Performance Test Automation with AWS Code Pipeline, Blazemeter, & Dynatrace AppMon

CMG Boston – April 2017

Joe Sicree

Sr. Consultant @ CGI

@JoeSicree - Joseph.Sicree@cgi.com

CGI

Rob Jahn

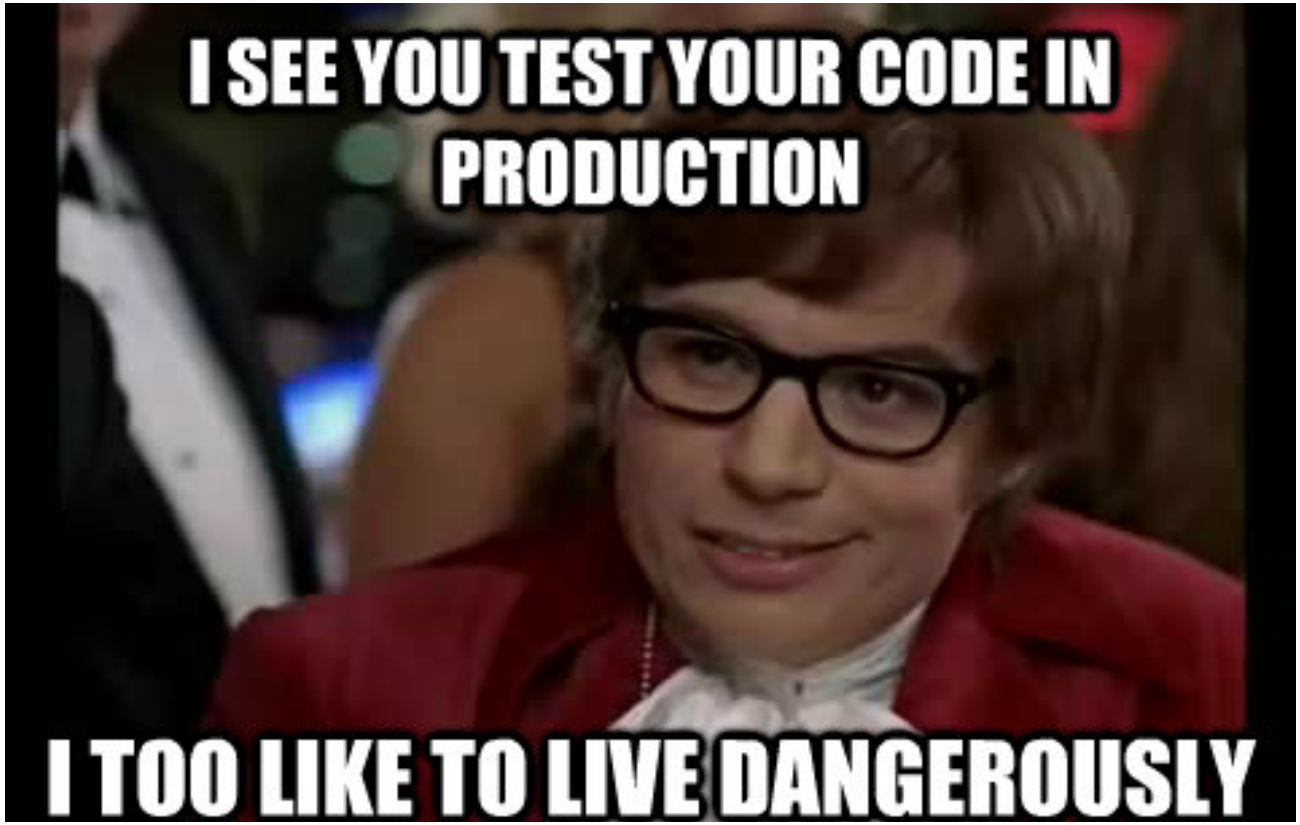
Technical Director @ Elyxor

Rob.Jahn@elyxor.com

elyxor

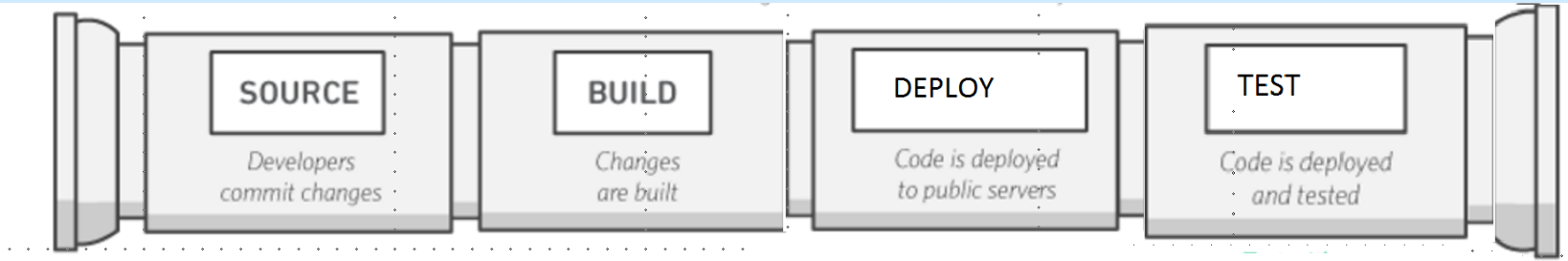
Agenda

- Introduction
- What is AWS CodePipeline?
- How does Blazemeter and Dynatrace AppMon fit in?
- Live Demo: 3 Performance Engineering Use Cases
- Q&A


I SEE YOU TEST YOUR CODE IN PRODUCTION
I TOO LIKE TO LIVE DANGEROUSLY

What is AWS CodePipeline?

Amazon CodePipeline is a continuous integration and continuous delivery service for fast and reliable application and infrastructure updates. CodePipeline builds, tests, and deploys your code every time there is a code change, based on the release process models you define.



3 Minute Explainer Video on https://aws.amazon.com/codepipeline/

Why a CI/CD pipeline?

Adoption of DevOps Principals

- Many builds with small change sets
- Quick feedback loops
- Continuous Integration & Deployments

Implications

- More regression testing (**including performance!!**)
- Automation a must
- Flexible tooling and infrastructure

Enablers

- IaaS
- Continuous Integration (Pipelines)
- SaaS testing and monitoring tools
- Automatic test analysis across builds

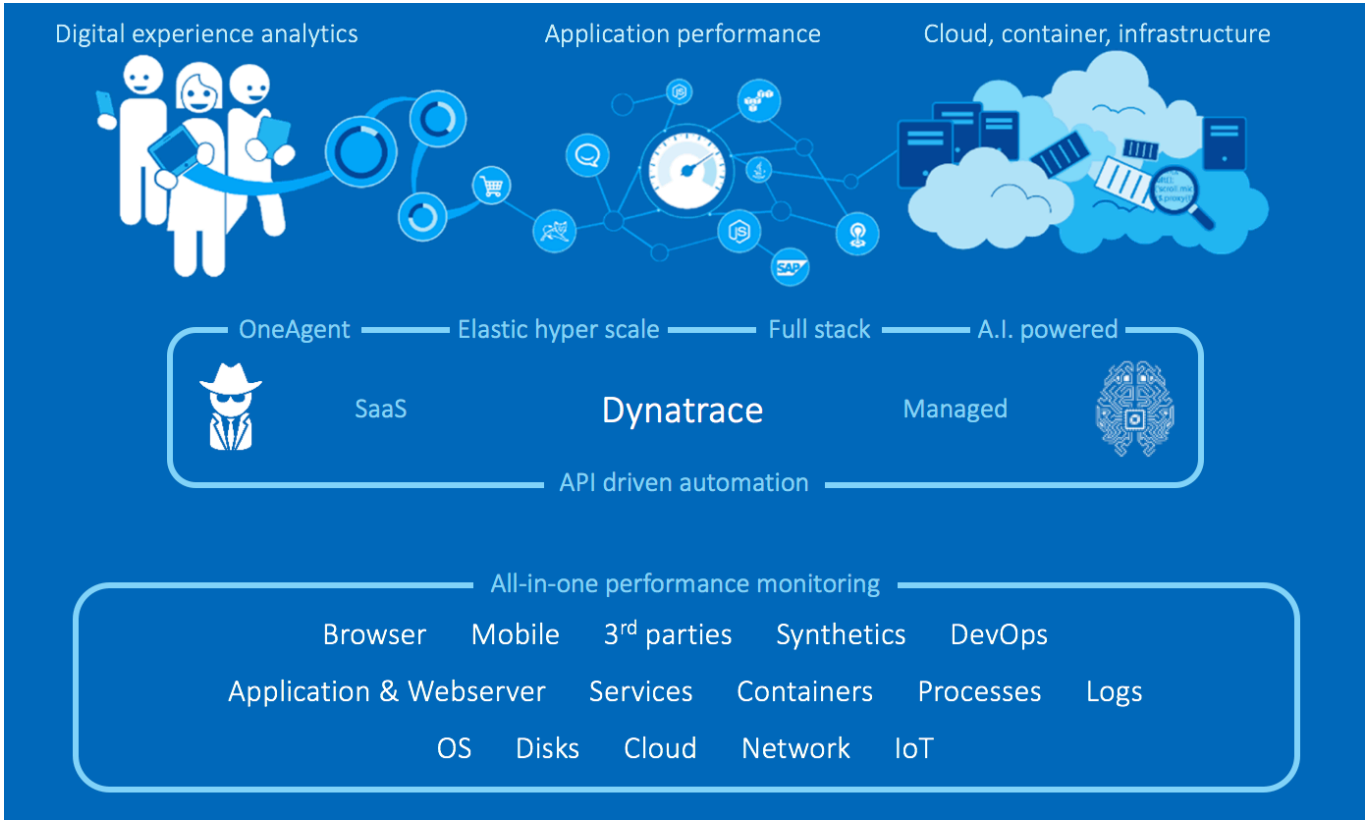
Benefits

- Low cost and on-demand test environment
- Find issues earlier in the SDLC
- Prevent issues getting to production
- Empower team

What is Blazemeter?

Learn more at https://www.blazemeter.com

What is Dynatrace?



Learn more at https://www.dynatrace.com

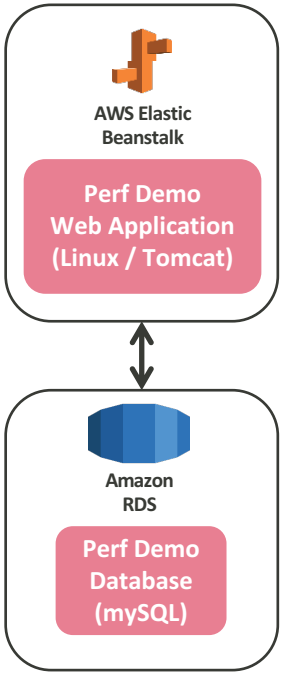

BRACE YOURSELVES
A LIVE DEMO IS
COMING

Demo Overview

- The following demo will walk us through a simple pipeline for a web application called Perf-Demo.
- The demo will focus on the following use cases:
  - Missed Service Level Agreements (SLA) for a mix of transactions
  - Memory leaks
  - Poor performance when accessing a database
- The demo will use a simple web application to demonstrate these use cases.

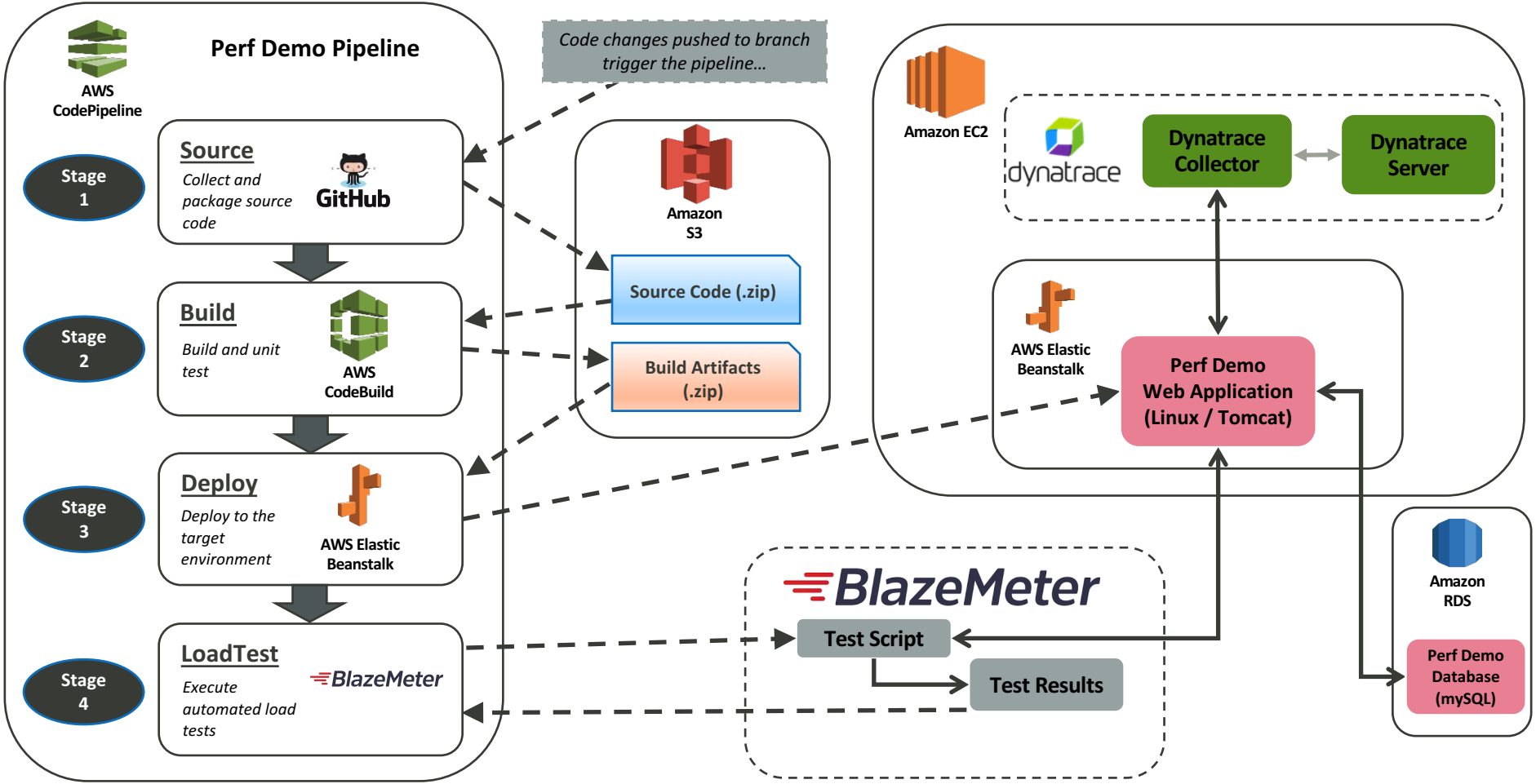
Demo Application

- Java Spring web application that provides a number of REST web services for use in performance testing.
- The web application provides the following services:
  - **calculateCompoundInterest** – Calculates compound interest for one or more accounts over a specified time period.
  - **addDataToMap** – Populates a hash map with data using a simple key
  - **getFinancialRecordsByDate** – Fetches financial data for one or more stocks on a specified date.



Get the code at https://github.com/jsicree/perf-demo

Demo Environment



Demo #1 – Slow Services

The Service

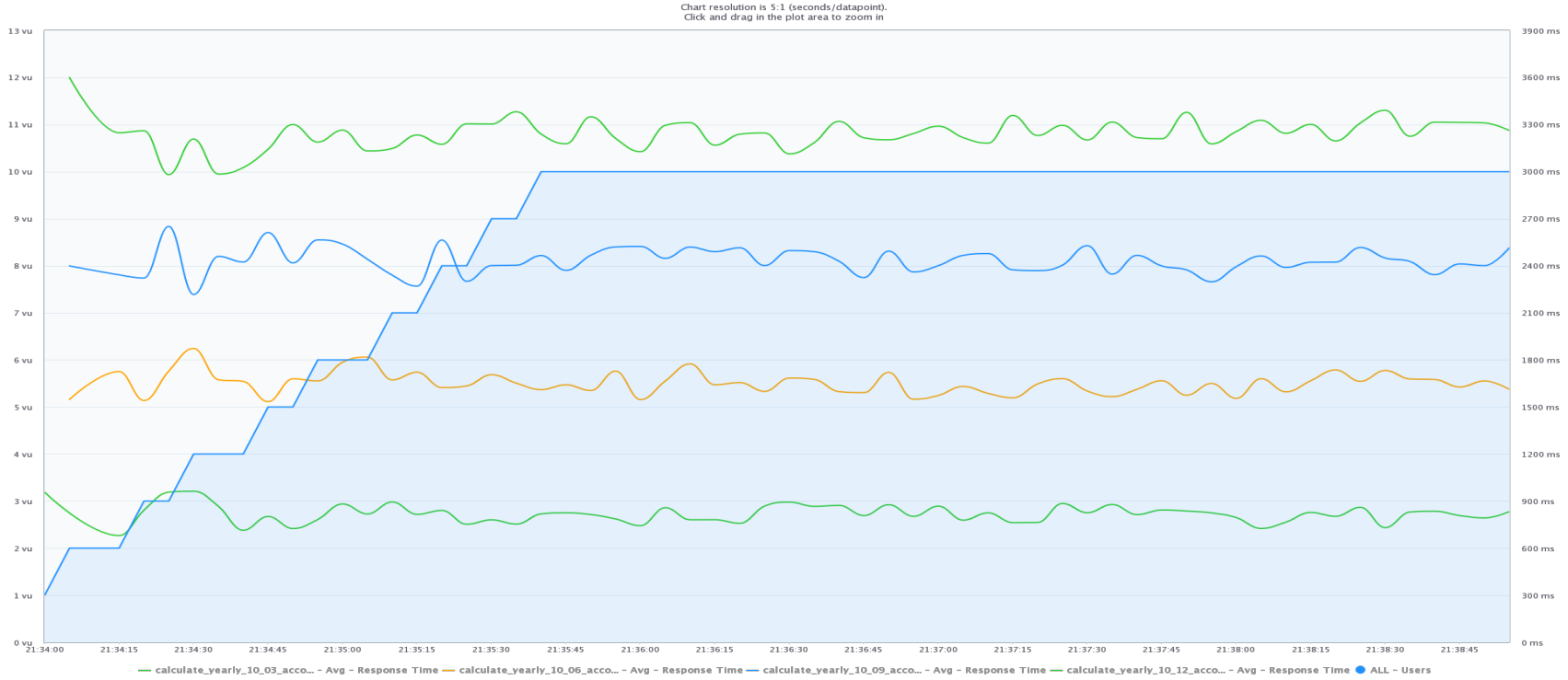
The computeCompoundInterest service computes the compound interest for a collection of accounts for a requested number of monthly or yearly intervals.

The Problem

The initial load test for the service using a mix of 3, 6, 9 and 12 accounts showed that the average response time was above the expected SLA for the requests with more accounts.

Demo #1 – Slow Services (V1 Test)

Exceeding 1 Second SLA for Response Time



Demo #1 – Slow Services (cont.)

The Analysis

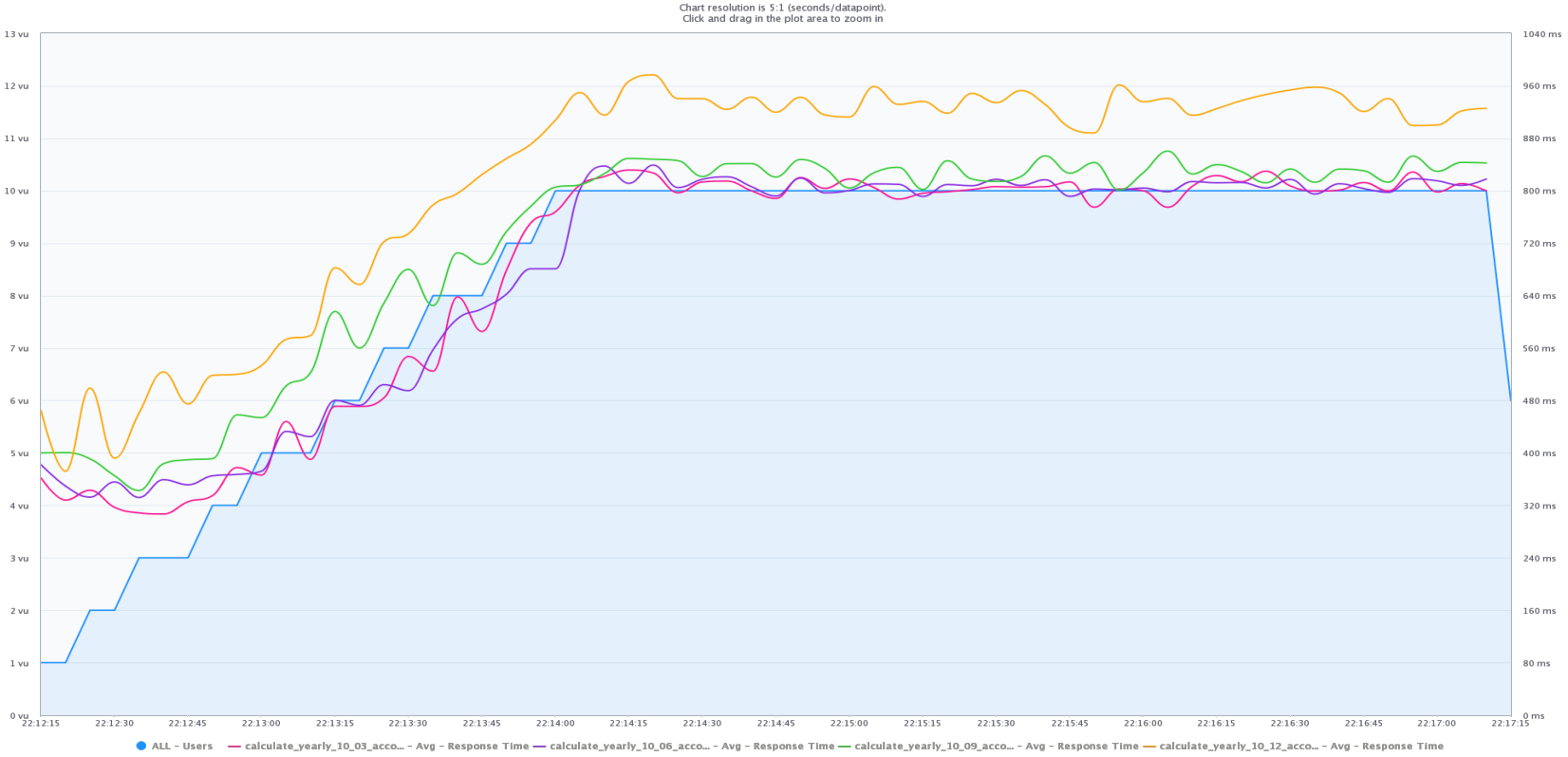
The underlying service was implemented synchronously. The more accounts included in the request, the longer the web service call took.

The Solution

The service was refactored to run asynchronously using threads.

Demo #1 – Slow Services (v2 Test)

All calls are under the 1 second SLA



Demo #2 – Memory Leak

The Service

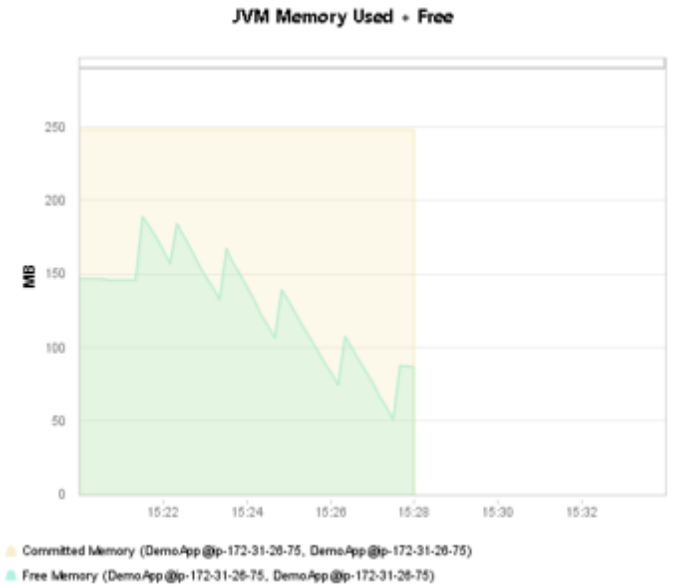
The service addDataToMap adds a specified block of bytes to a hash map using a simple key.

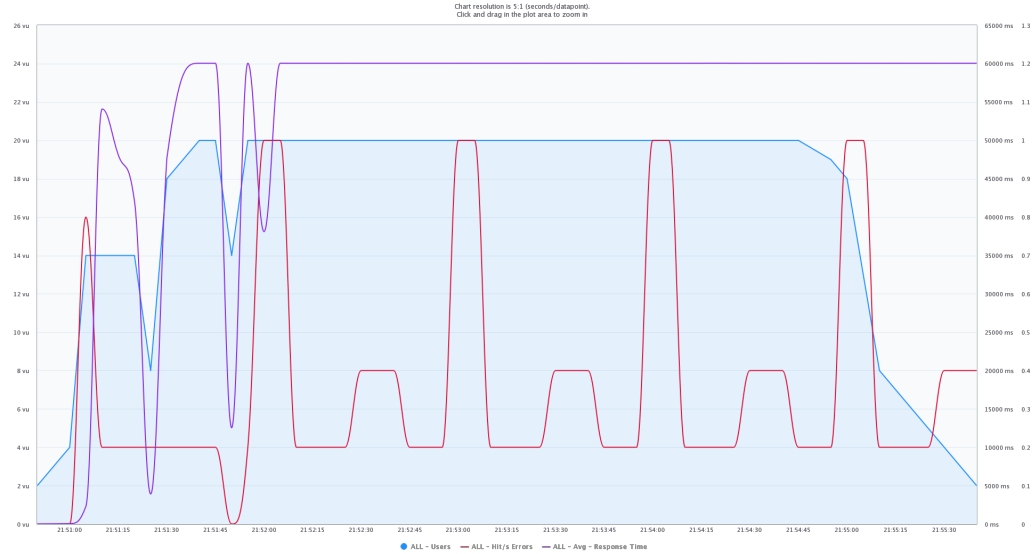
The Problem

The initial load test for the service showed that memory usage continuously increased, eventually causing an OutOfMemory error.

Demo #2 – Memory Leak (V1 Test)

Out of Memory caused server to crash. Timeout of request as a result





Demo #2 – Memory Leak (Cont.)

The Analysis

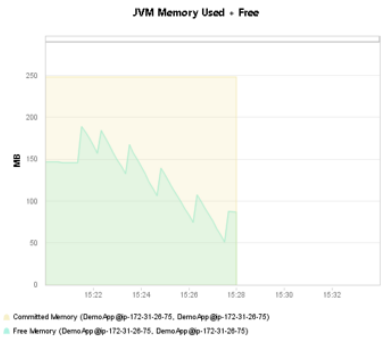
Service was implemented with a key class that did not implement an equals() method. As data was added to the map by each service call, values in the hash map were not being overwritten as intended. Instead, the new values were being appended to the map. This resulted in a memory leak that eventually led to OutOfMemory errors.

The Solution

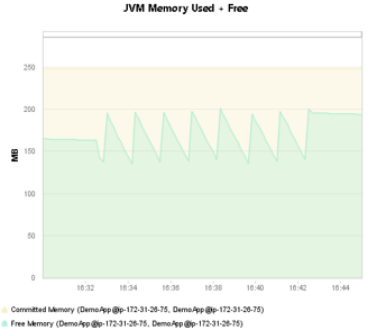
The bad key class was refactored.

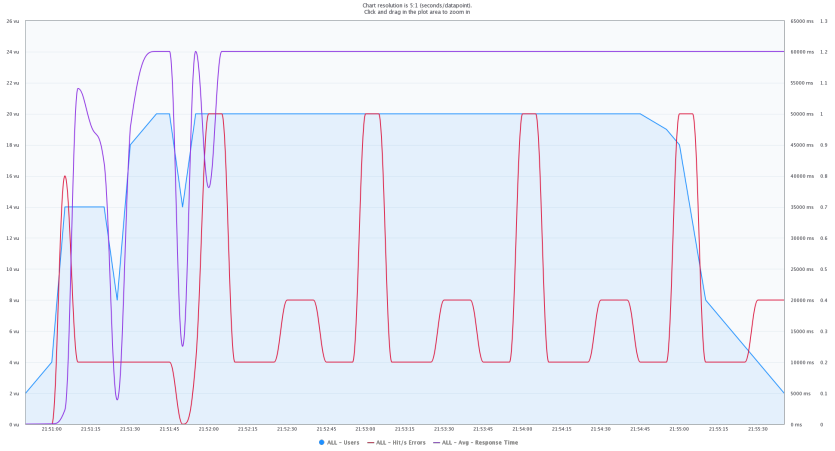
Demo #2 – Memory Leak (V2 Test)

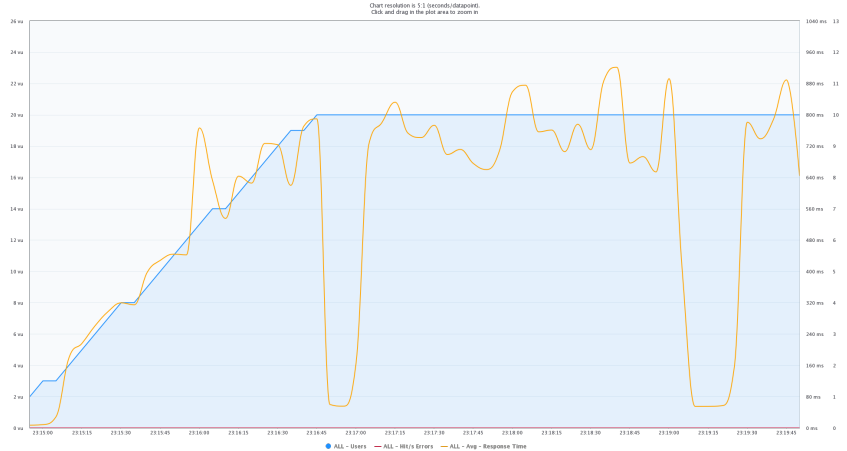
Before (v1)



After (v2)







Demo #3 – Poor DB Performance

The Service

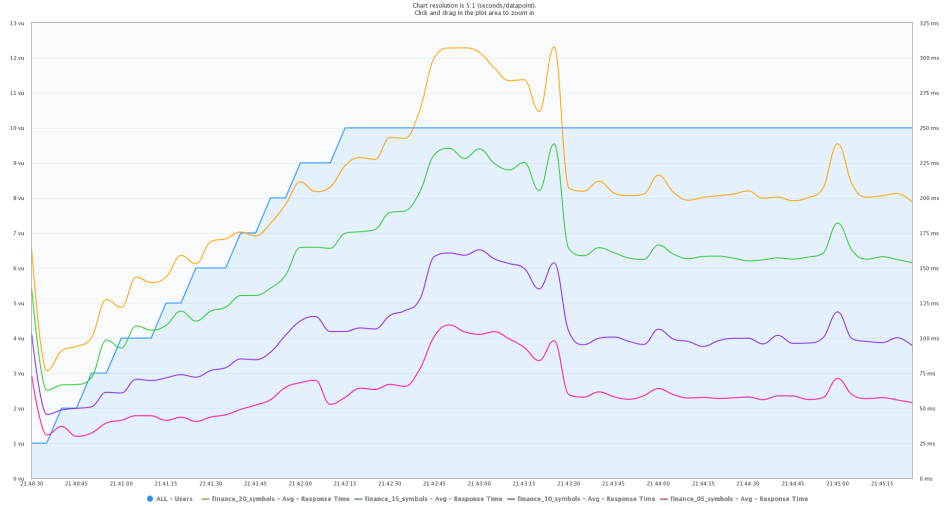
The service getFinancialRecordsByDate fetches financial data for a specified list of stock symbols for a specified date from database tables

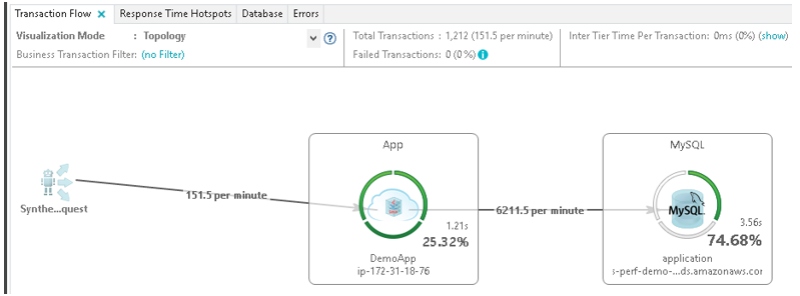
The Problem

The initial load test for the service showed that there were a large number of database calls for each service call.

Demo #3 – Poor DB Performance (V1 Test)

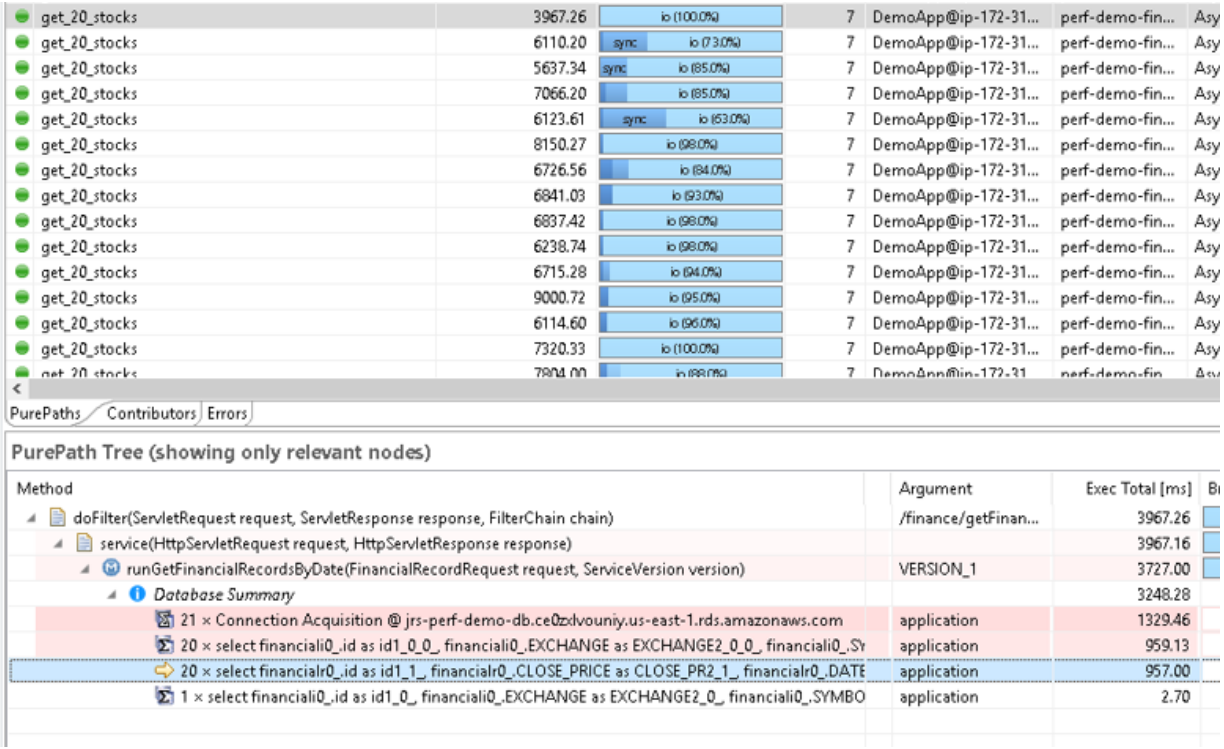
Poor Response times and much of the time in the database





Demo #3 – Poor DB Performance (V1 Test)

41 SQL calls Per Service call.



get_20_stocks	3967.26	io (100.0%)	7	DemoApp@ip-172-31...	perf-demo-fin...	Asy
get_20_stocks	6110.20	sync io (73.0%)	7	DemoApp@ip-172-31...	perf-demo-fin...	Asy
get_20_stocks	5637.34	sync io (85.0%)	7	DemoApp@ip-172-31...	perf-demo-fin...	Asy
get_20_stocks	7066.20	io (85.0%)	7	DemoApp@ip-172-31...	perf-demo-fin...	Asy
get_20_stocks	6123.61	sync io (63.0%)	7	DemoApp@ip-172-31...	perf-demo-fin...	Asy
get_20_stocks	8150.27	io (98.0%)	7	DemoApp@ip-172-31...	perf-demo-fin...	Asy
get_20_stocks	6726.56	io (84.0%)	7	DemoApp@ip-172-31...	perf-demo-fin...	Asy
get_20_stocks	6841.03	io (93.0%)	7	DemoApp@ip-172-31...	perf-demo-fin...	Asy
get_20_stocks	6837.42	io (98.0%)	7	DemoApp@ip-172-31...	perf-demo-fin...	Asy
get_20_stocks	6238.74	io (98.0%)	7	DemoApp@ip-172-31...	perf-demo-fin...	Asy
get_20_stocks	6715.28	io (94.0%)	7	DemoApp@ip-172-31...	perf-demo-fin...	Asy
get_20_stocks	9000.72	io (95.0%)	7	DemoApp@ip-172-31...	perf-demo-fin...	Asy
get_20_stocks	6114.60	io (96.0%)	7	DemoApp@ip-172-31...	perf-demo-fin...	Asy
get_20_stocks	7320.33	io (100.0%)	7	DemoApp@ip-172-31...	perf-demo-fin...	Asy
get_20_stocks	7804.00	io (88.0%)	7	DemoApp@ip-172-31...	perf-demo-fin...	Asy

PurePaths | Contributors | Errors

PurePath Tree (showing only relevant nodes)

Method	Argument	Exec Total [ms]
doFilter(ServletRequest request, ServletResponse response, FilterChain chain)	/finance/getFinan...	3967.26
service(HttpServletRequest request, HttpServletResponse response)		3967.16
runGetFinancialRecordsByDate(FinancialRecordRequest request, ServiceVersion version)	VERSION_1	3727.00
Database Summary		3248.28
21 × Connection Acquisition @ jrs-perf-demo-db.ce0zdvouniy.us-east-1.rds.amazonaws.com	application	1329.46
20 × select financiali0_.id as id1_0_0_, financiali0_.EXCHANGE as EXCHANGE2_0_0_, financiali0_.Sy	application	959.13
20 × select financialr0_.id as id1_1_, financialr0_.CLOSE_PRICE as CLOSE_PR2_1_, financialr0_.DATE	application	957.00
1 × select financiali0_.id as id1_0_, financiali0_.EXCHANGE as EXCHANGE2_0_, financiali0_.SYMBO	application	2.70

Demo #3 – Poor DB Performance (cont)

The Analysis

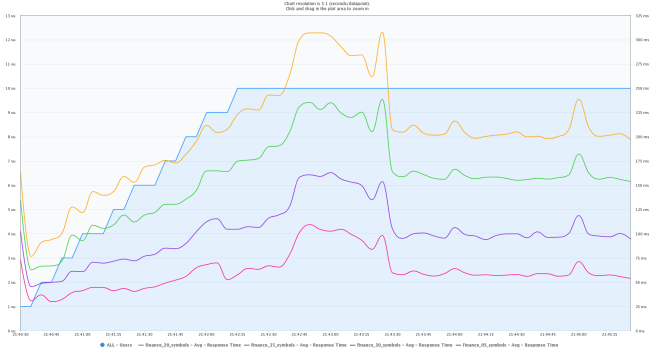
Service suffered from an N+1 query problem where to fetch the data for N stocks, N+1 database queries were made.

The Solution

Refactored to use a SQL join clause to reduce the number of database queries to 1 per service call

Demo #3 – Poor DB Performance (V2 Test)

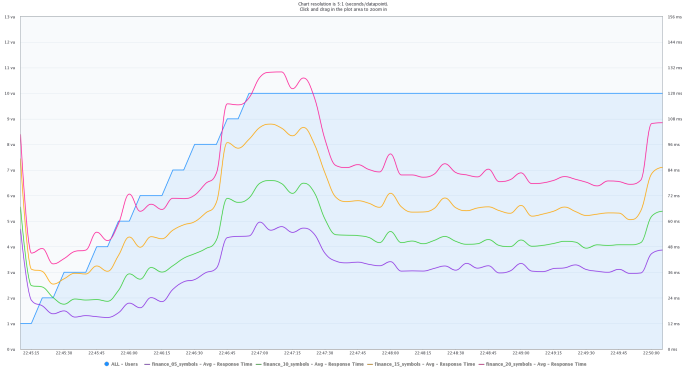
Before – Least performant call maxed at 300ms, then ran at approx. 200ms



PurePath Tree (showing only relevant nodes)

Method
doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
service(HttpServletRequest request, HttpServletResponse response)
runGetFinancialRecordsByDate(FinancialRecordRequest request, ServiceVersion version)
Database Summary
21 × Connection Acquisition @ jrs-perf-demo-db.ce0zdvouniy.us-east-1.rds.amazonaws.com
20 × select financiali0_.id as id1_0_0_, financiali0_.EXCHANGE as EXCHANGE2_0_0_, financiali0_.SY
20 × select financialr0_.id as id1_1_, financialr0_.CLOSE_PRICE as CLOSE_PR2_1_, financialr0_.DATE
1 × select financiali0_.id as id1_0_, financiali0_.EXCHANGE as EXCHANGE2_0_, financiali0_.SYMBO

After – Least performant call maxed at 130ms, ran at < 100ms. Database shows fewer SQL calls per web service call.



PurePath Tree (showing only relevant nodes)

Method	Argument	Exec Total [ms]
doFilter(ServletRequest request, ServletResponse response, FilterChain chain)	/finance/getFinan...	3113.14
service(HttpServletRequest request, HttpServletResponse response)		3113.04
getFinancialRecordsByDate(FinancialRecordRequest)		*2875.71*
runGetFinancialRecordsByDate(FinancialRecordRequest request, ServiceVersion ve	VERSION_2	2875.71
Database Summary		2872.68
1 × Connection Acquisition @ jrs-perf-demo-db.ce0zdvouniy.us-east-1.rc	application	1797.81
20 × select financiali0_.id as id1_0_0_, financiali0_.EXCHANGE as EXCHANG	application	1072.89
1 × select financialr0_.id as id1_1_, financialr0_.CLOSE_PRICE as CLOSE_PR	application	1.97


SAW BUG IN DEMO
NO ONE ELSE NOTICED

Try it yourself !

1. Setup AWS account for use of EBS, EC2, CodePipeline, CodeBuild: https://aws.amazon.com
2. Setup Blazemeter Trial Account: https://www.blazemeter.com/
3. Setup a Dynatrace AppMon Personal Licence: http://bit.ly/dtpersonal
4. Fork Demo App Code as to perform Builds and Run it – *Readme files have a lot of details* - https://github.com/jsicree

Contact reminders …

Joe Sicree

Sr. Consultant @ CGI

@JoeSicree -- Joseph.Sicree @cgi.com

Rob Jahn

Technical Director @ Elyxor

Rob.Jahn@elyxor.com

This presentation and the demo source code can be found at: https://github.com/jsicree/perf-demo