Continuous Development & Integration

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
Continuous integration

Definition
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

- Checkin frequency
- Branches
- Broken build
- Build time
- Test coverage
- Build feedback
- Notifications
- Code reviews
Patterns and anti-patterns 1/8

Checkin frequency

- **Anti-pattern**: Infrequent checkins
  - Problem:
    - Large number of changes
    - Developers don’t check in code
    - Big merge conflicts
  - Solution:

- **Pattern**:

- **Checkin frequency**
  - Branches
  - Broken build
  - Build time
  - Test coverage
  - Build feedback
  - Notifications
  - Code reviews
Checkin frequency

- **Anti-pattern**: Infrequent checkins
  - Problem:
    - Large number of changes
    - Developers don’t check in code
    - Big merge conflicts
  - Solution:
    - Split features into smaller tasks
- **Pattern**: Check in code as frequently as possible (multiple times a day)

- **Checkin frequency**
  - Branches
  - Broken build
  - Build time
  - Test coverage
  - Build feedback
  - 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:**

• Checkin frequency
• **Branches**
  • Broken build
  • Build time
  • Test coverage
  • Build feedback
  • 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:
    - Using feature toggles instead of feature branches
    - Short lived branches per task (NB: not feature)
- **Pattern: Trunk based development**

- **Checkin frequency**
- **Branches**
  - Broken build
  - Build time
  - Test coverage
  - Build feedback
  - Notifications
  - Code reviews
Broken build

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

- **Solution:**

- **Pattern:**

- **Checkin frequency**
- **Branches**
- **Broken build**
- **Build time**
- **Test coverage**
- **Build feedback**
- **Notifications**
- **Code reviews**
Broken build

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

- **Solution:**
  - Fix broken build quickly
  - Or revert commit

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

- **Checkin frequency**
- **Branches**
- **Broken build**
- **Build time**
- **Test coverage**
- **Build feedback**
- **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:
• 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
• Checkin frequency
• Branches
• Broken build
  • Build time
  • Test coverage
  • Build feedback
  • Notifications
  • Code reviews
Patterns and anti-patterns 5/8

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

• Pattern:

• Checkin frequency
• Branches
• Broken build
• Build time
• Test coverage
• Build feedback
• Notifications
• Code reviews
Patterns and anti-patterns 5/8

- Test coverage
  - Anti-pattern: Poor test coverage
    - Why? - Implementing CI methodologies in old project/missing guidelines
    - Problem:
      - Status is green
      - Still bugs and problems in build
    - Solution:
      - Better test 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

- Checkin frequency
- Branches
- Broken build
- Build time
- Test coverage
- Build feedback
- Notifications
- Code reviews
Patterns and anti-patterns 6/8

• Build feedback
  • Anti-pattern: no feedback
    • Why? - Not wanting to spam developers
    • Problem:
      • No-one will fix the build
    • Solution:

• Pattern:

• Checkin frequency
• Branches
• Broken build
• Build time
• Test coverage
• Build feedback
• Notifications
• Code reviews
Patterns and anti-patterns 6/8

• Build 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

• Checkin frequency
• Branches
• Broken build
• Build time
• Test coverage
• Build feedback
• 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:

• Checkin frequency
• Branches
• Broken build
• Build time
• Test coverage
• Build feedback
• 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

• Checkin frequency
• Branches
• Broken build
• Build time
• Test coverage
• Build feedback
• 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
      • Longer time between commit and feedback
    • Solution:

• Pattern:
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
    • Longer time between commit and feedback
  • Solution:
    • Pair programming
    • Asynchronous code reviews after commit
  • Pattern: Pair programming

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

• Avoid merge conflicts by merging to master often
  • Trunk-based development
  • 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 CD
Continuous delivery and deployment

Continuous Integration

Continuous Delivery

Continuous Deployment

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

- 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 delivery

• 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
Who can implement continuous delivery

• 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 delivery needed
Who can implement continuous delivery

- 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 delivery 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 minim 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 due to immutable infrastructure
  • 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
  • Deploy 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 delivery

- 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
  • 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)
• Add CI settings file to project folder
• Add new file to git and commit
• Gitlab runs tests automatically
Gitlab

- Possible to see terminal output
• Setup new pipeline
### Bitbucket

- Bitbucket gives suggested setup file
- Change file to run "./gradlew test"

```yaml
# This is a sample build configuration for Java (Gradle).
# Check our guides at https://confluence.atlassian.com/x/zw-5Mw for more examples.
# Only use spaces to indent your .yml configuration.
# -----
# You can specify a custom docker image from Docker Hub as your build environment.
image: openjdk:8

pipelines:
default:
  - step:
     caches:
       - gradle
     script: # Modify the commands below to build your repository.
       # You must commit the Gradle wrapper to your repository
       # https://docs.gradle.org/current/userguide/gradle_wrapper.html
       - bash ./gradlew test
```

**Your configuration looks good!**

Simply commit the `bitbucket-pipelines.yml` file to kick off your CI/CD pipeline.
Bitbucket

<table>
<thead>
<tr>
<th>Pipeline</th>
<th>Status</th>
<th>Started</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td><img src="https://bitbucket.org/repo" alt="R" /></td>
<td><strong>Initial Bitbucket Pipelines configuration</strong>&lt;br&gt;Reklaam  a520105  master</td>
<td><img src="https://i.imgur.com/3VJ5y.png" alt="Green Check" /> <strong>Successful</strong>&lt;br&gt;10 minutes ago</td>
</tr>
</tbody>
</table>

- Bitbucket commits file, runs task
Bitbucket

- Possible to see terminal output
Bitbucket - pipeline example

Gitlab - pipeline example

task: remove double auth

12 jobs from task/remove-double-auth in 12 minutes 35 seconds (queued for 3 seconds)

https://medium.com/@ryzmen/gitlab-fast-pipelines-stages-jobs-c51c829b9aa1
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://engineering.fb.com/web/rapid-release-at-massive-scale/

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/y3sAaMtibGAl
- https://softwareengineering.stackexchange.com/questions/121664/when-to-do-code-reviews-when-doing-continuous-integration