

The background features a collage of various icons related to web development and technology. These include a hand cursor pointing at a screen, a globe, a target, a magnifying glass, a notepad, a smartphone, a heart, a keyboard, and a document with a greater-than sign. The icons are rendered in a light gray, semi-transparent style, creating a layered effect.

Web-Entwicklung 2

Vorlesung 6

Fachbereich Wirtschaft - Fachhochschule Münster
Bachelor of Science Wirtschaftsinformatik Wintersemester 2013/2014



Kapitel 2: Rails In Action

MScIS Peter Grosskopf
MScIS Thomas Hollstegge

Fachbereich Wirtschaft - Fachhochschule Münster
Bachelor of Science Wirtschaftsinformatik Wintersemester 2013/2014

Agenda

- Vorstellung Leeze.ms im Detail
 - Vorstellung MVP
 - Wireframes
 - Weitere Planungsaspekte (Technologien, Deployment, Testing)
- Bootstrapping
- Umsetzung User Stories

Projektvorbereitung

Zielbestimmung

Vision:

Entwicklung der ersten Portalseite für den
Fahrradtourismus im Münsterland.

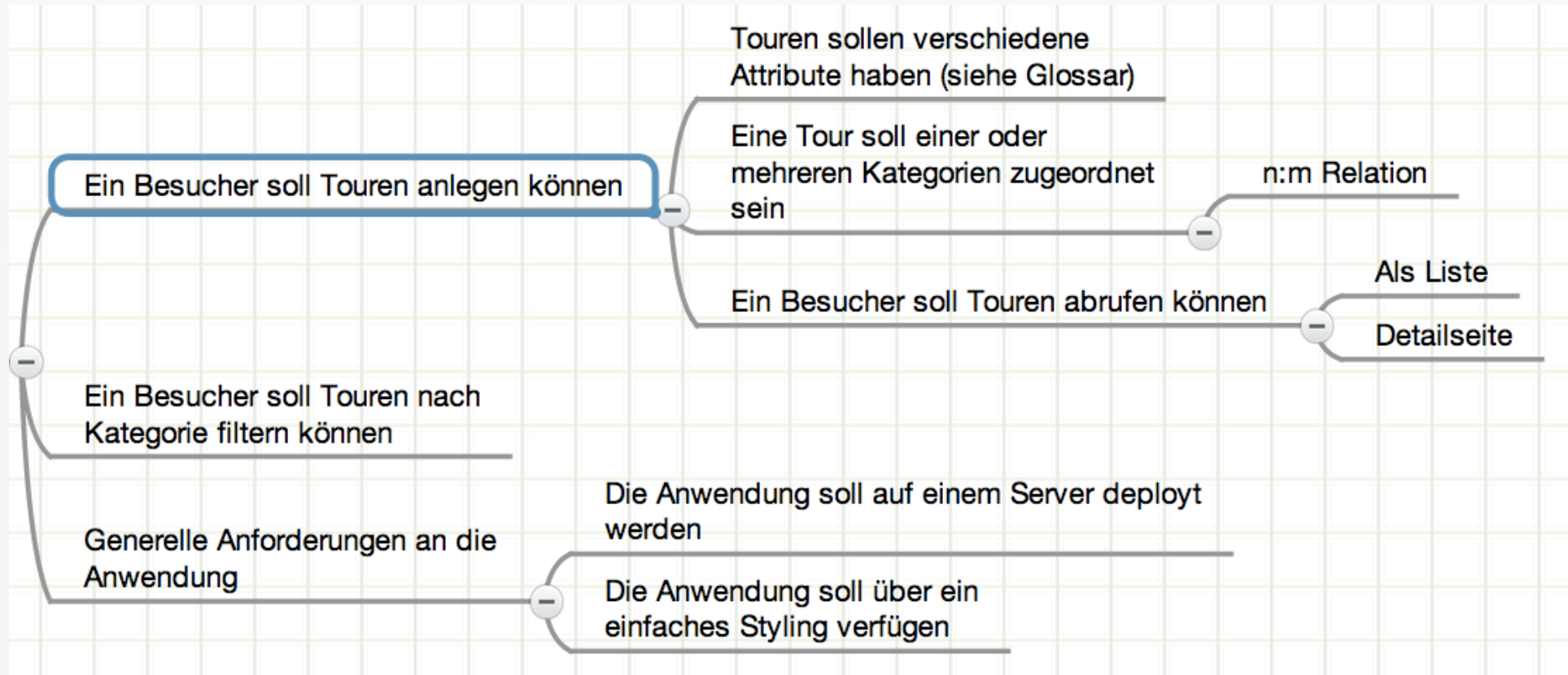
USPs:

Gute Bedienbarkeit

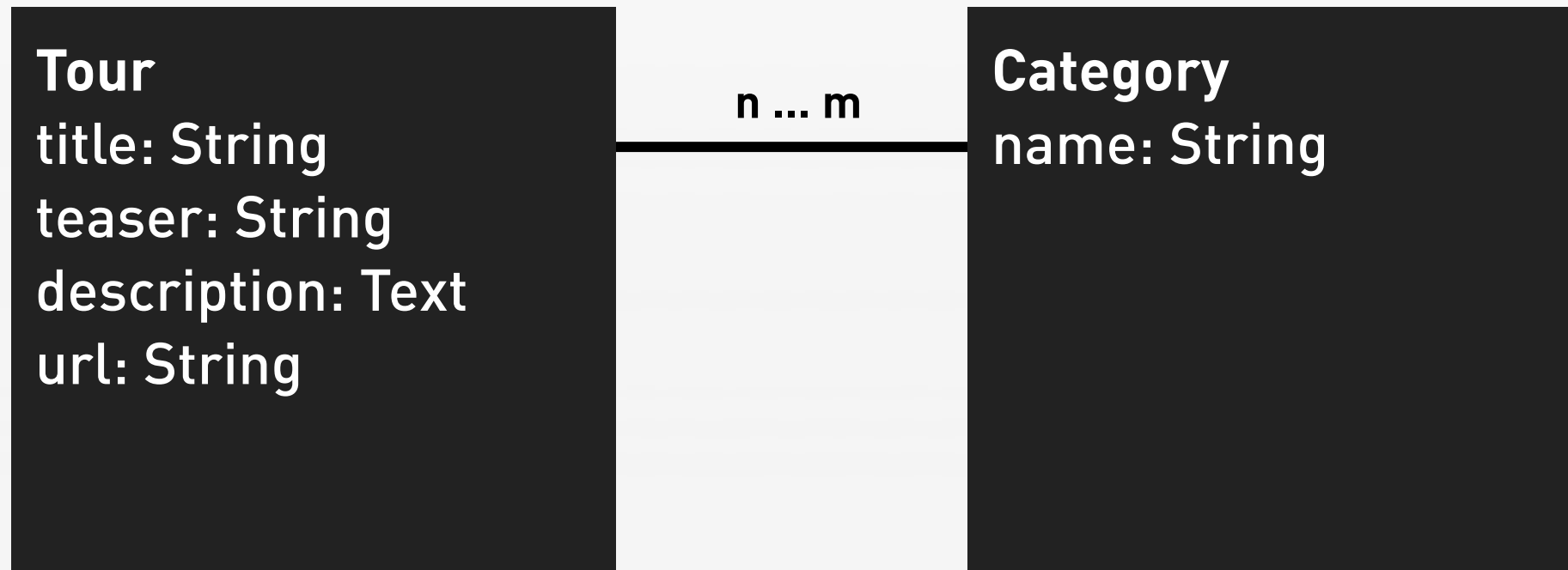
Nutzerfreundlichkeit

Einsatz von fortgeschrittenen Kartentechnologien

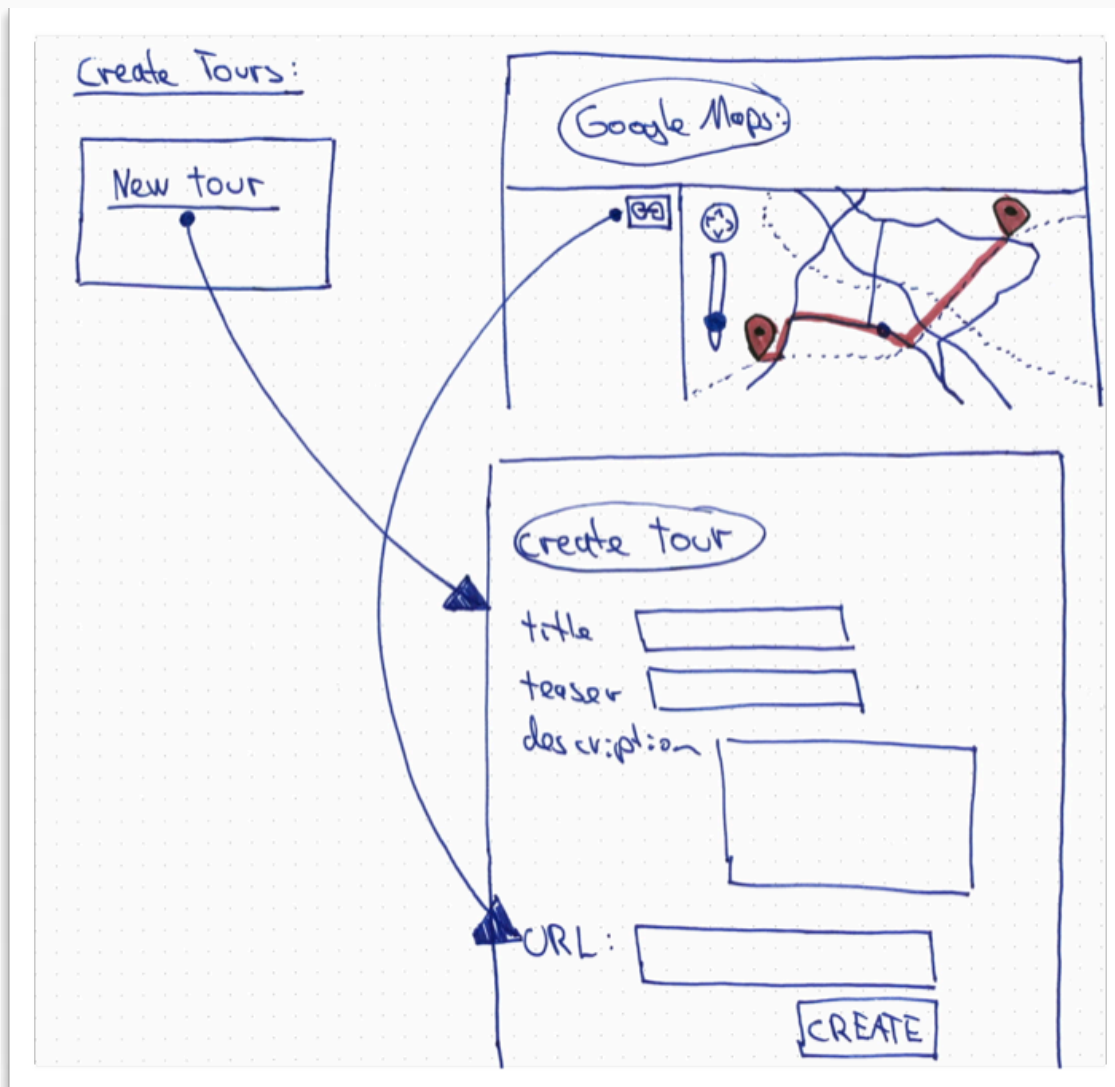
Minimum Viable Product



Datenbankschema MVP



Wireframes MVP



Wireframes MVP

Add categories to tour:

create / edit tour:

...

URL:

category:

Tour Detail

URL

ADS

Description

Weitere Planungsparameter

- Technologien:
 - Ruby on Rails
 - Rspec - Testing
 - SQLITE
- Deployment:
 - In der Cloud auf <http://www.heroku.com/>

Bootstrapping

- Gems auf den aktuellsten Stand bringen:
 - `gem update`
 - `gem install rails`
- Anwendung anlegen:
 - `rails new leeze.ms`
 - `cd leeze.ms`

Zusätzliche Gems definieren

```

source 'https://rubygems.org'
...
group :development, :test do
  gem 'rspec-rails'
  gem 'factory_girl_rails'
  gem 'guard-rspec'
  gem 'faker'
end

```

Rspec für Testing einbinden

Generatoren anpassen

```
config/application.rb

config.generators do |g|
  g.fixture_replacement :factory_girl
  g.view_specs false
  g.controller_specs false
  g.helper_specs false
end
```

Bootstrapping (contd.)

- Leeze.ms spezifische Gems installieren:
 - bundle install
- Rspec in Projekt verfügbar machen:
 - rails generate rspec:install

Taskboard



User Story 1

Ein Besucher soll Touren
anlegen können, um
empfehlenswerte Strecken mit
anderen Nutzern teilen zu
können.

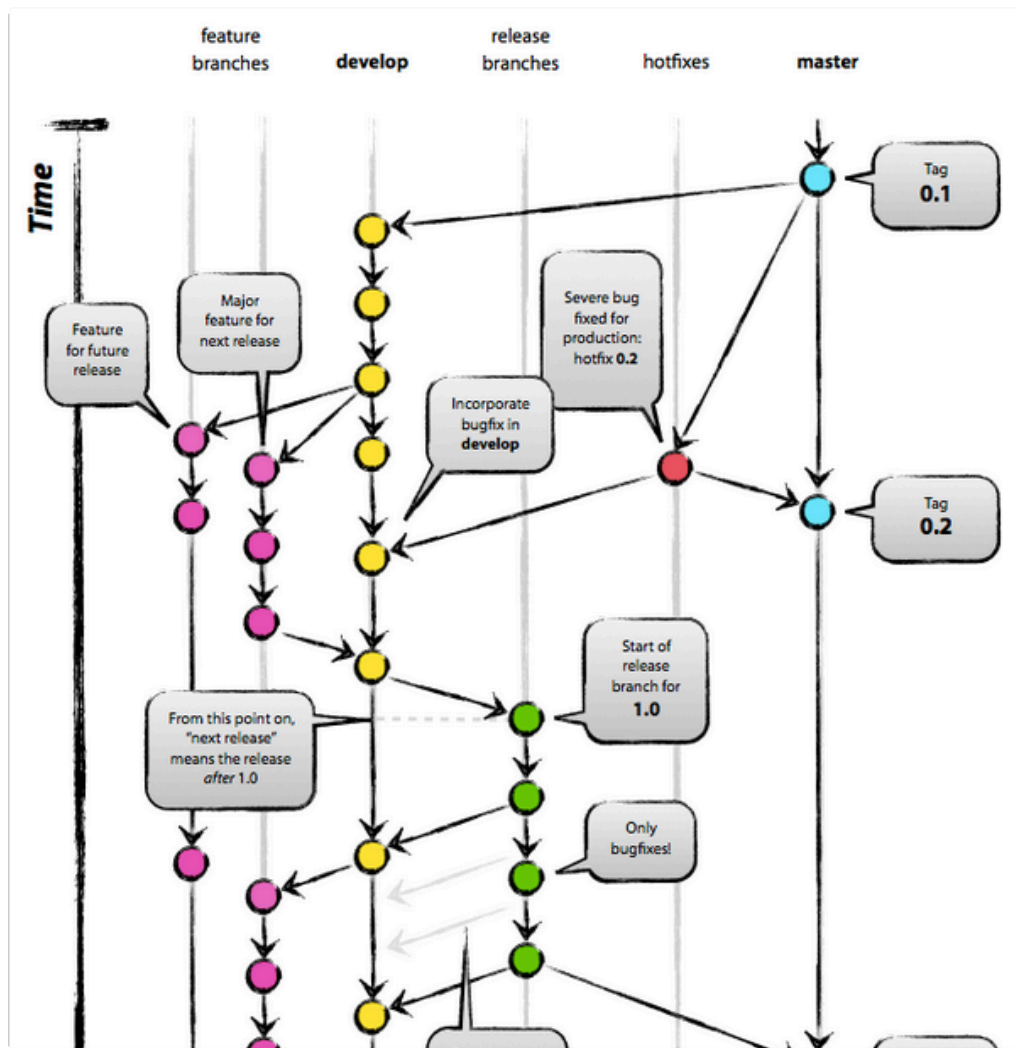
Git Workflow

Ziel:

Konflikte bei der Entwicklung von Features vermeiden
Unabhängigkeit von Features sicherstellen

- Nutzung von Branches:
 - master (aktuelle stabile Version)
 - develop (Entwicklungsversion mit fertigen Features)
 - Zusätzlich: Feature-Branches

Git Workflow



- **Basic-Workflow:**
 - Feature-Branch erzeugen
 - Feature entwickeln
 - Feature-Branch in Develop-Branch mergen
 - Wenn stabil, dann Develop in Master mergen
- Siehe: <http://nvie.com/posts/a-successful-git-branching-model/>

Git Workflow

- Anlegen des Develop-Branches:
 - `git checkout -b develop`
- Anlegen eines Feature-Branches:
 - z.B. `git checkout -b 1_create_tours`

User Story 1

- CRUD für Touren anlegen:
 - rails generate scaffold Tour title:string teaser:string description:text url:string
 - rake db:migrate
- Validation hinzufügen:

```
app/models/tour.rb

class Tour < ActiveRecord::Base
  validates :title, :teaser, :description, :url, presence: true
  validates :url, format: %r!\Ahttp(s?)://www.google.com/maps/l
end
```


User Story 1

- Category Model hinzufügen:
 - rails generate model Category name:string
- Seed-Daten für Categories:



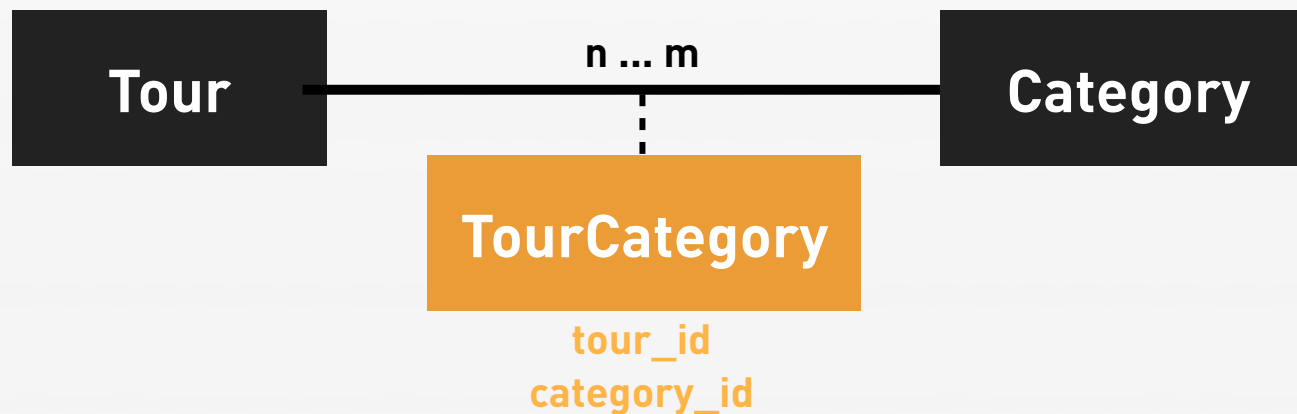
```
db/seeds.rb

Category.create name: 'Radtour'
Category.create name: 'Mountainbiketour'
Category.create name: 'Radwandern'
```

- rake db:migrate
- rake db:seed

User Story 1

- Join-Modell hinzufügen:



- rails generate model TourCategory tour:references category:references
- rake db:migrate

User Story 1

...

app/models/tour_category.rb

```
class TourCategory < ActiveRecord::Base
  belongs_to :tour
  belongs_to :category
end
```

Assoziation auf Model-Ebene herstellen

...

app/models/tour.rb

```
class Tour < ActiveRecord::Base
  has_many :tour_categories
  has_many :categories, through: :tour_categories
  validates :title, :teaser, :description, :url, presence: true
  validates :url, format: %r!\Ahttp(s?)://www.google.com/maps/|
end
```

Assoziation auf Model-Ebene herstellen

User Story 1

...

app/models/category.rb

```
class Category < ActiveRecord::Base
  has_many :tour_categories
  has_many :tours, through: :tour_categories
end
```

Assoziation auf Model-Ebene herstellen

User Story 1

```
app/views/tours/_form.html.erb

<div>
  <%= f.label :category %><br>
  <%= f.collection_select :category_ids,
    Category.all, :id, :name, { selected: @tour.category_ids,
    multiple: true } %>
</div>
```

Assoziation im Frontend bearbeitbar machen

HAML



Was ist HAML:
Beautiful, DRY,
well-indented, clear markup:
templating haiku.

- **Templating Engine für Ruby/Rails**
 - Als Ersatz für ERB gedacht
 - Parallele Benutzung von HAML und ERB ist aber möglich
- **Ziel:**
 - Weniger Aufwand beim Schreiben von Markup durch Einführung von Regeln

HAML

EXKURS

.erb

```
<section class="container">
  <h1><%= post.title %></h1>
  <h2><%= post.subtitle %></h2>
  <div class="content">
    <%= post.content %>
  </div>
</section>
```



.haml

```
%section.container
  %h1= post.title
  %h2= post.subtitle
  .content
    = post.content
```

HAML

- Regeln:

- %: Definition von HTML-Tags, z.B. %h1
- .: Definition von Klassen, z.B. .content
`<div class='content'></div>`
- #: Definition von IDs, z.B. #headline
`<div id='headline'></div>`

- Weitere Beispiele:

- siehe: <http://haml.info/tutorial.html>

User Story 1

```
app/helpers/tour_helper.rb

module ToursHelper
  def iframe_url(tour)
    tour.url + "&output=embed"
  end
end
```

- iFrame URL erzeugen

User Story 1

```
app/views/tours/show.html.erb

<p>
  <strong>Description:</strong>
  <%= @tour.description %>
</p>

<p>
  <iframe width=1200 height=600 frameborder=0 scrolling=no
marginheight=0 marginwidth=0 src=<%= iframe_url(@tour)%>></
iframe>
</p>
```

User Story 1

```
app/views/tours/index.html.erb

<h1>Listing tours</h1>

<ul>
  <% @tours.each do |tour| %>
    <li><%= link_to tour.title, tour %></li>
  <% end %>
</ul>

<br>

<%= link_to 'New Tour', new_tour_path %>
```


Taskboard



Deployment auf heroku

heroku:

Sehr einfach verwendbare Lösung für das Deployment von Rails-Anwendungen im Web

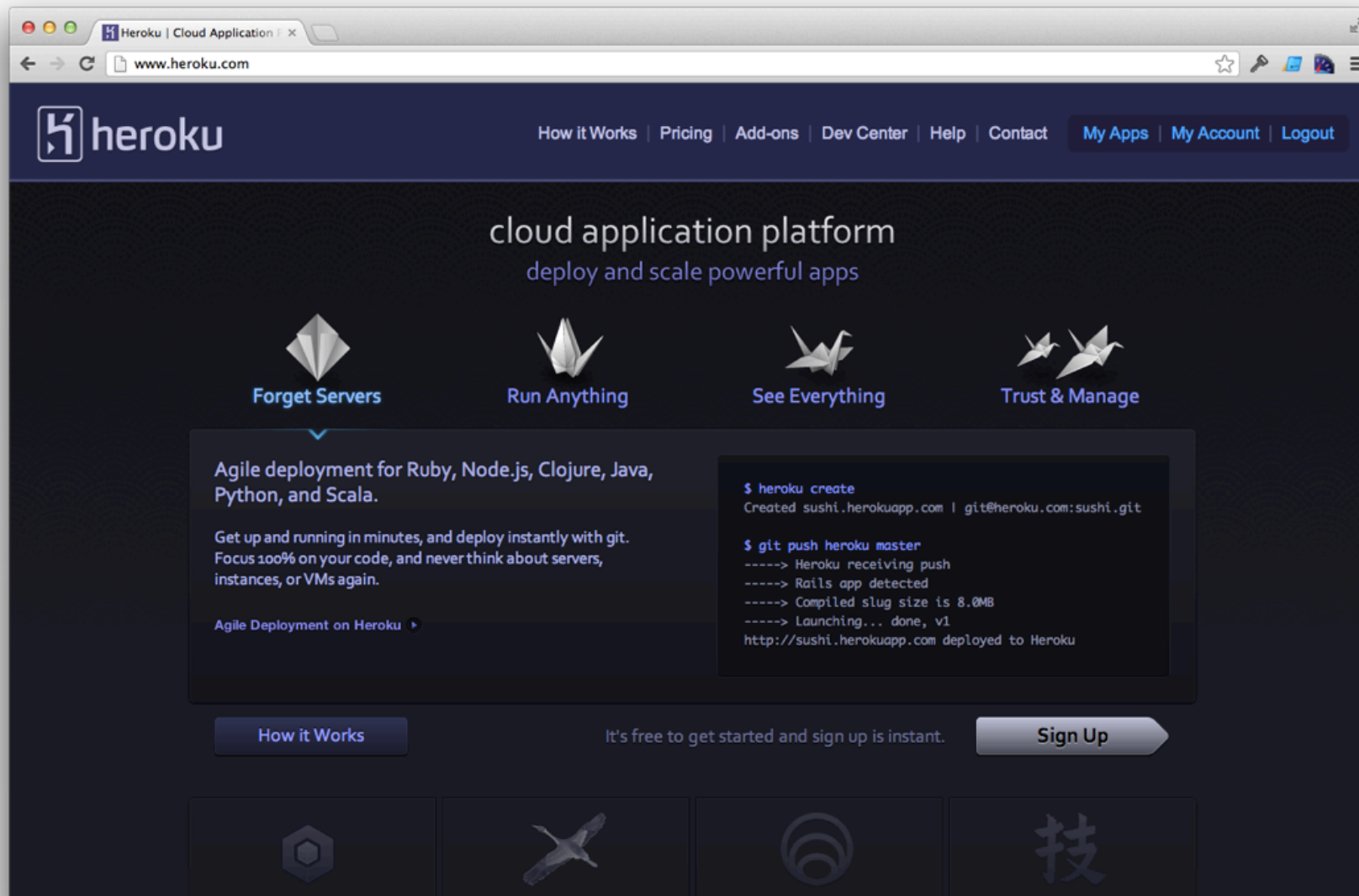
- **Vorteile:**

- Cloud-basiert (Abstraktion von physischen Servern)
- Schnelles Aufsetzen der Umgebung
- Kein Server-Setup und Maintenance
- Kein manuelles Aufsetzen von Deployment-Skripten
- Kostenlos nutzbar

- **Fazit:**

- Perfekt geeignet für Staging-Umgebungen

Deployment auf heroku



Deployment auf heroku

The screenshot shows the Heroku website's 'How it Works' page, specifically the 'Deploy' section. The page is divided into two main sections: 'Build' and 'Operate'. The 'Build' section includes 'Deploy', 'Connect', and 'Command'. The 'Operate' section includes 'Observe', 'Scale', and 'Relax'. Below these sections, there are three main articles: 'Instant Deployment', 'Continuous Deployment', and 'Poka-yoke (ボカヨケ)'. The 'Instant Deployment' article describes how deploying an app is simple and easy, requiring no special tools, just a plain `git push`. The 'Continuous Deployment' article describes how to easily create testing, staging, and production versions of an app and deploy to and between them instantly. The 'Poka-yoke (ボカヨケ)' article describes how Heroku follows a poka-yoke (mistake-proof) design philosophy, including role-based permissions, integrity checks during push, and robust release management and rollback controls. The architecture diagram on the right shows the 'Control Surface APIs' layer, the 'Dyno Manifold' layer, and the 'Logplex' layer. The 'Dyno Manifold' layer is further divided into 'Routing Dynos' and 'Process Types'.

Heroku | How it Works

www.heroku.com/how/deploy

heroku

How it Works | Pricing | Add-ons | Dev Center | Help | Contact | My Apps | My Account | Logout

Build | Operate

Deploy | Connect | Command | Observe | Scale | Relax

Instant Deployment

Deploying an app is simple and easy. No special tools needed, just a plain `git push`. Deployment is instant, whether your app is big or small.

[Read more about git deployment...](#)

Continuous Deployment

Easily create testing, staging, and production versions of your app and deploy to and between them instantly. The **dyno manifold** ensures all parts of your app are updated and bounced gracefully, **routing** continues seamlessly, and traffic is held for data migrations by custom maintenance **controls**.

Poka-yoke (ボカヨケ)

Move quickly and with confidence: Heroku follows a poka-yoke (mistake-proof) **design philosophy**, including role-based permissions, integrity checks during push, and robust release management and rollback **controls**. If in doubt, watch every detail of the deployment process in real-time with **Logplex**.

Control Surface APIs

Dyno Manifold

Logplex

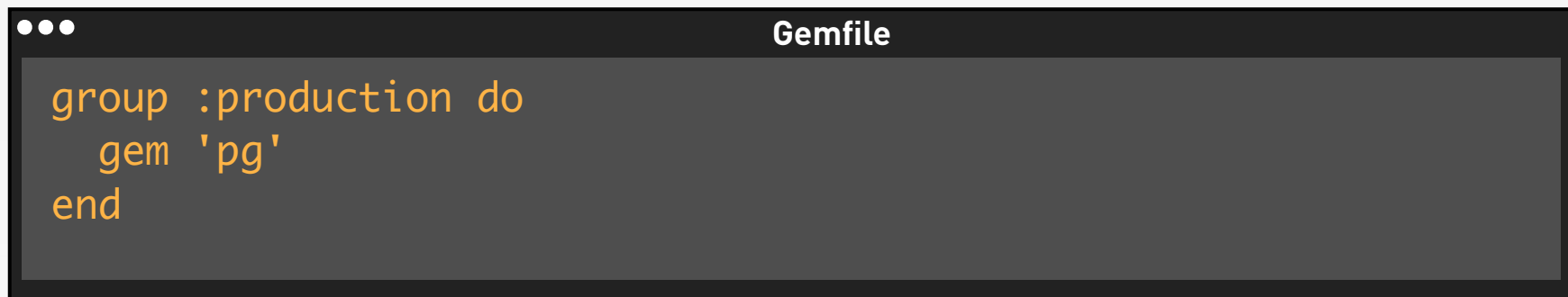
Routing Dynos

Process Types

Heroku | Business | Platform | Resources

Deployment auf heroku

- Heroku Tools installieren:
 - `gem install heroku`
- Heroku aufsetzen:
 - `heroku login`
 - `heroku apps:create leeze`
- Postgresql in Gemfile einfügen, weil heroku darauf basiert:



```
Gemfile

group :production do
  gem 'pg'
end
```

Deployment auf heroku

- `sqlite3` in development group
- Anwendung auf heroku deployen (ggf. vorher git Repo anlegen):
 - `git push heroku master`
- Datenbank initialisieren:
 - `heroku run rake db:migrate`
 - `heroku run rake db:seed`