Application Deployment

Christoph Matthies (christoph.matthies@hpi.de)

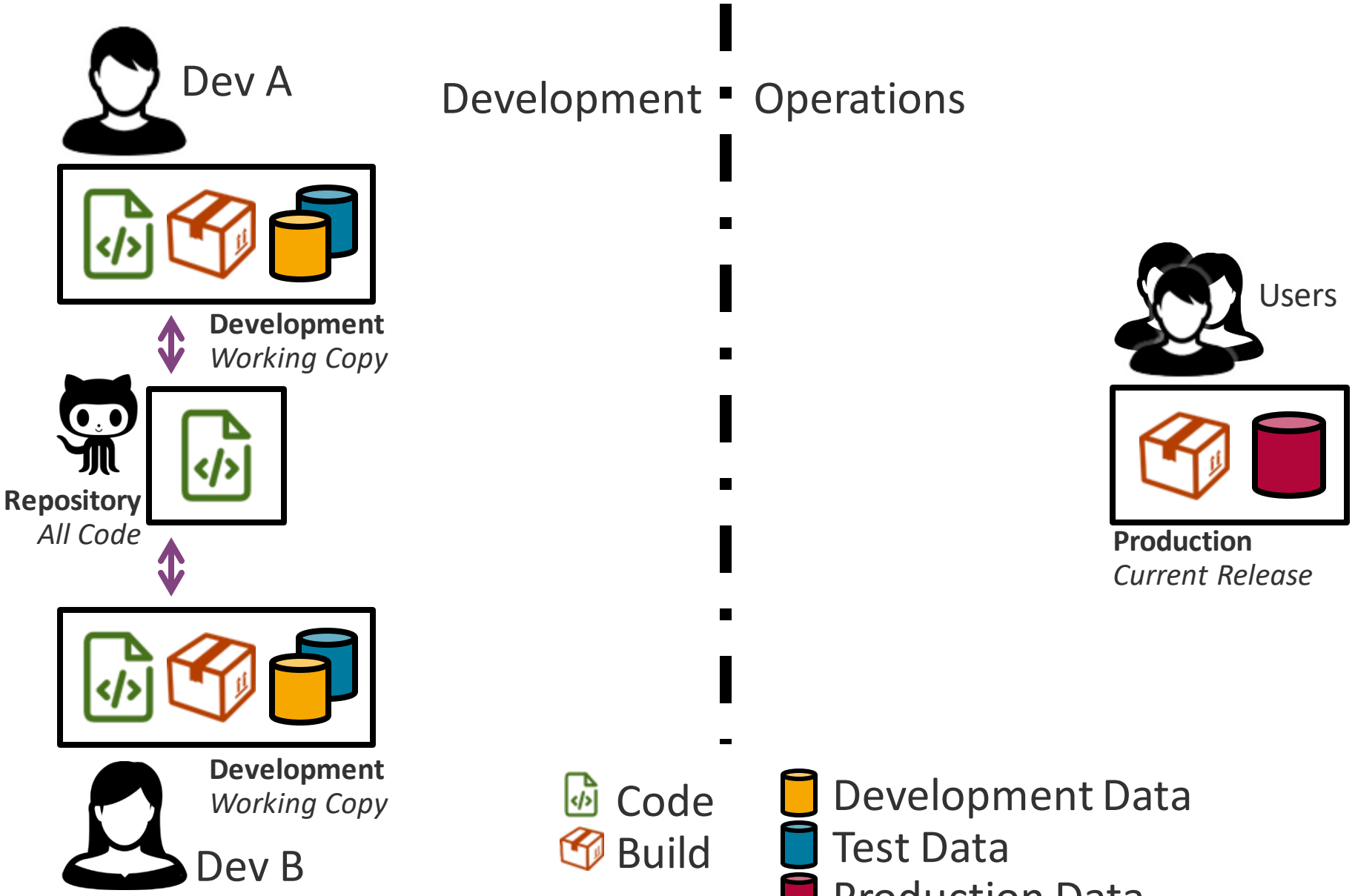
Software Engineering II
WS 2018/19

Enterprise Platform and Integration Concepts

Agenda

1. **DevOps**
2. Application Hosting Options
3. Automating Environment Setup
4. Deployment Scripting
5. Application Monitoring
6. Continuous Deployment and Scrum

Development vs. Operations



Development & Operations

Problems

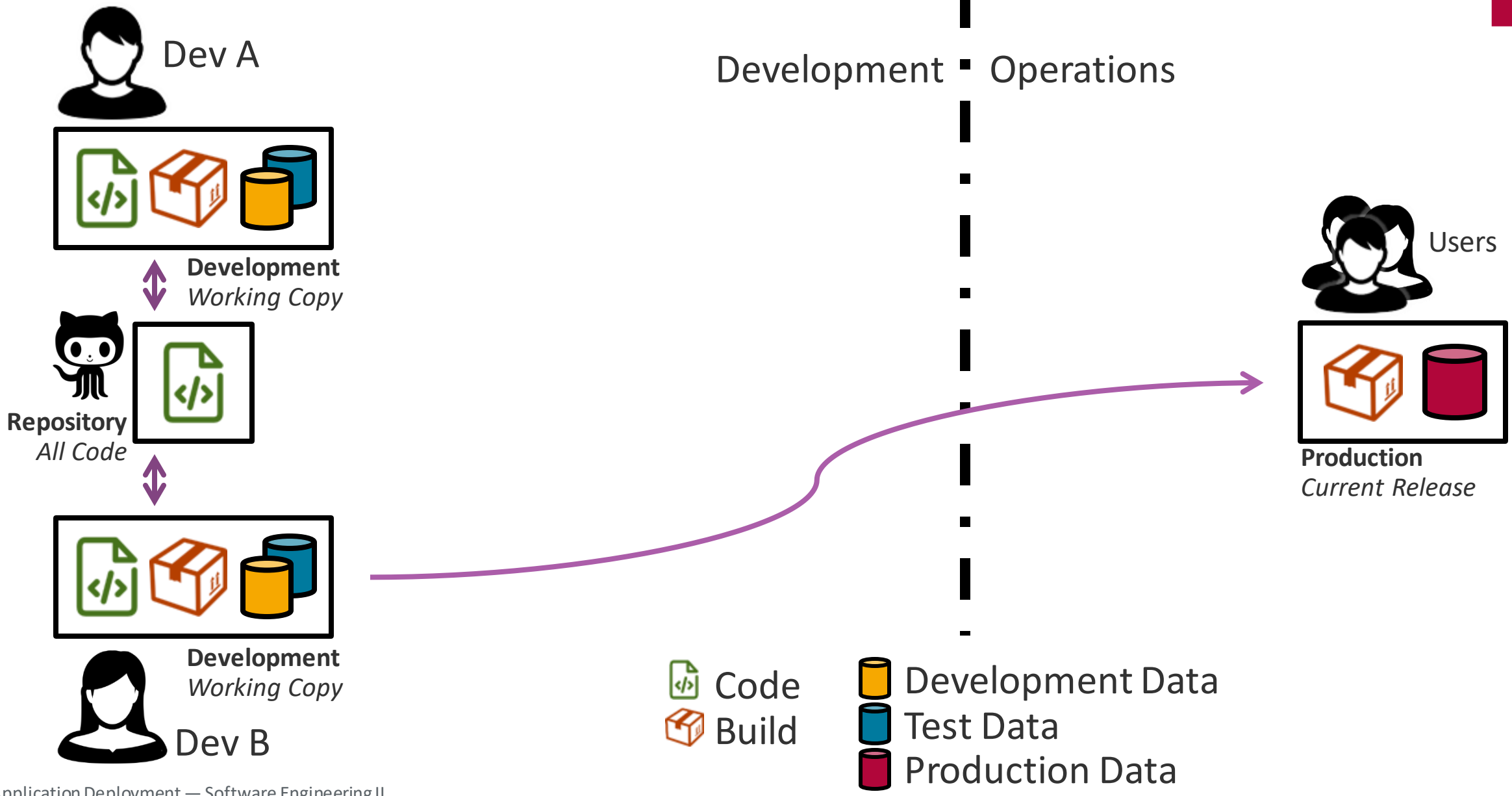
- Software needs to be operated
  - Developers vs. Admins
- **Short** deployment cycles
- Maintain **quality standards**

DevOps

- **Formalized** process for deployment
- Focus on communication, **collaboration**, and integration between Dev and Ops

"Agile for deployment"

Not DevOps


Dev A
Development
Working Copy
Repository
All Code
Development
Working Copy
Dev B
Development
Operations
Users
Production
Current Release
Code
Build
Development Data
Test Data
Production Data

Terminology

Release

- **Planned state** of the application
- Set of requirements
- Examples
  - Next big version with new shiny features
  - Urgent hotfix
  - Anything in-between

Version

- Could be anything
- A release has a **version number**

Terminology

Build

- Attempt to implement a release
  - **Snapshot** of application
- Often the output of the build tool
  - Not: the build script/tool/process
- Version number is
  “<Release Number>.<Build Number>”

Terminology

Environment

- A system on which the application can be deployed and used

To promote

- To deploy a build on the **next** environment

To release

- To promote a build to **production**
- Thereby finishing the release

Overview of Environments

Development

managed by developers

Development

- Where the developers work
- One per developer (if possible)

Integration

- Runs all tests
- A try-out version

Quality Assurance

- Professional manual testing

Operations

managed by admins

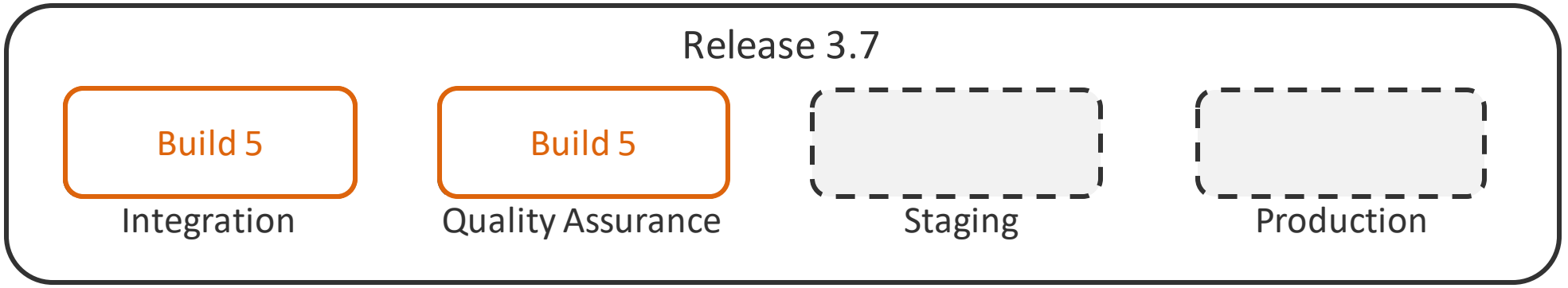
Staging

- Clone of production system
- Final rehearsal

Production

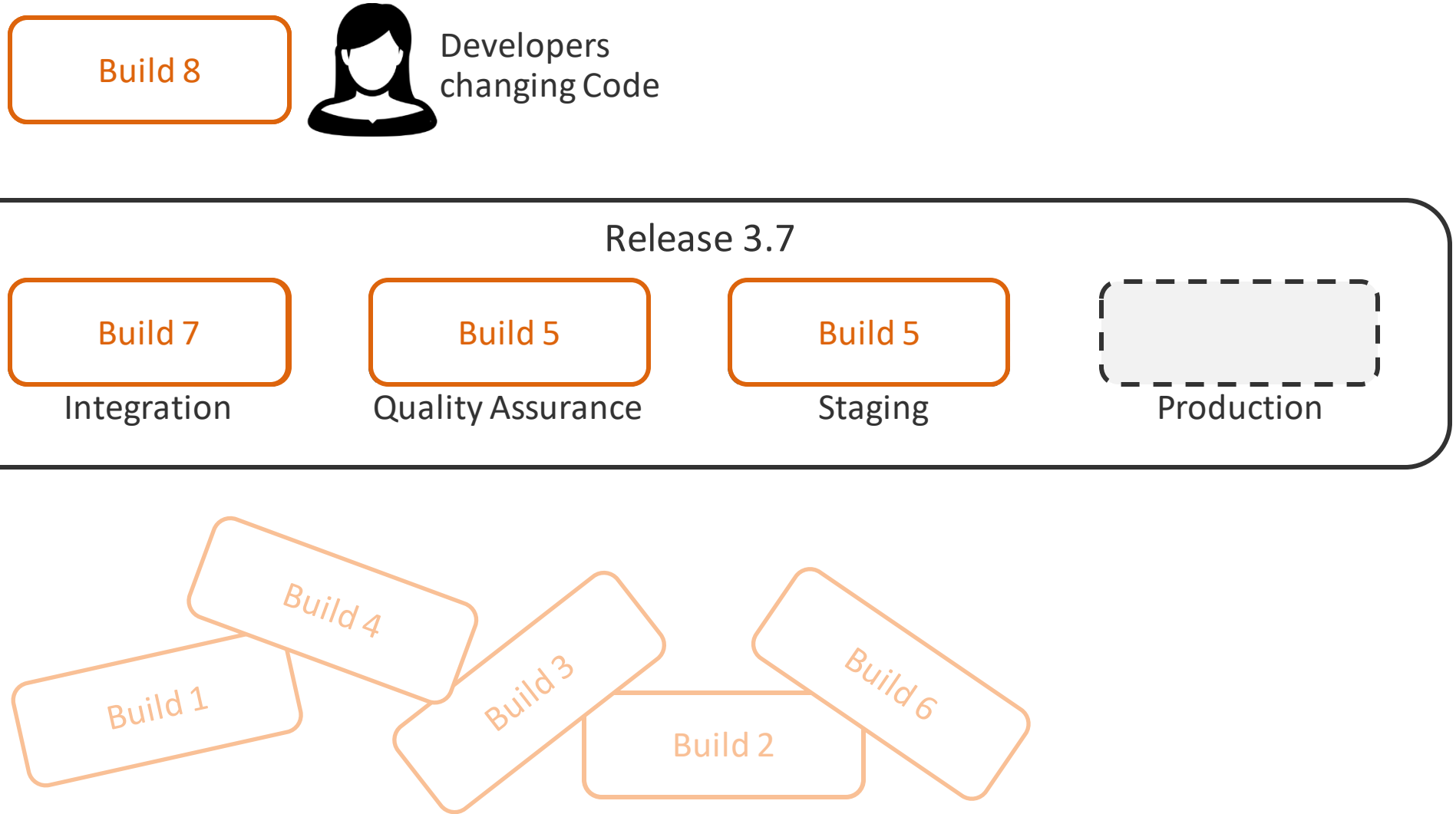
- The live system
- Failures are expensive here

Example

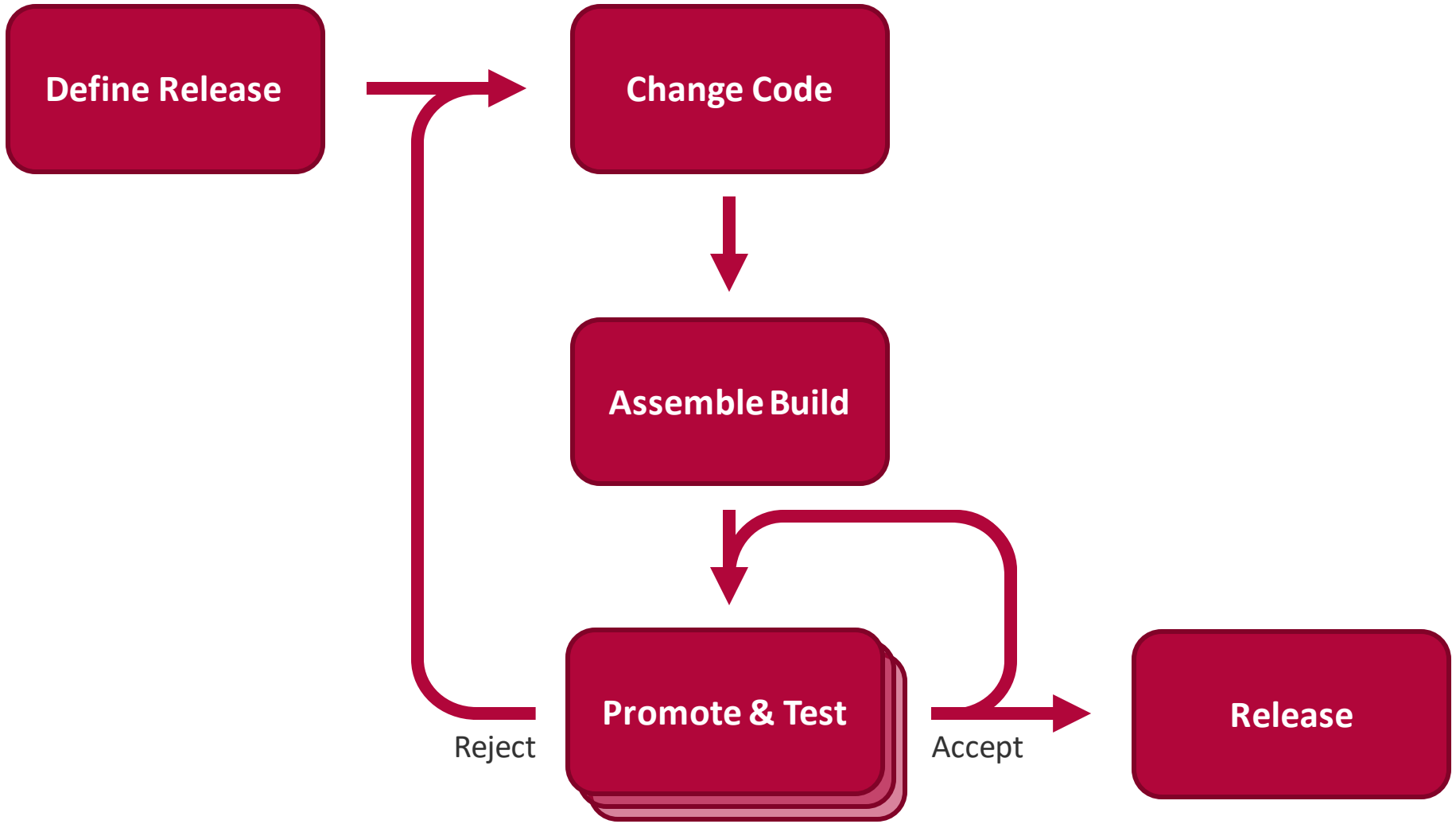

Release 3.7
Build 5
Integration
Build 5
Quality Assurance
Staging
Production

Build 4
Build 1
Build 3
Build 2

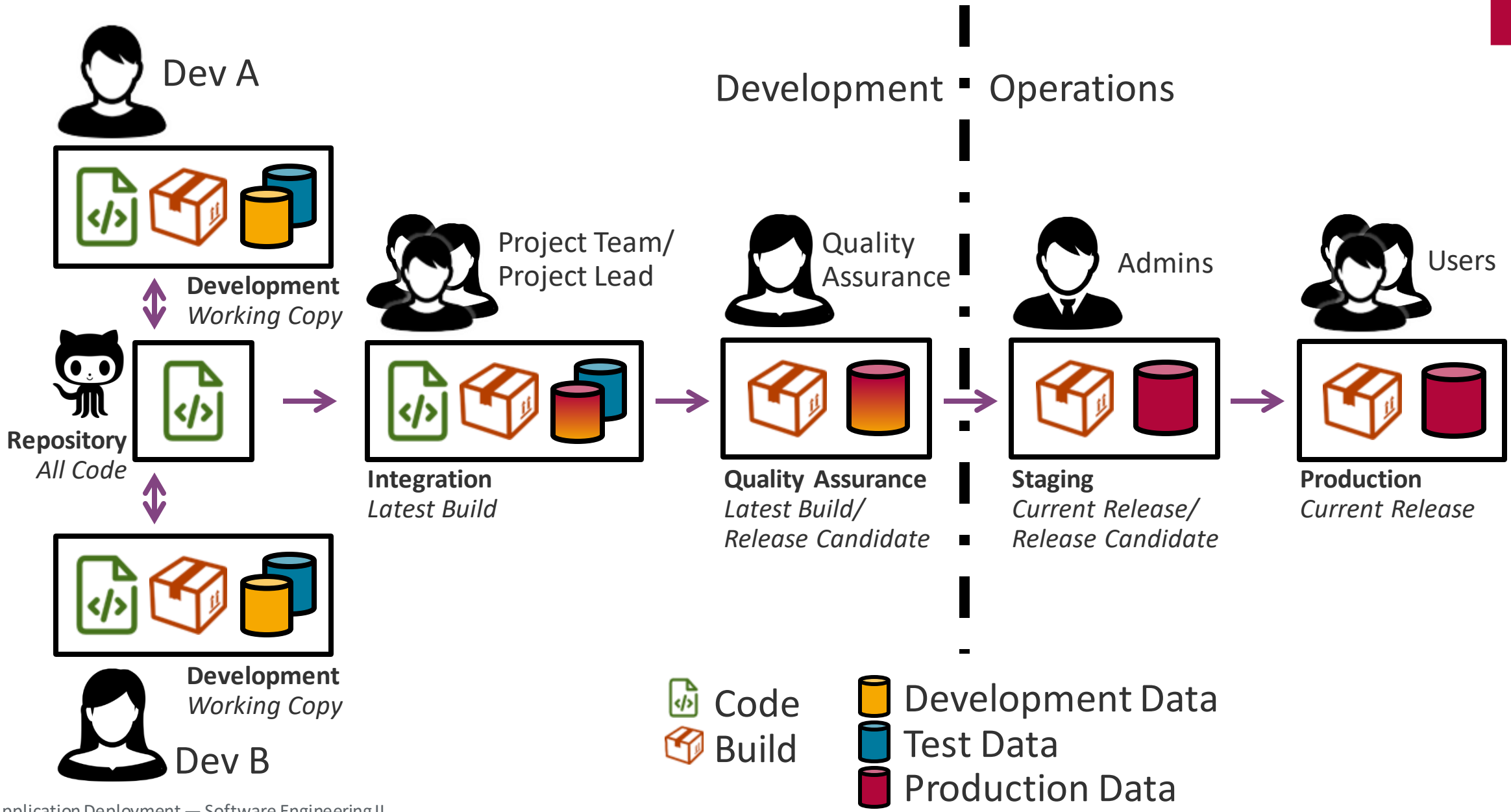
Example


Build 8
Developers changing Code
Release 3.7
Build 7
Integration
Build 5
Quality Assurance
Build 5
Staging
Production
Build 4
Build 1
Build 3
Build 2
Build 6

Workflow


Define Release
Change Code
Assemble Build
Promote & Test
Reject
Accept
Release

DevOps


Dev A
Development
Operations
Development
Working Copy
Project Team/
Project Lead
Quality
Assurance
Admins
Users
Repository
All Code
Integration
Latest Build
Quality Assurance
Latest Build/
Release Candidate
Staging
Current Release/
Release Candidate
Production
Current Release
Development
Working Copy
Dev B
Code
Build
Development Data
Test Data
Production Data

Implications

Builds are immutable

- If changed, previous testing was pointless

➔ Even the smallest change has to go through all environments

Many systems required

- Each environment has to be maintained
- Automation?

Deployment overhead

- Manual steps are potential for human failure
- Automation?

} Remainder of this lecture

Agenda

1. DevOps
2. **Application Hosting Options**
3. Automating Environment Setup
4. Deployment Scripting
5. Application Monitoring
6. Continuous Deployment and Scrum

Application Hosting Options

Choice of hosting options is driven by a variety of parameters

- Initial setup effort, cost, and required expertise
- Operational costs and effort
- Targeted service level agreements (SLAs)
- Legal considerations (data privacy, liability, etc.)

Platform as a Service (Paas)

Providers deliver OS, execution environment, database, web server, monitoring, etc.

Advantages

- Minimal effort and knowledge required for setup
- Only platform development knowledge (e.g. Python, Ruby) needed, no need for hardware / OS maintenance
- Possibility to scale up quickly and easily

Disadvantages

- Usually fixed environment with little variation points
- Provider SLA targets might differ from yours, e.g. downtime, response times
- Limited technical support

Examples: Heroku, Azure Compute, Google App Engine

Infrastructure as a Service (IaaS)

Providers deliver virtual private servers (VPS) with requested configuration
Setup of execution environment, database servers, etc. is up to customers

Advantages

- Flexibility regarding execution environment
- Avoid management of underlying hardware
- Dynamic on-demand scaling of resources

Disadvantages

- Server administration know-how and efforts required
- It's still a VM: Potential performance drops, Disk I/O, etc.

Examples: Amazon EC2, Google Compute Engine, Rackspace Cloud, DigitalOcean

Dedicated Hosting

Providers allocate *dedicated* hardware, classical approach

Advantages

- Complete control over server, down to bare metal, full power always available
- No virtualization-related performance issues
- More control over network configuration
- Dedicated SLAs

Disadvantages (compared to Iaas)

- No easy scaling of resources
- Administration efforts for servers, e.g. monitor disk failures

Examples: Hetzner, OVH, Rackspace, Host Europe

Own datacenter

You host your own servers

Advantages

- Complete control over data, security, operations, network etc.
- Custom designed servers possible
- Add cabinets in available space with low cost

Disadvantages

- Huge upfront costs, e.g. space, cooling, fiber, hardware
- Expanding the space of the datacenter is expensive
- Provide around the clock support, monitoring, personnel, etc.
- Not feasible for small companies

Examples: Google, Facebook

Agenda

1. DevOps
2. Application Hosting Options
3. Automating Environment Setup
   - **Virtualization**
   - Provisioning
4. Deployment Scripting
5. Application Monitoring
6. Continuous Deployment and Scrum

Setting up an Environment

Main challenges in preparing infrastructure:

- Minimize the effort required to repeatedly setup identical execution environments
- Without relying on “administration gurus”

Solutions:

- *DevOps*, i.e. a strong collaboration between the development and the operations team
- A strong bias towards automation

Where to Start With "Deploying"?

- Hosted solutions aren't always feasible for initial experiments
- Maintaining local installs of server stacks
  in different versions can get cumbersome
- Development vs. production environment differences
  result in *"it works on my machine"* problems
- Don't want to force all developers to use
  same development environment (e.g. choice of OS)

Possible solution: VirtualBox + Vagrant (https://www.vagrantup.com/)

- "Deploy" to a virtual machine on your local OS for development

http://code.tutsplus.com/tutorials/vagrant-what-why-and-how--net-26500

Agenda

1. DevOps
2. Application Hosting Options
3. Automating Environment Setup
   - Virtualization
   - **Provisioning**
4. Deployment Scripting
5. Application Monitoring
6. Continuous Deployment and Scrum

Next Step: Automate VM Configuration

Virtualization software provides a VM.

Provisioning tools configure it, e.g. install required software.

Why not provision manually?

- Error prone, repetitive tasks
- Documentation has to be kept up-to-date
- Explicit knowledge transfer required if Admin changes

One provisioning tool example: Chef (http://chef.io, https://github.com/chef/chef)

- Formalize software install and configuration state into *recipes*
- Recipes are shared (https://supermarket.chef.io/cookbooks)
- Ensure software is installed based on dependencies
- Ensure that files, packages, and services are in the prescribed state

Common alternative: Puppet (https://puppetlabs.com/)

Provisioning Summary

Create your VM, e.g. describe it with Vagrant.

Using provisioning tools, you can:

- Define the required packages for all required servers
- Install and configure necessary services
- Create the directory structure for your application
- Create custom configuration files (e.g., database.yml)

Not touched here but also possible:

- Use templates to create different files based on variables
- Environments (staging vs. production)
- Central management of configuration files that are automatically transferred to clients

Agenda

1. DevOps
2. Application Hosting Options
3. Automating Environment Setup
4. **Deployment Scripting**
5. Application Monitoring
6. Continuous Deployment and Scrum

Environment is set – How to deploy?

Necessary steps after the server is configured:

- Checkout code changes
- Update your dependencies (i.e. gems)
- Run database migrations, restart application servers
- Optional: Restart index servers, setup new Cron jobs, etc.

Remember: Automation!

- Easiest: **Travis CI** supports deploying to many hosting providers (http://docs.travis-ci.com/user/deployment/)
  - Deploy after all the tests pass
- Alternative: Capistrano (https://github.com/capistrano/capistrano)
  - Prepares the server for deployment (possibly using provisioning tools)
  - Deploy the application as updates are made

Deployment with Travis CI

Travis Continuous Integration and Deployment Workflow:

tests are run

1. `before_install`
2. `install`
3. `before_script`
4. `script`
5. `after_success` or `after_failure`
6. `after_script`

optional steps

7. `before_deploy`
8. `deploy`
9. `after_deploy`

A non-zero exit-status is these phases means the build is marked as *failed.* The build is *not* deployed to the hosting provider.

Otherwise it is deployed in the deploy step.

- A custom `after_success` step can be used to deploy to own servers (http://docs.travis-ci.com/user/deployment/custom/)

http://docs.travis-ci.com/user/build-lifecycle/

Agenda

1. DevOps
2. Application Hosting Options
3. Automating Environment Setup
4. Deployment Scripting
5. **Application Monitoring**
6. Continuous Deployment and Scrum

Monitoring your servers and application

Keep an eye on server health and applications:

- Get alerts when components fail or exceed predefined thresholds
- Examples:
  - Uptime Robot—HTTP GET / ping every 5 mins (https://uptimerobot.com/)
  - Nagios—Monitor infrastructure, down to switches and services (http://nagios.org)

Monitor application errors and performance bottlenecks:

- Monitor errors that happen at runtime, discovered by users
- Notifications upon application errors, slow downs
- Good idea: Protocols for error fixing!
- Examples:
  - Errbit—Collect and organize errors (https://github.com/errbit/errbit)
  - New Relic—Performance monitoring, response times, SQL (http://newrelic.com/)

Agenda

1. DevOps
2. Application Hosting Options
3. Automating Environment Setup
4. Deployment Scripting
5. Application Monitoring
6. **Continuous Deployment and Scrum**

Deploying 50 times a day? Continuous Delivery

Advantages:

- Users get a sense of “something happening” frequently, shorter feedback loop
- Business value of features immediately present
- Deploy scripts used often, less likely to contain errors
- Reduced amount of code changes per release → faster fixes, less downtime

Prerequisites/Disadvantages:

- Only feasible with extensive set of *good* tests
- Tests / deployment need to run fast *(Continuous Integration)*
- Additional training for developers *(DevOps)* required
- May not be feasible for applications that require planning or long-term support (e.g. operating systems)

Discussion:

Operating systems feature both CD (rolling releases) and classical approaches (LTS releases)

Continuous Deployment vs. Scrum

How do 50 deployments a day fit into Scrums notion of Sprints?

Some ideas (let's discuss):

- Intermediate Reviews for individual stories by the PO
  - At sprint review, each finished story is already running in production
  - Review meetings become shorter, more of a high level overview
- Get faster feedback from stakeholders for next Scrum meeting
- Deploying to staging or testing systems becomes part of the definition of done
- Acceptance of features not only based on PO approval but stakeholder approval?
  - A/B testing?
- "Working software is the primary measure of progress" —*Agile Manifesto*
  - Is software that is not deployed *working*? *(DevOps)*

Summary

1. DevOps
2. Application Hosting Options
3. Automating Environment Setup
4. Deployment Scripting
5. Application Monitoring
6. Continuous Deployment and Scrum

Conclusion: **Automate everything**!

Image Credits

- thenounproject.com
  - Box designed by Mourad Mokrane
  - Bricks designed by Trammie Anderson