



**Hasso
Plattner
Institut**

IT Systems Engineering | Universität Potsdam

Application Deployment

Softwaretechnik II 2012/13

Thomas Kowark

Outline

2

- Options for Application Hosting
- Automating Environment Setup
- Deployment Scripting
- Application Monitoring
- Continuous Deployment and Scrum

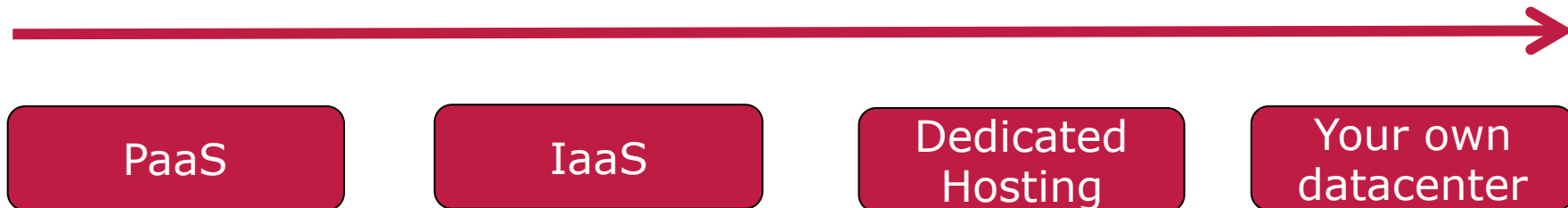
Hosting Options

3

- 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.)

Low Effort
Little Control

High Effort
High Control



Platform as a Service (Paas)

4

- Providers deliver Operating System, Execution environment, Database, Web Server, Monitoring, etc.
- Advantages
 - Minimal effort and knowledge required for setup (see Heroku Doku, for example)
 - Possibility to scale-up easily
- Disadvantages
 - Usually fixed environment with little variation points
 - Provider SLA targets might differ from yours (Downtime, Response Times, etc.)
 - Limited Technical support
- Examples: Heroku, Force.com, Azure Compute, Google App Engine, (EngineYard)

Infrastructure as a Service

5

- Providers deliver virtual private servers with requested configuration
- Setup of execution environment, database servers, etc. is up to customers
- Advantages
 - Flexibility w.r.t. execution environment
 - Control over VM parameters
- Disadvantages
 - 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, (EngineYard)

Dedicated Hosting

6

- Providers allocate dedicated hardware
- Setup similar to IaaS
- Advantages
 - No virtualization-related performance issues
 - More control over network configuration (e.g. racking machines up as needed)
 - Dedicated SLAs
- Disadvantages
 - High upfront cost
 - Administration efforts
- Examples: Hetzner, GoDaddy, Rackspace, Host Europe

Setting up the Production Environment

7

Scenario: Mixture of IaaS and Dedicated Hosting

- For Heroku Deployment, please refer to the Heroku documentation
- Own infrastructure is out of scope

Step 1: Preparing the infrastructure

- Main Challenges:
 - How to minimize the efforts required to repeatedly setup identical execution environments for your application?
 - Without relying on “administration gurus”?
- Solutions:
 - DevOps, i.e., a strong collaboration between the development and the operations team
 - A strong bias towards automations

Where to start?

8

- Dedicated Servers and VPS not always feasible for initial experiments
- Possible solution: Virtual Box + Vagrant

Vagrant (<http://www.vagrantup.com>)

- DSL for describing the basic parameters of a virtual machine
- Allows for simple recovery in case of VM errors
- Predefined and custom packaged boxes
- Possibility to create a multi-server setup
- Advantages:
 - File size reduced in compared to sharing suspended VMs
 - Same packages loaded with custom VM configurations

Vagrant in a nutshell

9

- **vagrant init lucid64 && vagrant up**
- vagrant ssh + your desired changes
- **vagrant package**
- **vagrant box add your_new_base_box_name package.box**

- Sample Vagrant File:

```
Vagrant::Config.run do |config|
  config.vm.customize ["modifyvm", :id, "--name", "app", "--memory", "512"]
  config.vm.box = "lucid64_with_ruby193"
  config.vm.host_name = "app"
  config.vm.forward_port 22, 2222, :auto => true
  config.vm.forward_port 80, 4567
  config.vm.network :hostonly, "33.33.13.37"
  config.vm.share_folder "hosttmp", "/hosttmp", "/tmp"
end
```

Next Step: Automate VM Configuration

10

- VM is up and running -> How to configure it automatically?
- Why not manually?
 - Error prone, repetitive tasks
 - Documentation has to be kept up-to-date
 - Explicit knowledge transfer required if Admin changes
- One sample solution: Puppet (<http://puppetlabs.com>)
 - Formalize server configuration into *manifests*
 - Ensure that files, packages, and services are in the prescribed state
 - Requires administration knowledge, i.e., services that are not specified will not start automatically
- Alternative: Chef (<http://wiki.opscode.com/display/chef/Home>)

Example: Install, Configure, and run Apache2 with Puppet

11

```
puppetrails/apache_package_file_service/modules/apache2/manifests/init.pp
```

```
class apache2 {
  package {
    "apache2":
      ensure => present,
      before => File["/etc/apache2/apache2.conf"]
  }

  file {
    "/etc/apache2/apache2.conf":
      owner   => root,
      group  => root,
      mode   => 644,
      source => "puppet:///modules/apache2/apache2.conf"
  }

  service {
    "apache2":
      ensure   => true,
      enable  => true,
      subscribe => File["/etc/apache2/apache2.conf"]
  }
}
```

```
$ sudo puppet apply --verbose manifests/site.pp
```

Tying the pieces together

12

- Describe your virtual machine with Vagrant
- With Puppet, you can
 - Define the required packages for all required servers
 - Install and configure necessary services
 - Create the directory structure for your application
 - Create configuration files (e.g., database.yml)
- Not touched here but also possible
 - Use templates to create different files based on variables
 - Control flow features (if-else and switch)
 - Environments (staging vs. production)
 - PuppetMaster (Central management of manifests that are automatically transferred to connected PuppetClients)
 - PuppetDashboard

Environment is set – How to deploy?

13

- Necessary steps:
 - Checkout code changes
 - Update your bundle
 - Database migrations
 - Restart application servers
 - Optional: Restart index servers, setup new Cron Jobs, etc.

- Remember: Automation!
 - Simple version: see swt2-ci
 - Capistrano (<https://github.com/capistrano/capistrano>)
 - ◇ Prepares the server for deployment
 - ◇ Deploy the application as updates are made

Capistrano

14

- Capistrano executes *tasks* in a Unix shell via ssh
- Once again: DSL to describe what needs to be done
- Setup: `$ capify .`

`capistrano/config/deploy.rb`

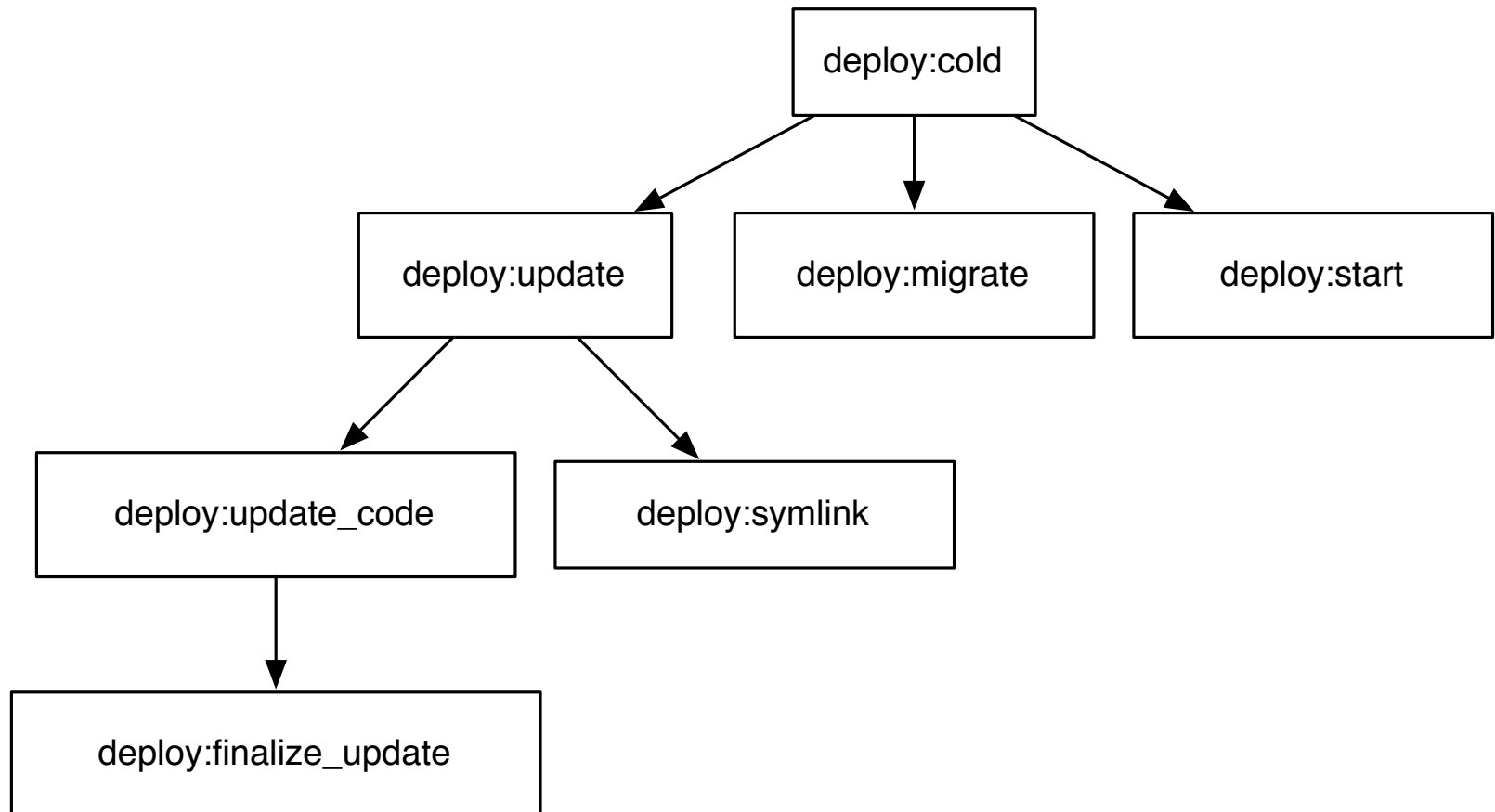
```
namespace :deploy do
  task :start do ; end
  task :stop do ; end
  desc "Restart the application"
  task :restart, :roles => :app, :except => { :no_release => true } do
    run "#{try_sudo} touch #{File.join(current_path, 'tmp', 'restart.txt')}"
  end
  desc "Copy the database.yml file into the latest release"
  task :copy_in_database_yaml do
    run "cp #{shared_path}/config/database.yml #{latest_release}/config/"
  end
end
before "deploy:assets:precompile", "deploy:copy_in_database_yaml"
```

- Create the Virtual Machine from the predefined box
 - -> correct operating system, Ruby installed, Puppet installed
- Apply the puppet manifests
 - -> all required packages loaded, services running, directory structure for the app created (e.g. /var/my_app/)
- Run cap deploy:setup
 - Directory structure for deployment
 - ◇ /releases
 - ◇ /shared
 - /log
 - /system
 - /pids

Deploying with Capistrano

16

- `$ cap deploy(:cold)`



Extended Capistrano Features (1/2)

17

- Hooks
- File Up/Download (e.g., retrieve log files)

capistrano2/download.rb

```
desc "Download the production log file"
task :download_log do
  download "#{current_path}/log/production.log", \
    "$CAPISTRANO:HOST$.production.log"
end
```

- Multistage deployment

- Larger projects might have multiple environments, e.g., for quality assurance, performance testing, etc.
- By setting multiple stages, we can reuse general commands and only alter what's needed in particular environments

```
set :stages, %w(beta production)
set :default_stage, "beta"
require 'capistrano/ext/multistage'
```

Extended Capistrano Features (2/2)

18

- Capture output from remote servers (e.g., `free -m | grep Mem`)
- Capture streams from remote servers (e.g., `tail on production.log`)
- Using `$ cap shell` to run commands simultaneously on multiple servers (e.g., `df -h`)

Should we really do these manually?

Monitoring your servers and application

19

- Keep an eye on server health and applications:
 - Get alerts when infrastructure components fail or exceed predefined thresholds
 - Examples:
 - ◇ Nagios (<http://nagios.org>)
 - ◇ newrelic (<http://newrelic.com>)

- Monitor application errors and performance bottlenecks
 - Breakdowns for long-running requests
 - Notifications upon application errors
 - Good idea: Protocols for error fixing!
 - Examples: airbrake (<http://airbrake.io>, open-source self-hosted alternativ: errbit), newrelic

Deploying 50 times a day? Continuous Deployment

20

- Advantages:
 - Users get a sense of “soemthing happening” frequently
 - Features are available on the spot
 - Error isolation -> reduced downtime for error detection

- Prerequisites/Disadvantages
 - Only feasible with extensive set of GOOD tests (see Chapter 3)
 - Testing needs to be fast and continuous
 - Deployment effort should be minimal (Capistrano, anybody?) and take reasonable amounts of time
 - Not feasible for applications with high availability requirements

Continuous Deployment vs. Scrum

21

- How do 50 deployments a day fit into Scrum's notion of Sprints?
- Some ideas (let's discuss):
 - Intermediate Reviews for individual features by the PO
 - Deploying to staging or testing systems becomes part of the definition of done
 - Acceptance of features not only based on PO approval but user approval?
 - ...

Outlook: Next Week

22

- 11am Slot for Co-located sprint planning
 - First: Short introduction by POs for all teams (V-2.16)
 - Afterwards: Individual Plannings for all teams.
 - All POs are constantly available via telephone/Skype/Email to answer upcoming questions (“Who’s doing that”, “How far are they”, etc.) on the spot
- 1:30pm Slot: Guest lecture by Mason Womack (Immobilienscout24)

Announcement: Oral Exam

23

- The oral exam will take place on Tuesday, Feb 19
- Topics:
 - Everything that is on the slides and was said within the lecture
 - You should be able to reflect on your project work
- Timeframe:
 - Tbd, but approximately 1h for 3 people
 - Slots also tbd in the next weeks
- Q&A session:
 - Feb 8 after the final presentation and retrospective
 - Via email on-demand