<td>Build 19</td>
<td>testPurchase</td>
<td>OK</td>
</tr>
<tr>
<td></td>
<td>testSearch</td>
<td>OK</td>
</tr>
<tr>
<td>Build 20</td>
<td>testPurchase</td>
<td>OK</td>
</tr>
<tr>
<td></td>
<td>testSearch</td>
<td>OK</td>
</tr>
</tbody>
</table>

## Architectural Data

<table>
<thead>
<tr>
<th># SQL</th>
<th># Excep</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>0</td>
<td>120ms</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>68ms</td>
</tr>
<tr>
<td>12</td>
<td>5</td>
<td>60ms</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>68ms</td>
</tr>
<tr>
<td>75</td>
<td>0</td>
<td>230ms</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>68ms</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>120ms</td>
</tr>
</tbody>
</table>

We identified a regression problem solved

Now we have the functional and architectural confidence
#1: Analyzing each Test

#2: Metrics for each Test

#3: Detecting Regression based on Measure
Quality-Metrics based Build Status

Details for all builds

Unit Tests

Performance Tests

Web API Tests

Analyze Load Test Results
Analyze the synthetic transactions of a load test and quickly identify your performance and scalability bottlenecks. Learn more about integrating load test tools with Dynatrace.

Integrate dynaTrace in your build process
Integrate dynaTrace with JUnit, Maven, Jenkins, ... Learn more about integrating dynaTrace in all stages of your delivery pipeline.
Pull data into Jenkins, Bamboo...
Make Quality a first-class citizen

„Too hard“

„not cool enough“

„we‘ll get round to this later“
Wolfgang Gottesheim


Twitter @gottesheim

wolfgang.gottesheim@dynatrace.com