Scaling with Continuous Deployment

Web 2.0 Expo
New York, NY, September 29, 2010

Brett G. Durrett (@bdurrett)
Vice President Engineering & Operations, IMVU, Inc.
An online community where members use 3D avatars to meet new people, chat, create and have fun with their friends.
Who is my audience?

Mix of engineering / product?
Who is my audience?

Mix of engineering / product?

How many from a startup?
Who is my audience?

- Mix of engineering / product?
- How many from a startup?
- How many believe iterating on your product is critical to the success of your business?
How quickly can your business iterate?
Can I interest you in some Continuous Deployment?
What is Continuous Deployment?

- Engineer commits code
- 20 minutes later it is live in production
- Repeat for about 50 commits *per day*
“Maybe this is just viable for a single developer … your site will be down. A lot.”

“It seems like the author either has no customers or very understanding customers”

Responses to February 2009 blog posting about Continuous Deployment at IMVU
(at the time IMVU had a $12 million run rate)
Benefits

- Regressions easy to find, correct
- Releases have zero overhead
- Rapid iteration using real customer metrics
Finding and Fixing Problems

- Each release has few changes, 1-3 commits
- Production issues correlate with check-in timestamp
- No overhead to producing a new release to correct issue

Identifying cause takes minutes
CD at IMVU: Simple Overview

Local tests pass, engineer commits code

Lots and lots of tests run

All tests pass?

Yes → Code deployed to % of servers

No → Revert commit (Blocks)

Metrics good?

Yes → Code deployed to all servers

No → Rollback (Blocks)

Code deployed to all servers

Metrics still good?

Yes → Win!

No → Rollback (Blocks)
1. Continuous integration system
2. Production monitoring and alerting
   - System performance
   - Business metrics
   - Trending is nice too 😊
3. Simple deploy / roll-back system
Commit to Making Forward Progress

- Require coverage for all new code
- Add coverage for bugs / regressions
- Understand and fix root cause of failures
Expect Some Hurdles

- Production outages
- New overhead
  - Tests
  - Build systems
- Production outages
- Frustration
- Production outages

(but well worth it)
Dealing with SQL

Problems

• Difficult to roll-back schema
• Alter statements lock / impact customers

Solutions

• New schema has formal review process
• No alter on large tables, create new table
  – Copy on read
  – Complete migration with background job
• Developed on trunk, not branch
  – “hidden” from customers by A/B experiment
  – 100% control, add QA to experiment

• Deployed daily during development

• Slow roll-out by increasing experiment %
  – Experiment closed = fully launched
Slow tests burden to scaling
• Can’t run all tests in sandbox
• Faster to debug on build cluster

If possible…
• Keep tests fast
• Keep tests specific
As the team grows…

- More likely to have test failures
- More people blocked as a result

Intermittent failures *very bad*
Eliminate the root cause
• Won’t catch issues that fail slowly
  – SELECT * FROM growing_table WHERE 1

• Some critical areas cause hard lock-ups
  – MySQL
  – Memcached

• Lack of test coverage of older code
  – Not an issue if you start with test coverage
Does Continuous Deployment Scale?

- Technical staff ~50 people
- 10 million monthly unique visitors
- Peak ~130K concurrent IM client logins
- It’s a real business!
  - $40 million run rate
  - Profitable and doubled revenue in 2009
Biggest challenges come with growth of the engineering organization
Build systems are a critical service
Build systems are a critical service
Run them that way
<table>
<thead>
<tr>
<th>Metric</th>
<th>Max</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>build_time</td>
<td>34.0m</td>
<td>10.1m</td>
</tr>
<tr>
<td>push_time</td>
<td>4.3m</td>
<td>3.1m</td>
</tr>
<tr>
<td>push_rollback</td>
<td>1</td>
<td>1.0000</td>
</tr>
<tr>
<td>commit_to_live</td>
<td>n/a</td>
<td>13.2m</td>
</tr>
</tbody>
</table>
### Overall Availability

<table>
<thead>
<tr>
<th>BB Availability</th>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
<th>Thu</th>
<th>Fri</th>
<th>Sat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>82.50%</td>
<td>40.00%</td>
<td>88.33%</td>
<td>60.00%</td>
<td>86.67%</td>
<td>0.00%</td>
</tr>
</tbody>
</table>

This chart illustrates the overall availability across different days and times, with availability percentages shown in pie charts.
• Initial implementation sequential builds
  – Scaled okay to ~20 engineers
  – Like trains running every 20 minutes
  – One “red” blocks all following builds

• Solution: build isolation
  – Enable testing single build without deploy
  – “Red” build pulled, allow other builds to pass
• Custom test-file runner with JS GUI
• PHP SimpleTest
• Python's built-in unittest
• Selenium Core with in-house API wrapper
• YUITest for browser JS unit tests
• Erlang Eunit
• Buildbot
Current Systems

• > 15,000 tests

• 86 web build servers
  – 62 Linux
  – 24 Windows

• ~ 10 minutes on build servers

• Deploy to cluster of ~700 servers
• Continuous Deployment is possible!

• Starting earlier is easier - baby steps

• The value of being able to iterate outweighs the challenges
Questions?
Thank You!

Brett G. Durrett
bdurrett@imvu.com
Twitter: @bdurrett

IMVU recognized as:

Inc. 500

Red Herring 100:
http://bit.ly/bbz5Ex

Best Place to Work:

(and we're hiring)
http://www.imvu.com/jobs