Debconf, Prizren
07.22/Manuel Traut

DEBCONF KOSOVO 22

# Build Debian based embedded images in the cloud

METTLER TOLEDO

# Your start today

METTLER TOLEDO

"Dieses Foto" von Unbekannter Autor ist lizenziert gemäß CC BY

"Dieses Foto" von Unbekannter Autor ist lizenziert gemäß CC BY-SA

"Dieses Foto" von Unbekannter Autor ist lizenziert gemäß CC BY

# Agenda




1 Motivation

2 Overview

3 APT repository and binary artifact storage

4 Jenkins on Kubernetes

5 Jenkins Shared Library

6 Building Debian Packages

7 Building Debian Images

8 What is good, not so good, missing?

9 How can we contribute to Debian? / References

# Motivation

## Proof of Concept: Using Debian instead of Yocto for building a platform

- Motivation
  - Better Security and LTS support
  - Platform as APT repository
  - Projects can focus on application development, build and integration

- To be shown
  - RFS size including QT6 and application <120MB
  - RAUC update from current solution -> Debian based image
  - Support application development on Windows 10
  - Reusable cloud-based CI for package- and image-build

# Who has used?







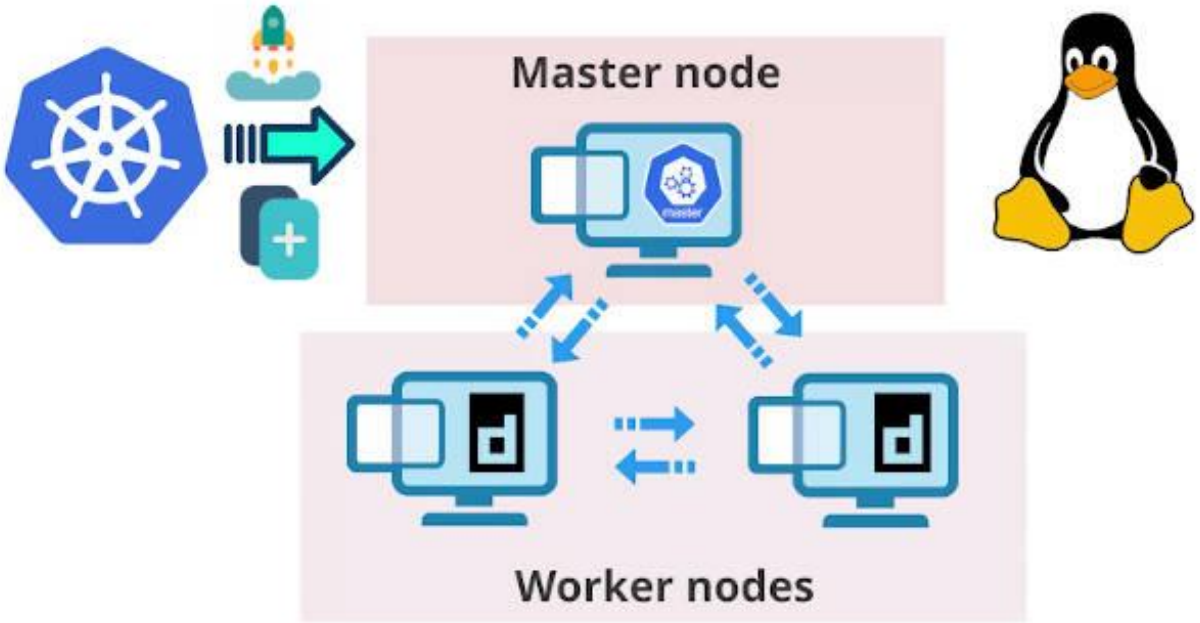
- Build a Debian package
- Debian on non-x86




# Motivation

## Why kubernetes?

- Container Images as runtime environment
- Physical resources/nodes can be added to the cluster as needed
- Infrastructure as code enables projects an easy start



# Agenda




1 Motivation

2 Overview

3 APT repository and binary artifact storage

4 Jenkins on Kubernetes

5 Jenkins Shared Library

6 Building Debian Packages

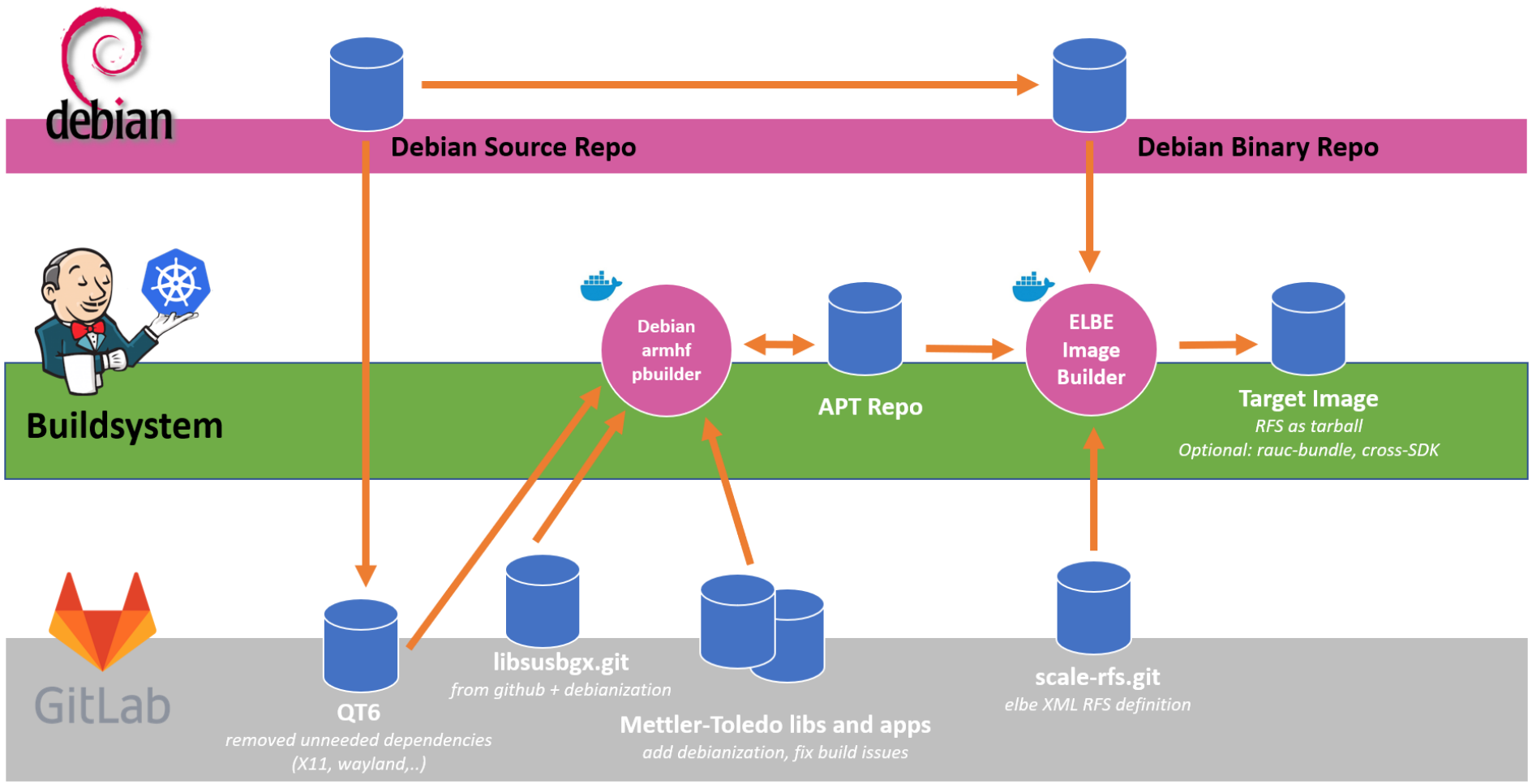
7 Building Debian Images

8 What is good, not so good, missing?

9 How can we contribute to Debian? / References

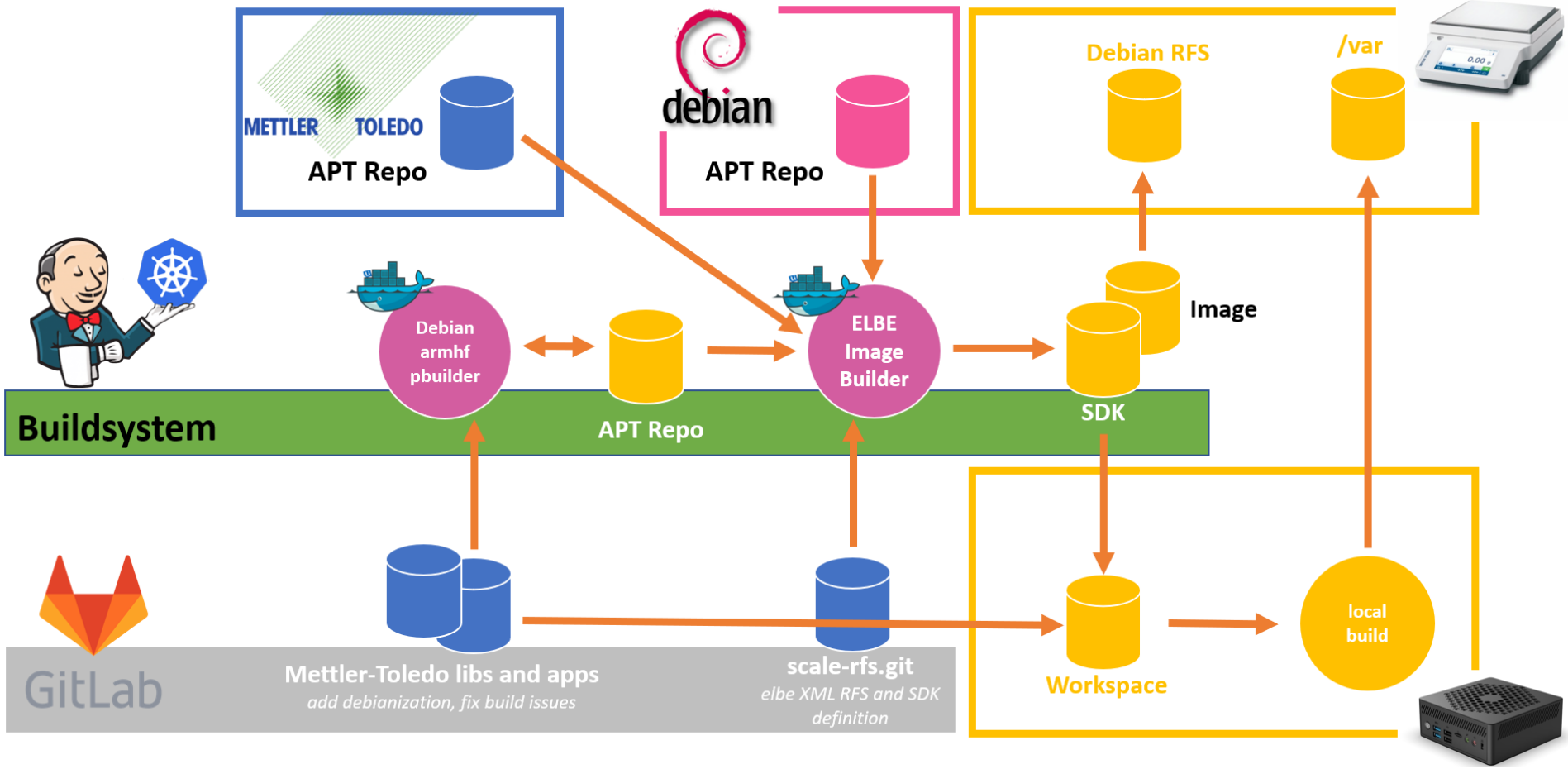
# Overview




debian
Debian Source Repo
Debian Binary Repo
Buildsystem
Debian armhf pbuilder
APT Repo
ELBE Image Builder
Target Image
RFS as tarball
Optional: rauc-bundle, cross-SDK
GitLab
QT6
removed unneeded dependencies
(X11, wayland,..)
libsusbgx.git
from github + debianization
Mettler-Toledo libs and apps
add debianization, fix build issues
scale-rfs.git
elbe XML RFS definition

# Overview




METTLER TOLEDO
APT Repo
debian
APT Repo
Debian RFS
/var
Debian armhf pbuilder
APT Repo
ELBE Image Builder
Image
SDK
Buildsystem
GitLab
Mettler-Toledo libs and apps
add debianization, fix build issues
scale-rfs.git
elbe XML RFS and SDK definition
Workspace
local build

# Agenda




1 Motivation

2 Overview

3 APT repository and binary artifact storage

4 Jenkins on Kubernetes

5 Jenkins Shared Library

6 Building Debian Packages

7 Building Debian Images

8 What is good, not so good, missing?

9 How can we contribute to Debian? / References

# APT Repository

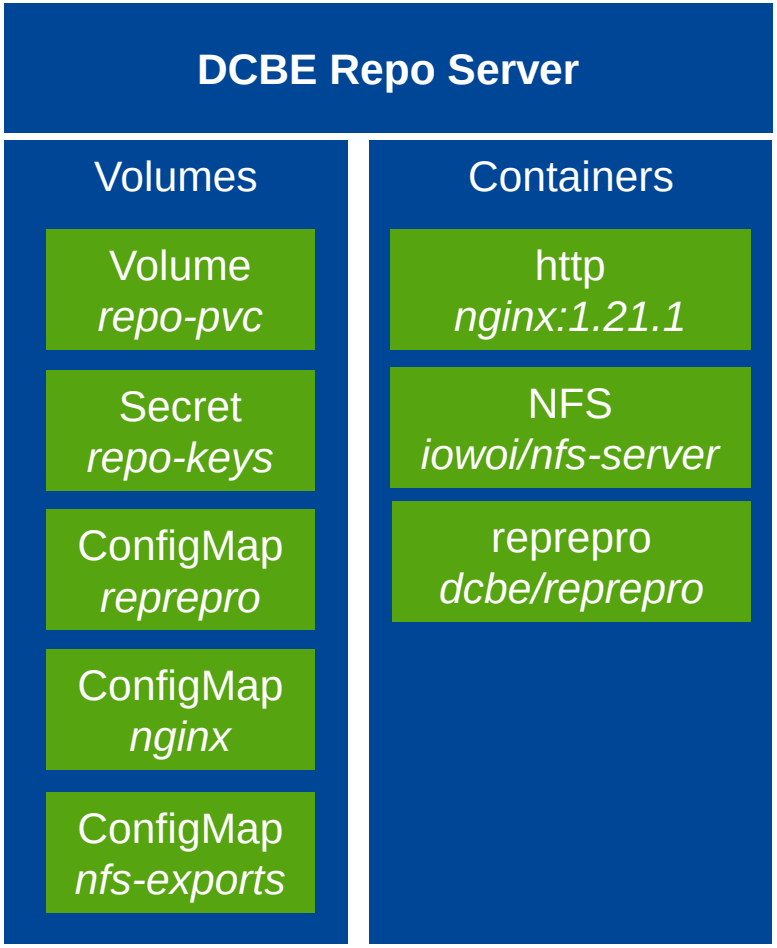





- reprepro
- Sth. else?

# APT repository and binary artifact storage






1. Mount NFS-share in package builder POD
2. Copy *.changes + Co to incoming on NFS
3. Incoming is processed by inoticoming
   → packages are available in the reprepro

- reprepro folder is available via nginx/http
- Keys for signing are stored in a k8s secret
- Publickey is also exported via http

- IP of NFS and HTTP is read from k8s and put into Jenkins Environment variables
- These variables are used in the pbuilder and elbe pipelines to use the proper locations
- Files placed on the NFS share are also accessible via HTTP, this is used for publishing Target Images

# Reposerver on Kubernetes

## Setup

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: dcbe-repo-server
spec:
  selector:
    matchLabels:
      app: dcbe-repo-server
  replicas: 1
  template:
    metadata:
      labels:
        app: dcbe-repo-server
```

```yaml
spec:
     containers:

     - name: dcbe-repo-server-http
       image: nginx:1.21.1
       ports:
       - containerPort: 80
       volumeMounts:
       - mountPath: /usr/share/nginx/html
         name: dcbe-repo-pvc
       - mountPath: /etc/nginx/conf.d
         name: dcbe-repo-nginx-config

     - name: dcbe-repo-server-reprepro
       image: reprepro:latest
       volumeMounts:
       - mountPath: /repo
         name: dcbe-repo-pvc
       - mountPath: /repo/conf
         name: dcbe-repo-distributions-conf
       - mountPath: /keys
         name: dcbe-repo-keys
         readOnly: true
```

# Reposerver on Kubernetes

## Setup

```
- name: dcbe-repo-server-nfs
      image: iowoi/nfs-server:2022-01-11
      ports:
        - name: nfs
          containerPort: 2049
        - name: mountd
          containerPort: 32765
        - name: mountd2
          containerPort: 32766
        - name: rpcbind
          containerPort: 111
        - name: rpcbind2
          containerPort: 32767
      securityContext:
        privileged: true
      volumeMounts:
        - mountPath: /share-rw
          name: dcbe-repo-pvc
        - mountPath: "/lib/modules"
          name: vm-modules
        - mountPath: /etc/exports
          name: dcbe-nfs-exports
          subPath: exports
```

```
volumes:
        - name: dcbe-repo-keys
          secret:
            secretName: dcbe-repo-keys
            optional: false
        - name: dcbe-repo-distributions-conf
          configMap:
            name: dcbe-repo-distributions-conf
        - name: dcbe-repo-nginx-config
          configMap:
            name: dcbe-repo-nginx-conf
        - name: dcbe-nfs-exports
          configMap:
            name: dcbe-repo-nfs-exports-conf
        - name: dcbe-repo-pvc
          persistentVolumeClaim:
            claimName: dcbe-repo-pvc
        - name: vm-modules
          persistentVolumeClaim:
            claimName: vm-modules-pvc
```

## Keys for signing repo

```
echo "DEPLOY APT SIGNING KEYS"

mkdir -p gpg
chmod 700 gpg

gpg --homedir=gpg --generate-key
GPGKEYID=`gpg --homedir=gpg -K | grep '^\ \ ' | sed 's/ //g'`

gpg --homedir=gpg --export-secret-keys --armor > .priv.key
gpg --homedir=gpg --export --armor > .pub.key
kubectl -n $1 create secret generic dcbe-repo-keys \
        --from-file=repo-priv=.priv.key --from-file=repo-pub=.pub.key
```

## reprepro configuration

```
cat > distributions << EOF
Origin: dcbe
Label: DCBE Repository
Description: DCBE debian package repository.
Codename: dcbe
Suite: bookworm
Architectures: i386 amd64 armhf arm64 source
Components: main
SignWith: $GPGKEYID
DebIndices: Packages Release . .gz .xz
UDebIndices: Packages . .gz .xz
DscIndices: Sources Release .gz .xz
Contents: . .gz .xz
EOF

cat > incoming << EOF
Name: dcbe
IncomingDir: /repo/incoming
TempDir: /tmp/reprepro
Default: dcbe
EOF

kubectl -n $1 create configmap dcbe-repo-distributions-conf \
        --from-file distributions --from-file incoming
```

# Reposerver on Kubernetes

## reprepro Container Image

```
FROM debian:bullseye-slim

USER root

ENV DEBIAN_FRONTEND noninteractive

ENV LANG en_US.UTF-8
ENV LANGUAGE en_US:en
ENV LC_ALL en_US.UTF-8

RUN apt-get update -y && \
    apt-get install -y \
        gnupg2 reprepro inoticoming locales && \
    apt-get clean -y && \
    rm -rf /var/lib/apt/lists/*

RUN echo "en_US.UTF-8 UTF-8" > /etc/locale.gen && locale-gen
RUN echo "root:root" | chpasswd

COPY start.sh /start.sh
RUN chmod 755 /start.sh

ENTRYPOINT /start.sh
```

```
#!/bin/bash
set -e

gpg --import /keys/repo-priv
gpg --import /keys/repo-pub
cat /keys/repo-pub | \
    gpg --dearmor > /repo/dcbe.gpg

mkdir -p /repo/incoming; chmod 777
/repo/incoming

inoticoming --foreground \
  /repo/incoming --suffix .changes \
  reprepro -s --basedir /repo \
   processincoming dcbe {} \;
```

# Reposerver on Kubernetes

## Setup NFS

```yaml
kind: ConfigMap
apiVersion: v1
metadata:
  name: dcbe-repo-nfs-exports-conf
data:
  exports: |
    /share-rw *(rw,no_subtree_check,no_root_squash)
```

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: dcbe-repo-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 100Gi
```

```yaml
kind: Service
apiVersion: v1
metadata:
  name: dcbe-repo-nfs-service
spec:
  ports:
    - name: nfs
      port: 2049
    - name: mountd
      port: 32765
    - name: mountd2
      port: 32766
    - name: rpcbind
      port: 111
    - name: rpcbind2
      port: 32767
  selector:
    app: dcbe-repo-server
```

```bash
NFSSERVER=`kubectl -n $1 get service dcbe-repo-nfs-service \
                   --template={{.spec.clusterIP}}`
```

# Agenda




1 Motivation

2 Overview

3 APT repository and binary artifact storage

4 Jenkins on Kubernetes

5 Jenkins Shared Library

6 Building Debian Packages

7 Building Debian Images

8 What is good, not so good, missing?

9 How can we contribute to Debian? / References

# Jenkins on Kubernetes

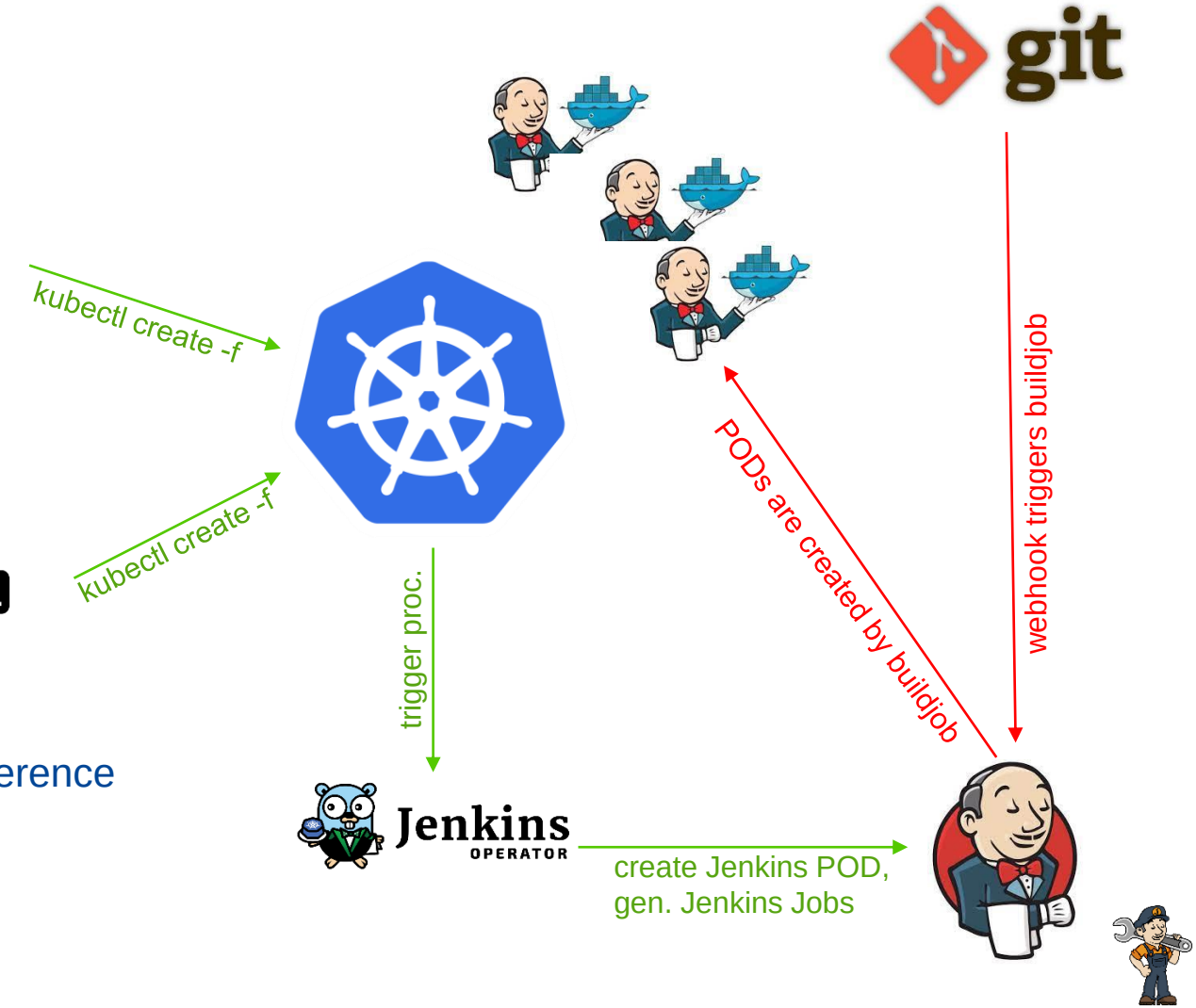


jenkins.yaml

- Container Image
- Job DSL reference

jenkins-conf.yaml

- Environment variables
- Jenkins Shared Lib reference
- Plugin configurations



## Setup

```
KCTL="kubectl -n $1 apply -f"

echo "DEPLOY JENKINS OPERATOR"
JOPURL="https://raw.githubusercontent.com/jenkinsci/kubernetes-operator/master"
$KCTL $JOPURL/config/crd/bases/jenkins.io_jenkins.yaml
$KCTL $JOPURL/deploy/all-in-one-v1alpha2.yaml

echo "DEPLOY JENKINS MASTER"
$KCTL dcbe-jenkins-conf.yaml
$KCTL dcbe-jenkins.yaml
```

## jenkins.yaml

```yaml
apiVersion: jenkins.io/v1alpha2
kind: Jenkins
metadata:
  name: dcbe
  labels:
    app: jenkins-operator
    jenkins-cr: dcbe
    watch: "true"
spec:
  configurationAsCode:
    configurations:
    - name: dcbe-jenkins-conf
  groovyScripts:
    configurations:
    - name: dcbe-jenkins-conf
```

```yaml
master:
   plugins:
    - name: blueocean
      version: "1.25.5"
   containers:
     - name: jenkins-master
       image: jenkins/jenkins:2.319.1-lts
       resources:
         limits:
           cpu: 1500m
           memory: 8Gi
         requests:
           cpu: 1000m
           memory: 8Gi
 seedJobs:
   - id: jenkins-operator
     targets: "*.jenkins"
     description: "Job DSL Reader"
     repositoryBranch: main
     repositoryUrl: INSERT_JOBS_REPO_URL
```

## jenkins-conf.yaml

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  labels:
    app: jenkins-operator
    jenkins-cr: dcbe
    watch: "true"
  name: dcbe-jenkins-conf
data:
  1-system-env.yaml: |
    jenkins:
      globalNodeProperties:
      - envVars:
          env:
          - key: "DOCKER_REGISTRY"
            value: 'INSERT_CONTAINER_REG'
          - key: "APT_REPO_NFS"
            value: 'INSERT_NFS_IP'
          - key: "APT_REPO_HTTP"
            value: 'INSERT_HTTP_IP'
```

```yaml
2-global-jsl.yaml: |
   unclassified:
     globalLibraries:
       libraries:
       - defaultVersion: "main"
         name: "dcbe"
         retriever:
           modernSCM:
             scm:
               git:
                 remote: "INSERT_SHAREDLIB_REPOURL"
                 traits:
                 - "gitBranchDiscovery"
```

# Agenda




1 Motivation

2 Overview

3 APT repository and binary artifact storage

4 Jenkins on Kubernetes

5 Jenkins Shared Library

6 Building Debian Packages

7 Building Debian Images

8 What is good, not so good, missing?

9 How can we contribute to Debian? / References

# Jenkins Shared Library



## Reusable Build Pipelines

- dockerimagebuild.groovy
  - Uses Kaniko for building Container Images
  - Input: Dockerfile
  - Output: Push Image to configured Container Registry
  - Used to build the Container Images that are used by the following pipelines

- pbuilder.groovy
  - Uses gbp and cowbuilder for building Debian Packages
  - Input: debian folder with watchfile oder debianized sources
  - Output: Publishes Packages in a reprepro APT Repository

- elbe.groovy
  - Uses "elbe buildchroot" for building Target Images and SDKs
  - Input: ELBE XML File, Packages from APT Repositories
  - Output: Image and optional SDK published on a http server

# Agenda




1 Motivation

2 Overview

3 APT repository and binary artifact storage

4 Jenkins on Kubernetes

5 Jenkins Shared Library

6 Building Debian Packages

7 Building Debian Images

8 What is good, not so good, missing?

9 How can we contribute to Debian? / References

# Packagebuild



- dpkg-buildpackage
- pbuilder
- cowbuilder
- git-build-package
- ??

# Jenkins Shared Library – Package Build

## Jenkinsfile in eg. qt6-base.git

```
@Library('dcbe') _

Map conf = [ 'armhf' : 'bookworm',
             'arm64' : 'bookworm'  ]

pbuilder(conf)
```

## jenkins-shared-library.git: vars/pbuilder.groovy #1

```
def call(Map conf) {
  node {
    def pbuilds = conf.collectEntries {
      ["${it.getKey()}-${it.getValue()}": pbuild(it.getKey(), it.getValue())]
    }
    parallel pbuilds
  }
}
```

# Jenkins Shared Library – Package Build

## jenkins-shared-library.git: vars/pbuilder.groovy #2

```
def pbuild(String arch, String release) {
  return {
    String podlabel = "pbuilder-${arch}-${release}-${UUID.randomUUID().toString()}"
    podTemplate(label: "${podlabel}", containers  : [
      containerTemplate(name                : 'pbuilder',
                        image               : "${env.REGISTRY}/pbuild-${arch}:${release}",
                        resourceRequestCpu: '8000m', resourceRequestMemory: '16Gi',
                        resourceLimitCpu  : '16000m', resourceLimitMemory : '16Gi',
                        privileged          : true,
                        command             : 'cat'), ],
                                      volumes     : [
      persistentVolumeClaim(claimName     : 'vm-modules-pvc',
                            mountPath     : '/lib/modules'),
      nfsVolume            (serverAddress : "${env.APT_REPO_NFS}",
                            serverPath    : '/share-rw',
                            mountPath     : '/repo'),
      configMapVolume      (configMapName : 'dcbe-pbuilder-hooks',
                            mountPath     : '/usr/lib/pbuilder/hooks'),
                                                  ], )
    {
      do_pbuild(String arch, String release, String podlabel)
    }
  }
}
```

## pbuilder hooks

```
cat > H10-update.sh << EOF
#!/bin/sh

echo "Package: *
Pin: origin "$HTTPSERVER"
Pin-Priority: 910" > /etc/apt/preferences

apt update
apt install -y curl
mkdir -p /usr/share/keyrings
curl -o /usr/share/keyrings/dcbe.gpg http://$HTTPSERVER/dcbe.gpg

echo "deb [signed-by=/usr/share/keyrings/dcbe.gpg] http://$HTTPSERVER dcbe main" \
     > /etc/apt/sources.list.d/dcbe.list
apt update
apt dist-upgrade -y
EOF

kubectl -n $1 create configmap dcbe-pbuilder-hooks \
  --from-file=H10-update.sh=H10-update.sh
```

# Jenkins Shared Library – Package Build

**jenkins-shared-library.git: vars/pbuilder.groovy #3**

```
def do_pbuild(String arch, String release, String podlabel)
{
  node ("${podlabel}")
  {
    container('pbuilder')
    {
      stage('prepare buildenv')
      {
        sh "modprobe binfmt_misc"
        sh "update-binfmts --enable qemu-arm --enable qemu-aarch64"
      }
      stage('build package')
      {
        sh "pbuild.sh"
      }
    }
  }
}
```

# Jenkins Shared Library – Package Build

## jenkins-shared-library.git: vars/pbuilder.groovy #4 (pbuild.sh)

```
# hooks are mounted as configmap, so they are not executable
mkdir -p /usr/lib/pbuilder/hooks-exec
cp -H /usr/lib/pbuilder/hooks/* /usr/lib/pbuilder/hooks-exec
chmod 755 /usr/lib/pbuilder/hooks-exec/*

git checkout ${env.GIT_BRANCH} -B build-${arch}-${release}

# if the project has a watch file, use it to retrieve the original sources
if [ -f debian/watch ]; then
  git branch upstream
  uscan --download-current-version
  gbp import-orig --no-interactive \
                  --debian-branch=build-${arch}-${release} \
                  --upstream-branch=upstream ../*orig.tar.*
fi
gbp buildpackage --git-pbuilder \
                 --git-dist=${release} \
                 --git-arch=${arch} \
                 --git-debian-branch=build-${arch}-${release} \
                 --git-upstream-tree=BRANCH \
                 --git-pbuilder-options="--hookdir /usr/lib/pbuilder/hooks-exec \
                                         --debbuildopts '-b'"
cp ../*.buildinfo ../*.deb ../*.dsc ../*.tar.* ../*.changes /repo/incoming/
```

# Agenda




1 Motivation

2 Overview

3 APT repository and binary artifact storage

4 Jenkins on Kubernetes

5 Jenkins Shared Library

6 Building Debian Packages

7 Building Debian Images

8 What is good, not so good, missing?

9 How can we contribute to Debian? / References

# Building Debian Images

## Why ELBE?

- The ELBE initvm concept does not fit for using it in kubernetes
- However several ELBE subcommands for building Images and SDKs can be used in a POD
- Debos might be also a good choice (No SDK generation?)
- Imagebuilder can be replaced in the future

- elbe-buildchroot
  - debootstrap
  - install additional packages
  - finetuning
  - image generation
- elbe-buildsdk
  - get source package for each package that is on the target
  - check all source packages for binary packages ending with –dev
  - extract –dev packages to target sysroot
  - install proper Debian cross-toolchain into host sysroot
  - generate a shell script to setup the SDK (yocto-like)

# Jenkins Shared Library – ELBE Image Build

## Jenkinsfile in eg. dcbe-qemu-img.git

```
@Library('dcbe') _

elbe("dcbe-qemu-arm64.xml", false)
```

## Modifications to a default elbe file (see github.com/linutronix/elbe/examples)

```
<url-list>
  <url>
    <binary>http://DCBE_REPO_IP dcbe main</binary>
    <raw-key>
DCBE_REPO_RAW_KEY
    </raw-key>
  </url>
</url-list>
```

```
<raw-preference>
Package: *
Pin: origin "DCBE_REPO_IP"
Pin-Priority: 910
</raw-preference>
```

# Jenkins Shared Library – ELBE Image Build

**jenkins-shared-library.git: vars/elbe.groovy #1**

```
def call(String elbexml, Boolean buildsdk) {
  String podlabel = "elbe-${elbexml}-${UUID.randomUUID().toString()}"

  podTemplate(label: "${podlabel}", containers: [
    containerTemplate(name                 : 'elbe',
                      image                : "${env.REGISTRY}/elbe:latest",
                      resourceRequestCpu   : '2000m',
                      resourceRequestMemory: '16Gi',
                      resourceLimitCpu     : '2000m',
                      resourceLimitMemory  : '16Gi',
                      privileged           : true,
                      command              : 'cat'),
                                               ],
                                       volumes: [
    persistentVolumeClaim(claimName        : 'vm-modules-pvc',
                          mountPath        : '/lib/modules'),
    nfsVolume(serverAddress                : "${env.APT_REPO_NFS}",
              serverPath                   : '/share-rw',
              mountPath                    : '/repo'),
                                               ],)
```

# Jenkins Shared Library – ELBE Image Build

## jenkins-shared-library.git: vars/elbe.groovy #2

```
node ("${podlabel}") {
  container('elbe') {
    stage('build image') {
      sh """#!/bin/bash
modprobe binfmt_misc
update-binfmts --enable qemu-arm --enable qemu-aarch64
sed -i \"s/DCBE_REPO_IP/${env.APT_REPO_HTTP}/g\" ${elbexml}
mv ${elbexml} ${elbexml}.in
wget http://${env.APT_REPO_HTTP}/dcbe.gpg
cat dcbe.gpg | gpg --enarmour | grep -v Comment > dcbe.key
sed -i \"s/PGP ARMORED FILE/PGP PUBLIC KEY BLOCK/g\" dcbe.key
awk '/DCBE_REPO_RAW_KEY/ { system ( \"cat dcbe.key\" ) } \
    !/DCBE_REPO_RAW_KEY/ { print; }' ${elbexml}.in > ${elbexml}
elbe buildchroot -t image ${elbexml}
         """
    }
    if (buildsdk) { stage('build image') { sh "elbe buildsdk image" } }
    stage('upload images') {
      sh "rm -rf image/chroot image/target image/repo image/sysroot"
      String tf = "/repo/images/${elbexml}/${currentBuild.startTimeInMillis}"
      sh "mkdir -p ${tf}; cp -a image/* ${tf}/"
    }
  }
}
```

# Building Debian Packages



## Customized QT6, Mettler-Toledo Software and OSS that is not in Debian

- QT as packaged currently in Debian comes with a lot of runtime-dependencies (X11, DRM, GL, Mesa, cups, mysql, wayland..)
- We have a product requirement: RFS < 120 MB
- We decided to modify the debianization of Debians QT package
  → Reduced build- and therefore also less runtime-dependencies

# Agenda




1 Motivation

2 Overview

3 APT repository and binary artifact storage

4 Jenkins on Kubernetes

5 Jenkins Shared Library

6 Building Debian Packages

7 Building Debian Images

8 What is good, not so good, missing?

9 How can we contribute to Debian? / References

# What is good?



- **setup.sh** – For setting up the environment from scratch on any Kubernetes namespace
- Parallel build of Debian packages for multiple architectures
- Works on any kubernetes cloud
- Working with self-signed APT
- SDK, rauc-bundles and images can be build

# What is missing or not so nice



- Concept for managing different debianizations of a single package (e.g. QT6)
  - Maybe as seen for the linux kernel in Debian
- Automatic rebase of customized packaging
  and rebuild of the package if there is a new version in Debian
- Detect what needs to be rebuilt if a certain package changes
  - Check if integration of meta-debian or isar could solve this
- Reproduce an old imagebuild if there are newer packages on the mirror
  → E.g. howto build snapshot.debian.org for own repos?
- debian/changelog generation (for own applications)
- Debian uses gitlab-ci – Shall we also use it?
- I see a lot of similarities with Mobian
  - Check if there are areas for collaboration

# Agenda




1 Motivation

2 Overview

3 APT repository and binary artifact storage

4 Jenkins on Kubernetes

5 Jenkins Shared Library

6 Building Debian Packages

7 Building Debian Images

8 What is good, not so good, missing?

9 How can we contribute to Debian? / References

# How can we contribute to Debian?



## First challenge

- Improve the open-source culture at Mettler-Toledo

## Ideas

- Switch to gitlab-ci and contribute
- Contribute to packaging
  - Allow installing QT6 with reduced runtime-dependencies
  - Add new packages that are relevant for embedded, eg. Libusbgx
- Help with LTS / eLTS for certain packages