DevOps, microservices and stress-free incidents: How to have your cake and eat it too

Peter Holditch
Principal PreSales Engineer


pholditch@appdynamics.com

@pholditch

Have your agenda and eat it too

- How did we get here?
  - Why DevOps
  - Why Microservices
- Where DevOps and miroservices collide
- Joint DevOps & microservices management requirements
- Delivering these requirements: buy vs build
- Business case for buying a solution

Why DevOps?

- The rate of change of systems has grown exponentially
  - 2 releases a year, delivered complete with a big operations manual is not a recipe for high velocity delivery
- Keeping systems reliable in the face of this rate of change requires
  - Automation of the software release (manufacturing) process
  - Collaboration across the software lifecycle
    - “traditional ops” run platforms, delivery pipelines
    - Developers (formally) take responsibility for application troubleshooting in production

Why microservices?

- Loose coupling enables agility
  - Choose best language for the job by service
  - Different rates of change per service
  - Different NFRs per service
  - Easier to coordinate activity within smaller development teams
- Focused domain expertise
- Better accountability to business stakeholders

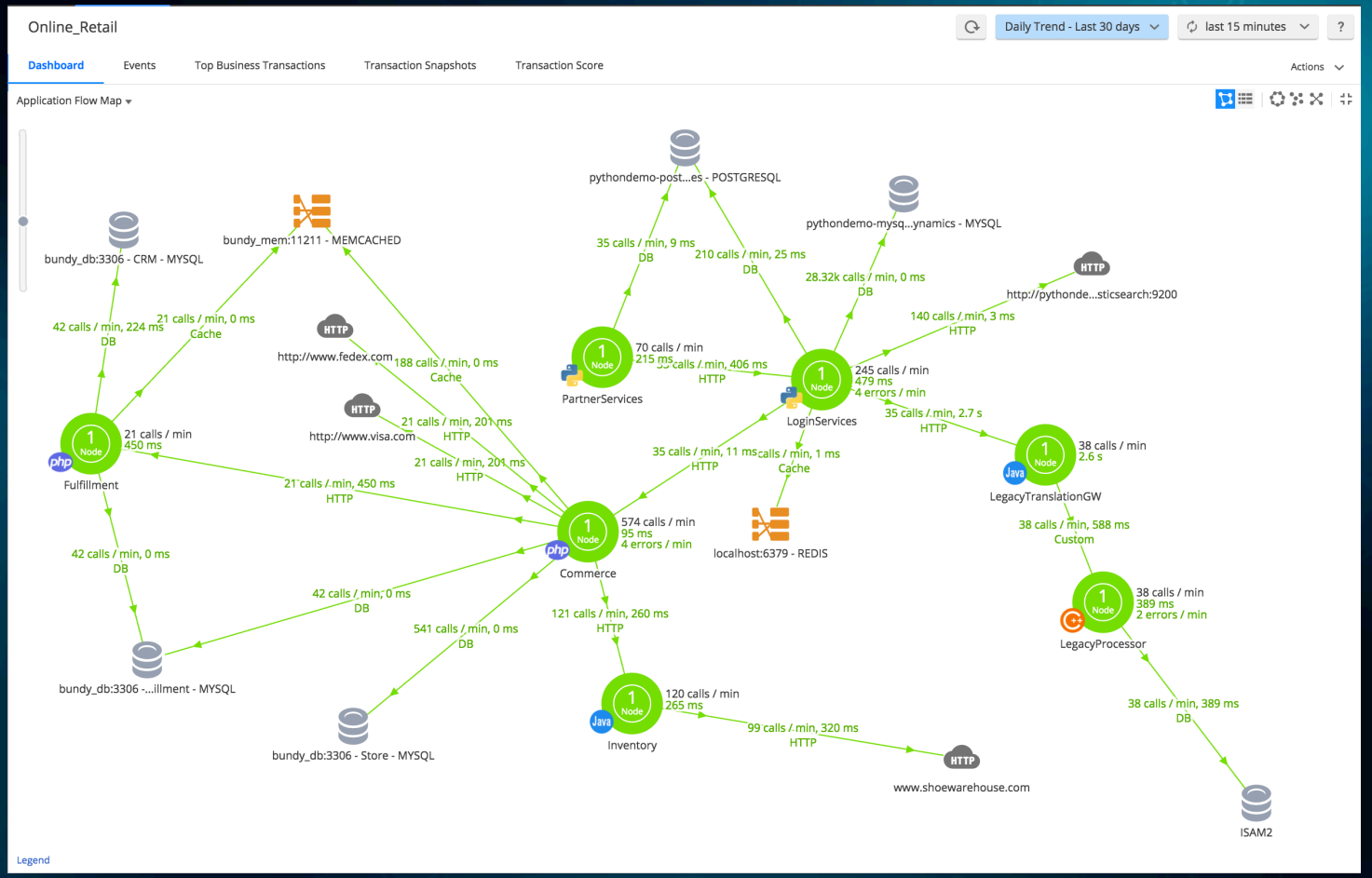
Where DevOps and microservices collide

- Trade off dev time simplicity vs operational complexity
- Troubleshooting : how many technology “throats to choke”?
  - Triage vs analysis
  - Is each piece managed in the same way?
  - Dev time from each team to support => the dreaded war room
- Maturity: once a service is stable and the team move on…
  - Who owns the service now?
  - Who can troubleshoot it?

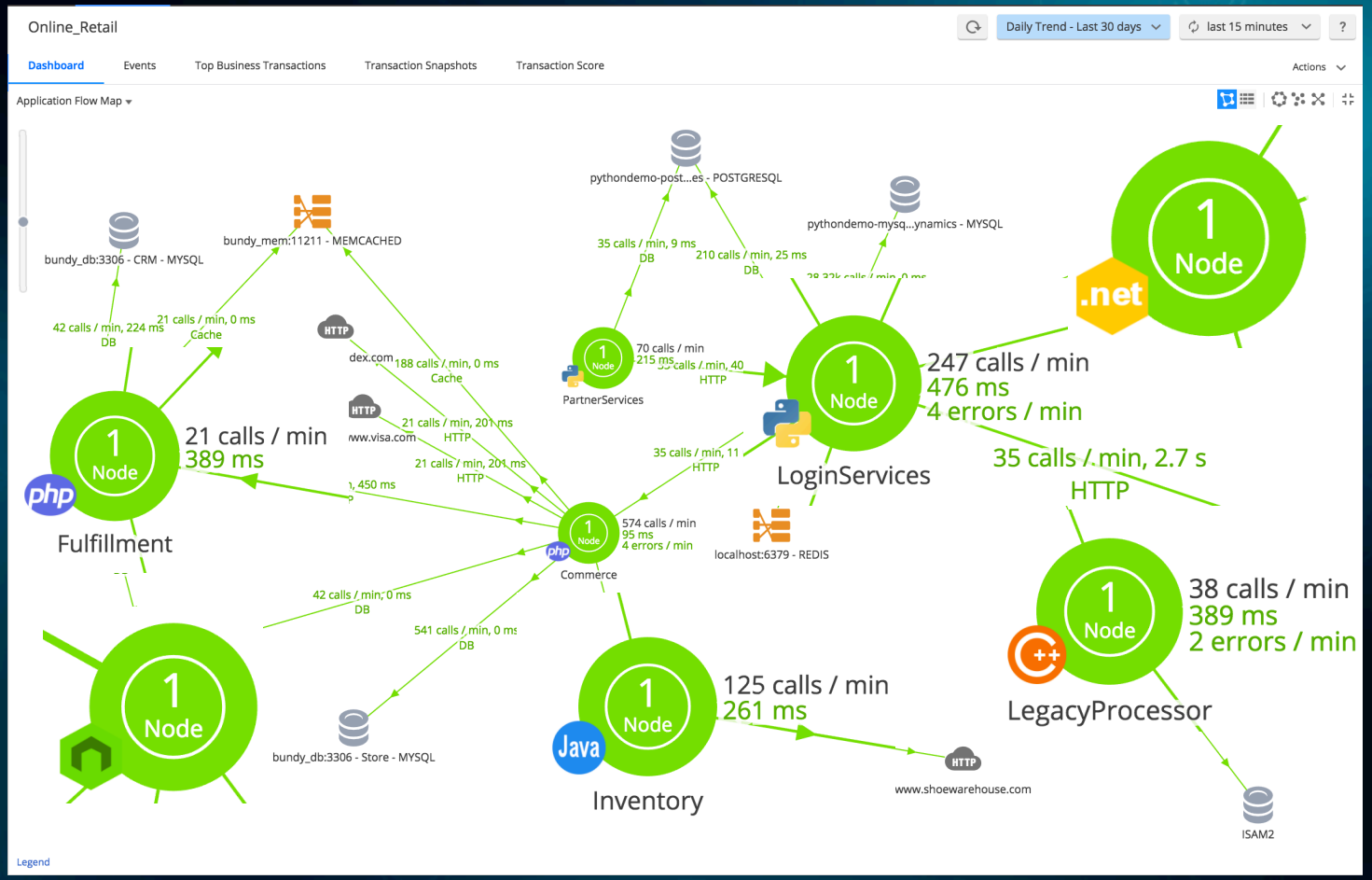
Requirements for unified monitoring of microservices

- For developers
  - Language support
  - Low touch
    - In implementation and production handover
    - Provide detailed code level visibility when necessary
- For Ops
  - Ease of use; simplify troubleshooting (“shift left”)
    - Consistency
    - Comprehensive end to end visibility, fault domain isolation
- For DevOps
  - Foster dev/ops collaboration
  - Provide feedback throughout delivery pipeline

Automatically trace application traffic end to end

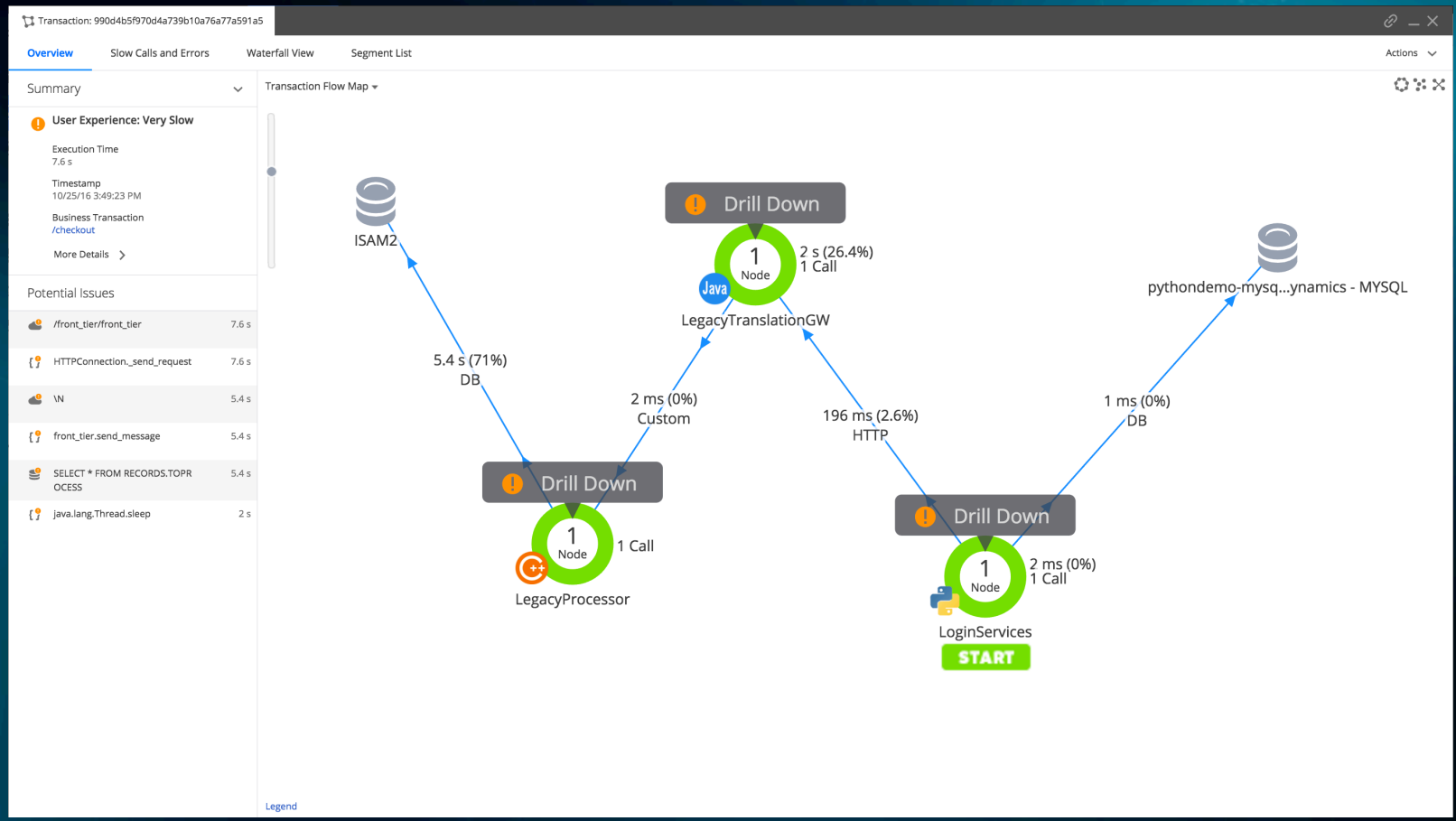


Support polyglot applications

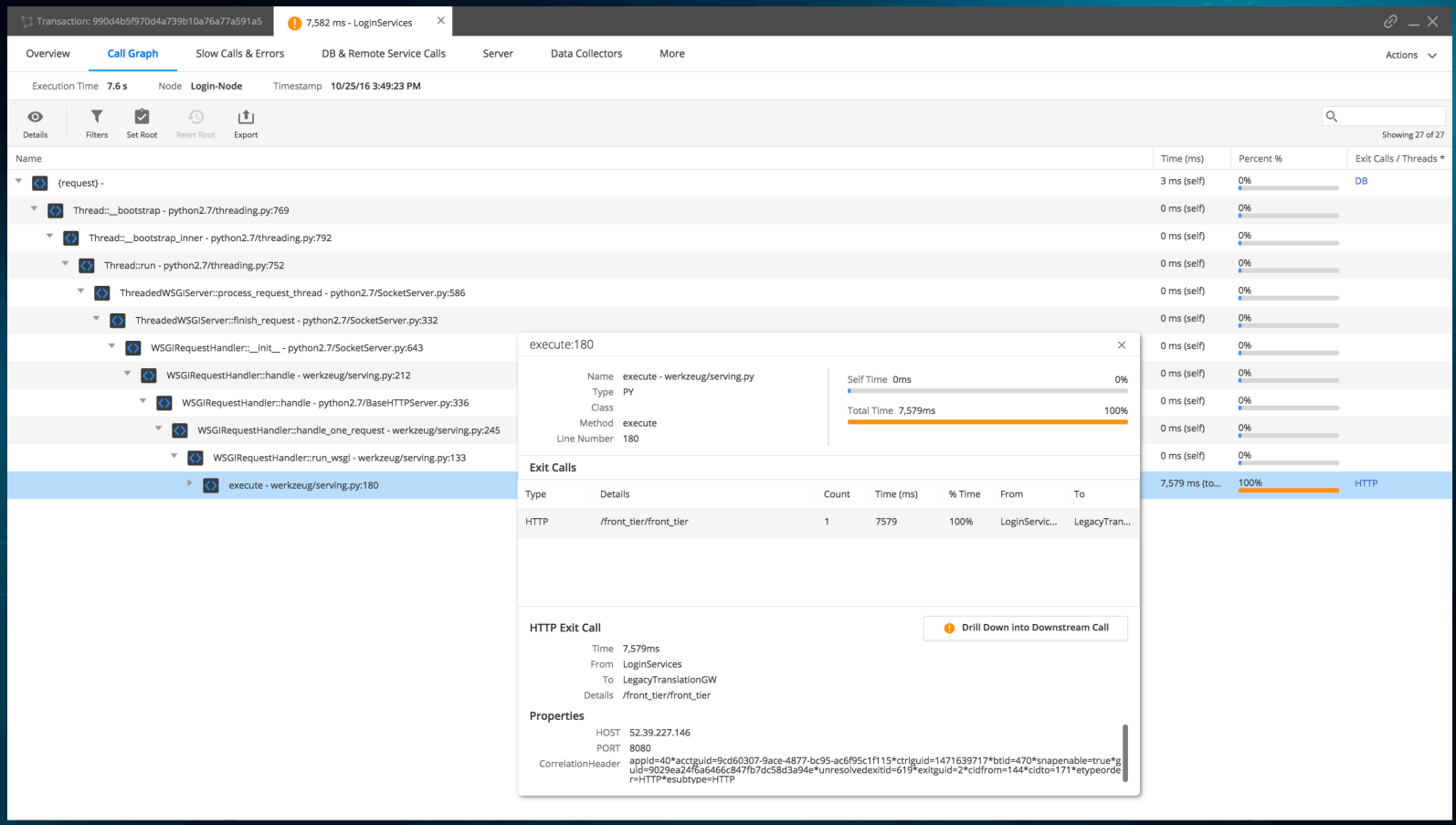


Identify flow per transaction

Troubleshoot individual transactions…



…down to code level



I can build that myself with ELK / zipkin / …

- Sure, it's only software… Let's get on with it!
  - Stand up Elastic Search and suck in your logs. Job done?
  - Now, standardise your log messages so they're consistent
  - Now, flow a transaction identifier end to end and add it to the messages, or use zipkin to trace transactions
- That's several man months of work already...
  … and you still have no code level visibility...
  … nor any easy to use UI to allow that essential "shift left"
- Now imagine you acquire a product or a company ☹

Bonus: architectural benefits

- Architectural governance
  - Per service dependency analysis
  - Manage Conway's law effects
- Evolve and manage risk in the system over time
- Data for capacity planning
- Achieve maturity with less risk of "hitting a wall"
- Evolution toward #noops?

The business case for unified monitoring

- Save developer time in production troubleshooting typically between 7-20% of developer time
- Save developer time in performance testing
- Save developer time implementing custom monitoring
- Save developer time in production handoff
- Ease staff on-boarding

Free 1 dev
Free .5 dev
Free .75 dev
Free .25 dev

"1 month to 1 day" UBS

- [SaaS] avoid running costs of monitoring back-ends

Summary

- Microservices shift complexity from development to production
- DevOps is not a panacea for production complexity: developers have day jobs!
- Production complexity needs to be managed to allow efficient (stress–free) operation
- AppDynamics offers an excellent, cost-justifiable, solution to these issues

Questions?

Register for a free trial!

www.appdynamics.com/free-trial