Making Infrastructure as Awesome as Agile Development
About Me

Bitten by the Agile bug in 2004; DevOps in 2011

Developer, Coach, Instructor, Architect,

Recently a “DevOps Engineer”

Software Maker

Markus Silpala  markus@silpala.com  @msilpala
How About You?
Making Infrastructure as Awesome as Agile Development
Agile Development

What's so Awesome about it?
Once Upon a Time...

http://xkcd.com/303/

Markus Silpala @msilpala
Before Agile

- Silos: Requirements → Code → Test → Deploy
- Large, infrequent releases
- Big suites of slow, manual regression tests
- Lots of bugs, tracked and rarely fixed
Agile

- Silos removed: Build -> Deploy
- "Whole Team" and Collective Ownership
- Frequent feedback loops and collaboration
- Automation of build, test, and many things
- Fewer bugs, caught and fixed sooner
- Happier team; happier customers
Technical Agile

- Test-Driven Development (TDD)
- Continuous Integration (CI)
- Simplification and Automation
- Refactoring (by the original meaning)
- Zero tolerance for bugs
Agile Mindset

- Small, tested increments of everything
- Collective ownership
- Team learning and knowledge sharing as a goal
- Unified vision and clear goals
- All voices heard
- Inspect and adjust—all the things!

Markus Silpala
"Process" Agile

- Many (most?) instances of Scrum, SAFe, etc.
- Focus on processes, ceremonies, activities
- Top-down implementation
- Assignment of roles and tasks
- "Scrum But"
AWESOME Agile

- Requires Technical and Mindset Agile
- Nearly no bugs
- Happy and confident team members*
- Happy and confident management
- Happy and confident customers
*Caveat*

- “Build -> Deploy” still leaves a silo wall in place
- The awesome often slows down or pauses around deployment time.
- Often the build team moves forward with new features while deployment is in progress.
Enter DevOps

Tearing down the next silo

Markus Silpala
DevOps Culture

- Deep Collaboration
- Incremental, tested change and improvement
- Rampant automation
- Frequent delivery
DevOps Tools

- Source control & programming languages
- API-driven cloud environments for ops (usually)
- Infrastructure-Automation tools
  - e.g. Puppet, Chef, Salt, Ansible
What's Awesome about DevOps

- Even more consistency between environments
- Even faster feedback on features—from real users!
- One less handoff: one less place to drop the ball
- Even faster feedback on operational features
- Greater production stability

http://www.slideshare.net/nicolefv/the-data-on-devops-making-the-case-for-awesome
And yet...
“Now that we've adopted agile and we do incremental, automatically-tested releases, I think I'd like a different team to handle the testing.”

-No one, ever, after being part of a strong agile team

“I think I'd rather just do development and let someone else handle the deployments and admin.”

-Several people I have met on strong teams
What's less awesome about DevOps

- Building is slow
- Being on call
- Testing is a lot less obvious
- Rubs against more of the high-friction org areas
  - Security, compliance, finance, etc.
- Even broader skills needed on the team
Making it More Awesome

or, maybe: making DevOps hurt less?

Markus Silpala
@msilpala
Slow Builds

THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY CODE'S COMPILING."

HEY! GET BACK TO WORK!

COMPILING!

OH. CARRY ON.

http://xkcd.com/303/

Markus Silpala

@msilpala
Slow Builds

THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:

"MY CODE'S COMPILING."

HEY! GET BACK TO WORK!

OH. CARRY ON.

CONVERGING!

http://xkcd.com/303/

Markus Silpala
Slow Builds

Potential fixes:

- Docker
- Vagrant + VirtualBox

Keep an eye on the new kid: Otto
Being On Call

- Empower the team to fix the issues that ring the pager
- Aim to fix every issue before it alerts again
- Make the on-call rotation a feedback driver: learn to fix and prevent issues
- Goal: get to zero pages in a typical rotation
Being On Call

- Compensate people for being on call—regardless of whether they’re paged

- Some people volunteer for more shifts

- Others may get to opt out

- Being on call with no pages becomes a prize to strive for
Testing is Not Obvious

Tools you can use

- ServerSpec
- Test Kitchen
- Goss
- bats
- ChefSpec (if you’re using Chef)
- Self-validating deployments
Testing is Not Obvious
Testing is Not Obvious

- Write a tiny bit of code
Testing is Not Obvious

- Write a tiny bit of code
- Manually validate it
Testing is Not Obvious

• Write a tiny bit of code

• Manually validate it

• Write a test for that manual validation; see that it passes
Testing is Not Obvious

- Write a tiny bit of code
- Manually validate it
- Write a test for that manual validation; see that it passes
- Change or revert the code; confirm that the test fails for the right reason
Testing is Not Obvious

• Write a tiny bit of code
• Manually validate it
• Write a test for that manual validation; see that it passes
• Change or revert the code; confirm that the test fails for the right reason
• Restore the code so the test passes again
Testing is Not Obvious

• Write a tiny bit of code
• Manually validate it
• Write a test for that manual validation; see that it passes
• Change or revert the code; confirm that the test fails for the right reason
• Restore the code so the test passes again
• Repeat
Testing is Not Obvious

- Write a tiny bit of code
- Manually validate it
- Write a test for that manual validation; see that it passes
- Change or revert the code; confirm that the test fails for the right reason
- Restore the code so the test passes again
- Repeat
Testing is Not Obvious

• Write a tiny bit of code

• Write a test for that manual validation; see that it passes

• Change or revert the code; confirm that the test fails for the right reason

• Restore the code so the test passes again

• Repeat
Testing is Not Obvious

- Write a tiny bit of code
- Write a test for that manual validation; see that it passes
- Change or revert the code; confirm that the test fails for the right reason
- Restore the code so the test passes again
- Repeat
Testing is Not Obvious

• Choose a tiny bit of functionality

• Write a test for that manual validation; see that it passes

• Change or revert the code; confirm that the test fails for the right reason

• Restore the code so the test passes again

• Repeat
Testing is Not Obvious

• Choose a tiny bit of functionality

• Write a test for that functionality; see that it fails for the right reason

• Restore the code so the test passes again

• Repeat
Testing is Not Obvious

• Choose a tiny bit of functionality

• Write a test for that functionality; see that it fails for the right reason

• Write the code so the test passes

• Repeat
Testing is Not Obvious

• Choose a tiny bit of functionality

• Write a test for that functionality; see that it fails for the right reason

• Write the code so the test passes

• Repeat
Tests become a burden

- Small code change requires lots of test changes
- Test suites take a long time to run
- False failures: tests are too brittle

This will happen.

- These are all part of getting better at TDD
- TDD needs practice and whole-team buy-in
Monitoring-Driven Development

- Substitute “alert” or “dashboard graph” for “test”

- Write the alert first, muted. Then build the feature to make it pass.

- Your “test suite” runs all the time, in every environment.
Monitoring-Driven Development

- [http://liquidchicken.org/2015/03/04/monitoring-driven-development.html](http://liquidchicken.org/2015/03/04/monitoring-driven-development.html)
- [http://benjiweber.co.uk/blog/2015/03/02/monitoring-check-smells/](http://benjiweber.co.uk/blog/2015/03/02/monitoring-check-smells/)
- [https://medium.com/@jefferysmith/metrics-driven-development-db804bfdc2ac#.n7z0kt6uh](https://medium.com/@jefferysmith/metrics-driven-development-db804bfdc2ac#.n7z0kt6uh)
More High-friction org areas

- Security
- Compliance
- Finance
- Executive Management (when things go wrong)
Even more skills needed

- Linux: RHEL vs CentOS vs Ubuntu vs Debian
- Networking: DNS, TCP, UDP, interfaces, ports, routes, netstat, tcpdump
- Installation: source vs binary; tarball vs package
- Services, processes, logging, CPU, RAM
- SSH, tunnels, private/public keys
Even more skills needed

Options:

- Fully-integrated team (dev and ops)
- Embedded ops specialists
- Separate ops team as part of customer community
Separate DevOps Team

https://www.linkedin.com/pulse/difference-between-agile-team-devops-benjamin-wootton

Markus Silpala  @msilpala
Aim High

http://timothyfitz.com/2009/02/10/continuous-deployment-at-imvu-doing-the-impossible-fifty-times-a-day/

- At peak, a prod deployment every 9 min
- 50 deployments/day on average
- 4.4 machine-hours of tests
- unit, functional and web (Selenium) tests
Wrap-Up

- Agile is (or can and should be) awesome
- DevOps is awesome, though not as mature as Agile
- Test everything. Automate as you go
- Do it as a team
- Stick through the discomfort
- Aim high!

Markus Silpala
Questions?
Mini-Retrospective

How good a use of your hour was this session?

(5) Best use of the last hour that I can think of

(4) Highly valuable use of my time

(3) Valuable

(2) Meh

(1) I wish I had left early

Markus Silpala     markus@silpala.com     @msilpala
Thank You!

Markus Silpala  markus@silpala.com  @msilpala