gesis

Leibniz-Institut
für Sozialwissenschaften

Wie wir unser SSOAR.info mit Continuous Delivery aktuell halten

Kopfverschmerzfreie DSpace-Modifikationen mit Maven Overlays

Dipl.-Ing. Gerrit Hübbers, GESIS

Mitglied der Leibniz-Gemeinschaft

Maven Overlays

DSpaces Maven-Setup bietet bereits einen Overlay-Mechanismus an:

`./pom.xml` bindet
`./dspace/pom.xml` ein.

`./dspace/pom.xml` bindet
`./dspace/modules/pom.xml` ein.

`./dspace/modules/pom.xml` bindet zum Beispiel
`./dspace/modules/xmlui/pom.xml` ein.

./dspace/modules/xmlui/pom.xml

```
<overlay>
   <groupId>org.dspace</groupId>
   <artifactId>dspace-xmlui</artifactId>
   <type>war</type>
   <excludes>
      <exclude>WEB-INF/classes/**</exclude>
   </excludes>
</overlay>
```

Maven Overlays – Konzept

* Maven Overlays nehmen das spezifizierte Ursprungs-Maven-Artefakt,
* entpacken es,
* und *pasten* die Build-Artefakte des aktuellen Projektes *darüber* – ggf. gleichnamige Ursprungsdateien werden überschrieben

→ ein Maven-Overlay-Projekt überlagert also die Dateien des spezifizierten Ursprungs-Maven-Artefakts.

Maven Overlays

`./dspace-xmlui/**` → `./dspace/modules/xmlui/**`

`./dspace-rest/**` → `./dspace/modules/rest/**`

`./dspace-oai/**` → `./dspace/modules/oai/**`

`./dspace-api/**` → `./dspace/modules/additions/**`

`./dspace-services/**` → auch `./dspace/modules/additions/**`

Generell:

`./dspace-foo/**` → `./dspace/modules/foo/**`

Entwicklung auf dem Entwickler*-PC

Import des Projektes in Eclipse als Maven-Projekt. In Eclipse wird programmiert und gedebuggt.

Builden und Hochfahren der SSOAR-Instanz mit Cygwin:

```
pushd . && cd ${SSOAR_SOURCE_DIR} && stop-local-ssoar.sh && mvn package -Denv=$(whoami) -P \!dspace-jspui,\!dspace-lni,\!dspace-sword,\!dspace-swordv2,\!dspace-rdf && cd dspace/target/ssoar-installer/ && ant update && ant clean_backups && start-local-ssoar.sh && popd
```

start-local-ssoar.sh

```
export CATALINA_BASE="${HOME}/tomcat-instances/ssoar-$(whoami)"

# JMX aktivieren
export CATALINA_OPTS="-Xmx1024m -Xms1024m -XX:MaxPermSize=512m
-Dcom.sun.management.jmxremote
-Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote.ssl=false
-Dcom.sun.management.jmxremote.port=9000"

export JAVA_OPTS="-Xmx1024m -Xms1024m -XX:MaxPermSize=512m"

# Debugierbarkeit (?) aktivieren 1
export JPDA_ADDRESS=8000
export JPDA_TRANSPORT=dt_socket

# Debugierbarkeit aktivieren 2
/bin/sh "${CATALINA_HOME}"/bin/catalina.sh jpda start
```

stop-local-ssoar.sh

```
#!/bin/sh

CATALINA_BASE="${HOME}/tomcat-instances/ssoar-$(whoami)"
export CATALINA_BASE
/bin/sh "${CATALINA_HOME}"/bin/shutdown.sh
```

Entwicklung auf dem Entwickler*-PC

Wir verwenden eine (1) Tomcat-Distribution auf den Entwickler-PCs am Speicherort `${CATALINA_HOME}` .

Durch das Setzen von `${CATALINA_BASE}` kann man eine Tomcat-Distribution für beliebig viele Tomcat-Instanzen nebenläufig verwenden.

Entwicklung auf dem Entwickler*-PC

Debugging in Eclipse per Debug Configuration → Remote Java Application → localhost:8000

Breakpoints, Step-Debugging funktionieren prima.

JMX-Beans auf Port localhost:9000, zum Beispiel mit Java VisualVM.

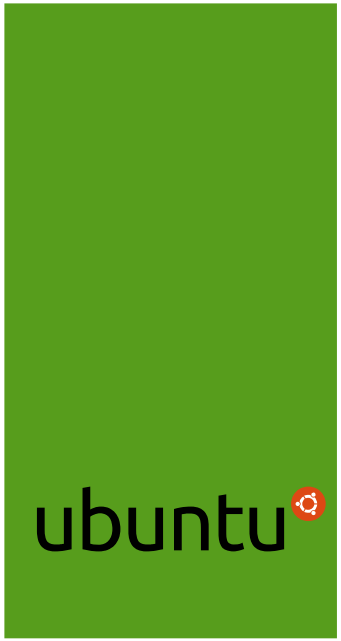
Entwicklung auf dem Entwickler*-PC

Debugging und JMX funktionieren auch in Staging- und Produktiv-Umgebung.

Dafür von Entwickler*-PC SSH-Port-Forwarding-Tunnel zum zu verbindenen Server aufbauen:

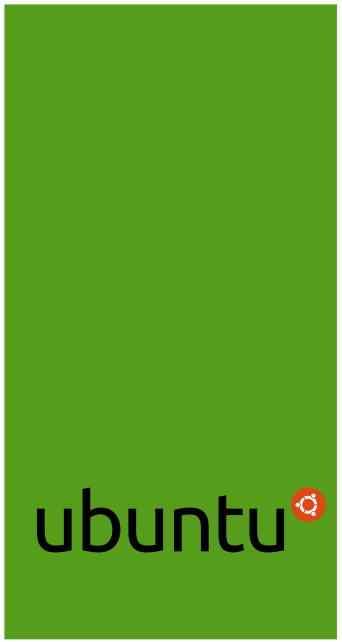
```
ssh -L 8000:localhost:8000 -L
9000:localhost:9000 me@dspace.example.com
```

SSOAR-Deployment-Pipeline



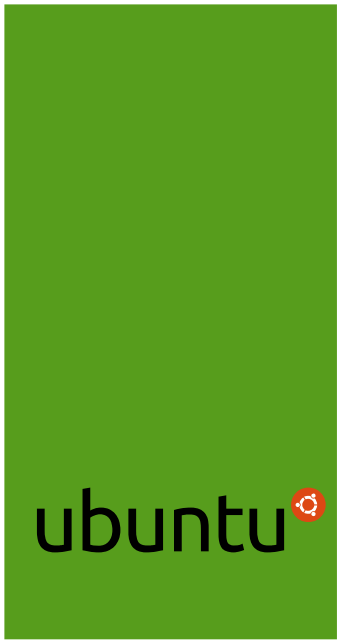
Entwickler*-PC | **Gitlab** | **Jenkins** | **Deployment Staging** | **Deployment Production**

git commit



Entwickler*-PC | **Gitlab** | **Jenkins** | **Deployment Staging** | **Deployment Production**

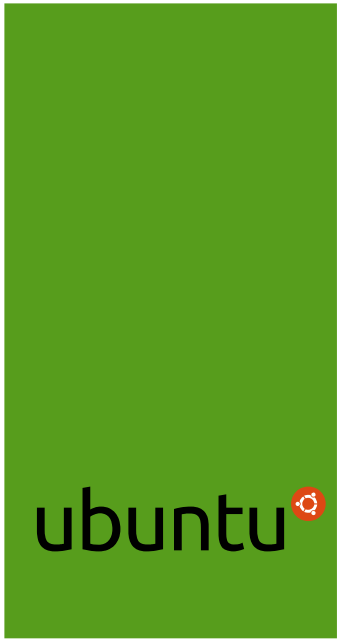
git push



Entwickler*-PC | **Gitlab** | **Jenkins** | **Deployment Staging** | **Deployment Production**

Gitlab an Jenkins:

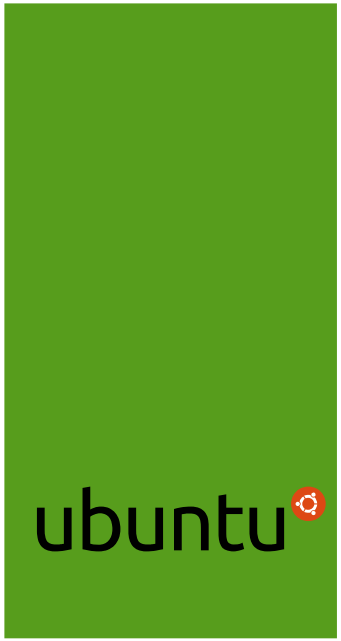
„Hey, ein neuer Commit ist da!“



Entwickler*-PC | **Gitlab** | **Jenkins** | **Deployment Staging** | **Deployment Production**

Jenkins an Gitlab:

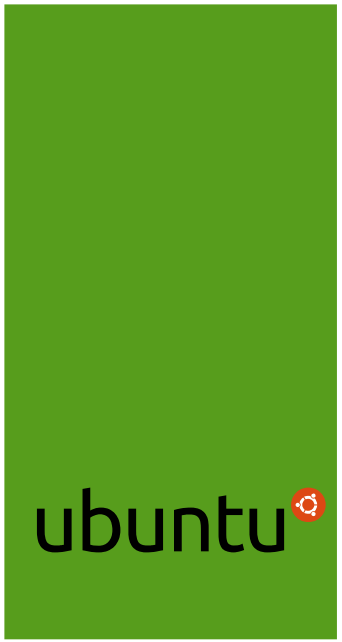
„Okay, ich pulls mir“



Entwickler*-PC | **Gitlab** | **Jenkins** | **Deployment Staging** | **Deployment Production**

Jenkins

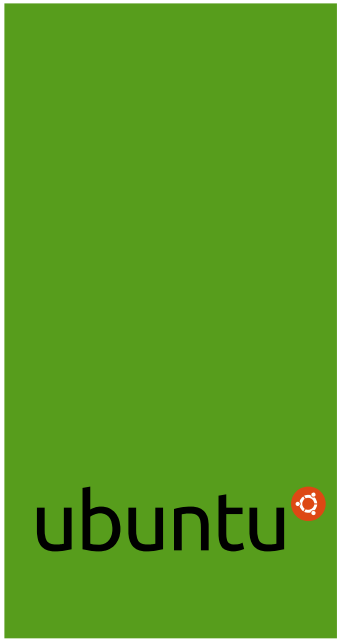
bau
test
beurteil

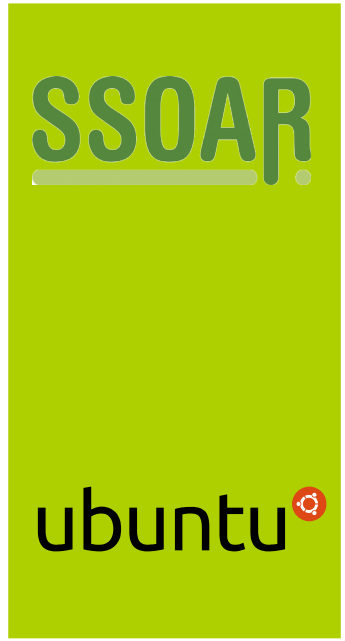


Entwickler*-PC | **Gitlab** | **Jenkins** | **Deployment Staging** | **Deployment Production**

Jenkins

Sieht gut aus! Ich deploye auf Staging. →



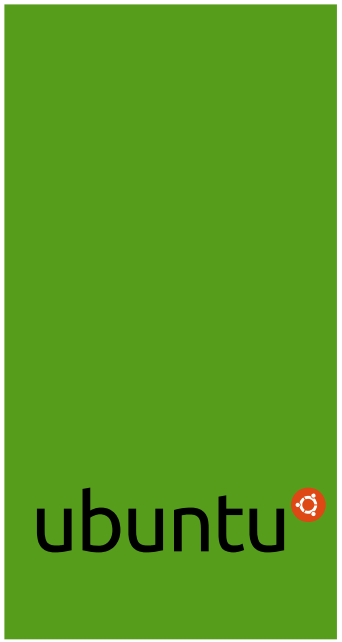


Entwickler*-PC | **Gitlab** | **Jenkins** | **Deployment Staging** | **Deployment Production**

Jenkins dann auf Staging:

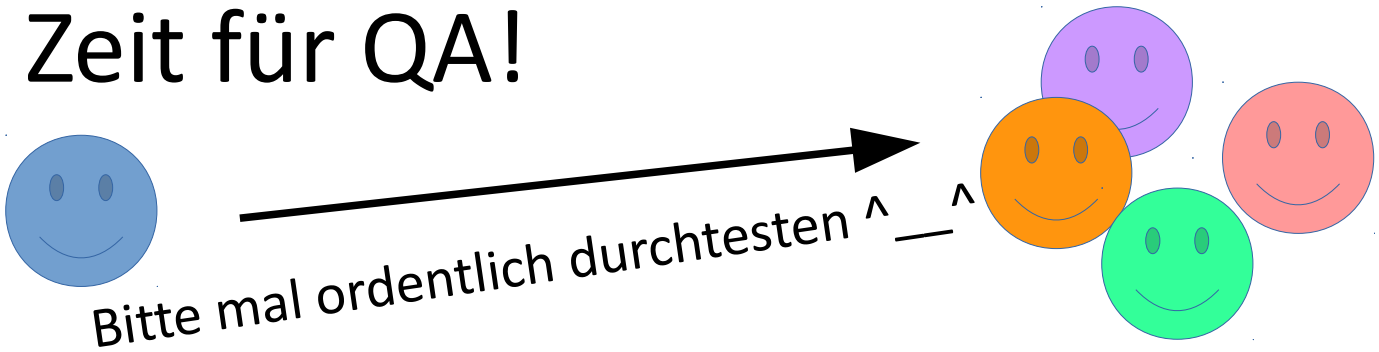
```
sudo service ssoar start
```

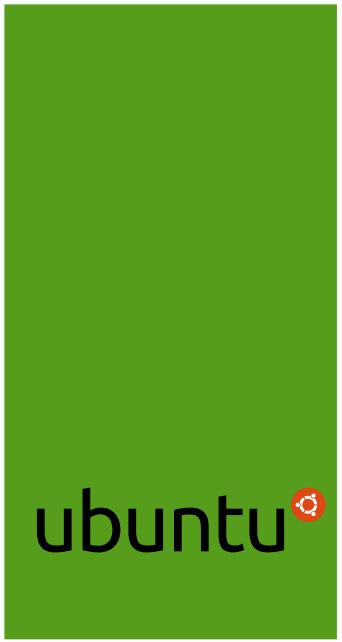
→



Entwickler*-PC | **Gitlab** | **Jenkins** | **Deployment Staging** | **Deployment Production**

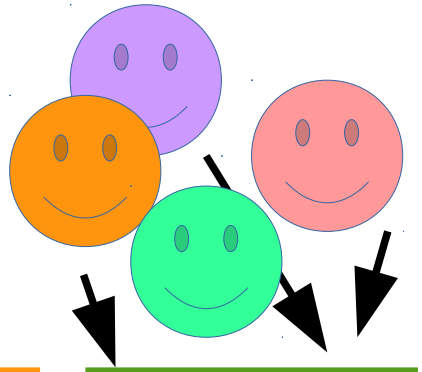
Zeit für QA!

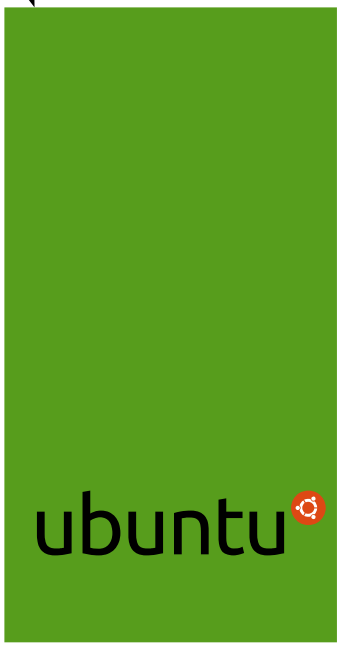




Entwickler*-PC | **Gitlab** | **Jenkins** | **Deployment Staging** | **Deployment Production**

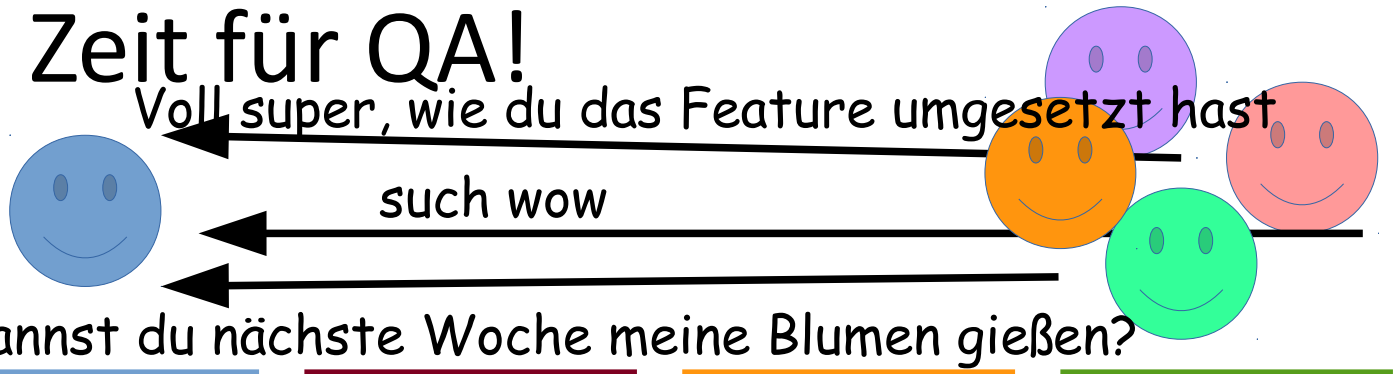
Zeit für QA!

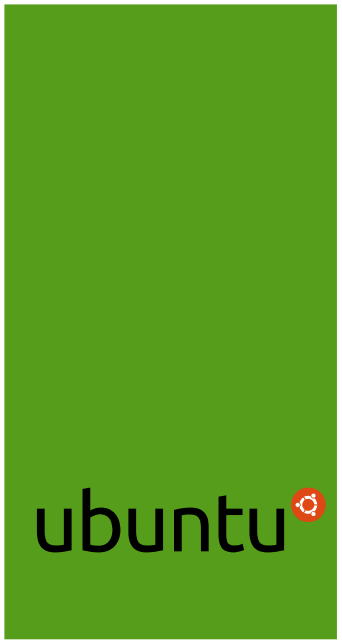




Entwickler*-PC | **Gitlab** | **Jenkins** | **Deployment Staging** | **Deployment Production**

Zeit für QA!





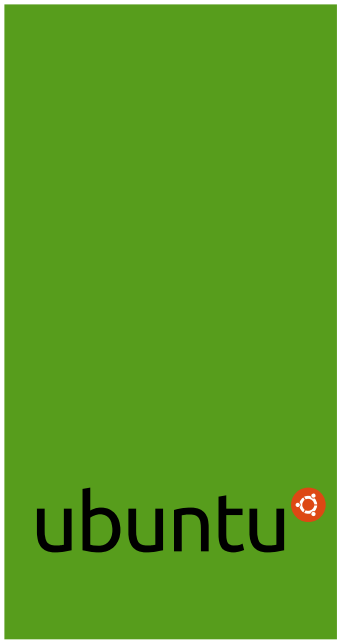
Entwickler*-PC | **Gitlab** | **Jenkins** | **Deployment Staging** | **Deployment Production**

Zeit für QA!

Entwickler*-PC | **Gitlab** | **Jenkins** | **Deployment Staging** | **Deployment Production**

Deployment auf ssoar.info

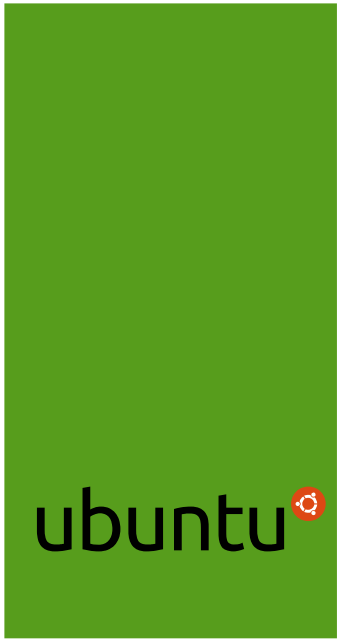
promote mir diesen Commit nach ssoar.info!

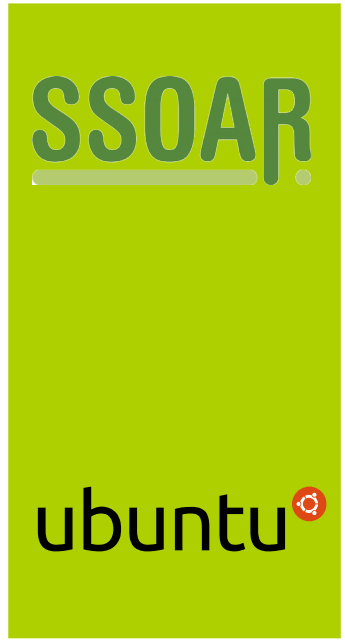


Entwickler*-PC | **Gitlab** | **Jenkins** | **Deployment Staging** | **Deployment Production**

Deployment auf ssoar.info

*mit
ssoar-prod.properties
neubau*

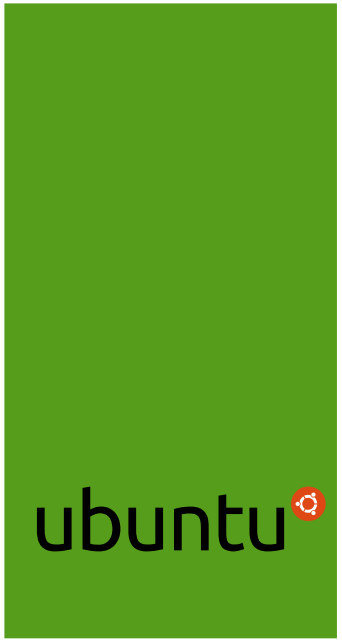




Entwickler*-PC | **Gitlab** | **Jenkins** | **Deployment Staging** | **Deployment Production**

Deployment auf ssoar.info

```
scp dspace-installer
&& ssh ssoar.info
&& sudo service ssoar stop
&& ant update
&& sudo service ssoar start
```



Entwickler*-PC | **Gitlab** | **Jenkins** | **Deployment Staging** | **Deployment Production**