Cloud Computing

Chapter 5: Continuous Integration

Summer Term 2017

Complex and Distributed IT Systems
TU Berlin

DevOps

- Developers (Dev) develop without knowledge of productive infrastructure
- Operators (Ops) push code as a black box as they are not involved in development
- Source of conflicts: failure in server or in development?
- Integrate both processes into continuous process (delivery, integration, …)

Continuous Integration, Deployment and Delivery

- Continuous Integration
  - Integrating changes from different developers in the team into a mainline as early as possible
- Continuous Deployment
  - Keeping the application deployable at any point or even automatically releasing to a test or production environment if the latest version passes all automated tests
- Continuous Delivery
  - Keeping the codebase deployable at any point and have all the configuration necessary to push it into production.

Continuous Integration

- Software development practice where members of a team integrate their work frequently, at least daily
- Term Continuous Integration originated from the Extreme Programming development process as one of its original twelve practices
- Basic Idea
  - Developers integrate code into a shared repository frequently (several times a day)
  - Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible
  - Small changes introduced to code $\Rightarrow$ Quick detection and localization of failures possible

Example

(martinfowler.com/articles/continuousIntegration.html)

- Development of a piece of code within several hours
  - Checkout the integrated source onto the local development machine and create a working copy from the mainline
  - Extend and/or modify the working copy $\Rightarrow$ altering the production code and adding/changing automated tests (e.g. a version of XUnit testing frameworks)
  - At the end or in regular intervals an automated build on the development machine is executed creating local executable and running automated tests
  - Once all builds and tests completed without errors, the overall build is considered to be good
  - Developed version is checked in back into the repository
- Problem occurs, if meanwhile additional changes arrived

Example

(martinfowler.com/articles/continuousIntegration.html)

- Extension in continuous delivery
  - Update my working copy with modifications and rebuild
  - Incompatible changes lead to failure in compilation or tests
  - Own responsibility to fix and repeat until a working copy is built that is properly synchronized with the mainline
  - Final commit of the changes into mainline allowed, updating repository
  - Next build performed on integration machine with the mainline code, as there is always a chance that something is missed on the local machine and the repository wasn't properly updated

Example

(martinfowler.com/articles/continuousIntegration.html)

- Summary
  - If a clash occurs between two developers, it is caught when the 2nd developer commits builds or the integration build fails
  - Either way the error is detected and fixed rapidly

⇒In CI environment a failed integration build never stay failed for long

⇒Everybody develops off a shared stable base and never gets so far away from that base that it takes very long to integrate back with it

Principles of Continuous Integration

- Original principles of Continuous Integration presented by Martin Fowler in 2006
  - Maintain a Single Source Repository
    - A single repository (GitHub, subversion, …) keeps track of all changes
  - Automate the builds
    - Getting the sources turned into a running system is a complicated process involving compilation, moving files, loading schemas into the databases, etc.
    - Automation with scripts such as make, Ant, MSbuild, …
    - Include everything in the automated build ⇒ anyone should be able to bring in a virgin machine, check the sources out of the repository, issue a single command, and have a running system on their machine.

Principles of Continuous Integration

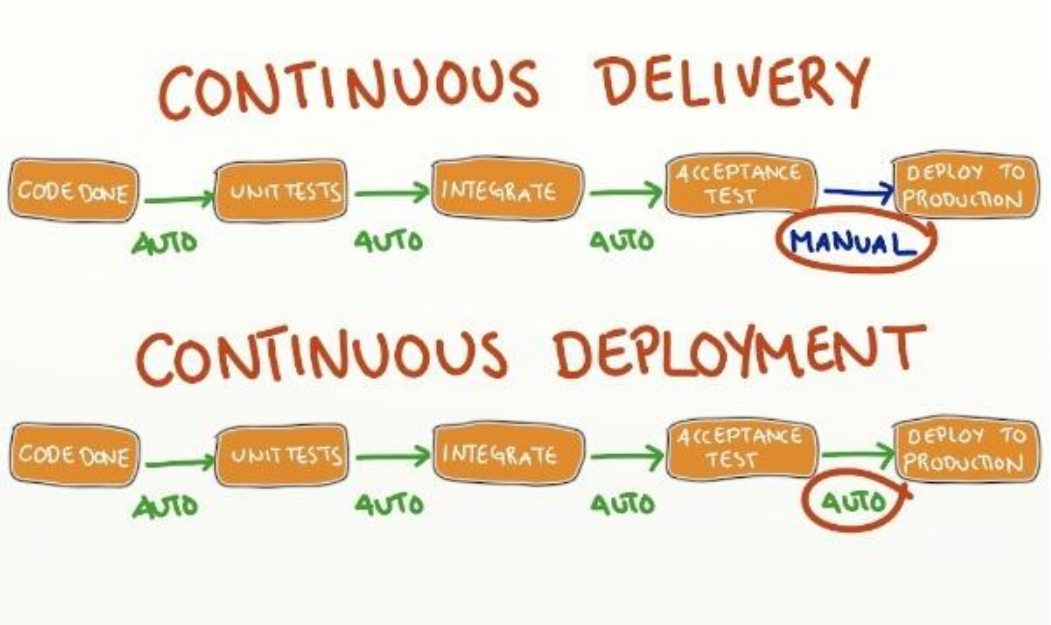
- Make the build self-testing and keep the builds fast
  - Self-testing code includes a suite of automated tests that can check a large part of the code base for bugs
  - Tests are kicked off from a simple command and result into overview, if any tests failed
  - Fast builds decisive for rapid feedback, otherwise significantly reduced turns on CI expected
  - Usually 2-3 staged pipeline (fast, basic tests in first stage allow quick response, sophisticated, longer tests afterwards)
- Daily commits to the baseline by everyone on the team
  - Integration is primarily about communication
    - CI allows developers to tell other developers about the changes they have made
    - Break work into smaller chunks

Principles of Continuous Integration

- Every commit (to the baseline) should be built
- Clone the production environment and test there
- Make it easy to get the latest deliverables
- Every team member can see the results of the latest build
- Automate build deployment
  - Maintain multiple environments, one to run commit tests, one or more to run secondary tests
  - Automated deployment moves executables between environments ⇒ Scripts/Systems necessary
  - Deployment into production requires automated rollback as well
  - Being able to automatically revert also reduces a lot of the tension of deployment, encouraging people to deploy more frequently and thus get new features out to users quickly

Continuous deployment

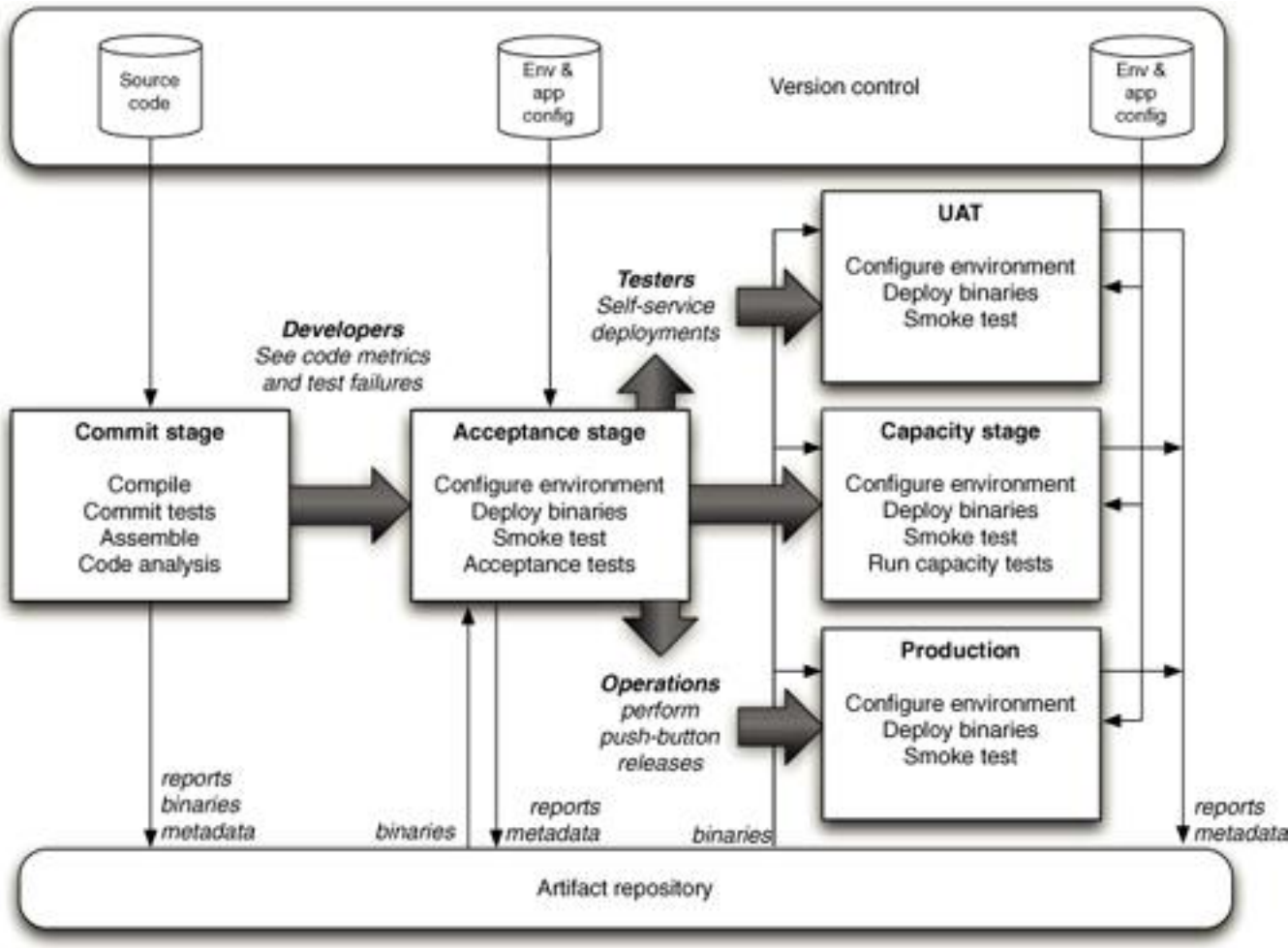
- Extension of continuous integration to minimize time elapsed between writing new line of code and this code being used by users, in production
- Prerequisites
  - Automated description, deployment, roll-back of the infrastructure and the code over different stages: development, testing, production
  - Instrumentation needed to ensure that any suggestion of lowered quality results in aborting the deployment process, or rolling back the new features, and triggers human intervention



codeship.com/continuous-integration-essentials

Deployment pipeline

- Push executables into increasingly production-like environments to ensure the software will work in production [1]



Basic principles

- Automated Deployment
  - Every deployment step is automated, no manual steps allowed
  - Continuous Deployment works because of automation
- Automated Rollback
  - Restore previous version if necessary => lowers fear of pushing regularly
  - Regular backups are necessary
- Deploy to Staging and use staging environment
  - Automatically push the master branch into a staging application including automated tests
  - Detect errors immediately that might affect the application
- Automatically deploy to production

Continuous Delivery

- Software can be released to production anytime
  - Software is deployable throughout its lifecycle
  - Developer team prioritizes keeping the software deployable over working on new features
  - Anybody can get fast, automated feedback on the production readiness of their systems anytime somebody makes a change
  - Push-button deployments of any version of the software to any environment on demand is possible
- Difference between Continuous Delivery and Continuous Deployment?
  - Similar terms coined around the same time
  - Difference as a business decision about frequency of deployment into production
    - Continuous Delivery is deployable anytime if needed
    - Continuous Deployment is actually deploying every change into production, every day or more frequently

Tools for Infrastructure as code

- Managing and provisioning computer data centers through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools
  - Applied to bare-metal and VMs
  - Descriptions stored in versioning systems
- Well known tools
  - Ansible using Python released in 2012: Red Hat's open source IT automation engine for cloud provisioning, configuration management, application deployment, intra-service orchestration, and other IT needs on multi-tier architectures
  - Chef using Ruby, Erlang released 2009: Automate infrastructure by managing servers in the cloud, on-premises, or in a hybrid environment.
  - Puppet using Ruby relased 2005: Open source server automation tool for configuration and management. It works on Linux, Unix, and Windows systems and performs administrative tasks (such as adding users, installing packages, and updating server configurations) based on a centralized specification

Tools for Continous Delivery

- Jenkins
  - Server-based system running in a servlet container such as Apache Tomcat
  - Supports many version control tools including CVS, Subversion, Git …
  - Executes Apache Ant, Maven and arbitrary shell scripts and Windows batch commands
  - Distributes builds and test loads on multiple machines
  - Open source CI tool written in Java
  - Cross-platform tool with feature extension through plugins
  - Well established in the community and very flexible
- Travis CI
  - Pioneer in this field of cloud Continuous Integration tools
  - Open source service free for all open source projects hosted on the GitHub => platform agnostic as hosted
  - It is configured using .travis.yml files and supports a variety of different languages
  - Travis uses the virtual machines to build applications
- Other tools: TeamCity, Go CD, Bamboo, GitLab CI, Circle CI, CodeShip, …

Tools for automating deployment

- Kubernetes
  - Open-source system for automating deployment, scaling and management of containerized applications across clusters of hosts
  - Supports a range of container tools such as Docker
  - Automatic binpacking: places containers based on their resource requirements and other constraints
  - Automatic scale-out based on CPU-usage
  - Storage orchestration: Automatically mount the storage system, either from local storage, a public cloud provider or a network storage system
  - Automated rollouts and rollbacks: rollback changes, restart containers and other activities for self-healing
  - Configuration management: Deploy and update application configuration without rebuilding the image
- Further tools extend current CI systems (Jenkins, Chef, Bamboo, Travis CI, …) or are currently developed (ElectricFlow, Go, UrbanCode Deploy, …)

References

[1] Jez Humble, David Farley: Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, Addison-Wesley Professional 2010