CI, CD, CD, and DevOps

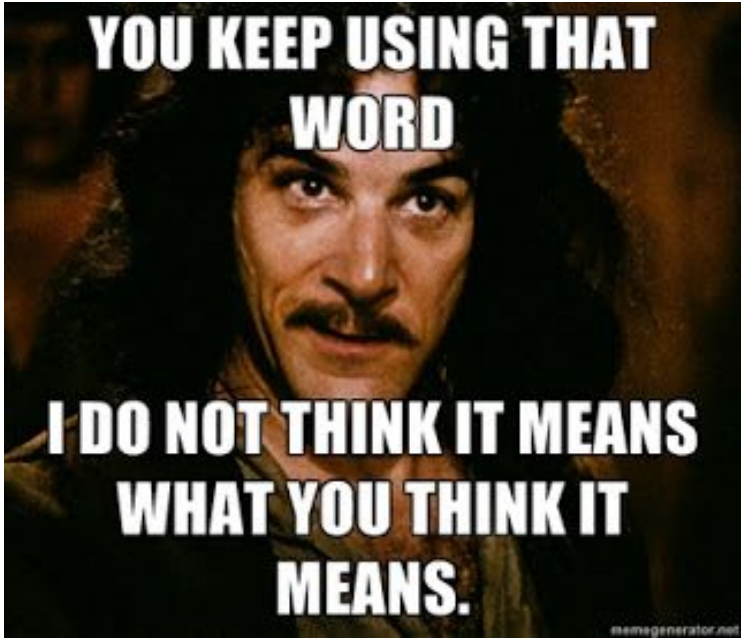
PAYCHEX®

Who Am I

- Senior Software Developer at Paychex
- Been there 9+ years
- Oracle OCA (PL/SQL) and OCE (SQL)

Goals of this presentation

- Define CI, CD, CD, and DevOps
- Elaborate on guiding principles of each
- Call out obstacles



Definitions

- Continuous Integration (CI)
  - Developers **integrate** code (push) to a shared repository very **frequently**
- Continuous Delivery (CD)
  - Every good build **is releasable** to production
- Continuous Deployment (CD)
  - Every good build actually **gets released** to production
- DevOps
  - **Culture** of **communication** and **collaboration** between Dev and Ops
  - **Culture** of **automation**

Continuous Integration (CI)

- **Goal**: Lessen the pain of integration by doing it more frequently

- Integrate early and often
  - Push changes to shared code base at least daily
  - Many advocate multiple daily pushes

- Integrations should be verified by automated processes
  - Shared builds
  - Static code analysis
  - Unit tests

- Integration failures are show stoppers and should be fixed immediately
  - Failures should happen in a “safe” place

What should CI provide?

- Ability to propagate changes rapidly
- Integration problems are detected early and quickly
- Integrating small changes is easer than integrating big changes
- Minimize parallel development conflicts
- Better communication between resources

CI Principles

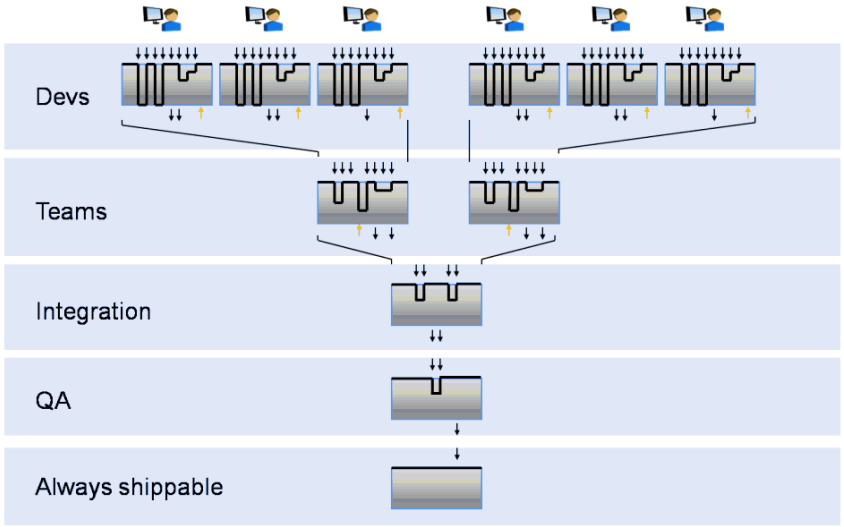
- Centralize builds and give everyone access to output and artifacts
- Maintain a single source control system
- Rapid integration into code base
- Builds are fast
- Automated feedback on integrations

CI Risks

- Pushing more frequently probably means more frequent failures
- Problems might propagate faster if safety guards are not in place
- Difficult to manage risk in large teams with varying or inconsistent schedules
- Automation is only as good as its ability to handle the unexpected
- Large shared things tend to suffer from the “tragedy of the commons”

Multi-Stage CI

- Integration in layers
  - Helps scales CI to large orgs with multiple teams
  - Safety nets/gates to manage risk



http://www.drdobbs.com/tools/multi-stage-continuous-integration/212201506?pgno=2

Continuous Delivery (CD)

- **Goal**: Every change that is entered is automatically certified for deployment
  - Includes configurations
  - Doesn’t include decision to deploy

- Continuous Delivery is the nexus of CI
  - It adds the final stages needed to deploy to production (including infrastructure)

- All environmental changes are made
  - Quickly
  - Safely
  - Sustainably

Selling points of continues delivery

- Reduce risk of releases
- Create sustainable processes
- Faster time to market
- Lower costs
- Improve productivity
- Happier people

Continuous Delivery Principles

- Continuous Delivery is a logical extension of CI
- Automated pipeline from dev checkin through production deployment
  - Release of deployment to prod is still optional/manual but it is push button
- Releases should be boring
  - Automate everything
  - Keep releases small
- Manage all configuration as code
  - All environments built equally through consistent, repeatable process
  - No snowflakes
- Strive to prevent DEV and OPS from breaking prod while enabling them to fix/improve it

Continuous Delivery risks

- You need really, really, really good automated tests/validations
- Automation is only as good as its ability to handle the unexpected
- Harder to do in a monolithic or stateful architectures
- Must be able to embrace failure and react quickly
- Risk may be too great in highly regulated industries

Continuous Deployment (CD)

- **Goal**: Deliver value to users by releasing all validated changes to production

- Logical extension of Continuous Delivery
  - Performs final step of auto deploying “green” builds to prod

Continuous Deployment Benefits

- Beyond Continuous Delivery, there isn't too much added
- Maximizes delivery of changes to production without human interaction
  - Might not always be a good thing…

Continuous Deployment Principles

- Applications/systems are designed for failure
  - Crumple zones
  - Penalty boxes
  - Safety nets
- Be able to rollback bad changes quickly
- Servers are cattle
  - Don’t name them

DevOps

- **Goal**: Create a culture that facilitates the CD's in an Agile environment
- DevOps ≠ CD ≠ Agile
- The "C"s of DevOps Culture (I'm copyrighting this)
  - Collaboration
  - Cooperation
  - Communication
  - Compassion
  - Cross-functional teams
    - not to be confused with cross-functional resources

DevOps Guiding principles

- DevOps is beyond tools, automation, and resource scheduling
- Teams should have access to all of the dev and ops personnel they need to complete their tasks on schedule (whether Agile or Waterfall)
- Everyone shares responsibility for the finished state of the whole stack
  - Dev, Ops, Test, Management, Product, Program etc.
  - Emphasize freedom, responsibility, and empathy
  - You break it, you fix it
  - Help your partners succeed
- DevOps embraces failure
  - Build safety nets
  - Learn from failure
  - Do better next time

DevOps is not…

- DevOps is about Continuous Delivery
  - Though it could help you get there or do it better
- DevOps is not a job title or an organization
  - You don’t do it by yourself
  - You are not “serviced by” DevOps
- DevOps is not about technology
  - You don’t need the Cloud or Docker
  - Legacy apps welcome
- DevOps is not solely about automation
  - You can still practice DevOps if you fall short of CD
- DevOps is not easy
  - It’s a massive culture shift for most established organizations

Recap of Themes

- Build smaller so you can build faster
- Everything should self test/validate
- Isolate, automate, and version every change
- Break down barriers between Dev and Ops

What do you need to do to get started?

- You need management buy in
- You need to define what these concepts mean for your organization
  - Be careful about trying to cram yourself into someone else's solutions
  - Do what is valuable to your business
- Set results driven goals to keep the focus on what delivers value
- "Kaizen" your process. No one nails it right away.

Managing up

- Bad:
  - We need to do mainline development because that gets us to CI
  - We need to do CD because everyone's doing it
  - We need to automate everything

- Good:
  - We want to decrease our cycle time so users get value faster
    - A routine bug fix should be able to go from checkin to prod in X days
    - Our scheduled release cadence should be X downtime releases and Y online release per month
  - It takes too long to test process A so we should develop an auto test suite that can be completed in X business days
    - The XYZ team is going to use sprints 4 and 5 to automate it

What is Paychex up to?

- We are trying to go all in on Agile, CI, CD, DevOps
- Started a few years ago with a mandate from our SVP of Product Development/IT (the top)
- We are making progress in some areas, struggling in others
- It’s hard in a huge Enterprise org in a highly regulated industry

Technologies in the Pipeline

- VMWare
- OpenShift
- Puppet
- Etcd
- Docker
- Delphix
- Bitbucket (Atlassian Git)
- Jenkins
- XLD (Deploys)
- XLR (Orchestration)
- AppD
- Splunk
- EM
- Jira/Confluence

Daas

- We are heavily leveraging Delphix
  - Developer self service for their isolation environments
  - Complex “post refresh” scripts to manage our test environments
- All DDL, code, and configs from dev are source controlled
  - Built through an automated process
  - Staged and deployed via push button processes
  - Does not include DBA stuff (yet)
- Struggling with test data management (not because of Delphix)
  - Culture change to get people to automate their tests/configs

The End

PAYCHEX®