Advanced Software Engineering

03.12.2015

From prototype to product
(make it work 24/7)

DI Stefan Strobl

Introduction – Operating your Software

- Classic waterfall from an operations perspective
  - Plan, specify, design, build, test, deploy
  - No/little incentive to think about operations before testing, and even there, only limited

- In an agile environment
  - Potentially shippable code every day
  - How do we find out?
  - Ship to a near production environment as often as possible

A short story on how not to

- Dev team has been developing and shipping test builds to test env for months
- Environments are self managed by dev/test team
- All functional tests are passing
- Even some load tests are performed
- In short: everything looks well and everybody is feeling positive about the upcoming release
- Some savy developers have even written a deployment handbook detailing all (?) the steps necessary to set up a new environment
- Two weeks before production, first release candidate is handed over to operations for deployment on pre-production
- but...

... from here it is going downhill, and fast!

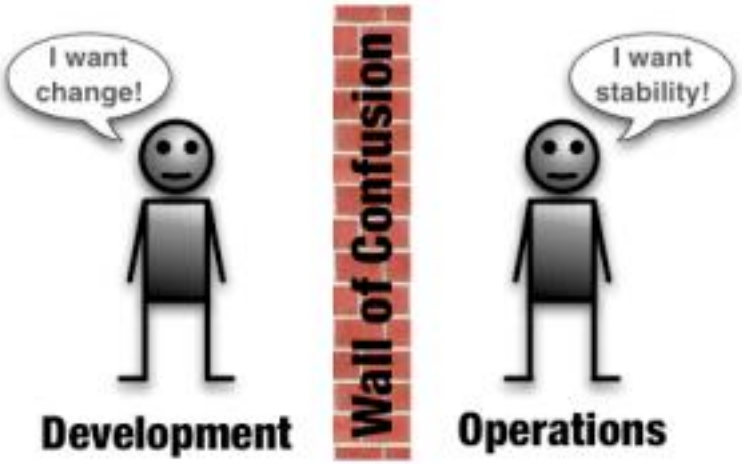
- In the ensuing ping pong game between development and operations the following issues have to be addressed
  - missing database configuration, application does not startup
  - -> **configuration management**
  - first time running in clustered environment, every second request fails with "session not found"
  - -> **clustering**
  - database much bigger than test datasets, performance is seriously affected on certain queries
  - -> **performance**
  - after running for two days with only minor usage from the user tests, the servers run out of memory
  - -> well... also **performance**, or **load testing**, or **monitoring** (last of which does not actually solve the problem)
- In the end: a functional software left a bad first impression disgruntling both operators and users

Short Recap

- Continuous Integration, Continuous Delivery (previous lecture)
- Create an automated pipeline to build, test, assemble and run your code
- For today's lecture considered as precondition (although all of the given points are equally relevant in non-agile/CI environments)
- By having your build pipeline in place, your deploy pipeline gets a lot faster

DevOps - Buzzword or Future?

- Difficult to draw clear line between development and operations
- Still in many cases two different departments/silos
- DevOps is more about breaking down walls then classic "who does what"
- It is not about who does it, but when and how it is done
- It is about considering operations from the beginning
- It is about knowing how "the other side" works



Configuration Management (CM)

- Definition:

Configuration Management ... is a management process for establishing and maintaining consistency of a product's performance, its functional and physical attributes, with its requirements, design and operational information, throughout its life. (ANSI/EIA-649-B)

- Which configuration to manage?
  - Build configuration
  - Product configuration
  - Application server/database configuration
  - OS configuration
  - System configuration

CM – Build Configuration

- The state of your source code is configuration
- The (internal and external) dependencies of your build are configuration
- The (implementation) state of your requirements is configuration
- The state of your defects (at build time) is configuration
- The documentation of executed tests (test plans) is configuration

CM – Application Level

- Often done in database and / or alongside source code
- Application should recognize and warn about or even fail on "wrong" configuration
- Keep configuration in as few places as possible
- Make it easy to view/change configuration
- Make clear distinction between user data and configuration in database - ideally, store in different "namespaces"
- Make clear distinction between environment specific and product specific configuration

CM – System Level

- Where to draw the line between application and system
  - e.g. Java options/memory configuration, especially settings like user.timezone or user.language
  - Remember previous lecture, what happens if the locale is set to a different language (e.g. Turkish)
- Tools like Puppet help a lot
  - Avoid manual "tinkering" for correctly setting up an environment
  - By automating you are forced to create a consolidated view of the characteristics of your system
- Server virtualization and "ready to go template images" make replicating environments a lot easier
  - Have room for extra environments
  - Have room for experiments

Clustering vs. Load Balancing

- Different clustering modes, different implications
  - Full session replication
  - Partial (delta) session replication
  - Database level clustering (Oracle RAC)
- Load balancing
  - Sticky session
  - Round robin (also DNS round robin)
  - Active/passive
  - Hardware vs. software
- Tradeoff between load distribution and fault tolerance
- Always perform fail-over tests on your setup

Clustering & Caching

- A clustered setup has strong implications on your caching strategies
- In-Process Caching
  - One cache per-process (higher overall memory usage)
  - Possibilities for inconsistencies between individual caches
  - Be extra careful with cache size in on-heap scenarios
- Distributed Caching
  - Slower due to additional overhead in form of network latency and object serialization
  - More complex to operate
  - Scales much better
  - No risk of taking down the main application with OutOfMem

Clustering & Session Serialization

- Activated session replication (full or delta) means each change to your session is replicated to all other nodes!
- Everything in your session has to be serializable
  - Implement java.lang.Serializable
  - Correctly handle transient fields
- Might generate a lot of network traffic
- Be careful with UI frameworks that do/support server-side state saving (JSF, Vaadin, …)
- Keep session size as small as possible
- Keep session as stable as possible

Focus: Master Node Election

- Used/needed for
  - Ensuring something is executed only once (e.g. scheduled job)
  - Ensuring messages are handled in correct order
  - Have one node to mediate or delegate
- Automatic master node election is difficult to get right (unless you have a single resource to sync on)
- ... and has some ugly constraints (split brain for example)
- Manual master node election
  - might result in down time
  - possibility for human error

Performance (testing)

- Test vs. development
  - Frequently internal (white box) know how/specific configuration required
  - QS-departments often do not have the necessary skills
  - Best done in collaboration
- Testing is only the "last" step to verify
  - Considering performance implications during design & development
  - Do your homework – know your numbers
- Target potential bottlenecks first
  - Limited thread/connection pools
  - Frequently used pages (e.g. welcome page/dash board)

Performance - from a database perspective

- Use a clone (ideally anonymized) of the production database
- Think about the resulting database queries (especially when using ORM tools)
- Be especially careful when operating on lists / result sets
  - What will you do with them?
  - Lazy loading of child entities
  - n+1 queries problem
- Think about indexes that fit your query patterns
  - use explain plans
  - make sure statistics are up to date
- Optimize based on data/facts and not on assumptions

Performance - from a system integration perspective

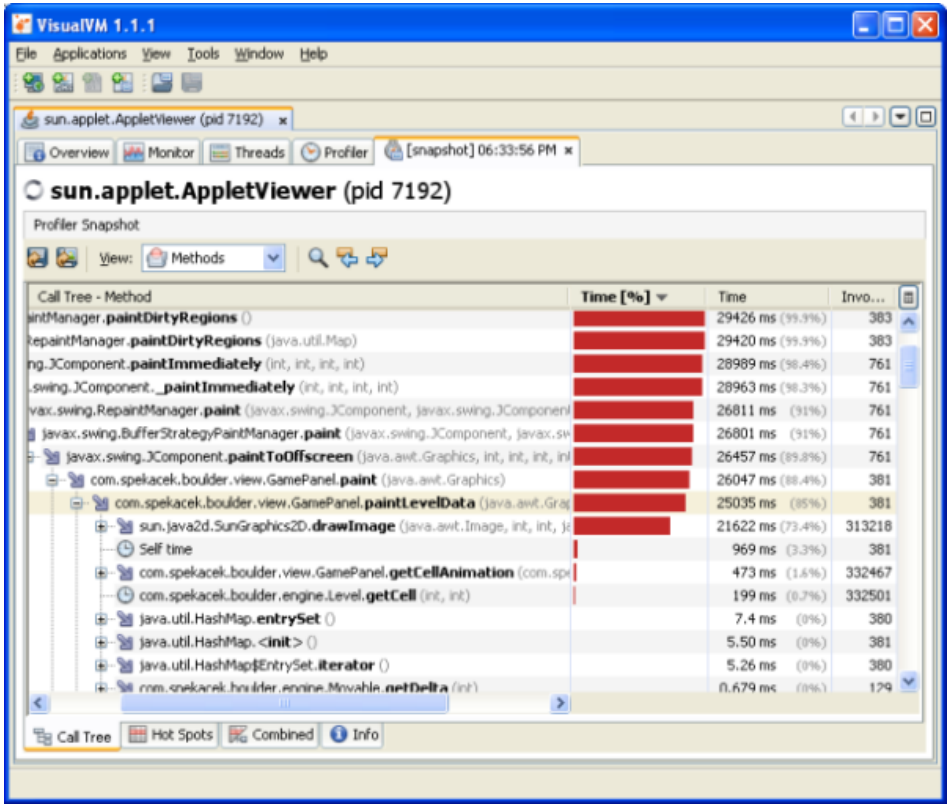
- Be aware of all calls that are "leaving" your system
  - Are there SLAs?
  - Make sure you can make clear statements about actual performance
  - Minimize the amount of round trips required
- Make sure you know about timeouts and how the system reacts
  - Timeouts tend to bubble up. Increasing the timeout on a lower level might result in timeouts on a different (higher) level
  - Example: web service timeout vs. transaction timeout vs. session timeout vs. browser request timeout
  - Some timeouts are not easy to influence (e.g. browser timeout)
- Consider automatic retries if you can correctly detect specific errors
  - However be aware of worst case scenario
  - e.g. timeout of 5 minutes * 3 retries means your user request might run and block resources for 15 minutes

Performance - profiling

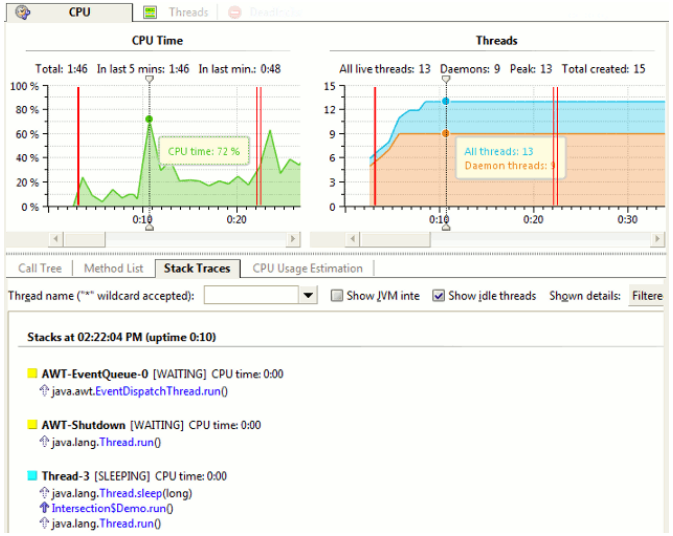
- Tracing
  - Usually done through byte code instrumentation
  - Delivers invocation counts
  - Can significantly influence runtime performance
  - Not suitable for production environments
- Sampling
  - Periodically queries stacks of running threads to estimate the slowest parts of the code.
  - No invocation counts
  - Negligible performance impact

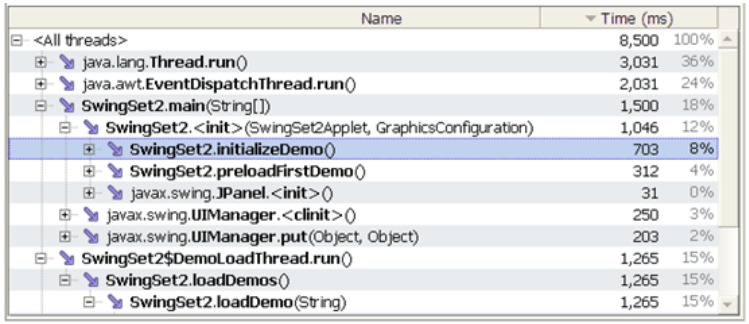
Performance – Profiling Tools

- JVisualVM (included in JVM)



- YourKit (commercial)





Performance - measuring

- “Manual” measuring
  - Good to see call durations at specific points
  - Good for runtime behavior (e.g. hardly affects performance)
  - Good for adaptive measuring/reporting
  - Bad if really done "manually" -> too much boiler plate code
  - Bad for measuring "everything" (e.g. finding the needle in the hay stack)

- Pitfall
  - Always use System.currentNanos() for measurement
  - System.currentTimeMillis() resolution based on timer interrupt (e.g. 10 ms)

Performance - measuring with AOP/Interceptor

- @Measured Annotation, Performance Interceptor
- StopWatch API (commons-lang, Perf4J, Spring)

```
protected Object invokeUnderTrace(MethodInvocation invocation,
Log logger) throws Throwable {
	String name = createInvocationTraceName(invocation);
	StopWatch stopWatch = new StopWatch(name);
	stopWatch.start(name);
	try {
		return invocation.proceed();
	} finally {
		stopWatch.stop();
		logger.trace(stopWatch.shortSummary());
	}
}
```

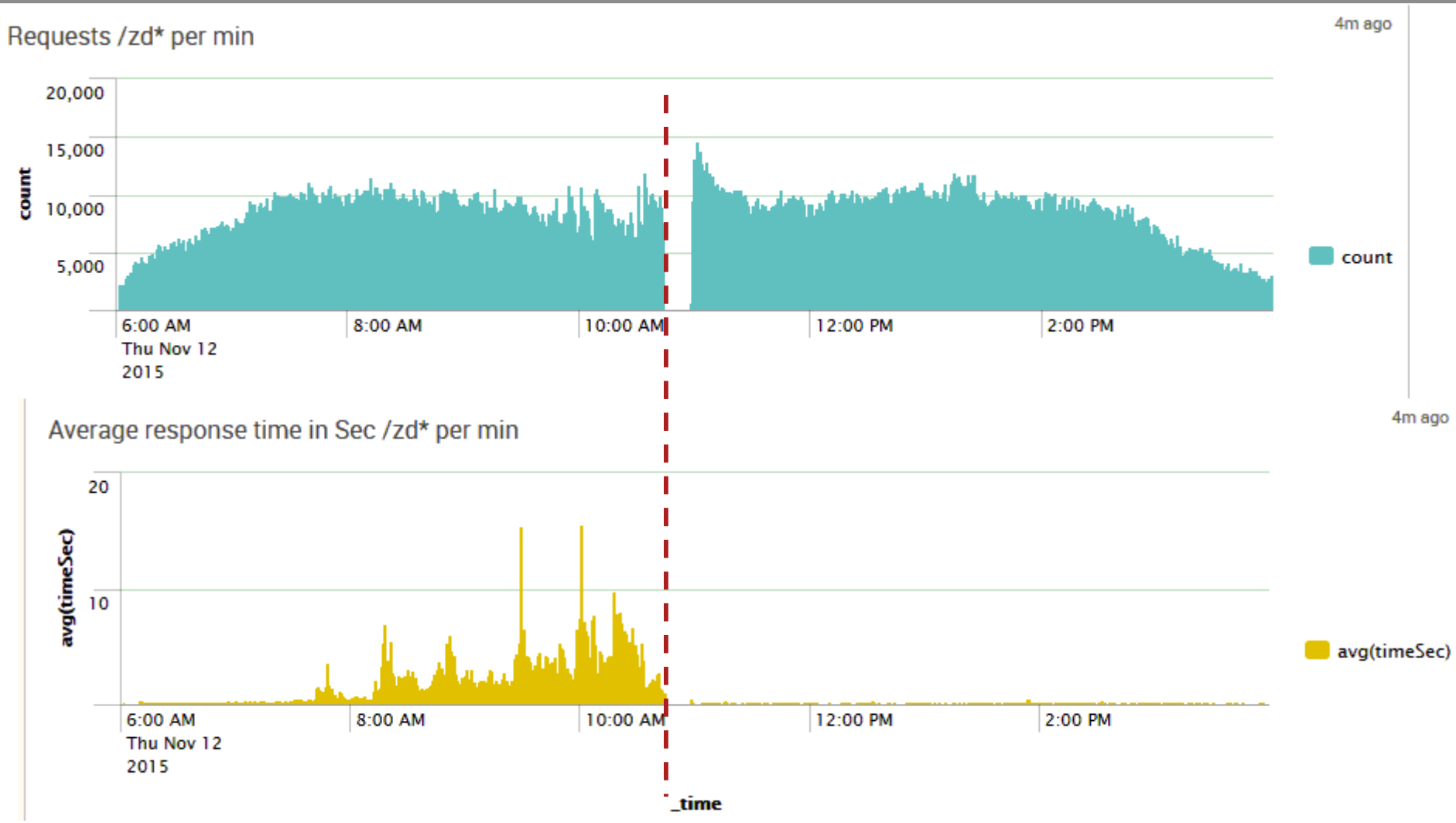
Monitoring

- Frequently seen as a pure operations task
- Difficult to detect and (even more) pinpoint application level problems
- Basic monitoring is easy
  - System states (e.g. database server down)
  - System resources (e.g. available memory)
  - Java behavior (e.g. GC intervals, heap state, ...)
  - Infrastructure state (e.g. queue sizes, thread pool size)
- All of the above only indicate "disaster" cases - and not if anything goes wrong/weary in my application
- No way for operations to define application level points for monitoring

Application level monitoring

- Goal - bring domain specific knowledge into operations
- First step: vertical "health check“ / Heartbeat
  - Is the UI reachable
  - Does the UI reach the backend
  - Does the backend reach and write to the database
  - Can the backend reach other required systems
- Later: application specific monitoring e.g.
  - `ThreadPoolTaskExecutorJmx`
  - `ReminderInboxJmx`
  - `PersistenceCacheJmx`
  - `MovementTelexProcessingJmx`
- Highly specific to the monitored application
- A lot of application specific monitoring tasks can also be handled by database queries
  - E.g. amount of open tasks

Application level by log analysis



Application Monitoring – Example with JMX

```
@ManagedResource(
   objectName = "at.vie.m2i.bean:name=reminderInboxJmx",
   description = "Monitor the state of reminder inboxes")
public class ReminderInboxJmx {
   @Inject
   private MessageDao messageDao;


   @ManagedOperation(description = "retrieve the number of unread
        reminders for the given inbox")
   @ManagedOperationParameter(name = "inboxName", description =
        "The unique inbox Name to identify the inbox")
   public Long getUnreadReminderCountForInbox(String inboxName) {
        return messageDao.getUnreadMessageCount(inboxName);
   }
```

Summary

- Configuration Management – Think about what your application (and operations) needs to correctly setup your software including its environment
- Clustering & Load Balancing – Think about what has to be done to make your software run reliable and fast
- Performance – Always think about the performance implications of your day to day design decisions
- Monitoring – Know how your application is doing

Deployment is just a part of dev/ops cooperation, not the whole thing

- John Allspaw

Resources

- Bob Aiello and Leslie Sachs. Configuration Management Best Practices. Pearson Education, 2011.
- ISO. Quality management systems – Guidelines for configuration management (ISO 10007:2003). Tech. rep. ISO, 2003. URL: http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=36644.
- Larry Klosterboer. Implementing ITIL Configuration Management. IBM Press, 2008.
- Puppet: http://puppetlabs.com/
- http://dev2ops.org/2010/02/what-is-devops/
- http://www.rajiv.com/blog/2009/03/17/technology-department/
- http://www.yourkit.com/
- https://visualvm.java.net/
- http://www.kitchensoap.com/2009/12/12/devops-cooperation-doesnt-just-happen-with-deployment/
- https://blogs.oracle.com/dholmes/entry/inside_the_hotspot_vm_clocks