Jenkins Showcase

Wie Bareos-Pakete entstehen

Andreas Rogge - Bareos GmbH & Co. KG – www.bareos.com

Wer bin ich?

Andreas Rogge

- hauptberuflicher Bareos-Entwickler
- Langzeit-Nutzer von Bareos
- ursprünglich Systemadministrator

Wo arbeite ich?

Bareos GmbH & Co. KG

- finanziert das Bareos Projekt
- stellt Dienstleistungen rund um Bareos bereit
  - Binärpakete mit Langzeit-Support
  - technischer Kundendienst
  - Schulungen
  - Partnerschaften mit Dienstleistern vor Ort

Übersicht

- Worum geht es überhaupt?
- Wie geht das manuell?
- und in Jenkins?
- Lessons learned

Continous Delivery

Continous Delivery

- automatische Build-Pipeline

Continous Delivery

- automatische Build-Pipeline
- schnelles, verlässliches Feedback

Continous Delivery

- automatische Build-Pipeline
- schnelles, verlässliches Feedback
- kleinere Entwicklungsinkremente

Continous Delivery

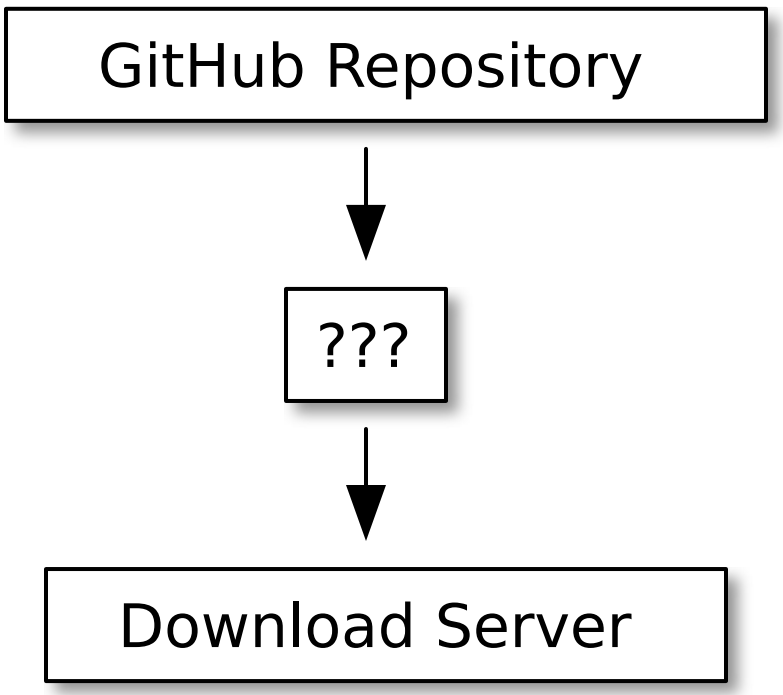
- automatische Build-Pipeline
- schnelles, verlässliches Feedback
- kleinere Entwicklungsinkremente
- stets ein Release möglich

Continous Delivery

- automatische Build-Pipeline
- schnelles, verlässliches Feedback
- kleinere Entwicklungsinkremente
- stets ein Release möglich

macht Releases schnell, einfach und langweilig

Aufgabenstellung



Was ist Bareos?

Was ist Bareos?

- Backup-Software

Was ist Bareos?

- Backup-Software
  - Daemons

Was ist Bareos?

- Backup-Software
  - Daemons
  - Plugins

Was ist Bareos?

- Backup-Software
  - Daemons
  - Plugins
  - Tools

Was ist Bareos?

- Backup-Software
  - Daemons
  - Plugins
  - Tools
- Tech-Stack

Was ist Bareos?

- Backup-Software
  - Daemons
  - Plugins
  - Tools
- Tech-Stack
  - C/C++

Was ist Bareos?

- Backup-Software
  - Daemons
  - Plugins
  - Tools
- Tech-Stack
  - C/C++
  - Python

Was ist Bareos?

- Backup-Software
  - Daemons
  - Plugins
  - Tools
- Tech-Stack
  - C/C++
  - Python
  - CMake

Was ist Bareos?

- Backup-Software
  - Daemons
  - Plugins
  - Tools
- Tech-Stack
  - C/C++
  - Python
  - CMake
  - PostgreSQL

Unterstützte Plattformen

- RHEL 8/9/10
- Fedora 40/41/42
- SUSE 15
- Debian 11/12/13
- Ubuntu 20/22/24 LTS
- Universeller Linux-Client
- FreeBSD
- macOS
- Microsoft Windows
- ~~Solaris~~
- ~~AIX~~
- ~~HP-UX~~

Entwicklungsprozess

Entwicklungsprozess

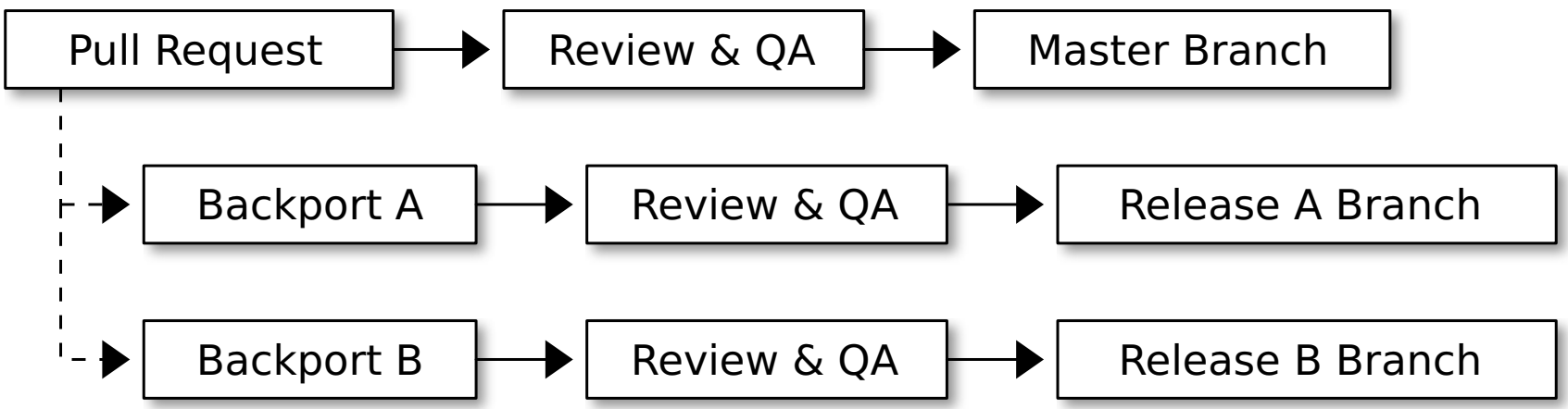
- GitHub Flow

Entwicklungsprozess

- GitHub Flow
- Release-Branches und Backports

Entwicklungsprozess

- GitHub Flow
- Release-Branches und Backports



Wie baut man Bareos?

```
$ git clone -q git@github.com:bareos/bareos.git
$ cd bareos
$ cmake -S . -B builddir
...
$ cmake --build builddir
...
$ cd builddir
$ ctest -V -S CTestScript.cmake
...
$ sudo cmake --install .
```

Wie wird paketiert?

RPM

rpmbuild

DEB

dpkg-buildpackage

FreeBSD

make package

MacOS

CPack

Windows

NSIS

Solaris

IPS

Was Jenkins leistet

Was Jenkins leistet

- native Pakete für alle Plattformen erstellen

Was Jenkins leistet

- native Pakete für alle Plattformen erstellen
- Dokumentation generieren

Was Jenkins leistet

- native Pakete für alle Plattformen erstellen
- Dokumentation generieren
- Aufwändige Tests ausführen

Was Jenkins leistet

- native Pakete für alle Plattformen erstellen
- Dokumentation generieren
- Aufwändige Tests ausführen
- Repository-Metadaten erzeugen

Was Jenkins leistet

- native Pakete für alle Plattformen erstellen
- Dokumentation generieren
- Aufwändige Tests ausführen
- Repository-Metadaten erzeugen
- Paketinstallationen testen

Was Jenkins leistet

- native Pakete für alle Plattformen erstellen
- Dokumentation generieren
- Aufwändige Tests ausführen
- Repository-Metadaten erzeugen
- Paketinstallationen testen
- fertige Pakete / Dokumentation hochladen

Was Jenkins leistet

- native Pakete für alle Plattformen erstellen
- Dokumentation generieren
- Aufwändige Tests ausführen
- Repository-Metadaten erzeugen
- Paketinstallationen testen
- fertige Pakete / Dokumentation hochladen
- Pull Request prüfen und automatisch mergen


bareos
open source data protection
Wie geht das mit Jenkins?
Andreas Rogge - Bareos GmbH & Co. KG - www.bareos.com

Build-Pipeline

Build-Pipeline

- Beschreibung des Build-Ablaufs

Build-Pipeline

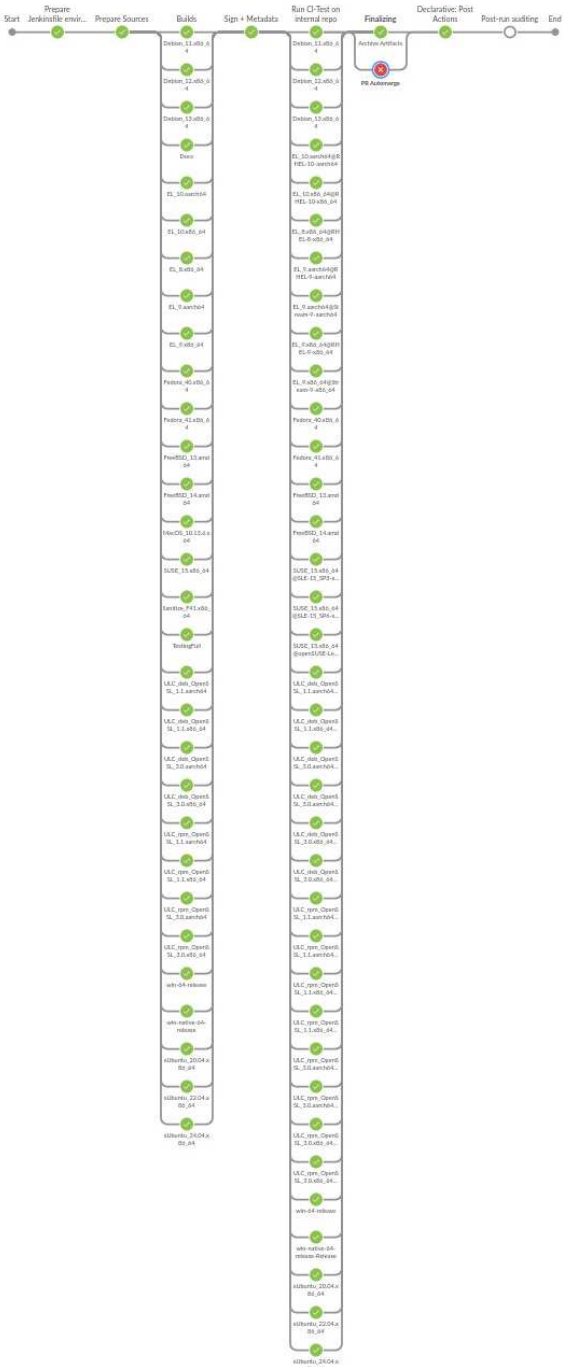
- Beschreibung des Build-Ablaufs
- Kann Anpassung über Parameter erlauben

Build-Pipeline

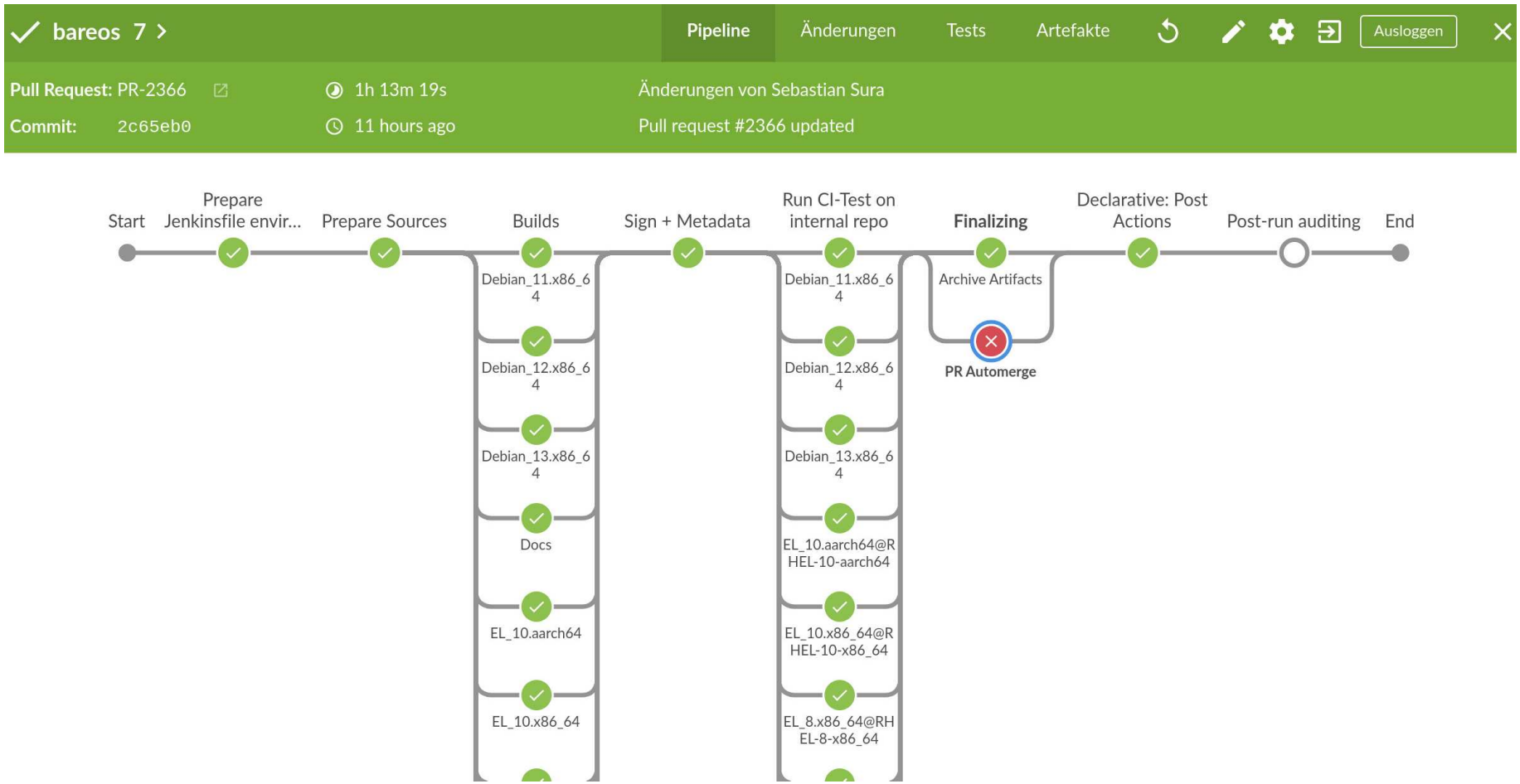
- Beschreibung des Build-Ablaufs
- Kann Anpassung über Parameter erlauben
- Teilt den Build in Stages auf

Build-Pipeline

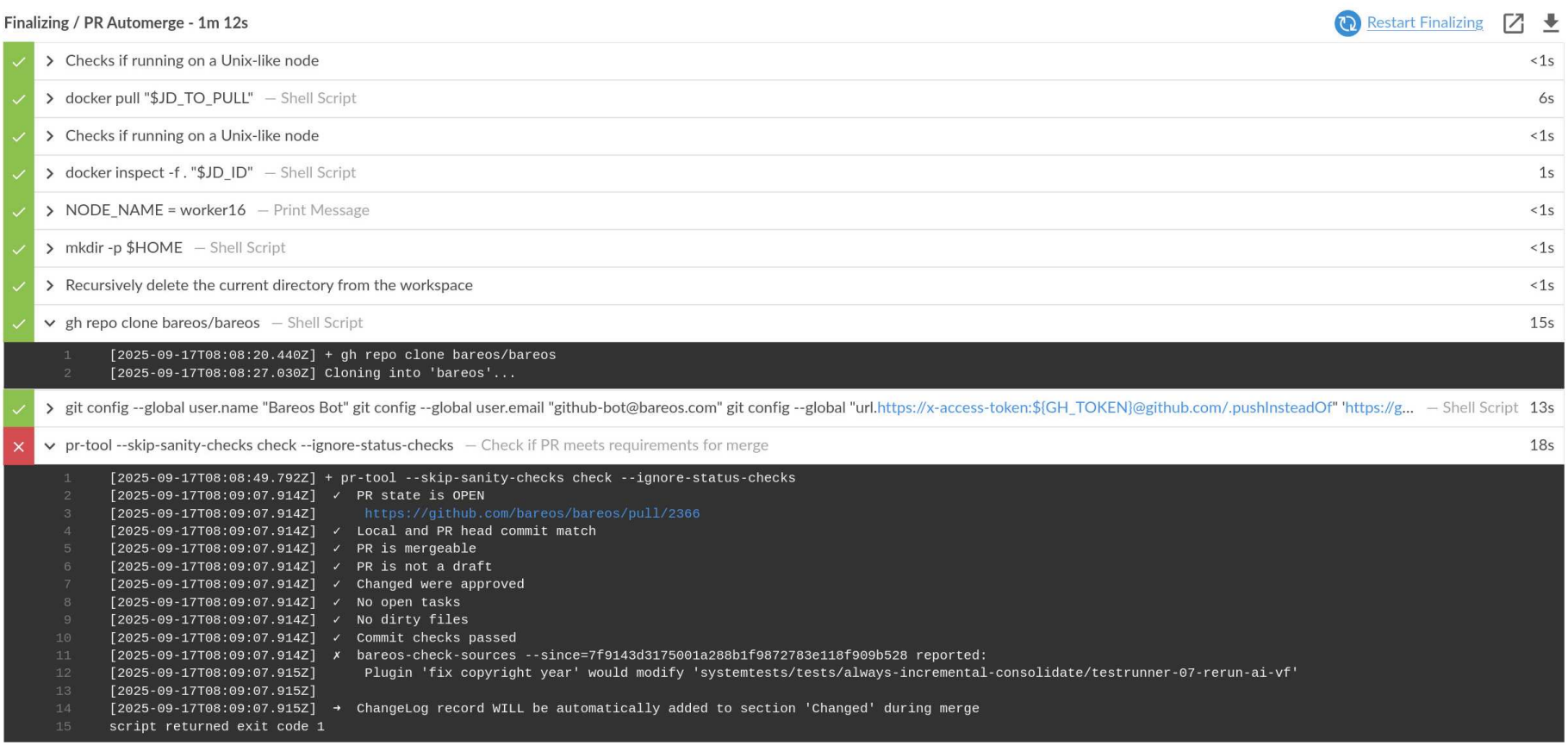
- Beschreibung des Build-Ablaufs
- Kann Anpassung über Parameter erlauben
- Teilt den Build in Stages auf
- Spezifiziert wo welche Stage ausgeführt wird



Bareos Pipeline



Automerge Stage



Jenkinsfile

Jenkinsfile

- liegt im Source-Verzeichnis

Jenkinsfile

- liegt im Source-Verzeichnis
- deklarative DSL

Jenkinsfile

- liegt im Source-Verzeichnis
- deklarative DSL
- gemischt mit Groovy

Jenkinsfile

- liegt im Source-Verzeichnis
- deklarative DSL
- gemischt mit Groovy
- sehr mächtig und flexibel

Jenkinsfile

- liegt im Source-Verzeichnis
- deklarative DSL
- gemischt mit Groovy
- sehr mächtig und flexibel
- kaum Werkzeuge verfügbar

Jenkinsfile

- liegt im Source-Verzeichnis
- deklarative DSL
- gemischt mit Groovy
- sehr mächtig und flexibel
- kaum Werkzeuge verfügbar
- schwierig/nervig zu debuggen

Jenkinsfile

```
pipeline {
    parameters {
      string(name: 'message',
             defaultValue: 'Hello, World')
    }
    agent { label 'Linux' }
    stages {
        stage('Hello') {
            steps {
                sh "echo ${params.message}'
            }
        }
    }
}
```

Parameter

Parameter

- Pipelines unterstützen Parameter

Parameter

- Pipelines unterstützen Parameter
- nur bei manuellen Starts setzbar

Parameter

- Pipelines unterstützen Parameter
- nur bei manuellen Starts setzbar
- brauchen immer sinnvolle Standardwerte

Parameter

- Pipelines unterstützen Parameter
- nur bei manuellen Starts setzbar
- brauchen immer sinnvolle Standardwerte

 das geht immer erst beim zweiten Build

Build-Agenten und Label

Build-Agenten und Label

- Stages brauchen unterschiedliche Umgebungen

Build-Agenten und Label

- Stages brauchen unterschiedliche Umgebungen
- Jenkins kann nahezu beliebige Systeme anbinden

Build-Agenten und Label

- Stages brauchen unterschiedliche Umgebungen
- Jenkins kann nahezu beliebige Systeme anbinden
- Können über Label selektiert werden

Build-Agenten und Label

- Stages brauchen unterschiedliche Umgebungen
- Jenkins kann nahezu beliebige Systeme anbinden
- Können über Label selektiert werden
- Agenten können nach Bedarf gestartet und gestoppt werden

Docker-Integration

Stages in Containern ausführen

Docker-Integration

Stages in Containern ausführen

```
agent {
  docker {
    label 'Linux && Docker'
    image 'fedora/fedora-minimal:41'
    registryUrl 'https://quay.io'
  }
}
```

Datenaustausch

Datenaustausch

- Agents haben getrennte Arbeitsverzeichnisse

Datenaustausch

- Agents haben getrennte Arbeitsverzeichnisse
- nötige Dateien müssen übertragen werden

Datenaustausch

- Agents haben getrennte Arbeitsverzeichnisse
- nötige Dateien müssen übertragen werden
- kein Automatismus

Datenaustausch

- Agents haben getrennte Arbeitsverzeichnisse
- nötige Dateien müssen übertragen werden
- kein Automatismus
- integrierte Lösung: `stash`/`unstash`

Datenaustausch

- Agents haben getrennte Arbeitsverzeichnisse
- nötige Dateien müssen übertragen werden
- kein Automatismus
- integrierte Lösung: `stash`/`unstash`
- wir verwenden einfach `rsync`

Parallele Ausführung

```
pipeline {
  agent none (1)
  stages {
    stage('Builds') {
      parallel {
        stage('Build Linux') {
          agent { label 'Linux' } (2)
          steps { ... }
        }
        stage('Build FreeBSD') {
          agent { label 'FreeBSD' } (2)
          steps { ... }
        }
        ...
```

(1) explizit keinen Agenten anfordern

(2) hier werden Agenten reserviert

Zugangsdaten

Zugangsdaten

- Jenkins kann diverse Zugangsdaten verwalten

Zugangsdaten

- Jenkins kann diverse Zugangsdaten verwalten
- Bereitstellung in der Pipeline

Zugangsdaten

- Jenkins kann diverse Zugangsdaten verwalten
- Bereitstellung in der Pipeline
  - Umgebungsvariablen

Zugangsdaten

- Jenkins kann diverse Zugangsdaten verwalten
- Bereitstellung in der Pipeline
  - Umgebungsvariablen
  - temporäre Dateien

Zugangsdaten

- Jenkins kann diverse Zugangsdaten verwalten
- Bereitstellung in der Pipeline
  - Umgebungsvariablen
  - temporäre Dateien
  - SSH-Keys direkt als SSH-Agent

Zugangsdaten

- Jenkins kann diverse Zugangsdaten verwalten
- Bereitstellung in der Pipeline
  - Umgebungsvariablen
  - temporäre Dateien
  - SSH-Keys direkt als SSH-Agent
  - Zertifikate als Java Keystore

Zugangsdaten

- Jenkins kann diverse Zugangsdaten verwalten
- Bereitstellung in der Pipeline
  - Umgebungsvariablen
  - temporäre Dateien
  - SSH-Keys direkt als SSH-Agent
  - Zertifikate als Java Keystore
- Im Log werden diese bestmöglich maskiert

Zugangsdaten

String-Credential

```
withCredentials([
    string(credentialsId: 'user-pin', variable: 'PIN')
    ]) {
  script {
    sh 'pkcs11-tool --pin $PIN ...'
  }
}
```

SSH Private-Key

```
sshagent(credentials: [ 'key1', 'key2' ]) {
  script {
    sh 'rsync -a $WORKSPACE/results user@host:results'
  }
}
```

Multibranch Pipeline

Multibranch Pipeline

- Pipeline baut nur einen Branch

Multibranch Pipeline

- Pipeline baut nur einen Branch
- Multibranch-Pipeline baut mehrere / alle

Multibranch Pipeline

- Pipeline baut nur einen Branch
- Multibranch-Pipeline baut mehrere / alle
- Generiert eine Pipeline pro Branch / PR

Default Jenkinsfile

Default Jenkinsfile

- Jenkinsfile außerhalb des Code-Repositories

Default Jenkinsfile

- Jenkinsfile außerhalb des Code-Repositories
- muss in Jenkins selbst abgelegt werden

Default Jenkinsfile

- Jenkinsfile außerhalb des Code-Repositories
- muss in Jenkins selbst abgelegt werden
- ein Jenkinsfile kann ein anderes laden

Default Jenkinsfile

- Jenkinsfile außerhalb des Code-Repositories
- muss in Jenkins selbst abgelegt werden
- ein Jenkinsfile kann ein anderes laden
- kleines Jenkinsfile als Bootstrap

Default Jenkinsfile

- Jenkinsfile außerhalb des Code-Repositories
- muss in Jenkins selbst abgelegt werden
- ein Jenkinsfile kann ein anderes laden
- kleines Jenkinsfile als Bootstrap
- "echtes" Jenkinsfile im getrennten Repository

Default Jenkinsfile

```
node {
  label ('master')
  stage('Prepare Jenkinsfile environment') {
    def targets = [ env.BRANCH_NAME, 'master' ]
    dir('CD') {
      checkout resolveScm(
        source: [ ... ],
        targets: targets
      )
    }
  }
  load 'CD/Jenkinsfile'
}
```

Dynamische Stages

Dynamische Stages

- große Pipelines deklarativ nicht möglich

Dynamische Stages

- große Pipelines deklarativ nicht möglich
  - einfach nicht mehr wartbar

Dynamische Stages

- große Pipelines deklarativ nicht möglich
  - einfach nicht mehr wartbar
  - es gibt es Limit wie viel man deklarieren kann

Dynamische Stages

- große Pipelines deklarativ nicht möglich
  - einfach nicht mehr wartbar
  - es gibt es Limit wie viel man deklarieren kann
- Stages erst zur Laufzeit generieren

Dynamische Stages

- große Pipelines deklarativ nicht möglich
  - einfach nicht mehr wartbar
  - es gibt es Limit wie viel man deklarieren kann
- Stages erst zur Laufzeit generieren
  - lohnt bei vielen ähnlichen Stages

Dynamische Stages

- große Pipelines deklarativ nicht möglich
  - einfach nicht mehr wartbar
  - es gibt es Limit wie viel man deklarieren kann
- Stages erst zur Laufzeit generieren
  - lohnt bei vielen ähnlichen Stages
  - kann in Groovy programmiert werden

Dynamische Stages

```
stage('Builds') {
  steps {
    script {
      stages["RHEL 9"] = make_linux_stage("RHEL", "9")
      stages["FreeBSD 14"] = make_freebsd_stage(13)
      stages["Documentation"] = make_docs_stage()
    }
  }
  parallel stages (1)
}
```

Dynamische Stages

```
stage('Builds') {
  steps {
    script {
      stages["RHEL 9"] = make_linux_stage("RHEL", "9")
      stages["FreeBSD 14"] = make_freebsd_stage(13)
      stages["Documentation"] = make_docs_stage()
    }
  }
  parallel stages (1)
}
```

(1) erwartet eine Map [Name → Stage]

Dynamische Stages

```
stage('Builds') {
  steps {
    script {
      stages["RHEL 9"] = make_linux_stage("RHEL", "9")
      stages["FreeBSD 14"] = make_freebsd_stage(13)
      stages["Documentation"] = make_docs_stage()
    }
  }
  parallel stages 1
}
```

1 erwartet eine Map [Name → Stage]

⚠ Scriptcode und kein deklarativer Code

Dynamische Stages

```
def make_docs_stage(String name) {
  return {  (1)
    stage(name) { (2)
      node("Sphinx") {
        checkout scm
        sh 'sphinx-build docs/manuals/source docs-out'
        rsync_stash()
      }
    }
  }
}
```

Dynamische Stages

```
def make_docs_stage(String name) {
  return {  (1)
    stage(name) {  (2)
      node("Sphinx") {
        checkout scm
        sh 'sphinx-build docs/manuals/source docs-out'
        rsync_stash()
      }
    }
  }
}
```

(1) hier wird eine Closure zurückgegeben

(2) eine Scripted Stage, keine deklarative Stage

Build Matrix

YAML-Beschreibung der Stages

Build Matrix

YAML-Beschreibung der Stages

```
ULC_rpm:
  "OpenSSL_1.1":
    type: scripted
    image: rhel8
    build_script: CD/rpm/build-ulc.sh
    arch:
      - x86_64
      - aarch64
FreeBSD:
  "14":
    type: freebsd
    arch:
      - amd64
```

Build Matrix

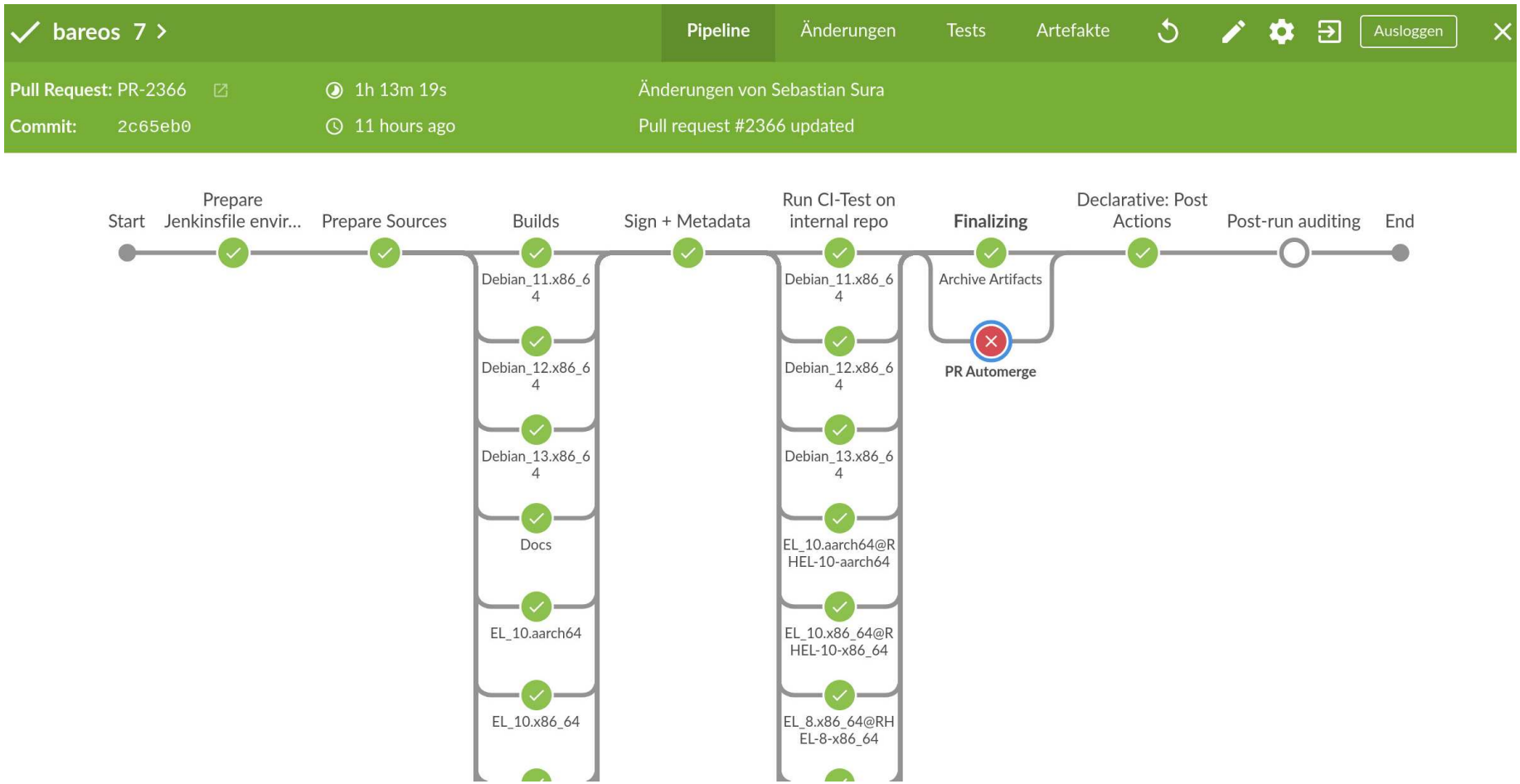
Verarbeitung in Jenkins

Build Matrix

Verarbeitung in Jenkins

```
matrix = readYaml file: "bareos/.matrix.yml"
matrix.each { dist, versions ->
  versions.each { vid, v ->
    v.arch.each { a ->
      def name = "${dist}_${vid}.${arch}"
      if (v.type == "scripted") {
        stages[name] = make_scripted_stage(name, a, v)
      } else if (v.type == "freebsd") {
        stages[name] = make_freebsd_stage(name, a, v)
      }
    }
  }
}
```

Bareos Pipeline



Wieviel CI Code?

Wieviel CI Code?

Sprache	LoC
Jenkinsfile	1.700
Shellskripte	2.600
Python	500
Batch / PowerShell	180
	~5.000

Und die Builder?

Und die Builder?

Sprache	LoC
Python	1.000
Ansible	2.000
Shellskripte	100
	~3.000

Und die Builder?

Sprache	LoC
Python	1.000
Ansible	2.000
Shellskripte	100
	~3.000

und dazu noch 120 Container-Images

Systeme

Systeme

- Jenkins selbst

Systeme

- Jenkins selbst
- Compiler Cache

Systeme

- Jenkins selbst
- Compiler Cache
- RSync Stash

Systeme

- Jenkins selbst
- Compiler Cache
- RSync Stash
- Container Registry

Systeme

- Jenkins selbst
- Compiler Cache
- RSync Stash
- Container Registry
- 8 unterschiedliche Arten von Build-Agenten
  - insgesamt 40 Stück

Systeme

- Jenkins selbst
- Compiler Cache
- RSync Stash
- Container Registry
- 8 unterschiedliche Arten von Build-Agenten
  - insgesamt 40 Stück
- etliche Test-Dependencies

Lessons learned

Was wir in den letzen 6 Jahren zu dem Thema gelernt haben.

über Jenkins

über Jenkins

- Groovy-Kenntnisse sind hilfreich

über Jenkins

- Groovy-Kenntnisse sind hilfreich
- Manchmal braucht man Java-Kenntnisse

über Jenkins

- Groovy-Kenntnisse sind hilfreich
- Manchmal braucht man Java-Kenntnisse
- Dokumentation teilweise lückenhaft

über Jenkins

- Groovy-Kenntnisse sind hilfreich
- Manchmal braucht man Java-Kenntnisse
- Dokumentation teilweise lückenhaft
- Größenbeschränkung Jenkinsfile

über Jenkins

- Groovy-Kenntnisse sind hilfreich
- Manchmal braucht man Java-Kenntnisse
- Dokumentation teilweise lückenhaft
- Größenbeschränkung Jenkinsfile
- Seltsamkeiten im Jenkinsfile

über Jenkins

- Groovy-Kenntnisse sind hilfreich
- Manchmal braucht man Java-Kenntnisse
- Dokumentation teilweise lückenhaft
- Größenbeschränkung Jenkinsfile
- Seltsamkeiten im Jenkinsfile
  - CPS-Transformation

über Jenkins

- Groovy-Kenntnisse sind hilfreich
- Manchmal braucht man Java-Kenntnisse
- Dokumentation teilweise lückenhaft
- Größenbeschränkung Jenkinsfile
- Seltsamkeiten im Jenkinsfile
  - CPS-Transformation
  - `"${var}"` ist kein String

über Jenkins

- Groovy-Kenntnisse sind hilfreich
- Manchmal braucht man Java-Kenntnisse
- Dokumentation teilweise lückenhaft
- Größenbeschränkung Jenkinsfile
- Seltsamkeiten im Jenkinsfile
  - CPS-Transformation
  - `"${var}"` ist kein String
- Plugins kommen und gehen

über CD Allgemein

über CD Allgemein

- Pipelines sind auch Programmcode

über CD Allgemein

- Pipelines sind auch Programmcode
- Lokaler Build sollte analog zum CI gehen

über CD Allgemein

- Pipelines sind auch Programmcode
- Lokaler Build sollte analog zum CI gehen
- Schnittstellen definieren ist wichtig

über CD Allgemein

- Pipelines sind auch Programmcode
- Lokaler Build sollte analog zum CI gehen
- Schnittstellen definieren ist wichtig
- häufig ungeplanter Pflegeaufwand

über CD Allgemein

- Pipelines sind auch Programmcode
- Lokaler Build sollte analog zum CI gehen
- Schnittstellen definieren ist wichtig
- häufig ungeplanter Pflegeaufwand
- insgesamt Pflegeintensiv

Fazit

Fazit

- wirklich viel Aufwand, aber ROI ist großartig

Fazit

- wirklich viel Aufwand, aber ROI ist großartig
- steigert die Produktivität ungemein

Fazit

- wirklich viel Aufwand, aber ROI ist großartig
- steigert die Produktivität ungemein
- aktuelle Velocity wäre ohne CD nicht vorstellbar

Fazit

- wirklich viel Aufwand, aber ROI ist großartig
- steigert die Produktivität ungemein
- aktuelle Velocity wäre ohne CD nicht vorstellbar
- ist trotzdem keine "Silver Bullet"