APACHE SLING & FRIENDS TECH MEETUP
BERLIN, 25-27 SEPTEMBER 2017

Developers' Dues For Successful DevOps

Julian Sedding

adaptTo()

About me – Julian Sedding

- Freelance consultant
- CQ 5/AEM since 2008
- ASF Committer – jsedding@apache.org
  - Apache Jackrabbit
  - Apache Sling
  - Apache HttpComponents

adaptTo()

What is DevOps?

What is DevOps?

“
[…] there are three primary practice areas that are usually discussed in the context of DevOps.

- Infrastructure Automation – […]
- Continuous Delivery – […]
- Site Reliability Engineering – […]”

— from https://theagileadmin.com/what-is-devops/

Infrastructure Automation

“create your systems, OS configs, and app deployments as code.”

Continuous Delivery

“build, test, deploy your apps in a fast and automated manner.”

Site Reliability Engineering

“operate your systems; monitoring and orchestration, sure, but also designing for operability in the first place.”

Operability

What is Operability?

> “Operability is the ability to keep […] a system […] in a safe and reliable functioning condition, according to pre-defined operational requirements.”

— from https://en.wikipedia.org/wiki/Operability

adaptTo()

Operational Requirements

- Performance
- Availability
- Predictable resource usage
- Scalability
- Easy to diagnose
- Configurability (at runtime)

adaptTo()

Threats to Operability

- Excessive or unpredictable resource usage
  - CPU
  - Memory / Heap
  - I/O – disk, network, sockets, …
- Dependencies to 3rd party systems
- Programming errors, e.g. deadlocks

adaptTo()

Operability Mindset

- How will your code impact the system?
  - Don’t (only) guess. Test, measure and monitor!
- Is the behavior of your code adaptable at runtime?
- Can it be disabled?
- What happens if the unexpected happens?
- Does it provide adequate information for analysis?

Simple Good Practices

adaptTo()

Logging - always use appropriate levels

- ERROR and WARN messages should be clear and actionable for operations.
- INFO messages are for generally interesting events during normal operation.
- TRACE or DEBUG may require the source code to fully understand what's happening.

adaptTo()

Logging - a side note on Slf4J

- Always use format patterns.
  E.g. `LOG.debug("x:{}, y:{}", x, y)`
- Never use String concatenation when logging. E.g. ~~`LOG.debug("x:" + x)`~~
- Only guard expensive calls with `LOG.isDebugEnabled()` and friends.
  E.g. `LOG.debug("x:{}", expensiveX()))`

adaptTo()

Exception Handling

- Never swallow exceptions. Ever. Always log (with stack trace) or re-throw.
- Deal with exceptions as soon as possible. It only gets harder further from the cause.
- Usually, there is no need to invent custom exception classes.

Monitoring

- Expose key characteristics via JMX
- What is interesting?
  - Rate of events over time, e.g. requests / sec
  - Size of data-structures, e.g. size of a job queue
  - Durations, e.g. duration of data import
  - Statistical variation of values over time, average, percentiles, etc.

Configuration

What needs to be configurable?

- Values unknown during development
- Values that vary across deployments
- Values that may change over time
- Turning on/off (new) features, e.g. via `ConfigurationPolicy.REQUIRE`

Example: 3rd party integration via HTTP

About The Example

- Real-world scenario
- Caused by lack of key “good practices”
- Illustrates a solution that follows some simple good practices and what this can lead to.

The Scenario

- Symptom: all publish systems down
- Cause: internal 3rd party system is down
  - Hang on, this shouldn't pull down publishers?!
- Cause (take 2): HTTP requests to 3rd party cannot complete and never time out
  - 3rd party HTTP requests are made during page rendering, blocking all page rendering; eventually server thread-pool may become exhausted

Diagnosis

- No meaningful log messages
- Thread dumps show multiple waiting threads pointing to a class that uses `HttpClient`
- The `HttpClient` has no timeout by default
- 3rd party system confirmed to be down

First Aid

- Timeouts for `HttpClient` instances are set programmatically -> no runtime config
- Disabling the relevant OSGi Component may lead to NullPointerExceptions

➔ No (easy) options left to stabilize the system!

adaptTo()

Quick Fixes

- Implemented timeout OSGi configuration for the failed OSGi Component
- What about other usages of `HttpClients`?

➔ Use hard-coded timeout via utility class

Why do we need to fix so many places?

adaptTo()

Architecture Considerations

- How to avoid repeating the same failure?
- Should a developer *using* `HttpClient` need to
  - … implement its configuration?
  - … know about the best configuration up-front?
  - … know about its life-cycle?
- Can we make usage easier?
- Can we make configuration more consistent?

adaptTo()

`HttpClient` Configuration Support

- Configure `HttpClient` via OSGi configuration
- Use pre-configured `HttpClient`, available as service (inject using `@Reference`)
- Choose between the default configuration or a named configuration

— https://github.com/code-distillery/httpclient-configuration-support

DEMO

Avoiding A Repeated Disaster

Lessons learned during extensive testing

- Always close `HttpResponses` – otherwise connection pool may become blocked
- Use `ResponseHandler` – auto-closes response
- Better: consume and close `InputStream` (from `HttpResponse#getContent()`) – allows connection re-use

BTW: this is all documented – just not very intuitive

Possible Next Steps

- Safety net for unclosed `HttpResponse` objects
- Monitoring (JMX MBeans)
  - Connection-pool statistics
  - Request rates, durations, response sizes
- Web Console Plugin
  - Overview of configurations and consuming services
  - Simpler UI for configurations?
- Configurable Caching?

Similar Solutions

- wcm.io Caravan
  - http://caravan.wcm.io/commons/httpclient/
- Netflix Hysterix
  - https://github.com/Netflix/Hystrix

Conclusion

Practice Simple Good Practices

It pays off to be mindful about

- Logging
- Exception handling
- Monitoring
- Configuration

adaptTo()

Beware of 3rd Party Integrations

- Extra complexity, extra risks – be extra careful
- Decouple integrated systems
- Prevent cascading failures
- Always use timeouts
- Consider blacklisting or a back-off strategy

adaptTo()

Testing: Orders of Magnitude Matter

Test your code under realistic conditions

- Volume and distribution of test data and load: use real data if available, otherwise randomize
- Concurrent execution: validate correctness, look out for contention
- Monitor how often your code is executed. As frequently as expected? Why not?

Red Flags

- 3rd party integrations
- Batch processes (interrupt, re-start, throttle)
- Similar (boiler-plate) code copied repeatedly
- “This is not part of my task” – An opportunity to factor out orthogonal concerns?

Thank you for your time!

Questions?