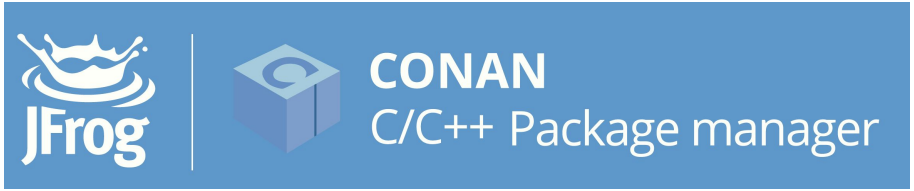
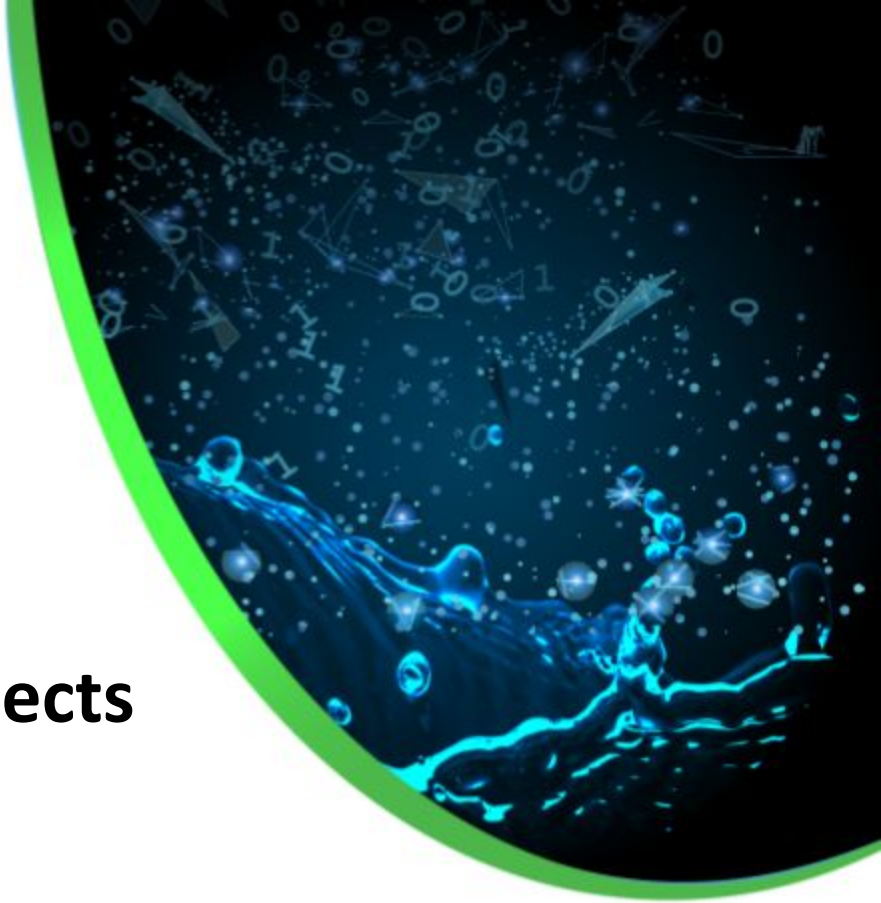
Welcome!

PREREQUISITES

- Wi-Fi enabled Mac or PC
- SSH client
- Internet Browser

1. Register and launch https://jfrog.orbitera.com/c2m/trial/1289
2. ssh conan@<IP>





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

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

Speakers

Yann Chaysinh

Carlos Zoido

Lab 0 - Setup

```
# https://jfrog.orbitera.com/c2m/trial/1289

$ ssh conan@<orbitera-IP>
# Use password from orbitera
$ git clone
https://github.com/conan-ci-cd-training/conan_ci_cd.git
```

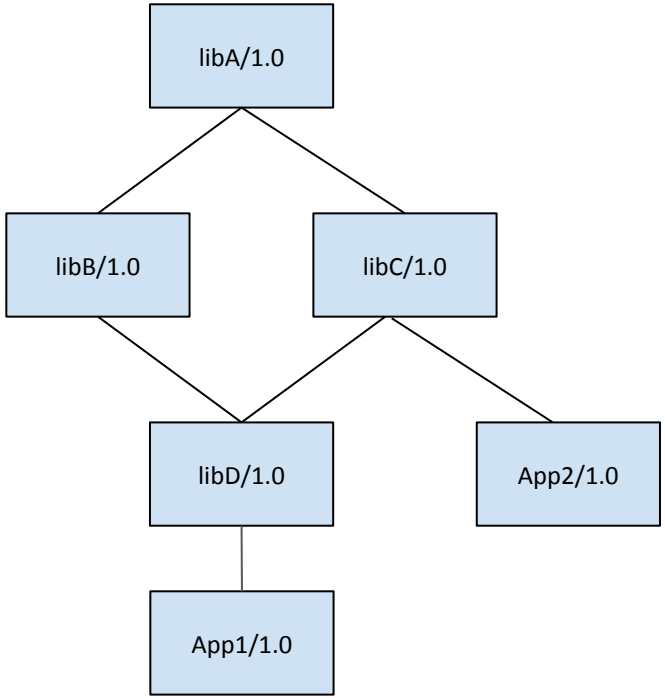
```
vm-testdriveinstance-1289-88142
-------------------


------ 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

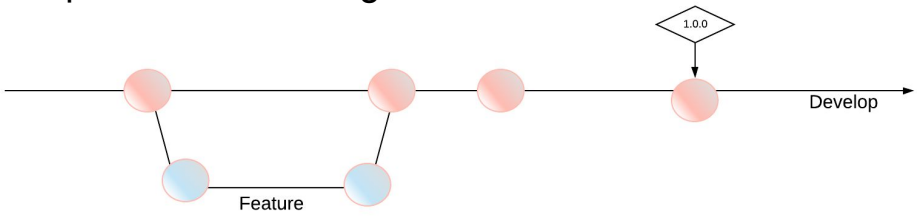
The Story: Mycompany components

- 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
- All librairies are static



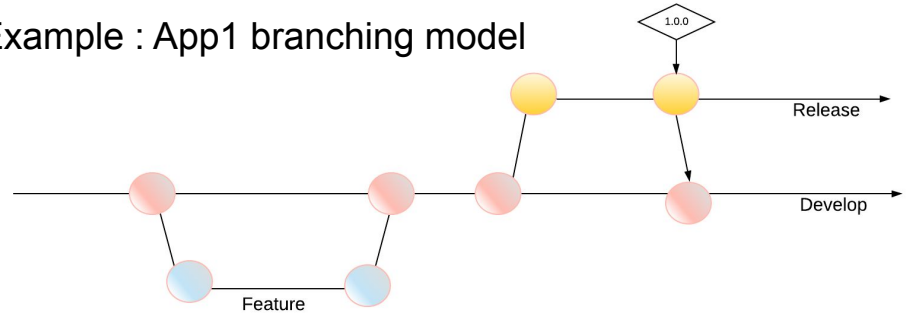
The Story: Code workflow

Example : libA branching model

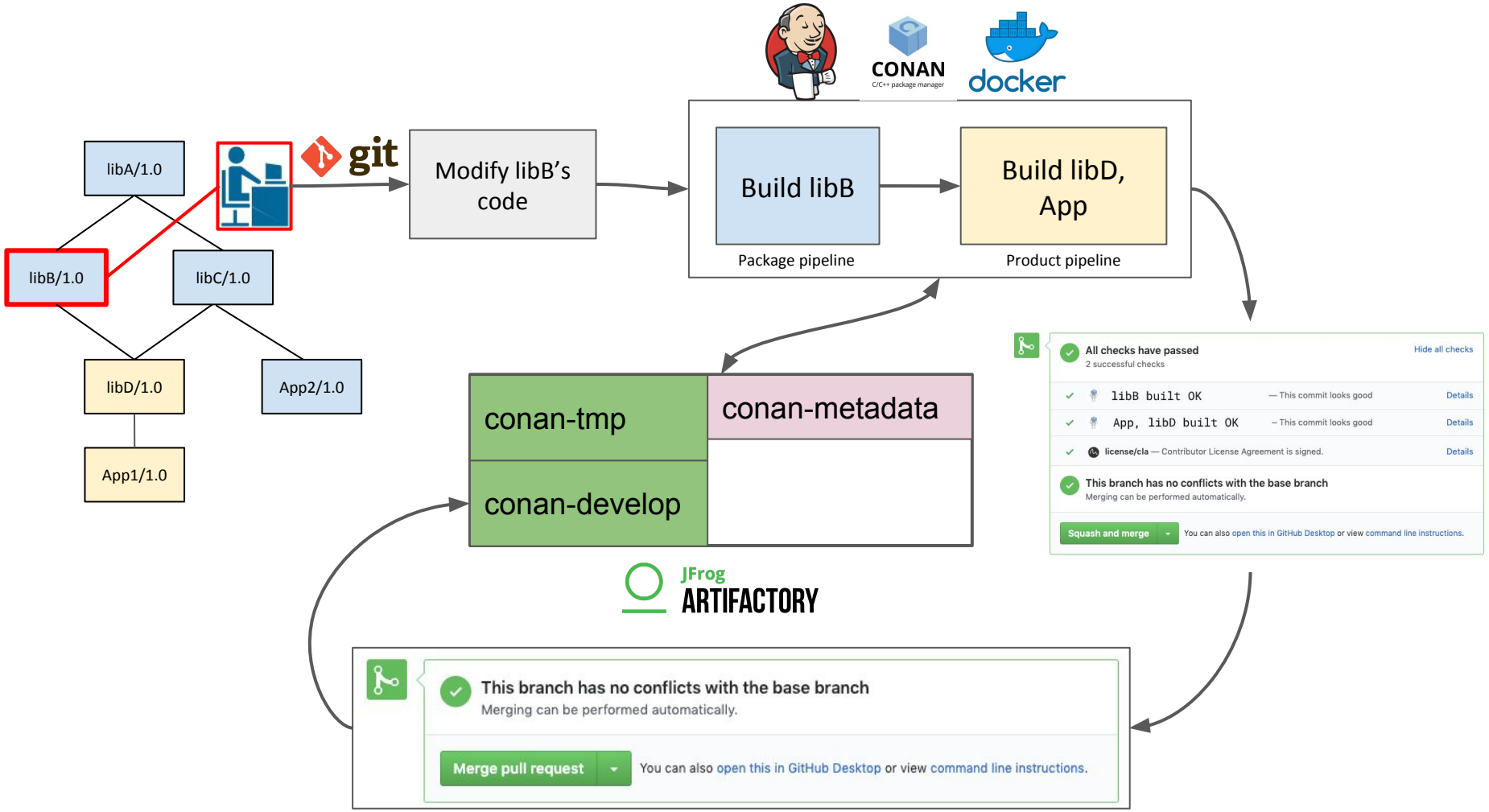


* libB, libC and libD follow the same flow and have their own code repository

Example : App1 branching model



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


libA/1.0
libB/1.0
libC/1.0
libD/1.0
App2/1.0
App1/1.0
git
Modify libB's code
CONAN
C/C++ package manager
docker
Build libB
Build libD, App
Package pipeline
Product pipeline
conan-tmp
conan-metadata
conan-develop
JFrog
ARTIFACTORY
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

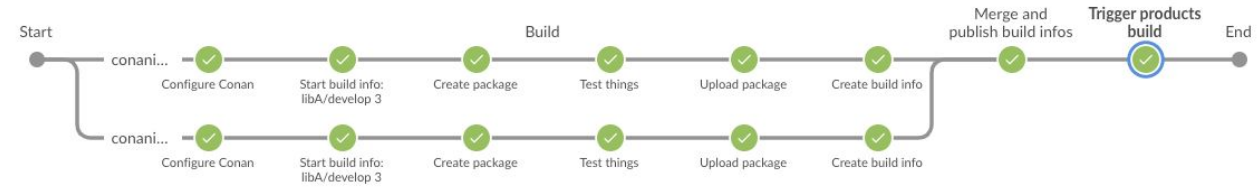
Lab 1: Jenkins and environment bootstrapping

```
$ cd conan_ci_cd/setup_jenkins && chmod +x bootstrap.sh
$ ./bootstrap.sh <artifactory_password> <jenkins_credential>
```

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

Check Jenkins

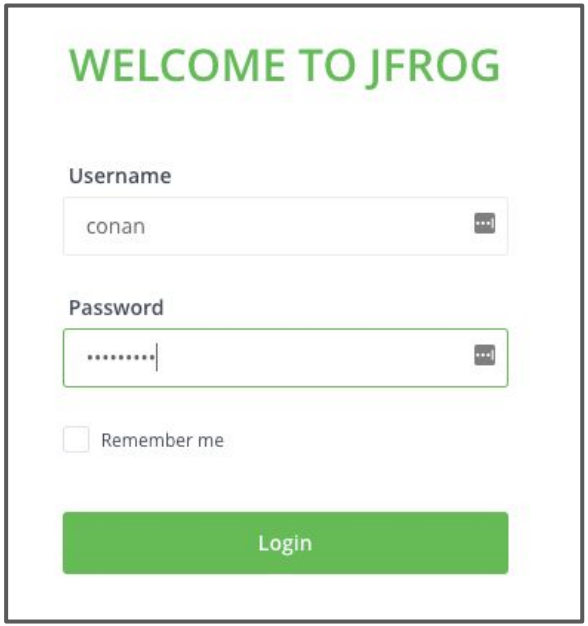


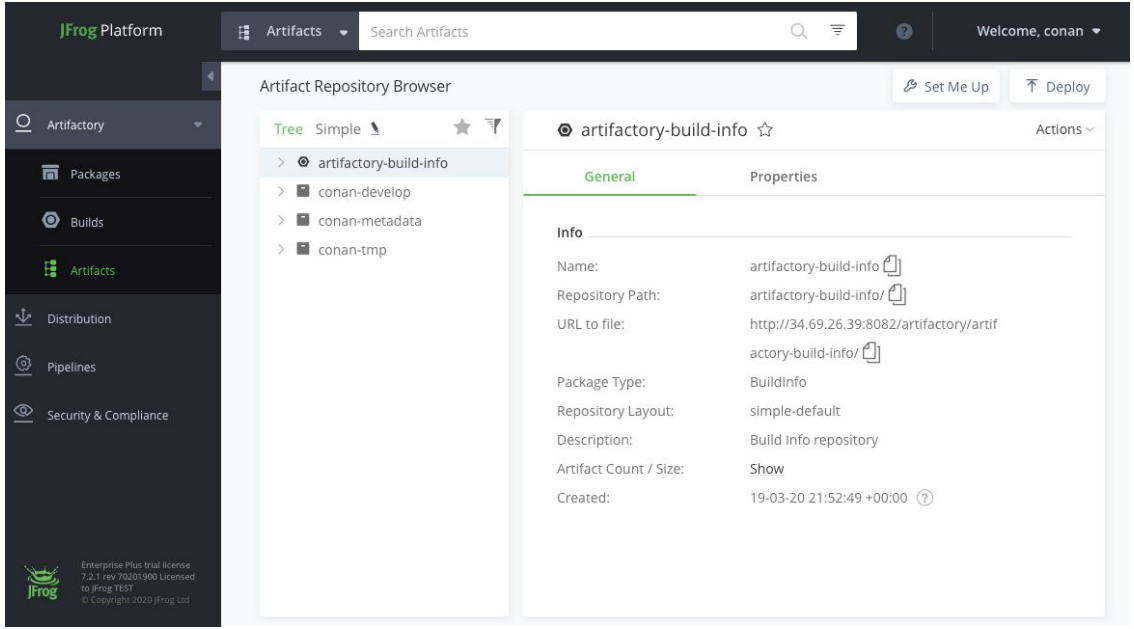
Username: **administrator**
Password: **<Jenkins Credential>**
in orbitera e-mail with JFrog Test Drive Details

Artifactory

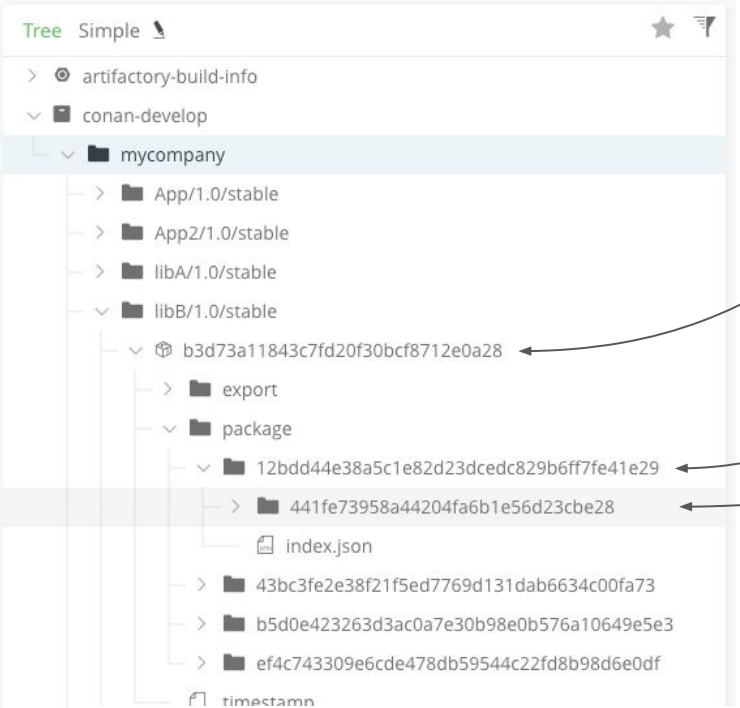
- Universal Binary repository manager
- Checksum based storage
- Build Info : Binary dependency tracker
- Properties : metadata applied to any artifacts. Could be used for automation (search, download, move, delete)

Check Artifactory





Check Artifactory



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)**, <u>new revisions will affect package id's of dependents</u>**

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/libA.git",
           "revision": "auto"}
```

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

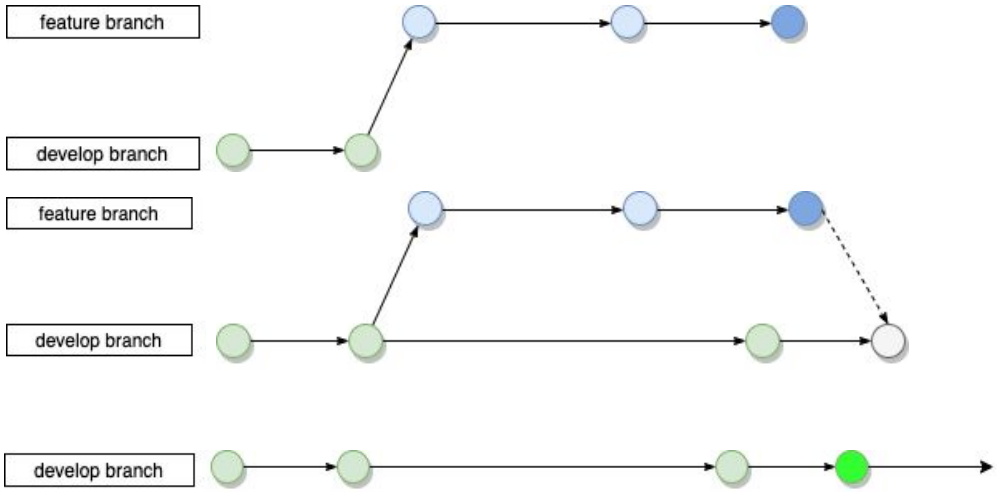
```
$ conan config install https://github.com/conan-ci-cd-training/settings.git
```

Outline

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

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

Phase 1: Developer works on a feature branch of libB

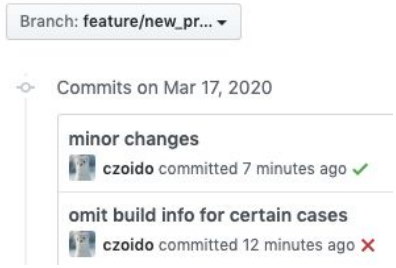
Developer is commiting changes for libB.
Will the lib build OK and pass the tests for all configurations?

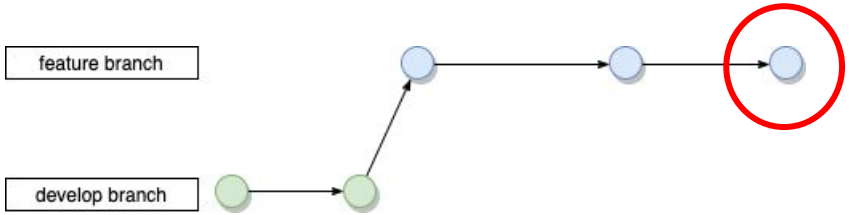
Package (libB) pipeline

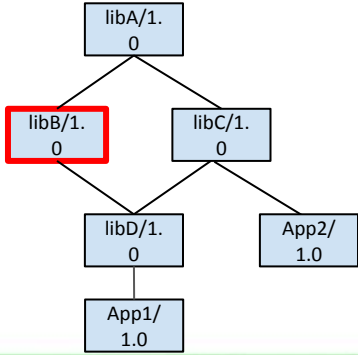
Build, test and create the package for the modified code

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

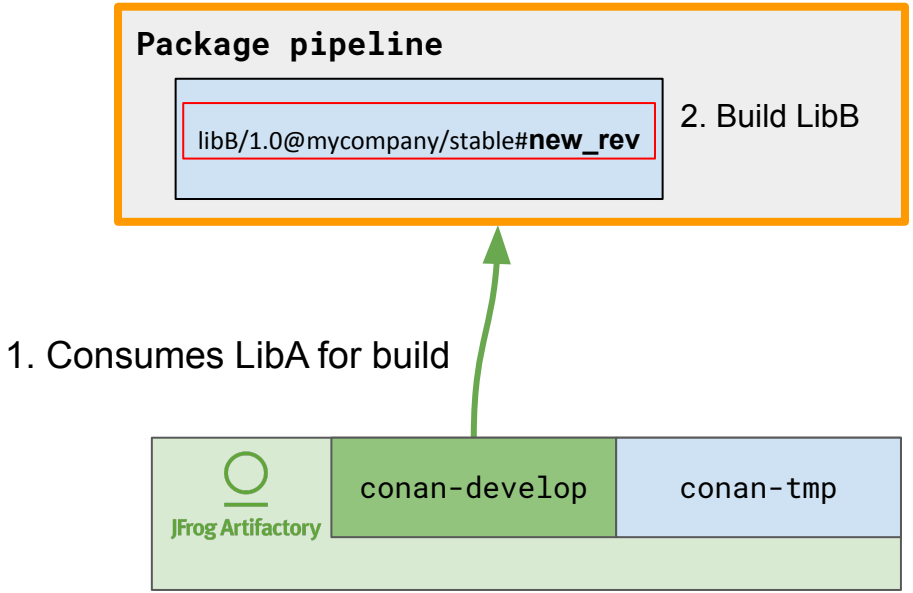
Notify libB







Phase 1: Developer works on a feature branch of libB



Lab 2 - Create the library in the CI using lockfiles

Goal:

- Check that the changes in the developer's feature branch build for all configurations

Task:

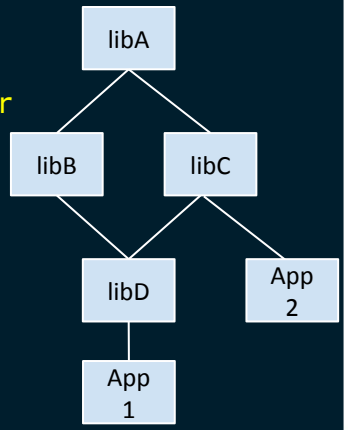
- Clone git repo and checkout the feature branch
- Calculate the graph for the library with all the latest requirements from conan-develop
- Build the library for different profiles using the lockfiles

Success:

- Lib builds OK

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
```



Outline

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

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

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

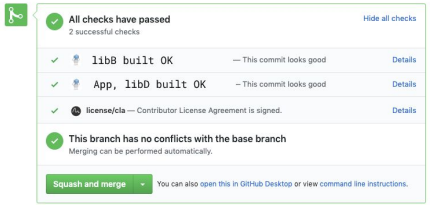
New rrev of libB

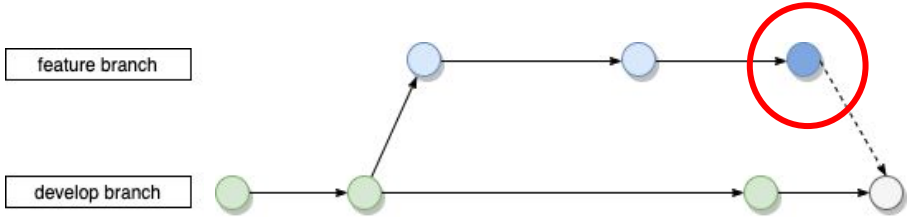
Product (App) pipeline

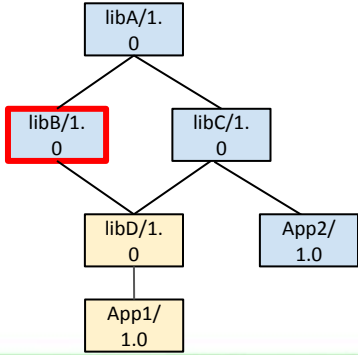
Build, test and create the affected product (App) for **the new revision of libB**. New binaries for **libD** will also be generated

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

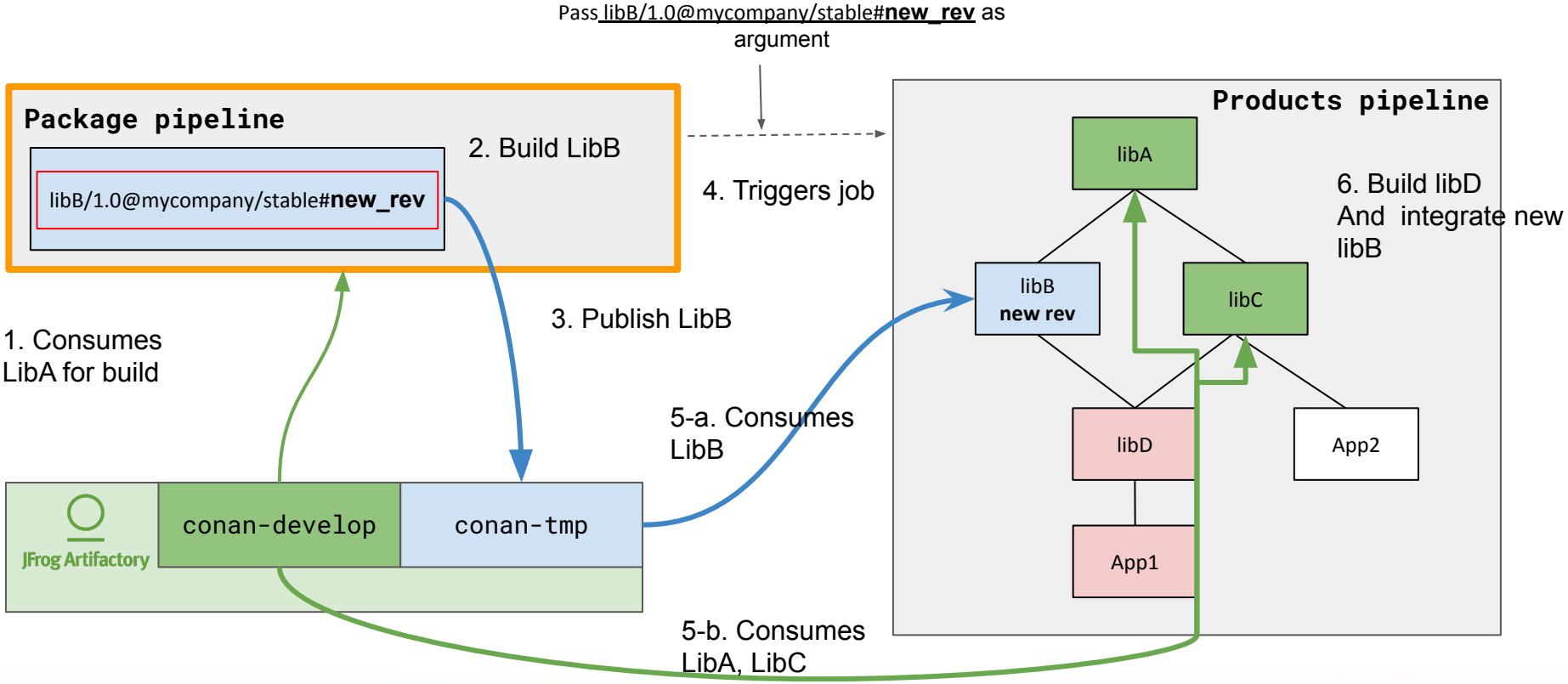
Notify:
libB ✓
libD ✓
App ✓







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:

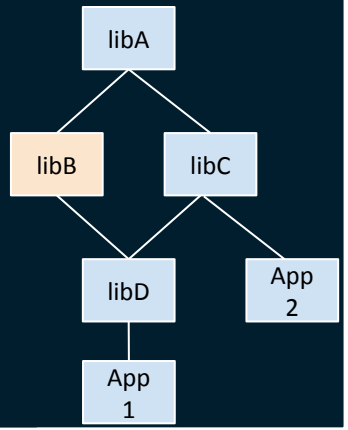
- Get the name of the recipe
- Get the version of the recipe
- Get the revision we have just created

Success:

- Getting the complete reference for the new libB

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 <name>/<version>@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:

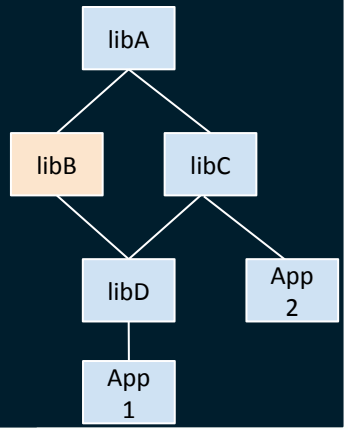
- Upload libB/1.0@mycompany/stable#new_rev to conan-tmp

Success:

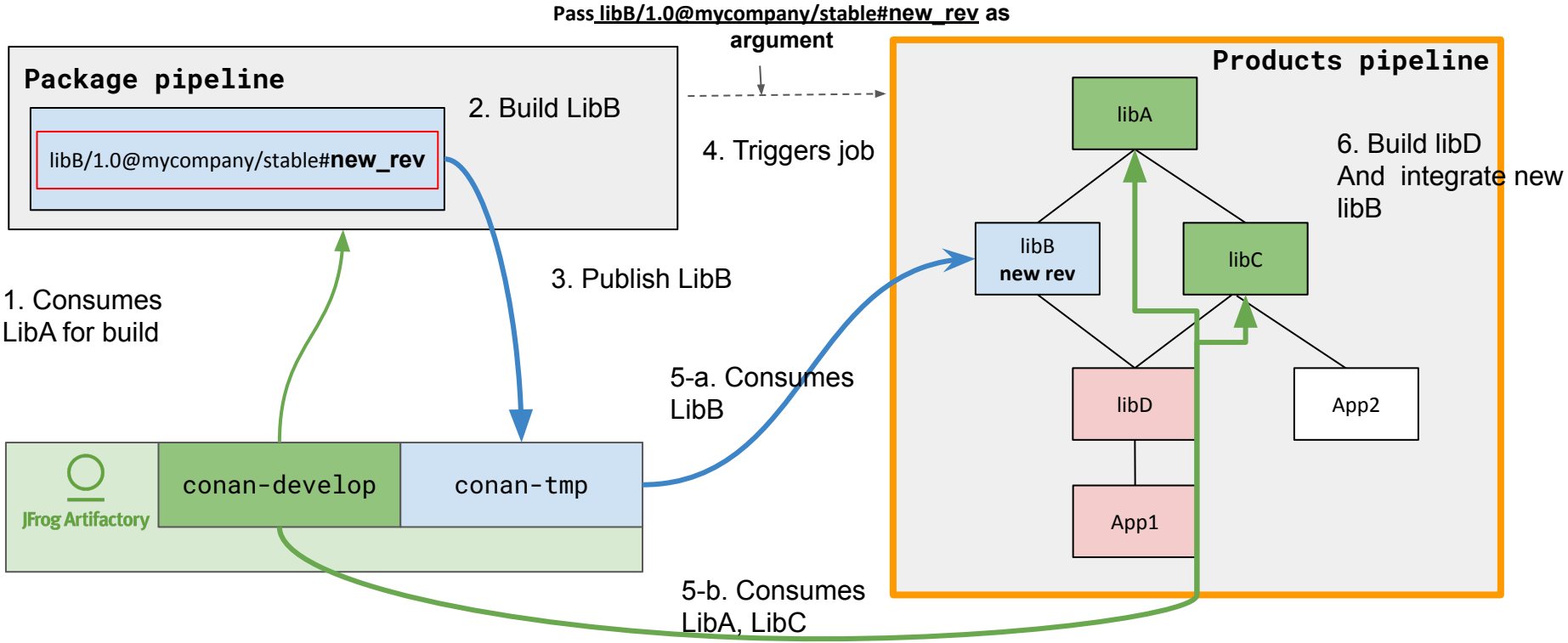
- libB is uploaded

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  --force

# now we are ready to launch the product pipeline
```



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



Lab 5 - Check affected products

Goal:

- Getting the list of affected products

Task:

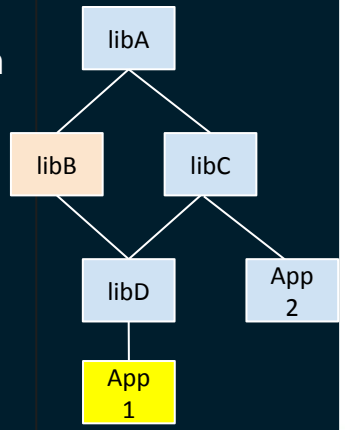
- Calculate App and App2 product's graph
- Check if libB/1.0@mycompany/stable is a node in the graph of any of them

Success:

- Check that libB/1.0@mycompany/stable is in the App/1.0@mycompany/stable graph but not in the App2

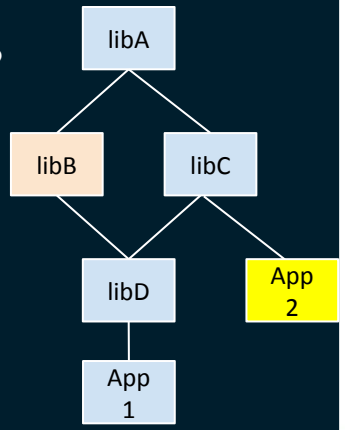
Lab 5.a: Check if App is affected

```
# we are in a new pipeline/job to simulate that, clear the cache
$ conan remove "*" -f

$ conan graph lock App/1.0@mycompany/stable
--profile=release-gcc6 --lockfile=app_graph.lock
-r=conan-develop

$ conan graph build-order app_graph.lock --json=build_order.json
--build

$ grep -q "libB/1.0@mycompany/stable" app_graph.lock && echo $?
```



Lab 5.b: Check if App2 is affected

```
$ conan graph lock App2/1.0@mycompany/stable
--profile=release-gcc6 --lockfile=app2_graph.lock
-r=conan-develop

$ conan graph build-order app2_graph.lock
--json=build_order.json --build

$ grep -q "libB/1.0@mycompany/stable" app2_graph.lock && echo $?
```



Lab 6 - Inject libB’s new revision in App’s graph

Goal:

- Create a lockfile with the latest versions of App and its dependencies and the new revision of libB

Task:

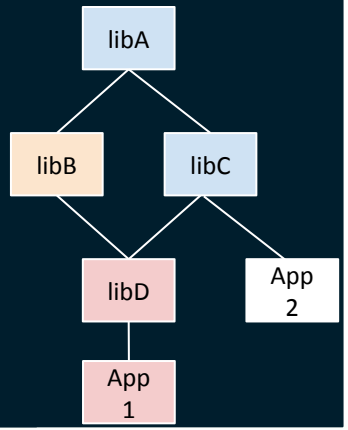
- Install App for the profile getting deps from conan-develop (latest)
- Install the new revision of libB from conan-tmp with --update
- Recalculate the graph with the contents of the cache

Success:

- See the lockfile with the new revision of libB and the rest of latest libraries

Lab 6 - Inject libB's new revision in App's graph

```
# 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

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

$ cat app_release.lock
```



Lab 7 - Build the graph using the lockfile

Goal:

- Build App with the new revision of libB → build libs affected by this revision → libD

Task:

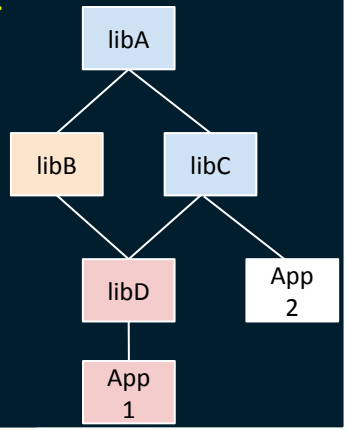
- Calculate the build order using the lockfile
- Build libD → update lockfile → recalculate build order
- Build App → update lockfile → recalculate build order

Success:

-

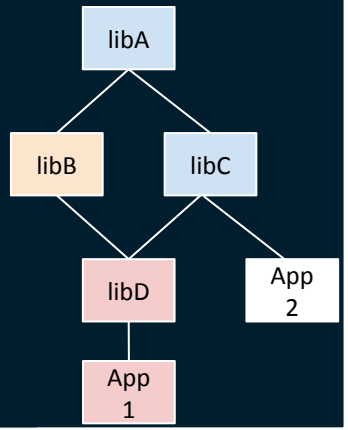
Lab 7.a - Build the graph using the lockfile

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

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

$ conan install libD/1.0@mycompany/stable#<rev_from_build_order>
--build libD --lockfile conan.lock

# lockfileD is updated with the node libD marked as built

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



Lab 7.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#<rev_from_build_order>
--build App --lockfile conan.lock

$ conan graph update-lock app_release.lock conan.lock

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



Outline

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

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

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

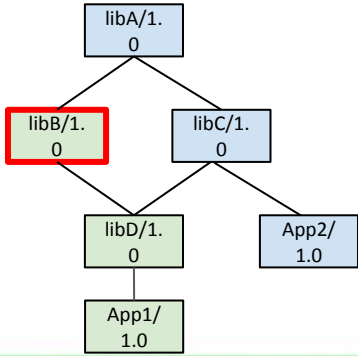
New rrev of libB

Product (App) pipeline

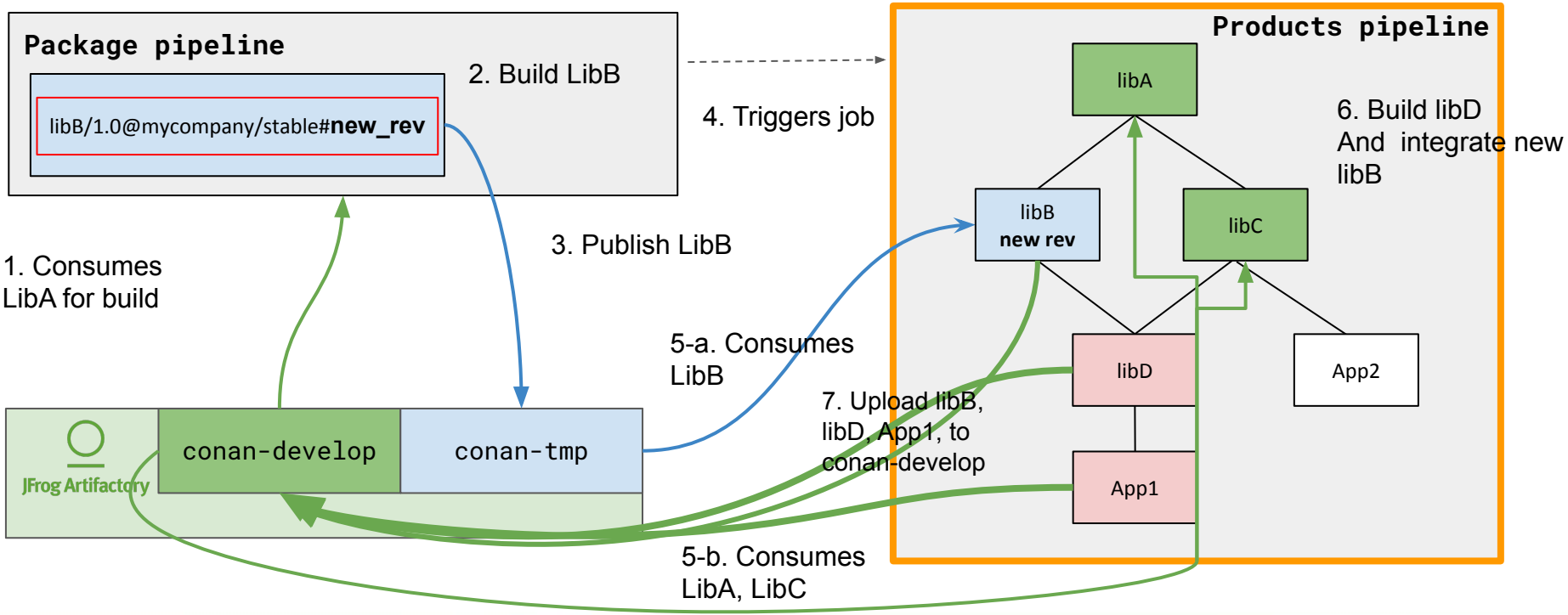
Build, test and create the affected product (App) for **the new revision of libB**. New binaries for **libD** will also be generated

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

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



Phase 3: The developer opens a PR with a feature for libB



Lab 8 - 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:

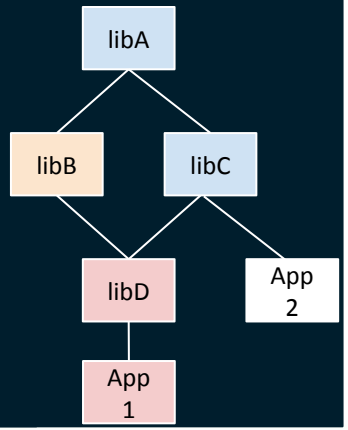
- Upload new revision of libB and new binaries of libD and App to conan-develop

Success:

- All the new binaries are uploaded

Lab 8 - Upload new packages to conan-develop

```
$ conan upload libD/1.0@mycompany/stable -r conan-tmp --confirm
--force --all
$ conan upload App/1.0@mycompany/stable -r conan-tmp --confirm
--force --all
```



Lab 9 - Upload lockfile to conan-metadata

Goal:

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

Task:

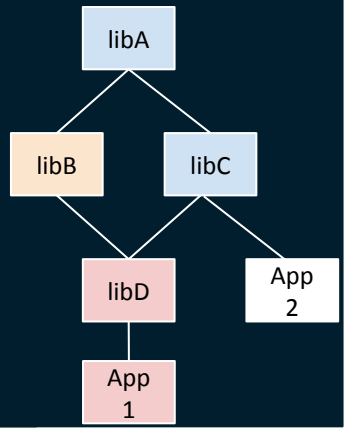
- Use the Artifactory API to upload the file to conan-metadata (generic repo)

Success:

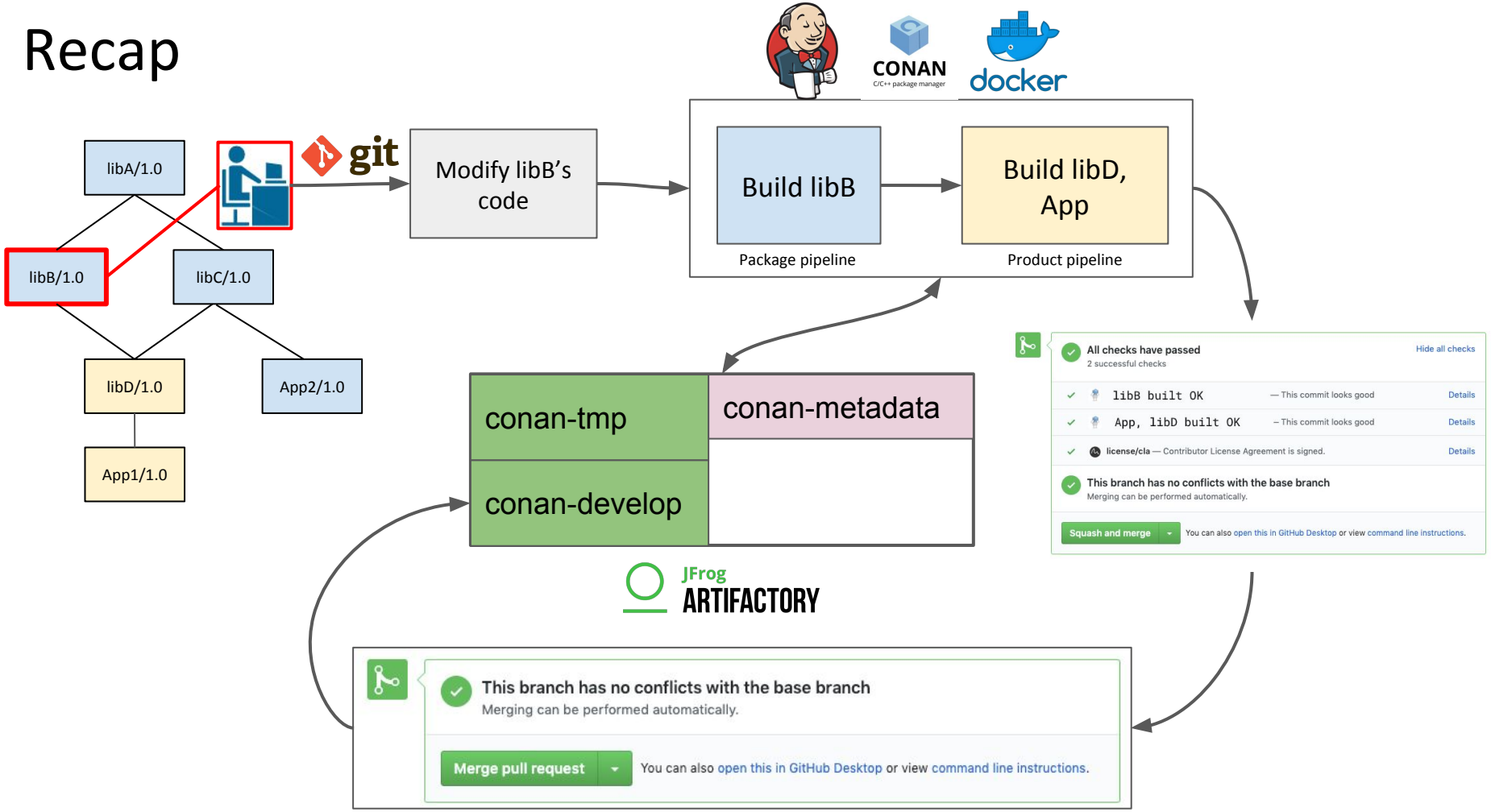
- Check that the file has been uploaded to conan-metadata

Lab 9 - Upload lockfile to conan-metadata

```
$ curl --user "conan:conan2020" --header "Content-Type:
application/json"
http://jfrog.local:8081/artifactory/conan-metadata/App/1.0@mycom
pany/stable/conanio-release/conan.lock --upload-file
app_release.lock
```



Recap


CONAN
C/C++ package manager
docker
libA/1.0
libB/1.0
libC/1.0
libD/1.0
App2/1.0
App1/1.0
git
Modify libB's code
Build libB
Package pipeline
Build libD, App
Product pipeline
conan-tmp
conan-metadata
conan-develop
JFrog
ARTIFACTORY
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.

Outline

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

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 > cppApp > 15

Build Name	Agent	Build Agent	Started	Duration	Principal	Artifactory Principal
cppApp		Conan Client/1.X	12-03-20 13:24:58 +0100	0.0 seconds	-	admin

Published Modules | Environment | Xray Data | Issues | Diff | Release History

2 Modules

Filter by Module ID

Export folder (recipe, manifest, …)

Module ID	Number Of Artifacts	Number Of Dependencies
App/1.0@mycompany/stable	3	6
App/1.0@mycompany/stable:f2da1f970f1b2ff76ac05575fda7050c7c808486	3	9

Package folder (binaries)

Build Info - Intro

Published Modules | Environment | Xray Data | Issues | Diff | Release History

< Back To All Modules

Module Details: App/1.0@mycompany/stable

Compare With Previous Build

3 Artifacts

Filter by Artifact Name

Artifact Name	Type	Repo Path
conanfile.py		release-bundles/myApp/1.0.0-15/winter-conan-prod-local/mycompany/App/1.0/stable/93ea609c432a13ce2740e01f9e055ec6/export/conanfile.py
conanmanifest.txt		release-bundles/myApp/1.0.0-15/winter-conan-prod-local/mycompany/App/1.0/stable/93ea609c432a13ce2740e01f9e055ec6/export/conanmanifest.txt
conan_sources.tgz		release-bundles/myApp/1.0.0-15/winter-conan-prod-local/mycompany/App/1.0/stable/93ea609c432a13ce2740e01f9e055ec6/export/conan_sources.tgz

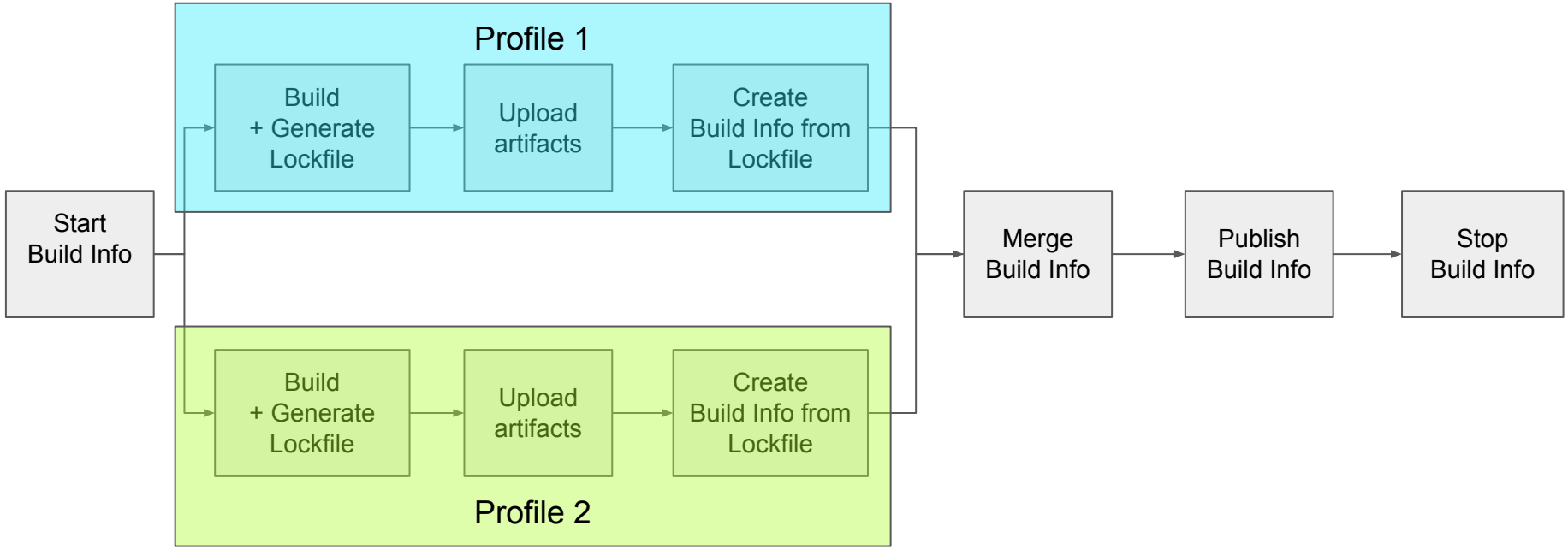
6 Dependencies

Filter by Dependency ID

Dependency ID	Sc...	Type	Repo Path
libA/1.0@mycompany/stable :: conanfile.py			winter-conan-dev-local/mycompany/libA/1.0/stable/cbb5d19d6e4cdc8f995148850d9eec5a/export/conanfile.py
libA/1.0@mycompany/stable :: conanmanifest.txt			winter-conan-dev-local/mycompany/libA/1.0/stable/cbb5d19d6e4cdc8f995148850d9eec5a/export/conanmanifest.txt
libB/1.0@mycompany/stable :: conanfile.py			winter-conan-dev-local/mycompany/libB/1.0/stable/f0b66841a87fa6610ac05890f3324184/export/conanfile.py
libB/1.0@mycompany/stable :: conanmanifest.txt			winter-conan-dev-local/mycompany/libB/1.0/stable/f0b66841a87fa6610ac05890f3324184/export/conanmanifest.txt
libC/1.0@mycompany/stable :: conanfile.py			winter-conan-dev-local/mycompany/libC/1.0/stable/396dc0c3968b2d8695a3f0efb31dafc9/export/conanfile.py
libC/1.0@mycompany/stable :: conanmanifest.txt			winter-conan-dev-local/mycompany/libC/1.0/stable/396dc0c3968b2d8695a3f0efb31dafc9/export/conanmanifest.txt

All artifacts have to be in Artifactory !

Lab10 - Generate, merge and publish a Build Info



Lab10 - Generate, merge and publish a Build Info

Goal:

- Generate and merge 2 Build Info

Task:

- Create 1 lockfile per profile (Debug and Release)
- Using the conan_build_info client:
  - Generate a Build Info (JSON file) per lockfile
  - Merge the 2 Build Info
  - Publish Build Info

Success:

- See the Build Info for 2 profiles in Artifactory

Lab10 - Generate, merge and publish a Build Info

```
$ git clone https://github.com/conan-ci-cd-training/App2.git && cd App2
$ conan_build_info --v2 start app2 1 && cat ~/.conan/artifacts.properties

# create App2 release package
$ conan graph lock . --profile=debug-gcc6 --lockfile=app2_debug.lock -r conan-develop
$ conan create . mycompany/stable --lockfile=app2_debug.lock

# create App2 debug package
$ conan graph lock . --profile=release-gcc6 --lockfile=app2_release.lock -r conan-develop
$ conan create . mycompany/stable --lockfile=app2_release.lock

# upload App2
$ conan upload App2/1.0@mycompany/stable -r conan-develop --confirm --force

# create build infos
$ conan_build_info --v2 create debug_bi.json --lockfile=app2_debug.lock --user=conan --password=conan2020 && cat debug_bi.json
$ conan_build_info --v2 create release_bi.json --lockfile=app2_release.lock --user=conan --password=conan2020 && cat release_bi.json

# create the aggregated build info
$ conan_build_info --v2 update --output-file app2_bi.json debug_bi.json release_bi.json && cat app2_bi.json

# publish the build info and remove build properties
$ conan_build_info --v2 publish app2_bi.json --url=http://jfrog.local:8081/artifactory --user=conan --password=conan2020
$ conan_build_info --v2 stop && cat ~/.conan/artifacts.properties
```

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

Build Info - Good to know

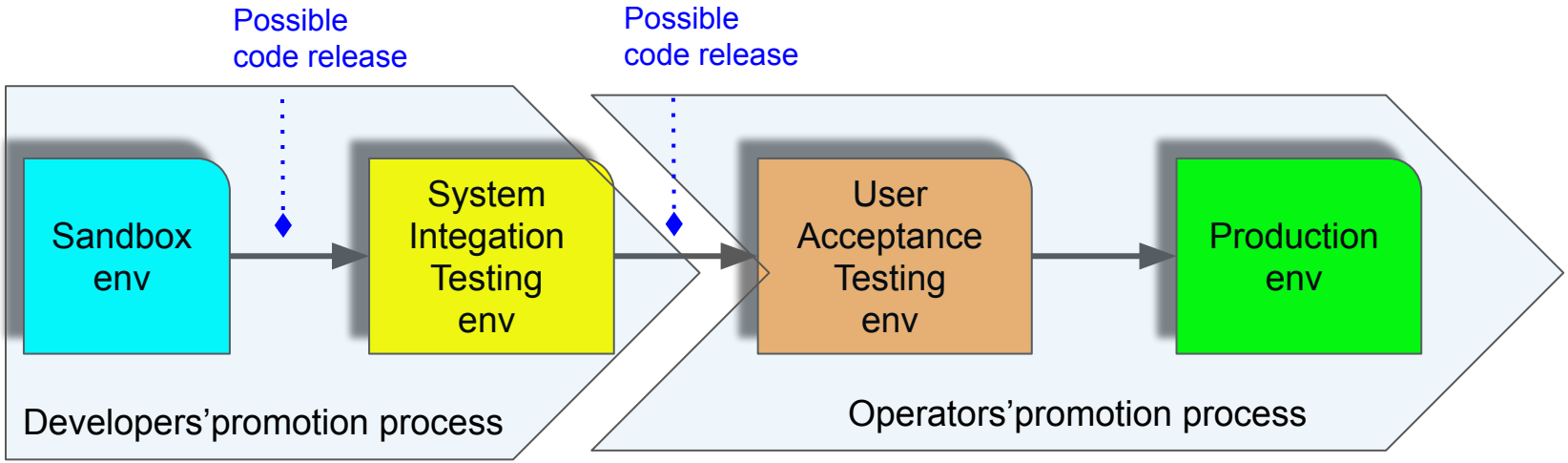
- MAY NOT fit the use case where an artifact is referenced by multiple Build Info AND NOT considered as a Build Info dependency
- 2 alternatives of Build Info :
  a. All the files from the Artifact section should be packaged into an archive which will be the result of your Build Info
  b. Aggregate all the files using your own custom properties.
     - A property can be a key/value pair or a key/list of values

Outline

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

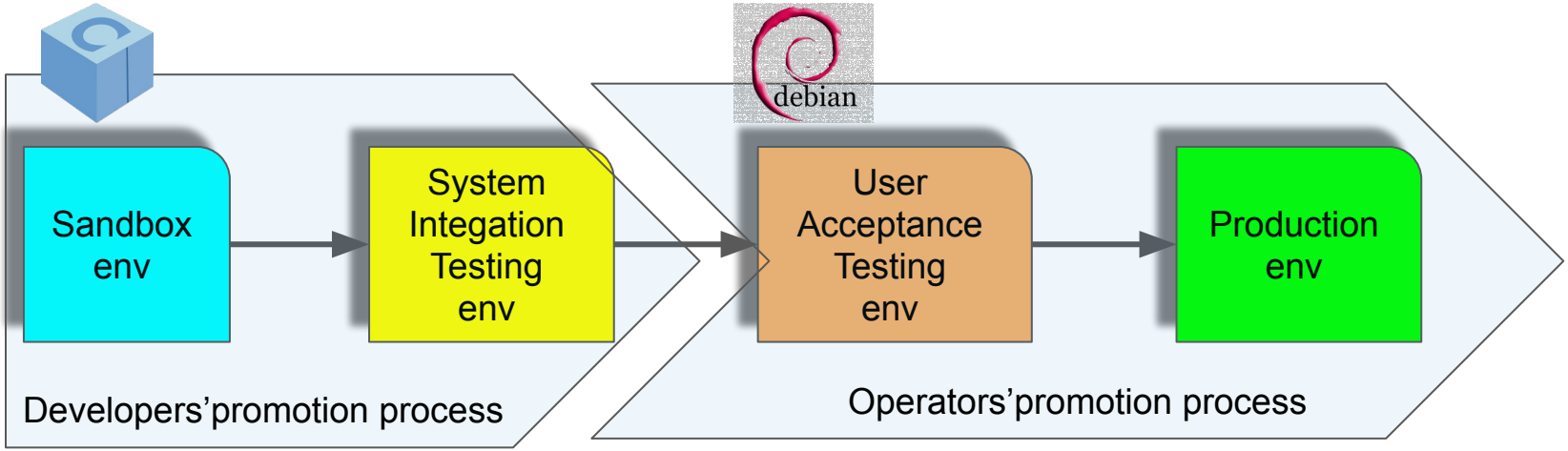
Promotion vs Release

- 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 !



Dev and Ops promotion process

- Dev promotion process = promoting App Conan package
- Ops promotion process = promoting App Debian package



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 promotion
  - 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

Lab11 - Build Info Promotion

Goal:

- Promote Build Info by move without dependencies

Task:

- Create a debian package from the App2 conan package + upload it to Artifactory
- Create a build info referencing the debian package + adding a lockfile as a dependency
- Publishing the Build Info and promote it

Success:

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

Lab11 - Build Info Promotion

```
$ jfrog rt c --interactive=false  --url=http://jfrog.local:8081/artifactory --user=conan --password=conan2020
art7
# generate and upload Debian package from App2 Conan package
$ cd ~/conan_ci_cd/labs && chmod +x generateDebianPkg.sh && ./generateDebianPkg.sh conan conan2020

# create custom Build Info
$ jfrog rt u debian_gen/myapp2_1.0.deb app-debian-sit-local/pool/ --build-name=myapp2 --build-number=1
$ jfrog rt d conan-metadata/app_release.lock --build-name=myapp2 --build-number=1
$ jfrog rt bad myapp2 1 conan_package.tgz
$ jfrog rt bce myapp2 1
$ jfrog rt bp myapp2 1

# Promote with JFrog CLI
$ jfrog rt bpr myapp2 1 app-debian-uat-local --status="SIT_OK"  --comment="passed integration tests"
--include-dependencies=false --copy=false
```

Promotion - Good to know

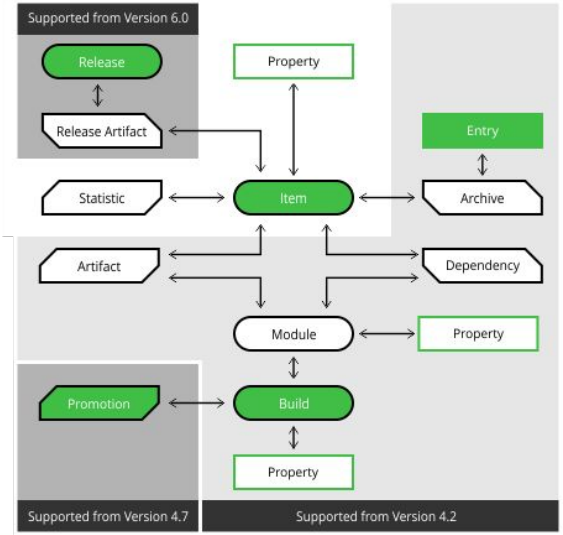
- When promoting by copy :
  - This will create more artifacts (not binaries)
  - Any AQL and filespec have to target a repository name
- Build Info promotion with / without dependencies
  - Depends on your project structure and delivery process
- Limitation : A unique target repository

Outline

- Introduction
- Conan reminder
- CI
- Build info in Artifactory
- Promotion in Artifactory
- **Appendix**
- Pipelines and optimizations?

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
```

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

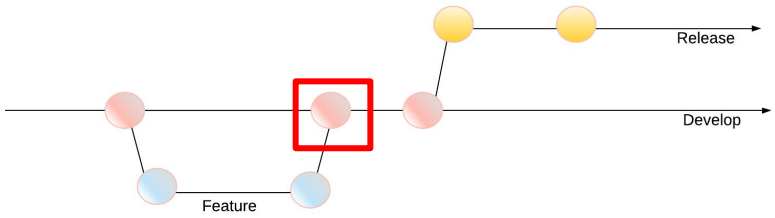
Download all artifacts from a Build Info

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

$ jfrog rt ping

# download all conan_package.tgz from build info + extract content into
folders
$ jfrog rt download --spec=filespec.json
```

Outline

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

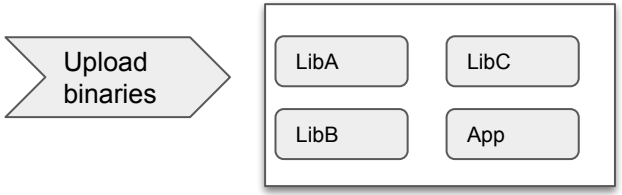
Bugs found in Sandbox



PR & Merge code from Feature to Develop branch

Code workflow

conan-develop repo

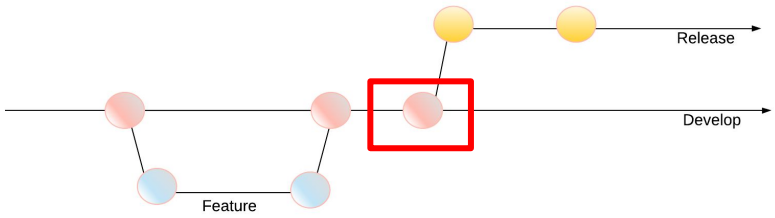


Binary workflow

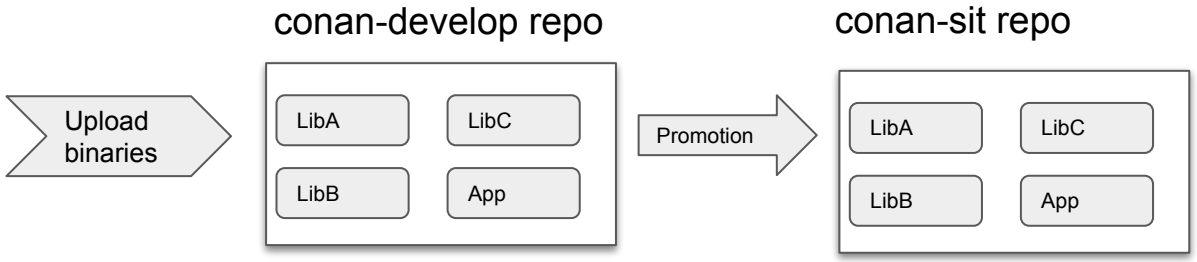
Deployment to Sandbox - Test KO

Delivery status

Bug fixes : App1 promoted to SIT



Code workflow

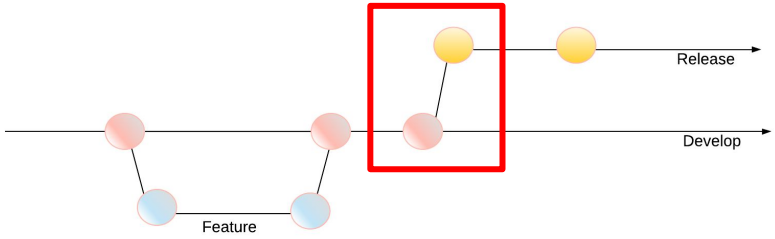


Binary workflow

Deployment to Sandbox - Test OK

Delivery status

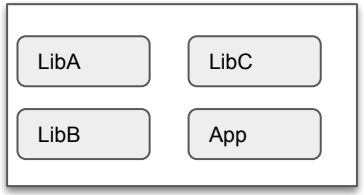
Bugs found in SIT



Create Release branch

Code workflow

conan-sit repo

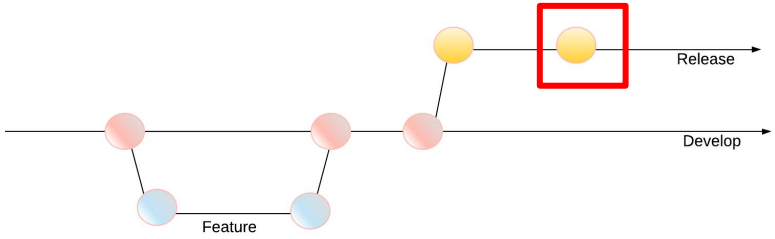


Binary workflow

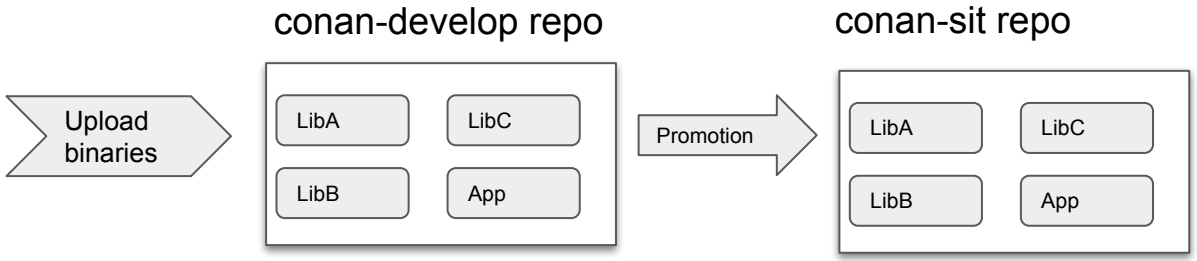
Deployment to SIT - Test KO

Delivery status

App1 : Code release



Code workflow



Binary workflow

Deployment to Sandbox - Test OK
Deployment to SIT - Test OK
Ready for Code Release

Delivery status