Structured Performance Tests

•••

Adam Burke, 2020
@AdamBurkeware

Performance Tests Are Tests

- Performance tests are hard
- They are just tests though
- Systematic and repeatable
- Structure tests like tests
- Putting the pieces together with DevOps tooling

Background

- Developer, Tester, Team Lead, Architect, Researcher
- Electronic Trading, Low Latency, High Performance, Big Data, Process Mining
- Agile, Tooling, Continuous Delivery, Test & Deployment Frameworks
- Java, P[J]ython, Robot, C, R, DBs, etc
- Frangipani Labs, QUT BPM group, previously banks, exchanges, software houses and regulators

Business Process Management
#processscience #bpmatqut

QUT the university for the real world

Performance Tests Are Hard

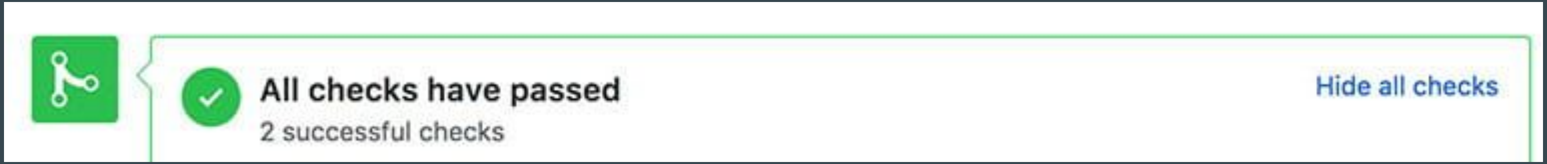
- Always testing against a model and a production scenario
  - Eg 3x volume, partial outage, slow network
- Control different elements
  - Machine
  - Network
  - Out-of-process, end-to-end tests
  - Input parameters in the large
  - Read outputs through instrumentation
  - Scenarios require more attention

- Microbenchmarks are also hard
  - .. and not the topic of this talk

Performance Tests Need Structure Too

If it hurts, do it more often - Martin Fowler (and others)

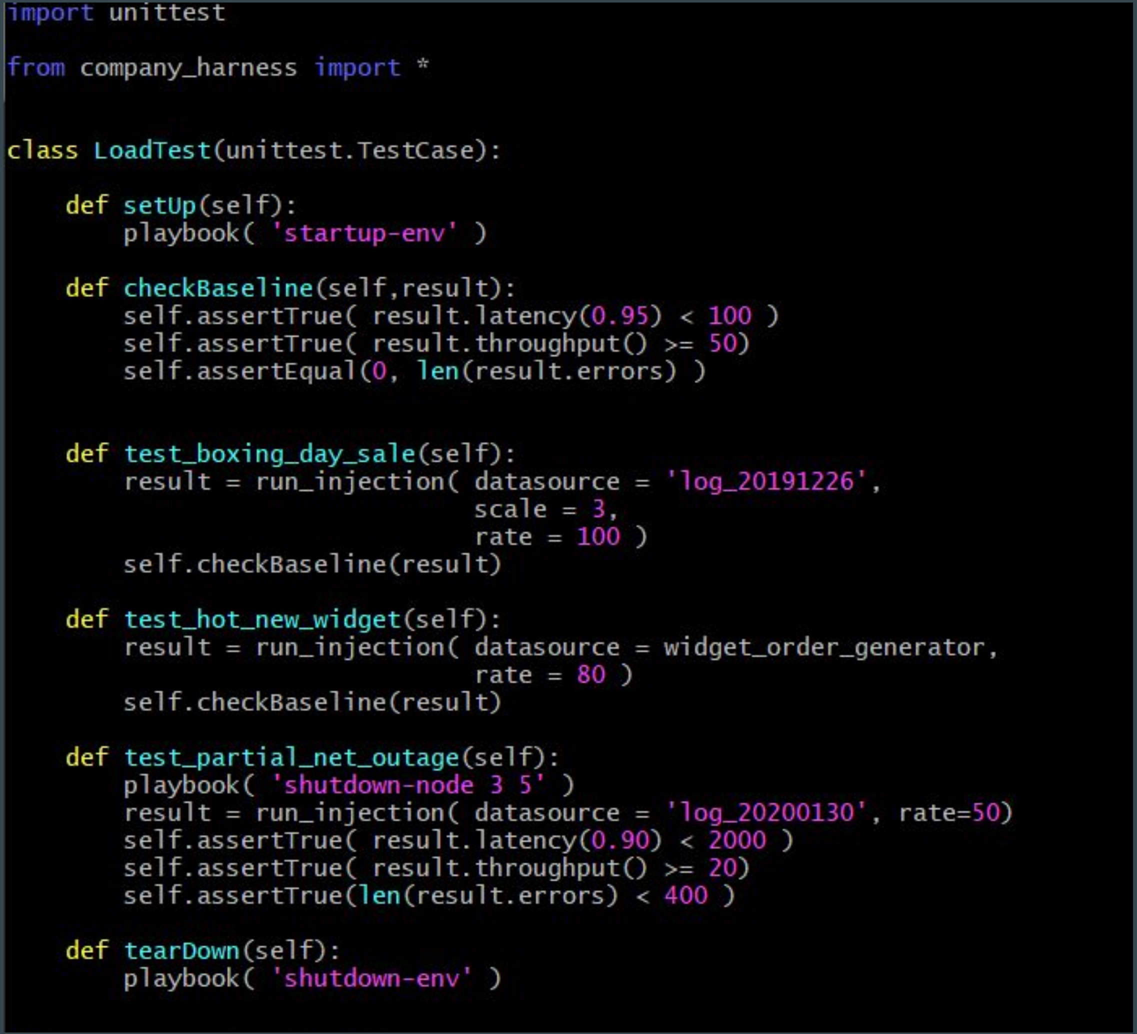
- Cheap, repeatable tests are more valuable than expensive one-offs
- Fast feedback on inadvertent performance degradation
- Plug into continuous integration pipeline


All checks have passed
2 successful checks

NUnit Concepts In Performance Tests

`@BeforeClass`	Redeploy environment
`@Before`	Configure run and restart process
`@Test`	One test scenario (2x volume, Christmas Eve pattern)
`assertTrue()`	Check benchmark met - post-run or asynch
`@After`	Shutdown processes
`@AfterClass`	Clear environment

Example



```
import unittest

from company_harness import *


class LoadTest(unittest.TestCase):

    def setUp(self):
        playbook( 'startup-env' )

    def checkBaseline(self,result):
        self.assertTrue( result.latency(0.95) < 100 )
        self.assertTrue( result.throughput() >= 50)
        self.assertEqual(0, len(result.errors) )


    def test_boxing_day_sale(self):
        result = run_injection( datasource = 'log_20191226',
                                scale = 3,
                                rate = 100 )
        self.checkBaseline(result)

    def test_hot_new_widget(self):
        result = run_injection( datasource = widget_order_generator,
                                rate = 80 )
        self.checkBaseline(result)

    def test_partial_net_outage(self):
        playbook( 'shutdown-node 3 5' )
        result = run_injection( datasource = 'log_20200130', rate=50)
        self.assertTrue( result.latency(0.90) < 2000 )
        self.assertTrue( result.throughput() >= 20)
        self.assertTrue(len(result.errors) < 400 )

    def tearDown(self):
        playbook( 'shutdown-env' )
```

When and How to Assert

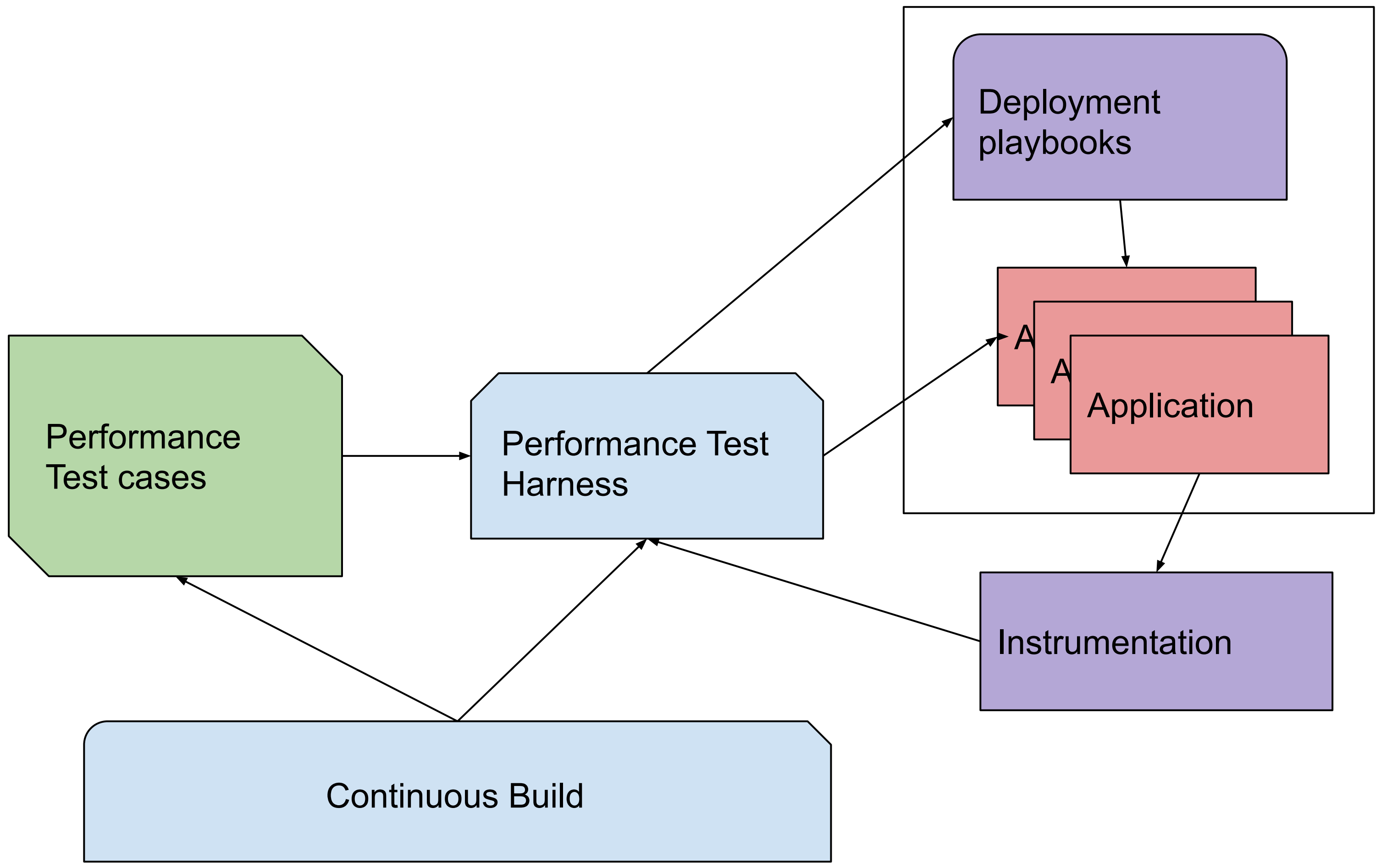
- The hardware is a parameter, so it can’t be shared
- Measure either post-run or asynch

- Post-run
  - Simpler,
  - Can’t fast fail

- Asynchronously and on separate hardware
  - Test and reuse production instrumentation
    - See e.g.: Charity Majors (@mipsytipsy) on ops and observability

Test Against A Model

- Follow all the disciplines to make test production-like, but
- ... there's only one production
- You are always testing against a model, choose it consciously



@AdamBurkeware

Data Choice Challenges

- Performance test parameters are inherently large
- Ensure the application is the bottleneck, not the test harness

- Replay
  - Realistic, but hard to associate scenarios
- Generated events
  - Requires work to vary data from prototype data
  - eg to spread load across instances

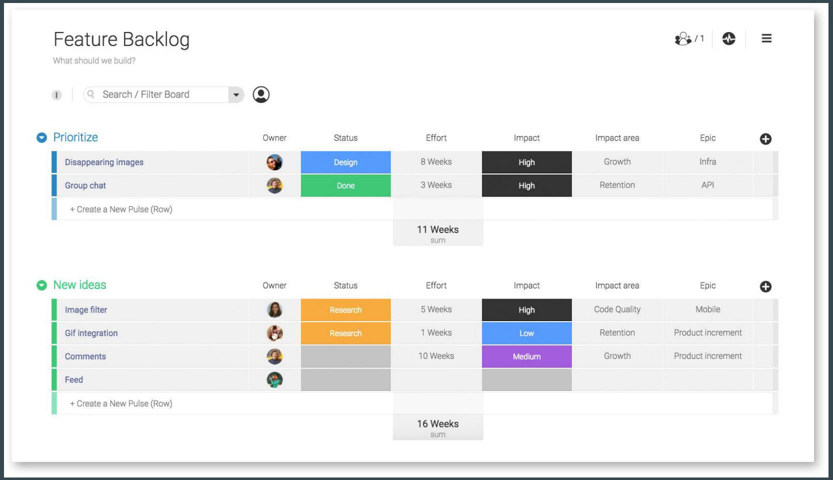
- One Giant Test Case problem

ONE GIANT REPLAY TEST LOG
ME TRYING TO UNDERSTAND WHAT FAILED

Agile Teams Are Bad At Recurring Infrequent Tasks

- Daily - probably ok
- Weekly - less likely
- Less often than weekly ...
  - Ends up on a backlog competing with features
  - Effort spirals as gets more out of date
- Run from continuous build - ie > daily

- All teams are pretty bad at infrequent tasks
  - requires bureaucracy or long-cycle ritual



Feature Backlog

What should we build?

Search / Filter Board

Prioritize	Owner	Status	Effort	Impact	Impact area	Epic
Disappearing images		Design	8 Weeks	High	Growth	Infra
Group chat		Done	3 Weeks	High	Retention	API
+ Create a New Pulse (Row)						
			11 Weeks sum			

New ideas	Owner	Status	Effort	Impact	Impact area	Epic
Image filter		Research	5 Weeks	High	Code Quality	Mobile
Gif integration		Research	1 Weeks	Low	Retention	Product increment
Comments			10 Weeks	Medium	Growth	Product increment
Feed						
+ Create a New Pulse (Row)						
			16 Weeks sum			

DevOps Tools Make Performance Tests Cheap

Separated hardware	Cloud
Build traceability and signoff	Continuous build
Clean production-like version and config	Deployment playbooks
Reconfiguration for different cases	Deployment playbooks
Data production and collection	Process monitoring APIs and instrumentation

Some Established Frameworks

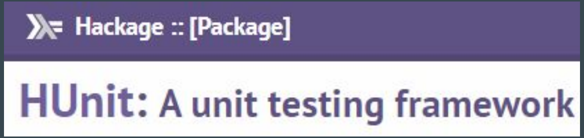
APACHE JMeter™

- Leverage if it makes sense - some good pre-cooked tooling
- Focus on mechanics of capture and replay
- Can be web-centric and less relevant to back-ends
- Need to design structure of tests still there

… But Don’t Forget These

python™

puppet


Hackage :: [Package]

HUnit: A unit testing framework

Se

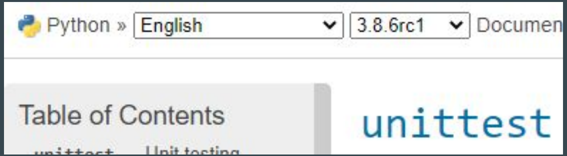

Python » English 3.8.6rc1 Documen

Table of Contents

unittest

unittest — Unit testing

ANSIBLE

docker

CHEF

JUnit λ

Performance Tests Are Tests

- Performance tests are hard
- They are just tests though
- Systematic and repeatable
- Structure tests like tests
- Putting the pieces together with DevOps tooling

Thanks for your time.

@AdamBurkeware