
PUZZLE ITC
changing IT for the better

OpenShift 3 für Java Entwickler

Thomas Philipona

1. Was ist OpenShift?
2. CI / CD Workflow
3. Java EE ein Beispiel
4. Demo

Agenda

1

Was ist OpenShift V3?

Was ist OpenShift 3?

“next generation PaaS” OpenShift Enterprise V3 von Red Hat

Neu Implementation, V2 wurde verworfen und komplett neu gebaut.

V2	**V3**
Namespace/Domain	Project
Gear	Docker Container
Cartridge	Docker Base Image
rhc	oc (OpenShift Client)

OpenShift basiert auf etablierten Open Source Konzepten

Docker

Kubernetes

OpenShift 3 (1/2)

Container Platform as a Service (PaaS)

Multinode Platform, um Applikationen in Containern zu betreiben

One Platform runs it all!

Fancy CLI und Gui (Selfservice)

OpenShift 3 (2/2)

Base Images für RHEL 6 und 7 patched analog Standard RHEL

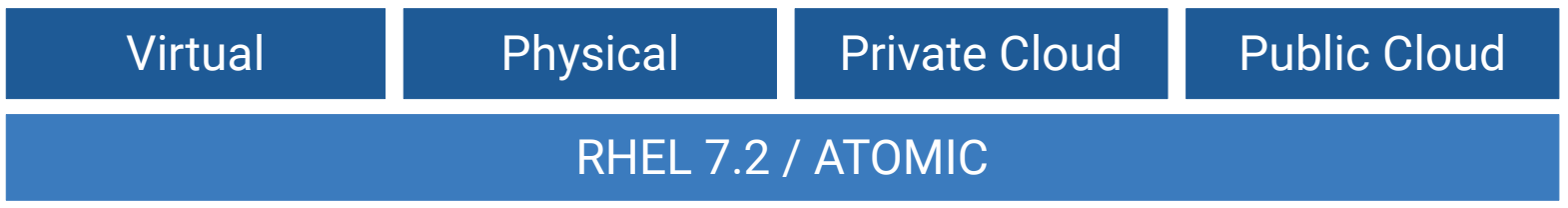
Redeployments werden durch Basis Image Update getriggert

Standard Deployment Mechanismus und Workflow

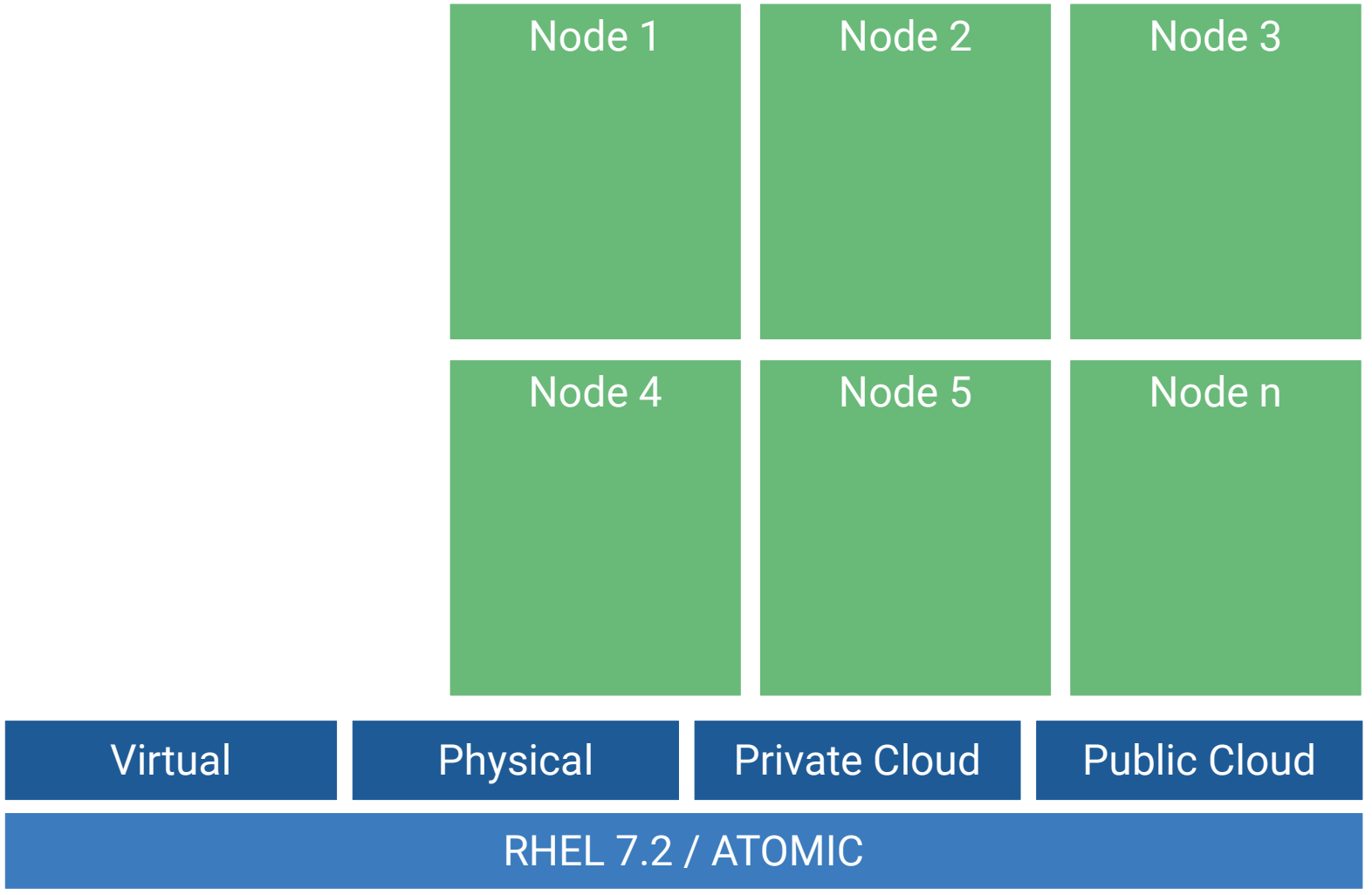
Autoscaling

Container Security

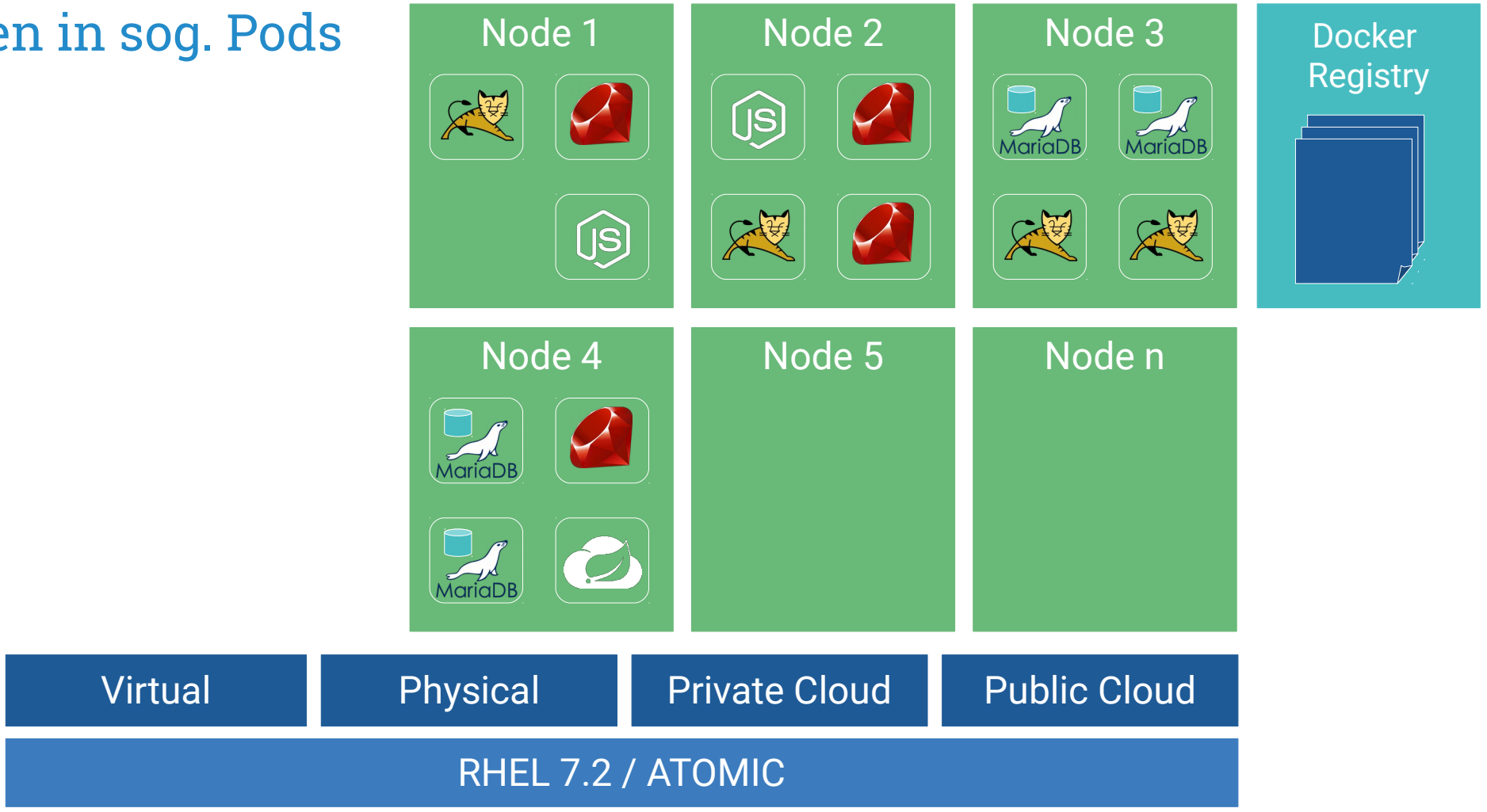
RHEL 7.2


Virtual
Physical
Private Cloud
Public Cloud
RHEL 7.2 / ATOMIC

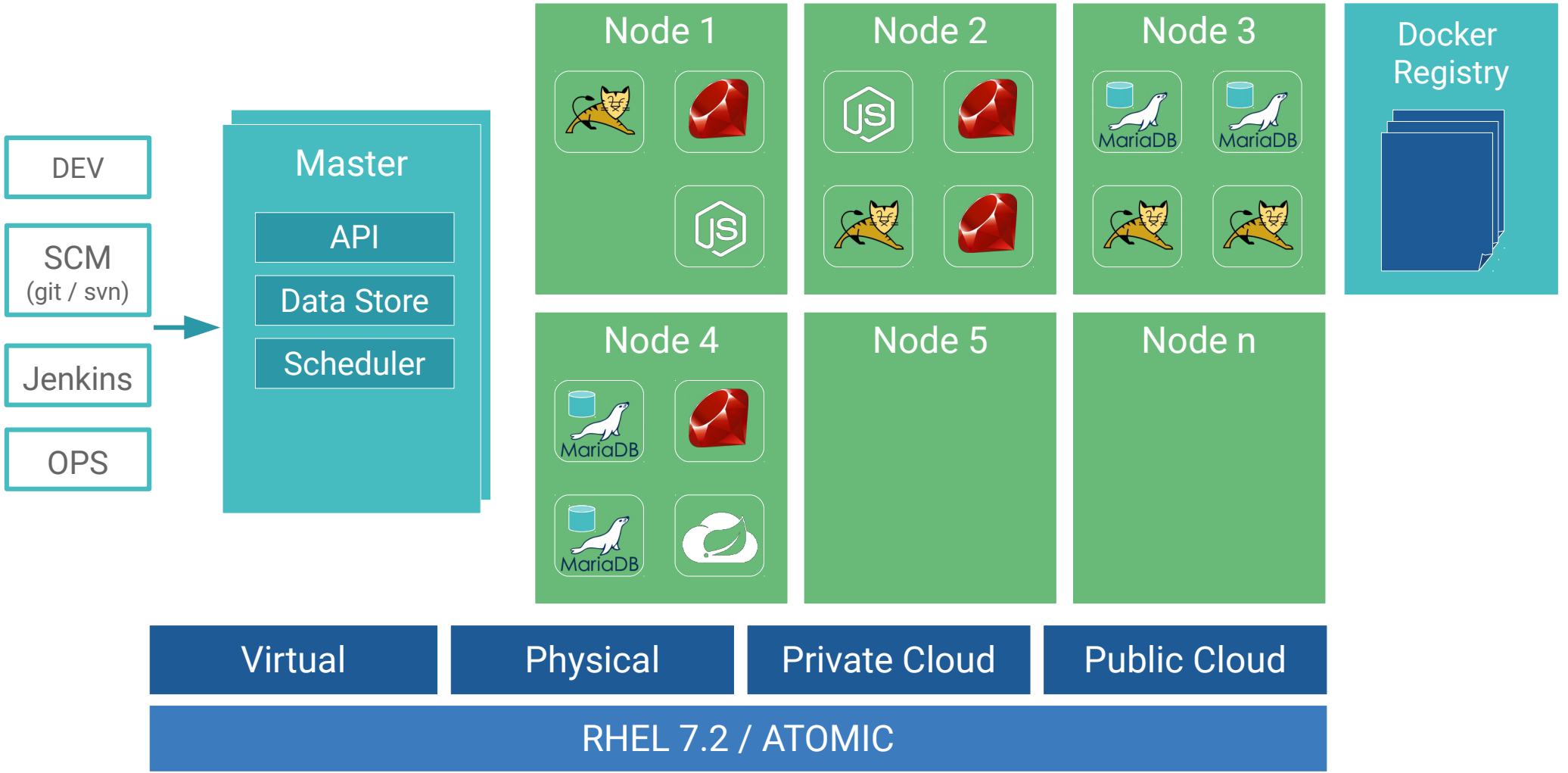
Compute Nodes



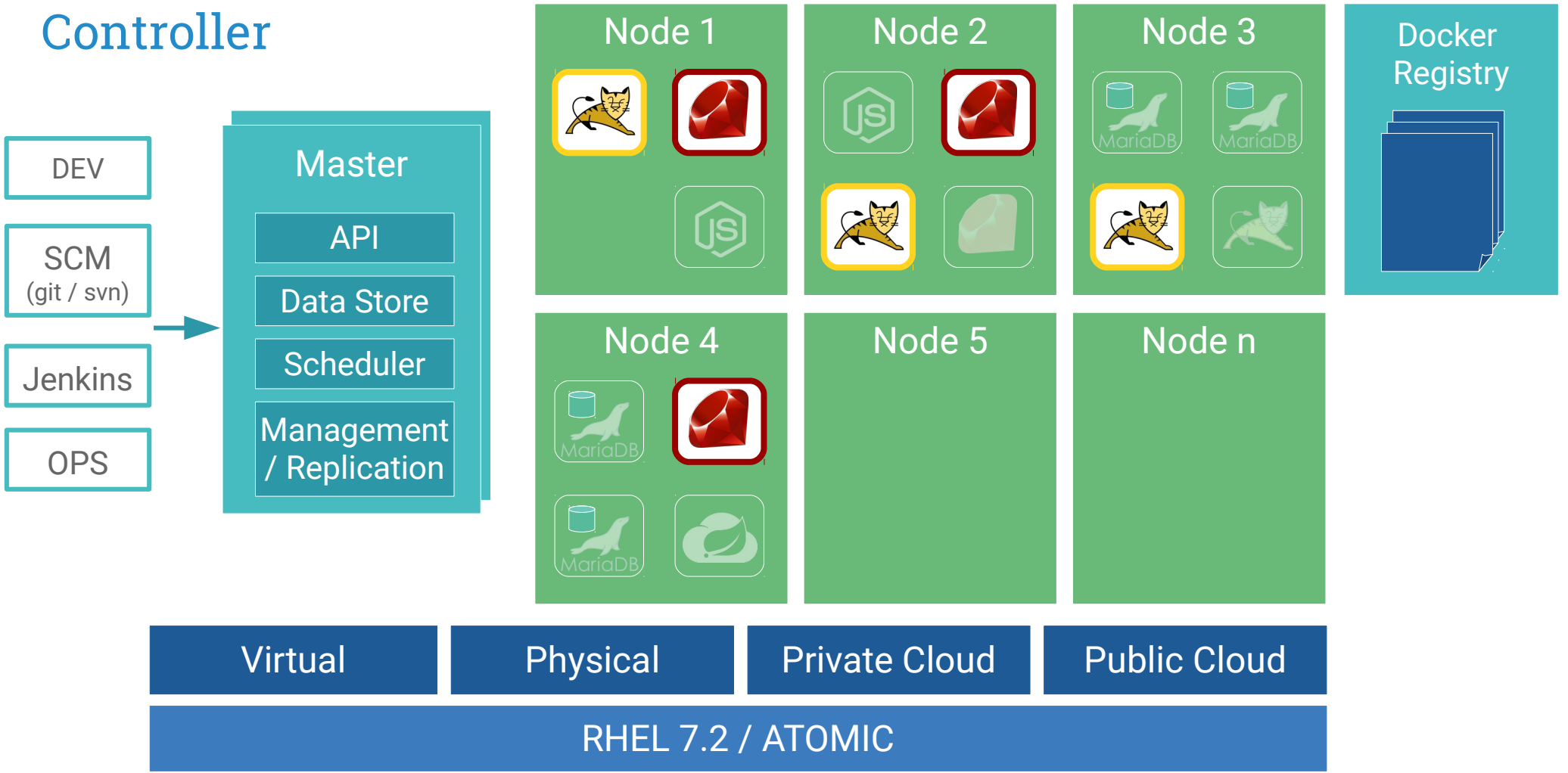
Applikationen laufen in sog. Pods



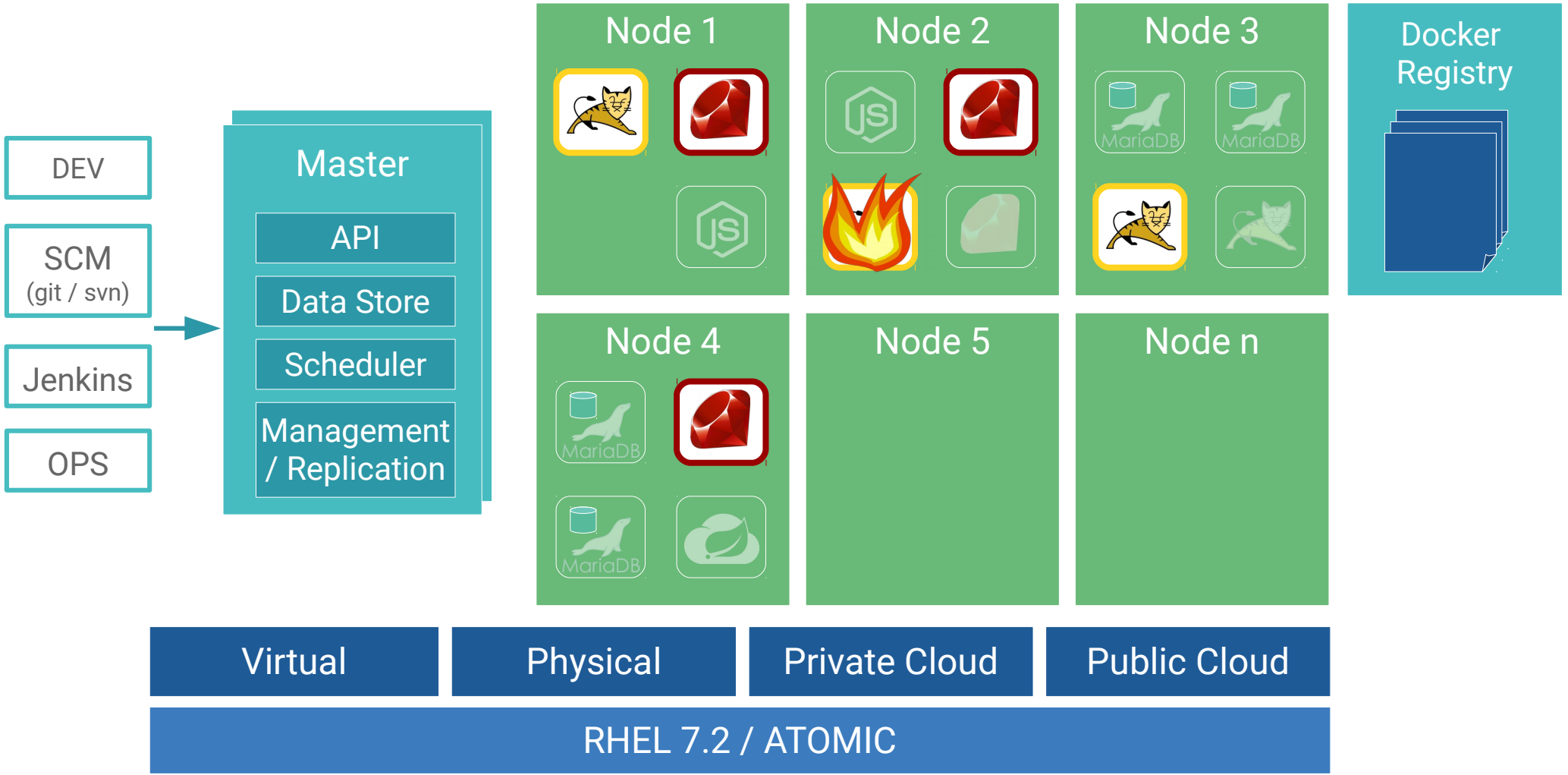
OSE 3 Master


DEV
SCM
(git / svn)
Jenkins
OPS
Master
API
Data Store
Scheduler
Node 1
Node 2
Node 3
Node 4
Node 5
Node n
MariaDB
MariaDB
MariaDB
Docker
Registry
Virtual
Physical
Private Cloud
Public Cloud
RHEL 7.2 / ATOMIC

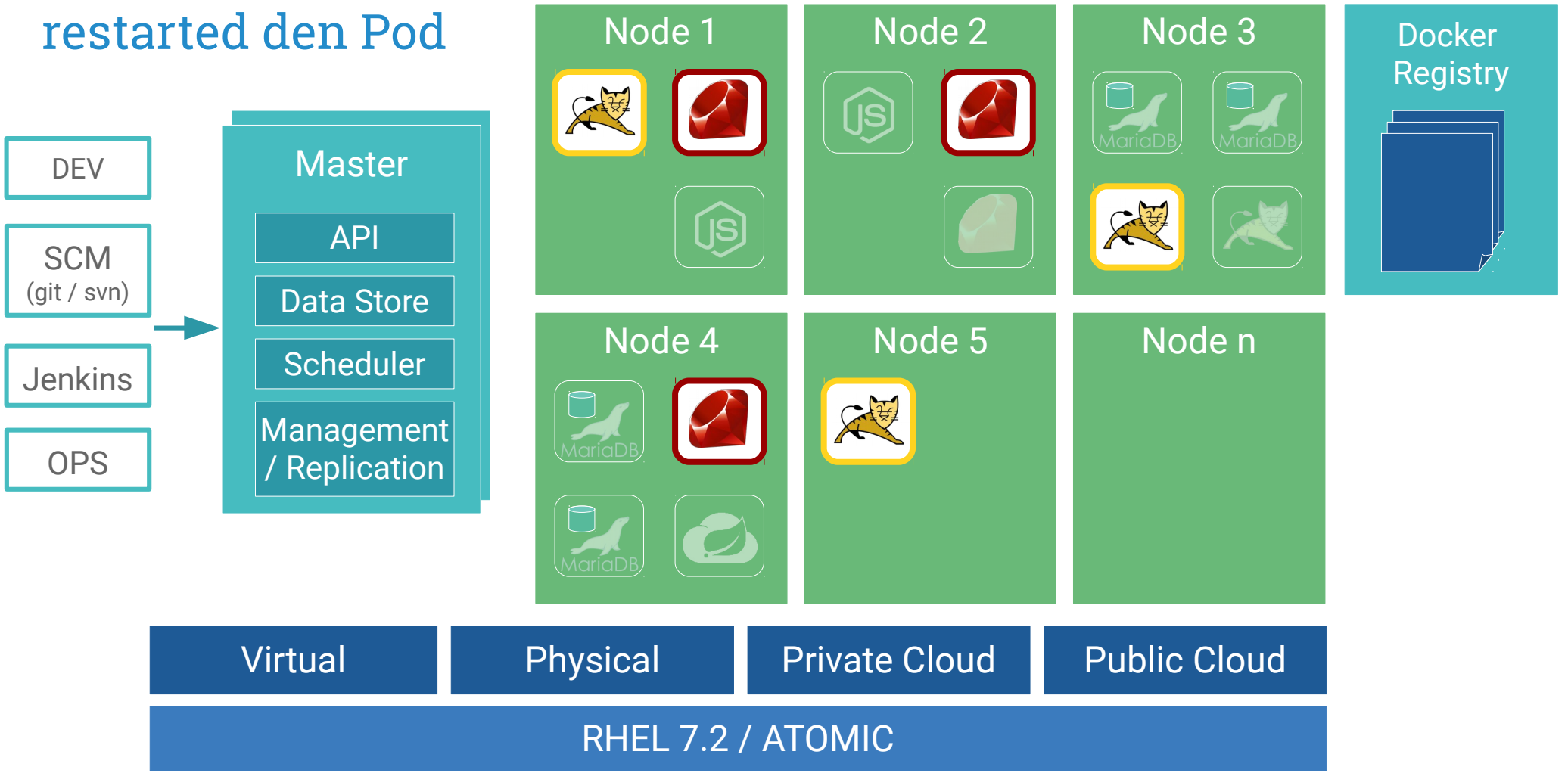
Replication Controller



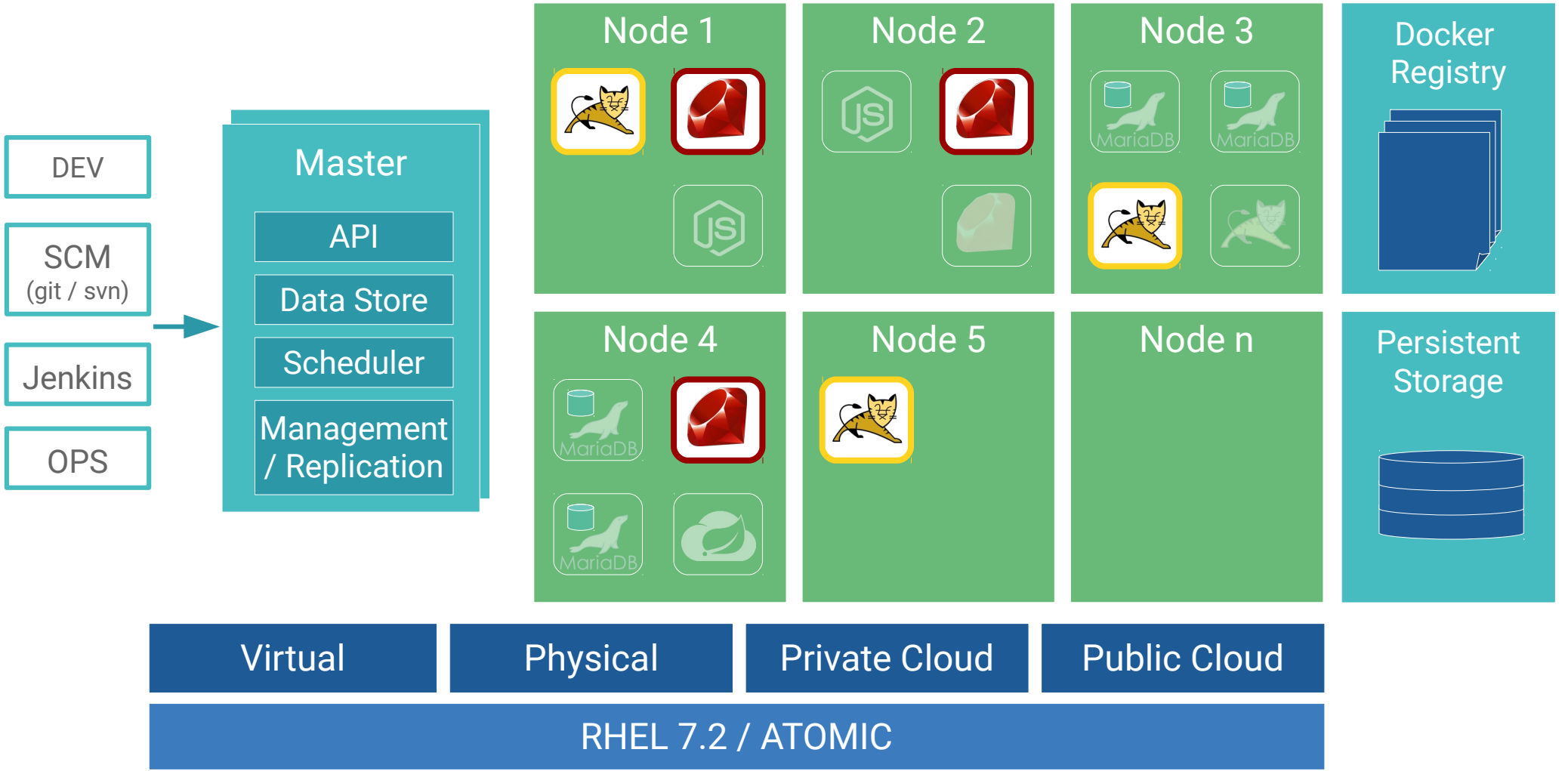
Container Fails



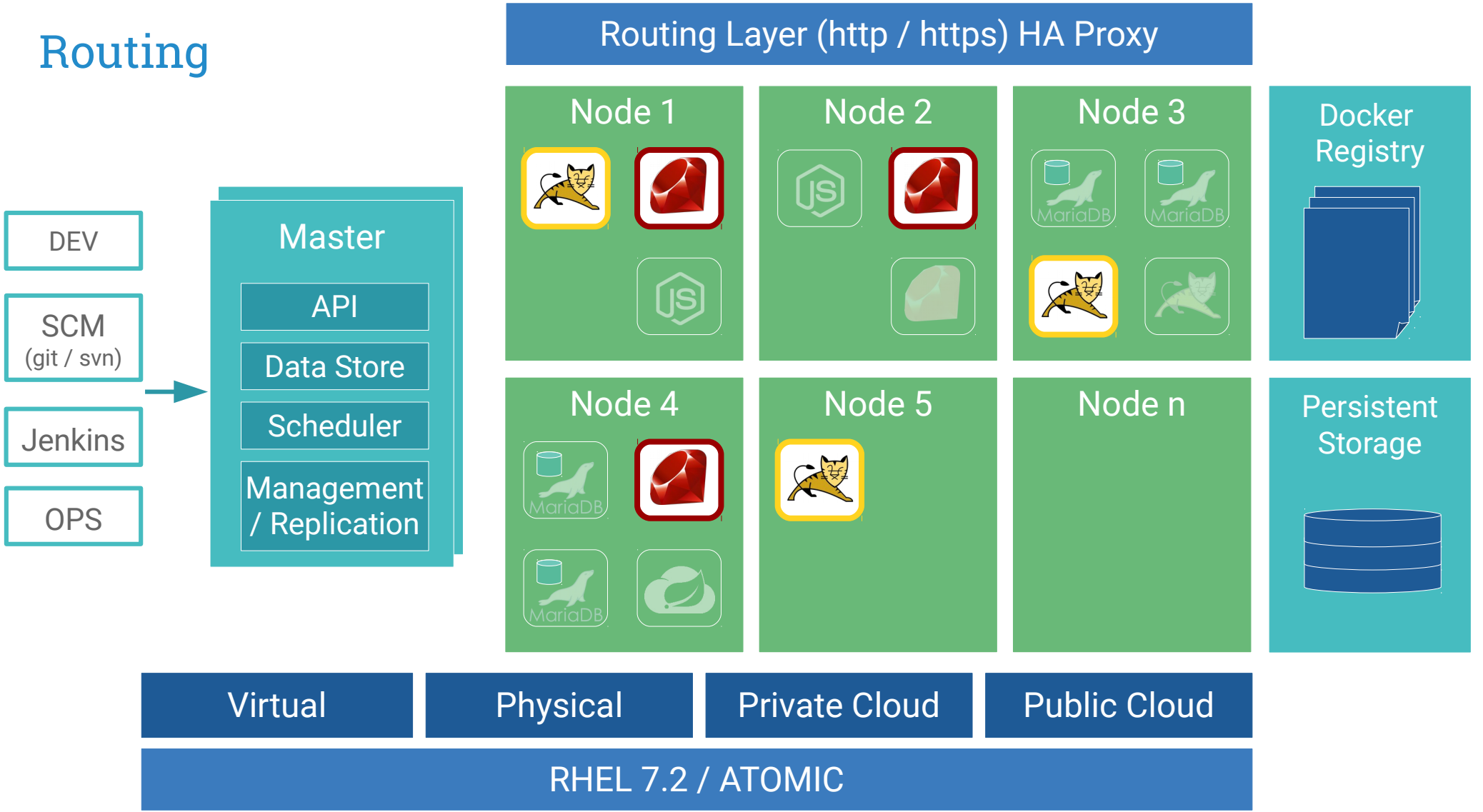
Kubernetes restarted den Pod



Persistent Storage



Routing


Routing Layer (http / https) HA Proxy
Node 1
Node 2
Node 3
Docker Registry
Node 4
Node 5
Node n
Persistent Storage
MariaDB
MariaDB
MariaDB
DEV
SCM
(git / svn)
Jenkins
OPS
Master
API
Data Store
Scheduler
Management / Replication
Virtual
Physical
Private Cloud
Public Cloud
RHEL 7.2 / ATOMIC

Beispiel Java Projekt

- War deployed in Wildfly 10
- Maria DB (mit Persistent Volume)

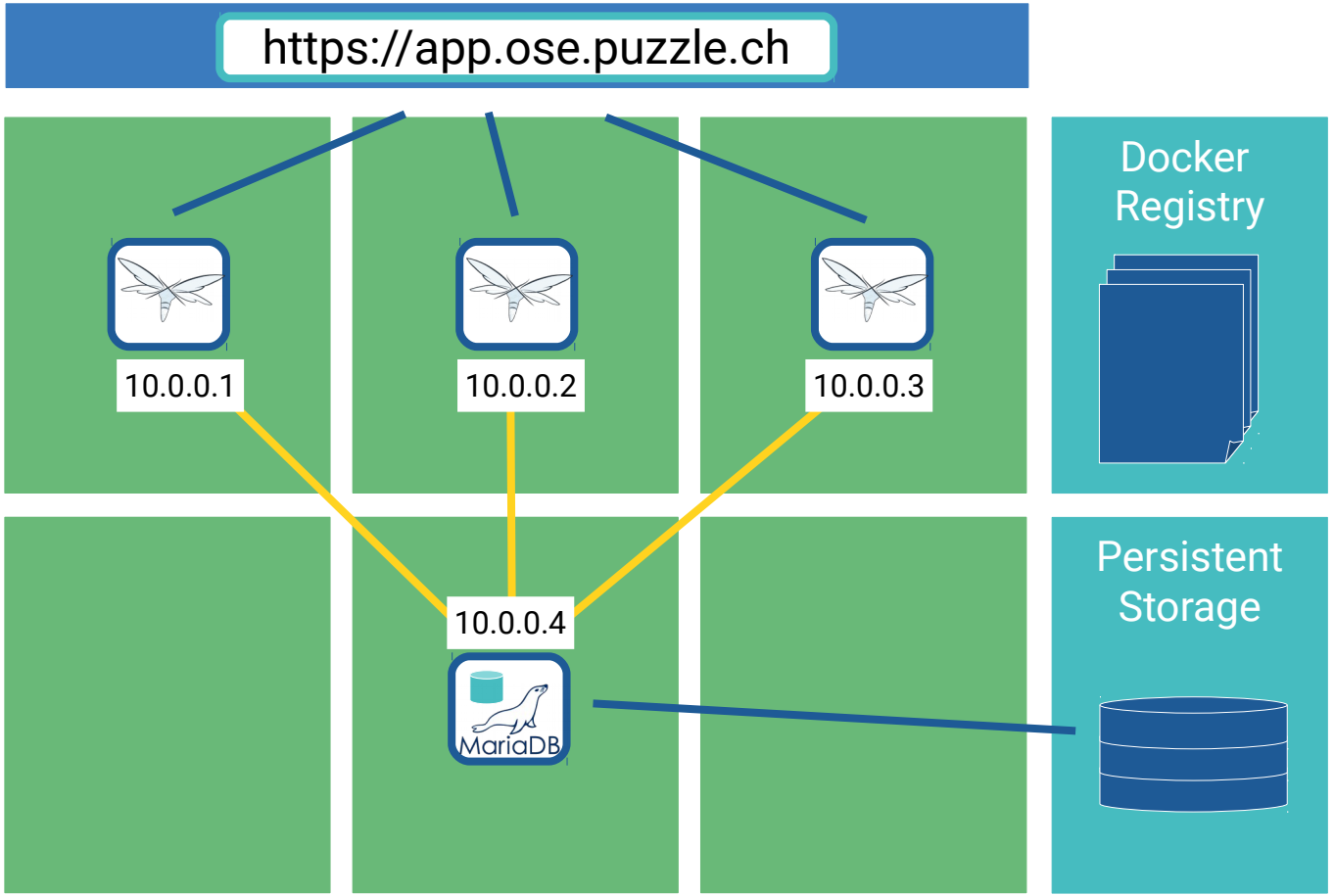
Beispiel Java App OSE 3 Projekt (1)

Via Route im Netz verfügbar
Loadbalancing auf Pods

oc scale dc app –replicas=3

Verbunden über transparentes Software Defined Network (SDN)

Projekt Setup als Json exportier- / importierbar

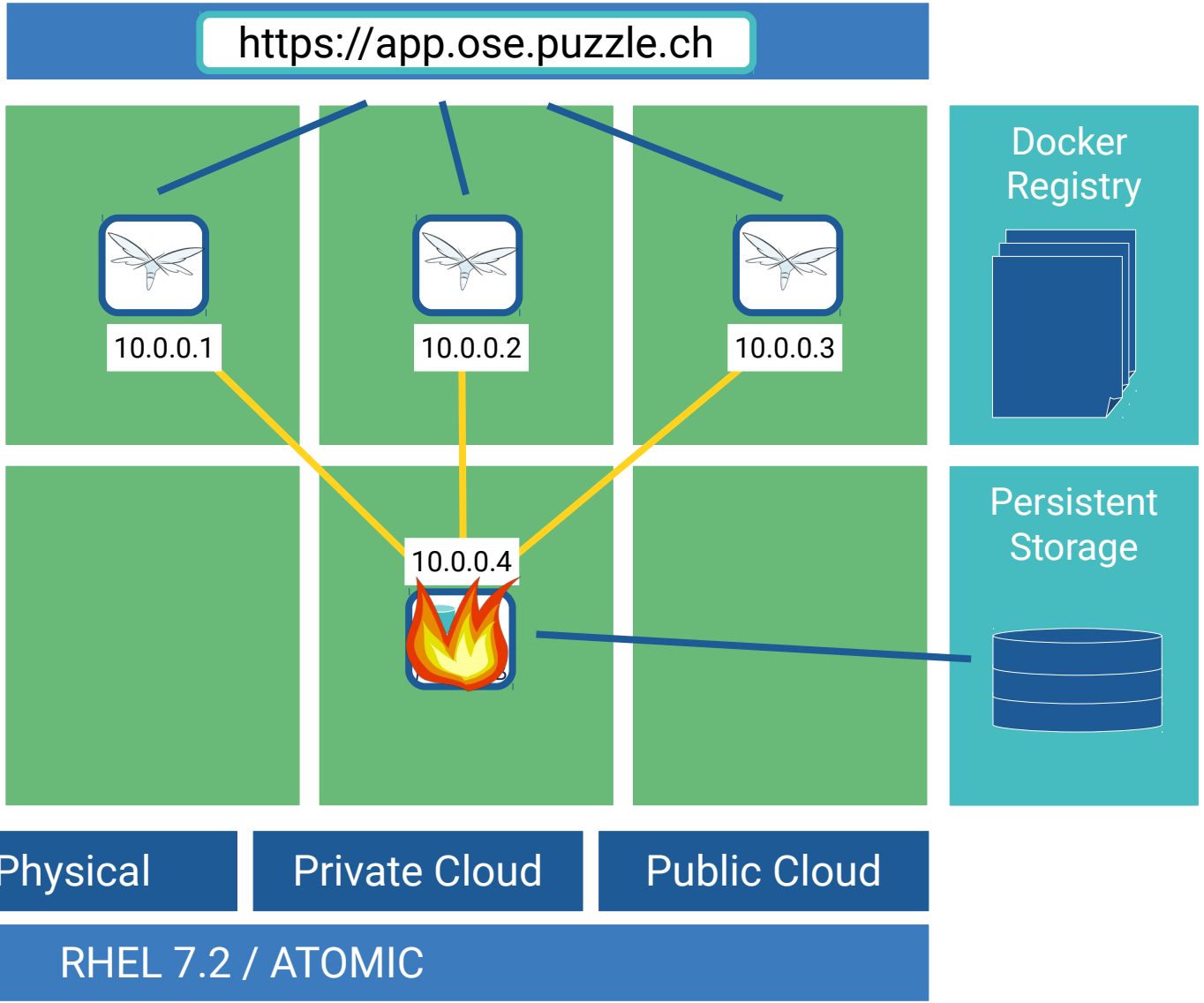


Virtual | Physical | Private Cloud | Public Cloud

RHEL 7.2 / ATOMIC

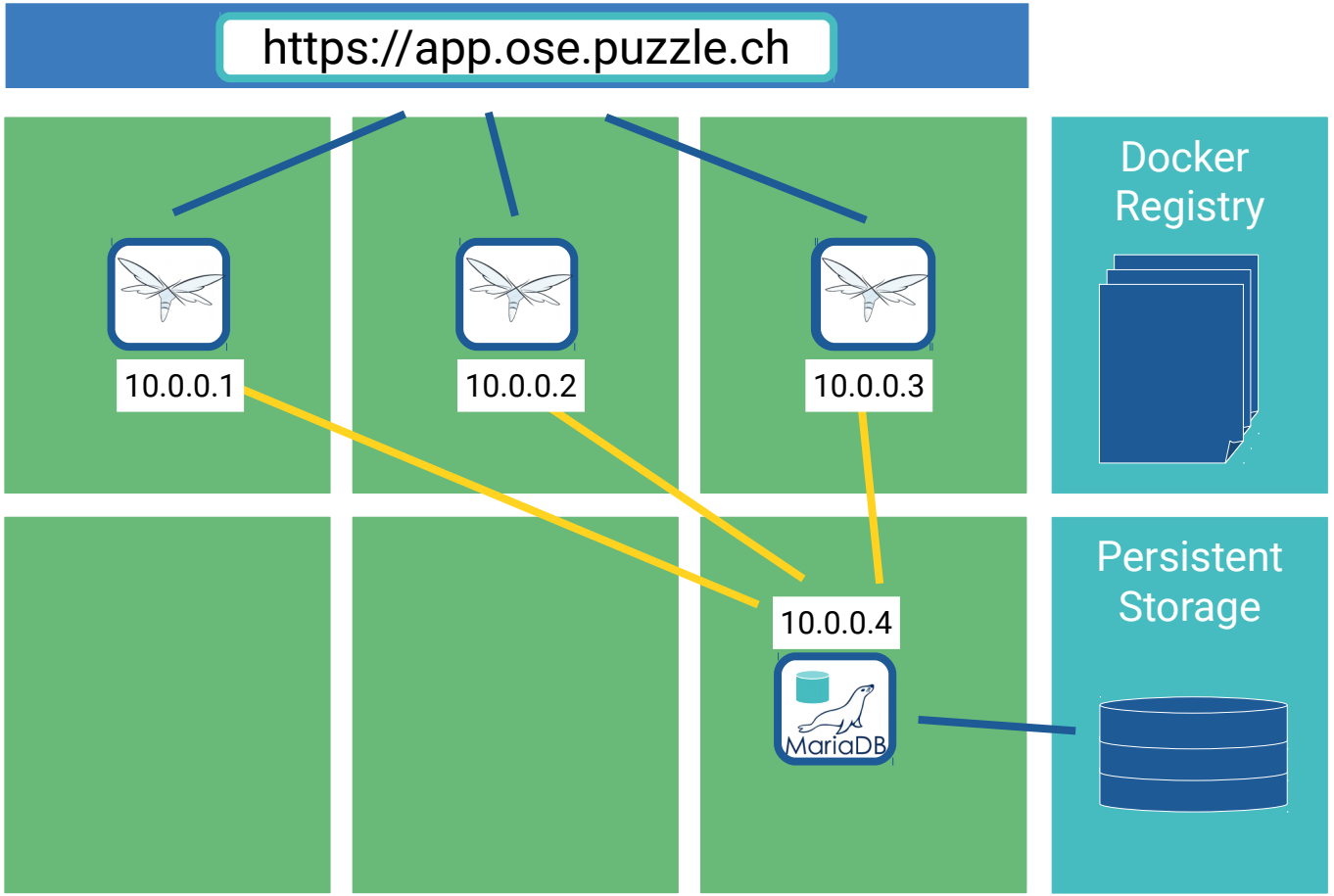
Beispiel Java App
OSE 3 Projekt (2)

Maria DB Pod failed



Beispiel Java App
OSE 3 Projekt (3)

Pod wird neu gestartet
- Gleiche Adresse
- Persistent Volume wandert mit



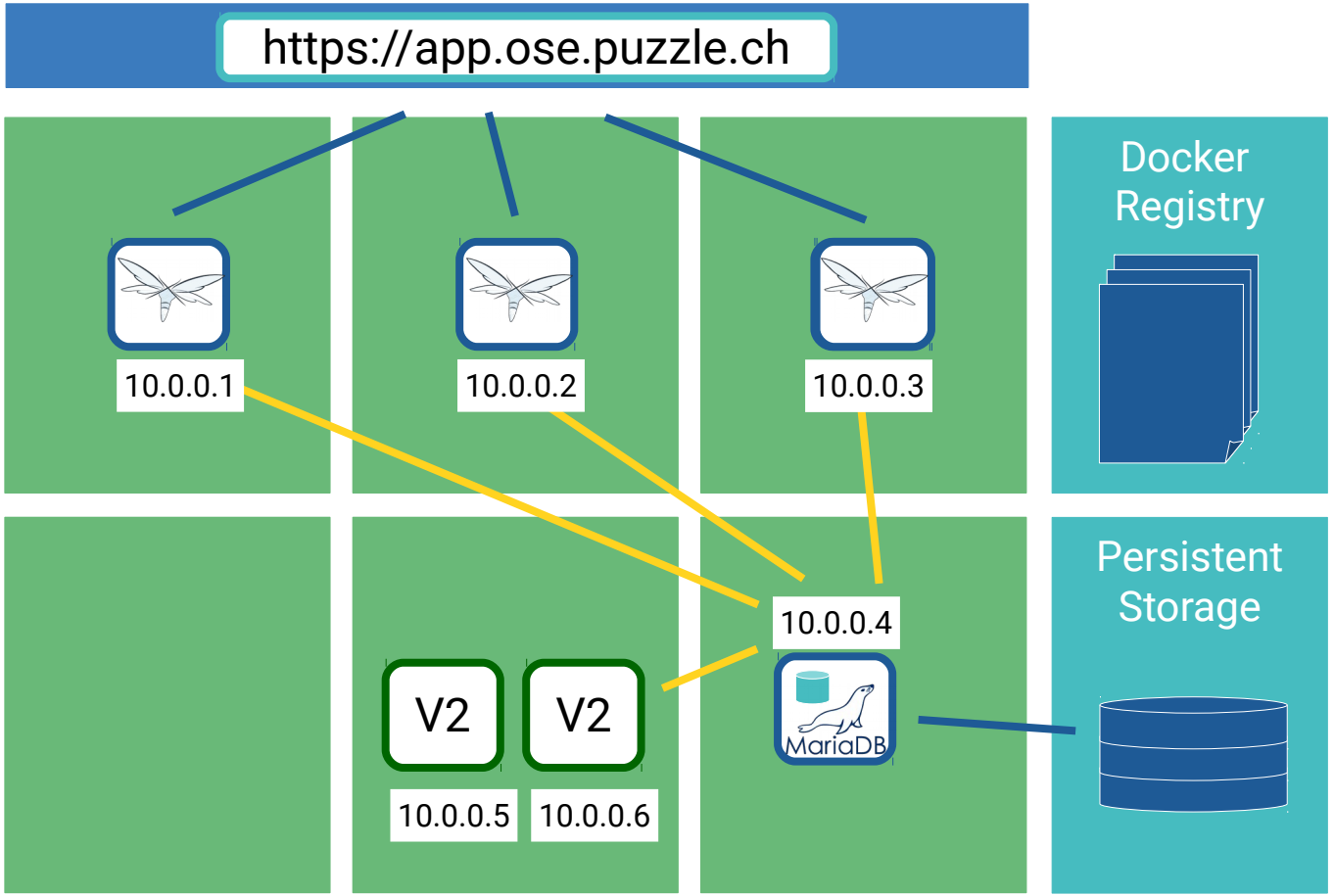
Virtual | Physical | Private Cloud | Public Cloud

RHEL 7.2 / ATOMIC

Beispiel Java App
OSE 3 Projekt (4)

Rolling Update

Applikation in neuer Version wird deployed



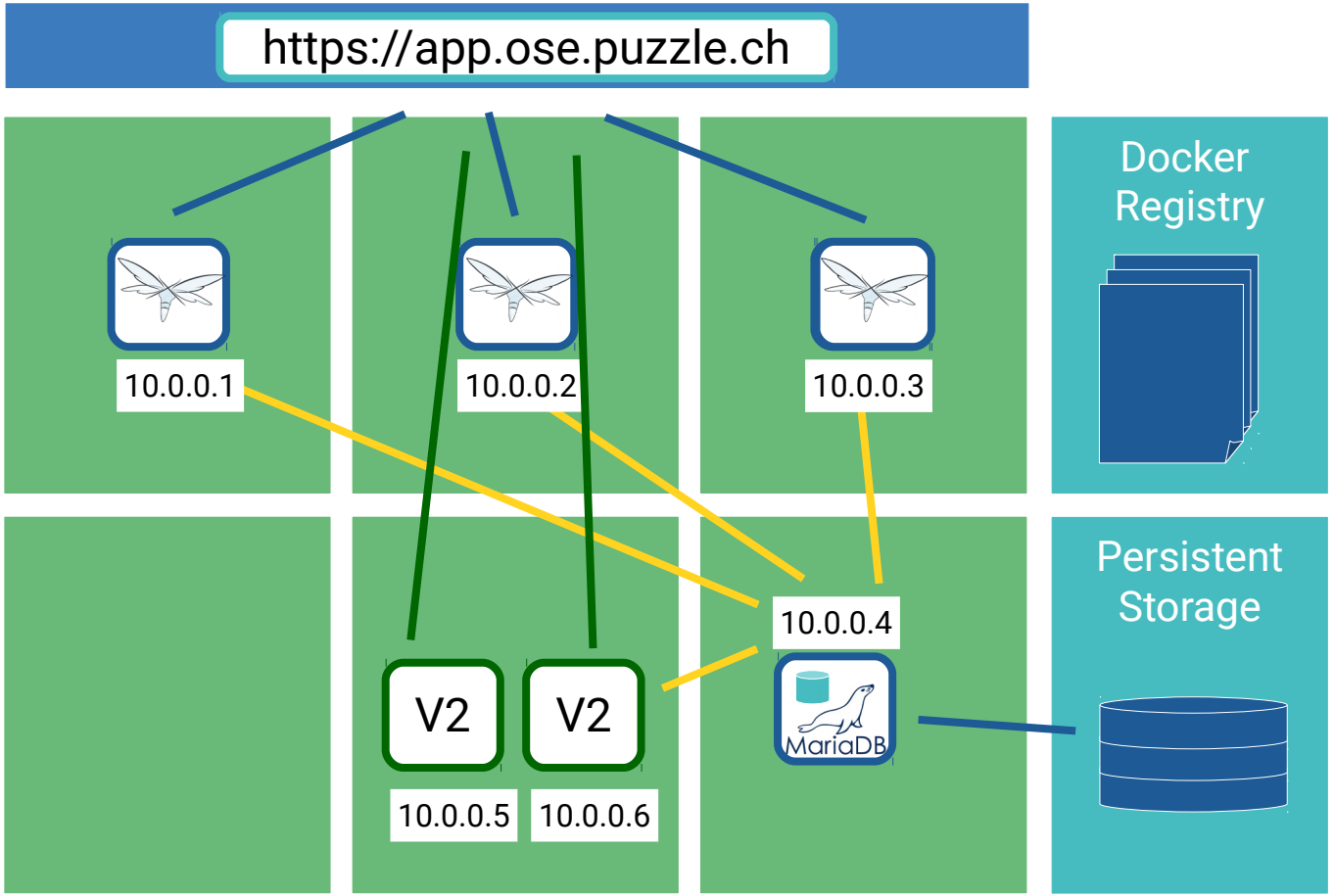
Virtual | Physical | Private Cloud | Public Cloud

RHEL 7.2 / ATOMIC

Beispiel Java App
OSE 3 Projekt (5)

Rolling Update

Traffic wird auf Pods gerouted



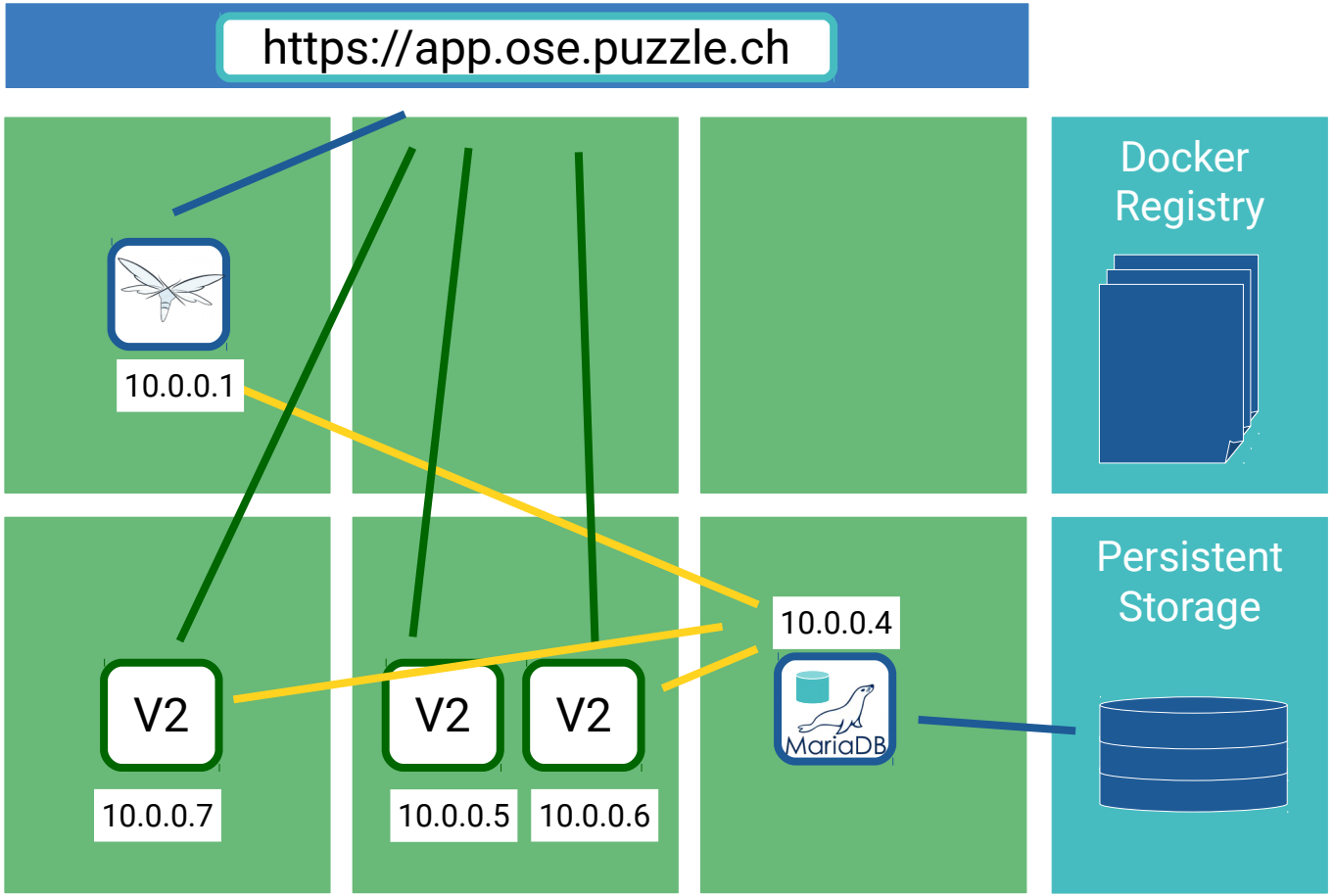
Virtual | Physical | Private Cloud | Public Cloud

RHEL 7.2 / ATOMIC

Beispiel Java App
OSE 3 Projekt (6)

Rolling Update

„Alte“ Container werden entfernt



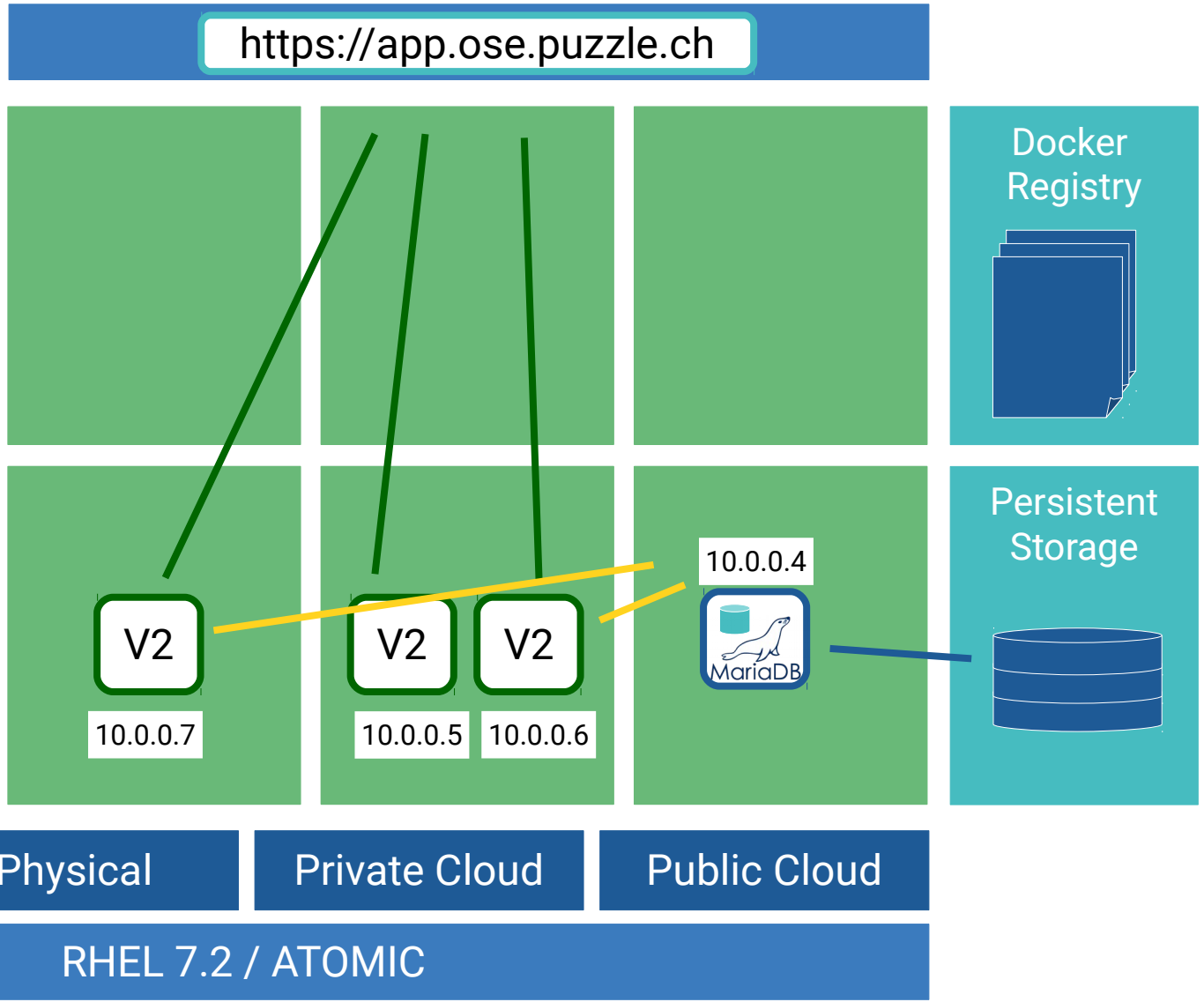
Virtual | Physical | Private Cloud | Public Cloud

RHEL 7.2 / ATOMIC

Beispiel Java App
OSE 3 Projekt (7)

Rolling Update

„Alte“ Container werden entfernt



Deployment Strategies

Rolling Strategy

- Applikation muss damit umgehen können, dass sie kurzzeitig in zwei verschiedenen Versionen läuft.
- DB Changes

Recreate Strategy

- Alte Pods werden downscaled,
- Neue Up

Config via Env

Beispiel, wie im Wildfly dann die DB configuriert werden kann.

```
<datasource jndi-name="java:jboss/datasources/${env.OPENSHIFT_POSTGRESQL_DATASOURCE}" enabled="${postgresql.enabled}" use
    <connection-url>
jdbc:postgresql://${env.OPENSHIFT_POSTGRESQL_DB_HOST}:${env.OPENSHIFT_POSTGRESQL_DB_PORT}/${env.OPENSHIFT_POSTGRESQL_DB_N
    </connection-url>
    <driver>postgresql</driver>
    <security>
      <user-name>${env.OPENSHIFT_POSTGRESQL_DB_USERNAME}</user-name>
      <password>${env.OPENSHIFT_POSTGRESQL_DB_PASSWORD}</password>
    </security>
    <validation>
        <check-valid-connection-sql>SELECT 1</check-valid-connection-sql>
        <background-validation>true</background-validation>
        <background-validation-millis>60000</background-validation-millis>
        <!--<validate-on-match>true</validate-on-match>-->
    </validation>
    <pool>
        <flush-strategy>IdleConnections</flush-strategy>
    </pool>
</datasource>
```

Config via Env

Beispiel, wie im Wildfly dann die DB configuriert werden kann.

```
<datasource jndi-name="java:jboss/datasources/${env.OPENSHIFT_POSTGRESQL_DATASOURCE}" enabled="${postgresql.enabled}" use
    <connection-url>
jdbc:postgresql://${env.OPENSHIFT_POSTGRESQL_DB_HOST}:${env.OPENSHIFT_POSTGRESQL_DB_PORT}/${env.OPENSHIFT_POSTGRESQL_DB_N
    </connection-url>
    <driver>postgresql</driver>
    <security>
      <user-name>${env.OPENSHIFT_POSTGRESQL_DB_USERNAME}</user-name>
      <password>${env.OPENSHIFT_POSTGRESQL_DB_PASSWORD}</password>
    </security>
    <validation>
        <check-valid-connection-sql>SELECT 1</check-valid-connection-sql>
        <background-validation>true</background-validation>
        <background-validation-millis>60000</background-validation-millis>
        <!--<validate-on-match>true</validate-on-match>-->
    </validation>
    <pool>
        <flush-strategy>IdleConnections</flush-strategy>
    </pool>
</datasource>
```

Config via Env

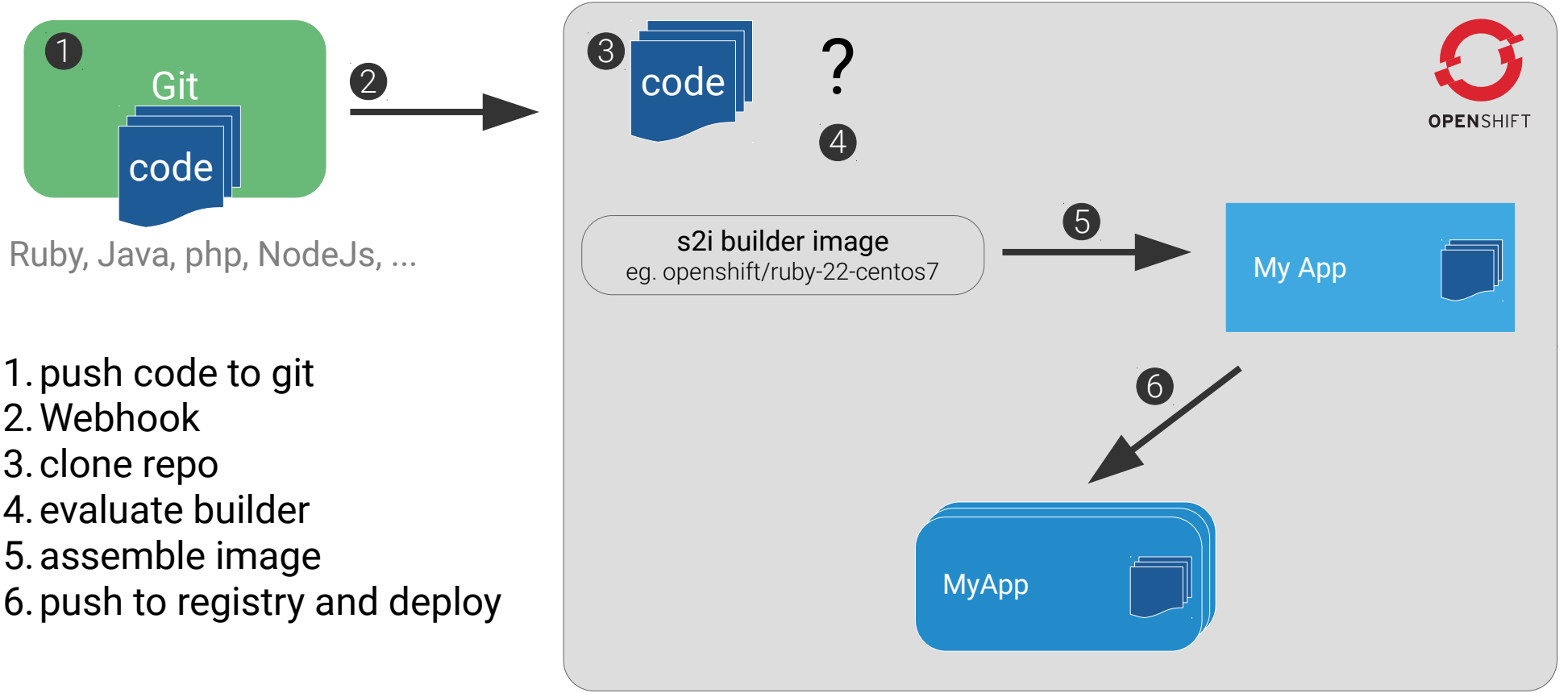
Beispiel, wie im Wildfly dann die DB configuriert werden kann.

```
<datasource jndi-name="java:jboss/datasources/${env.OPENSHIFT_POSTGRESQL_DATASOURCE}" enabled="${postgresql.enabled}" use
    <connection-url>
jdbc:postgresql://${env.OPENSHIFT_POSTGRESQL_DB_HOST}:${env.OPENSHIFT_POSTGRESQL_DB_PORT}/${env.OPENSHIFT_POSTGRESQL_DB_N
    </connection-url>
    <driver>postgresql</driver>
    <security>
    </
    <v
    </
    <pool>
        <flush-strategy>IdleConnections</flush-strategy>
    </pool>
</datasource>
```

```
$ oc exec app-1-szlkv env |grep POSTGRESQL_SERVICE

POSTGRESQL_SERVICE_PORT=5432
POSTGRESQL_SERVICE_HOST=172.30.232.156
```

Wie kommt die Software auf OpenShift?

Workflow: Source To Image (S2I)

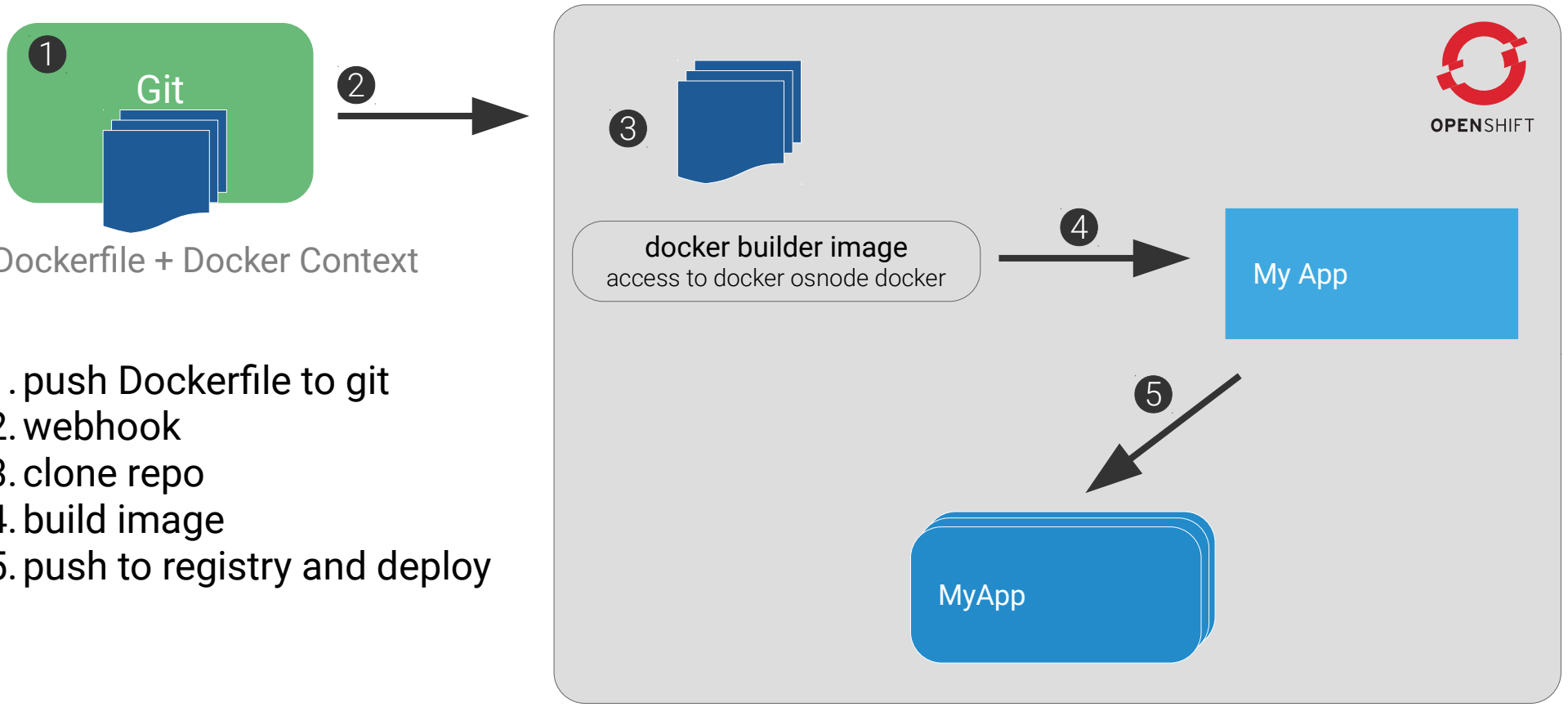


1. push code to git
2. Webhook
3. clone repo
4. evaluate builder
5. assemble image
6. push to registry and deploy

Workflow: Source To Image (S2I) Beispiel

https://github.com/appuio/example-php-sti-helloworld

Workflow: Docker Builds

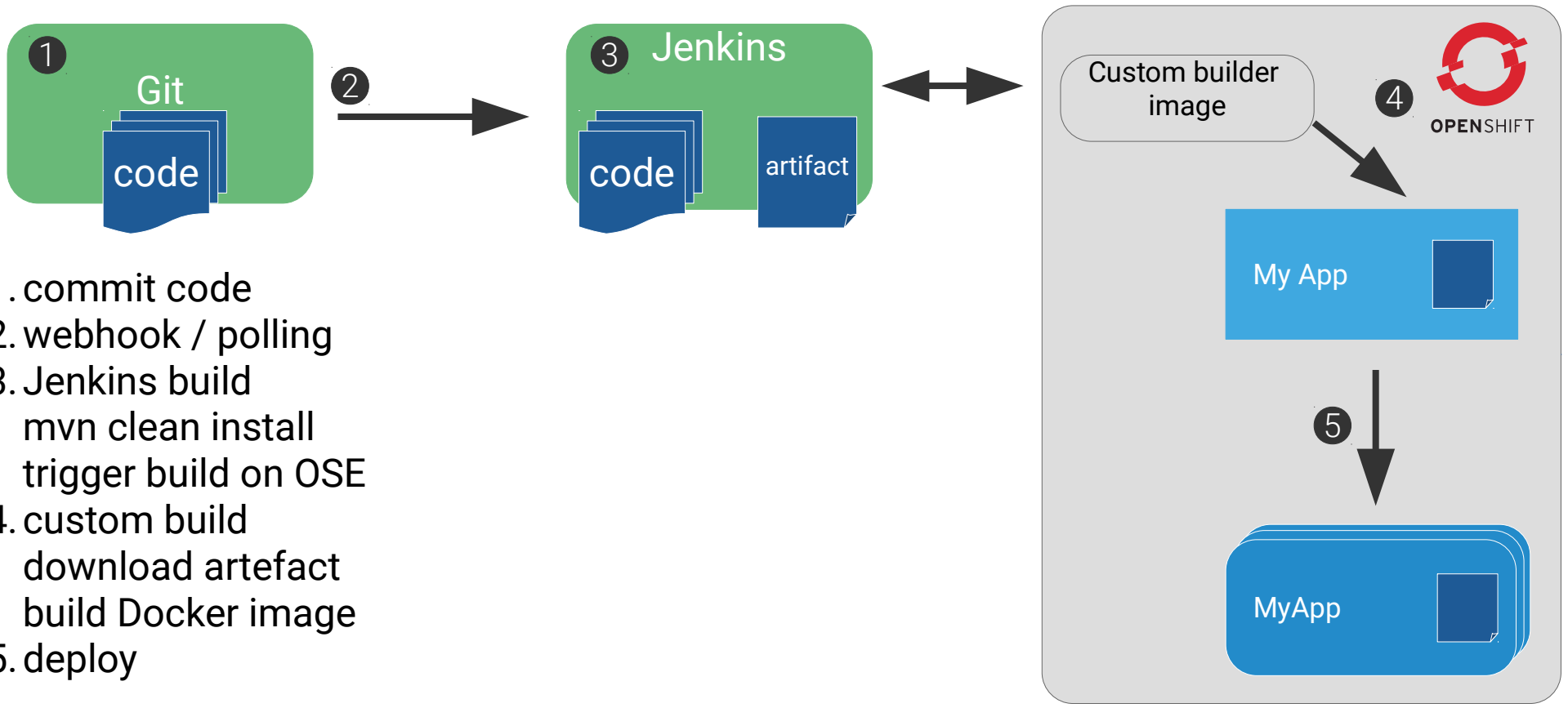


1. push Dockerfile to git
2. webhook
3. clone repo
4. build image
5. push to registry and deploy

Workflow: Docker Builds Beispiel

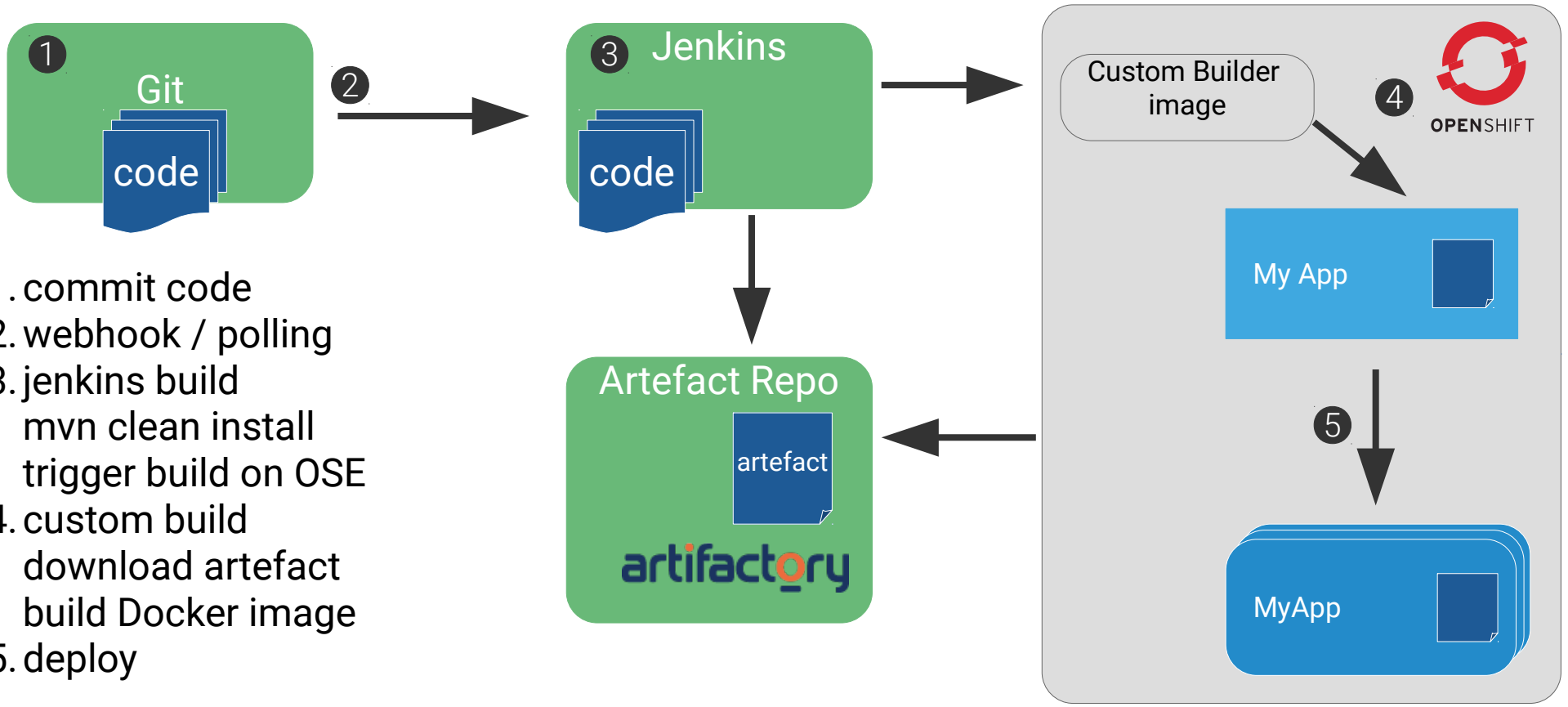
https://github.com/appuio/example-php-docker-helloworld

Workflow: Jenkins (1)



1. commit code
2. webhook / polling
3. Jenkins build
   mvn clean install
   trigger build on OSE
4. custom build
   download artefact
   build Docker image
5. deploy

Workflow: Jenkins (2)



1. commit code
2. webhook / polling
3. jenkins build
   mvn clean install
   trigger build on OSE
4. custom build
   download artefact
   build Docker image
5. deploy

2

CI / CD Workflow

Continuous Integration

- Versionsverwaltung
- kompilieren und testing
- Integration der Software
- Reporting
- Deployment auf Integration

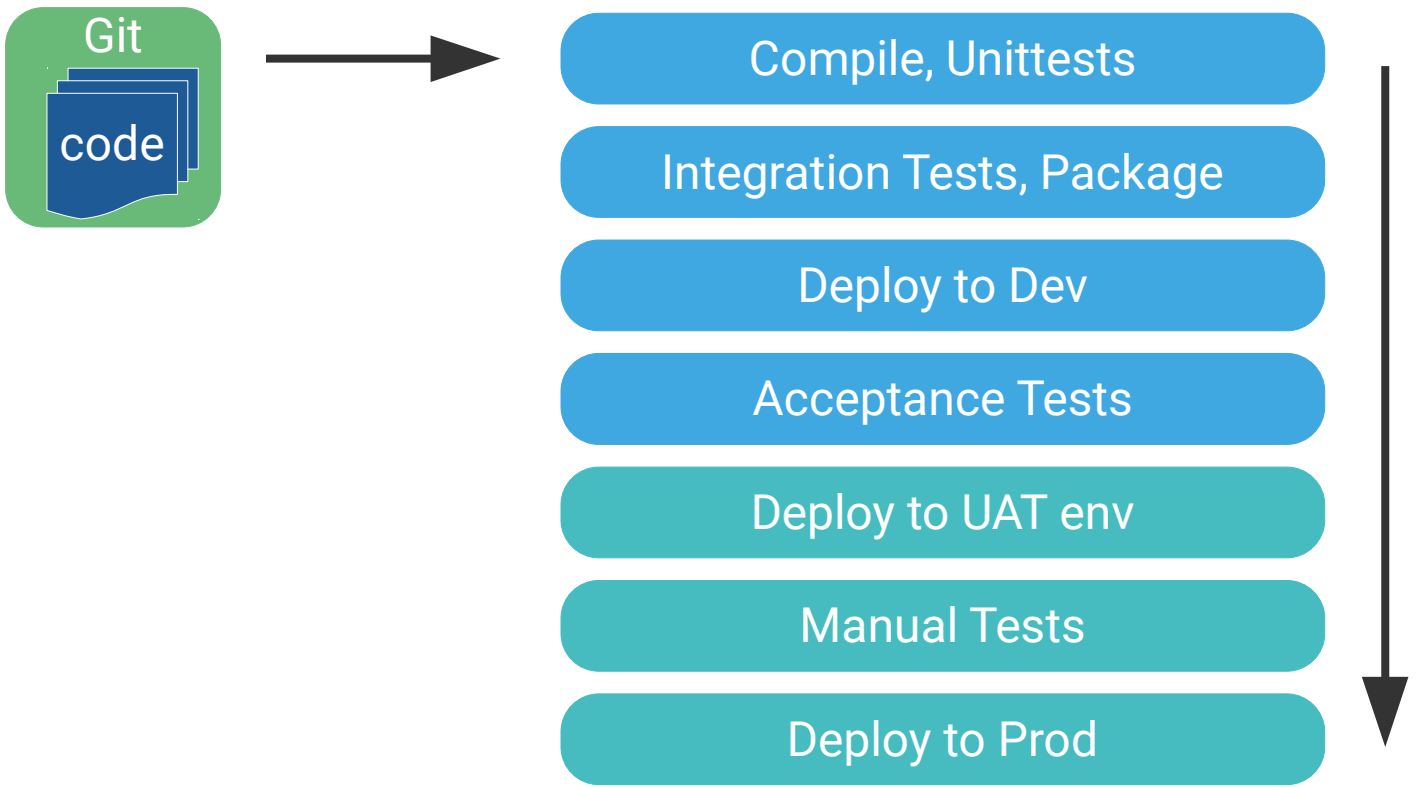
Continuous Delivery

- Erweiterung von CI
- Ein Artefakt für alle Umgebungen
- Voll automatisches Deployment auf **Knopfdruck**
- Prod Release auf **Knopfdruck**

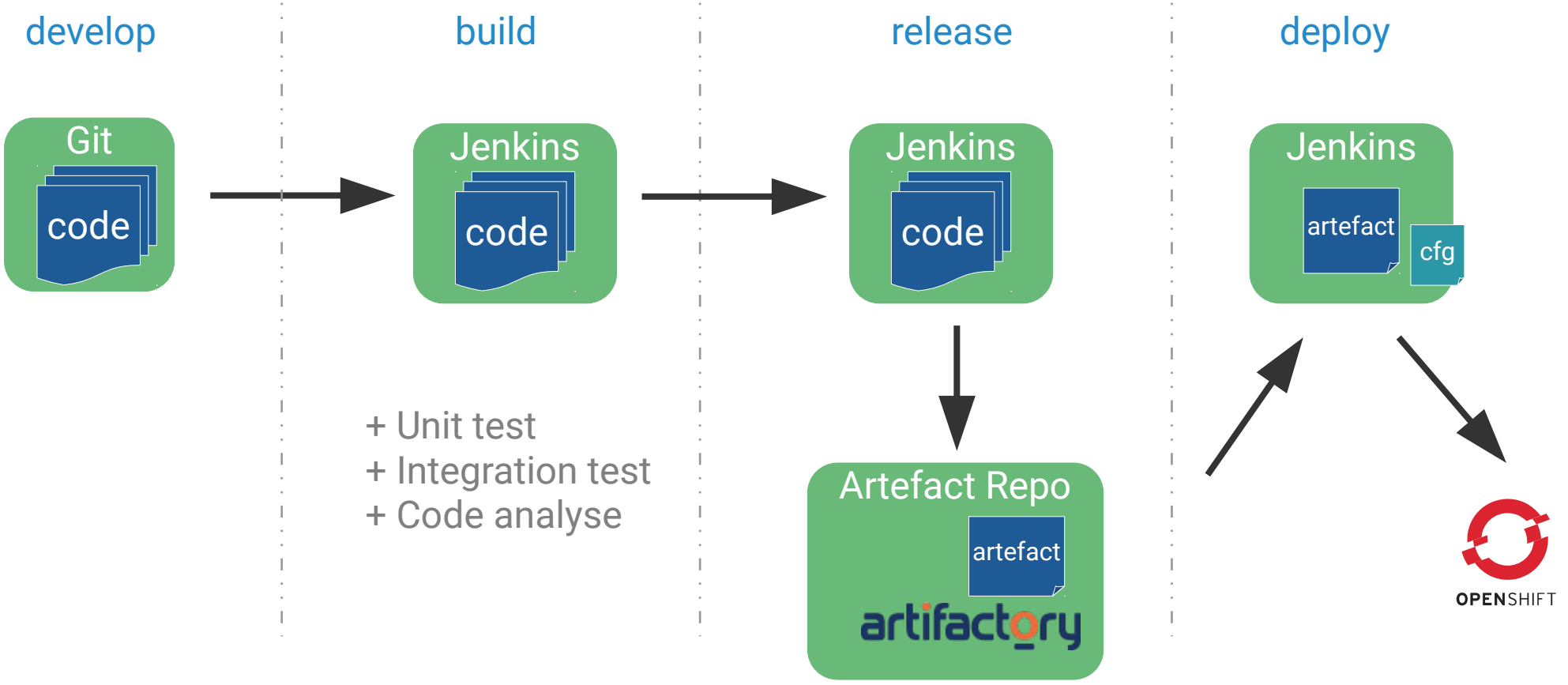
Continuous Deployment

- Jedes Release wird automatisch in Produktion deployed
- Unterbruchsfreihe Deployments nötig
- Blau - Grün Deployments

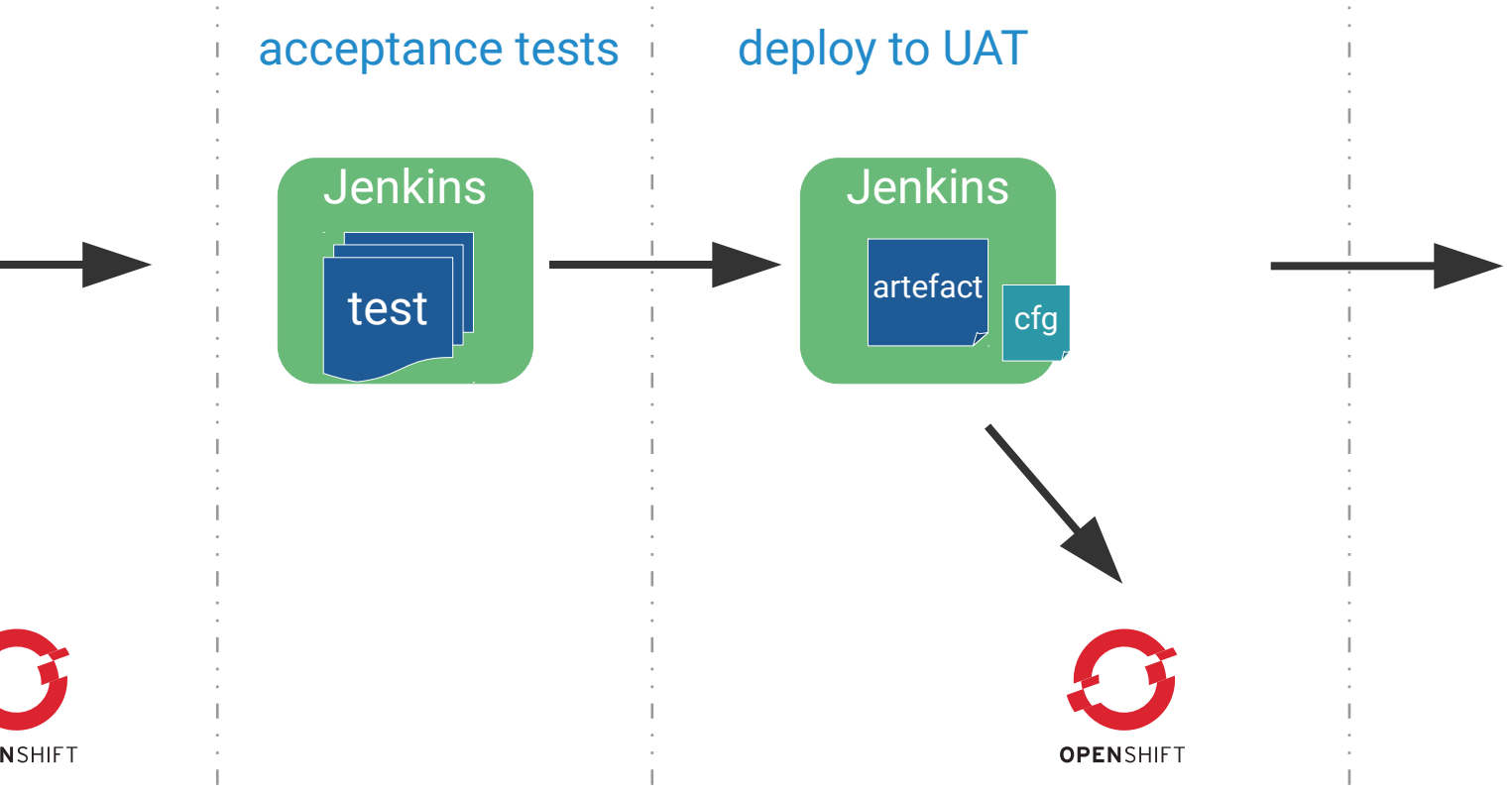
Toplevel CI / CD Workflow



Konkretes Beispiel Java mit Maven (1)



Konkretes Beispiel Java mit Maven (2)



Challenges

Maven / Artefaktrepository

Maven Version Mechanismus für Continuous Delivery nicht geeignet

Jeder Commit ist ein potentieller Release Kandidat kein SNAPSHOT

Maven release plugin in Lifecycle integriert.

mvn release:prepare → mvn release:perform

Viele Releases → Artefaktrepository clean up

Konfiguration

Java Property Files, liegen im Classpath

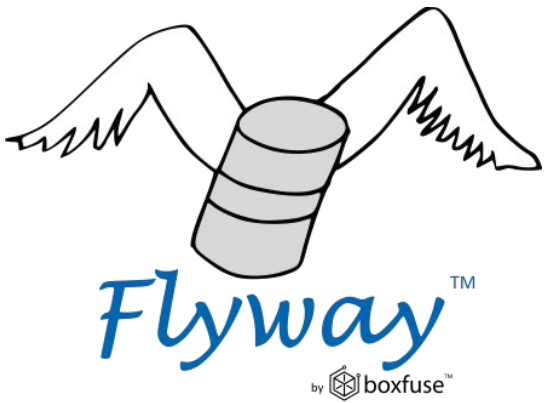
Das selbe Artefakt auf allen Umgebungen

Integration in bestehende Konfigurationsmanagement

Puppet / Ansible / AMW ist via Lifecycle Hooks während deployment integrierbar

Automated Database Migration

Database Migrations are a part of the delivered application and executed by an OpenShift Lifecycle Hook or the application itself.



LIQUIBASE

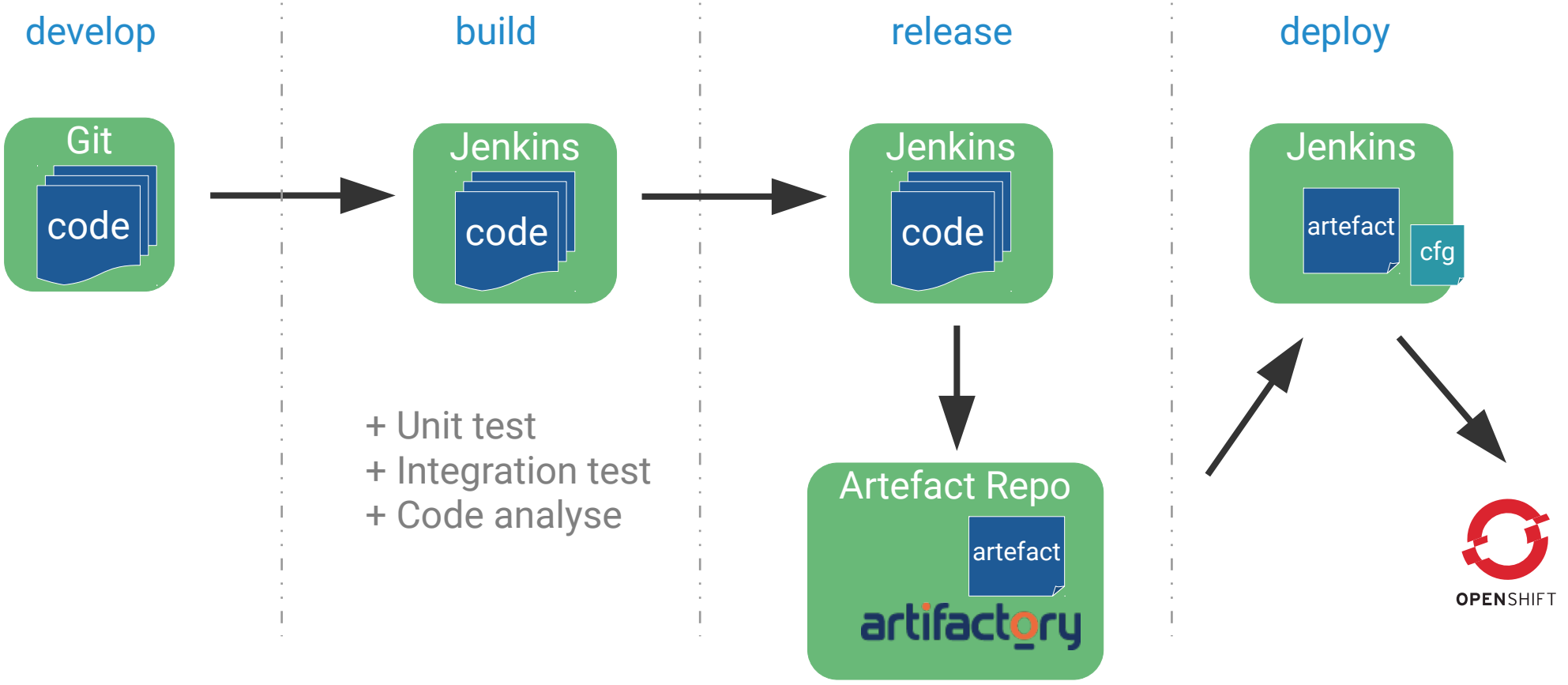
```
$ rake db:migrate
```

```
$ php bin/console doctrine:schema:update
```

3

Java EE Beispiel Pipeline

Konkretes Beispiel Java mit Maven



Pre-Build-Schritte

Maven Goals aufrufen

Maven-Version: (Standard)

Goals: versions:set -DnewVersion=1.0.$BUILD_NUMBER

Erweitert...

Löschen

Pre-Build-Schritt hinzufügen

Build

Maven-Version: Maven 3.2.1

Stamm-POM: pom.xml

Goals und Optionen: clean install

Erweitert...

Post-Build-Schritte

nur bei erfolgreichen Builds | nur bei erfolgreichen oder instabilen Builds | immer ausführen

Unter welchen Bedingungen sollen die Post-Build-Schritte ausgeführt werden?

Maven Goals aufrufen

Maven-Version: Maven 3.2.1

Pipeline als Jobs

S	W	Name
		PITC-tphilipona build development
		PITC-tphilipona deploy development
		PITC-tphilipona deploy integration development
		PITC-tphilipona release development

Symbol: S M L

Jenkins Pipeline Plugin als Alternative

Build Jobs als Groovy Scripts implementieren.

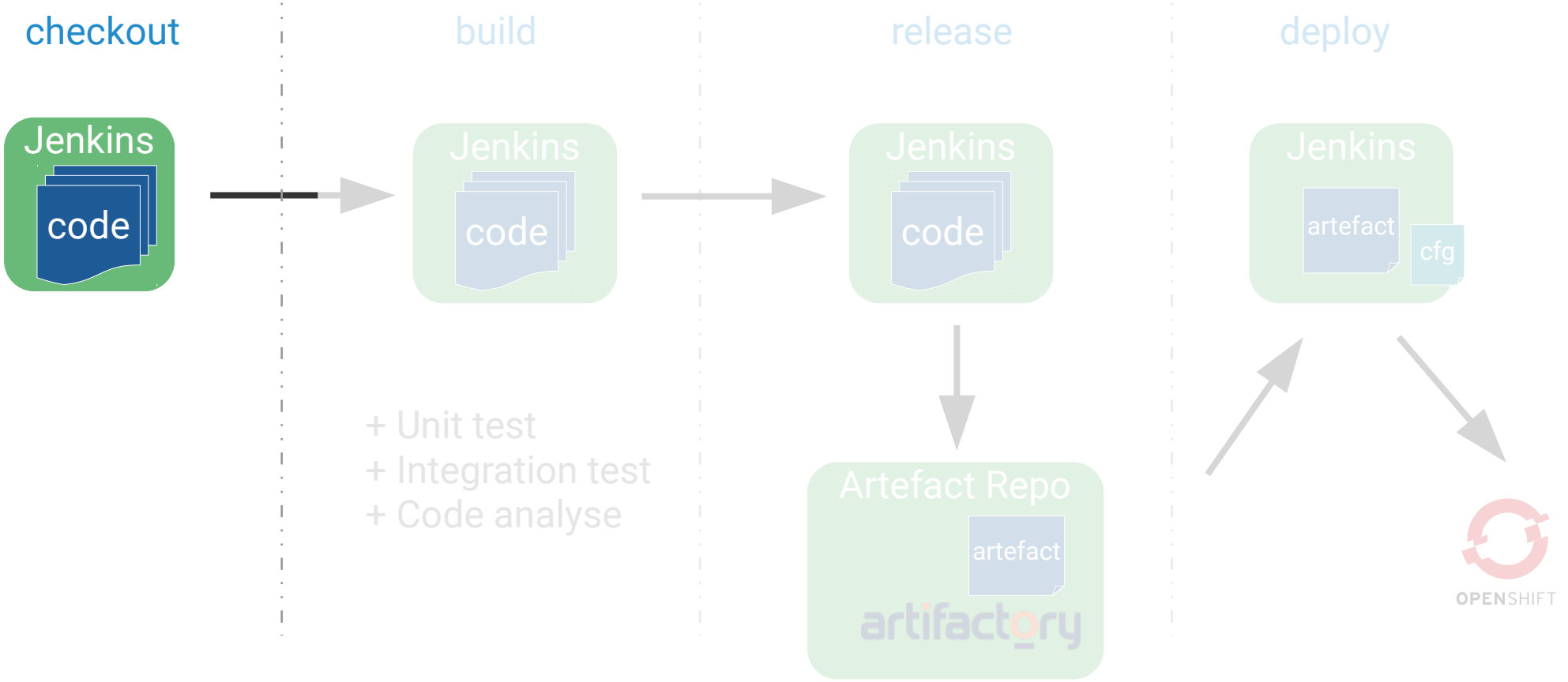
Build Job liegt im Git Repository neben dem Code

Jenkins verfügt nur noch über die Umgebungsinfo

- Git / Artifactory / OpenShift Accounts

https://wiki.jenkins-ci.org/display/JENKINS/Pipeline+Plugin

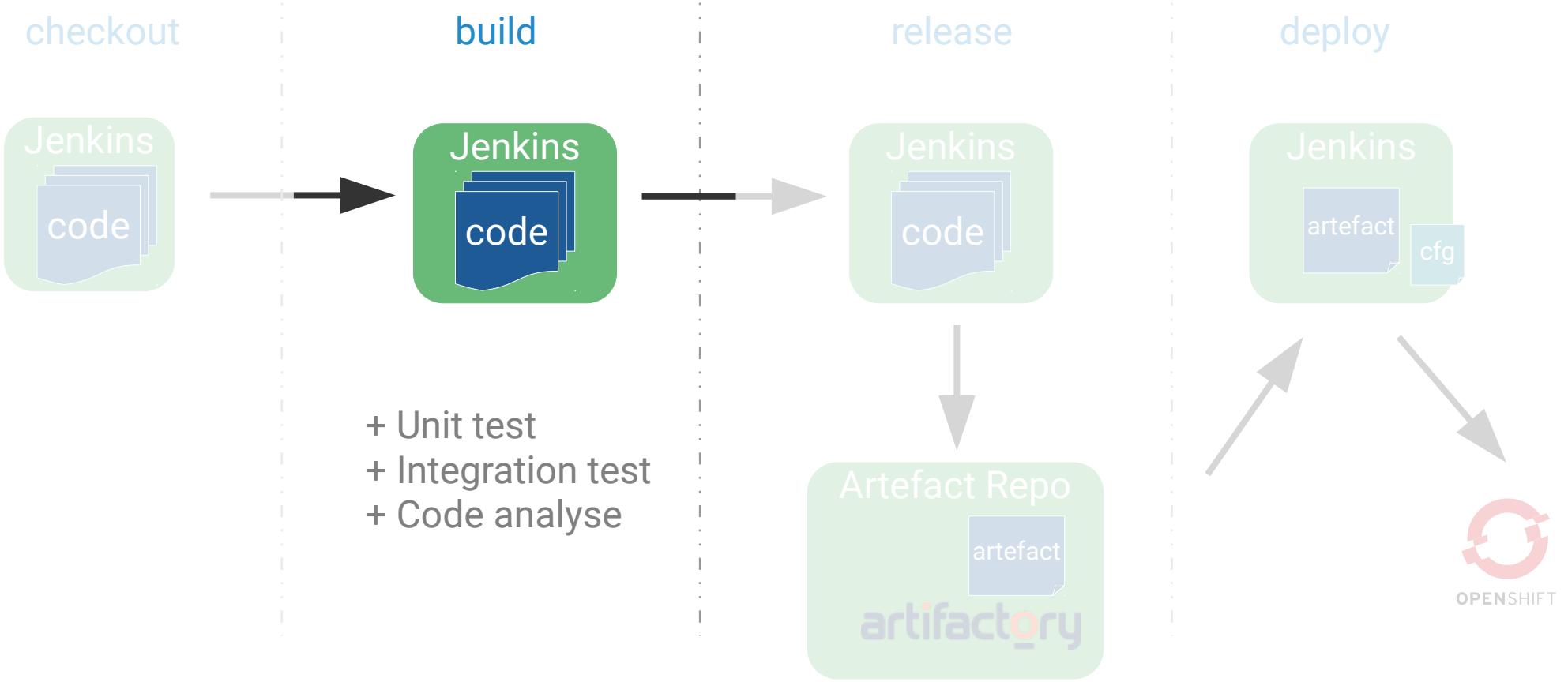
Stage 1


checkout
build
release
deploy
Jenkins
code
Jenkins
code
Jenkins
code
Jenkins
artefact
cfg
+ Unit test
+ Integration test
+ Code analyse
Artefact Repo
artefact
artifactory
OPENSHIFT

Stage Checkout

```
// configure Tools and Environment.
def mvnHome = tool 'maven33'
def javaHome = tool 'jdk8_openjdk'
def buildVersion = "1.0.$env.BUILD_NUMBER"

// Stage 1
stage 'Checkout'
// checkout Code
git url: 'https://gitlab.puzzle.ch/tphilipona/appuio-javaee-example.git'
```

Stage 2


checkout
build
release
deploy
Jenkins
code
Jenkins
code
Jenkins
code
Jenkins
artefact
cfg
+ Unit test
+ Integration test
+ Code analyse
Artefact Repo
artefact
artifactory
OPENSHIFT

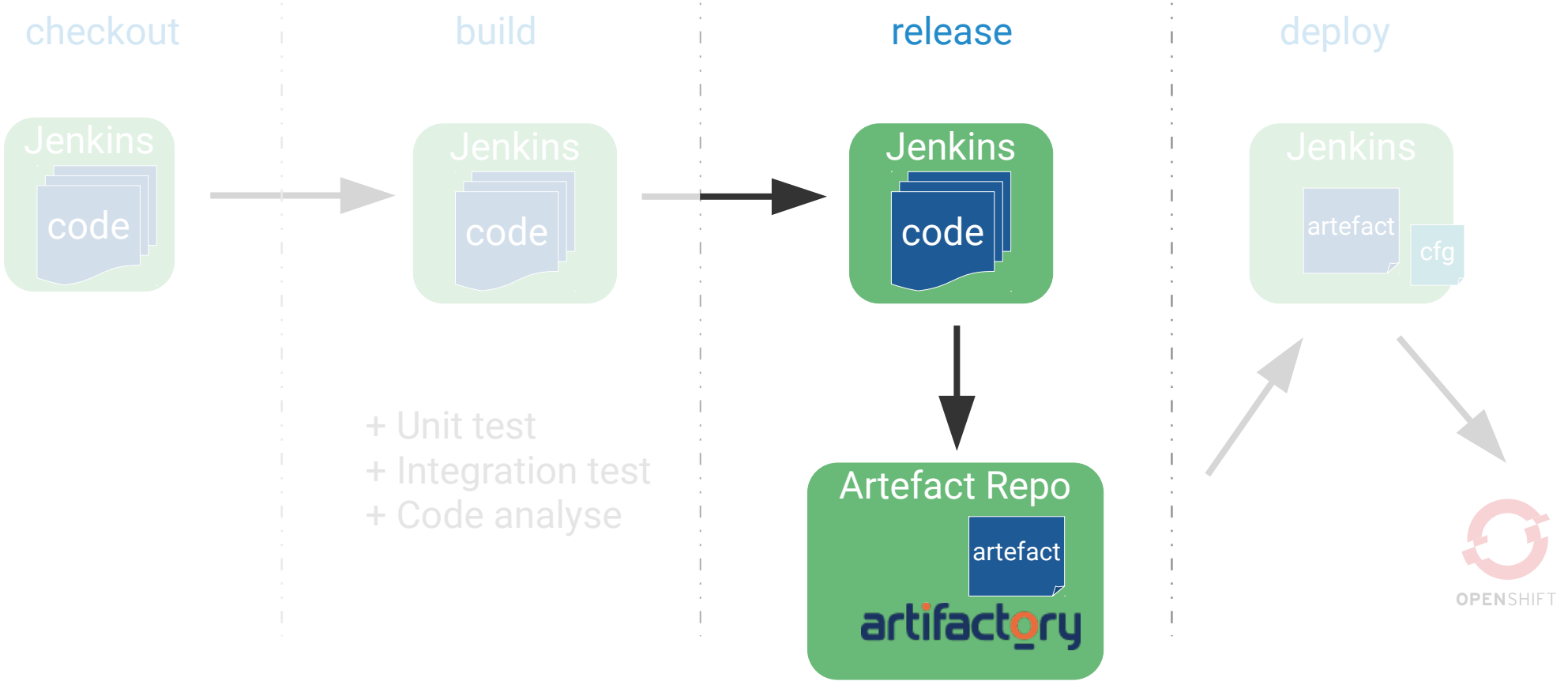
Pipeline Stage Build

```
stage 'Build'

sh "${mvnHome}/bin/mvn versions:set -DnewVersion=${buildVersion}"

sh "${mvnHome}/bin/mvn clean install"


// prepare Dockerfile and Context

sh "tar cvfz docker-context.tar.gz Dockerfile target/javaee7-
wildfly-example.war"

archive 'docker-context.tar.gz'
```

Dockerfile

```
FROM openshift/wildfly-100-centos7


ADD ./target/javaee7-wildfly-example.war
/wildfly/standalone/deployments/ROOT.war


CMD $STI_SCRIPTS_PATH/run
```

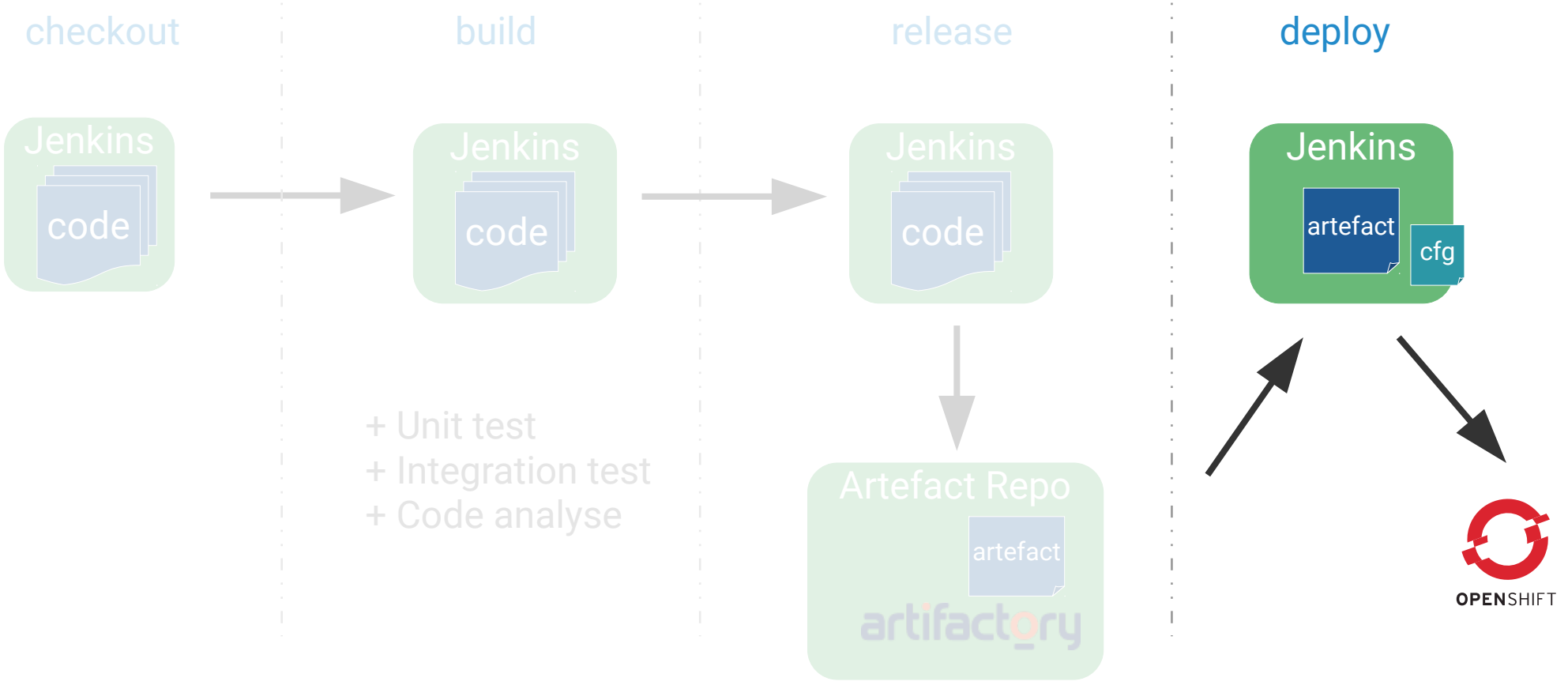
Stage 3


checkout
build
release
deploy
Jenkins
code
Jenkins
code
Jenkins
code
Jenkins
artefact
cfg
+ Unit test
+ Integration test
+ Code analyse
Artefact Repo
artefact
artifactory
OPENSHIFT

Pipeline Stage Release

```
stage 'Release'

// upload to artifactory

sh "${mvnHome}/bin/mvn deploy:deploy-file
-Durl=https://artifactory.puzzle.ch/artifactory/libs-release-local
-Dfile=target/javaee7-wildfly-example.war -DrepositoryId=puzzle-releases
-DgroupId=ch.appuio.example -DartifactId=javaee7-wildfly-example
-Dversion=${buildVersion}"

// push to releaseBranch

sh """
 git checkout -b release${buildVersion}
 git add pom.xml
 git commit -m 'Released Version: ${buildVersion}'
 git push -u origin release${buildVersion}

"""
```

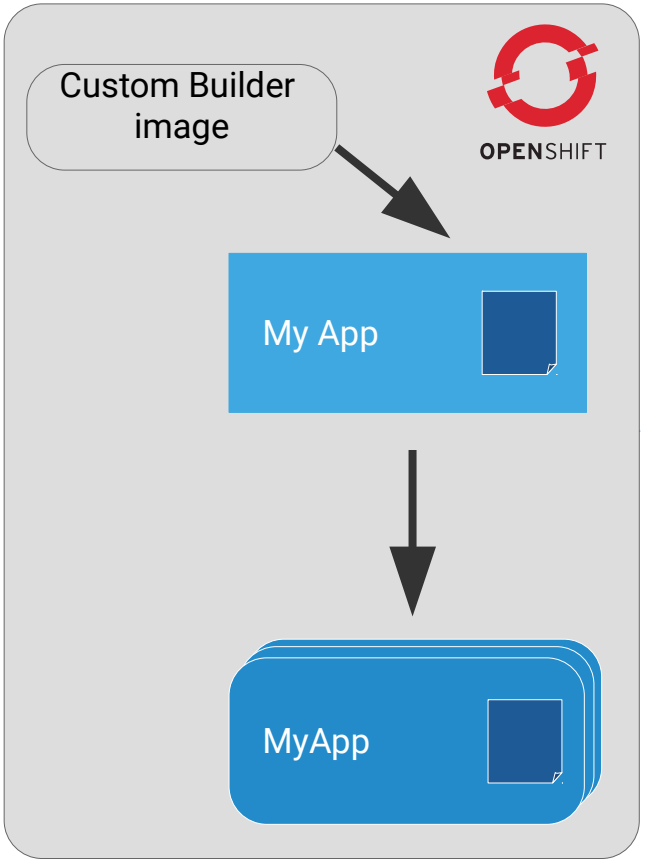
Stage 4


checkout
build
release
deploy
Jenkins
code
Jenkins
code
Jenkins
code
+ Unit test
+ Integration test
+ Code analyse
Artefact Repo
artefact
artifactory
Jenkins
artefact
cfg
OPENSHIFT

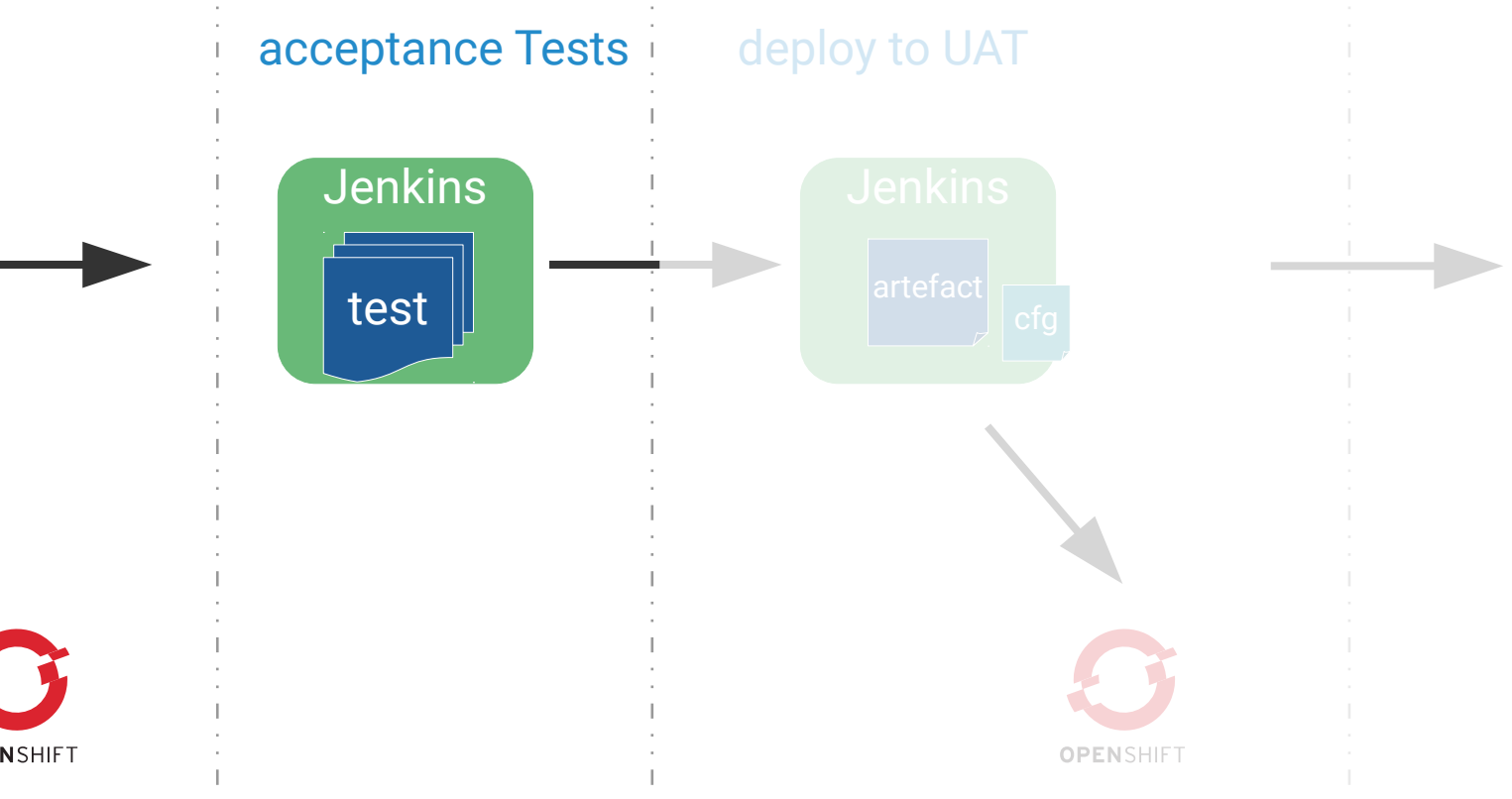
Pipeline Stage deploy

```
// Stage Deploy

stage 'deploy'

step([$class: 'OpenShiftBuilder',
    apiURL: 'https://victory.rz.puzzle.ch:8443',
    authToken: env.openshift_password,
    bldCfg: 'appuiojavaee7test', buildName: '',
    checkForTriggeredDeployments: 'false', commitID: "$env.BUILD_NUMBER",
    namespace: 'appuio-javaee7-example', showBuildLogs: 'true', verbose:
'false'])
```

Pipeline Stage deploy



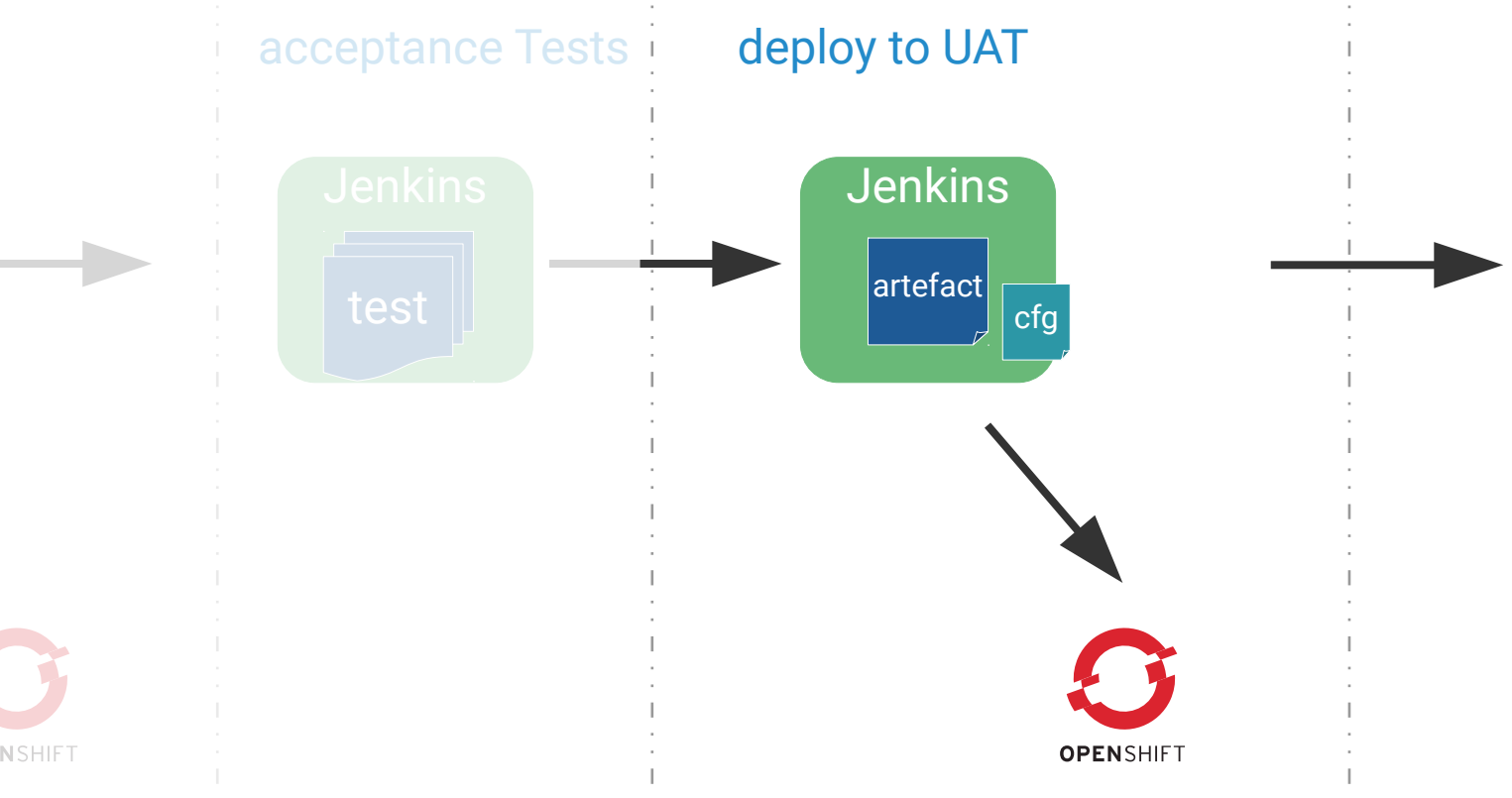
Stage 5


acceptance Tests
deploy to UAT
Jenkins
test
Jenkins
artefact
cfg
OPENSHIFT

Pipeline Stage acceptance tests

```
// Stage acceptance tests

stage 'acceptance tests'


sh "${mvnHome}/bin/mvn -f javaee-example-acceptancetests/pom.xml
-Dch.appuio.example.javaeetest.backendurl=$backend clean test"
```

Stage 6


acceptance Tests
deploy to UAT
Jenkins
test
Jenkins
artefact
cfg
OPENSHIFT

Stage View


checkout
build
release
deploy
acceptance tests
deploy to uat
smoke tests
Average stage times:
(Average full run time: ~1min 5s)
409ms
9s
5s
21s
6s
13s
6s
#60
Apr 14
10:05
16 commits
390ms
9s
6s
21s
6s
13s
6s
#59
Apr 14
10:03
15 commits
441ms
9s
5s
19s
failed
#58
Apr 14
09:59
13 commits
328ms
9s
5s
23s
6s
13s
6s
#57
Apr 14
09:57
479ms
9s
6s
23s
6s
14s
6s
sentinel

Demo

https://jenkins.puzzle.ch/job/PITC-tphilipona-testpipeline

4

Fazit

Fazit (1/3)

Schnelle Workflows mit OpenShift möglich.

Komplett Automatisierte Workflows von A - Z

OSE 3 als Basis für Continuous Integration und Delivery geeignet.

Automatisierung ist der Schlüssel zum Erfolg.

Fazit (2/3)

So viel wie möglich automatisieren

Das Artefakt wird einmal gebuildet und durch die Pipeline gegeben.

Artefakte beinhalten keine umgebungsspezifische Konfiguration

Woher kommt die Umgebungsspezifische Konfig?

Fazit (3/3)

Continuous Delivery: Jeder Commit ist ein potentielles Release

Pipeline muss schnell sein und rasch Feedback geben

Redundanzen vermeiden

Resources

DataSheet:

https://www.redhat.com/en/files/resources/cl-openshift-enterprise-3-red-hat-inc0328839mm-201512.pdf

Dokumentation:

https://docs.openshift.com

Resources:

https://www.openshift.com/enterprise/resources.html

Getting Started:

https://docs.openshift.com/enterprise/3.1/cli_reference/index.html

Q & A

Thank you!