

Digital Innovation Ruhr, 24.10.18

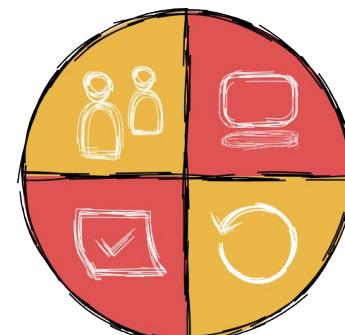
Es muss nicht immer gleich Docker sein
IT Automation, die zu einem passt

Sandra Parsick

mail@sandra-parsick.de
@SandraParsick

Zu meiner Person

- Sandra Parsick
- Freiberuflicher Softwareentwickler und Consultant im Java-Umfeld
- Schwerpunkte:
 - Java Enterprise Anwendungen
 - Agile Methoden
 - Software Craftmanship
 - Automatisierung von Entwicklungsprozessen
- Trainings
- Workshops
- Softwerkskammer Ruhrgebiet
- Twitter: @SandraParsick
- Blog:
<http://blog.sandra-parsick.de>
- E-Mail: mail@sandra-parsick.de



Motivation



Warum Ansible,
wenn Docker auch
diese Probleme löst?

Ansible for Developers

Aber dafür haben wir
doch Docker



Lass unsere Deployment mit Ansible automatisieren

Wir wollen doch nächstes Jahr oder später Docker einsetzen

Zu aufwendig zwischendurch Ansible einzuführen

Aber alle verwenden doch Docker



Wieso wird Ansible
mit Docker
gleich gesetzt?

Conference-
Driven-
Development?

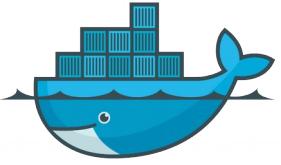
Ist der Unterschied
Container und
Provisionierungswerzeug
nicht klar?

Container vs. Provisionierungswerzeuge

Container - Allgemein

Container

- verpacken Anwendungen und ihre Abhängigkeiten zu einer Einheit
- isolieren diese von anderen Anwendungen
- standardisieren die Art und Weise der Auslieferung von Anwendungen

Container \in {  docker } ?



Fat JAR



Provisionierungswerkzeug - Allgemein

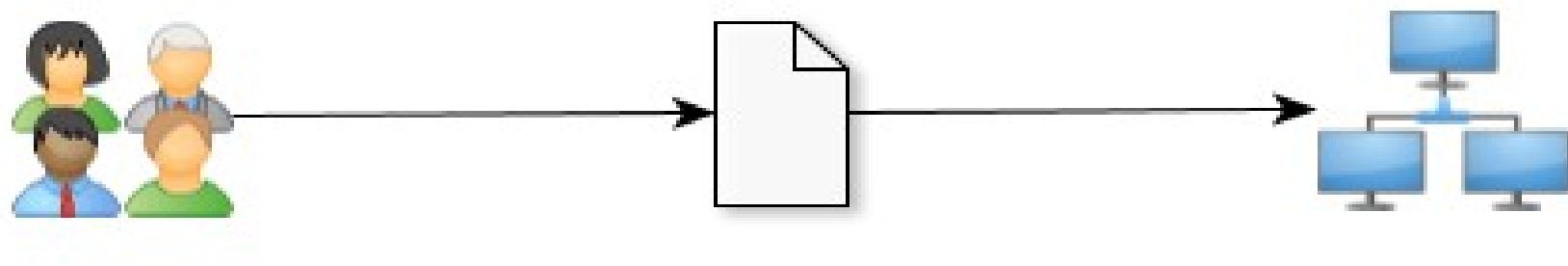
Provisionierungswerkzeug

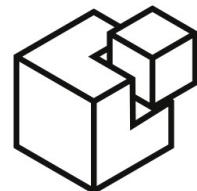
- automatisiert die Provisionierung eines Servers

Server-Provisionierung

- eine Menge an Schritten, um einen Server mit Daten und Software vorzubereiten
 - Resourcen zuweisen und konfigurieren
 - Middleware installieren und konfigurieren
 - Anwendungen installieren und konfigurieren

„Infrastructure As Code“





SALTSTACK



CHEF™



Puppet

CFEngine



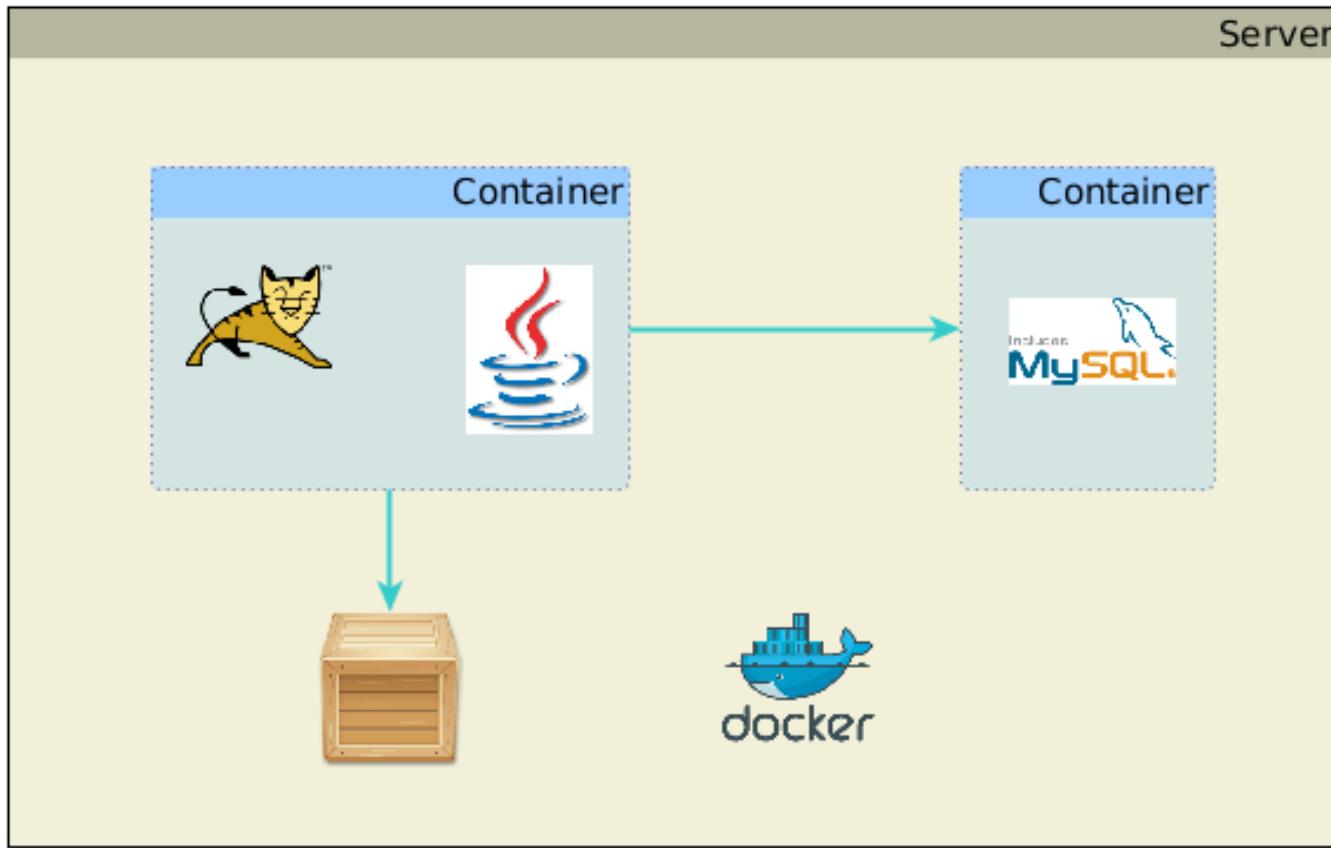
Ansible

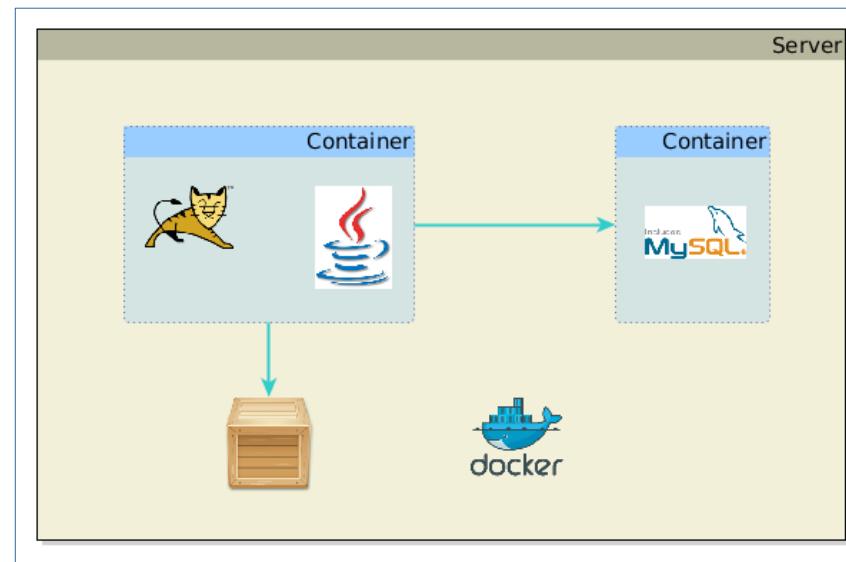
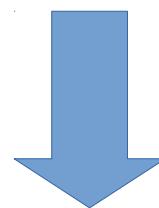
Und nun?

Status Quo



Wunsch – aber notwendig für Automatisierung?





Reiseroute

- Automatisierung der bestehenden Infrastruktur mit Ansible
- Wiederverwendung der Ansible Playbooks für die Docker-Image-Erstellung
- Automatisierung des Docker-Image-Lifecycles mit Ansible
- Verteilung der Docker-Container auf die Server mit Ansible

Erste Reiseetappe

Automatisierung der bestehenden Infrastruktur mit Ansible

Ausgangslage

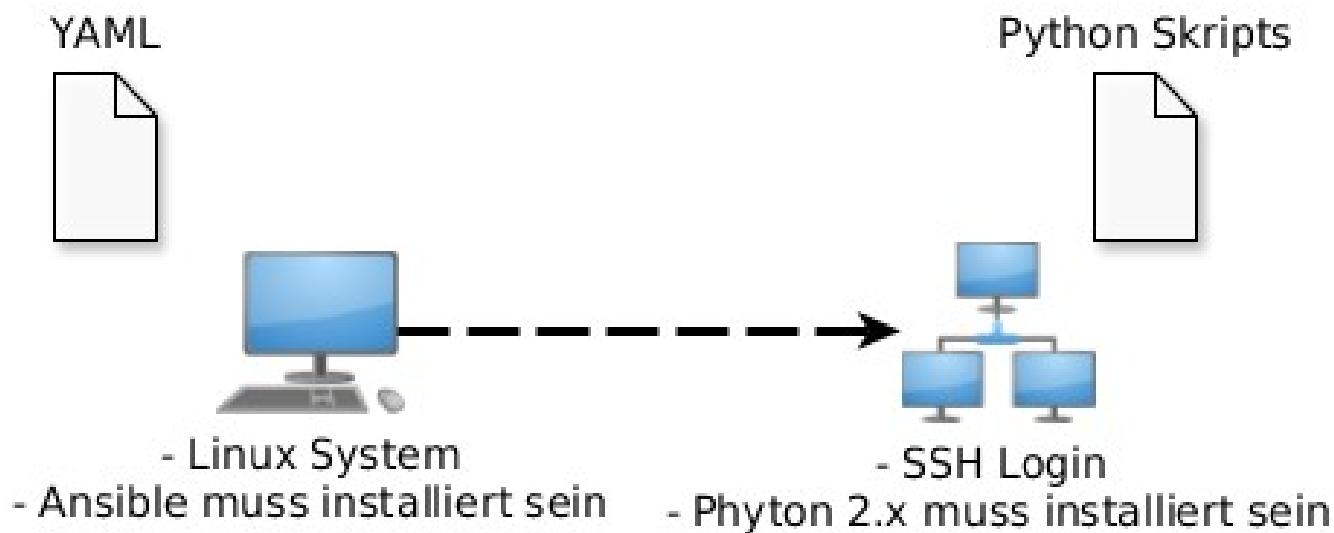


Ansible

- Provisionierungswerkzeug
- Sprache: Python
- Ansible Skripte (genannt *Playbooks*): YAML



Funktionsweise



Exkurs: YAML

YAML

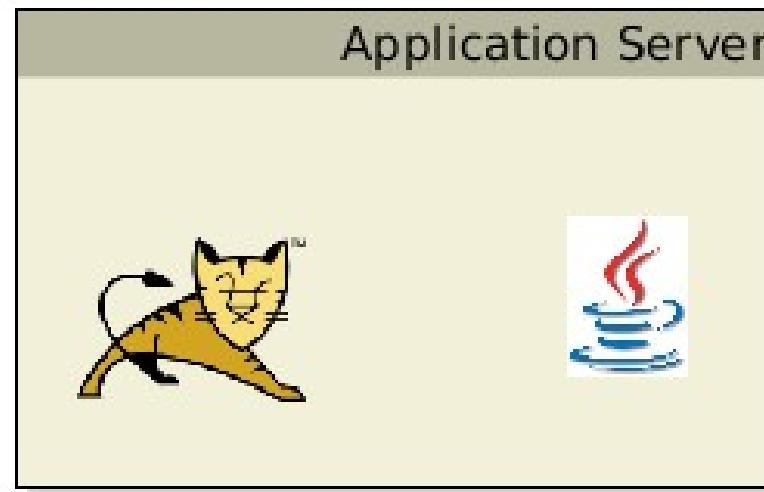
```
---
```

```
foo: "bar"
baz:
  - "qux"
  - "quxx"
corge: null
grault: 1
garply: true
waldo: "false"
fred: "undefined"
emptyArray: []
emptyObject: {}
emptyString: ""
```

JSON

```
{
  "foo": "bar",
  "baz": [
    "qux",
    "quxx"
  ],
  "corge": null,
  "grault": 1,
  "garply": true,
  "waldo": "false",
  "fred": "undefined",
  "emptyArray": [],
  "emptyObject": {},
  "emptyString": ""
}
```

Setup Application Server Playbook



```
1 - hosts: application-server
2   vars:
3     tomcat_version: 8.5.12
4     tomcat_base_name: apache-tomcat-{{ tomcat_version }}
5
6   tasks:
7     - name: ensure group vagrant exists
8       group: name=vagrant state=present
9
10    - name: ensure user vagrant exists
11      user: name=vagrant group=vagrant state=present
12
13    - name: install java
14      apt: name=openjdk-8-jdk state=present
15      become: yes
16      become_method: sudo
17
18    - name: Download current Tomcat 8 version
19      local_action: get_url url="http://archive.apache.org/dist/tomcat/tomcat-8/v{{ tomcat_version }}/bin/{{ tomcat_base_name }}.tar.gz" dest=/tmp
20
21    - name:
22      file: name=/opt mode=777
23      become: yes
24      become_method: sudo
25
26    - name: Install Tomcat 8
27      unarchive: src=/tmp/{{ tomcat_base_name }}.tar.gz dest=/opt creates=/opt/{{ tomcat_base_name }} owner=vagrant group=vagrant
28
29    - name: Set link to tomcat 8
30      file: src=/opt/{{ tomcat_base_name }} dest=/opt/tomcat state=link force=yes
31
32    - find: paths="/opt/{{ tomcat_base_name }}/bin" patterns="*.sh"
33      register: result
34
35    - name: ensure tomcat scripts are executable
36      file: name={{item.path}} mode=755
37      with_items: '{{ result.files }}'
38
39    - name: install tomcat as service
40      copy: src=tomcat.service dest=/etc/systemd/system/
41      become: yes
42      become_method: sudo
43
```

```
1 - hosts: application-server
2   vars:
3     tomcat_version: 8.5.12
4     tomcat_base_name: apache-tomcat-{{ tomcat_version }}
5
6   tasks:
7     - name: ensure group vagrant exists
8       group: name=vagrant state=present
9
10    - name: ensure user vagrant exists
11      user: name=vagrant group=vagrant state=present
12
13    - name: install java
14      apt: name=openjdk-8-jdk state=present
15      become: yes
16      become_method: sudo
17
```

Inventories

Production

```
1 [application-server]
2 192.168.33.10
3 ubuntu-server db_host=mysql01
4
5 [mysql-db-server]
6 mysql[01:10]
7
8 [oracle-db-server]
9 db-[a:f].oracle.company.com
10
11 [database-server:children]
12 mysql-db-server
13 oracle-db-server
14 |
15 [application-server:vars]
16 message="Welcome"
17
18 [database-server:vars]
19 message="Hello World!"
```

Test

```
1 [application-server]
2 192.168.33.10|
3
4 [database-server]
5 192.168.33.10
6
```

```
1 - hosts: application-server
2   vars:
3     tomcat_version: 8.5.12
4     tomcat_base_name: apache-tomcat-{{ tomcat_version }}
5
6   tasks:
7     - name: ensure group vagrant exists
8       group: name=vagrant state=present
9
10    - name: ensure user vagrant exists
11      user: name=vagrant group=vagrant state=present
12
13    - name: install java
14      apt: name=openjdk-8-jdk state=present
15      become: yes
16      become_method: sudo
17
```

Ansible Modules

Module Index

- [All Modules](#)
- [Cloud Modules](#)
- [Clustering Modules](#)
- [Commands Modules](#)
- [Crypto Modules](#)
- [Database Modules](#)
- [Files Modules](#)
- [Identity Modules](#)
- [Inventory Modules](#)
- [Messaging Modules](#)
- [Monitoring Modules](#)
- [Net Tools Modules](#)
- [Network Modules](#)
- [Notification Modules](#)
- [Packaging Modules](#)
- [Remote Management Modules](#)
- [Source Control Modules](#)
- [Storage Modules](#)
- [System Modules](#)
- [Utilities Modules](#)
- [Web Infrastructure Modules](#)
- [Windows Modules](#)

```
- name: Download current Tomcat 8 version
  local_action: get_url url="http://archive.apache.org/dist/tomcat/tomcat-8/v{{ tomcat_version }}/bin/{{ tomcat_base_name }}.tar.gz"

- name:
  file: name=/opt mode=777
  become: yes
  become_method: sudo

- name: Install Tomcat 8
  unarchive: src=/tmp/{{ tomcat_base_name }}.tar.gz dest=/opt/{{ tomcat_base_name }} creates=/opt/{{ tomcat_base_name }} owner=vagrant group=vagrant

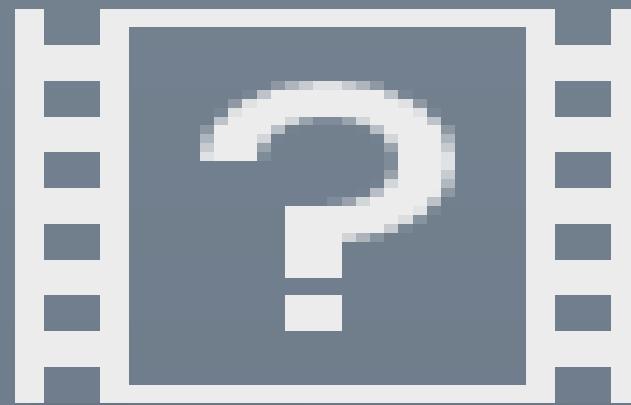
- name: Set link to tomcat 8
  file: src=/opt/{{ tomcat_base_name }} dest=/opt/tomcat state=link force=yes

- find: paths="/opt/{{ tomcat_base_name }}/bin" patterns="*.sh"
  register: result

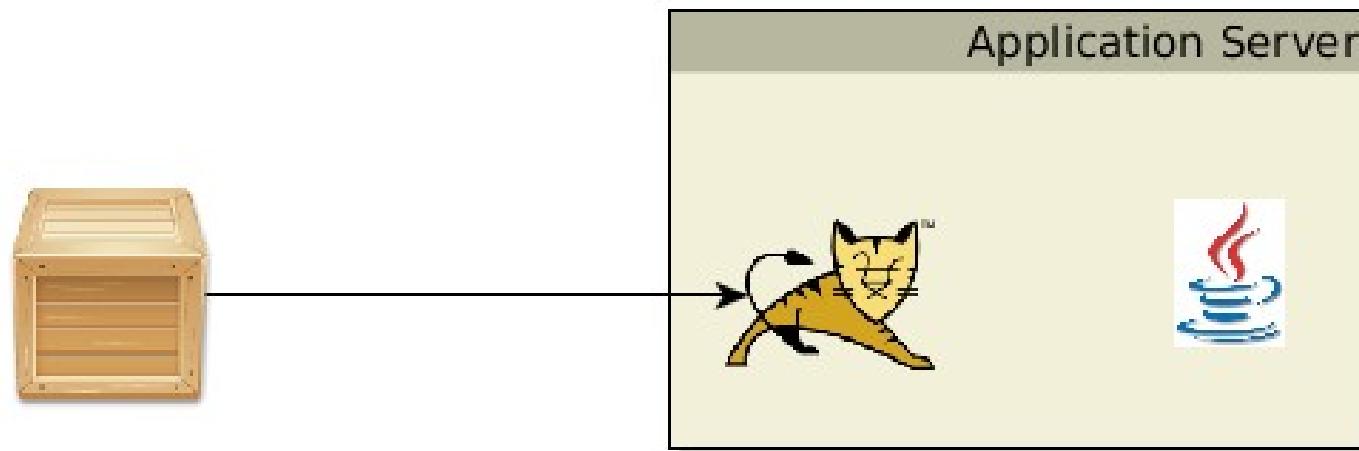
- name: ensure tomcat scripts are executable
  file: name={{item.path}} mode=755
  with_items: '{{ result.files }}'

- name: install tomcat as service
  copy: src=tomcat.service dest=/etc/systemd/system/
  become: yes
  become_method: sudo
```

Setup Application Server Playbook



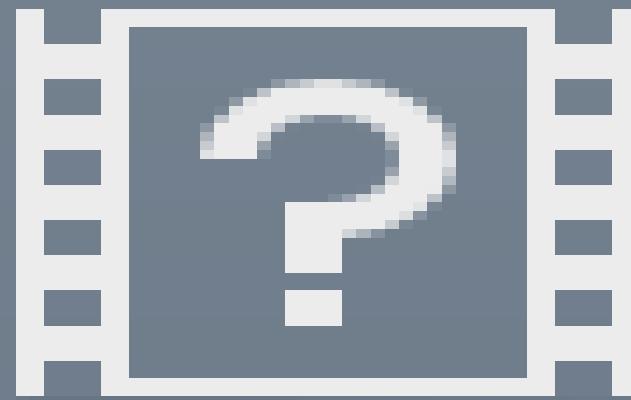
Java Webapplikation Deployment



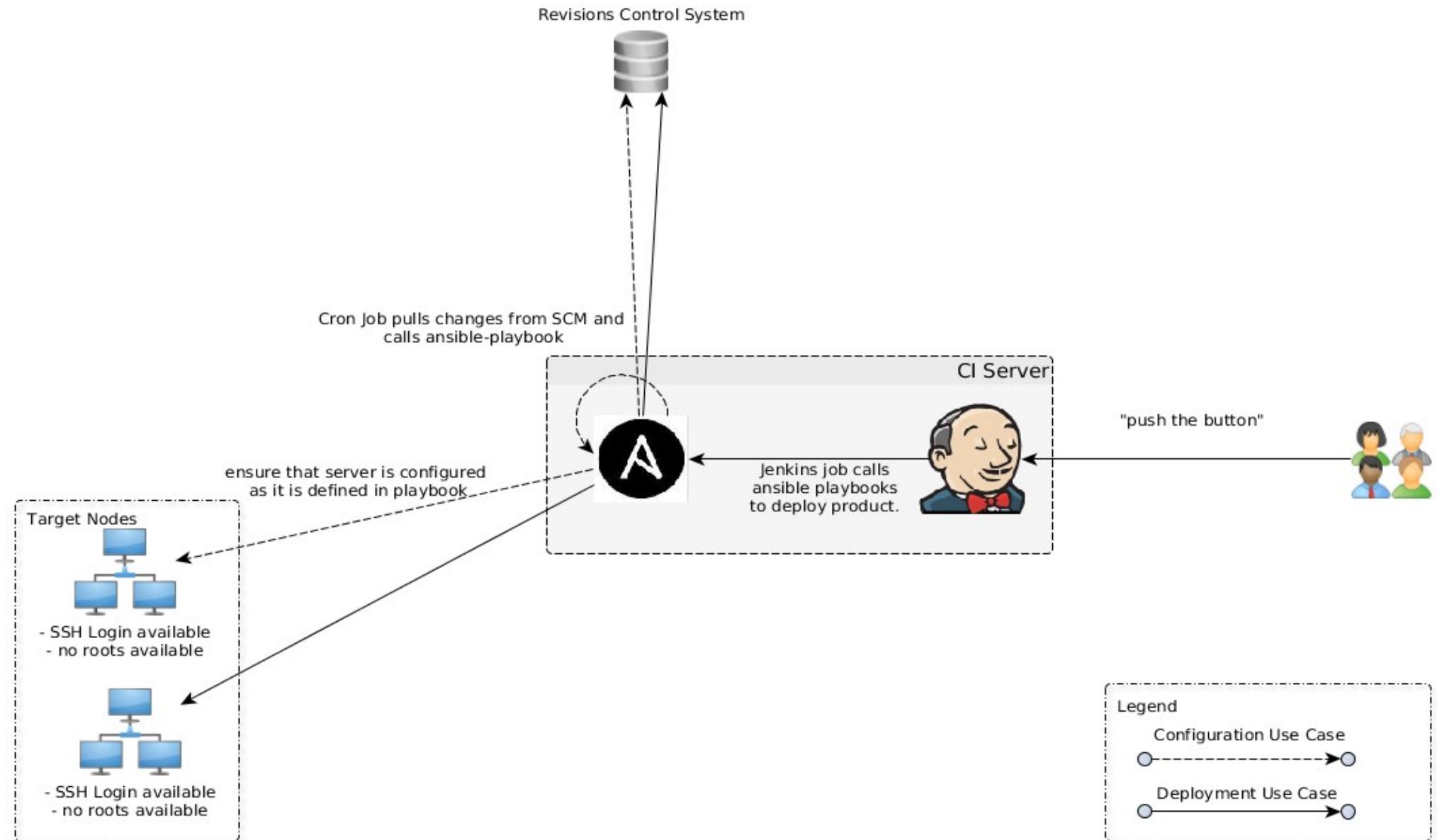
Deploy Application Playbook

```
1 - hosts: application-server
2 vars:
3   webapp_source_path: ./demo-web-application-1.war
4   webapp_target_name: demo
5   tomcat_app_base: /opt/tomcat/webapps
6 tasks:
7   - name: stop tomcat
8     service: name=tomcat state=stopped
9     become: true
10
11  - name: wait tomcat shutdown
12    wait_for: port=8080 state=stopped timeout=60
13
14  - name: cleanup {{ webapp_target_name }}
15    file: name={{tomcat_app_base}}/{{ webapp_target_name }} state=absent
16
17  - name: delete previous backup
18    file: path={{ tomcat_app_base }}/{{ webapp_target_name }}.war.previous state=absent
19
20  - name: create new backup
21    command: mv {{ tomcat_app_base }}/{{ webapp_target_name }}.war {{ tomcat_app_base }}/{{ webapp_target_name }}.war.previous ignore_errors: yes
22
23
24  - name: copy webapp {{ webapp_source_path }} to {{ webapp_target_name }}
25    copy: src={{ webapp_source_path }} dest={{ tomcat_app_base }}/{{ webapp_target_name }}.war mode=660
26
27  - name: start tomcat
28    service: name=tomcat enabled=yes state=started
29    become: true
30
31  - name: wait for tomcat to start
32    wait_for: port=8080 timeout=60
33
```

Deploy Application Playbook



Ansible Infrastruktur



Weitere Features

- Role
- Vault – Verschlüsselung
- Facts
- Dynamische Inventories
- Playbook Debugger
- Networking Support

Wie werden Ansible Skripte getestet?

- ansible-playbook --check
- ansible-playbook --syntax-check
- Jenkins + Vagrant
- Rspec tests



ServerSpec Tests

```
1 require 'spec_helper'
2
3 v describe package('openjdk-8-jdk') do
4   it { should be_installed }
5 end
6
7 v describe command('ls /etc/systemd/system/tomcat.service') do
8   its(:exit_status) { should eq 0 }
9 end
10
11 v describe command('ls /opt/tomcat') do
12   its(:exit_status) { should eq 0 }
13 end
14
```

ServerSpec Tests

```
plain-ansible : bash — Konsole
Datei Bearbeiten Ansicht Lesezeichen Einstellungen Hilfe
sparsick@sparsick-ThinkPad-T460s ~ /dev/NetBeansProjects/ansible-docker-talk/plain-ansible $ rake spec
/usr/bin/ruby2.3 -I/var/lib/gems/2.3.0/gems/rspec-support-3.6.0/lib:/var/lib/gems/2.3.0/gems/rspec-core-3.6.0/lib /var/lib/gems/2.3.0/gems/rspec-core-3.6.0/exe/rspec --pattern spec/ansible_demo/\*_spec.rb
...
Finished in 1.5 seconds (files took 6.24 seconds to load)
3 examples, 0 failures

sparsick@sparsick-ThinkPad-T460s ~ /dev/NetBeansProjects/ansible-docker-talk/plain-ansible $ rake spec
/usr/bin/ruby2.3 -I/var/lib/gems/2.3.0/gems/rspec-support-3.6.0/lib:/var/lib/gems/2.3.0/gems/rspec-core-3.6.0/lib /var/lib/gems/2.3.0/gems/rspec-core-3.6.0/exe/rspec --pattern spec/ansible_demo/\*_spec.rb
..F

Failures:

1) Command "ls /opt/tomcat1" exit_status should eq 0
On host `ansible_demo'
Failure/Error: its(:exit_status) { should eq 0 }

  expected: 0
  got: 2

  (compared using ==)
  sudo -p 'Password: ' /bin/sh -c ls\ /opt/tomcat1
# ./spec/ansible_demo/app_spec.rb:12:in `block (2 levels) in <top (required)>'

Finished in 1.54 seconds (files took 6.73 seconds to load)
3 examples, 1 failure

Failed examples:

rspec ./spec/ansible_demo/app_spec.rb:12 # Command "ls /opt/tomcat1" exit_status should eq 0
/usr/bin/ruby2.3 -I/var/lib/gems/2.3.0/gems/rspec-support-3.6.0/lib:/var/lib/gems/2.3.0/gems/rspec-core-3.6.0/lib /var/lib/gems/2.3.0/gems/rspec-core-3.6.0/exe/rspec --pattern spec/ansible_demo/\*_spec.rb failed
sparsick@sparsick-ThinkPad-T460s ~ /dev/NetBeansProjects/ansible-docker-talk/plain-ansible $ █
```

Reiseroute

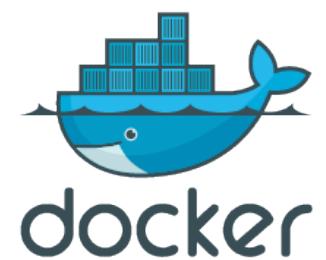
- Automatisierung der bestehenden Infrastruktur mit Ansible
- Wiederverwendung der Ansible Playbooks für die Docker-Image-Erstellung
- Automatisierung des Docker-Image-Lifecycles mit Ansible
- Verteilung der Docker-Container auf die Server mit Ansible

Zweite Reiseroute

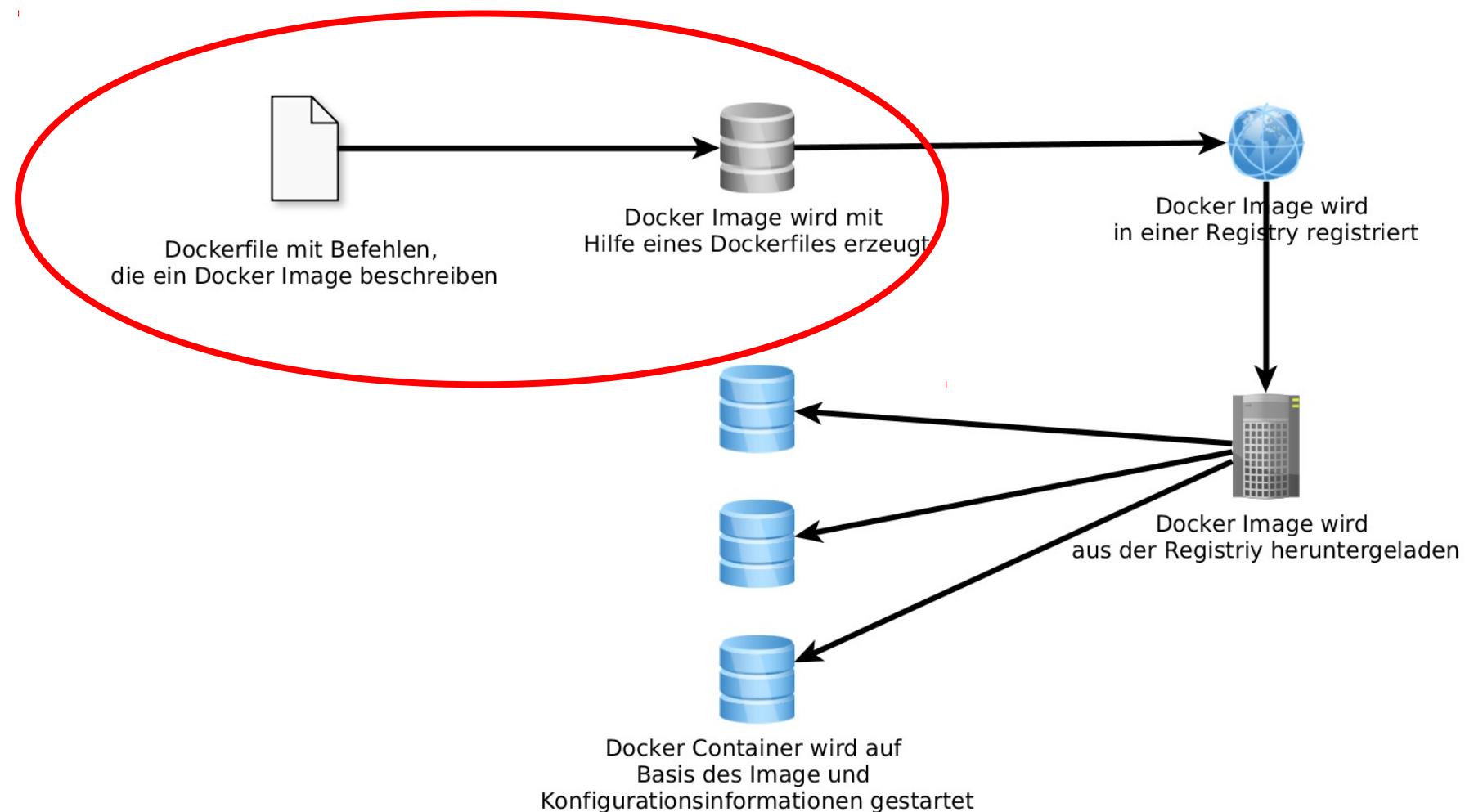
Wiederverwendung der Ansible Playbooks für die Docker-Image- Erstellung

Docker

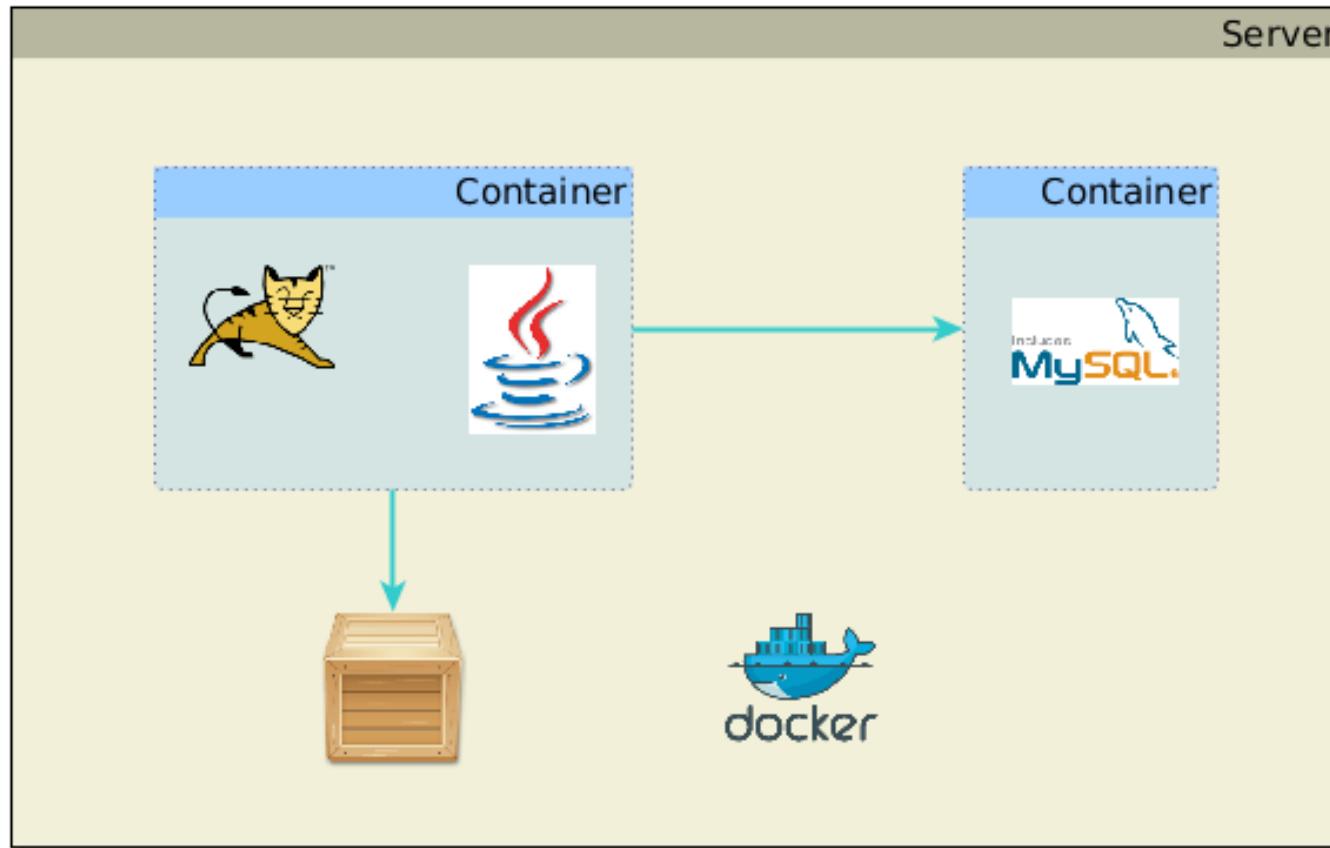
- Verwaltungswerkzeug für Container
- Weitere Werkzeuge aus dem Docker Universum (Auszug):
 - Docker Compose: Hilft beim Definieren und beim Laufen von Multi-Container Anwendungen
 - Docker Registry: Repository Manager für Docker Images



Docker Lifecycle



Reminder - Zielinfrastruktur



Was brauchen wir?

- Zwei Dockerfiles
 - Mysql.df
 - Tomcat.df
- WAR Datei wird über Volume eingebunden

Dockerfile ohne Ansible (tomcat.df)

```
1 FROM ubuntu:16.04
2
3 RUN apt-get update -y && \
4     apt-get install openjdk-8-jre curl -y && \
5     curl http://archive.apache.org/dist/tomcat/tomcat-8/v8.5.13/bin/apache-tomcat-8.5.13.tar.gz -o /opt/tomcat.tar.gz && \
6     tar -xf /opt/tomcat.tar.gz -C /opt && \
7     rm -f /opt/tomcat.tar.gz && \
8     ln -s /opt/apache-tomcat-8.5.13 /opt/tomcat && \
9     apt-get remove curl -y
10
11 ENV CATALINA_HOME /opt/tomcat
12 ENV PATH $CATALINA_HOME/bin:$PATH
13 WORKDIR $CATALINA_HOME
14
15 ENTRYPOINT [ "catalina.sh"]
16 CMD ["run"]
17 EXPOSE 8080
18
```

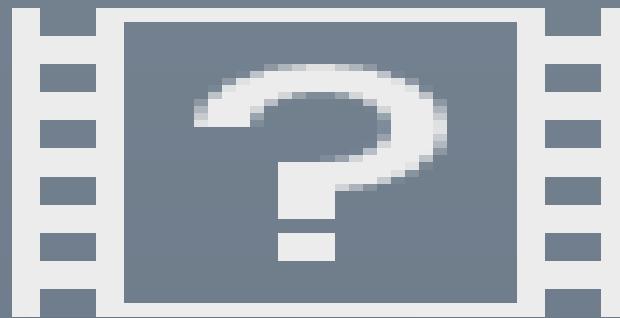
Dockerfile mit Ansible (tomcat.df)

```
1 FROM williamyeh/ansible:ubuntu16.04
2
3 WORKDIR /tmp
4 COPY plain-ansible/* /tmp/
5
6 RUN echo application-server > inventory
7 RUN ansible-playbook -i inventory setup-app.yml --connection=local
8
9 ENV CATALINA_HOME /opt/tomcat
10 ENV PATH $CATALINA_HOME/bin:$PATH
11 WORKDIR $CATALINA_HOME
12
13 ENTRYPOINT [ "catalina.sh"]
14 CMD [ "run"]
15 EXPOSE 8080
16
```

Dockerfile mit Ansible (mysql.df)

```
1 FROM williamyeh/ansible:ubuntu16.04
2
3 WORKDIR /tmp
4 COPY plain-ansible/* /tmp/
5
6 RUN echo database-server > inventory
7 RUN ansible-playbook -i inventory setup-db.yml --connection=local
8
9 USER mysql
10 ENTRYPOINT ["/usr/bin/mysqld_safe"]
11 EXPOSE 3306
12
```

Dockerfile mit Ansible Demo



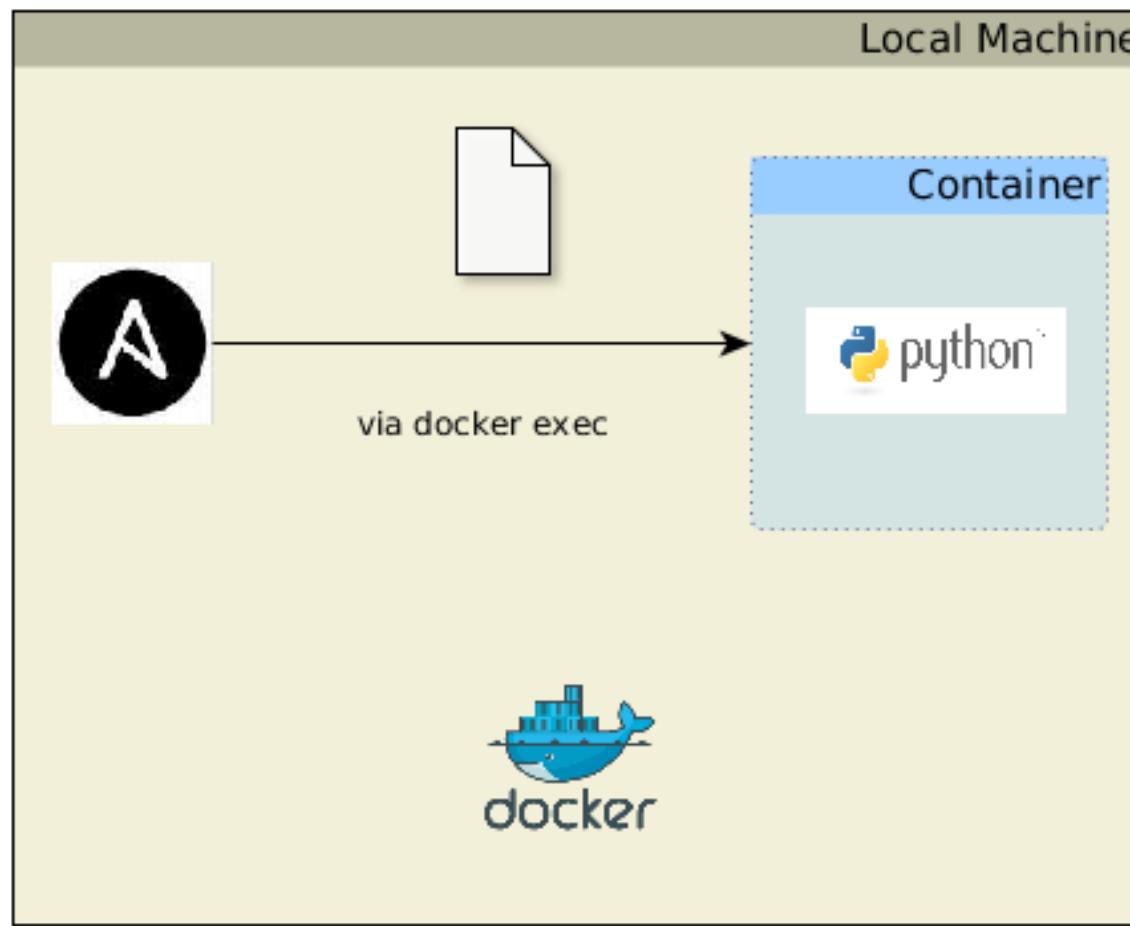
Dockerfile mit Ansible - Pros und Cons

- ✓ Vorhandene Skripte können wiederverwendet werden
 - ✗ Es werden Abhängigkeiten in das Image mit gepackt, die die Anwendung nicht braucht
 - Image wird unnötig groß
 - ✗ Alternative: Ansible + Python (De)-Installation als RUN Schritt
 - Image-Build-Dauer erhöht sich
- ✓ Reicht aus um generell seine Anwendung im Docker Container zu testen

Ansible Connection Type: Docker

- Neuer Connection Type seit Ansible 2.0
- Funktionsweise:
 - Ansible führt die Playbooks über docker exec in einem laufendem Container aus
 - Bedingung: Im Container muss Python installiert sein

Ansible Connection Type: Docker



Ansible Connection Type: Docker

```
docker run --name database --rm -td sparsick/python:plain
ansible-playbook -i inventories/docker -u root setup-db.yml
docker commit database sparsick/mysql:ansibledocker
docker stop database|
```

```
1 # inventories/docker
2
3 [database-server]
4 database ansible_connection=docker
5
6 [application-server]
7 tomcat ansible_connection=docker
8
```

Ansible Connection Type: Docker - Pros und Cons

- ✓ Vorhandene Skripte können wiederverwendet werden
 - ✗ Base Image muss Python Installation haben
 - ✗ Mehr Schritte erforderlich
- ✓ Reicht aus um generell seine Anwendung im Docker Container zu testen
- ✓ Keine Dockerfiles

Alternativen

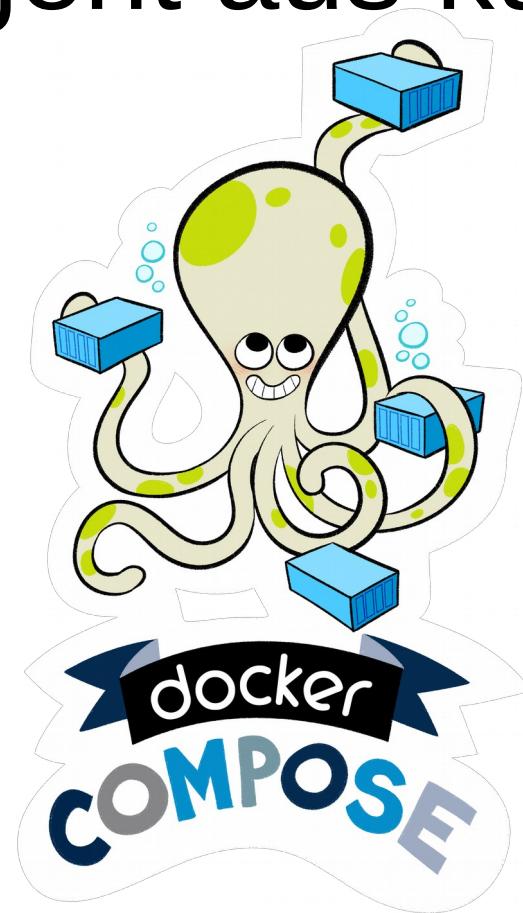
- Template
- Ansible-Container
- Packer
- Rocker
- Etc.
- Überblick verschafft Talk „Docker Container Loading“ von Roland Huß

Docker Container Lifecycle (lokal)

```
docker build -f tomcat.df -t sparsick/tomcat:ansible .
docker build -f mysql.df -t sparsick/mysql:ansible .

docker run -it --name database -d --rm sparsick/mysql:ansible
docker run -it --name tomcat --rm --link database -v $(pwd)/
plain-ansible/demo-web-application-1.war:/opt/tomcat/webapps/
demo.war -p 8080:8080 sparsick/tomcat:ansible
```

Es geht aus kürzer

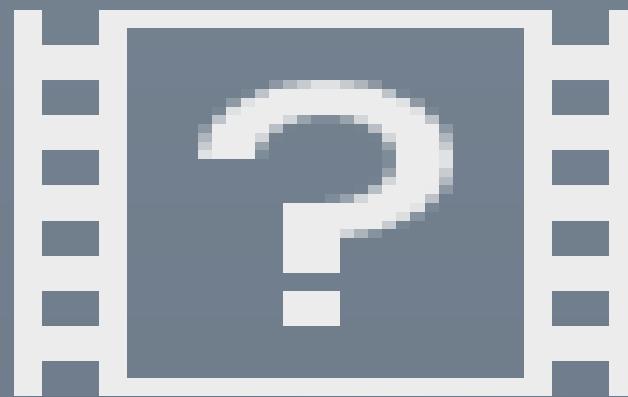


Docker Compose



```
1 version: '3'  
2 services:  
3   database:  
4     image: "sparsick/mysql:ansible"  
5   tomcat:  
6     image: sparsick/tomcat:ansible  
7     ports:  
8       - "8080:8080"  
9     links:  
10    - database  
11  volumes:  
12    - ./plain-ansible/demo-web-application-1.war:/opt/tomcat/webapps/demo.war  
13
```

Docker Compose Demo



Docker Compose



```
1 version: '3'  
2 services:  
3   database:  
4     build:  
5       context: .  
6       dockerfile: mysql.df  
7   tomcat:  
8     build:  
9       context: .  
10      dockerfile: tomcat.df  
11     ports:  
12       - "8080:8080"  
13     links:  
14       - database  
15     volumes:  
16       - ./plain-ansible/demo-web-application-1.war:/opt/tomcat/webapps/demo.war
```

Docker Compose – Ansible Connection Type Docker



```
1 version: '3'  
2 services:  
3   database:  
4     image: "sparsick/mysql:ansibledocker"  
5     entrypoint: "/usr/bin/mysqld_safe"  
6     expose:  
7       - 3306  
8   tomcat:  
9     image: sparsick/tomcat:ansibledocker  
10    entrypoint: "/opt/tomcat/bin/catalina.sh"  
11    command: run  
12    working_dir: "/opt/tomcat"  
13    ports:  
14      - "8080:8080"  
15    links:  
16      - database  
17    volumes:  
18      - ./plain-ansible/demo-web-application-1.war:/opt/tomcat/webapps/demo.war
```

Reminder: Dockerfile



```
1 FROM williamyeh/ansible:ubuntu16.04
2
3 WORKDIR /tmp
4 COPY plain-ansible/* /tmp/
5
6 RUN echo database-server > inventory
7 RUN ansible-playbook -i inventory setup-db.yml --connection=local
8
9 USER mysql
10 ENTRYPOINT ["/usr/bin/mysqld_safe"]
11 EXPOSE 3306
```

```
1 FROM williamyeh/ansible:ubuntu16.04
2
3 WORKDIR /tmp
4 COPY plain-ansible/* /tmp/
5
6 RUN echo application-server > inventory
7 RUN ansible-playbook -i inventory setup-app.yml --connection=local
8
9 ENV CATALINA_HOME /opt/tomcat
10 ENV PATH $CATALINA_HOME/bin:$PATH
11 WORKDIR $CATALINA_HOME
12
13 ENTRYPOINT [ "catalina.sh"]
14 CMD [ "run"]
15 EXPOSE 8080
16
```

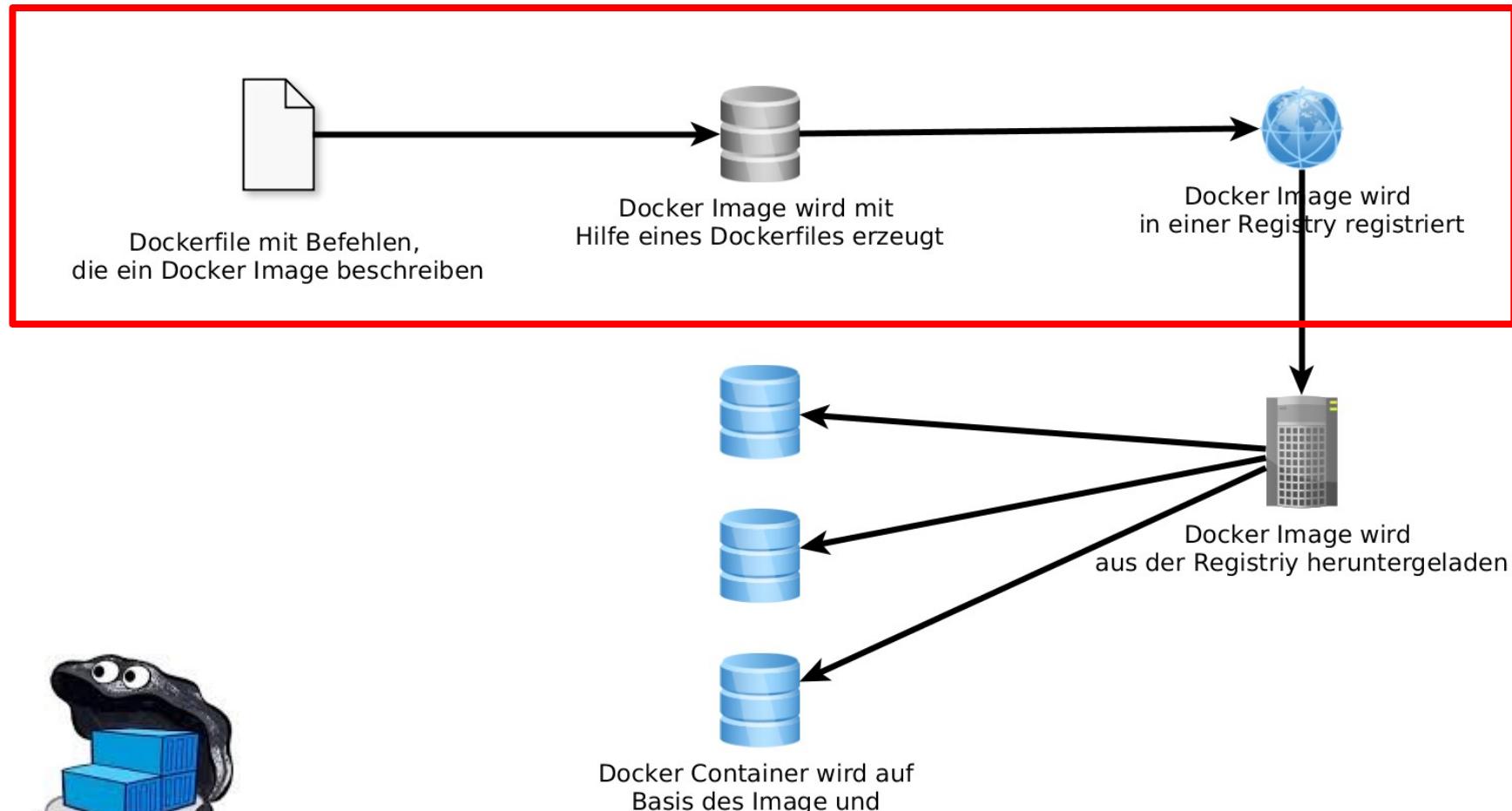
Reiseroute

- Automatisierung der bestehenden Infrastruktur mit Ansible
- Wiederverwendung der Ansible Playbooks für die Docker-Image-Erstellung
- Automatisierung des Docker-Image-Lifecycles mit Ansible
- Verteilung der Docker-Container auf die Server mit Ansible

Dritte Reiseroute

Automatisierung des Docker-Image-Lifecycles mit Ansible

Docker Lifecycle





Docker Registry Lifecycle

```
docker run --rm -d -p 5000:5000 --name registry registry:2
docker build -f tomcat.df -t sparsick/tomcat:ansible .
docker build -f mysql.df -t sparsick/mysql:ansible .

docker tag sparsick/mysql:ansible localhost:5000/sparsick/
mysql:plain
docker push localhost:5000/sparsick/mysql:plain

docker tag sparsick/tomcat:ansible localhost:5000/sparsick/
tomcat:plain
docker push localhost:5000/sparsick/tomcat:plain

# testing
curl -s http://localhost:5000/v2/sparsick/tomcat/tags/list
curl -s http://localhost:5000/v2/sparsick/mysql/tags/list

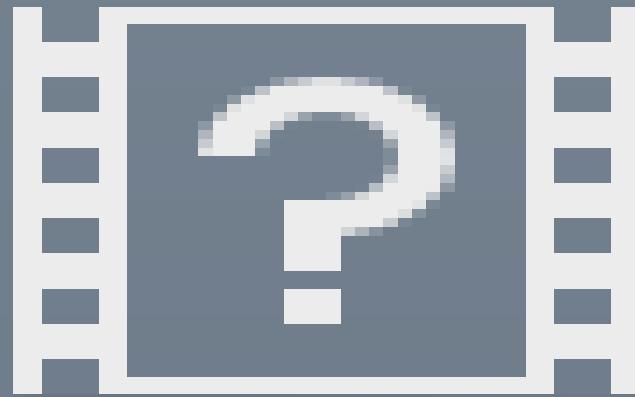
docker stop registry
```

Docker Registry Lifecycle mit Ansible

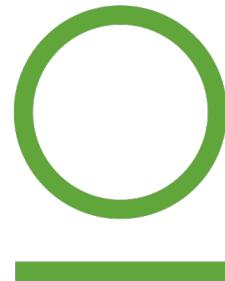
```
5   - name: build and push tomcat image
6     docker_image:
7       name: sparsick/tomcat
8       dockerfile: tomcat.df
9       path: "{{ playbook_dir }}/../docker-image-ansible"
10      push: yes
11      repository: "{{ registry_hostname }}:5000/sparsick/tomcat:ansible"
```



Docker Registry Lifecycle mit Ansible Demo



Docker Registry Alternativen



JFrog Artifactory



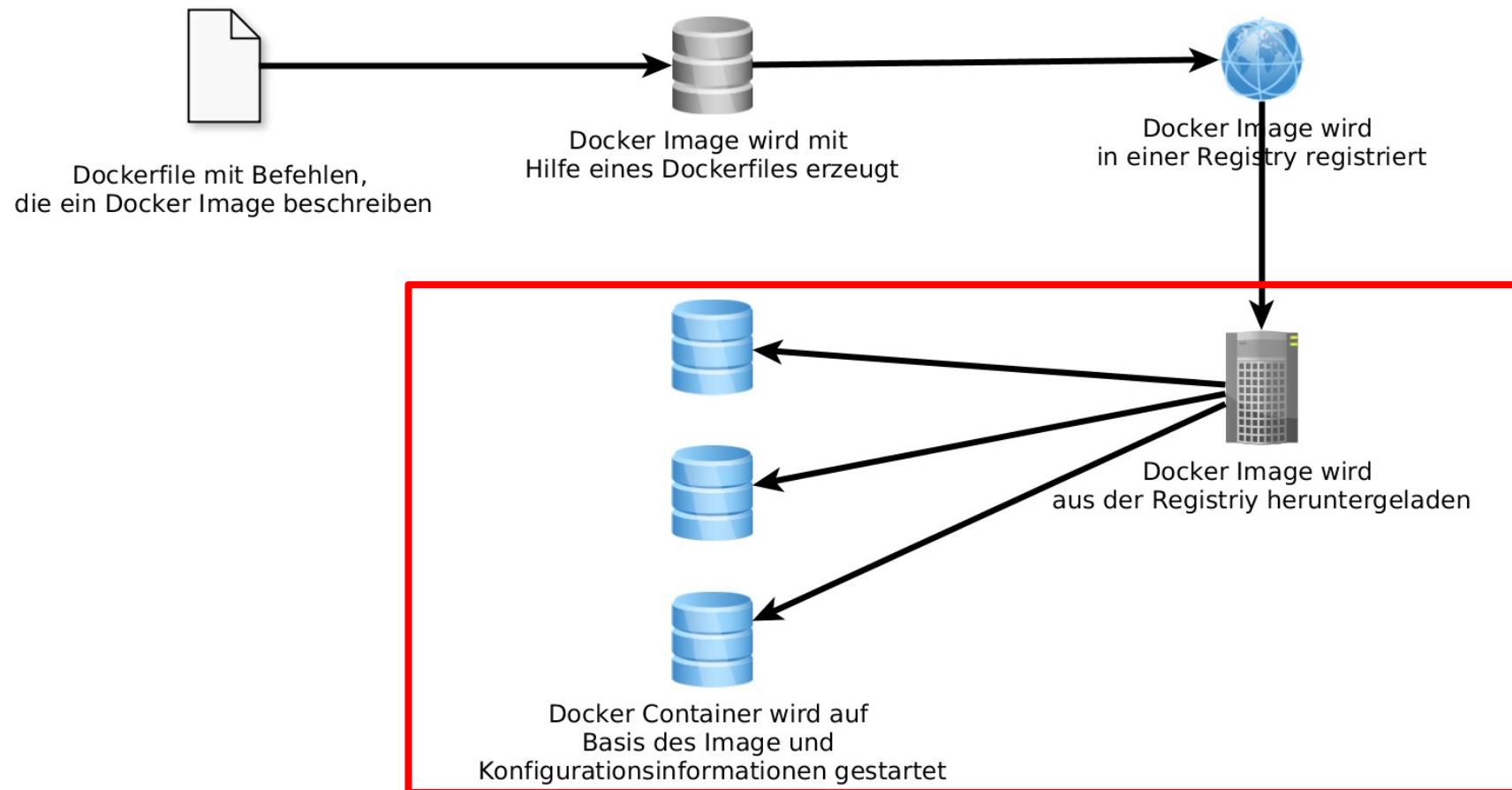
Reiseroute

- Automatisierung der bestehenden Infrastruktur mit Ansible
- Wiederverwendung der Ansible Playbooks für die Docker-Image-Erstellung
- Automatisierung des Docker-Image-Lifecycles mit Ansible
- Verteilung der Docker-Container auf die Server mit Ansible

Letzte Reiseroute

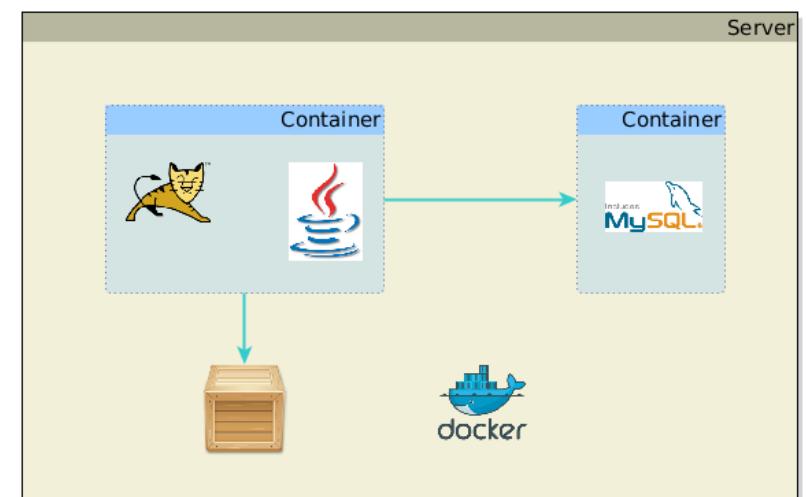
Verteilung der Docker-Container auf
die Server mit Ansible

Docker Lifecycle



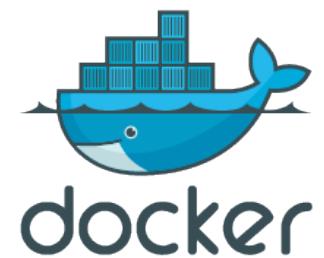
Was brauchen wir?

- Docker Installation automatisieren
- Docker Registry Installation automatisieren
- Container auf das Zielsystem verteilen
- WAR Datei wird über Volume eingebunden
→ muss auf den Zielsystem kopiert werden



Setup-dockerd.yml (Ausschnitt)

```
12 - name: install required package
13   apt: name="{{ item }}" state=present
14   with_items:
15     - apt-transport-https
16     - ca-certificates
17     - software-properties-common
18     - python-docker
19     - python-pip
20
21 - name: add Docker's official GPG key
22   apt_key: url="http://apt.dockerproject.org/gpg" state=present
23
24 - name: add Docker's repository
25   apt_repository: repo="deb https://apt.dockerproject.org/repo/ ubuntu-xenial main" update_cache=yes
26
27 - name: install latest Docker Engine
28   apt: name="docker-engine"
29
30 - name: add user vagrant to group docker
31   user: name=vagrant groups=docker state=present
32
33 - name: install docker compose
34   pip: name=docker-compose
35
```



Setup-docker-registry.yml (Ausschnitt)

```
11
12     - name: start docker registry
13       docker_container:
14         name: registry
15         image: registry:2
16         state: started
17         ports:
18           - "5000:5000"
```



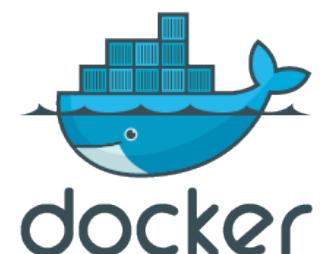
Build-and-push-images.yml (Ausschnitt)

```
5  - name: build and push tomcat image
6    docker_image:
7      name: sparsick/tomcat
8      dockerfile: tomcat.df
9      path: "{{ playbook_dir }}/../docker-image-ansible"
10     push: yes
11     repository: "{{ registry_hostname }}:5000/sparsick/tomcat:ansible"
12
13 - name: build and push mysql image
14   docker_image:
15     name: sparsick/mysql
16     dockerfile: mysql.df
17     path: "{{ playbook_dir }}/../docker-image-ansible"
18     push: yes
19     repository: "{{ registry_hostname }}:5000/sparsick/mysql:ansible"
```



Deploy-docker-container.yml (Ausschnitt)

```
4 - name: start database container
5   docker_container:
6     name: database
7     image: localhost:5000/sparsick/mysql:ansible
8     state: started
9
18 - name: copy java webapp archive
19   copy: src="./demo-web-application-1.war" dest="/opt/webapps/demo-web-application-1.war" owner=vagrant group=vagrant
20
21 - name: start tomcat container
22   docker_container:
23     name: tomcat
24     image: localhost:5000/sparsick/tomcat:ansible
25     state: started
26     volumes:
27       - "/opt/webapps/demo-web-application-1.war:/opt/tomcat/webapps/demo.war"
28     links:
29       - "database"
30     ports:
31       - "8080:8080"
32
```



Alternative Ansible mit Docker Compose Syntax

Docker-compose muss auf dem Zielmaschine installiert werden

```
12 - name: copy java webapp archive
13   copy: src="../demo-web-application-1.war" dest="/opt/webapps/demo-web-application-1.war" owner=vagrant group=vagrant
14
15 - docker_service:
16   project_name: java_web
17   definition:
18     version: '2'
19   services:
20     database:
21       image: "localhost:5000/sparsick/mysql:ansible"
22     tomcat:
23       image: localhost:5000/sparsick/tomcat:ansible
24       ports:
25         - "8080:8080"
26     links:
27       - database
28     volumes:
29       - /opt/webapps/demo-web-application-1.war:/opt/tomcat/webapps/demo.war
30
```



Ansible Playbook Call

```
ansible-playbook -i inventories/test -u vagrant \
setup-dockerd.yml \
setup-docker-registry.yml \
build-and-push-images.yml \
deploy-docker-container.yml \
--extra-vars "registry_hostname=192.168.33.11"
```

Reiseroute

- Automatisierung der bestehenden Infrastruktur mit Ansible
- Wiederverwendung der Ansible Playbooks für die Docker-Image-Erstellung
- Automatisierung des Docker-Image-Lifecycles mit Ansible
- Verteilung der Docker-Container auf die Server mit Ansible





Ich bin von Kopf bis Fuß auf
„Docker“ eingestellt

(frei nach Marlene Dietrich)

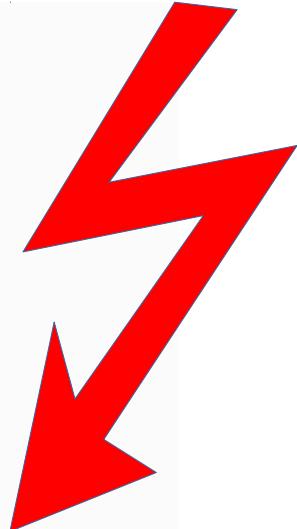
Checkliste

- Verschlankung der Docker Image
 - Brauche ich wirklich ein kompletten Tomcat?
 - Nicht besser ein Fat Jar?
 - Spring Boot
 - Dropwizard
- Docker Image resultiert aus dem Build
- Ausweitung auf Produktionsumgebungen
- Anpassung der Implementierung
- Nicht gleichzeitig Kubernetes einführen, sondern mit einfacheren Orchestrierungswerkzeuge anfangen

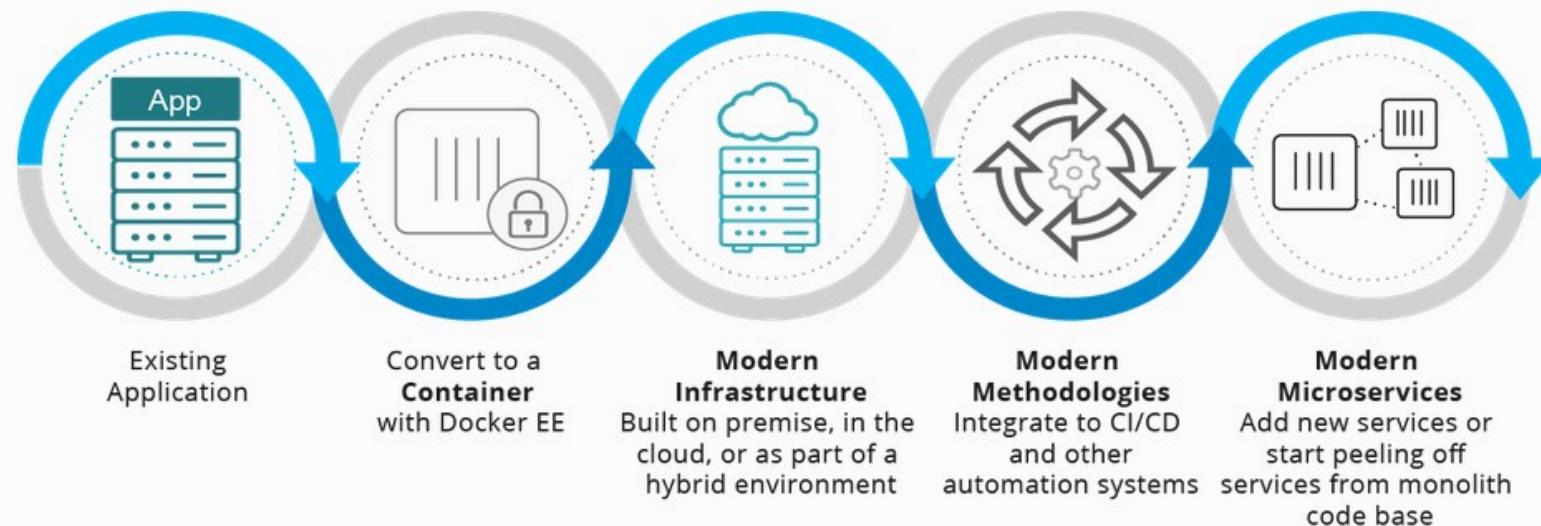


Modernize Traditional Applications with Docker Enterprise Edition

Bring Portability, Security, and Efficiency to Your Traditional Applications Without Changing Application Code



The Modernize Traditional Applications program from Docker is a short-term cooperative engagement between Docker and industry leading partners that will assist you with transforming expensive and difficult to maintain existing apps into efficient, secure, and portable modern apps ready for the hybrid cloud.



Docker und Java

- Vor Java 10:
 - JVM erkennt nicht, wie viel Memory ihr im Container zur Verfügung stehen
- Grund:
 - JVM erhält ihre System Resource Information nicht von cgroups, sondern von Linux Werkzeugen, die länger als cgroups existieren (Beispiel: /proc/meminfo, /proc/vmstat)
 - Dasselbe Problem haben auch top, free, ps

Docker und Java

```
$> java -jar memory-consumer-1.0-SNAPSHOT.jar
```

```
Initial free memory: 4332MB
```

```
Max memory: 4336MB
```

```
Reserve: 2047MB
```

```
Free memory: 2288MB
```

```
$> docker run sparsick/docker-java8-demo
```

```
Initial free memory: 4334MB
```

```
Max memory: 4336MB
```

```
Reserve: 2047MB
```

```
Free memory: 2287MB
```

```
$> docker run -m256M sparsick/docker-java8-demo
```

```
Initial free memory: 4334MB
```

```
Max memory: 4336MB
```

```
Reserve: 2047MB
```

```
Killed
```

```
$> docker run -m256M -e JAVA_OPT=' -Xms64M -Xmx256M' sparsick/docker-java8-demo
```

```
Initial free memory: 227MB
```

```
Max memory: 228MB
```

```
Reserve: 181MB
```

```
Free memory: 48MB
```

Docker und Java

```
161
| 162 max_memory() {
163     # High number which is the max limit until which memory is supposed to be
164     # unbounded.
165     local mem_file="/sys/fs/cgroup/memory/memory.limit_in_bytes"
166     if [ -r "${mem_file}" ]; then
167         local max_mem_cgroup="$(cat ${mem_file})"
168         local max_mem_meminfo_kb="$(cat /proc/meminfo | awk '/MemTotal/ {print $2}')"
169         local max_mem_meminfo="$(expr $max_mem_meminfo_kb \* 1024)"
170         if [ ${max_mem_cgroup:-0} != -1 ] && [ ${max_mem_cgroup:-0} -lt ${max_mem_meminfo:-0} ]
171             then
172                 echo "${max_mem_cgroup}"
173             fi
174         fi
175     }
176
```

Aus: Fabric8 Java Base Image OpenJDK 8 (JDK)

Docker und Java

```
#For JDK 8u131+ and JDK 9
$> docker run -m256M -e JAVA_OPT='-XX:+UnlockExperimentalVMOptions -XX:+UseCGroupMemoryLimitForHeap -XX:MaxRAMFraction=1' sparsick/docker-java8-demo
Initial free memory: 227MB
Max memory: 228MB
Reserve: 182MB
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
    at MemoryConsumer.main(MemoryConsumer.java:27)

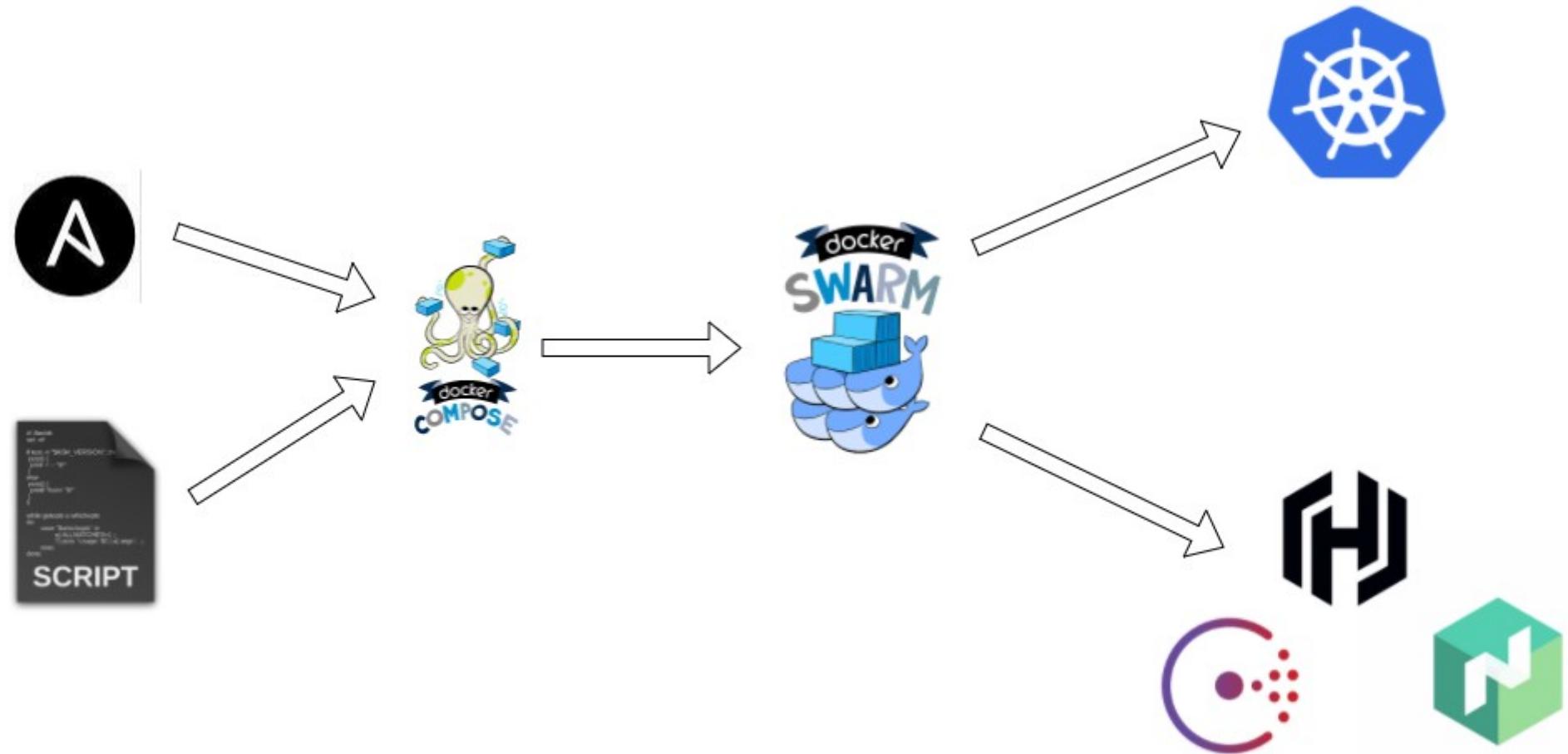
$> docker run -m256M -e JAVA_OPT='-XX:+UnlockExperimentalVMOptions -XX:+UseCGroupMemoryLimitForHeap -XX:MaxRAMFraction=1 -Xms64M' sparsick/docker-java8-demo
Initial free memory: 227MB
Max memory: 228MB
Reserve: 181MB
Free memory: 50MB
```

Docker und Java

```
$> docker run -m256M sparsick/docker-java10-demo
Initial free memory: 120MB
Max memory: 121MB
Reserve: 96MB
Free memory: 24MB
```

Und was ist mit Kubernetes?

Einfach anfangen (KISS)



Wann Container einsetzen?

Mögliche Einsatzszenarien für Container

- Anwendung muss automatisch skaliert werden
- Unternehmen hat unterschiedliche Technologiestacks und will die Verteilung vereinheitlichen
- Lokale Testumgebung
- Bei Integrationstests innerhalb eines Builds (Testcontainer)

Integrationstests mit Container



Migrationstests von DB Skripte

```
public class DbMigrationITest {  
  
    @Rule  
    public MySQLContainer mysqlDb = new MySQLContainer();  
  
    @Test  
    public void testDbMigrationFromTheScratch(){  
        Flyway flyway = new Flyway();  
        flyway.setDataSource(mysqlDb.getJdbcUrl(), mysqlDb.getUsername(), mysqlDb.getPassword())  
  
        flyway.migrate();  
    }  
}
```

Migrationstests von DB Skripte

TESTS

```
Running db.migration.DbMigrationITest
INFO - eryClientProviderStrategy - Found docker client settings from environment
INFO - ckerClientProviderStrategy - Found Docker environment with Environment variables, system properties and defaults. Resolved:
  dockerHost=unix:///var/run/docker.sock
  apiVersion='{UNKNOWN_VERSION}'
  registryUrl='https://index.docker.io/v1/'
  registryUsername='sparsick'
  registryPassword='null'
  registryEmail='null'
  dockerConfig='DefaultDockerClientConfig[dockerHost=unix:///var/run/docker.sock,registryUsername=sparsick,registryPassword=<null>,registryEmail=<null>]'

INFO - DockerClientFactory      - Docker host IP address is localhost
INFO - DockerClientFactory      - Connected to docker:
  Server Version: 17.05.0-ce
  API Version: 1.29
  Operating System: Linux Mint 18.2
  Total Memory: 19511 MB
    i Checking the system...
    ✓ Docker version is newer than 1.6.0
    ✓ Docker environment has more than 2GB free
    ✓ File should be mountable
    ✓ Exposed port is accessible
INFO - [mysql:latest]           - Creating container for image: mysql:latest
INFO - [mysql:latest]           - Starting container with ID: 2668be66c2631e49b5bcb4e180665d223525ec896ea78034326076d5f9063d53
INFO - [mysql:latest]           - Container mysql:latest is starting: 2668be66c2631e49b5bcb4e180665d223525ec896ea78034326076d5f9063d53
INFO - [mysql:latest]           - Waiting for database connection to become available at jdbc:mysql://localhost:32769/test using query 'SELECT 1'
INFO - [mysql:latest]           - Obtained a connection to container (jdbc:mysql://localhost:32769/test)
INFO - [mysql:latest]           - Container mysql:latest started
INFO - VersionPrinter          - Flyway 4.0.3 by Boxfuse
INFO - DbSupportFactory         - Database: jdbc:mysql://localhost:32769/test (MySQL 5.7)
INFO - DbValidate               - Successfully validated 2 migrations (execution time 00:00.011s)
INFO - MetaDataTableImpl        - Creating Metadata table: `test`.`schema_version`
INFO - DbMigrate                - Current version of schema `test`: <> Empty Schema <>
INFO - DbMigrate                - Migrating schema `test` to version 1.0.0 - create person table
INFO - DbMigrate                - Migrating schema `test` to version 2.0.0 - add column job title
INFO - DbMigrate                - Successfully applied 2 migrations to schema `test` (execution time 00:00.133s).
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 13.9 sec
```

Integrationstest für die Persistenzschicht

```
public class PersonRepositoryITest {

    @Rule
    public MySQLContainer mysqlDb = new MySQLContainer();

    @Test
    public void saveAndLoadAPerson() {
        Flyway flyway = new Flyway();
        flyway.setDataSource(mysqlDb.getJdbcUrl(), mysqlDb.getUsername(), mysqlDb.getPassword());
        flyway.migrate();

        PersonRepository personRepositoryUnderTest = new PersonRepository(flyway.getDataSource());
        Person person = new Person("Alice", "Bob");
        personRepositoryUnderTest.save(person);

        List<Person> persons = personRepositoryUnderTest.findAllPersons();

        assertThat(persons.size(), Is.is(1));
        assertThat(persons.get(0), Is.is(person));
    }
}
```

Testcontainers

- Temporary database containers - spezielle MySQL, PostgreSQL, Oracle XE und Virtuoso container
- Webdriver containers - Dockerized Chrome oder Firefox browser für Selenium/Webdriver Operationen mit automatischer Videoaufnahme
- Kafka container – Dockerized Kafka (einzelner Knoten)
- Generic containers – irgendein Docker Container
- Docker compose – Wiederverwendung von Docker Compose YAML Datei
- Dockerfile containers – Container direkt von einem Dockerfile

Was haben wir gesehen?

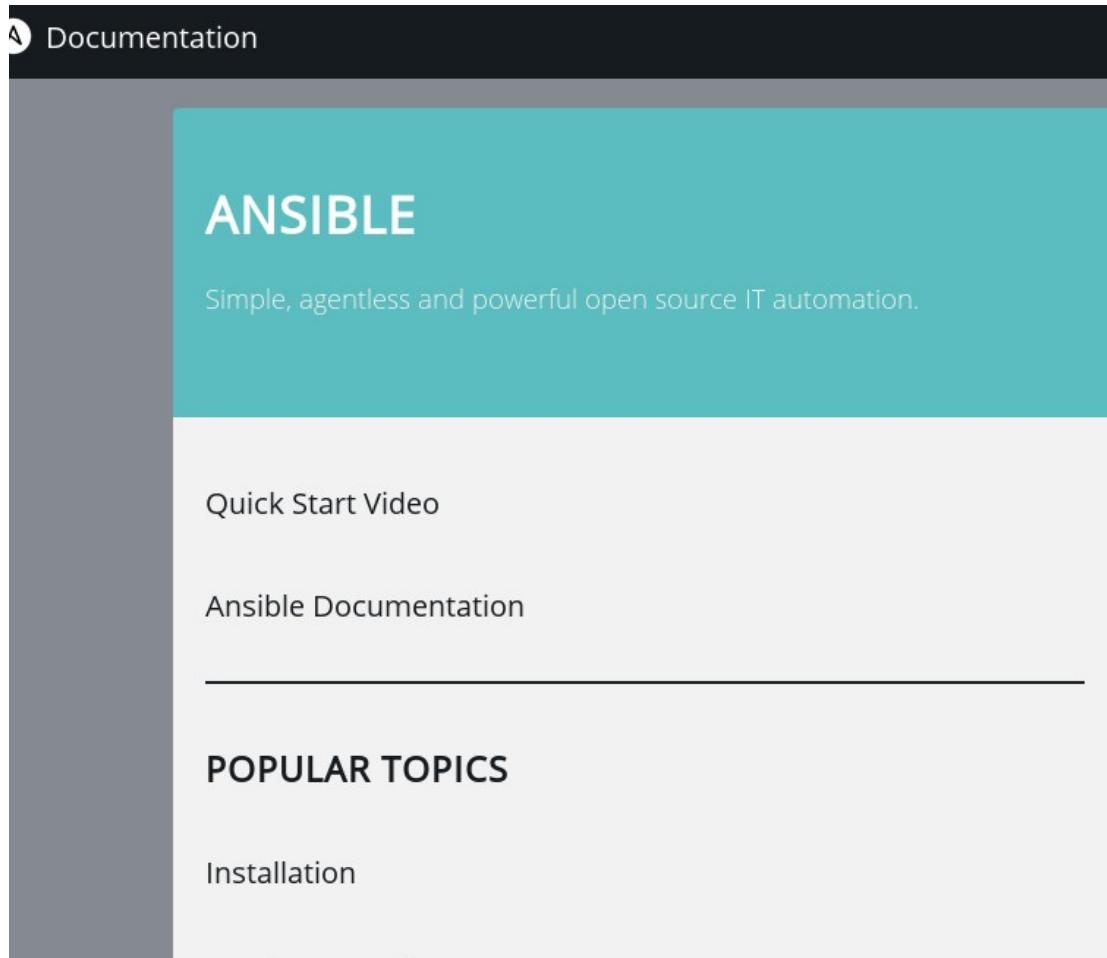
- Provisionierungswerkzeuge (PW) können aktuelle Infrastrukturprobleme lösen
- Einsatz eines PW verbaut nicht den Weg hin zu einer Containerisierung der Infrastruktur
- PW kann bei der Umstellung helfen
- PW erleichtert das Container-Deployment



Fragen?

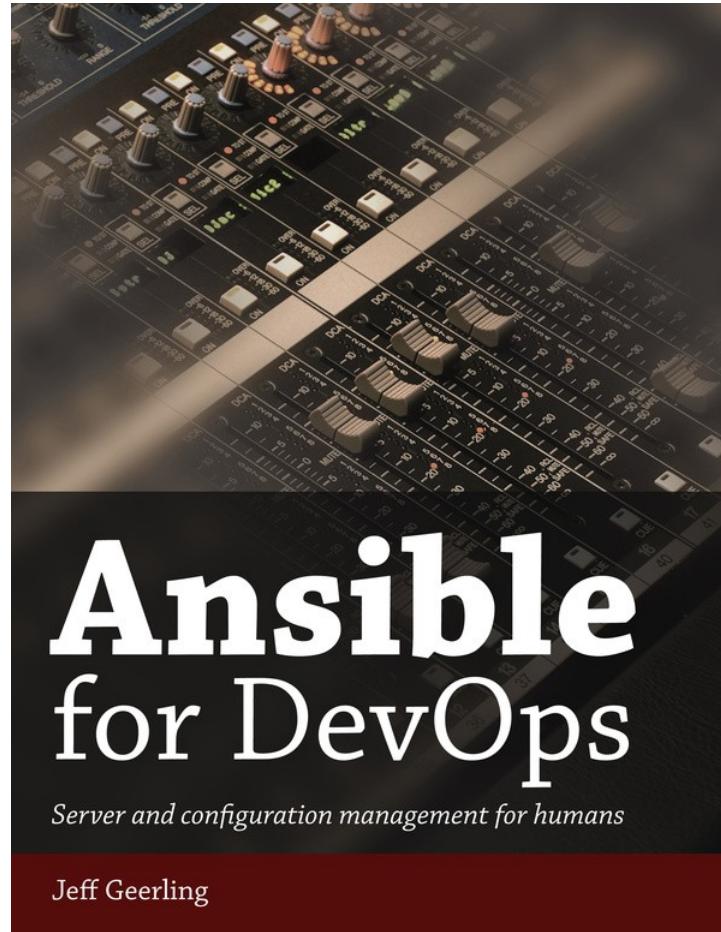
@SandraParsick
mail@sandra-parsick.de
<https://github.com/sparsick/ansible-docker-talk.git>

Weitere Informationen



<http://docs.ansible.com/>

Weitere Informationen



Weitere Informationen

The image shows the front cover of the Java aktuell magazine, Winter 2016 issue. The title 'Java aktuell' is prominently displayed at the top in large blue letters. Below it, the subtitle 'Praxis. Wissen. Networking. Das Magazin für Entwickler' and the tagline 'Aus der Community – für die Community' are visible. The central headline 'Java ist vie...' is partially visible. To the left, there's a cluster of hexagonal icons representing various Java-related concepts like email, links, files, and security. To the right, a large graphic of overlapping arrows in red, yellow, green, and blue points upwards. The text 'Ansible – warum Konfigurationsmanagement auch für Entwickler interessant sein kann' is written below the graphic, attributed to Sandra Parsick. At the bottom, there are three small boxes: one for 'JUnit 5', one for 'Ansible', and one for 'Spring Boot Starter'. The iJUG Verband logo is also present.

04-2016 | Winter | www.ijug.eu

Praxis. Wissen. Networking. Das Magazin für Entwickler
Aus der Community – für die Community

Software organisieren

Java ist vie...

D: 4,90 EUR A: 5,60 EUR CH: 9,80 CHF Bezahlbox: 5,30 EUR ISSN 2191-4977

Ansible – warum Konfigurationsmanagement auch für Entwickler interessant sein kann

Sandra Parsick

JUnit 5
Das nächste große Release steht vor der Tür

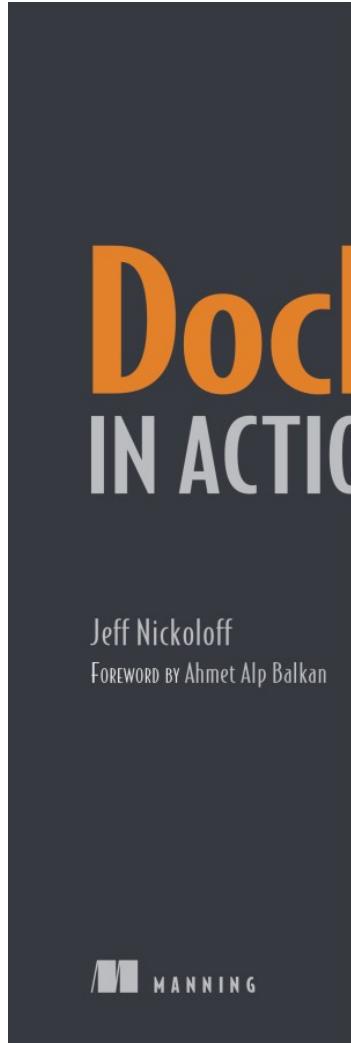
Ansible
Konfigurationsmanagement auch für Entwickler

Spring Boot Starter
Komfortable Modularisierung und Konfiguration

iJUG
Verband

<http://bit.ly/2cZ0lrZ>

Weitere Informationen



A screenshot of the Docker Documentation website. The header includes the "docker docs" logo and links for "What is Docker?", "Product", "Get Docker", "Docs", "Community", "Create Docker ID", and "Sign In". The main section features the heading "Docker Documentation" and a brief description: "Docker provides a way to run applications securely isolated in a container, packaged with all its dependencies and libraries." Below this are two buttons: "Get Docker" (dark red) and "Get Started" (light red). A navigation bar at the bottom offers links to "Guides", "Product manuals", "Glossary", "Reference", and "Samples".

<https://docs.docker.com/>

Weitere Informationen

fachartikel



□ Sandra Parsick, geb. Kosmalta

(mailto:sandra-parsick.de)

Ist als freiberufliche Softwareentwicklerin und Beraterin im Java-Umfeld tätig. Seit 2008 beschäftigt sie sich mit agiler Softwareentwicklung in verschiedenen Rollen. Ihre Schwerpunkte liegen im Bereich der Java Enterprise-Anwendungen, agiles Methoden, Software-Crossmanagement und in der Automatisierung von Softwareentwicklungsprozessen. In ihrer Freizeit engagiert sie sich in der Softwarekammer Ruhrgebiet.

Es muss nicht gleich Docker sein – IT-Automation, die zu einem passt

Docker ist in aller Munde und wird gerne als „Allheilmittel“ für Deployment-Probleme propagiert. Das führt zu der Annahme, automatisierte Deployments wären nur mit Docker möglich, obwohl Provisionierungswerzeuge wie Ansible Lösungen außerhalb der Container-Welt anbieten. Deren Einsatz wird oft gar nicht in Betracht gezogen, weil irgendwann – in ferner Zukunft – doch Docker im Unternehmen eingesetzt werden soll. Die Automatisierung wird immer weiter verschoben, weil der Aufwand in einem Schritt zu groß ist, obwohl Ansible mit wenig Mühe in der Gegenwart helfen könnte. Dieser Artikel zeigt, wie Ansible auf dem Weg zu einer Dockerisierung der Infrastruktur jetzt schon Probleme lösen kann und wie ein gemeinsamer Einsatz beider Technologien die Vorteile beider Welten kombiniert.

Wer kennt es nicht: Die Bereitstellung einer neuen Testinfrastruktur dauert ewig, weil sie manuell installiert werden muss; Die Produktionsnahme des neuen Releases hat wieder länger gedauert als geplant, weil die Konfiguration der Produktionsumgebung sich schon wieder von denen der Testumgebungen unterscheidet. Die Entwickler Alice und Bob kommen von einer Konferenz wieder, bei der sie gelernt haben, wenn sie ihre Deployments auf Docker umstellen, löst es ihre oben genannten Probleme. Doch der Betrieb wird die Umstellung auf Docker – eventuell, vielleicht – in einem oder zwei Jahren machen wollen.

Entwickler Carol schlägt vor, bis der Betrieb mit Docker so weit ist, die Serverkonfiguration und Deployments mit einem Provisionierungswerzeug wie Ansible zu automatisieren und davon ausgehend, die Dockerisierung der Infrastruktur voranzutreiben. Die Kollegen lehnen diesen Vorschlag ab, da sie sich für die Technologie Docker entschieden haben und aus ihrer

Sicht wäre es zu viel Aufwand, eine andere Technologie als Zwischenlösung zu nehmen. Somit quälen sie sich – eventuell, vielleicht – weitere ein bis zwei Jahre mit ihren Infrastrukturproblemen. Wie die Reise hätte aussehen können, wenn Carols Vorschlag angenommen worden wäre, beschreiben die nächsten Abschnitte.

Ausgangslage und Reiseroute
Als Ausgangspunkt dient folgendes Setup: Eine Java-Webapplikation wird auf einem Tomcat-Server deployt und braucht eine MySQL-Datenbank. Die Datenbank und der Tomcat-Server werden auf zwei separaten Servern betrieben (siehe Abbildung 1).



Abb. 1: Ausgangsinfrastruktur

OBJEKTspektrum Online Themenspecial DevOps 2017

<http://bit.ly/2p7mxyB>

Weitere Informationen



[Über uns](#) | [Media](#) | [Kontakt](#) | [Impressum](#)



<https://bit.ly/2S4PCHJ>

Management

Entwicklung

Betrieb

News

Termine

IT-Jobs

Suchbegriff

Themen:

Datenbanken

Netzwerke

Internet of Things

Java

Agile Softwareentwicklung

DevOps

Microservices

Sicherheit

» Entwicklung » Methoden

Sandra Parsick

14. August 2018

IT-Automatisierung mit oder ohne Docker



© enanuchit / Fotolia.com

Oft werden Container mit klassischen Provisionierungswerkzeugen verglichen, obwohl Container andere Problemstellungen lösen. Dieser Artikel zeigt, worin sich Container bzw. Werkzeuge aus dem Container-Umfeld und Provisionierungswerkzeuge unterscheiden und beleuchtet, in welchen Situationen es Sinn macht, auf ein klassisches Provisionierungswerkzeug zu setzen, wann das Container-Ökosystem die bessere Lösung ist oder auch eine Kombination aus beidem.

Autorin



Sandra Parsick

Sandra Parsick ist als freiberufliche Softwareentwicklerin und Consultant im Java-Umfeld tätig. Seit 2008 beschäftigt sie sich mit agiler... >> [Weiterlesen](#)

Weitere Informationen

- Talk „Docker Container Loading“ von Roland Huß
<https://github.com/ro14nd-talks/docker-container-loading/blob/master/docker-container-loading.pdf>
- Serverspecs <http://serverspec.org/>
- Testcontainers <https://www.testcontainers.org/>
- Docker und Java
 - <https://banzaicloud.com/blog/java-resource-limits/>
 - <https://developers.redhat.com/blog/2017/03/14/java-inside-docker/>
 - <https://blog.docker.com/2018/04/improved-docker-container-integration-with-java-10/>

Bildnachweise

<https://pixabay.com/de/vortrag-vorlesung-schule-2044619/>

<https://pixabay.com/de/spielsteine-figuren-holz-bunt-1743307/>

<https://pixabay.com/de/fragezeichen-birne-denken-idee-2010011/>

<https://pixabay.com/de/hamburg-hafen-kr%C3%A4ne-containerschiff-2103261/>

<https://pixabay.com/de/tafel-pfeile-entscheidung-rechts-2084777/>