Orchestrating the Continuous Delivery Process

@SachaLabourey
CEO and Founder, CloudBees
Continuous Delivery Overview

App Lifecycle

BUILD  TEST  STAGE  Deploy  Run

Process

Continuous Delivery

Continuous Integration  Deployment

Sub-Processes

Managed by...

...Enterprise  Development Teams

...SMB  Development Teams  Release / Ops Teams

Feedback Loop

DevOps

Development Teams
Continuous Delivery Defined

• For the business person:
  – Continuous delivery is a methodology that allows you to deliver new software and update existing software faster, with lower risk

• For the technical person:
  – Continuous delivery encompasses the set of activities you use to ensure you are ready to deploy code and configuration changes into production at all times

• In relation to Continuous Integration:
  – CD extends CI to include the concept of deployment and testing for production readiness
Automation

Do I really need that? Let’s see...
HP LaserJet FutureSmart Firmware (10M+ LOC)

- Development costs growing 2.5x from 2004-2008
- Up to 10 different branches (driven by each product release)
- 80-90% of resources just porting existing FW to new products
- Unable to add new products to the plans due to lack of FW resources
- 6 weeks + to get through a complete testing cycle (mainly manual)
- Ongoing customer issues with consistency and lack of features
- Marketing had essentially given up asking for FW innovations
HP LaserJet FutureSmart Firmware (10M+ LOC)

From 2008...

- Code Integration: 10%
- Detailed Planning: 20%
- Porting Code: 25%
- Current Product Support: 25%
- Manual Testing: 15%
- Capacity for Innovation: ~5%

... to 2011

- Continuous Integration: 2%
- Agile Planning: 5%
- One Main Branch: 15%
- One Branch CPE: 10%
- Most Testing Automated: 5%
- Capacity for Innovation: ~40%

Automation Everywhere!

Continuous Integration and Test System

Testing Levels

- Pre-commit testing. Done by the developer on his/her own development machine before committing to the main trunk (at the bare minimum, includes running "qbar" locally).

- Commit testing. Happens automatically at the time a developer commits new code to the main trunk of the SCM (broken up into Stage 1 and Stage 2 for easier auto-revert). This is now our "Integration Queueing" (IQ) system.

- Quick-turn broad-based testing by pillar (runs every 2 hours, across multiple test machines in parallel). Intended as a quick feedback loop to find broad-based failures from new commits in as narrow a commit window as feasible.

- Same purpose as L2, but on real hardware so can know that the full build, firmware download, boot, and general capability are working end-to-end. Runs every 4 hours.

- Full regression test suite of all automated tests. Kicks off at midnight daily and provides complete view of the quality of the system. If passing rate drops substantially, should have seen a dip in L2 testing (otherwise, update the L2 list).
It all started with developers: CI

Push code change ➔ Kick off build, tests, more tests ➔ Archive artifacts (library, .war file)
Deploy artifacts to staging area ➔ Connect to needed systems/services ➔ Run functional and system tests
Ops Has Different Needs Than Dev

“Treating Configuration as Code”
But the Toolchains Intersect for Dev and Ops

Inspired by: http://www.infoq.com/presentations/devops-patterns
©2014 CloudBees, Inc. All Rights Reserved
Why the Difference in Toolchain Choices?

- **Dev**
  - Speed, coding, functionality, quality
  - Favor local optimization on process, pet tools, fast feedback

- **Ops**
  - Auditing, security, compliance, availability, risk avoidance
  - Favor central control, reproducibility, traceability, process
  - Much larger set of constraints, rules, surrounding systems
  - Legacy issues

*Jenkins is common ground*
Continuous Integration

- Source Code
- Unit Tests
- Quality Analysis
- Binaries
Continuous Delivery Process

- Source Code (Dev)
- Unit Tests (QA)
- Quality Analysis (QA)
- Binaries (Dev)
- Perfs Environment (QA)
- Func Test Environment (QA)
- UAT Environment (QA)
- Staging Environment (OPS)
- Production Environment (OPS)
- Release (Dev)
- GO LIVE!

Diagram showing the flow of a continuous delivery process from source code to production environment.
Jenkins Orchestrates the Toolchain

Continuous Delivery with Jenkins

BUILD  TEST  STAGE  DEPLOY  RUN
CloudBees Continuous Delivery Platform
CloudBees Continuous Delivery Platform

CloudBees Network

SOLUTIONS

CORE

HYBRID INFRASTRUCTURE SUPPORT

OPERATIONS AT SCALE
Flexible, manageable & reliable @ scale

Hybrid deployments, abstraction layer

Cloud deployments benefit from elasticity/on-demand cloud attributes

Customer AWS VPC

CloudBees Managed Masters and Elastic Slaves

Customer Private Data Center

IaaS

Pivotal

AWS

CloudBees AWS Account
Many Build Jobs

Issues

• Requires many plugins
• Workflow definition distributed across many jobs
• Failures cause big problems!
• Limited re-usability

Build Pipeline View

Ref: http://www.infoq.com/articles/orch-pipelines-jenkins
Jenkins Workflow – A Big Deal for CD (coming soon!)

With Workflow

- New job type
- Single place for definition
- Retry-able and restartable
- Reusable workflows
- Text-based DSL

Build Pipeline View

New visualizations coming!
Conclusions

• The Dev and Ops automation tool chains to support continuous delivery are different, but intersect at Jenkins

• CloudBees offers a continuous delivery platform based on Jenkins that runs on-prem, in the cloud, or bridges those worlds simply and securely

• New features being surfaced in Jenkins are extending its reach even further to support continuous delivery