Continuous Performance Evaluation using Open Source Tools

Markus Dlugi, RETIT GmbH

Motivation

- Continuous Integration (CI) has reached widespread adoption
- Functional requirements are tested with automated tests
- Non-functional requirements such as performance are often neglected

How can we track performance if the application changes every day?

Every hour?

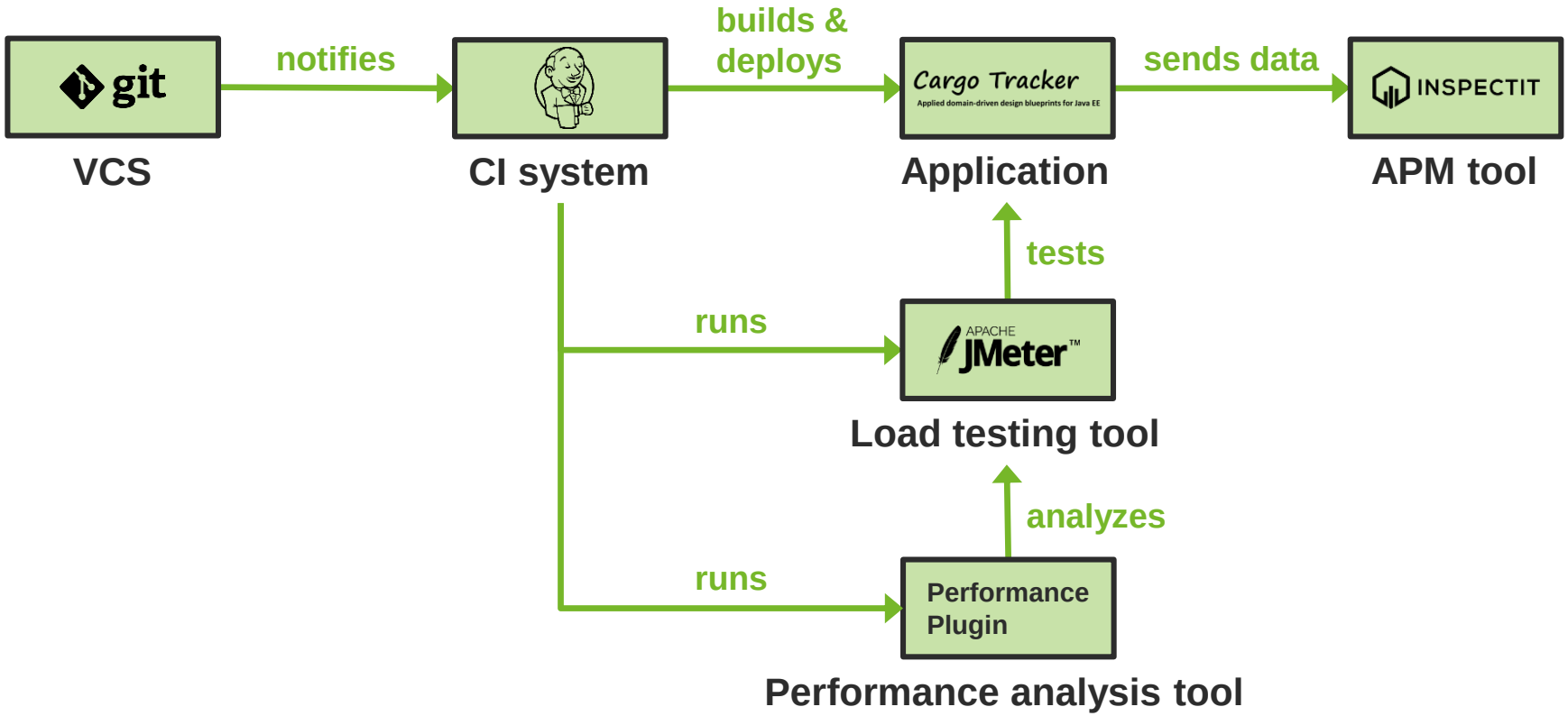
Every minute?

Motivation

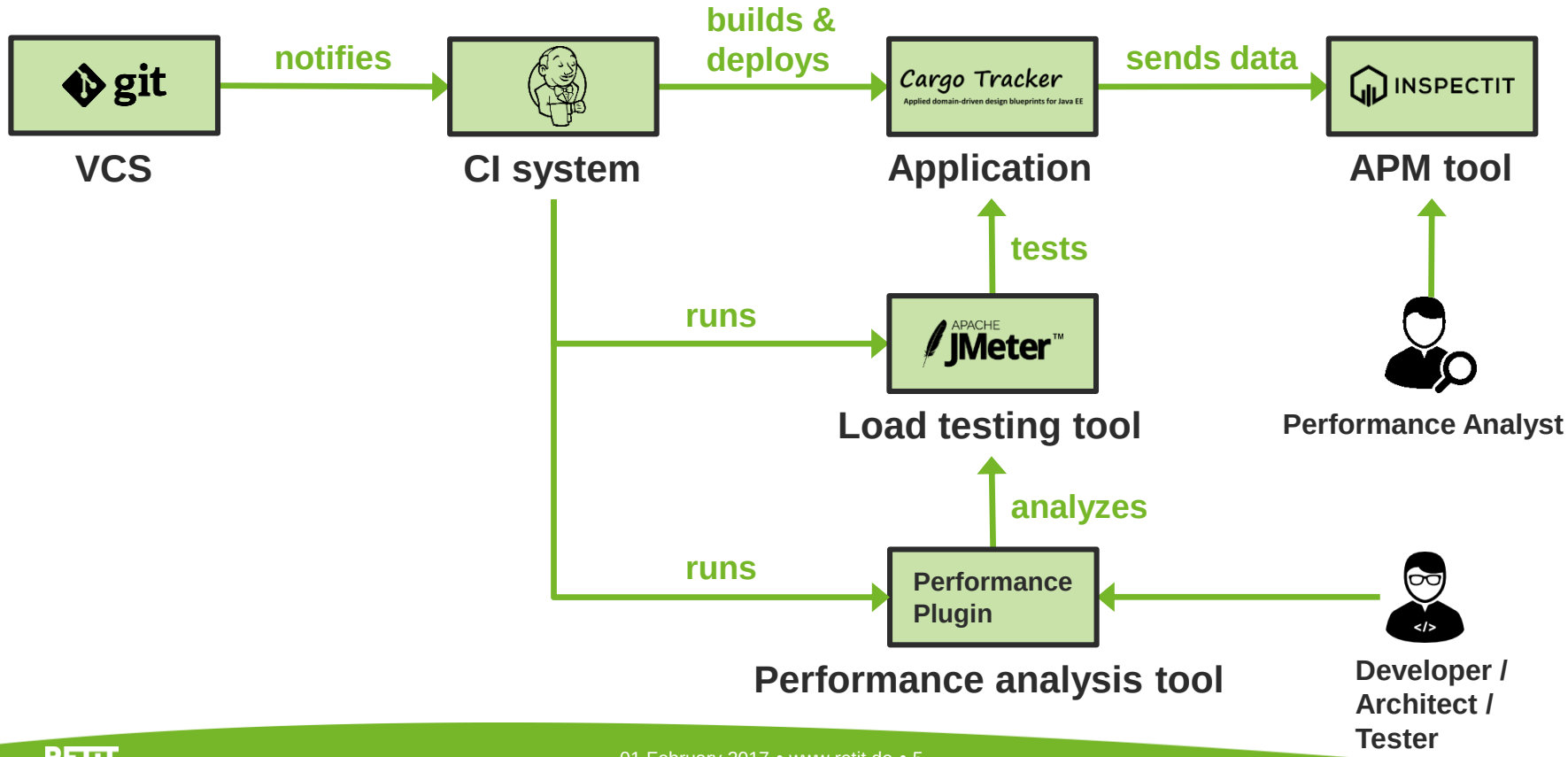
- Modern APM tools often have features to monitor the performance over multiple application versions
- However, they generally come with a high price tag

Maybe we can also do it with Open Source Software?

General workflow



General workflow



Alternatives: Version Control System

Alternatives: Continuous Integration Systems

Open Source alternatives:

CruiseControl

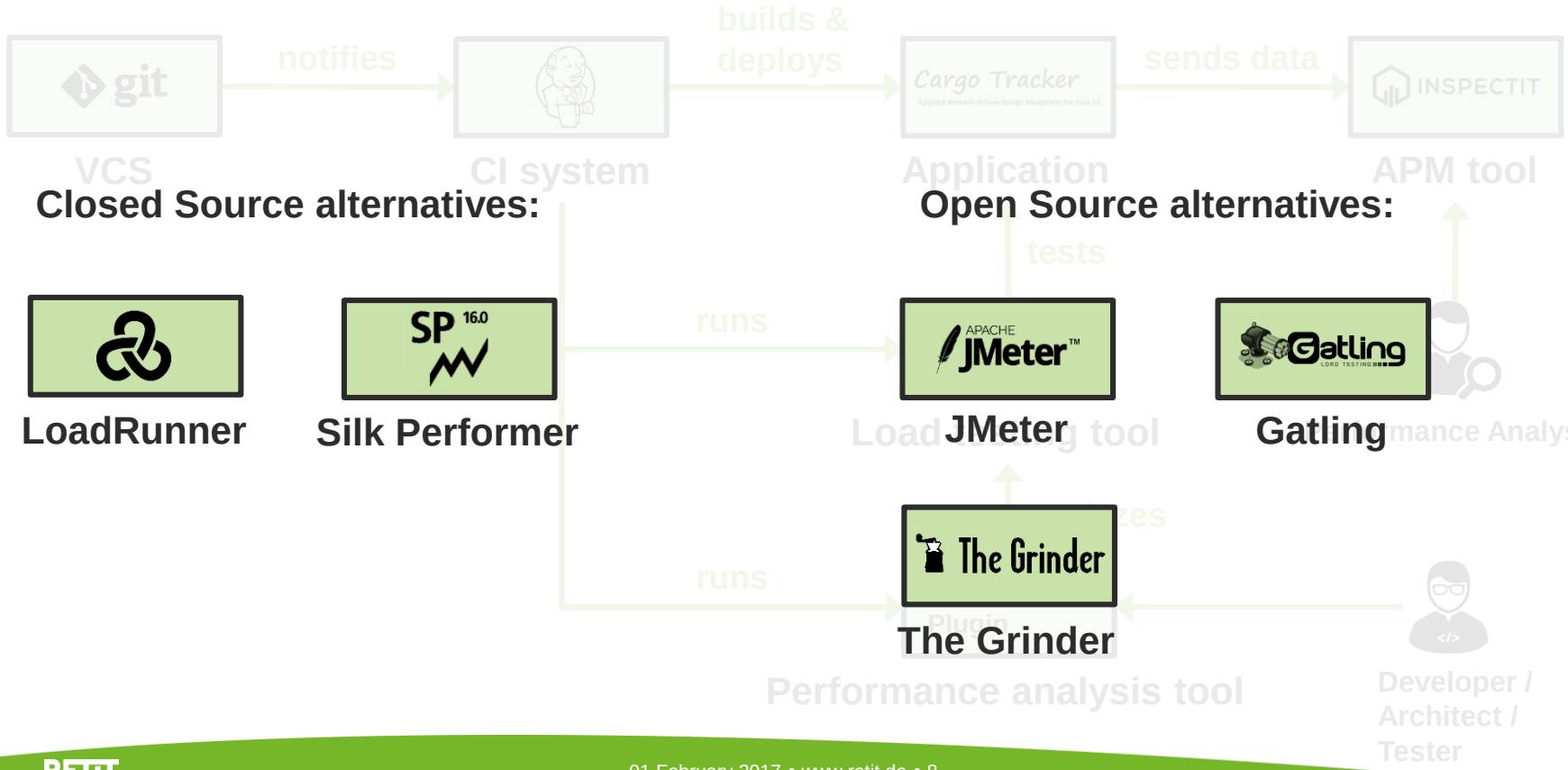
Jenkins

Closed Source alternatives:

Bamboo

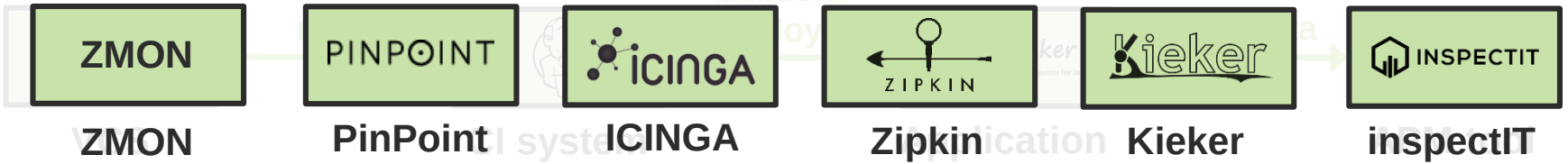
TeamCity

Alternatives: Load Testing Tools



Alternatives: APM Tools

Open Source alternatives:



ZMON PinPoint ICINGA Zipkin Kieker inspectIT

Closed Source alternatives:

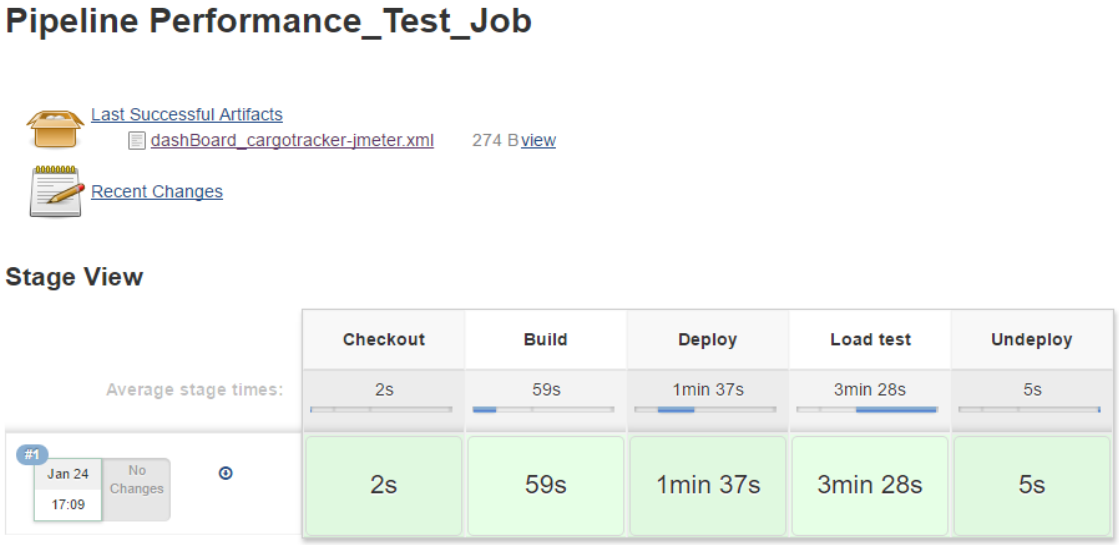
AppDynamics Dynatrace Instana New Relic and many more

Demo

https://github.com/RETIT/continuous-performance-evaluation

Tip #1: Use Pipelines

- Pipelines in Jenkins allow to specify all steps using Groovy scripts
  - → Avoid manual configuration
  - → Treat pipeline as code

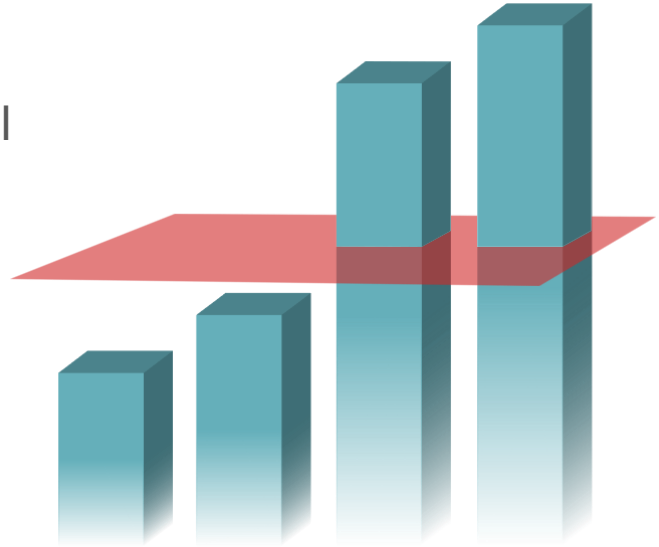


Tip #2: Use Load Tests correctly

- Load tests only yield meaningful results **if** used correctly
- When designing your tests, make sure that:
  - Your workload resembles how users would access the application
  - You achieve steady state performance (test runs long enough?)
  - Your test environment is comparable to your production environment
  - Your test environment has no external influences (e.g., from the load test environment)

Tip #3: Find good Thresholds

- Thresholds are a good way of notifying developers about performance problems
- However, good thresholds are hard to find
  - If the threshold is too low, developers will start ignoring the warnings
  - If the threshold is too high, potential problems might go unnoticed
- Each application and load test is different
  - Finding good thresholds takes time
  - Consider using absolute thresholds
  - You might also need to remove ramp-up using custom scripts

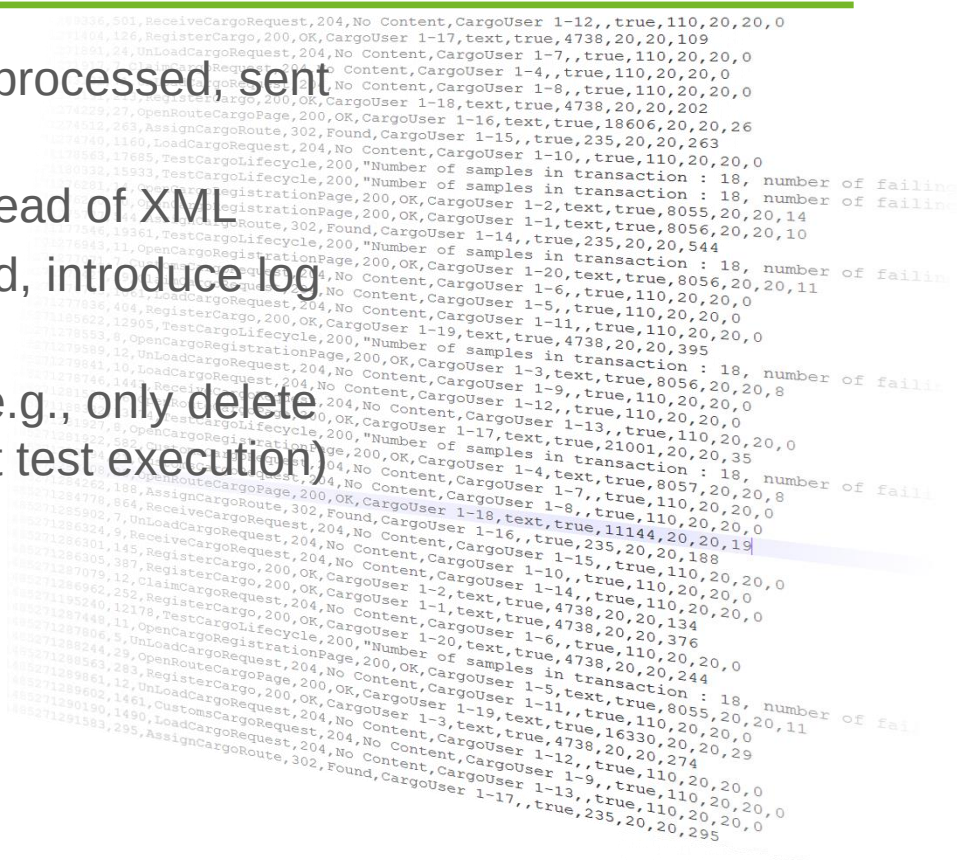


Tip #4: Run the Pipeline often

- Ideally, you would run your performance tests after every commit
- If you can’t do that, minimum is once a day
- Try to minimize the pipeline run time so you can run it more often

Tip #5: Optimize your Pipeline

- Minimize data which needs to be processed, sent or stored
  - Use JMeter's CSV output instead of XML
  - Don't store every Jenkins build, introduce log rotation
  - Use partial database resets (e.g., only delete data generated during the last test execution) instead of full restores
- Use caching effectively
  - Maven's local repository
  - Docker's image cache



Markus Dlugi
dlugi@retit.de

RETIT
Resource Efficient Technologies & IT Systems