Continuous Integration & Deployment

Kristiina Rahkema
According to wikipedia:

"Software development is the process of **conceiving**, **specifying**, **designing**, **programming**, **documenting**, **testing**, and **bug fixing** involved in creating and maintaining applications, frameworks, or other software components."

Today:

- Continues integration
- Continuous delivery
- Continuous deployment
- Benefits, challenges, different approaches

Beneficial concepts, but not a silver bullet
Definition

Continuous integration
Continuous integration

Definition in Wikipedia:
• Practice of merging copies of developers working code to master several times a day

Definition by Martin Fowler:
• Members of team integrate their work frequently - at least daily
• Each integration is verified by automatic build and test

Definition by Atlassian
• Developers merge their changes to master as often as possible
• Changes are validated by automatic build and test

Definition by many developers
• Using a CI tool/server to run tests automatically on feature branches
• Merging code before release
Patterns and anti-patterns

Anti-pattern:
"..just like a pattern, except that instead of a solution it gives something that looks superficially like a solution but isn’t one."

- Martin Fowler

- Infrequent commits
- Feature branches
- Long broken build
- Build time
- Poor code coverage
- Build feedback
- Spam notifications
- Code reviews
Patterns and anti-patterns 1/8

Frequency of code commits/pushes

- **Anti-pattern:** Infrequent code commits/pushes
  - Problem:
    - Large number of changes
    - Developers don’t push code
    - Big merge conflicts
  - Solution:

- **Pattern:**

---

- **Infrequent commits**
  - Feature branches
  - Long broken build
  - Build time
  - Poor code coverage
  - Build feedback
  - Spam notifications
  - Code reviews
Patterns and anti-patterns 1/8

Frequency of code commits/pushes

- **Anti-pattern:** Infrequent code commits/pushes
  - Problem:
    - Large number of changes
    - Developers don’t push code
    - Big merge conflicts
  - Solution:
    Split features into smaller tasks
- **Pattern:** Commit/push code as frequently as possible (multiple times a day)

- **Infrequent commits**
  - Feature branches
  - Long broken build
  - Build time
  - Poor code coverage
  - Build feedback
  - Spam notifications
  - Code reviews
Patterns and anti-patterns 2/8

Branches

- **Anti-pattern: Feature branches**
  - Why? - Allows parallel implementation of different features
  - Problem:
    - Long lived feature branches
    - No changes from mainline -> "Continuous isolation"
    - Very difficult merges
  - Solution:

- **Pattern:**

- **Feature branches**
  - Infrequent commits
  - Long broken build
  - Build time
  - Poor code coverage
  - Build feedback
  - Spam notifications
  - Code reviews

- **Code reviews**
  - Anti-pattern: require code review before merge
  - Why? - Senior developers should check changes
  - Problem:
    - Merges to trunk are gated
    - Longer time between commit and feedback
  - Solution:
    - Pair programming
    - Asynchronous code reviews after commit
Branches

• **Anti-pattern: Feature branches**
  • Why? - Allows parallel implementation of different features
  • Problem:
    • Long lived feature branches
    • No changes from mainline -> "Continuous isolation"
    • Very difficult merges
  • Solution:
    • Using feature toggles instead of feature branches
    • Short lived branches per task (NB: not feature)
• **Pattern: Trunk based development**

• **Infrequent commits**
• **Feature branches**
  • Long broken build
  • Build time
  • Poor code coverage
• **Build feedback**
• **Spam notifications**
• **Code reviews**
Patterns and anti-patterns 3/8

Broken build

• **Anti-pattern: Long broken build/red status**
  • Why? - All changes are pushed into master
  • Problem:
    • Broken build blocks master
    • Others cannot push code

• Solution:

• **Pattern:**


• Infrequent commits
• Feature branches
• **Long broken build**
• Build time
• Poor code coverage
• Build feedback
• Spam notifications
• Code reviews
Broken build

• **Anti-pattern: Long broken build/red status**
  • Why? - All changes are pushed into master
  • Problem:
    • Broken build blocks master
    • Others cannot push code

• Solution:
  • Fix broken build quickly
  • Or revert commit

• **Pattern: fixing broken build takes priority**

• Infrequent commits
• Feature branches
• **Long broken build**
• Build time
• Poor code coverage
• Build feedback
• Spam notifications
• Code reviews
Patterns and anti-patterns 4/8

• Build time
  • Anti-pattern: long build time
    • Why? - Need to run all kinds of test suits
    • Problem:
      • Developers need to wait for results before moving on
      • Infrequent checkins
    • Solution:

  • Pattern:

• Infrequent commits
• Feature branches
• Long broken build
• Build time
• Poor code coverage
• Build feedback
• Spam notifications
• Code reviews
Patterns and anti-patterns 4/8

- Build time
  - Anti-pattern: long build time
    - Why? - Need to run all kinds of test suits
    - Problem:
      - Developers need to wait for results before moving on
      - Infrequent checkins
    - Solution:
      - Only run unit tests with build
      - Other tests can be run later on in the pipeline
  - Pattern: build time as short as possible, fast feedback
  
- Infrequent commits
- Feature branches
- Long broken build
  
- Build time
  - Poor code coverage
  - Build feedback
  - Spam notifications
  - Code reviews
Patterns and anti-patterns 5/8

• How well code is tested
  • Anti-pattern: Poor code coverage
    • Why? - Implementing CI methodologies in old project/missing guidelines
    • Problem:
      • Status is green
      • Still bugs and problems in build
    • Solution:
  • Pattern:

• Infrequent commits
• Feature branches
• Long broken build
• Build time
• Poor code coverage
• Build feedback
• Spam notifications
• Code reviews
Patterns and anti-patterns 5/8

• How well code is tested
  • Anti-pattern: Poor code coverage
    • Why? - Implementing CI methodologies in old project/missing guidelines
    • Problem:
      • Status is green
      • Still bugs and problems in build
    • Solution:
      • Better code coverage
      • Including tests with all new code
  • Pattern:
    • Rely on test where coverage is good enough
    • When discovering a bug, always add tests that cover it

• Infrequent commits
• Feature branches
• Long broken build
• Build time
• Poor code coverage
• Build feedback
• Spam notifications
• Code reviews
• Build success/failure feedback
  • Anti-pattern: no feedback
    • Why? - Not wanting to spam developers
    • Problem:
      • No-one will fix the build
    • Solution:

• Pattern:

• Infrequent commits
• Feature branches
• Long broken build
• Build time
• Poor code coverage
• Build feedback
• Spam notifications
• Code reviews
Patterns and anti-patterns 6/8

- Build success/failure feedback
  - Anti-pattern: no feedback
    - Why? - Not wanting to spam developers
    - Problem:
      - No-one will fix the build
    - Solution:
      - Use a dashboard
      - Automatic e-mail/slack notifications
  - Pattern: notify developers of broken build

- Infrequent commits
- Feature branches
- Long broken build
- Build time
- Poor code coverage
- Build feedback
- Spam notifications
- Code reviews
Patterns and anti-patterns 7/8

• Notifications
  • Anti-pattern: spam notifications
    • Why? - Wanting to give detailed feedback to developers
    • Problem:
      • Too many notifications
      • E.g. for each passing build in a big team
      • Ignoring messages
      • Overlooking failed builds
    • Solution:

• Pattern:

• Infrequent commits
• Feature branches
• Long broken build
• Build time
• Poor code coverage
• Build feedback
  • Spam notifications
• Code reviews
Patterns and anti-patterns 7/8

• Notifications
  • Anti-pattern: spam notifications
    • Why? - Wanting to give detailed feedback to developers
    • Problem:
      • Too many notifications
      • E.g. for each passing build in a big team
      • Ignoring messages
      • Overlooking failed builds
    • Solution:
      • Notify only when checkin fails
      • Notify when build is fixed
  • Pattern: give feedback, but keep notifications to a minimum

• Infrequent commits
• Feature branches
• Long broken build
• Build time
• Poor code coverage
• Build feedback
• Spam notifications
• Code reviews
Patterns and anti-patterns 8/8

- **Code reviews**
  - **Anti-pattern: require code review before merge**
    - Why? - Senior developers should check changes
  - Problem:
    - Merges to trunk are gated (trunk = master/main)
    - Longer time between commit and feedback
  - Solution:

- **Pattern:**

- **Infrequent commits**
- **Feature branches**
- **Long broken build**
- **Build time**
- **Poor code coverage**
- **Build feedback**
- **Spam notifications**
- **Code reviews**
• Code reviews
  • Anti-pattern: require code review before merge
    • Why? - Senior developers should check changes
  • Problem:
    • Merges to trunk are gated (trunk = master/main)
    • Longer time between commit and feedback
  • Solution:
    • Pair programming
    • Asynchronous code reviews after commit
• Pattern: Pair programming

• Infrequent commits
• Feature branches
• Long broken build
• Build time
• Poor code coverage
• Build feedback
• Spam notifications
• Code reviews
Benefits of continuous integration

• Avoid merge conflicts by merging to master often
  • Trunk-based development  \( (\text{trunk} = \text{master/main}) \)
  • Or short lived branches for tasks
• Collaboration and co-ownership
  • Everyone works on master, is responsible for green build
• Fast feedback and reducing risks
  • Discovering mistakes early
• CI make it easier to implement continuous delivery and deployment
Continuous delivery and deployment

Continuous Integration
- Unit Test
- Platform Test
- Deliver to Staging
- Application Acceptance tests
- Deploy to Production
- Post deploy tests
  - Auto
  - Manual
  - Auto

Continuous Delivery
- Unit Test
- Platform Test
- Deliver to Staging
- Application Acceptance tests
- Deploy to Production
- Post deploy tests
  - Auto
  - Manual
  - Auto

Continuous Deployment
- Unit Test
- Platform Test
- Deliver to Staging
- Application Acceptance tests
- Deploy to Production
- Post deploy tests
  - Auto
  - Auto
  - Auto

Derived from: https://puppet.com/blog/continuous-delivery-vs-continuous-deployment-what-s-diff
Benefits of continuous deployment

- Reliability
- Smaller changes
- Deliver software with fewer bugs
- Easier to roll back changes
- Feedback
- Release new features early and often, more feedback
- Quickly responding to market conditions
- Collaboration
- Shared responsibility and better collaboration
Benefits of continuous deployment

• Reliability
  • Smaller changes
  • Deliver software with fewer bugs (per time unit)
  • Easier to roll back changes

• Feedback
  • Release new features early and often, more feedback
  • Quickly responding to market conditions

• Collaboration
  • Shared responsibility and better collaboration
Who can implement continuous deployment?
Who can implement continuous deployment?

- Web applications
- Applications on devices connected to the internet
- Applications on devices sometimes connected to the internet
- More difficult:
  - Applications on devices that are not connected to the internet, manual deployment needed
Difficulties (converting from conventional deployment)

- Processes
  - Can everything be automated?
  - Feature branches?
  - Setup of code reviews

- Processes
  - Developer habits
  - Testing
  - Investment in tools and hardware
Difficulties (converting from conventional deployment)

- Developer habits
  - Need to take CD into account when writing code
  - Architecture that allows small changes
  - Database migrations
  - Take responsibility
  - Small changes

- Processes
- Developer habits
- Testing
- Investment in tools and hardware
Difficulties (converting from conventional deployment)

• Testing
  • Adding tests to existing code
  • All new code with tests
  • Adding tests when discovering bugs
  • When are there enough tests to trust the system?

• Processes
  • Developer habits
  • Testing
  • Investment in tools and hardware
Difficulties (converting from conventional deployment)

• Investment in tools and hardware
  • Automatic testing
  • Staging environment
  • Automating everything

• Processes
• Developer habits
• Testing
• Investment in tools and hardware
Different flavours

• Minimum in-service deployments
  • Set minimum number of instances that stay in service
  • Deploy new version to other instances

• Disadvantages:
  • Multiple stages, needs support
  • Difficult for infrastructure changes
  • Changes on live servers, possible recovery time on fails

• Advantages:
  • Increased testing capability, possible changes during deployment
  • No downtime, no additional infrastructure
  • Often quicker than rolling deployments

• Minimum in-service deployment
  • Rolling application updates
  • Blue/Green deployment
  • A/B testing
Different flavours

• Rolling deployments
  • Set maximum number of instances to update
  • Deploy new version to these instances
  • When ready start with next instances
• Disadvantages:
  • More complex than minimum in-service
  • Less efficient in deployment time
• Advantages:
  • No downtime
  • Pausing possible, limited multi version testing

• Minimum in-service deployment

• Rolling application updates
  • Blue/Green deployment
  • A/B testing
Different flavours

• Blue/Green deployment
  • Blue: legacy stack, green: new deployment
  • Entire infrastructure replicated in green environment with new application
  • Tests are run on green environment
  • Tests succeed, then users routed from blue to green
  • Blue environment is shut down
  • Disadvantages:
    • Advanced orchestration tooling, some additional costs
    • Some risks due to same database
    • User traffic floods servers
  • Advantages:
    • Reduced risk since no changes in infrastructure needed
    • Near zero downtime
    • Clean and controlled switch
    • Possible to test entire environment before switch

• Minimum in-service deployment
• Rolling application updates
• Blue/Green deployment
• A/B testing
Different flavours

- **A/B testing**
  - Deployment identical to Blue/Green
  - Small percentage of traffic sent to new green environment
- Disadvantages:
  - More moving parts
  - More complex
  - Requires full automation of everything
- Advantages:
  - Predictably scale capacity
  - Can test new features
  - Customer validation without huge impact and errors

- Minimum in-service deployment
- Rolling application updates
- Blue/Green deployment
- **A/B testing**
Examples

• Etsy - 50 deploys/day
• Facebook - quasi continuous deployment
• HP LaserJet firmware - continuous delivery for non web based software
• Authentiq - continuous delivery for mobile applications
Etsy

• Keeping Trunk clean
  • Each developer: full Etsy stack on virtual machine
  • Test changes in Jenkins before committing changes to trunk

• Process
  • Test locally
  • Deliver to staging (accessed by employees)
  • One-click deployments
  • Config/feature flags - enable and disable feature or its variants
  • Monitoring and detecting anomaly patterns

https://codeascraft.com/2010/12/08/track-every-release/
Fast releases at scale? - quasi continuous deployment

- Tens to hundreds of commits released every couple of hours
- First deployed for employees
- Emergency stop button
- Then to 2% of production
- Then to 100% of production

https://engineering.fb.com/web/rapid-release-at-massive-scale/
• Problem:
  • 2006-2007: difficulty keeping up with demand for new innovative features
  • 400-800 developers, 2 software releases per year
  • Majority of time spent on porting software to new products
  • Only 5% of time spent on new features

• Technical problem:
  • 12-15% of time spent on integrating code into trunk
  • 1 week from check-in to feedback
  • 6 weeks to get through complete testing cycle, 8 weeks for final feedback

• Change:
  • Eliminated separate branches
  • Automated testing
  • Culture where broken build stopped additional work

• Results:
  • Build cycle 3 hours (instead of 1 week)
  • 100 commits/day (instead of 1/day)
  • Test cycle 24 hours (instead of 6 weeks)
  • 40% of time spent on writing new features (instead of 5%)
Authentiq

• Continuous delivery for mobile apps
• Their goal
  • Build upon and integrate with git
  • Run tests after every commit
  • New build available for testers automatically
  • Parallel feature development
  • Automate publishing to app stores
  • Process to roll out hot-fixes
  • Build notifications in Slack
• Still using feature branches
• Using git flow to automate process

Tools

- Self hosted
  - Open source: Abstruse, Badwolf, CDS: Continuous Delivery Service, Concourse, Evergreen, GoCD, Hydra, Jenkins, Previs
  - Commercial: AppVeyor, Azure DevOps, Buildkite, Circle CI, Codacy, Codefresh, Drone
- Cloud based
  - Traditional CI tools: AppVeyor, AWS CodeBuild, Azure DevOps, CircleCI, Cirrus CI, Codacy, Code Climate, Codefresh, Codeship, Drone, Gitlab CI, Scrutinizer, Semaphore, shippable, TeamCity, Travis-ci, Bitbucket pipelines
  - Other: Appcenter (mobile applications), Assertible (Web services), Bitrise (mobile applications), continuousphp (php only), coveralls (code coverage), sourcelevel (code review), houndci (code style), Probo.CI (data driven applications), rocro (code review, load testing, documentation), saucelabs (web and mobile apps), sider (analysis with rulesets), styleci (web coding standards)
Thank you!
Any questions?
• Different tools:
  • https://github.com/ligurio/awesome-ci
  • https://blog.gds-gov.tech/that-ci-cd-thing-principles-implementation-tools-aa8e77f9a350

• Real life examples:
  • https://www.infoq.com/news/2014/03/etsy-deploy-50-times-a-day/
  • https://engineering.fb.com/web/rapid-release-at-massive-scale/
  • https://www.authentiq.com/blog/2018/02/23/continuous-delivery-mobile-apps/

• Different ways of doing CI:
  • https://www.reddit.com/r/devops/comments/cn9209/most_projects_dont_use_continuous_integration_and/

• CI anti-patterns:
  • https://techbeacon.com/devops/are-you-really-doing-continuous-integration-heres-how-tell
  • https://hackernoon.com/anti-patterns-of-continuous-integration-e1cafd47556d

• Different flavours:
  • https://dzone.com/articles/docker-amp-continuous-delivery-deployment-types

• Code reviews:
  • https://groups.google.com/forum/#!msg/continuousdelivery/LIJ1nva9Oas/y3sAaMtibGAJ
  • https://softwareengineering.stackexchange.com/questions/121664/when-to-do-code-reviews-when-doing-continuous-integration