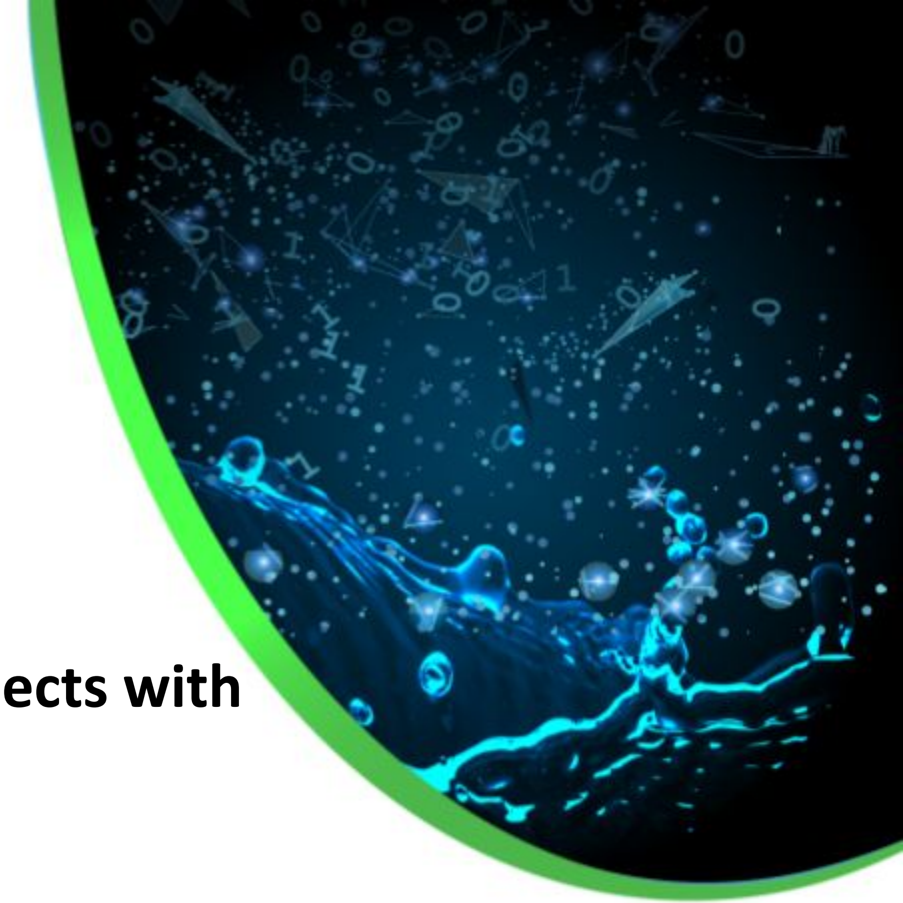
CONAN
C/C++ Package manager

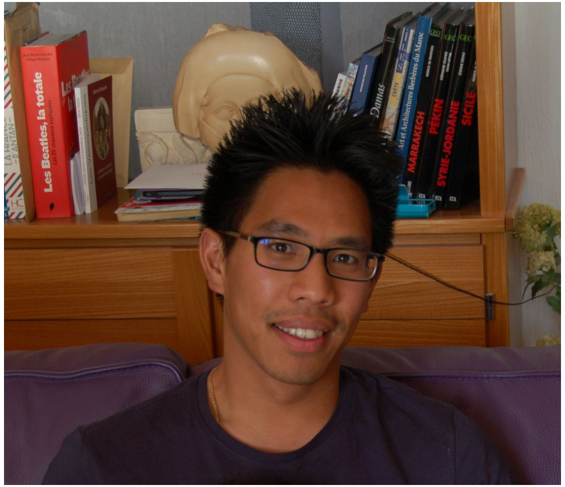


CI/CD Best Practices for C/C++ Projects with Conan and Artifactory

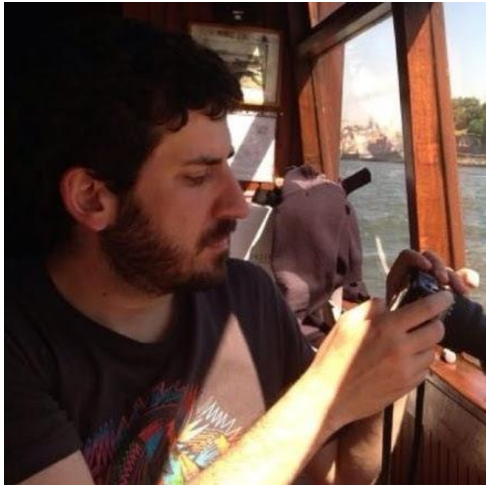
Yann Chaysinsh, Solution Engineer @ JFrog
Carlos Zoido, Conan developer @ JFrog

Coaches

Yann Chaysinh, Solution Engineer



Carlos Zoido, Conan SW engineer



JFrog

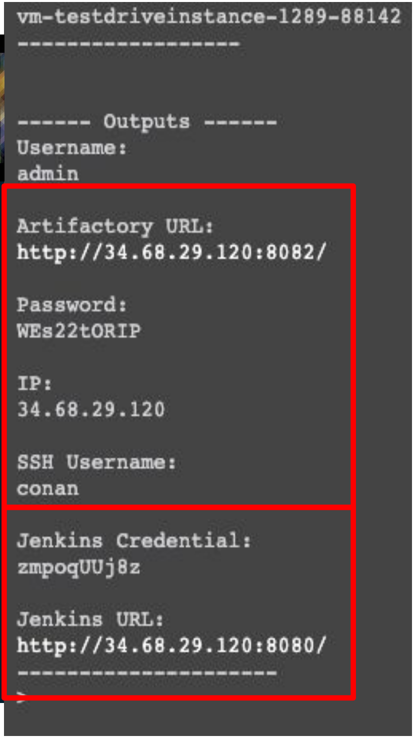
JFrog

Lab 1: Jenkins and environment bootstrapping

- Artifactory
    - Create CI user → **conan/conan2020**
    - Create repositories: conan-tmp, conan-develop, conan-metadata
    - Create permissions
- Conan
    - Download configuration from a git repo
    - Add conan remote and assign user conan
    - Build App, App2 and dependencies → upload them to Artifactory to populate the repos
- Jenkins
    - Create pipelines for all libraries in Jenkins

Lab 1 - Setup

```
ssh conan@<orbitera-IP>
# Use password from orbitera

git clone
https://github.com/conan-ci-cd-training/conan_ci_cd.git

cd conan_ci_cd/setup_jenkins

./bootstrap.sh <artifactory_password>
<jenkins_credential>
```



```
vm-testdriveinstance-1289-88142
------------------


------ Outputs ------
Username:
admin

Artifactory URL:
http://34.68.29.120:8082/

Password:
WEs22tORIP

IP:
34.68.29.120

SSH Username:
conan

Jenkins Credential:
zmpoqUUj8z

Jenkins URL:
http://34.68.29.120:8080/
---------------------
>
```

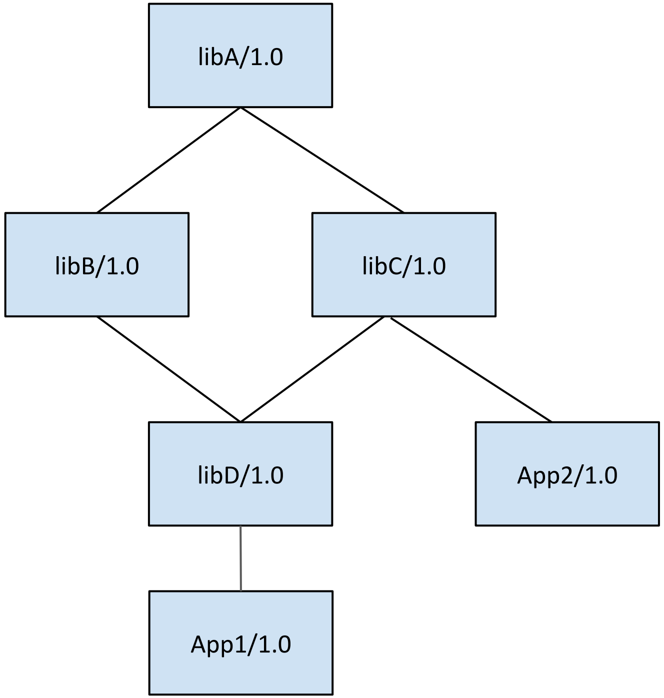
Outline

- **Introduction**
- Conan reminder
- CI
- Build info in Artifactory
- Promotion in Artifactory
- Appendix

The Story: Mycompany components

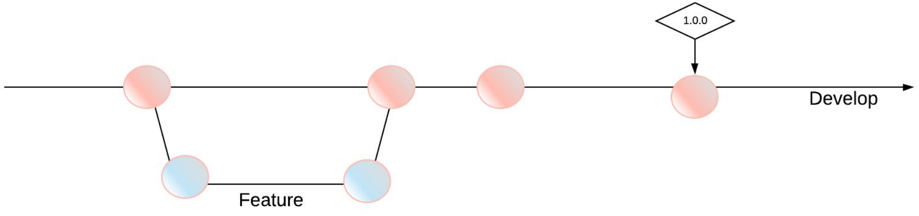
- 1 project providing 2 Apps which consume modules/libs
- All modules/libs are internal to the project and some of them are shared by the Apps

⚠



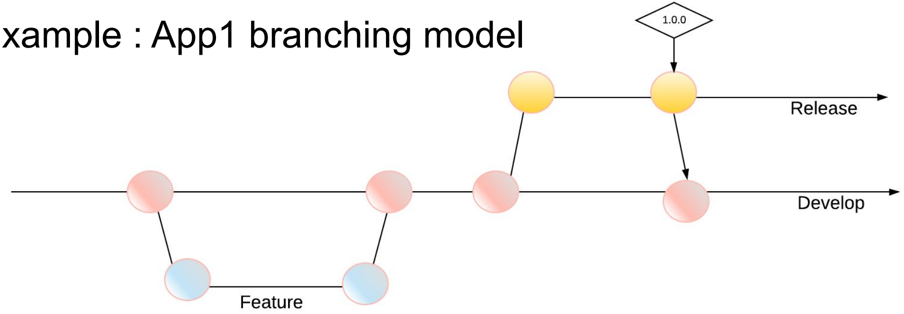
The Story: Code workflow

Example : libA branching model

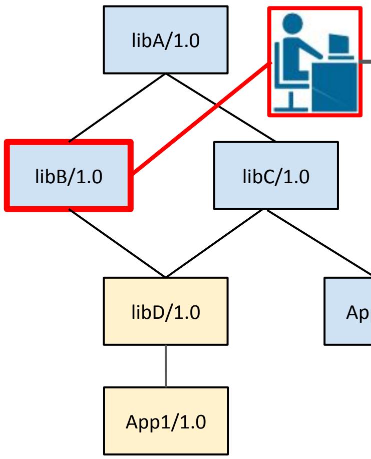


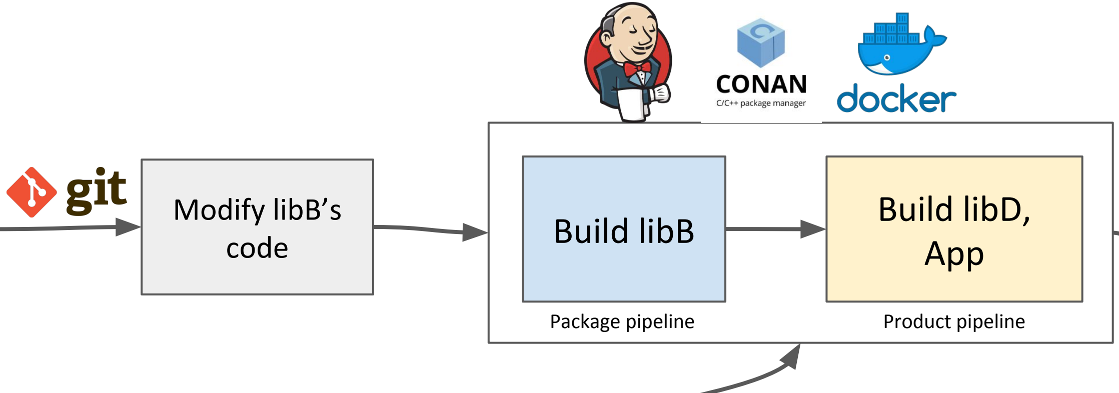
* libB, libC and libD follow the same flow and have their own code repository

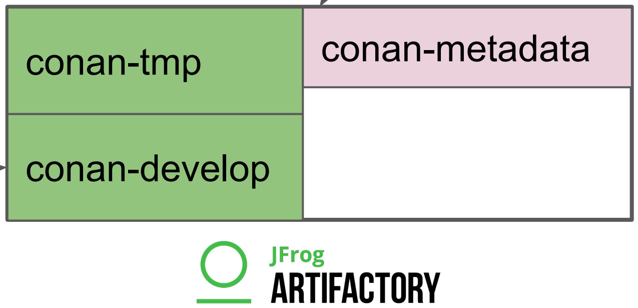
Example : App1 branching model

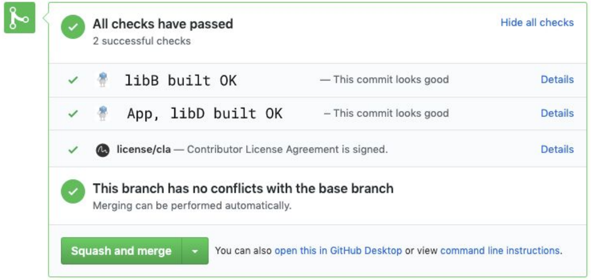


* App2 follows the same flow and has its own code repository








All checks have passed

2 successful checks

Hide all checks

✓ libB built OK — This commit looks good Details

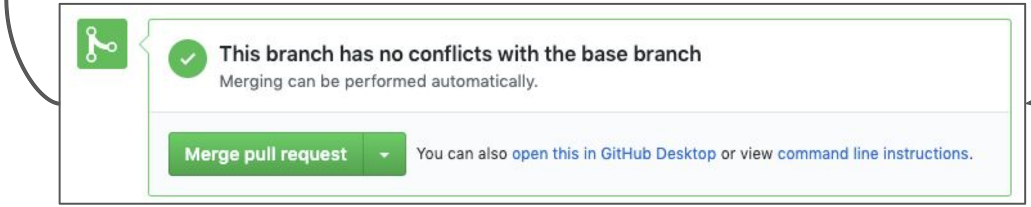
✓ App, libD built OK — This commit looks good Details

✓ license/cla — Contributor License Agreement is signed. Details

This branch has no conflicts with the base branch

Merging can be performed automatically.

Squash and merge ▾ You can also open this in GitHub Desktop or view command line instructions.


This branch has no conflicts with the base branch

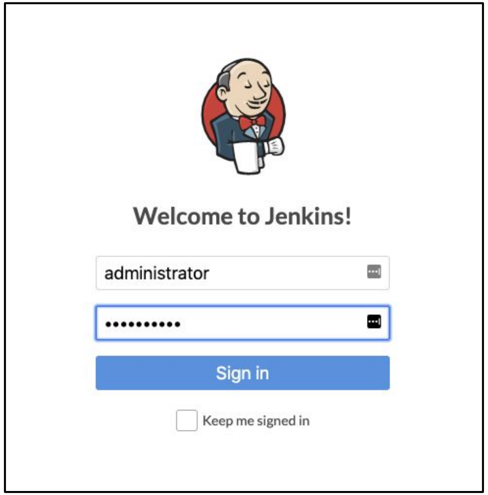
Merging can be performed automatically.

Merge pull request ▾ You can also open this in GitHub Desktop or view command line instructions.

The Story: Goals

- Speed up build time by always having binaries available
- Consuming the latest changes
- Know in advance that changes in libraries do not break the products
- Managing and monitoring the delivery process

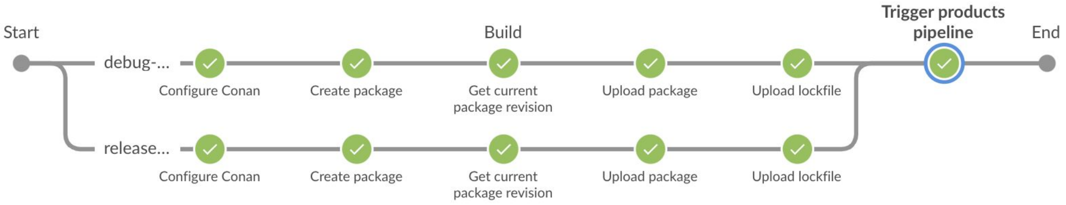
Check Jenkins

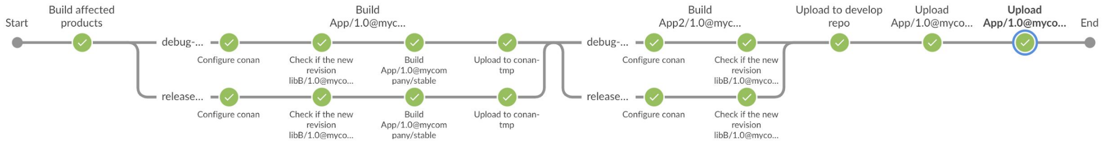


Welcome to Jenkins!

administrator

••••••••••



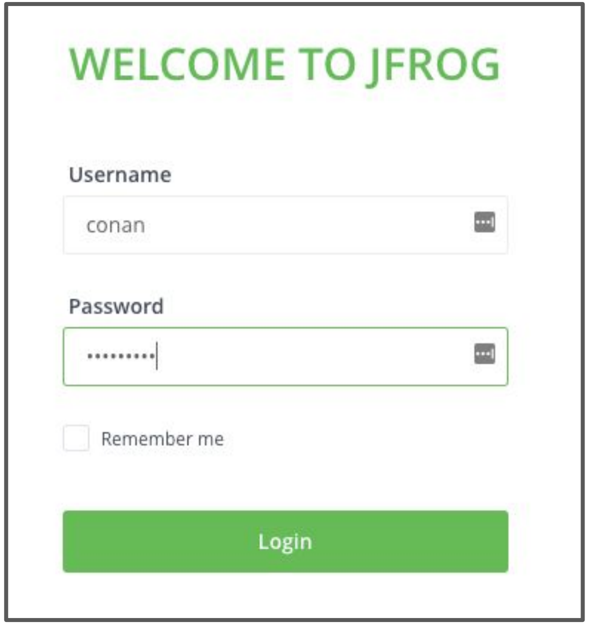


Username: **administrator**
Password: **<Jenkins Credential>**
in orbitera e-mail with JFrog Test Drive Details

Artifactory

- Universal Binary repository manager
- Checksum based storage
- Build Info : Binary dependency tracker
- Properties : metadata applied to any artifacts. Could be used for automation (search, download, move, delete)

Check Artifactory

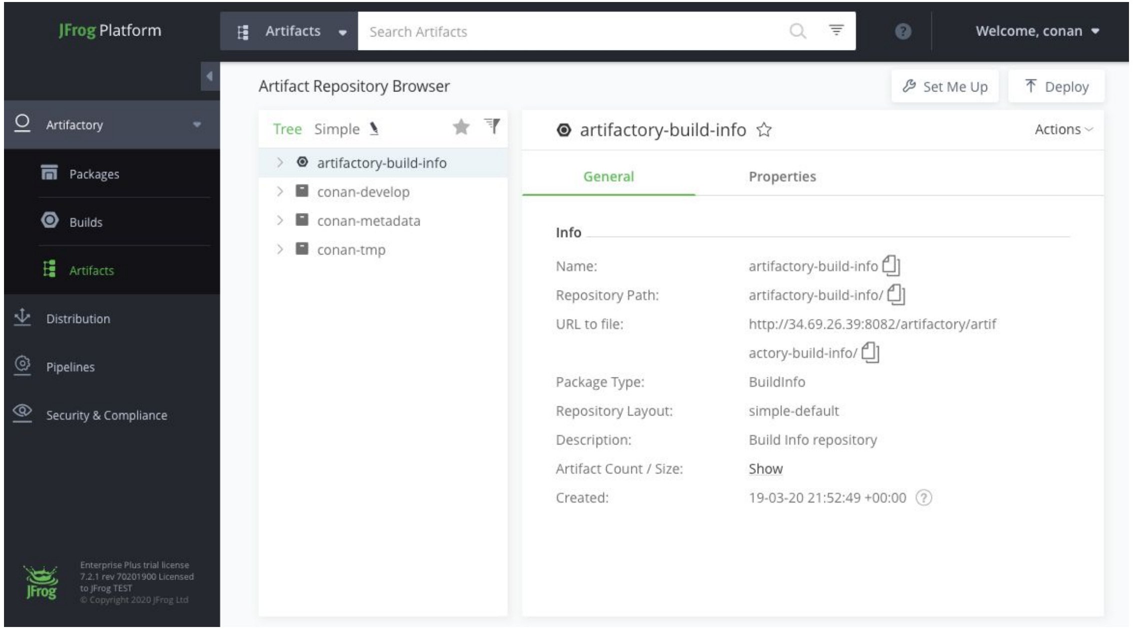


WELCOME TO JFROG

Username: conan

Password: •••••••••

□ Remember me



Info

Name: artifactory-build-info

Repository Path: artifactory-build-info/

URL to file: http://34.69.26.39:8082/artifactory/artifactory-build-info/

Package Type: BuildInfo

Repository Layout: simple-default

Description: Build Info repository

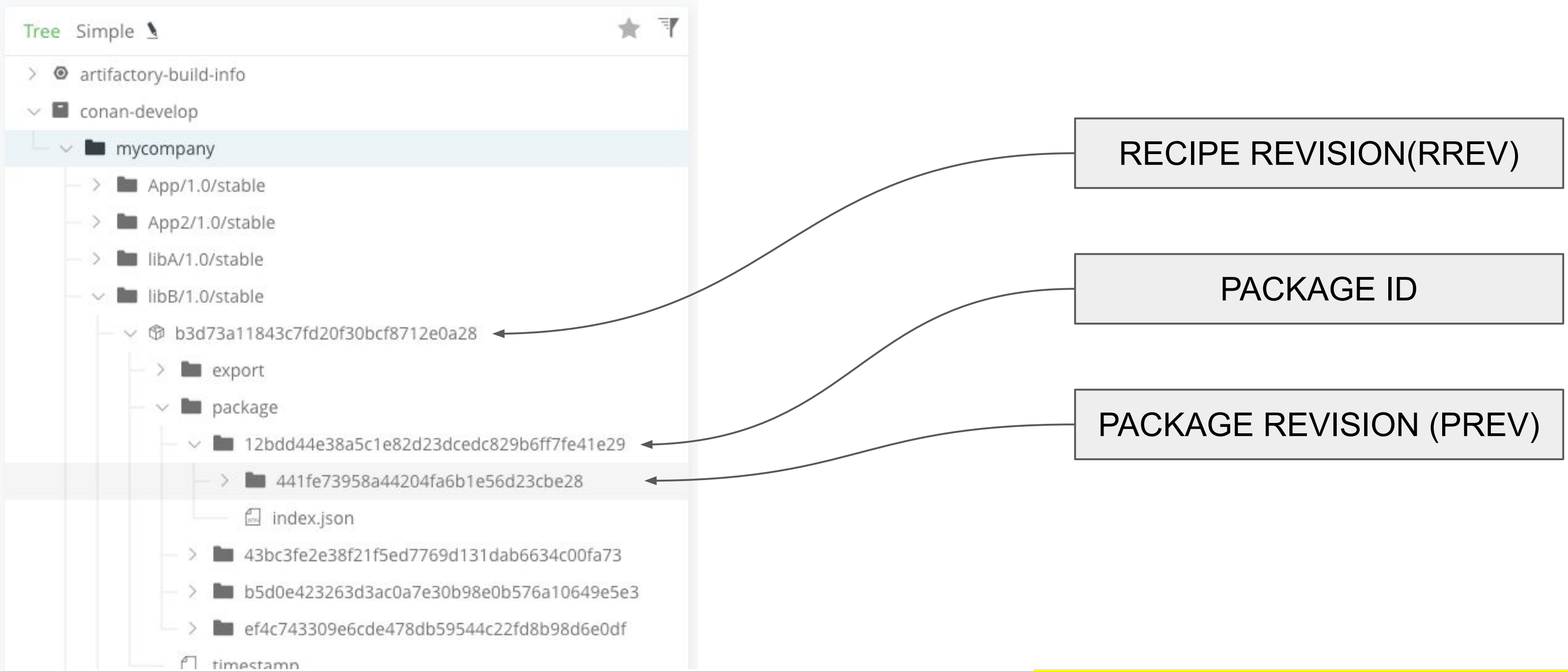
Artifact Count / Size: Show

Created: 19-03-20 21:52:49 +00:00

Enterprise Plus trial license 7.2.1 rev 70201900 Licensed to JFrog TEST

© Copyright 2020 JFrog Ltd

Check Artifactory



Tree Simple

- artifactory-build-info
- conan-develop
  - mycompany
    - App/1.0/stable
    - App2/1.0/stable
    - libA/1.0/stable
    - libB/1.0/stable
      - b3d73a11843c7fd20f30bcf8712e0a28
        - export
        - package
          - 12bdd44e38a5c1e82d23dcedc829b6ff7fe41e29
            - 441fe73958a44204fa6b1e56d23cbe28
            - index.json
          - 43bc3fe2e38f21f5ed7769d131dab6634c00fa73
          - b5d0e423263d3ac0a7e30b98e0b576a10649e5e3
          - ef4c743309e6cde478db59544c22fd8b98d6e0df
        - timestamp

RECIPE REVISION(RREV)

PACKAGE ID

PACKAGE REVISION (PREV)

<name>/<version>@<user>/<channel>#<rrev>:<pkg_id>#<prev>

Outline

- Introduction
- **Conan reminder:** revisions, package id mode, lockfiles
- CI
- Build info in Artifactory
- Promotion in Artifactory
- Appendix

Conan reminder: Revisions

- 2 types :
  - **Recipe :**
    - Id for tracking down any changes at the recipe level.
    - **RREV** = hash(sources, recipe, ...)
  - **Package:**
    - Id for tracking down any changes at the binary package level
    - **PREV** = hash(all the packaged files)
- Goal : **Update packages with changes without bumping the conan package/library version**

Conan reminder: Package ID modes

package_id = f(**settings**, **options**, **requirements**)

- **Settings**: operating systems, compilers, build types,…
- **Options**: shared, fPIC…
- **Requirements**: depending the package_id mode

Package ID modes for binary compatibility

- Can be more strict or more relaxed
- Choosing the right one is important, we will use **recipe_revision_mode** for our CI (quite strict)**, new revisions will affect package id's of dependents**

Conan reminder: Lockfiles

- A snapshot of a dependency graph at a given time.

```
{
 "version": "0.3",
 "profile_host":
"[settings]\narch=x86_64\narch_build=x86_64\nbuild_type=Release\ncompiler=gcc\ncompiler.libcxx=libstdc++11\ncompiler.version=6\nos=Linux\nos_build=Linux\n[options]\n[build_requ
ires]\n[env]\n",
 "graph_lock": {
   "nodes": {
     "0": {
       "options": "shared=False\nlibA:shared=False",
       "pref": "libB/1.0:ef4c743309e6cde478db59544c22fd8b98d6e0df",
       "path": "/var/lib/jenkins/libB/conanfile.py",
       "requires": [
         "1"
       ]
     },
     "1": {
       "options": "shared=False",
       "pref": "libA/1.0@mycompany/stable#d84a023833ae8b56bd8573d05962c937:57547fe65fffc300f05aa42ee64b3b02eeabb6d7#5bafcbf5f3eb1682dcac8e6810bf6e35"
     }
   }
 }
}
```

Conan reminder: Lockfiles use in CI

- Build with the **exact graph** of dependencies
- Use the lockfile to calculate the **build order** of a graph
- If **different nodes** in CI are building the same project, they can **update the lockfile** for the whole graph as they go building libraries
- Generate **Build Info** with the lockfiles (create and install commands will update and mark built libraries as built in the graphlock file)
- Also, lockfiles can be also stored in Artifactory, using a generic repo (conan-metadata repo)

Lockfiles cheatsheet

command	**Input lockfile**	**Output**
create / install / export / export-pkg	Yes (optional)	Update **lockfile**
graph lock	No	**lockfile** with the graph
graph build-order	Yes	**JSON** with build order
graph update-lock	Yes (requires 2 lockfiles)	Update oldest **lockfile**

Conan reminder: two more things

- We will use SCM mode for our examples: commits of source code will generate new RREV

```
class LibB(ConanFile):

    scm = {"type": "git",
           "url": "https://github.com/conan-ci-cd-training/libB.git",
           "revision": "auto"}
```

- Will share the Conan configuration among developers with a git repo

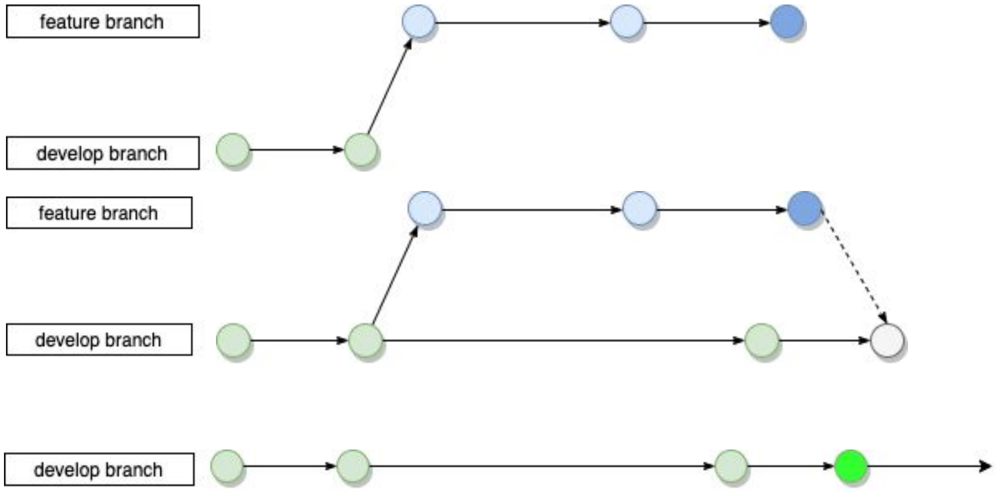
```
conan config install https://github.com/conan-ci-cd-training/settings.git
```

Outline

- Introduction
- Conan reminder: revisions, package id mode, lockfiles
- **CI: workflow**
- Build info in Artifactory
- Promotion in Artifactory
- Appendix

Phases in the workflow

Developers will make changes in the libraries and we want those changes to be seamlessly integrated in our products. Different phases:



Phase 1: Create the feature branch, start developing the feature.

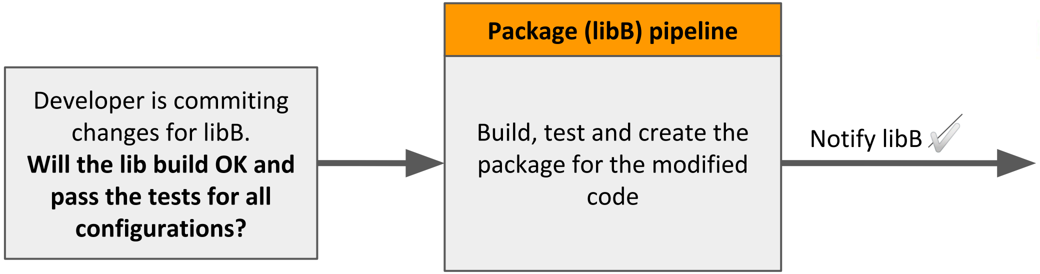
Phase 2: Make PR. Test over temptative commit

Phase 3: Merge the PR

Outline

- Introduction
- Conan reminder: revisions, package id mode, lockfiles
- **CI: workflow phase 1**
- Build info in Artifactory
- Promotion in Artifactory
- Appendix

Phase 1: Developer works on a feature branch of libB



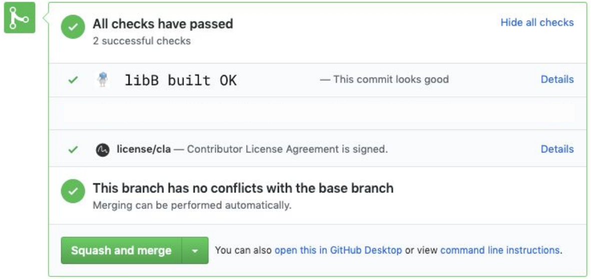
Developer is commiting changes for libB. **Will the lib build OK and pass the tests for all configurations?**

Package (libB) pipeline

Build, test and create the package for the modified code

Notify libB ✓

https://github.com/conan-ci-cd-training/libB/blob/develop/Jenkinsfile



All checks have passed
2 successful checks

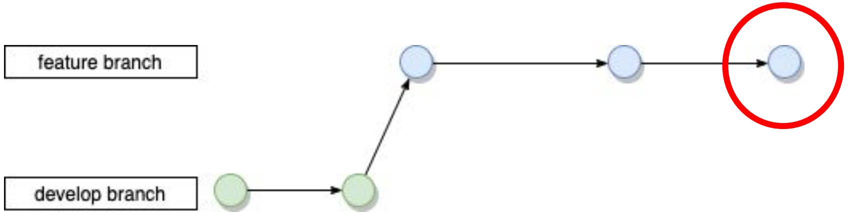
Hide all checks

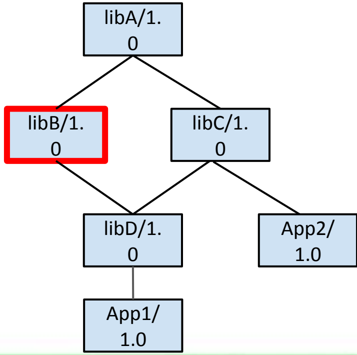
✓ libB built OK — This commit looks good Details

✓ license/cla — Contributor License Agreement is signed. Details

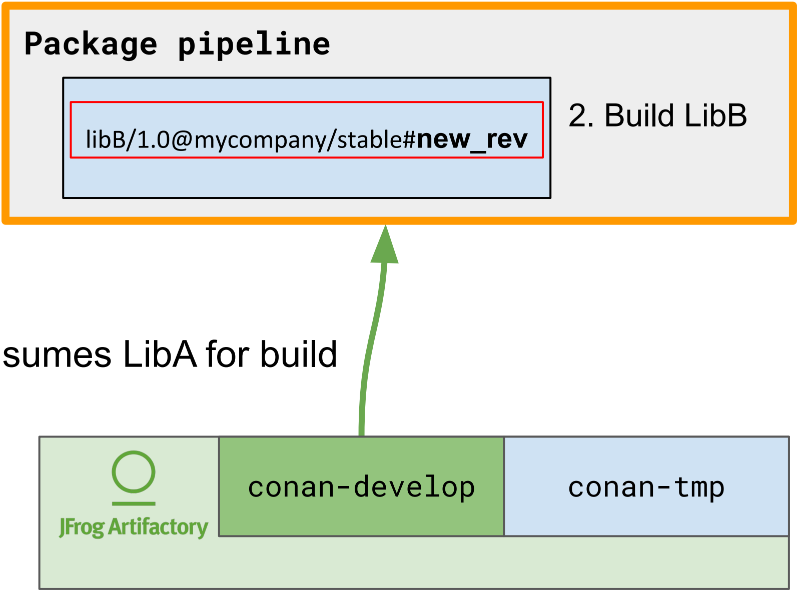
This branch has no conflicts with the base branch
Merging can be performed automatically.

Squash and merge ▾ You can also open this in GitHub Desktop or view command line instructions.





Phase 1: Developer works on a feature branch of libB



Package pipeline

libB/1.0@mycompany/stable#**new_rev**

2. Build LibB

1. Consumes LibA for build

Lab 2 - Create the library in the CI using lockfiles

Goal:

- Create a new libB with the modified sources of a developer’s feature branch

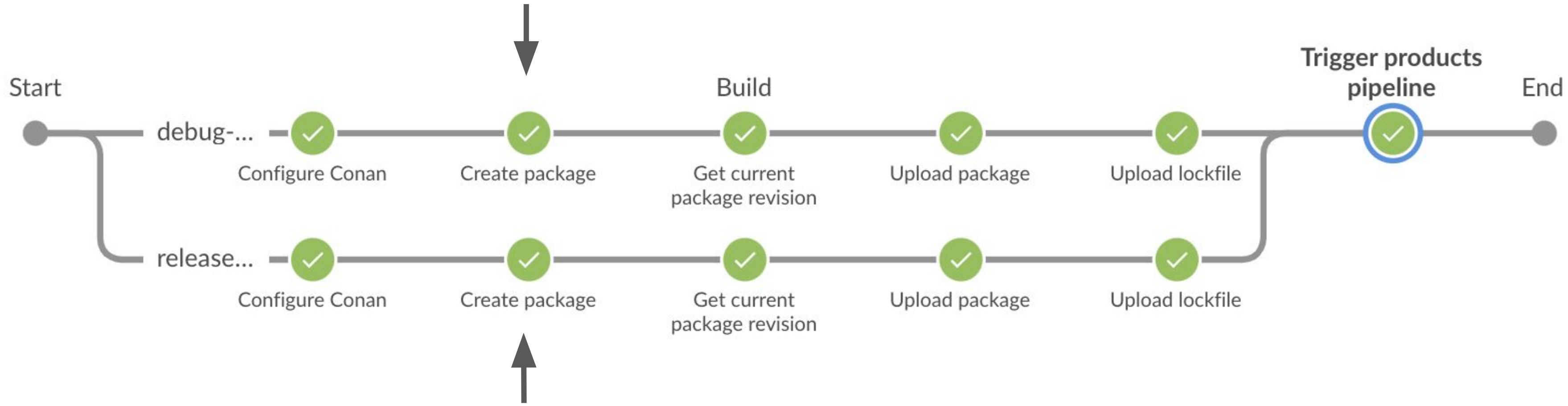
Task:

- Clone libB’s git repo and checkout the feature branch
- Calculate the graph for libB with all the latest requirements from conan-develop
- Build libB for different profiles using the lockfiles
- Search for libB in the local cache

Success:

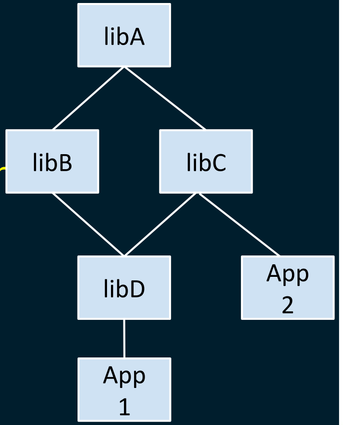
- Check that the new revision of libB is in the cache using the search command

Package (libB) pipeline



Lab 2 - Create the library in the CI using lockfiles

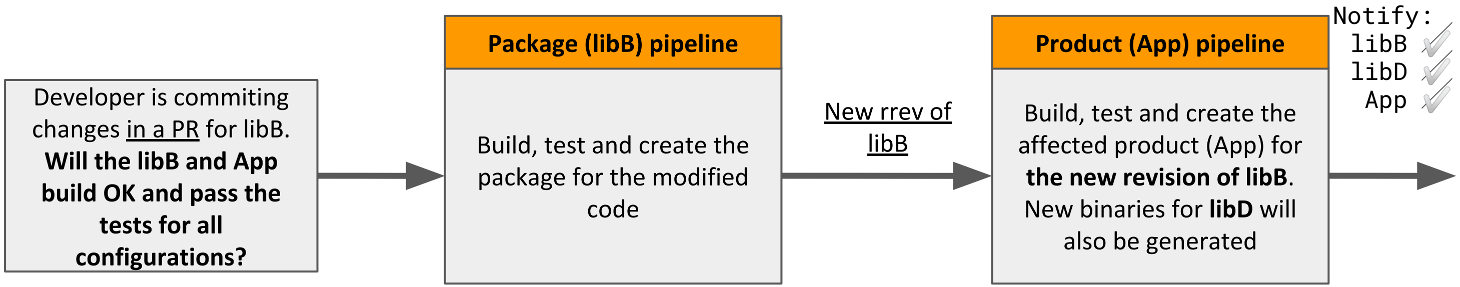
```
cd ../labs
git clone https://github.com/conan-ci-cd-training/libB.git
cd libB

# we work on our feature branch
git checkout feature/add_comments

# we want the library to be tested for different configurations → debug/release
# generate lockfiles for all configurations (debug and release)
conan graph lock libB/1.0@mycompany/stable --lockfile=../lockfiles/debug.lock -r
conan-develop --profile debug-gcc6
conan graph lock libB/1.0@mycompany/stable --lockfile=../lockfiles/release.lock -r
conan-develop --profile release-gcc6

# create packages with those lockfiles
conan create . mycompany/stable --lockfile=../lockfiles/debug.lock
conan create . mycompany/stable --lockfile=../lockfiles/release.lock
# check we have created a new revision of libB
conan search libB/1.0 --revisions
```



Outline

- Introduction
- Conan reminder: revisions, package id mode, lockfiles
- **CI: workflow phase 2**
- Build info in Artifactory
- Promotion in Artifactory
- Appendix

Phase 2: The developer opens a PR with a feature for libB



Developer is commiting changes in a PR for libB. **Will the libB and App build OK and pass the tests for all configurations?**

Package (libB) pipeline

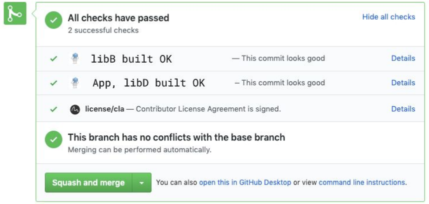
Build, test and create the package for the modified code

New rrev of libB

Product (App) pipeline

Build, test and create the affected product (App) for **the new revision of libB**. New binaries for **libD** will also be generated

```
Notify:
 libB ✓
 libD ✓
  App ✓
```



All checks have passed
2 successful checks

Hide all checks

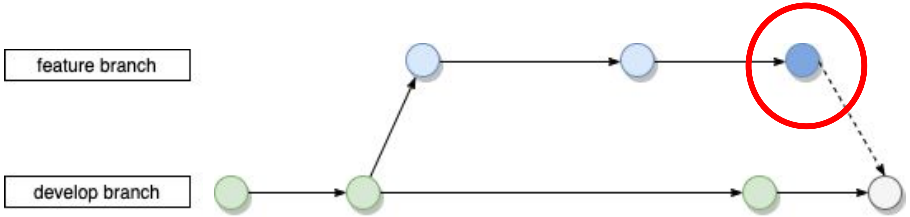
✓ `libB built OK`	— This commit looks good	Details
✓ `App, libD built OK`	– This commit looks good	Details
✓ **license/cla** — Contributor License Agreement is signed.		Details

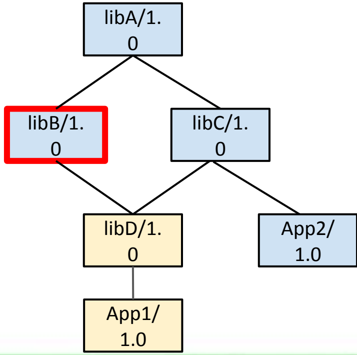
This branch has no conflicts with the base branch
Merging can be performed automatically.

You can also open this in GitHub Desktop or view command line instructions.

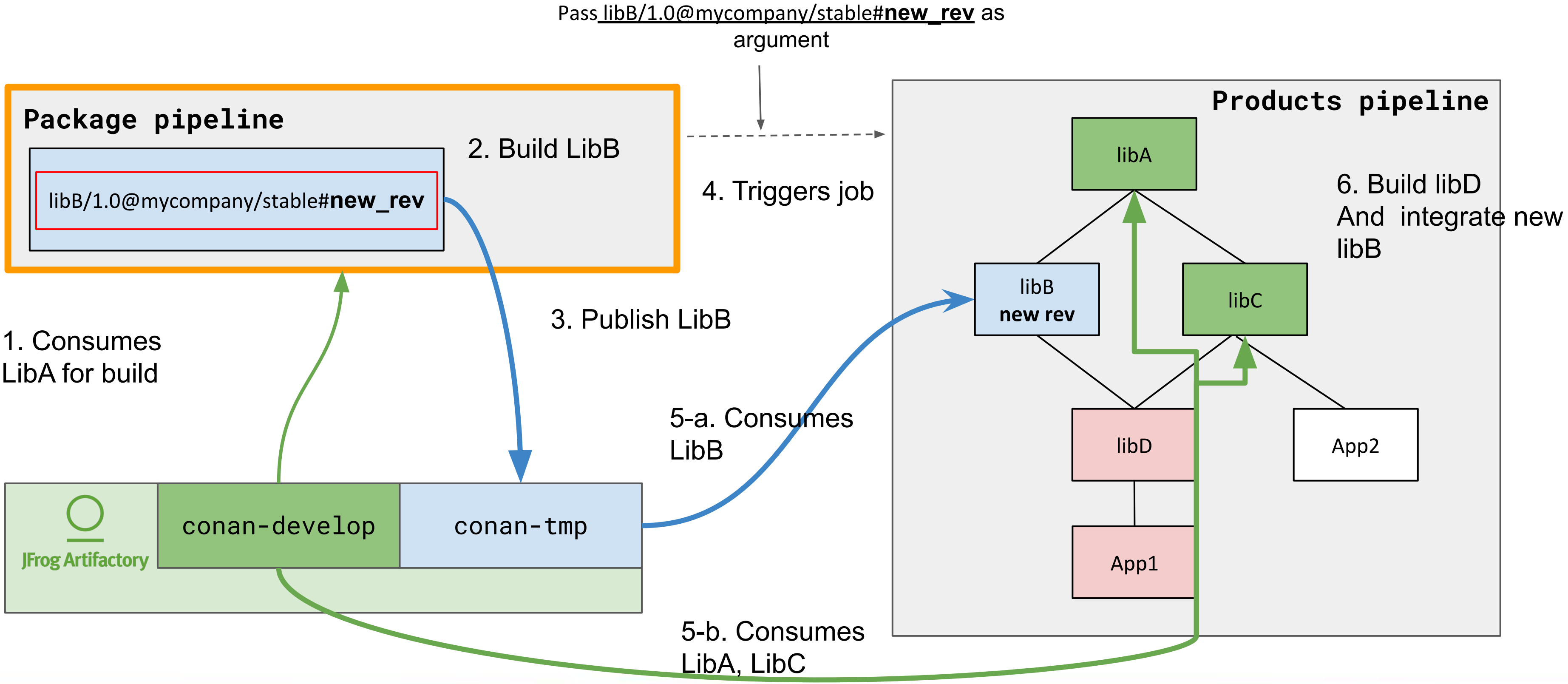
https://github.com/conan-ci-cd-training/libB/blob/develop/Jenkinsfile

https://github.com/conan-ci-cd-training/products/blob/master/Jenkinsfile





Phase 2: The developer opens a PR with a feature for libB



Lab 3 - Get the complete reference of the new libB

Goal:

- Getting the complete reference for the Conan package we have just created to upload use it as a parameter for the package’s pipeline and to upload to conan-tmp

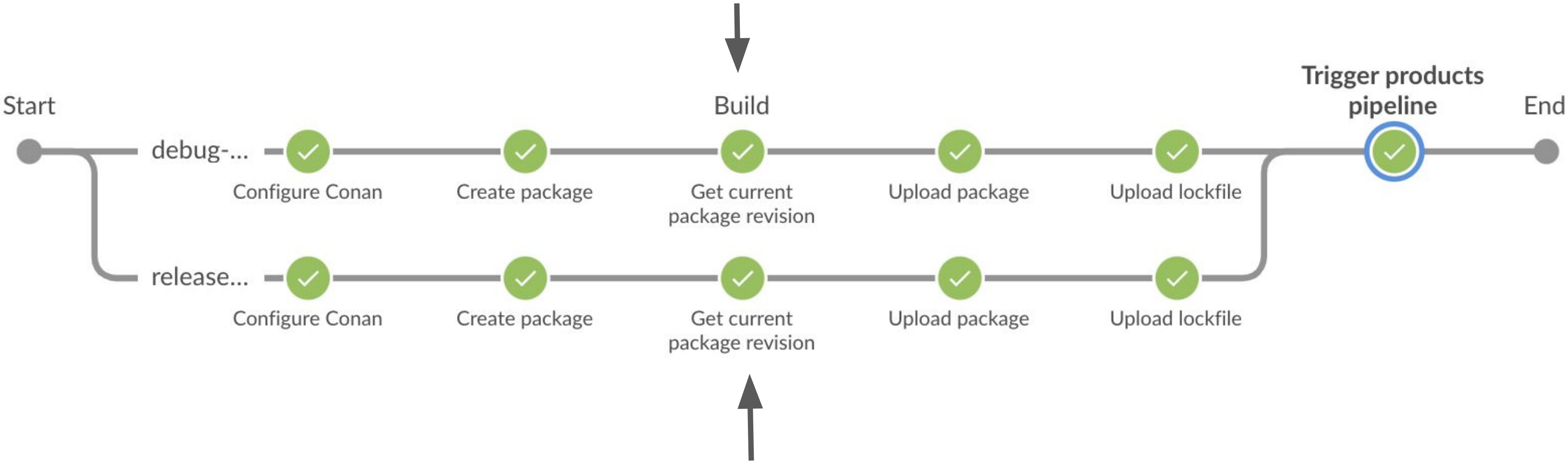
Task:

- Get the name of the recipe
- Get the version of the recipe
- Get the revision we have just created

Success:

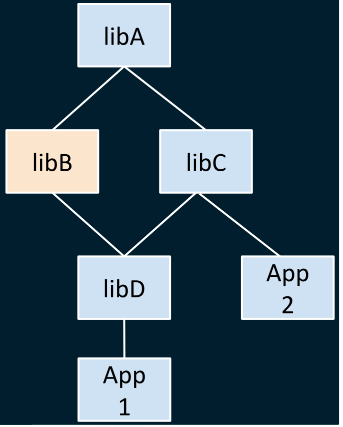
- Getting the data to construct the complete reference for the new libB: libB/1.0@mycompany/stable#a6c44191b4b5391c3678ae1d458375ec

Package (libB) pipeline



Lab 3: Get the complete reference of the new libB

```
cd ..

# get conan package <name> and <version>

conan inspect libB --raw name

conan inspect libB --raw version

# search with --revisions to get the newly created revision (remember only
one revision in the local cache)

conan search libB/1.0@mycompany/stable --revisions --raw
--json=libB_revision.json

cat libB_revision.json
```



Lab 4 - Upload new libB to conan-tmp

Goal:

- Uploading the new revision to repository conan-tmp so that we can get this library later from the product's pipeline

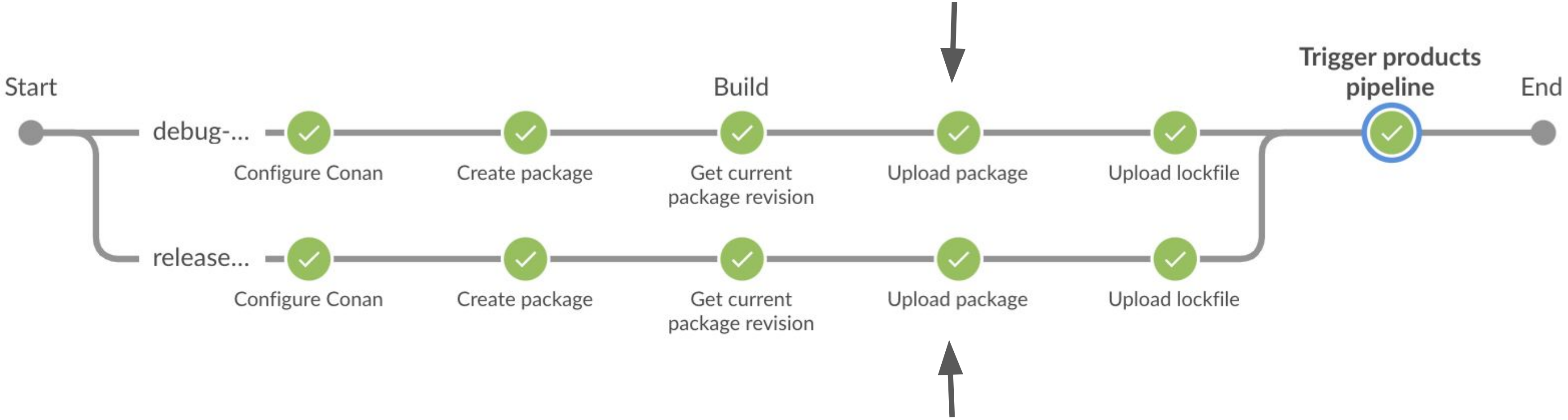
Task:

- Upload libB/1.0@mycompany/stable#new_rev to conan-tmp

Success:

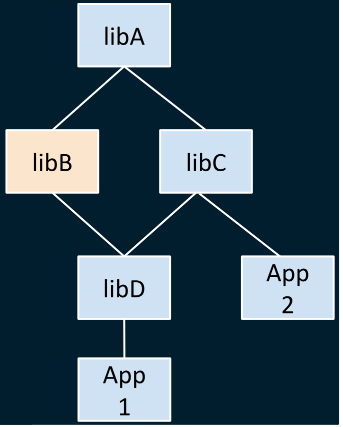
- Check that the new revision (**a6c44191b4b5391c3678ae1d458375ec**) of libB is in conan-tmp repo

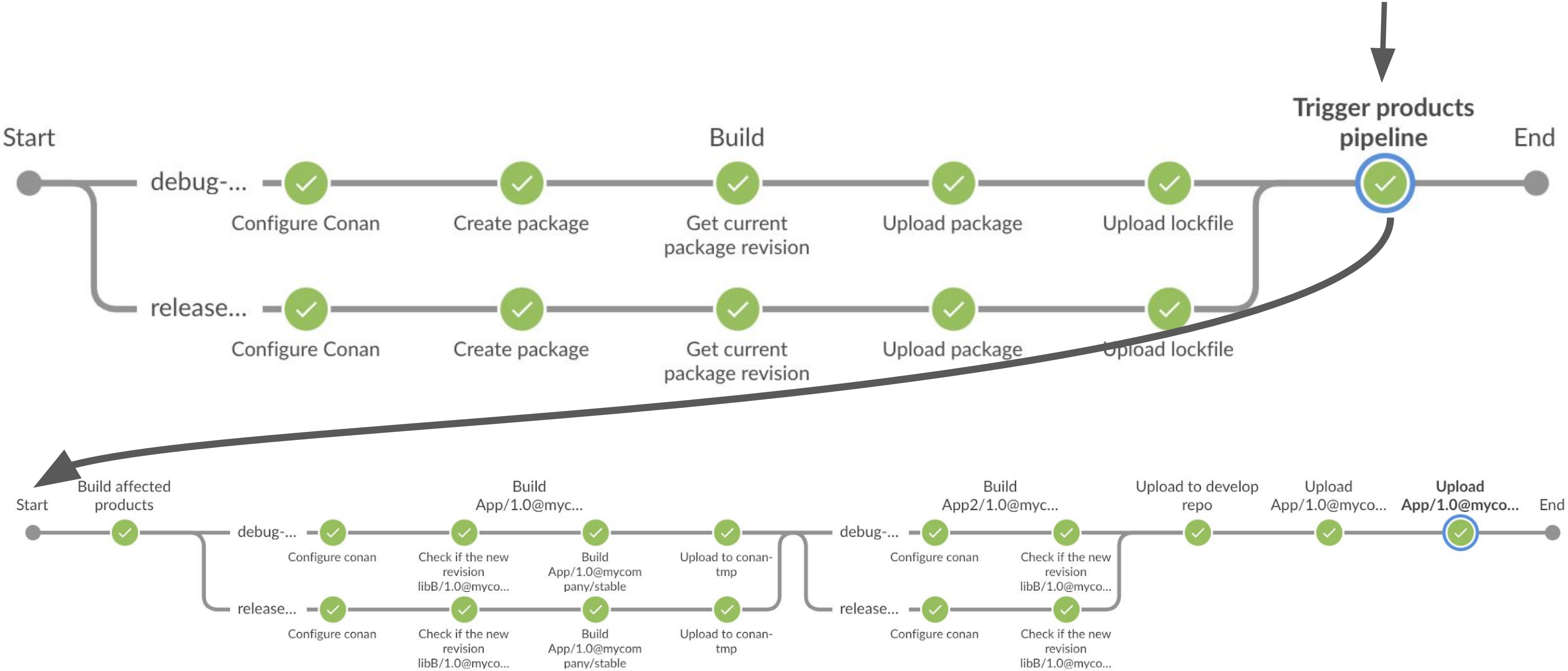
Package (libB) pipeline



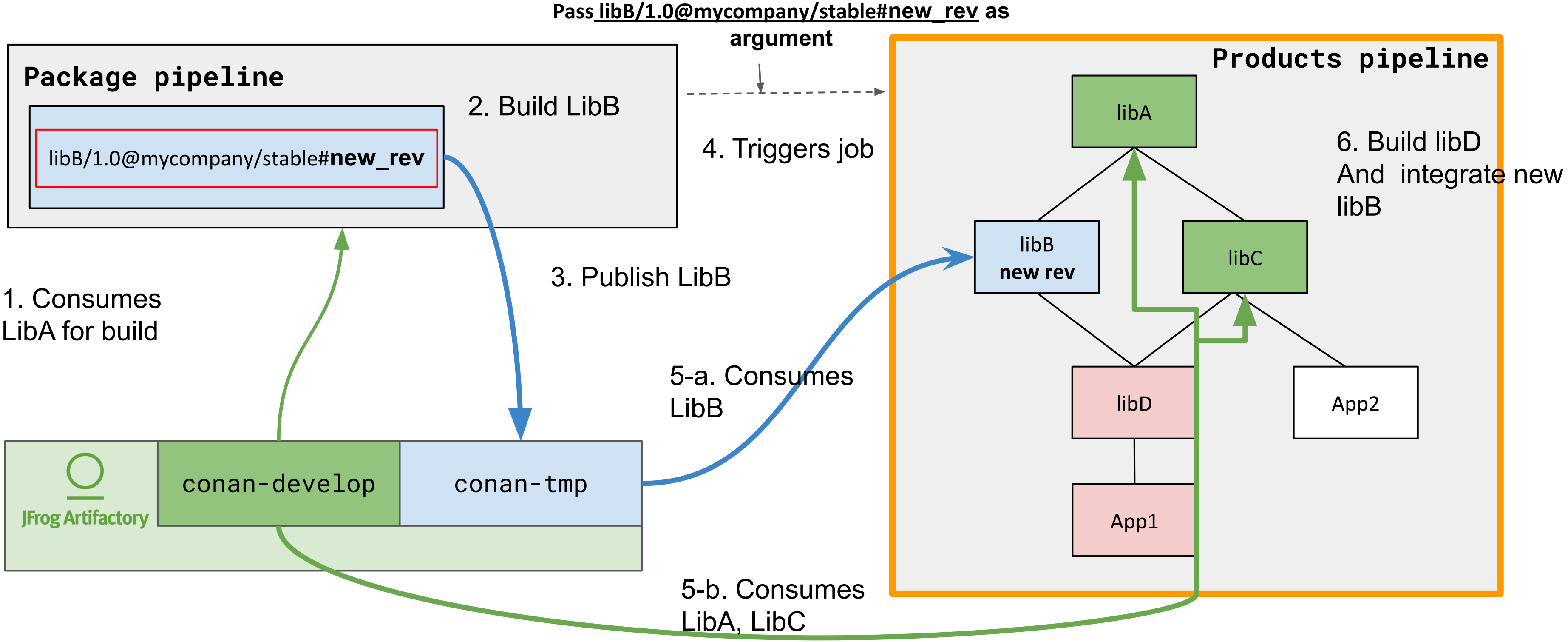
Lab 4: Upload new libB to conan-tmp

```
# upload the two generated packages for the new revisions of libB to
conan-tmp

conan upload libB/1.0@mycompany/stable#<new_revision> --all -r
conan-tmp --confirm

# now we are ready to launch the product pipeline

conan search libB/1.0@mycompany/stable -r conan-tmp --revisions
```




Start
Build
Trigger products pipeline
End
debug-...
Configure Conan
Create package
Get current package revision
Upload package
Upload lockfile
release...
Configure Conan
Create package
Get current package revision
Upload package
Upload lockfile
Start
Build affected products
Build App/1.0@myc...
Build App2/1.0@myc...
Upload to develop repo
Upload App/1.0@myco...
Upload App/1.0@myco...
End
debug-...
Configure conan
Check if the new revision libB/1.0@myco...
Build App/1.0@mycompany/stable
Upload to conan-tmp
release...
Configure conan
Check if the new revision libB/1.0@myco...
Build App/1.0@mycompany/stable
Upload to conan-tmp
debug-...
Configure conan
Check if the new revision libB/1.0@myco...
release...
Configure conan
Check if the new revision libB/1.0@myco...

Phase 2: The developer opens a PR with a feature for libB



Pass libB/1.0@mycompany/stable#new_rev as argument

Package pipeline

libB/1.0@mycompany/stable#**new_rev**

2. Build LibB

4. Triggers job

3. Publish LibB

1. Consumes LibA for build

5-a. Consumes LibB

5-b. Consumes LibA, LibC

Products pipeline

6. Build libD And integrate new libB

Lab 5 - Check if App/App2 need to be rebuilt

Goal:

- See if the new revision of libB is affecting App or App2 so that they have to be rebuilt

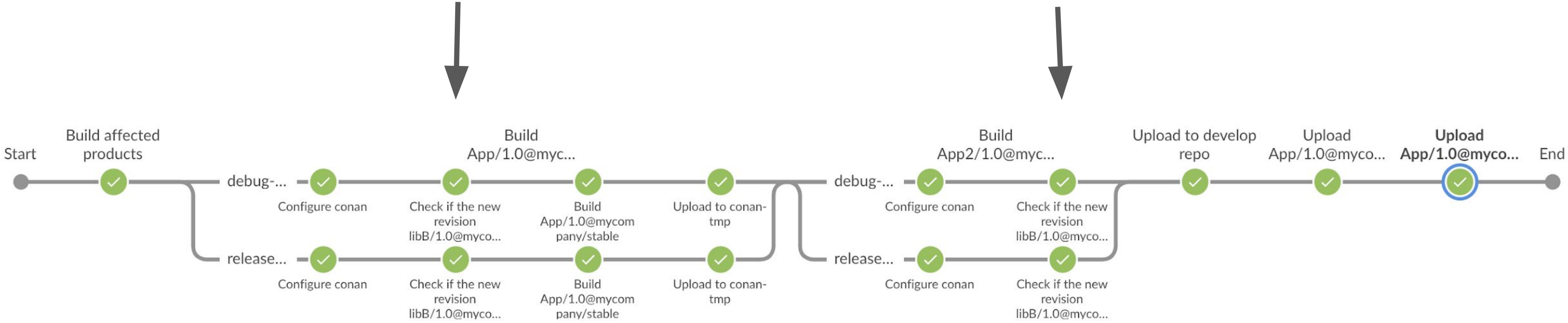
Task:

- Download the recipe for the new revision of libB
- Do the graph lock for each product using the conan-develop remote (latest revisions of libs)
- Calculate the build order with --build missing, will tell us if the new revision of libB is affecting App or App2

Success:

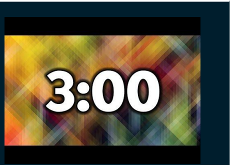
- The build order for App contains libD and App
- The build order for App2 is empty

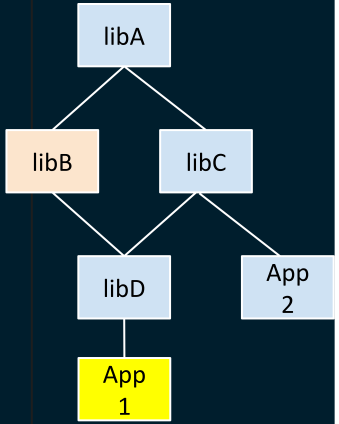
Products (App and App2) pipeline



Lab 5.a: Check if App is affected

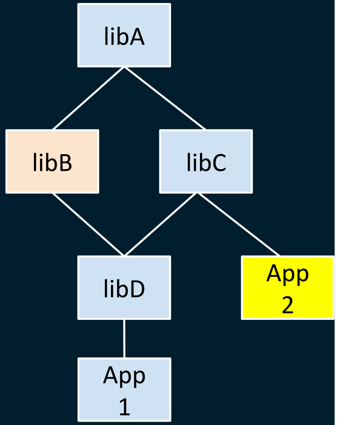
```
# In the CI we would be in other job with a clean conan cache
conan remove "*" -f

# update cache with a specific revision of libB (doesn’t update libA
in the cache)

conan download libB/1.0@mycompany/stable#<new revision> -r conan-tmp
--recipe

# check App
conan graph lock App/1.0@mycompany/stable --profile=release-gcc6
--lockfile=app_release.lock -r conan-develop

conan graph build-order app_release.lock --build missing --json
app_bo.json

cat app_bo.json
```





Lab 5.b: Check if App2 is affected

```
# we already have libB in the cache

# check App2

conan graph lock App2/1.0@mycompany/stable
--profile=release-gcc6 --lockfile=app2_release.lock -r
conan-develop

conan graph build-order app2_release.lock --build missing --json
app2_bo.json

cat app2_bo.json
```



Lab 6 - Build the graph using the lockfile

Goal:

- Follow the build order of App we got in lab5 → build libD and App

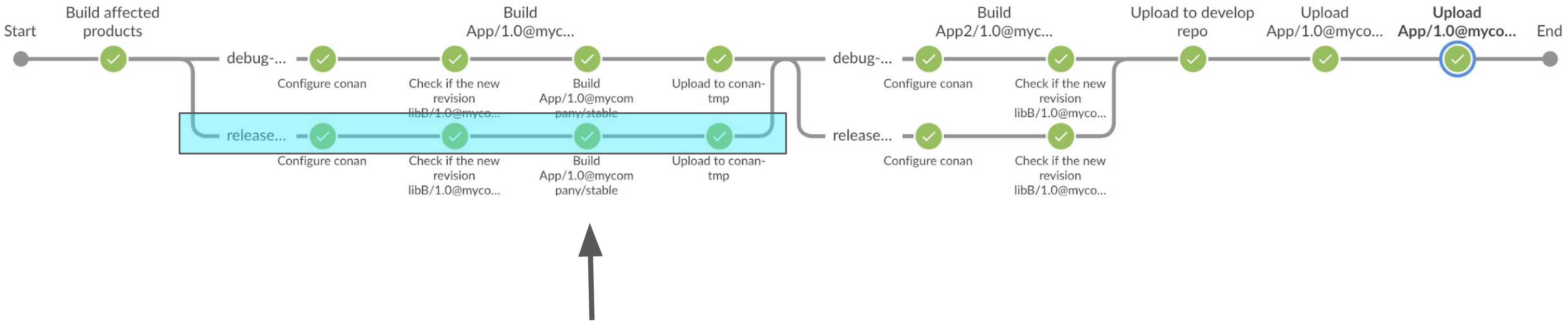
Task:

- Calculate the build order using the lockfile
- Build libD → update lockfile
- Build App → update lockfile
- Recalculate build order

Success:

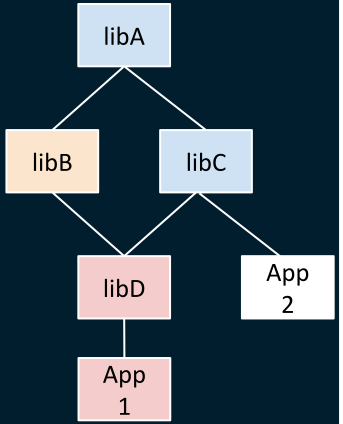
- After building App we recalculate the build order and the output is empty

Products (App and App2) pipeline



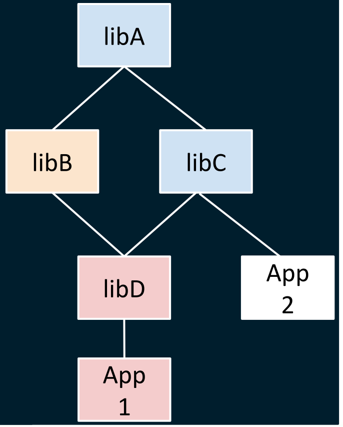
Lab 6.a - Build the graph using the lockfile

```
conan graph build-order app_release.lock --build missing --json
app_bo.json

# use the build-order → build D
cp app_release.lock conan.lock

# install libD with the lockfile. libD is marked as built in
conan.lock

conan install libD/1.0@mycompany/stable --build libD --lockfile
conan.lock

# update the original lockfile with update-lock
conan graph update-lock app_release.lock conan.lock

cat app_release.lock
```



Lab 6.b - Build the graph using the lockfile

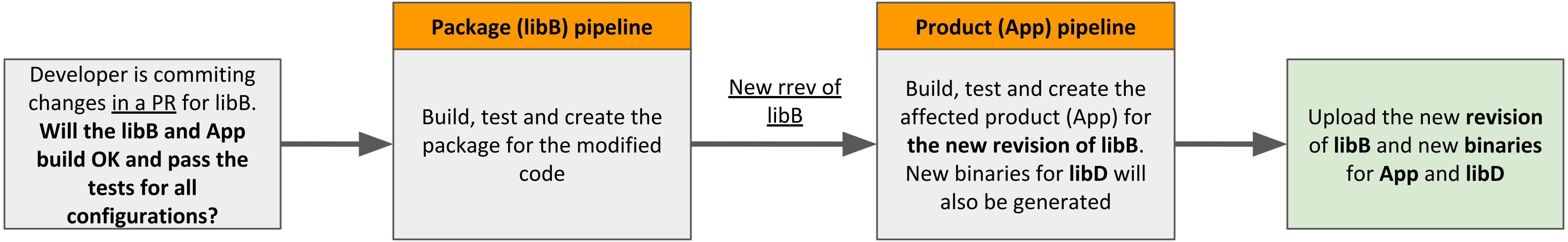
```
# the build order with the updated lockfile → build App
cp app_release.lock conan.lock

conan install App/1.0@mycompany/stable --build App --lockfile
conan.lock

conan graph update-lock app_release.lock conan.lock

conan graph build-order app_release.lock --build missing

cat app_release.lock
```



Outline

- Introduction
- Conan reminder: revisions, package id mode, lockfiles
- **CI: workflow phase 3**
- Build info in Artifactory
- Promotion in Artifactory
- Appendix

Phase 3: PR is merged in the target branch



Developer is commiting changes in a PR for libB. **Will the libB and App build OK and pass the tests for all configurations?**

Package (libB) pipeline

Build, test and create the package for the modified code

New rrev of libB

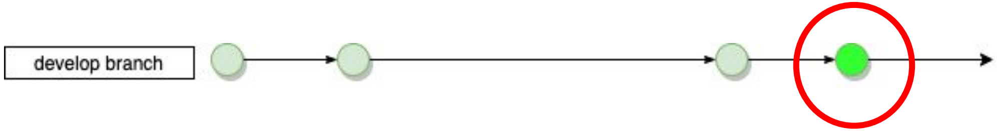
Product (App) pipeline

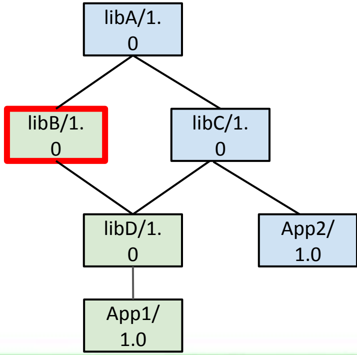
Build, test and create the affected product (App) for **the new revision of libB**. New binaries for **libD** will also be generated

Upload the new **revision** of **libB** and new **binaries** for **App** and **libD**

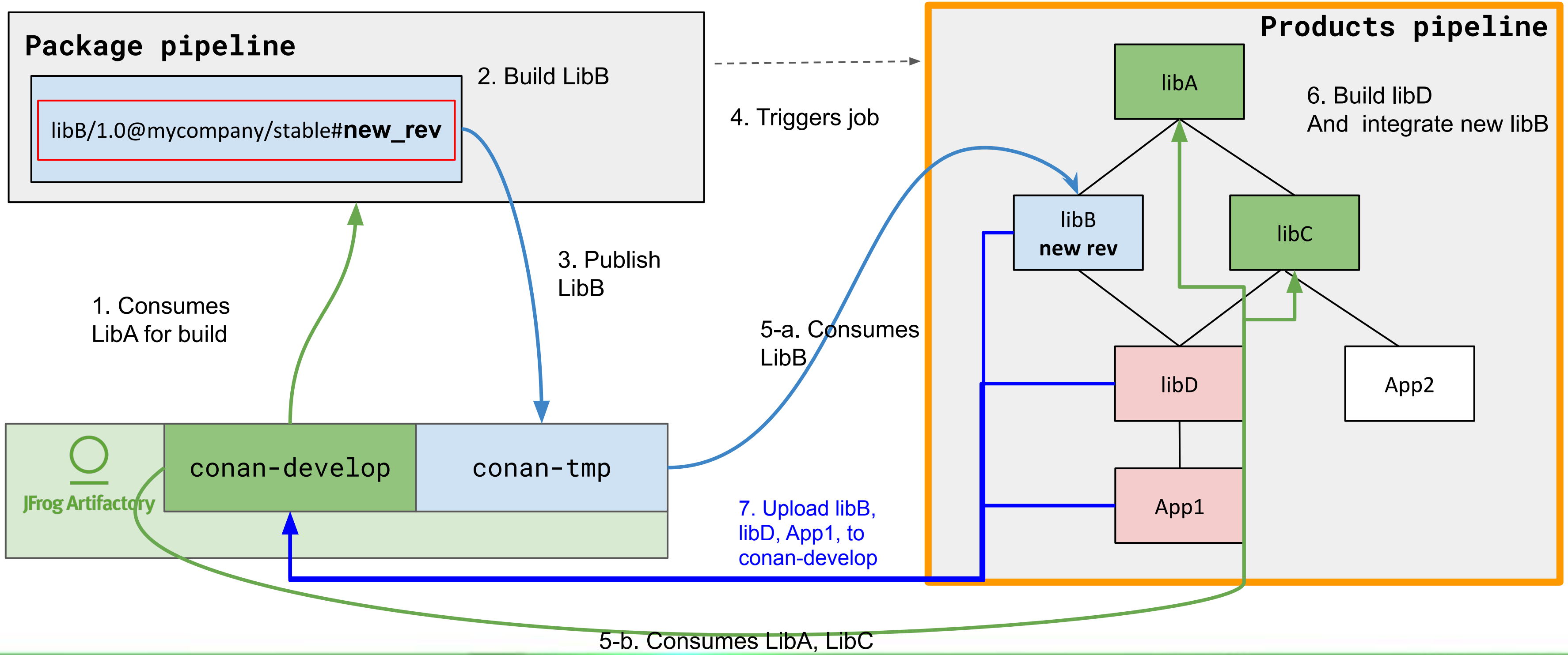
https://github.com/conan-ci-cd-training/libB/blob/develop/Jenkinsfile

https://github.com/conan-ci-cd-training/products/blob/master/Jenkinsfile





Phase 3: PR is merged in the target branch



Lab 7 - Upload new packages to conan-develop

Goal:

- Making the new binaries available for all developers so they don’t have to rebuild in their own machines

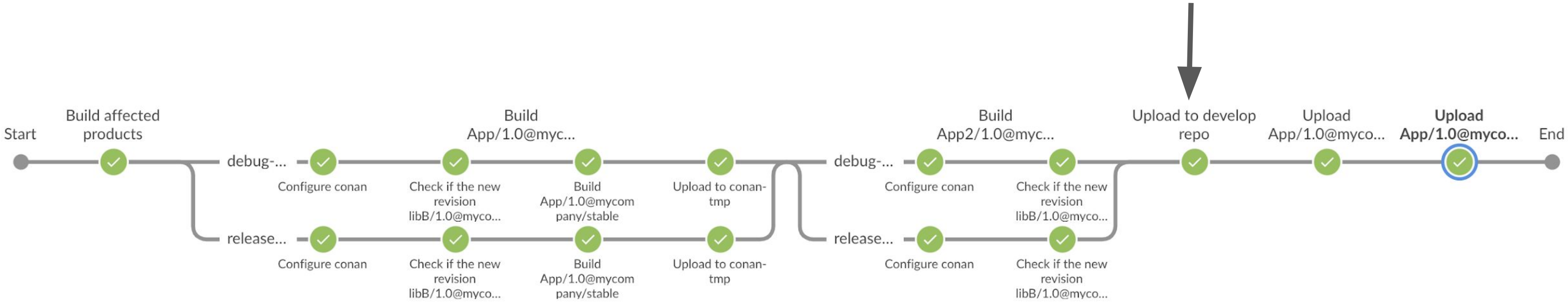
Task:

- Upload new revision of libB and new binaries of libD and App to conan-develop

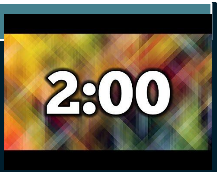
Success:

- All the new binaries are uploaded

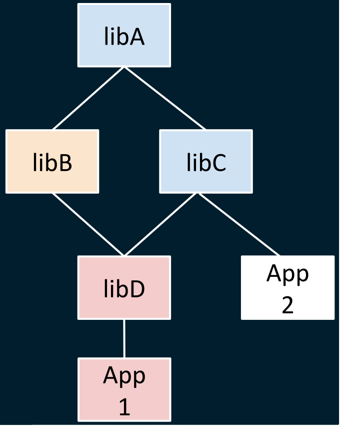
Products (App and App2) pipeline



Lab 7 - Upload new packages to conan-develop



```
# all new revisions and binaries will be uploaded

conan upload "*" -r conan-develop --confirm  --all --force
```



Lab 8 - Upload lockfile to conan-metadata

Goal:

- Storing lockfiles in Artifactory in case we want to use them later to install conan packages or generating build info.

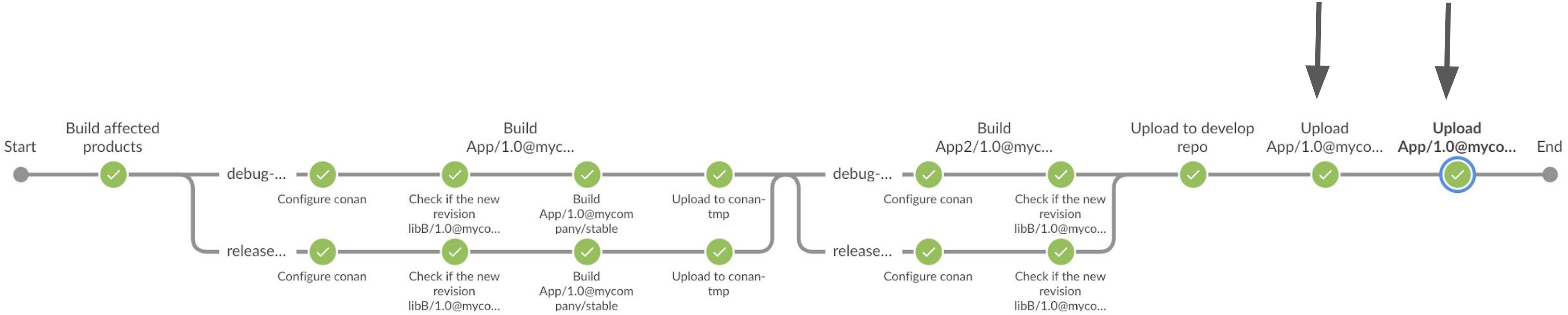
Task:

- Use the Artifactory API to upload the file to conan-metadata (generic repo)

Success:

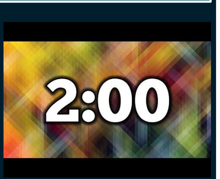
- Check that the file has been uploaded to conan-metadata

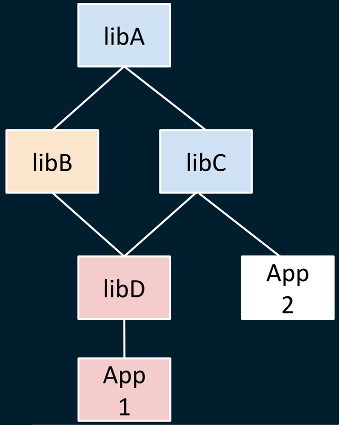
Products (App and App2) pipeline



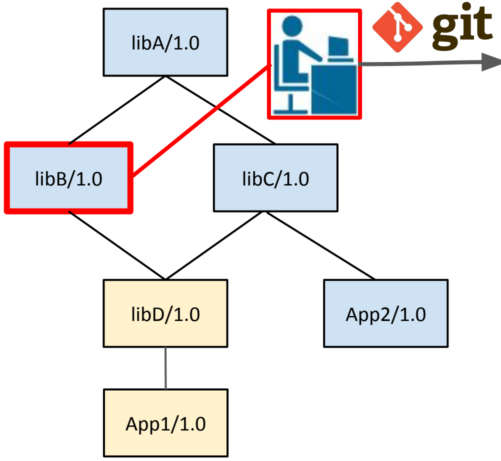
Lab 8 - Upload lockfile to conan-metadata

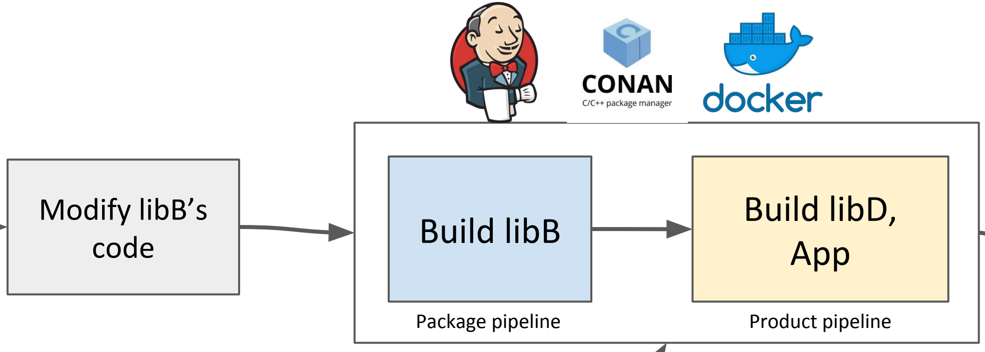
```
# upload the lockfile to conan-metadata repo
curl -u conan:conan2020 -X PUT
http://jfrog.local:8081/artifactory/conan-metadata/App/1.0@mycom
pany/stable/conan-app/1/gcc6-release/ -T app_release.lock

# assign properties
curl -u conan:conan2020 -X PUT
http://jfrog.local:8081/artifactory/api/storage/conan-metadata/A
pp/1.0@mycompany/stable/conan-app/1/gcc6-release/app_release.loc
k?properties=build.name=conan-app%7Cbuild.number=1%7Cprofile=gcc
6-release%7Capp.version=1.0
```

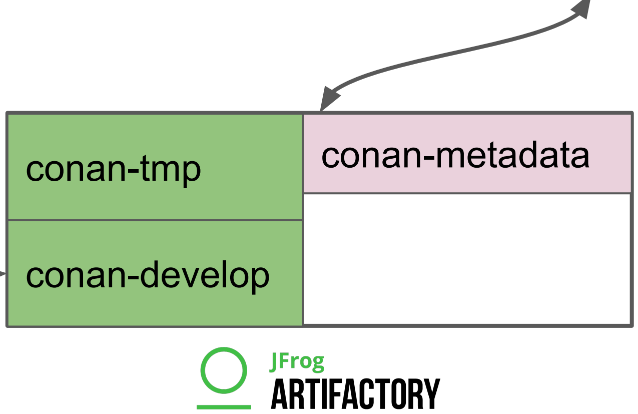


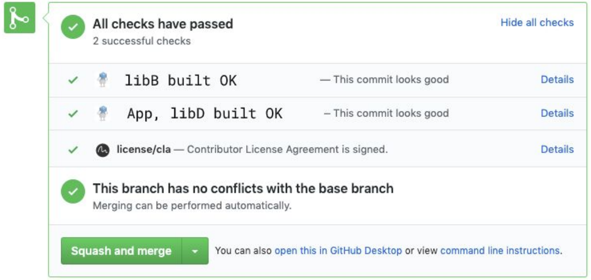


Recap









All checks have passed
2 successful checks
Hide all checks

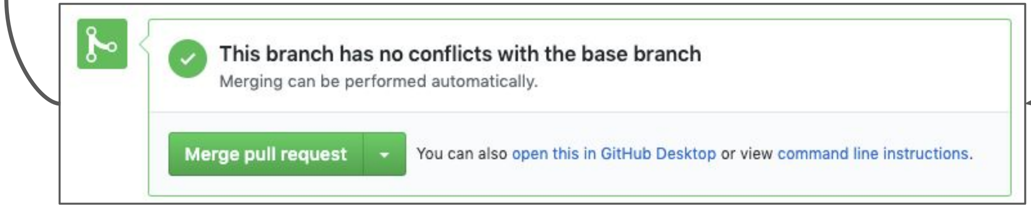
✓ libB built OK — This commit looks good Details

✓ App, libD built OK – This commit looks good Details

✓ license/cla — Contributor License Agreement is signed. Details

This branch has no conflicts with the base branch
Merging can be performed automatically.

Squash and merge ▾ You can also open this in GitHub Desktop or view command line instructions.



This branch has no conflicts with the base branch
Merging can be performed automatically.

Merge pull request ▾ You can also open this in GitHub Desktop or view command line instructions.

Too many binaries ...

- A specific revision of libA, which versions of App is using it ?
- Which CI build generated which packages ?
- Which RREV contains all the fully qualified packages
- For which architecture(s) and OS, did I build my App for ?
- ...

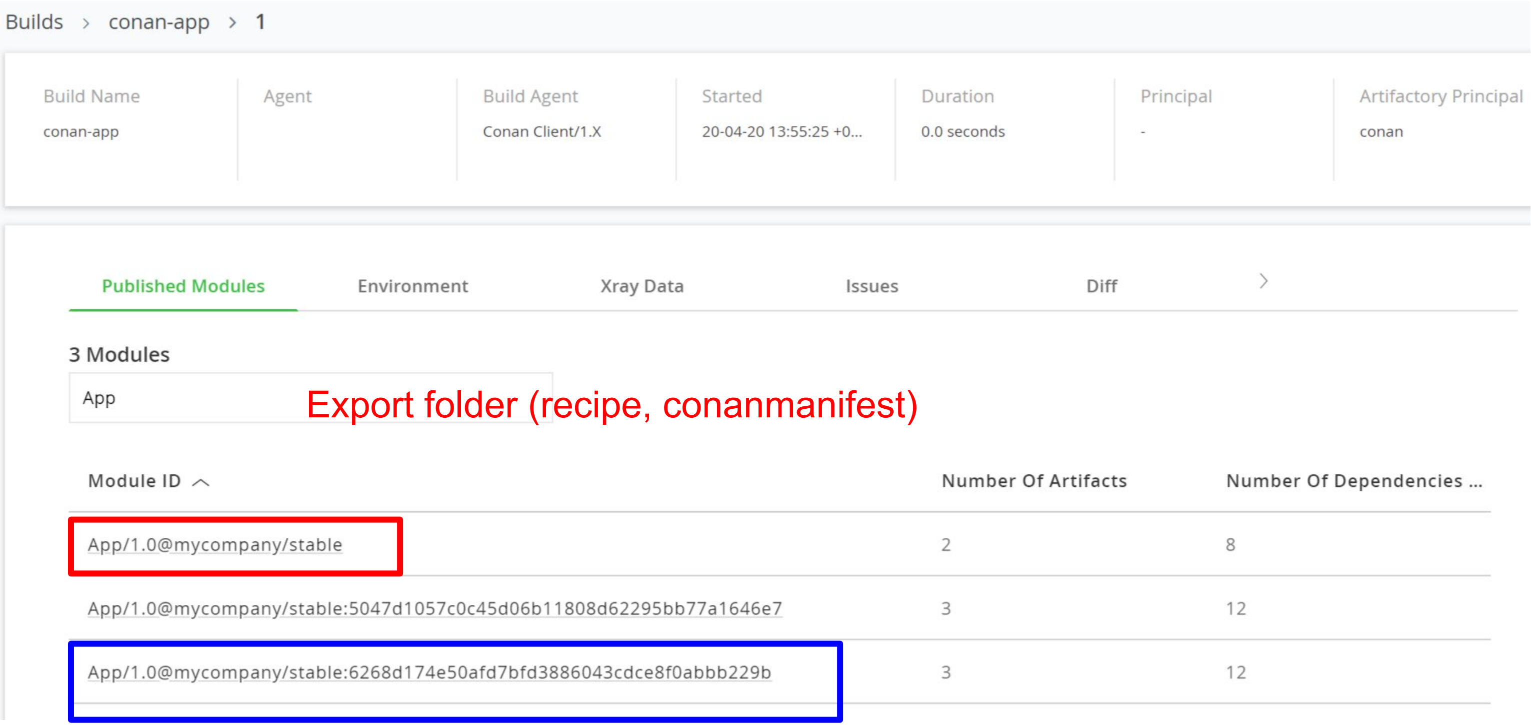
Outline

- Introduction
- Conan reminder: revisions, package id mode, lockfiles
- CI
- **Build info in Artifactory**
- Promotion in Artifactory
- Appendix

Build Info - Intro

- Bill Of Material (JSON file) listing **generated binaries and consumed dependencies**
- Can be built from a Lockfile (For Conan)
- Generated and published by the **conan_build_info** client, **CI plugins** and **JFrog CLI**
  - Only Jenkins and Azure devops plugins have specific instruction for Conan
- Possibility to merge multiple Build Info via the **conan_build_info** client

Build Info - Intro



Builds › conan-app › 1

Build Name	Agent	Build Agent	Started	Duration	Principal	Artifactory Principal
conan-app		Conan Client/1.X	20-04-20 13:55:25 +0...	0.0 seconds	-	conan

3 Modules

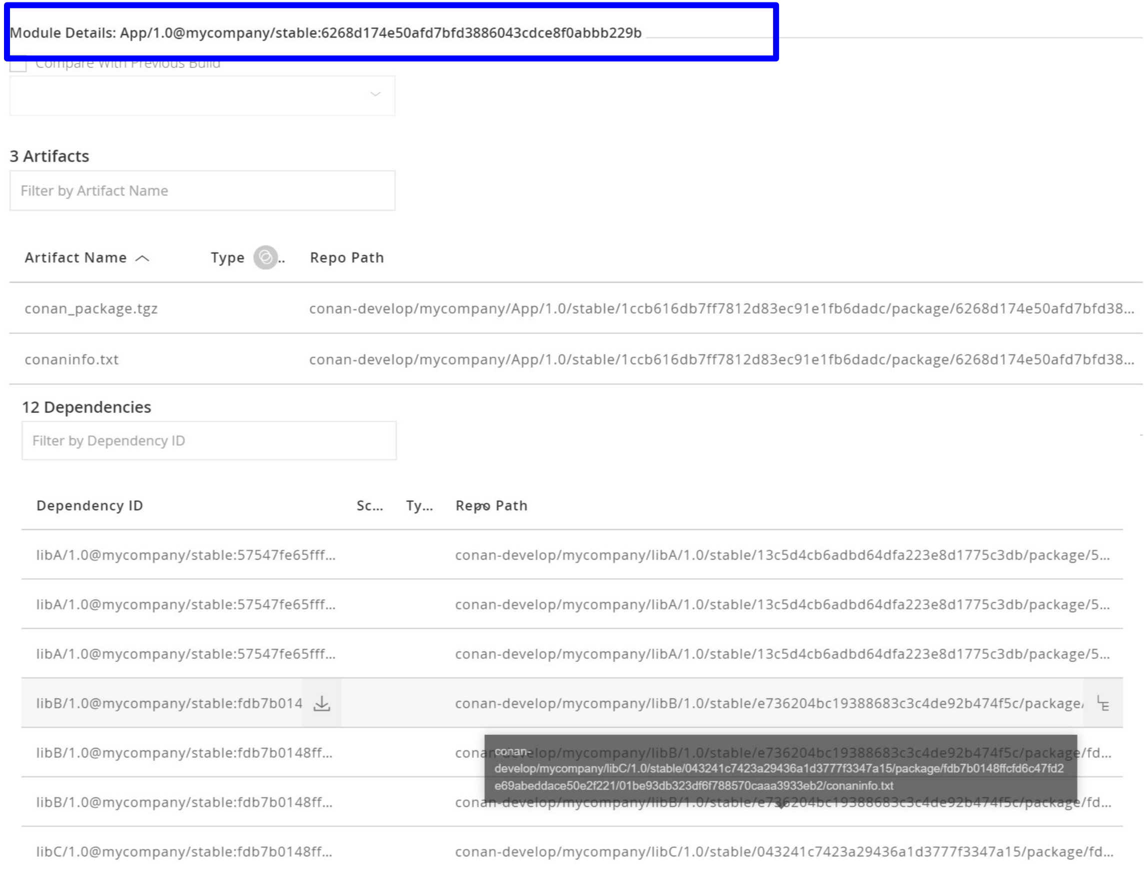
App

Export folder (recipe, conanmanifest)

Module ID	Number Of Artifacts	Number Of Dependencies ...
App/1.0@mycompany/stable	2	8
App/1.0@mycompany/stable:5047d1057c0c45d06b11808d62295bb77a1646e7	3	12
App/1.0@mycompany/stable:6268d174e50afd7bfd3886043cdce8f0abbb229b	3	12

Package folder (binaries)

Build Info - Intro



Module Details: App/1.0@mycompany/stable:6268d174e50afd7bfd3886043cdce8f0abbb229b

□ Compare With Previous Build

3 Artifacts

Filter by Artifact Name

Artifact Name	Type	Repo Path
conan_package.tgz		conan-develop/mycompany/App/1.0/stable/1ccb616db7ff7812d83ec91e1fb6dadc/package/6268d174e50afd7bfd38...
conaninfo.txt		conan-develop/mycompany/App/1.0/stable/1ccb616db7ff7812d83ec91e1fb6dadc/package/6268d174e50afd7bfd38...

12 Dependencies

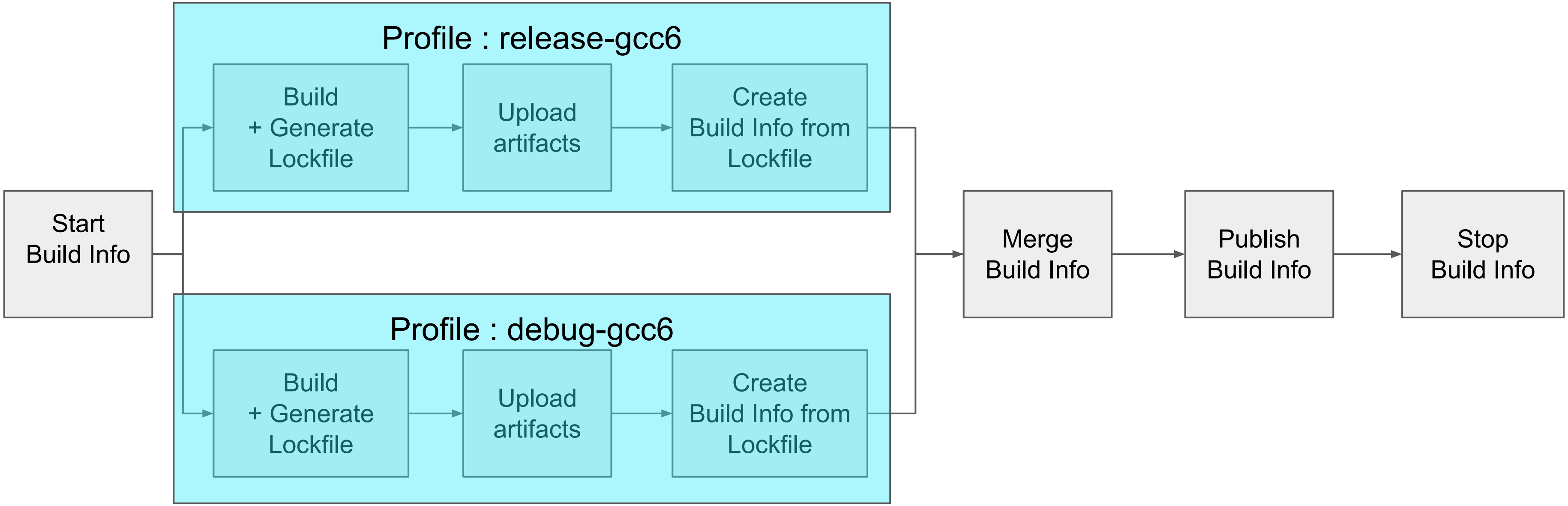
Filter by Dependency ID

Dependency ID	Sc...	Ty...	Repo Path
libA/1.0@mycompany/stable:57547fe65fff...			conan-develop/mycompany/libA/1.0/stable/13c5d4cb6adbd64dfa223e8d1775c3db/package/5...
libA/1.0@mycompany/stable:57547fe65fff...			conan-develop/mycompany/libA/1.0/stable/13c5d4cb6adbd64dfa223e8d1775c3db/package/5...
libA/1.0@mycompany/stable:57547fe65fff...			conan-develop/mycompany/libA/1.0/stable/13c5d4cb6adbd64dfa223e8d1775c3db/package/5...
libB/1.0@mycompany/stable:fdb7b014			conan-develop/mycompany/libB/1.0/stable/e736204bc19388683c3c4de92b474f5c/package/
libB/1.0@mycompany/stable:fdb7b0148ff...			conan-develop/mycompany/libB/1.0/stable/e736204bc19388683c3c4de92b474f5c/package/fd...
libB/1.0@mycompany/stable:fdb7b0148ff...			conan-develop/mycompany/libB/1.0/stable/e736204bc19388683c3c4de92b474f5c/package/fd...
libC/1.0@mycompany/stable:fdb7b0148ff...			conan-develop/mycompany/libC/1.0/stable/043241c7423a29436a1d3777f3347a15/package/fd...

conan-develop/mycompany/libC/1.0/stable/043241c7423a29436a1d3777f3347a15/package/fdb7b0148ffcfd6c47fd2e69abeddace50e2f221/01be93db323df6f788570caaa3933eb2/conaninfo.txt

All artifacts have to be in Artifactory !

Build info in parallel pipelines



Lab9 - Generate a Build Info for App

Goal:

- Create a Build Info for the App1 conan package using the **conan_build_info** client

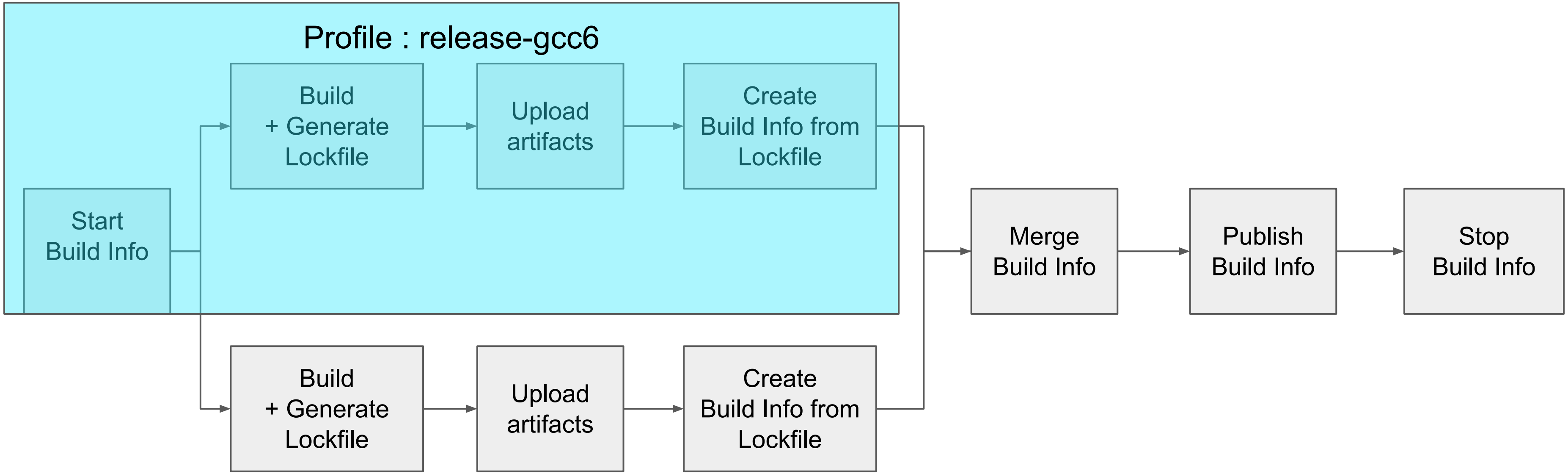
Task:

- Enable Build Info recording (start/stop instruction)
- Create and upload App in Release mode + Generate the Build Info (create instruction) from a lockfile
- Create and upload App in Debug mode + Generate the Build Info (create instruction) from a lockfile

Success:

- Check the generated json files

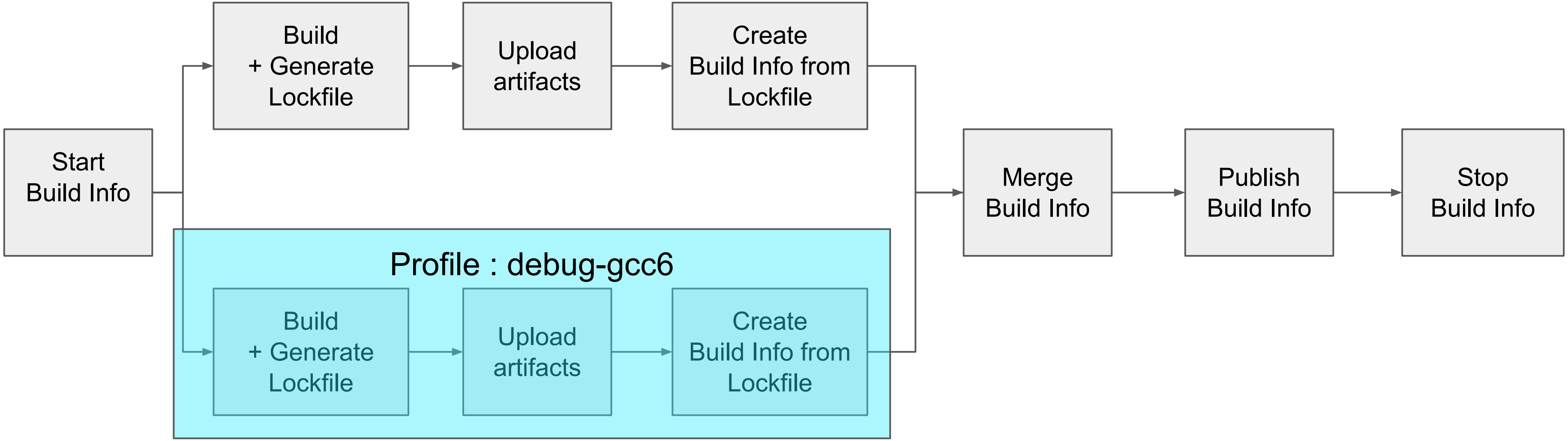
Lab9.a - Generate a Build Info for App1 for Release



Lab9.a - Generate a Build Info for App1 for Release

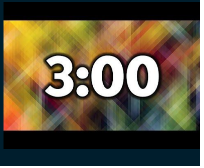
```
# disable/enable build properties
conan_build_info --v2 stop && cat ~/.conan/artifacts.properties
conan_build_info --v2 start conan-app 1 && cat ~/.conan/artifacts.properties

# create build info for release from the release lockfile for App1
conan_build_info --v2 create release_bi.json --lockfile=app_release.lock
--user=conan --password=conan2020 && cat release_bi.json
```

Lab9.b - Generate a Build Info for App1 for Debug

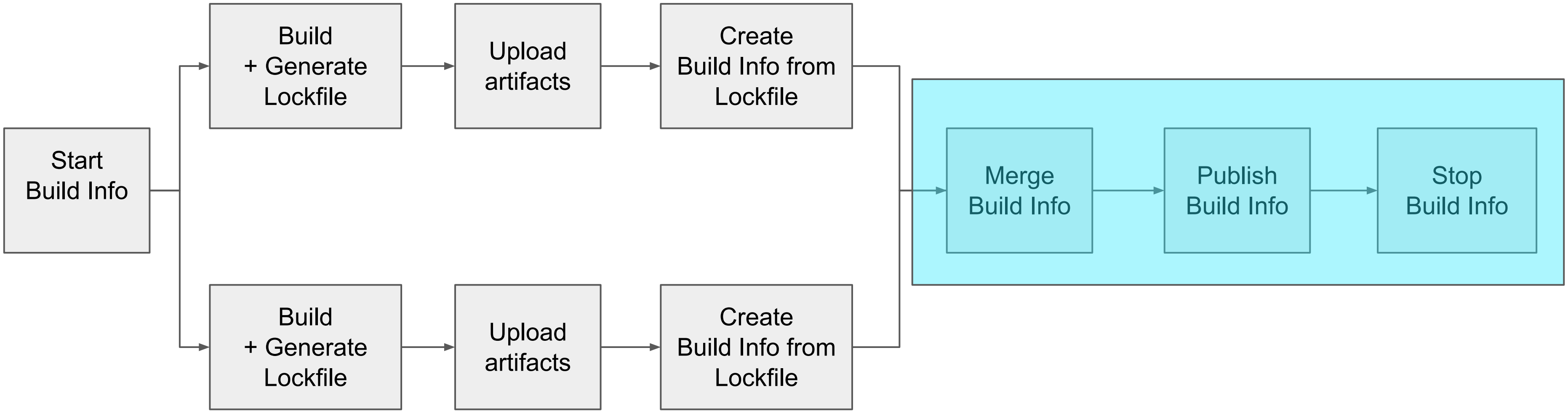


Lab9.b - Generate a Build Info for App1 in Debug

```
# redo lab 6 to generate libs in Debug + lab 7 to upload App in Debug
# current path : ~/conan_ci_cd/labs
./genAppDebug.sh

# create build info
conan_build_info --v2 create debug_bi.json --lockfile=app_debug.lock
--user=conan --password=conan2020 && cat debug_bi.json
```



Lab10 - Merge and publish a Build Info



Lab10 - Merge and publish a Build Info

Goal:

- Merge 2 Build Info for App1 and publish to Artifactory

Task:

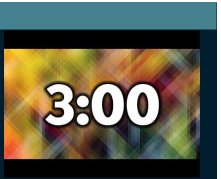
- Use update and publish instructions

Success:

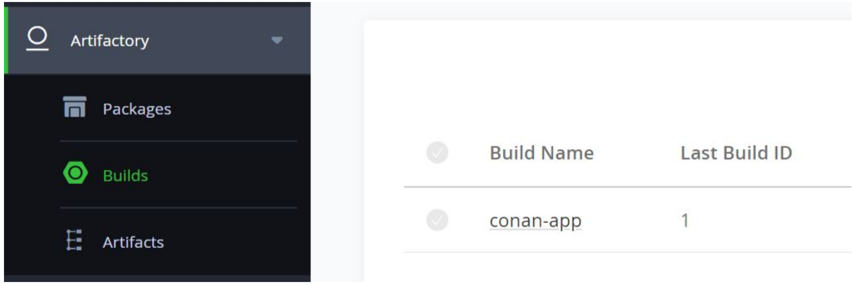
- See the Build Info for 2 profiles in Artifactory

Lab10 - Merge and publish a Build Info

```
# create the aggregated build info
conan_build_info --v2 update --output-file app_bi.json debug_bi.json
release_bi.json && cat app_bi.json

# publish the build info and remove build properties
conan_build_info --v2 publish app_bi.json
--url=http://jfrog.local:8081/artifactory --user=conan --password=conan2020

conan_build_info --v2 stop && cat ~/.conan/artifacts.properties
```



Lab10 - Merge and publish a Build Info

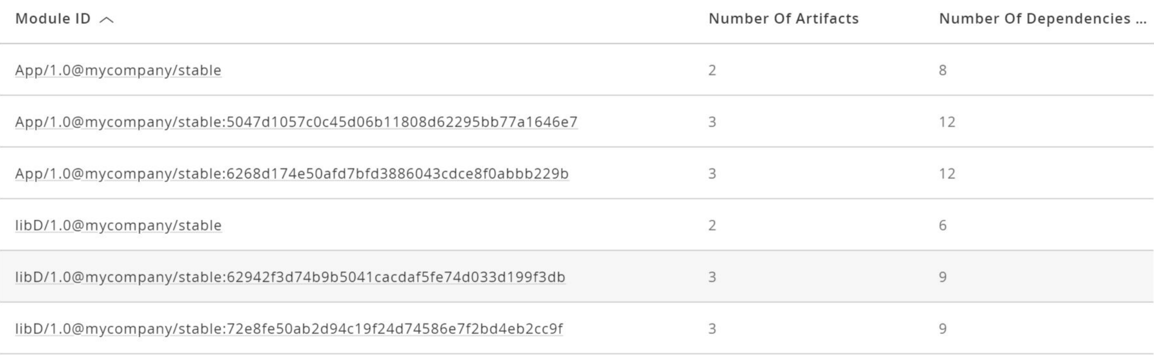


	Build Name	Last Build ID
✓	conan-app	1

Go to “Build” section and select conan-app

Check the Build Info content :

Generated modules in Build Info are libD and App (built components)



Module ID	Number Of Artifacts	Number Of Dependencies ...
App/1.0@mycompany/stable	2	8
App/1.0@mycompany/stable:5047d1057c0c45d06b11808d62295bb77a1646e7	3	12
App/1.0@mycompany/stable:6268d174e50afd7bfd3886043cdce8f0abbb229b	3	12
libD/1.0@mycompany/stable	2	6
libD/1.0@mycompany/stable:62942f3d74b9b5041cacdaf5fe74d033d199f3db	3	9
libD/1.0@mycompany/stable:72e8fe50ab2d94c19f24d74586e7f2bd4eb2cc9f	3	9

Build Info - Good to know

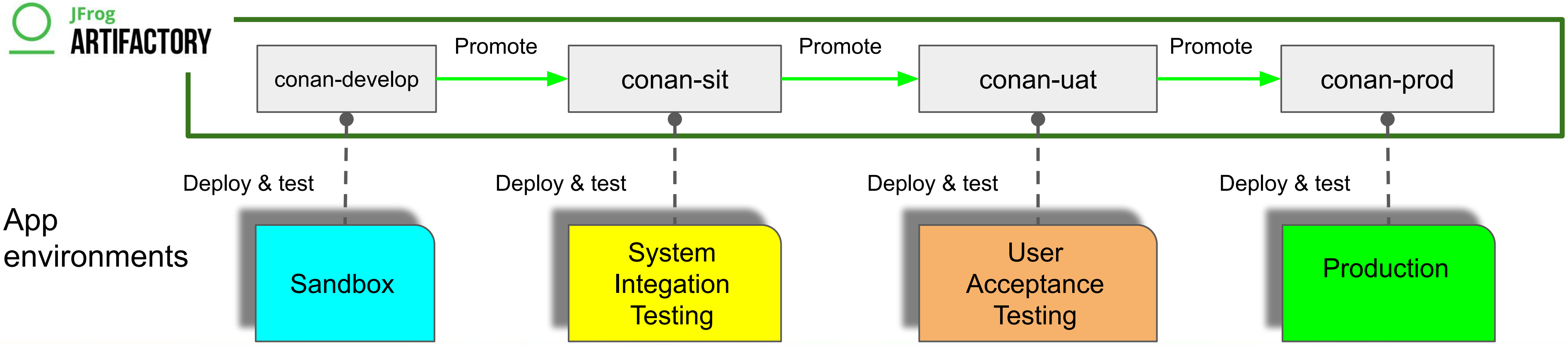
- An artifact in the “**Artifacts” section** is located if the following requirements are met :
  - Checksum/hash exists in the Artifactory DB
  - Build properties set on the artifacts
- An artifact in the **“Dependencies” section** is “located” if
  - its checksum/hash exists in the Artifactory DB
- No artifact upload = no Build properties assigned to the artifact
- See Appendix for Build Info limitation

Outline

- Introduction
- Conan reminder
- CI
- Build info in Artifactory
- **Promotion in Artifactory**
- Appendix

Promotion mechanism

- Monitor your binaries during the delivery process
- The component lifecycle is represented by a chain of repositories
- Consist in copying/moving a single or group of artifacts from a source repository to a target repository

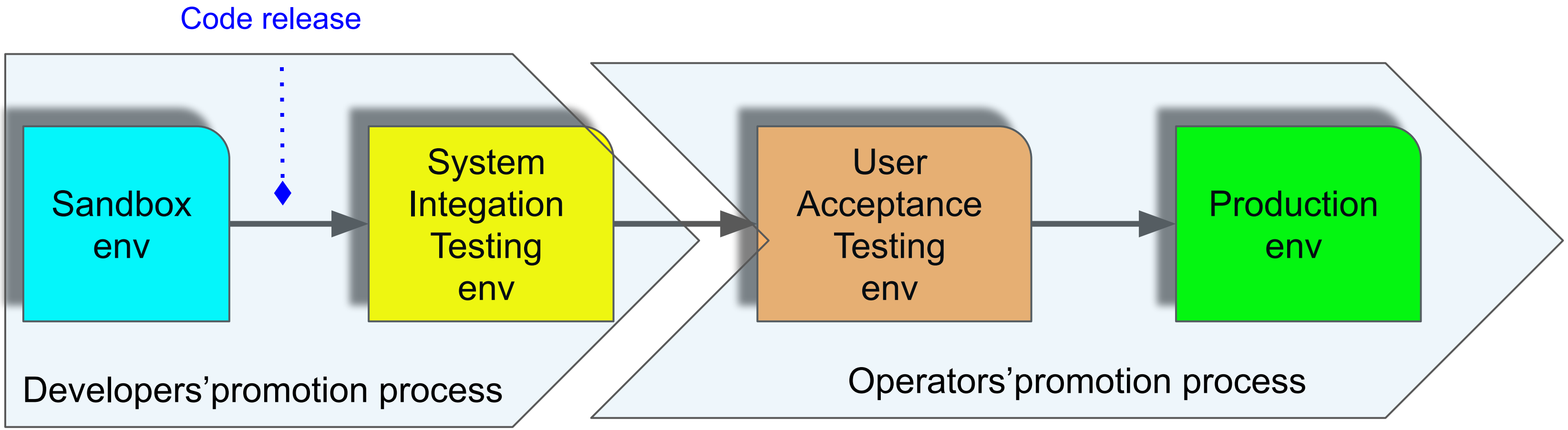


Promotion mechanism

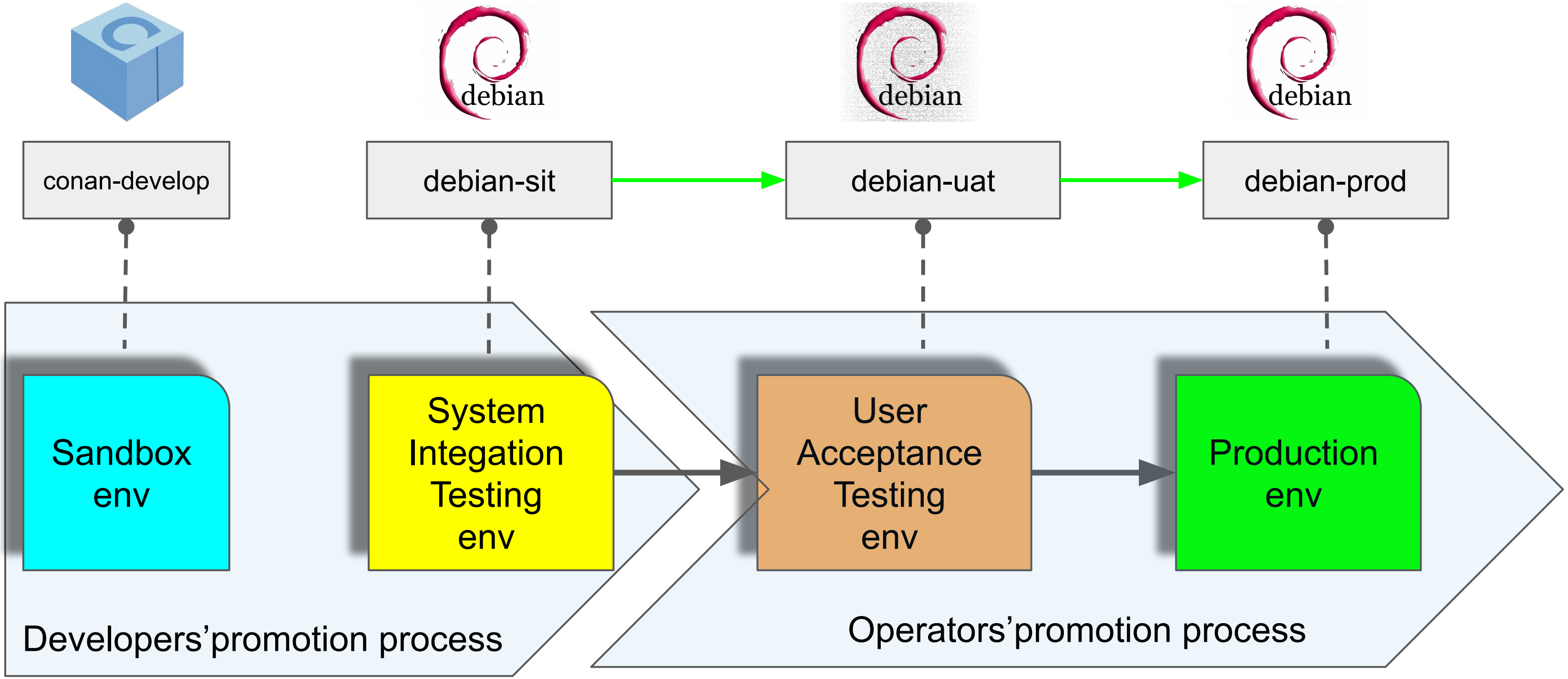
- Triggered automatically (CI/CD tool) or manually after passing a test in the delivery process
- 2 types of promotions
    - Artifact(s) promotion = copy or move 1 or more artifact
    - Build promotion = copy or move artifacts from a Build Info
        - Promotion status
        - Promote generated artifacts with or without build info dependencies

Promotion vs Release

- Artifactory doesn't generate releases, releasing is still handled by your build/release tools
- To deliver a product to production, there can be distinct promotion processes !


Code release
Sandbox env
System Integration Testing env
User Acceptance Testing env
Production env
Developers'promotion process
Operators'promotion process

Dev and Ops promotion process


conan-develop
debian
debian-sit
debian
debian-uat
debian
debian-prod
Sandbox env
System Integration Testing env
User Acceptance Testing env
Production env
Developers’promotion process
Operators’promotion process

Dev to Ops promotion process

- Goal :
  - Generate a **debian package** embedding App v1.0 (Release) for the ops and ease their promotion process with Build Info
- Implementation :
  - Retrieve App based on specific properties
  - Create a custom Build Info :
    - Artifact section : App debian package
    - Dependencies : lockfile + conan_package.tgz
  - Switching from the **conan_build_info** client to **the JFrog CLI**

Automation with JFrog CLI

- Lightweight tool running on the following OS : linux, windows, mac
- Optimized for massive actions : upload, download, search, update, move, copy, delete
- Checksum aware on uploads and downloads:
  - compute the checksum of the binary to upload and send it in the header request
  - Only upload binaries which checksum doesn’t exist in the Artifactory DB
- Easy way to manage Build Info

Lab11 - Configure the JFrog CLI

Goal:

- Connect the JFrog CLI to Artifactory

Task:

-

Success:

- Ping Artifactory + check read permission

Lab11 - Configure the JFrog CLI

```
cd promotion

jfrog rt c --interactive=false  --url=http://jfrog.local:8081/artifactory
--user=conan --password=conan2020 art7

# show current art7 profile
jfrog rt c show

# test connection by listing the repo content
jfrog rt search conan-metadata/
```

Lab12 - Download App based on properties

Goal:

- Use AQL (*) to retrieve a lockfile based on its properties (build.name, build.number, profile)
- Use the Conan Deploy Generator to deploy files locally

Task:

- Download a lockfile based on properties using AQL in a filespec
- Deploy conan_package.tgz in the current path

Success:

- Conan_package.tgz is downloaded and its content is exploded in App folder

* Artifactory Query Language : see Appendix for more details

Lab12 - Download App based on properties

3:00

```
# show filespec based on AQL
cat automation/filespec.json


# download lockfile based on properties + output “success”
jfrog rt download --spec=automation/filespec.json

# “deploy” the package referenced in the lockfile in the current path
conan install App/1.0@mycompany/stable --lockfile app_release.lock -g deploy -r conan-develop

ls -l App/

# execute the deployed App
./App/bin/App
```

Lab13 - Create and upload a debian package

Goal:

- Create and upload a debian package

Task:

- Create a debian package from the App binary
- Upload the debian package to Artifactory

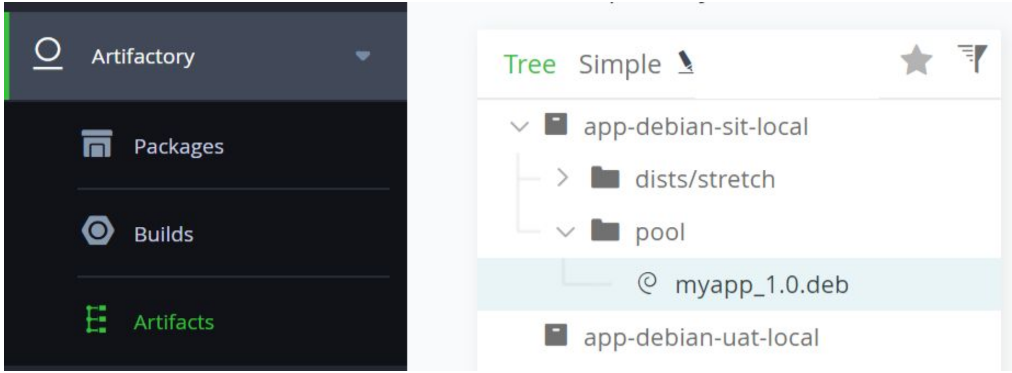
Success:

- Check the Debian package in Artifactory

Lab13 - Create and upload a debian package

```
./generateDebianPkg.sh conan conan2020
```

2:00



Lab 14 - Create a custom Build info

Goal:

- Create a Build Info using the **JFrog CLI** which can then be promoted by the ops team

Task:

- Create and publish a **custom build info** :
  - Artifact section : debian package
  - Dependencies section : app_release.lock + conan_package.tgz
- Publish the Build Info

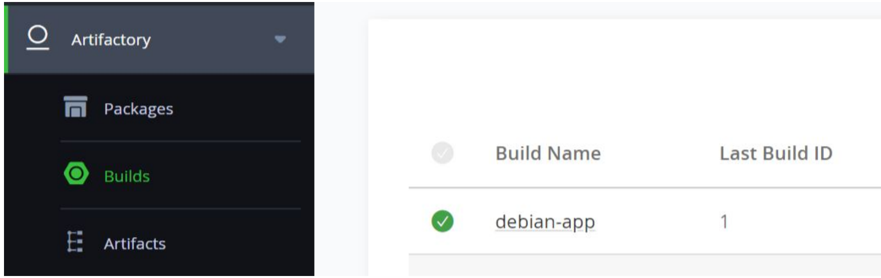
Success:

- Check the Build Info in Artifactory

Lab 14 - Create a custom Build info

```
# define “artifact section” in build info
# won’t be reuploaded as the JFrog CLI is checksum aware => output “status”:“sucess”
jfrog rt u debian_gen/myapp_1.0.deb app-debian-sit-local/pool/ --build-name=debian-app
--build-number=1

# define “dependency section” in build info => output “status”:“sucess”
jfrog rt bad debian-app 1 app_release.lock
jfrog rt bad debian-app 1 App/conan_package.tgz

# publish build info => check result in Artifactory in the build section
jfrog rt bp debian-app 1
```

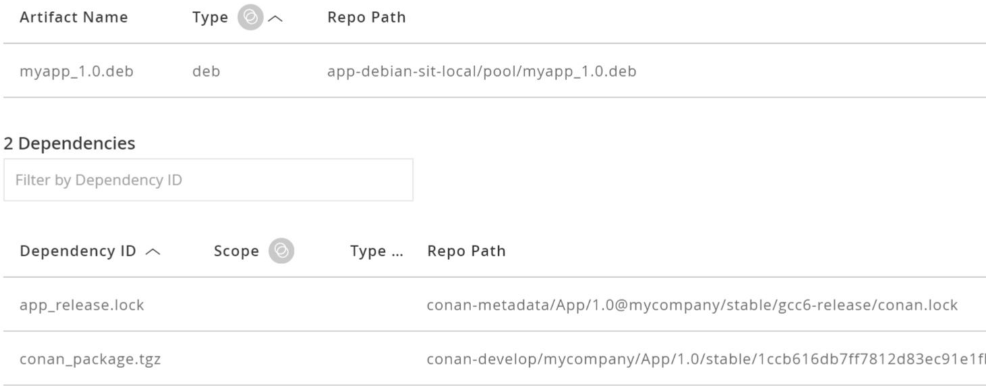
Lab 14 - Create a custom Build info



✓	Build Name	Last Build ID
✓	debian-app	1

Go to “Build” section and select debian-app

Check the Build Info content



Artifact Name	Type	Repo Path
myapp_1.0.deb	deb	app-debian-sit-local/pool/myapp_1.0.deb

2 Dependencies

Dependency ID	Scope	Type ...	Repo Path
app_release.lock			conan-metadata/App/1.0@mycompany/stable/gcc6-release/conan.lock
conan_package.tgz			conan-develop/mycompany/App/1.0/stable/1ccb616db7ff7812d83ec91e1f

Lab 15 - Build Info Promotion

Goal:

- Promote Build Info by move without dependencies using the **JFrog CLI**

Task:

- Use bpr (build promote) instruction

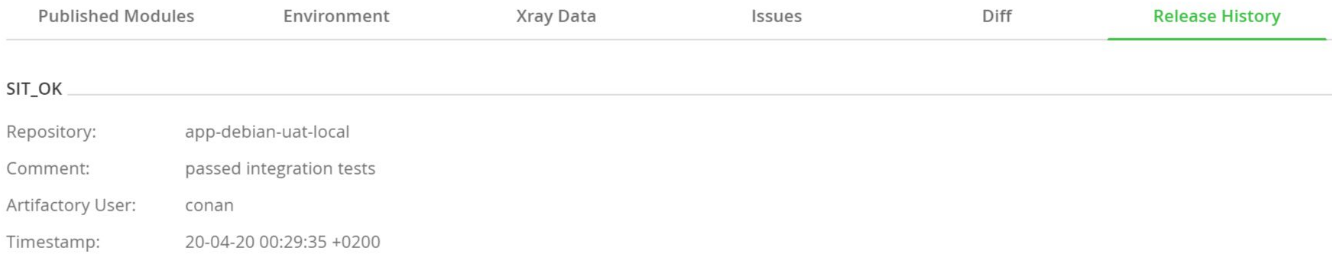
Success:

- See the Build Info Promotion in Artifactory
  - Check path in “published modules” tab
  - Check “Release history” tab

Lab15 - Build Info Promotion

```
jfrog rt bpr debian-app 1 app-debian-uat-local --status="SIT_OK"
--comment="passed integration tests" --include-dependencies=false --copy=false
```

Check the Build Info content and Release History tab



SIT_OK

Repository:	app-debian-uat-local
Comment:	passed integration tests
Artifactory User:	conan
Timestamp:	20-04-20 00:29:35 +0200

Promotion - Good to know

- When promoting by copy :
  - This will create more artifacts (not binaries)
  - Any AQL and filespec have to target a repository name
- Build Info promotion with / without dependencies
  - Depends on your project structure and delivery process
- Limitation : A unique target repository

Lab 16 - Trigger a build of libC in Jenkins

Goal:

- Test the whole pipeline in Jenkins

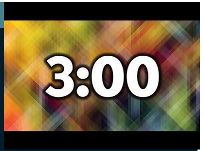
Task:

- Enter to Jenkins container where the source repositories are
- Make some changes in libC in branch develop
- Commit the changes
- Trigger pipeline in Jenkins for libC

Success:

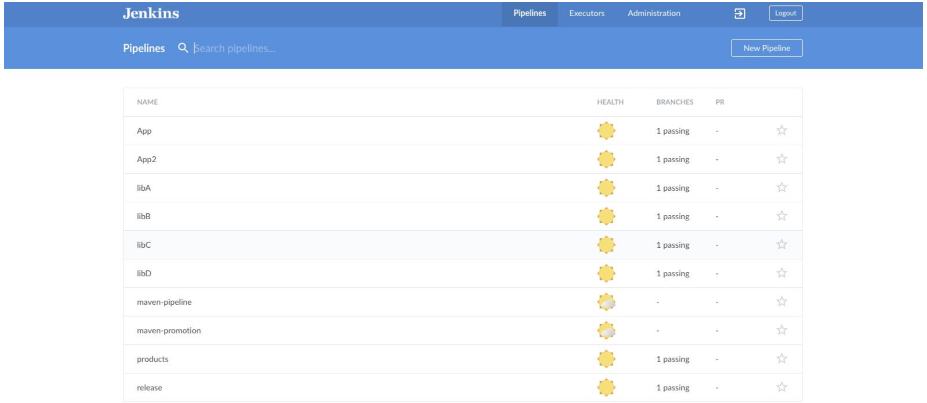
- Pipeline generates new versions of libD, App, App2 and uploads to conan-develop

Lab 16.a - Trigger a build of libC in Jenkins

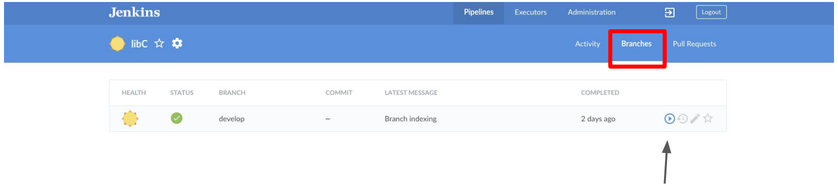


```
docker exec -it jenkins /bin/bash

cd /var/lib/jenkins/libC/

echo "#adding some stuff" >> conanfile.py

git commit -a -m "add stuff"

# now trigger the job for libC in Jenkins
```

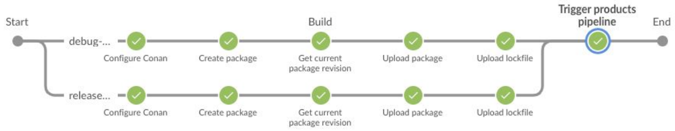
Lab 16.b - Trigger a build of libC in Jenkins

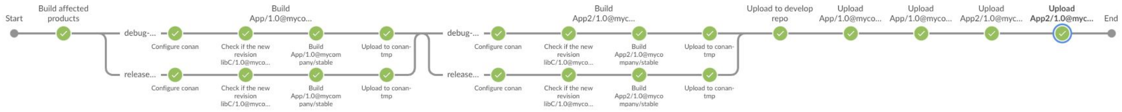


NAME	HEALTH	BRANCHES	PR
App	☀	1 passing	-
App2	☀	1 passing	-
libA	☀	1 passing	-
libB	☀	1 passing	-
libC	☀	1 passing	-
libD	☀	1 passing	-
maven-pipeline	⛅	-	-
maven-promotion	⛅	-	-
products	☀	1 passing	-
release	☀	1 passing	-

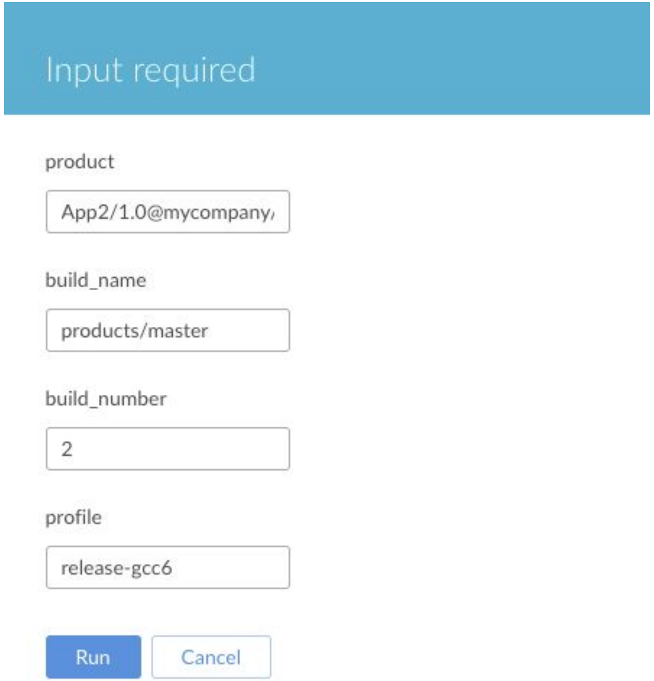


HEALTH	STATUS	BRANCH	COMMIT	LATEST MESSAGE	COMPLETED
☀	✓	develop	–	Branch indexing	2 days ago





Lab 16.b - Trigger a build of libC in Jenkins



Input required

product

App2/1.0@mycompany/

build_name

products/master

build_number

2

profile

release-gcc6

Lab16... Homework

Have a look at the different Jenkinsfiles:

- Package pipeline: https://github.com/conan-ci-cd-training/libC/blob/develop/Jenkinsfile
- Products pipeline: https://github.com/conan-ci-cd-training/products/blob/master/Jenkinsfile
- Promotion process: https://github.com/conan-ci-cd-training/release/blob/master/Jenkinsfile

Summary

- Use at least two Artifactory repos, tmp repo as an exchange repo and develop to have the binaries that other developers will consume.
- Use revisions with a package id mode that takes them into account to do “automatic versioning” and integrate your changes quickly
- Use lockfiles
  - For reproducibility: calculate the build order of a graph with fixed revisions
  - To generate build info for Artifactory
  - Store them in Artifactory and retrieve them later to deploy
- Always use config install to have the same configuration in all Conan clients

Resources

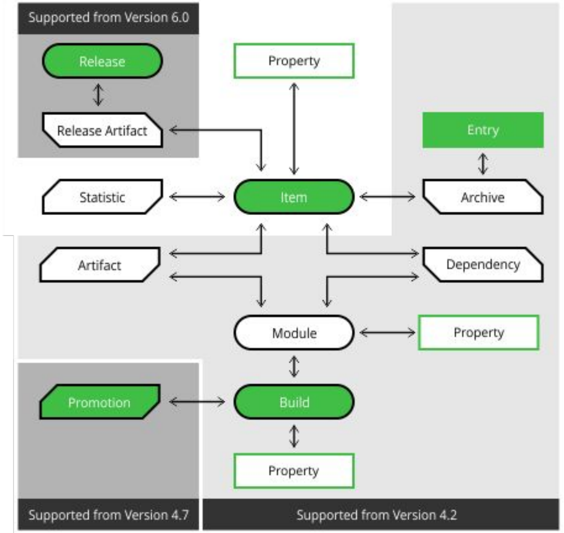
- Docs: https://docs.conan.io/
  - Read carefully, explore.
- Issues:
  - CppLang slack (community)
  - Github issues (https://github.com/conan-io/conan) “official” support
- Following trainings:
  - conandays@jfrog.com
- Other Conan questions?
  - info@conan.io

Outline

- Introduction
- Conan reminder
- CI
- Build info in Artifactory
- Promotion in Artifactory
- **Appendix**

Automation with AQL

- Artifactory Query Language ~ SQL for Artifactory
- JSON formatted requests and responses
- String, Date, Time operators
- Sorting, limiting results
- Non admin can only use item domain



List artifact of a Build Info

build_info_artifacts.json

```
builds.find({
    "name": "app1",
    "number": "2",
}).include("module.artifact.item.name", "module.artifact.item.path")
```

```
# with creds or access token
curl -uadmin:<PASS> -XPOST -T build_info_artifacts.json
http//jfrog.local:8081/artifactory/api/search/aql
```

List dependencies filtered on property

build_info_deps.json

```
builds.find({
    "name": "app1",
    "number": "2",
    "module.artifact.dependency.@conan.settings.os" : "Linux"
}).include("module.dependency.item.name", "conan.settings.build_type",
"module.dependency.item.path")
```

```
# with creds or access token
curl -uadmin:<PASS> -XPOST -T build_info_deps.json
http//jfrog.local:8081/artifactory/api/search/aql
```

List artifacts based on a property value

artifact_search.json

```
items.find({
    "repo": "conan-develop",
    "name": "conaninfo.txt",
    "$or": [
         { "@conan.settings.os": "Linux" }, { "@conan.settings.os": "Windows" }
     ]
}).include("repo","path","name","@conan.settings.os","@conan.settings.arch","@conan.settings.build_type")
```

```
# with creds or access token
curl -uconan:conan2020 -XPOST -T artifact_search.json
http//jfrog.local:8081/artifactory/api/search/aql
```

Download a file using the CLI and filespec with AQL

automation/filespec.json

```
{
  "files": [{
    "aql": {
      "items.find": {
        "repo": "conan-metadata",
        "name" : { "$match" : "*.lock"},
        "$and": [
          { "@build.name": "conan-app" }, { "@build.number": "1" }
        ]}
  }}]
```

```
# JFrog CLI should have be configured before
jfrog rt download --spec=automation/filespec.json
```

Build Info - Warning

- MAY NOT fit the use case when :
  - An artifact is referenced by multiple Build Info (like unchanged recipe)
  - An artifact is NOT considered as a Build Info dependency

- Possible workaround :
  - All the files from the Artifact section should be packaged into an archive which will be the result of your Build Info

Outline

- Introduction
- Conan reminder
- CI
- Build info in Artifactory
- Promotion in Artifactory
- Appendix