

# Cloud Fundamentals

## Cloud Solutions FS 2017

Abteilung Informatik  
Hochschule für Technik Rapperswil

Autoren: Andreas Stalder, David Meister  
Datum: 20. März 2017

# Inhaltsverzeichnis

<b>1</b>	<b>Hello World</b>	<b>3</b>
1.1	Providerwahl . . . . .	3
1.2	Anleitung . . . . .	3
1.3	Vergleich mit Google App Engine . . . . .	9
<b>2</b>	<b>OSSM-Definition</b>	<b>10</b>
2.1	On-Demand . . . . .	10
2.2	Self-Service . . . . .	10
2.3	Scalable . . . . .	10
2.4	Measureable . . . . .	11
<b>3</b>	<b>Cloud Computing Patterns</b>	<b>12</b>
3.1	Process Offering . . . . .	12
3.2	Workload . . . . .	13
3.3	Komponenten . . . . .	13
3.4	Übersichtsgrafik . . . . .	14
<b>4</b>	<b>Self-Information</b>	<b>15</b>
<b>5</b>	<b>Preisrecherche</b>	<b>17</b>
<b>6</b>	<b>Preisvergleich Hosting vs IaaS vs PaaS</b>	<b>19</b>
<b>7</b>	<b>Twelve-Factor Apps</b>	<b>21</b>
7.1	Welche Anpassungen müssten Sie an Ihrer Applikation vornehmen? . . . . .	21

# 1 Hello World











## 1.1 Providerwahl

Wir haben uns für Swisscom als Provider entschieden, da es uns wichtig war, einen lokalen, heimischen Anbieter bezüglich Cloud Computing besser kennenzulernen. Swisscom ist in vielen Bereichen der IT und Telekommunikation schweizweit marktführend.

In einem Gastvortrag an der HSR im Herbst 2016 ist ein Vertreter der Swisscom erschienen und hat von den Vorzügen und der gelebten Swissness erzählt. Da wir noch nie mit PaaS Produkten im Allgemeinen gearbeitet haben, war dies die perfekte Möglichkeit um einen ersten Einblick in diese Thematik zu erhalten.

## 1.2 Anleitung

Der Einstieg fiel allgemein gesehen leicht. Swisscom verwendet „Cloud Foundry“ für ihre Application Cloud. Mittels Tutorials auf der Webseite wird man für den Einstieg gut „an der Hand genommen“.

<p>Unter <a href="https://docs.developer.swisscom.com/getting-started/">https://docs.developer.swisscom.com/getting-started/</a> findet man Tutorials für die verschiedenen PaaS Produkte wie z.B. Python, Java, Node.js</p>	<div><div><div>Getting Started</div><div>&gt; Web Console</div><div>&gt; Cloud Foundry Command Line Interface (cf CLI)</div><div>&gt; Develop and Manage Applications</div><div>&gt; Buildpacks</div><div>&gt; Integrating Service Instances with Applications</div><div>&gt; Service Marketplace</div><div>&gt; Billing</div><div>&gt; Cloud Foundry Concepts</div><div>API Reference</div><div>Release Notes</div></div><div><h3>Getting Started</h3><p>Page last updated: November 7, 2016</p><p>Welcome to Swisscom Application Cloud! To get started, please choose your preferred technology below or browse the documentation directly.</p><h4>Choose Your Technology</h4><div><div> Binary</div><div> Go</div><div> Java</div><div> .NET Core</div><div> Node.js</div><div> PHP</div><div> Python</div><div> Ruby</div><div> Clojure</div><div> Static Files</div></div></div></div>
--	--

Als allererstes benötigt man das Cloud Foundry Command Line Interface. Auf der Webseite sind Beschreibungen für die Installation unter Linux/Macintosh/Windows zu finden. (Windows in dieser Anleitung)

## Windows Installation

To use the cf CLI installer for Windows, perform the following steps:

1. Download [the Windows installer](#).
2. Unpack the zip file.
3. Double click the `cf CLI` executable.
4. When prompted, click Install, then Close.
5. To verify your installation, open a terminal window and type `cf`. If your installation was successful, the cf CLI help listing appears.

Nach erfolgreicher Installation ist das „cf“-Tool per Windows Commandline (cmd) benutzbar

```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\Master Dave>cf
cf version 6.25.0+787326d.2017-02-28. Cloud Foundry command line tool
Usage: cf [global options] command [arguments...] [command options]

Before getting started:
  config      login,l      target,t
  help,h      logout,lo

Application lifecycle:
  apps,a      run-task,rt  events
  push,p      logs          set-env,se
  start,st    ssh          create-app-manifest
  stop,sp     app
  restart,rs  env,e
  restage,rg  scale

Services integration:
  marketplace,m      create-user-provided-service,cups
  services,s         update-user-provided-service,uups
  create-service,cs  create-service-key,csk
  update-service    delete-service-key,dsk
  delete-service,ds service-keys,sk
  service          service-key
  bind-service,bs  bind-route-service,hrs
  unbind-service,us unbind-route-service,urs

Route and domain management:
  routes,r      delete-route      create-domain
  domains      map-route
  create-route  unmap-route

Space management:
  spaces      create-space      set-space-role
  space-users delete-space      unset-space-role

Org management:
  orgs,o      set-org-role
  org-users   unset-org-role

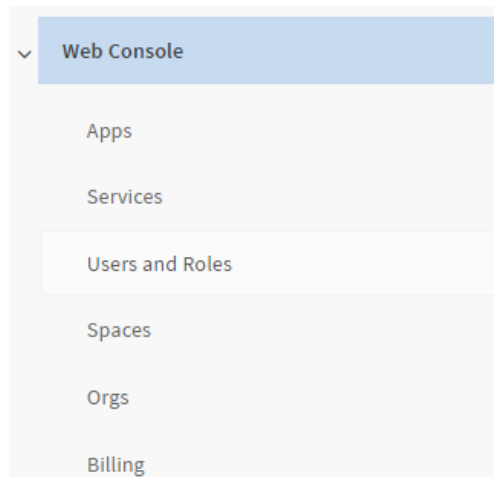
CLI plugin management:
  plugins      add-plugin-repo      repo-plugins
  install-plugin list-plugin-repos

Commands offered by installed plugins:

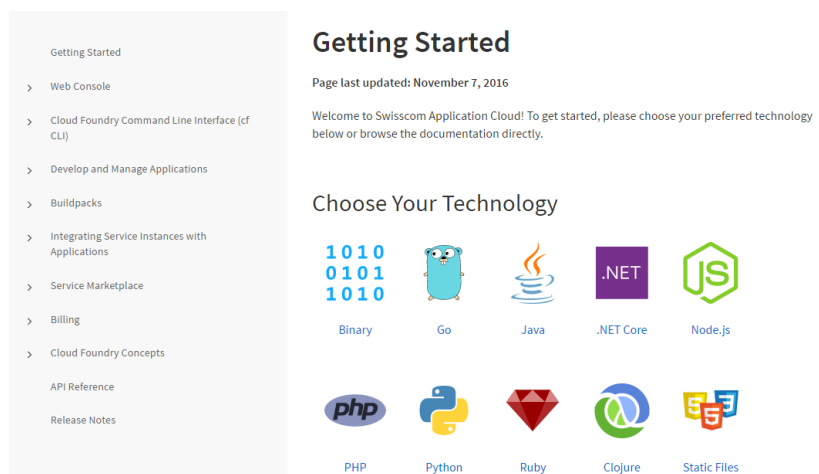
Global options:
  --help, -h      Show help
  -v             Print API request diagnostics to stdout

These are commonly used commands. Use 'cf help -a' to see all, with descriptions
See 'cf help <command>' to read about a specific command.
```

Nun können wir uns mit der Web Console von der Swisscom App Cloud vertraut machen. Auch hier gibt es ein Tutorial



Jetzt können wir das Tutorial für eine Java App durcharbeiten



Als Erstes sollte man die Prereqs überprüfen

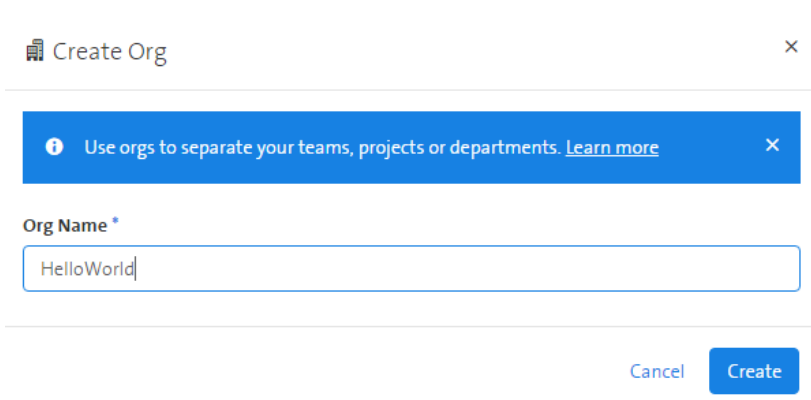
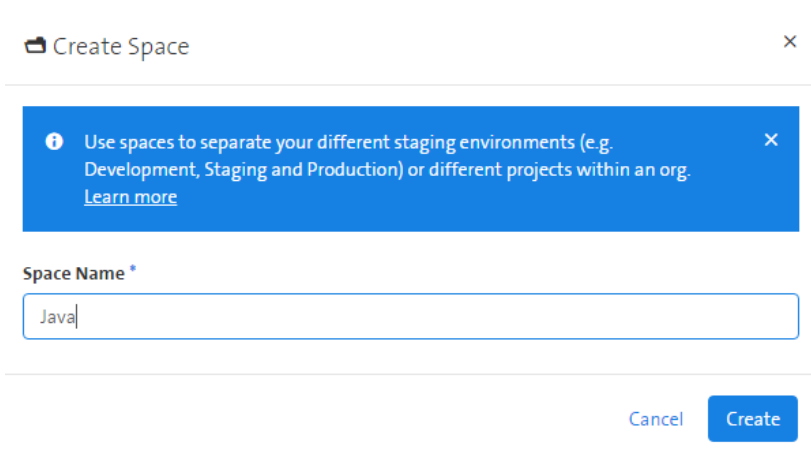
The tutorial assumes that you have:

- An [App Cloud account](#)
- The latest [Java JDK](#) installed locally
- The latest [Gradle](#) installed locally

Nun muss man sich über die cf Console an der Swisscom App Cloud anmelden. Wir skippen die Zuweisung an eine Org.

```
C:\Users\Master Dave>cf login -a https://api.lyra-836.appcloud.swisscom.com
API endpoint: https://api.lyra-836.appcloud.swisscom.com
Email> dmeister@hsr.ch
Password>
Authenticating...
OK
Select an org (or press enter to skip):
1. HSR-CloudSolution
2. HSR-CloudSolution_Personal
Org>

API endpoint: https://api.lyra-836.appcloud.swisscom.com (API version: 2.65.0)
User: dmeister@hsr.ch
No org or space targeted, use 'cf target -o ORG -s SPACE'
```

<p>Unter <a href="https://console.developer.swisscom.com">https://console.developer.swisscom.com</a> erstellen wir eine neue Org Namens „HelloWorld“.</p>	
<p>In dieser Org müssen wir einen Space erstellen.</p>	
<p>Wir weisen nun Org und Space mittels „cf target -o HelloWorld -s Java“ zu</p>	<pre data-bbox="616 1088 1430 1214">C:\git\Cloud-Solutions_Cloud-Fundamentals\src\cf-sample-app-java&gt;cf target -o HelloWorld -s Java API endpoint: https://api.lyra-836.appcloud.swisscom.com API version: 2.65.0 User: dneister@hsr.ch Org: HelloWorld Space: Java</pre>
<p>Wir kopieren nun die Java Beispielapp von Swisscom</p>	<pre data-bbox="616 1240 1430 1366">\$ git clone https://github.com/swisscom/cf-sample-app-java.git Cloning into 'cf-sample-app-java'... remote: Counting objects: 202, done. remote: Total 202 (delta 0), reused 0 (delta 0), pack-reuse 202 Receiving objects: 100% (202/202), 17.69 KiB   0 bytes/s, done. Resolving deltas: 100% (44/44), done. Checking connectivity... done.</pre>
<p>Nun können wir mit Gradle die App bauen. Es wurde ein jar generiert.</p>	<pre data-bbox="616 1393 1430 1644">C:\git\Cloud-Solutions_Cloud-Fundamentals\src\cf-sample-app-java&gt;gradle build :compileJava :processResources UP-TO-DATE :classes :jar :assemble :compileTestJava UP-TO-DATE :processTestResources UP-TO-DATE :testClasses UP-TO-DATE :test UP-TO-DATE :check UP-TO-DATE :build BUILD SUCCESSFUL Total time: 10.665 secs</pre>

Diese .jar File können wir nun mittels „cf push my-java-app“ -p <path to jar>, -n „pushen	<pre> C:\git\Cloud-Solutions_Cloud-Fundamentals\src\cf-sample-app-java&gt;cf push my-java-app -p build/libs/cf-sample-app-java-1.0.0.jar -n HelloHost Updating app my-java-app in org HelloWorld / space Java as dmeister@hsr.ch... OK  Creating route HelloHost.scapp.io... OK  Binding HelloHost.scapp.io to my-java-app... OK  Uploading my-java-app... Uploading app files from: C:\Users\MASTER~1\AppData\Local\Temp\unzipped-app186465975 Uploading 3.4M, 2106 files Done uploading OK  Starting app my-java-app in org HelloWorld / space Java as dmeister@hsr.ch... </pre>
Sobald die App erfolgreich gestartet wurde erscheinen folgende Meldungen	<pre> 1 of 1 instances running App started OK  App my-java-app was started using this command 'CALCULATED_MEMORY=\${PWD}/.java-buildpack/open_jdk_jre/bin/java-buildpack-memory-calculator-2.0.2_RELEASE -memory Sizes=metaspace:64m...stack:228k..-memoryWeights=heap:65,metaspace:10,native:15,stack:10 -memoryInitials=heap:100%,metaspace:100% -stackLHreads=300 -toMemory=SHEMORY_LIMIT) &amp;&amp; JAVA_OPTS="-Djava.io.tmpdir=\$TMPDIR -XX:OnOutOfMemoryError=\$PWD/.java-buildpack/open_jdk_jre/bin/killjava.sh \$CALCULATED_MEMORY" &amp;&amp; eval exec \$PWD/.java-buildpack/open_jdk_jre/bin/java \$JAVA_OPTS -cp \$PWD/. com.swisscom.cloud.cloudfoundry.sampleapp.java.ProductService'  Showing health and status for app my-java-app in org HelloWorld / space Java as dmeister@hsr.ch... OK  requested state: started instances: 1/1 usage: 1G x 1 instances urls: HelloHost.scapp.io last uploaded: Mon Mar 6 16:09:00 UTC 2017 stack: cflinuxfs2 buildpack: java-buildpack=v3.10-https://github.com/cloudfoundry/java-buildpack.git#193d6b7 java-main open-jdk-like-jre=1.8.0_111 open-jdk-like-memory-calculator=2.0.2_RELEASE  s      state      since                cpu    memory       disk        detail #0    running    2017-03-06 05:09:50 PM  0.0%   35.8M of 1G  134M of 1G </pre>
Mittels „cf app my-java-app“ kann man den Status der Java App abrufen	<pre> C:\git\Cloud-Solutions_Cloud-Fundamentals\src\cf-sample-app-java&gt;cf app my-java-app Showing health and status for app my-java-app in org HelloWorld / space Java as dmeister@hsr.ch... OK  requested state: started instances: 1/1 usage: 1G x 1 instances urls: HelloHost.scapp.io last uploaded: Mon Mar 6 16:09:00 UTC 2017 stack: cflinuxfs2 buildpack: java-buildpack=v3.10-https://github.com/cloudfoundry/java-buildpack.git#193d6b7 java-main open-jdk-like-jre=1.8.0_111 open-jdk-like-memory-calculator=2.0.2_RELEASE  s      state      since                cpu    memory       disk        detail #0    running    2017-03-06 05:09:50 PM  0.1%   56.2M of 1G  134M of 1G </pre>
Mittels „cf scale my-java-app“ erhält man eine Übersicht über die verwendeten Ressourcen Memory, Disk, Instances	<pre> C:\git\Cloud-Solutions_Cloud-Fundamentals\src\cf-sample-app-java&gt;cf scale my-java-app Showing current scale of app my-java-app in org HelloWorld / space Java as dmeister@hsr.ch... OK  memory: 1G disk: 1G instances: 1 </pre>
Horizontales Scaling geschieht mittels „cf scale my-java-app -i 3“. So werden nun 3 Instanzen verwendet	<pre> C:\git\Cloud-Solutions_Cloud-Fundamentals\src\cf-sample-app-java&gt;cf scale my-java-app -i 3 Scaling app my-java-app in org HelloWorld / space Java as dmeister@hsr.ch... OK </pre>

Vertikales Scaling geschieht mittels „cf scale my-java-app -m 2G“. So werden 2 GB RAM verwendet

```
C:\git\Cloud-Solutions_Cloud-Fundamentals\src\cf-sample-app-java>cf scale my-java-app -m 2G
This will cause the app to restart. Are you sure you want to scale my-java-app?
yes
Scaling app my-java-app in org HelloWorld / space Java as dmeister@hsr.ch...
OK
Stopping app my-java-app in org HelloWorld / space Java as dmeister@hsr.ch...
OK
Starting app my-java-app in org HelloWorld / space Java as dmeister@hsr.ch...
0 of 3 instances running, 3 starting
3 of 3 instances running
App started
OK
App my-java-app was started using this command `CALCULATED_MEMORY=${PWD}/.java-buildpack/open_jdk_jre/bin/java-buildpack-memory-calculator-2.0.2_RELEASE -memorySizes=metaspace:64m...stack:228k..-memoryWeights=heap:65,metaspace:10,native:15,stack:10 -memoryInitials=heap:100%,metaspace:100% -stackThreads=300 -totMemory=SMEMORY_LIMIT> && JAVA_OPTS="-Djava.io.tmpdir=$TMPDIR -XX:OnOutOfMemoryError=$PWD/.java-buildpack/open_jdk_jre/bin/killjava.sh $CALCULATED_MEMORY" && eval exec $PWD/.java-buildpack/open_jdk_jre/bin/java $JAVA_OPTS -cp $PWD/.com.swisscom.cloud.cloudfoundry.sampleapp.java.ProductService`
Showing health and status for app my-java-app in org HelloWorld / space Java as dmeister@hsr.ch...
OK
requested state: started
instances: 3/3
usage: 2G x 3 instances
urls: HelloHost.scapp.io
last uploaded: Mon Mar 6 16:09:00 UTC 2017
stack: cflinuxfs2
buildpack: java-buildpack=v3.10-https://github.com/cloudfoundry/java-buildpack.git#193d6b7 java-main open-jdk-like-jre=1.8.0_111 open-jdk-like-memory-calculator=2.0.2_RELEASE
```

#	state	since	cpu	memory	disk	detail
#0	running	2017-03-06 05:20:47 PM	0.1%	72.1M of 2G	134M of 1G	
#1	running	2017-03-06 05:20:47 PM	0.0%	48.1M of 2G	134M of 1G	
#2	running	2017-03-06 05:20:47 PM	0.1%	31.9M of 2G	134M of 1G	

Es gilt nun zu beachten, dass nach dem Scaling die Betriebskosten gestiegen sind.

#	STATUS	SINCE	CPU	MEMORY	DISK
0	Running	2 minutes	0.1%	60 MB	134 MB
1	Running	2 minutes	0.2%	61.9 MB	134 MB
2	Running	2 minutes	0.2%	32.7 MB	134 MB

Die Logfiles können einfach mittels „cf logs my-java-app -recent“ abgerufen werden

```
C:\git\Cloud-Solutions_Cloud-Fundamentals\src\cf-sample-app-java>cf stop my-java-app
Stopping app my-java-app in org HelloWorld / space Java as dmeister@hsr.ch...
OK
```



Mittels „cf stop my-java-app“ kann die App gestoppt werden. Falls sie wieder benötigt wird kann sie mittels „cf start my-java-app“ wieder hochgefahren werden.

```
C:\git\Cloud-Solutions_Cloud-Fundamentals\src\cf-sample-app-java>cf start my-jav
a-app
Starting app my-java-app in org HelloWorld / space Java as dmeister@hsr.ch...
0 of 1 instances running, 1 starting
1 of 1 instances running
App started
OK
App my-java-app was started using this command 'CALCULATED_MEMORY=${PWD}/.java-b
uildpack/open_jdk_jre/bin/java-buildpack-memory-calculator-2.0.2_RELEASE -memory
Sizes=metaspace:64m...stack:228k...-memoryWeights=heap:65,metaspace:10,native:15
...stack:10 -memoryInitials=heap:100%,metaspace:100% -stackThreads=300 -totMemory=
$MEMORY_LIMIT && JAVA_OPTS="-Djava.io.tmpdir=$TMPDIR -XX:OnOutOfMemoryError=$PW
D/.java-buildpack/open_jdk_jre/bin/killjava.sh $CALCULATED_MEMORY" && eval exec
$PWD/.java-buildpack/open_jdk_jre/bin/java $JAVA_OPTS -cp $PWD/. com.swisscom.c
loud.cloudfoundry.sampleapp.java.ProductService'
Showing health and status for app my-java-app in org HelloWorld / space Java as
dmeister@hsr.ch...
OK
requested state: started
instances: 1/1
usage: 1G x 1 instances
urls: HelloHost.scapp.io
last uploaded: Mon Mar 6 16:09:00 UTC 2017
stack: cflinuxfs2
buildpack: java-buildpack=v3.10-https://github.com/cloudfoundry/java-buildpack.g
it#193d6b7 java-main open-jdk-like-jre=1.8.0_111 open-jdk-like-memory-calculator
=2.0.2_RELEASE
```

	state	since	cpu	memory	disk	details
#0	running	2017-03-06 05:29:24 PM	0.0%	896K of 1G	1.3M of 1G	

### 1.3 Vergleich mit Google App Engine

Verglichen mit der Google App Engine wirkt die Swisscom Developer Console schlanker. Man findet sich viel leichter zurecht da das UI besser gestaltet wurde als bei Google.

Sobald man einmal die Tutorials gefunden hat, bedient sich die App Engine sehr ähnlich der Swisscom Application Cloud. Bei Swisscom muss die Cloud Foundry Shell lokal installiert werden, bei GAE kann man die Online Google Cloud Shell verwenden.

Das Pricing (siehe später) scheint bei Swisscom einfacher und transparenter als bei GAE.

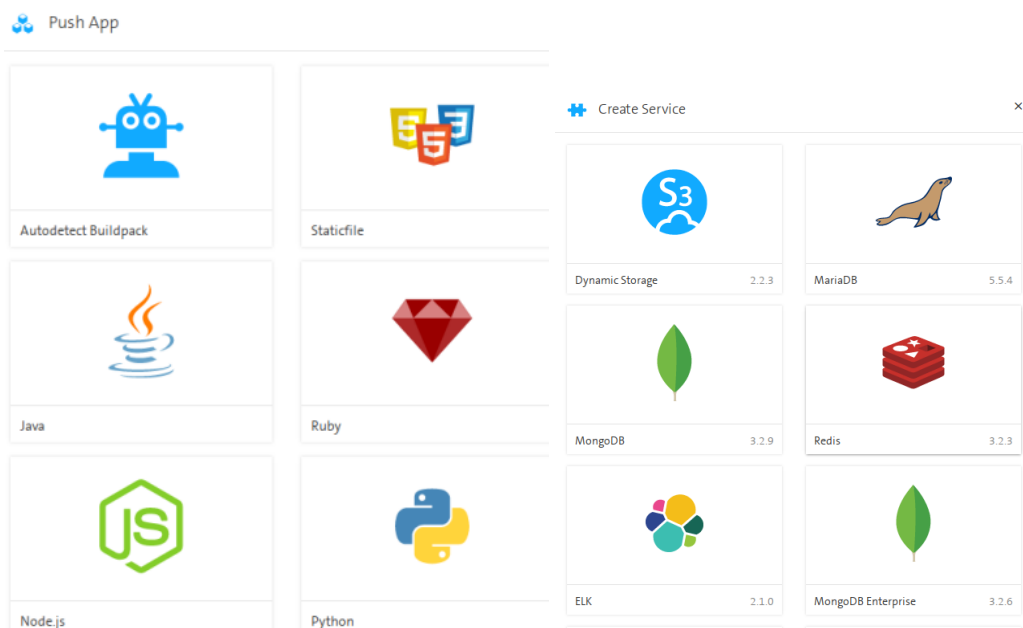
## 2 OSSM-Definition

### 2.1 On-Demand

Es gibt mehrere unterstützte Services und Programmiersprachen, die bereits verfügbar sind. Nach der Anmeldung und der Angabe der Kreditkarte können solche Instanzen erstellt werden und sind ohne Wartezeiten einsatzbereit. Dies merkt der User durch die vorbereiteten Templates, welche er schnell erstellen und starten kann.


### 2.2 Self-Service

Es gibt bereits viele verschiedene Applikation-Templates für viele Programmiersprachen. So kann der Benutzer einfach eine neue Applikation erstellen und auf die Website pushen. Die lokal erstellte Applikation wird entweder via CLI oder über die Weboberfläche in die Cloud geladen.



### 2.3 Scalable

Bei der Swisscom kann man mit Scale Up, wie auch Scale Out skalieren. Dies geschieht entweder mit der Cloud Foundry CLI oder über die Weboberfläche. Bei der Weboberfläche gibt es zum Teil noch Begrenzungen, da die Oberfläche noch nicht komplett an das CLI angepasst ist. (Zum Teil lässt sich maximal nur 2 GB Arbeitsspeicher einstellen). Der Festplattenspeicher kann nur über das CLI eingestellt werden. Die Skalierung muss von Hand erledigt werden, da es keine Möglichkeit zur automatischen Skalierung gibt.



myFancyApp

Started 0/1

Instances

-

1

+

Memory Limit

-

256 MB

+

Disk Limit 1 GB

Restart

Restage

Stop

Delete

## 2.4 Measureable

Bei der Cloudlösung der Swisscom sieht man die Kosten bereits beim Erstellen des Services. Dabei werden die Kosten aus der Anzahl Instanzen und der Arbeitsspeicherlimite berechnet. Es gibt eine Formel, welche genau zeigt wie teuer eine Applikation ist. Diese findet sich weiter unten im Bericht. Gezahlt wird die monatliche Rechnung per Kreditkarte. Auf der Rechnung sind alle wichtigen Punkte aufgelistet und es gibt keine versteckte Kosten.

App Name \*

myFancyApp

Instances

-

8

+

Memory Limit

-

2048 MB

+

Estimated Monthly Cost

CHF 384.12

Application Cloud Service	Amount excl. VAT	VAT in %	Amount incl. VAT
Service total incl. VAT in CHF			

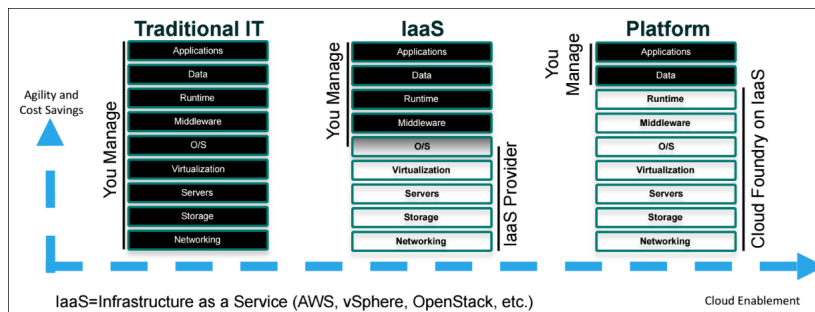
Invoice total	Amount incl. VAT
Invoice total incl. VAT in CHF	0.00

VAT totals	VAT	excl. VAT	incl. VAT
VAT 8%	0.00	0.00	0.00

### 3 Cloud Computing Patterns

Die Swisscom Application Cloud setzt das Service-Modell „Platform as a Service“ (PaaS) um. Im Bild sind die drei vorgängigen Servicemodelle im Vergleich mit PaaS ersichtlich.

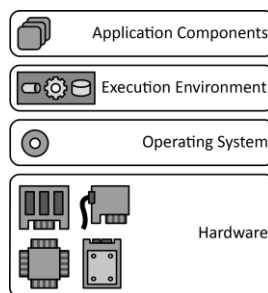


Swisscom verwendet Cloud Foundry für ihre Application Cloud Plattform. Cloud Foundry wird ebenfalls von vielen anderen Providern und Firmen (z.B. IBM, Pivotal, SAP, usw.) verwendet sowie weiterentwickelt. Somit ist Cloud Foundry ein Open Source Projekt. Diese Konstellation verhindert Vendor-lock-ins, d.b. es sollte sehr einfach sein die Provider für die eigenen Enterprise Applikationen auszutauschen. Dies generiert für die Kunden einen echten Mehrwert.

#### 3.1 Process Offering

Als Entwickler möchte man eine fix-fertige Umgebung aus einem Guss. Die Swisscom Application Cloud bietet deshalb fertige Umgebungen für Programmierer.

Diese Anforderungen können mit dem Pattern „Execution Environment“ umgesetzt werden.



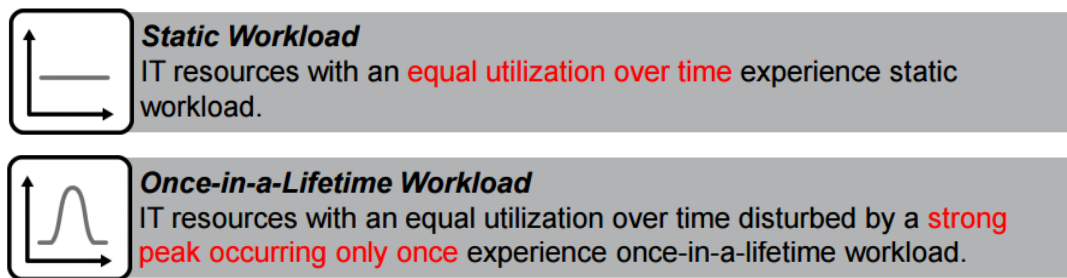
Für den Kunden ist vordergründig nicht viel sichtbar. Er kann seine Sources in die Cloud laden und von der Execution Environment ausführen lassen. Die App wird dann in einem Container ausgeführt und die darunterliegende Infrastruktur ist für den Kunden unbekannt.

## 3.2 Workload

Bei der Swisscom Application Cloud wird kein Auto-Scaling unterstützt. Dies verhindert den Einsatz bei gewissen Workload Patterns. Man kann problemlos die Applikation skalieren, jedoch muss dies manuell geschehen. Eine häufige Änderung des Workloads ist daher ungeeignet. Deshalb fallen folgende Workload Patterns wohl nicht in das Aufgabengebiet der Swisscom Application Cloud:

- Periodic Workload
- Unpredictable Workload
- Continuously Changing Workload

Hingegen sind die Workload Patterns „Static Workload“ und „Once-in-a-Lifetime“ Workload gut geeignet.



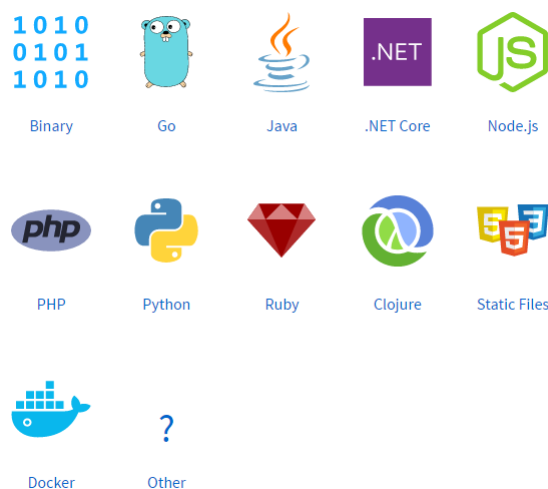
Bei einer statischen Workload muss offensichtlich nicht skaliert werden. Bei Events, welche einmalig auftreten ist es ebenfalls kein Problem, dies manuell zu skalieren.

## 3.3 Komponenten

Das Angebot in der Swisscom Application Cloud kann in zwei Gruppen unterteilt werden:

- Applications
- Services

Bei den Applications kann man als User seine Applikationen bereitstellen. Folgende Technologien stehen zur Verfügung:



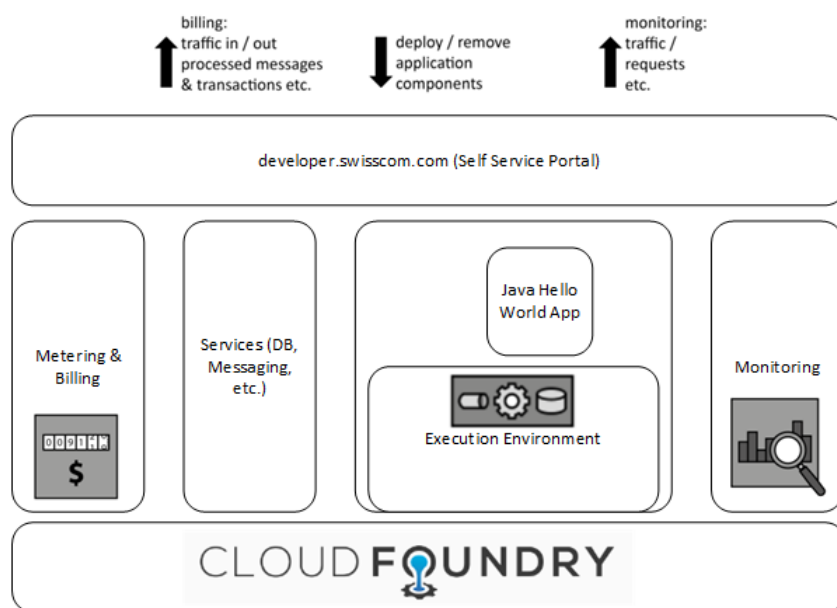
Als Services stehen bekannte Produkte unterschiedlicher Hersteller bereit. Diese wären beispielsweise MongoDB als Vertreter der NoSQL Familie. MariaDB fungiert als SQL Datenbank und Redis als Key-Value Store. Für Queueing und Messaging steht RabbitMQ zur Verfügung.

Swisscom Application Cloud stellt das CCP „Execution Environment“ dem Kunden zur Verfügung. Hypervisor als Process Offering ist an dieser Stelle fehl am Platz da man auf einer höheren Abstraktionsstufe arbeitet. Die Hardware resp. Virtualisierungsplattform ist hier nicht ersichtlich, resp. Black-Box.

Möchte man Map-Reduce Computing, so muss man sich selber darum kümmern, resp. es wird von Swisscom nichts out-of-the-box angeboten.

### 3.4 Übersichtsgrafik

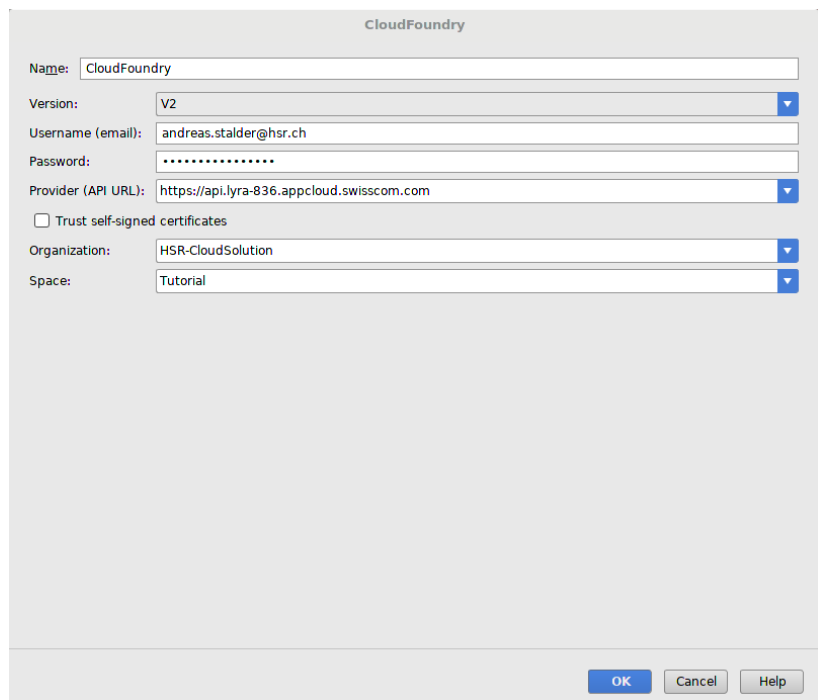
Die Hello World Java App kann gemäss dem Pattern „Elastic Platform“ platziert werden.



## 4 Self-Information

Bei der Installation gab es wenige Probleme. Hier sind die wichtigsten Schritte kurz aufgezeigt.

In IntelliJ importiert man sich das Projekt und lädt das Cloud Foundry Plugin herunter. Nun loggt man sich ein und erstellt eine Verbindung zu seiner Swisscom-Cloud. Die Organization und der Space werden nach dem Login sofort gefunden.

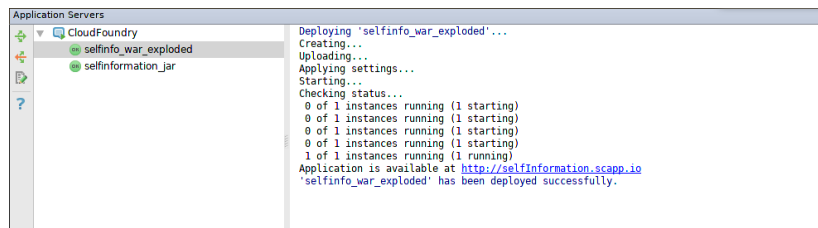


The screenshot shows the 'CloudFoundry' configuration window. It contains the following fields and options:

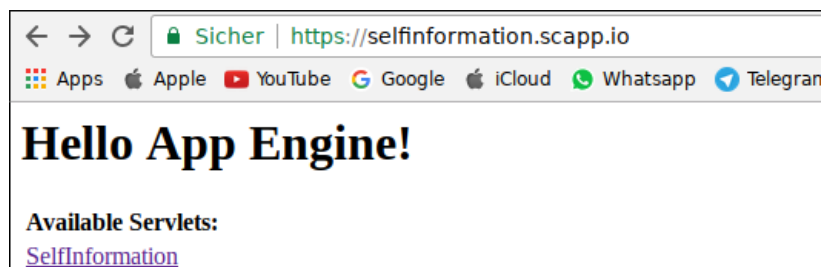
- Name: CloudFoundry
- Version: V2 (dropdown)
- Username (email): andreas.stalder@hsr.ch
- Password: (masked with dots)
- Provider (API URL): https://api.lyra-836.appcloud.swisscom.com (dropdown)
- ☐ Trust self-signed certificates
- Organization: HSR-CloudSolution (dropdown)
- Space: Tutorial (dropdown)

Buttons at the bottom: OK, Cancel, Help.

Nun wird die Applikation gebildet und auf die Swisscom-Cloud geladen. Wie man sieht, wird sofort eine Instanz erstellt und die Applikation ist erreichbar.



Hier sieht man nun, dass die Applikation bereits erreichbar ist.



### **Wie viel RAM steht Ihrer Cloud-Instanz maximal zur Verfügung?**

Der Cloud-Instanz steht bei der Swisscom maximal 2 GB zur Verfügung. Mehr kann man leider nicht zuweisen. Die App zeigt aber dies hier an:

- Maximum memory (megabytes): 1276
- Total memory (megabytes): 1276
- Total space (megabytes): 1007
- Free space (megabytes): 854
- Usable space (megabytes): 787

Wie es aussieht, kann die Applikation nicht die ganzen 2 GB nutzen. Vielleicht wird dies aber auch dynamisch angepasst.

### **Von welchem Hersteller stammt die JVM?**

- java.runtime.name: OpenJDK Runtime Environment
- java.vm.version: 25.111-b14
- java.vm.vendor: Oracle Corporation
- java.vm.name: OpenJDK 64-Bit Server VM

Die VM stammt von Oracle. Es wird das OpenJDK auf einem Linuxserver verwendet.

### **Welche IP-Adresse hat der Server laut der Self Information-Applikation?**

- Name 0: w9a98pllh480-1
- InetAddresses 0: /10.254.0.138
- Name 1: lo
- InetAddresses 1: /127.0.0.1

Es sind zwei Interface verbunden. Das Loopback und eines mit dem Namen: w9a98pllh480-1. Die IP-Adresse ist 10.254.0.138.



## 5 Preisrecherche

Swisscom verfolgt eine Usage-based Pricing Strategie. Man bezahlt also nur für laufende Dienste. Die Kosten setzen sich zusammen aus den Ressourcen für die App + zusätzliche Services. Seit Ende 2016 wird bei der Swisscom stündlich abgerechnet. Davor wurde in täglichen Intervallen abgerechnet. Die Preise sind einfachheitshalber aber meist in CHF pro Monat angegeben.

Für die Datenbankprodukte (MariaDB, MongoDB) hat Swisscom die Instanzen in drei verschiedenen Grössen kreiert (S, M, L). Man kann pro Grösse bis zu 10 Instanzen verwenden.

Bei RabbitMQ bezahlt man Messages pro Monat. Für 1 Mio. Messages bezahlt man CHF 13.20.-.

Für Diskspace bezahlt man einen Basispreis von CHF 9.00.-, damit hat man 100GB zur Verfügung. Für jede weiteren 100GB muss man zusätzliche CHF 6.00.- bezahlen.

Das Pricing für die Apps ist sehr einfach gehalten. Man hat nur 2 Faktoren, welche den Preis beeinflussen, nämlich Instanzen und Memory. Man bezahlt pro 64MB CHF 1.50.- pro Monat. Die Instanzen dienen als Faktor. Wählen wir beispielsweise für unsere Applikation 512MB mit 2 Instanzen, so kostet uns das  $(512MB/64MB) * 1.50CHF * 2 = 24CHF$  monatlich.

The image shows a pricing calculator interface with two sliders and a summary box. The first slider is labeled 'Memory (RAM)' with a subtext 'CHF 1.50 per 64 MB'. The slider is set to '512 MB'. The second slider is labeled 'Instances' with a subtext 'multiplier'. The slider is set to '2x'. Below these sliders is a summary box with three columns: 'Total', 'per month', and 'CHF 24.00'.

Memory (RAM)	Instances
512 MB	2x

Total	per month
	CHF 24.00

Vor der Erstellung einer App muss dem Account eine Kreditkarte hinterlegt werden. Eine Bezahlung auf Rechnung ist in der Public Application Cloud nicht möglich.

Für neue User steht eine „Welcome Offer“ zur Verfügung. Für die erste 3 Monate werden Kosten bis CHF 100.00.- pro Monat erlassen. So kann man als Interessent ausgiebig testen.

Die Preispolitik ist bei der Swisscom Application Cloud viel einfacher als bei AWS EC2. Die geforderte Konfiguration lässt sich weder mit Swisscom, noch mit AWS so einfach berechnen. Wir gehen nun von einer anderen Konfiguration aus:

- 2 Instanzen
- 512MB RAM pro Instanz
- 20GB Diskspace

Bei Swisscom kostet eine solche Konfiguration CHF 33.00.- monatlich. Bei AWS EC2 USD 11.44. Selbst wenn man die Währungsvorteile vom amerikanischen Dollar aussen vor lässt, ist Amazon doch sehr deut-

lich günstiger. Der Vergleich ist aber sehr gewagt, da bei Swisscom nicht ersichtlich ist, welche Computing Leistung man erhält. Bei AWS EC2 erhält man für diesen Preis eine sehr schwache CPU Leistung.

## 6 Preisvergleich Hosting vs IaaS vs PaaS

Bei der Preiskalkulation kamen wir auf folgendes Ergebnis:

Capital Expenses	VMware	Rackspace	Amazon	Swisscom
Servers	3612	1473	142	259
Storage	600	0	0	0
Networking	5833	0	0	0
Virtualization platform software	939.50	0	0	0
Host Windows Server & SQL Server licenses	0	0	0	0
Third-party software licenses	0	0	0	0
Total CapEx cost	10045	1473	142	259
Operating expenses (2 years)				
Power and cooling	1880	0	0	0
Data center	1187	0	0	0
Virtualization platform software support	5347	0	0	0
Windows Server & SQL Server host software support	7415	0	0	0
Third-party software support	0	0	0	0
IT administrative time cost	10291	2600	2600	2600
Third-party SW integration	0	0	0	0
Total OpEx cost	17413	1733	1733	1733
Totals				
Total cost	27458	35352	6010	8216

### Welche Variante ist die kostengünstigste?

Bei unserer Kalkulation ist die Variante von Amazon die günstigste Variante. Diese ist mit 6010 Euro für zwei Jahre knapp günstiger als die Swisscom Variante. Mit eigenen Servern ist man deutlich über diesem Preis.

### Auf welche Schwierigkeiten stösst man beim Vergleichen?

Es ist schwierig die Angebote miteinander zu vergleichen, da nicht jeder Anbieter das gleiche berechnen. Bei der Swisscom musste das Datenvolumen zum Beispiel nicht bezahlt werden. Oder VMware berechnet den Speicher eher teuer. Es ist auch sehr schwer abzuschätzen, was für eine Leistung man bekommt. Meistens stehen nur die CPU Kerne und nichts genaueres. So ist es schwer die Leistung zu vergleichen.

### Wie schätzen Sie die Preise für Cloud Computing insgesamt ein?

Die Preise sind sehr günstig. Man zahlt nur genau das was man benötigt. Dies ist bei eigenen Servern nicht so, da in diesem Fall die Server ja nicht die gesamte Zeit so ausgelastet sind, damit es sich lohnt.

### **Sollte man den Cloud-Anbieter rein nach gebotener Funktionalität und Preis auswählen oder gibt es weitere Entscheidungskriterien?**

Es gibt ganz klar noch weitere Entscheidungskriterien. So sollten zum Beispiel auch die SLAs betrachtet werden. Ist der Dienst nicht stabil oder hat man sonst viele Probleme, ist dies sicher auch ein Grund für einen Wechsel. Auch der Support kann entscheidend sein. Bei Problemen muss der Cloud-Anbieter unterstützen und die Probleme beseitigen. Soweit dies natürlich möglich ist.

### **Cloud-Provider oder klassisches Hosting?**

Bei einer solch kleinen Applikation muss man klar zum Cloudanbieter greifen. Mit einer eigenen Umgebung zahlt man um ein vielfaches mehr. In unserem Fall zahlt man 4x mehr für das klassische Hosting.

## 7 Twelve-Factor Apps

Als Beispielapplikation haben wir das Miniprojekt aus WED2 vom Jahr 2016 genommen. Dies ist eine kleine Notizenapp in Node.js geschrieben.

### 7.1 Welche Anpassungen müssten Sie an Ihrer Applikation vornehmen?

#### Codebase

Bei der Entwicklung der Notizenapp haben wir immer mit GitHub gearbeitet, da wir mehrere Entwickler waren. Da es eine kleine App war, haben wir aber immer nur ein Deploy geführt. Die Auftrennung in verschiedene Deploys war zu diesem Zeitpunkt noch nicht nötig, aber wäre schnell umsetzbar. Daher muss bei diesem Punkt nichts angepasst werden.

#### Dependencies

Da die App in Node geschrieben wurde, konnten wir bei den Dependencies auf npm zurückgreifen. In unserer package.json Datei gab es ein Dependencies Bereich mit allen Abhängigkeiten.

#### Config

Eine eigene Konfigurationsdatei gab es bei uns nicht. Dies müsste man für den 12-Faktoren Check eingebaut werden. Die Datenbankbindung könnte sicher in diese Konfigurationsdatei, sowie auch die Servereinstellungen wie der Port usw.

#### Backing services

Bei der Beispielapplikation wurde ein Backing services verwendet. Die Datenbank wurde zu Redis ausgelagert, damit man keinen lokalen Speicher verwenden musste.

#### Build, release, run

Wir haben keine Aufteilung in Build und Releases gemacht, da dies bei Node nicht üblich ist. Um dies einzurichten, könnte man Heroku Buildpack für Node.js einrichten, damit man verschiedene Builds hat. Weiter kann man via npm Parameter angeben. Zum Beispiel so: "npm install --production". Dies wurde aber nicht gemacht.

## **Processes**

Da wir nur ein Miniprojekt entwickelten, haben wir diesen Punkt nicht eingerichtet. Aber dies ist bei Node kein Problem. Unter Linux kann man einfach eine Systemd Service Konfigurationsdatei erstellen. Dieser wird in den Autostart verlinkt und so startet die Applikation immer in einem eigenen Service. Da die Daten in einer Datenbank gespeichert wurden, war die Applikation auch Stateless und konnte mehrmals gestartet werden.

## **Port binding**

Port Binding ist bei Node immer der Fall, da es sich bei Node um einen Webserver handelt. Bei unserem Beispiel konnten wir den Port 3000 einstellen. Das Port Binding könnte auch noch per Konfigurationsdatei für die Verschiedenen Bereiche (Development oder Running System) separat angegeben werden.

## **Concurrency**

Durch das asynchrone System von Node und ohne grössere Background Tasks, ist Concurrency ziemlich einfach. Einzig der Datenbankzugriff müsste wohl noch Concurrent gemacht werden, damit es bei der Datenbank nicht zu Race Conditions kommt.

## **Disposability**

Unsere Applikation unterstützt diesen Punkt. Node startet extrem schnell und die HTTP Requests sind auch kurz. Daher ist dieser Punkt bei so kleinen Node Projekten einfach erreichbar.

## **Dev/Prod parity**

Da wir hier keine Trennung hatten, da es sich nur um ein Miniprojekt handelte, ist uns dies gelungen. Bei einer Weiterentwicklung würde man dafür auf verschiedenen Branches auf GitHub arbeiten.

## **Logs**

In unserem Projekt wurde im Endprodukt kein Log eingebaut. Dies müsste noch eingebaut werden, damit man den 12-Faktor Standard erreicht. Im npm-Repo gibt es gute Logging Lösungen, welche dies einfach implementierbar machen.

## **Admin processes**

Admin Tasks werden bei Node, wie auch bei unserem Projekt in das /bin Verzeichnis verschoben. Danach wird mit eigenen Scripts gearbeitet, welche die Tasks im Laufenden Betrieb erledigen. Durch das schnelle starten von Node, sollte das kein Problem sein.

### **Wie unterscheidet sich diese Methode von IDEALen nach Fehling et al?**

Im Grunde wollen beide Methoden das selbe. 12-Faktor zeigt dies einfach in mehreren kleinen Kapiteln, anstelle von fünf grossen Kapiteln. Was aber im IDEAL fehlt oder weniger genau beschrieben ist, sind folgende Kapitel aus der 12-Faktor Methode:

- Codebase
- Build, release, run
- Port Binding
- Disposability
- Dev/prod partity
- Logs
- Admin processes

### **Welche der beiden Methoden scheint einfacher?**

Die 12-Faktor Methode scheint mir einfacher zu sein. Diese ist viel genauer beschrieben und man weiss bei jedem Punkt genau, was man anpassen muss.