MOPV041

MODERNISATION OF THE TOOLCHAIN AND CONTINUOUS INTEGRATION OF FRONT-END COMPUTER SOFTWARE AT CERN

P. Mantion, S. Deghaye, L. Fiszer, F.Irannejad, J. Lauener, M. Voelkle, CERN, Geneva, Switzerland

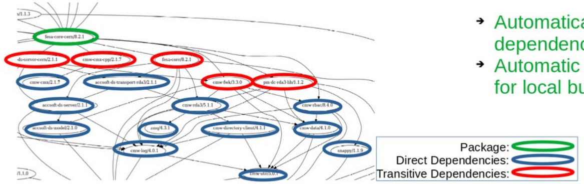
ICALEPCS 2021

Automatic Dependency Management with Conan

Legacy Makefile-based system

Each library must declare direct and transitive dependencies in Makefile

- ➔ Changing a transitive dependency requires manually editing many makefiles
- ➔ Produced artifacts might have inconsistent dependencies



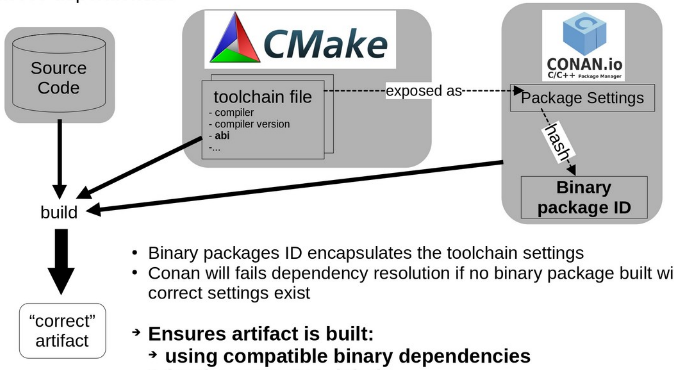
Conan + Cmake based system

JFrog CONAN C/C++ Package Manager

- Declare only **direct** dependencies in the package's Conanfile
- Conan transitively checks dependencies for conflicts

- ➔ Automatically ensures consistent dependencies versions
- ➔ Automatic setup of dependency paths for local build

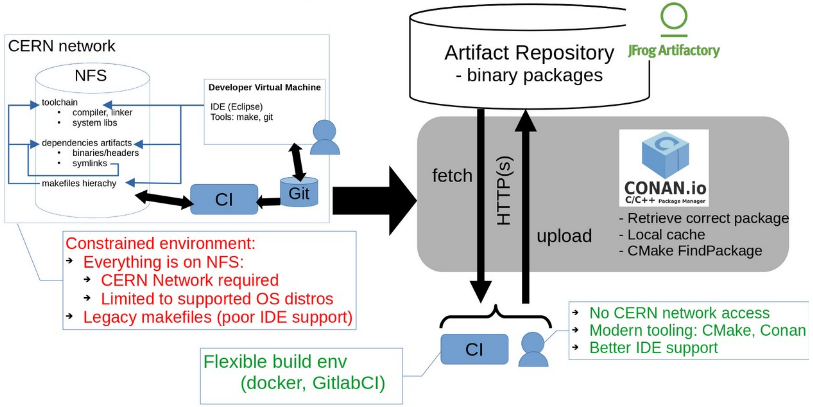
Cmake + Conan: support multiple toolchains & ensure consistent toolchain across dependencies



- Binary packages ID encapsulates the toolchain settings
- Conan will fails dependency resolution if no binary package built with the correct settings exist

- ➔ **Ensures artifact is built:**
  - ➔ **using compatible binary dependencies**
  - ➔ **for the correct toolchain**

More Flexible Development Environment



- Retrieve correct package
- Local cache
- CMake FindPackage

Constrained environment:

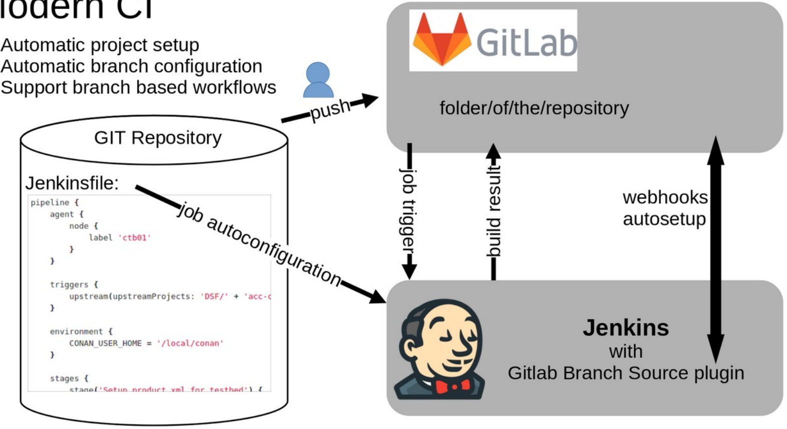
- ➔ Everything is on NFS:
  - ➔ CERN Network required
  - ➔ Limited to supported OS distros
- ➔ Legacy makefiles (poor IDE support)

Flexible build env (docker, GitlabCI)

- ➔ No CERN network access
- ➔ Modern tooling: CMake, Conan
- ➔ Better IDE support

Modern CI

- ➔ Automatic project setup
- ➔ Automatic branch configuration
- ➔ Support branch based workflows



Jenkinsfile:

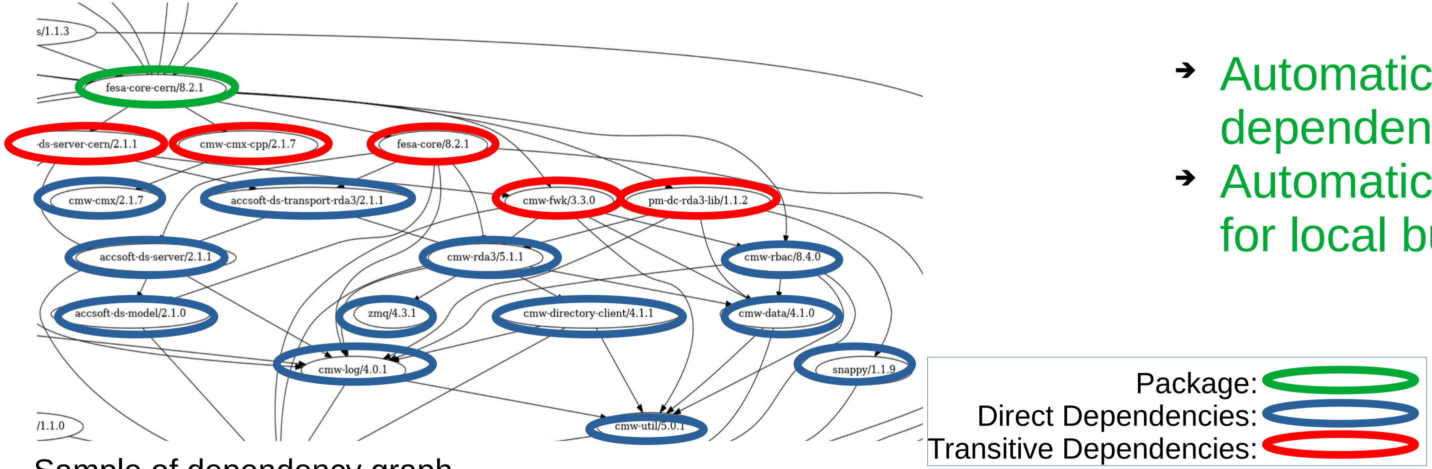
```
pipeline {
    agent {
        node {
            label 'ctb01'
        }
    }

    triggers {
        upstream(upstreamProjects: 'DSF/' + 'acc-c
    }

    environment {
        CONAN_USER_HOME = '/local/conan'
    }

    stages {
        stage('Setup product xml for testbed') {
```

Automatic Dependency Management with Conan

Legacy Makefile-based system

Each library must declare direct and transitive dependencies in Makefile

- Changing a transitive dependency requires manually editing many makefiles
- Produced artifacts might have inconsistent dependencies



Sample of dependency graph

Conan + Cmake based system

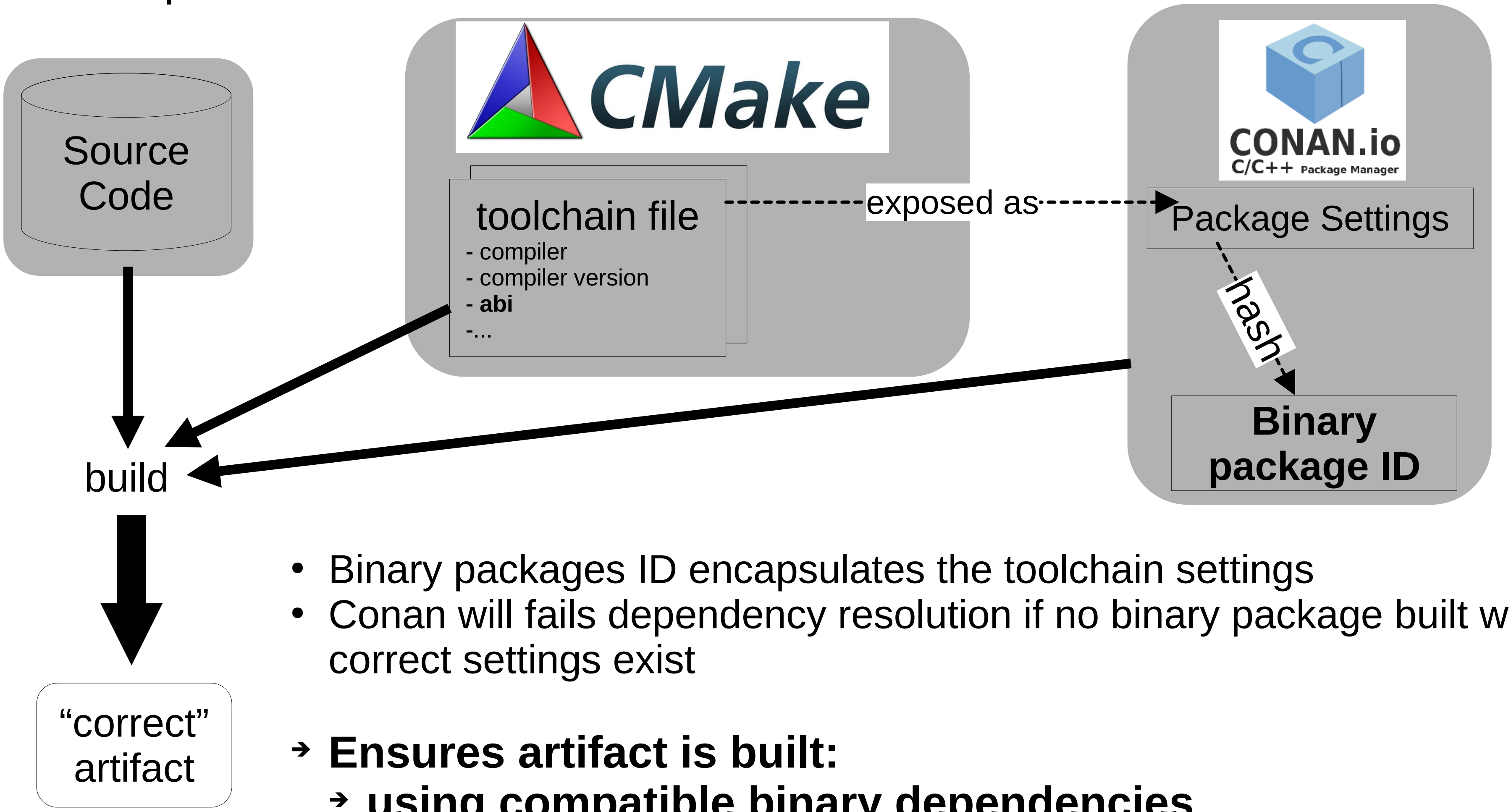
JFrog

CONAN
C/C++ Package Manager

- Declare only **direct** dependencies in the package’s Conanfile
- Conan transitively checks dependencies for conflicts

- Automatically ensures consistent dependencies versions
- Automatic setup of dependency paths for local build

Cmake + Conan: support multiple toolchains & ensure consistent toolchain across dependencies



Source Code

CMake

toolchain file

- compiler
- compiler version
- **abi**
-...

exposed as

CONAN.io C/C++ Package Manager

Package Settings

hash

Binary package ID

build

“correct” artifact

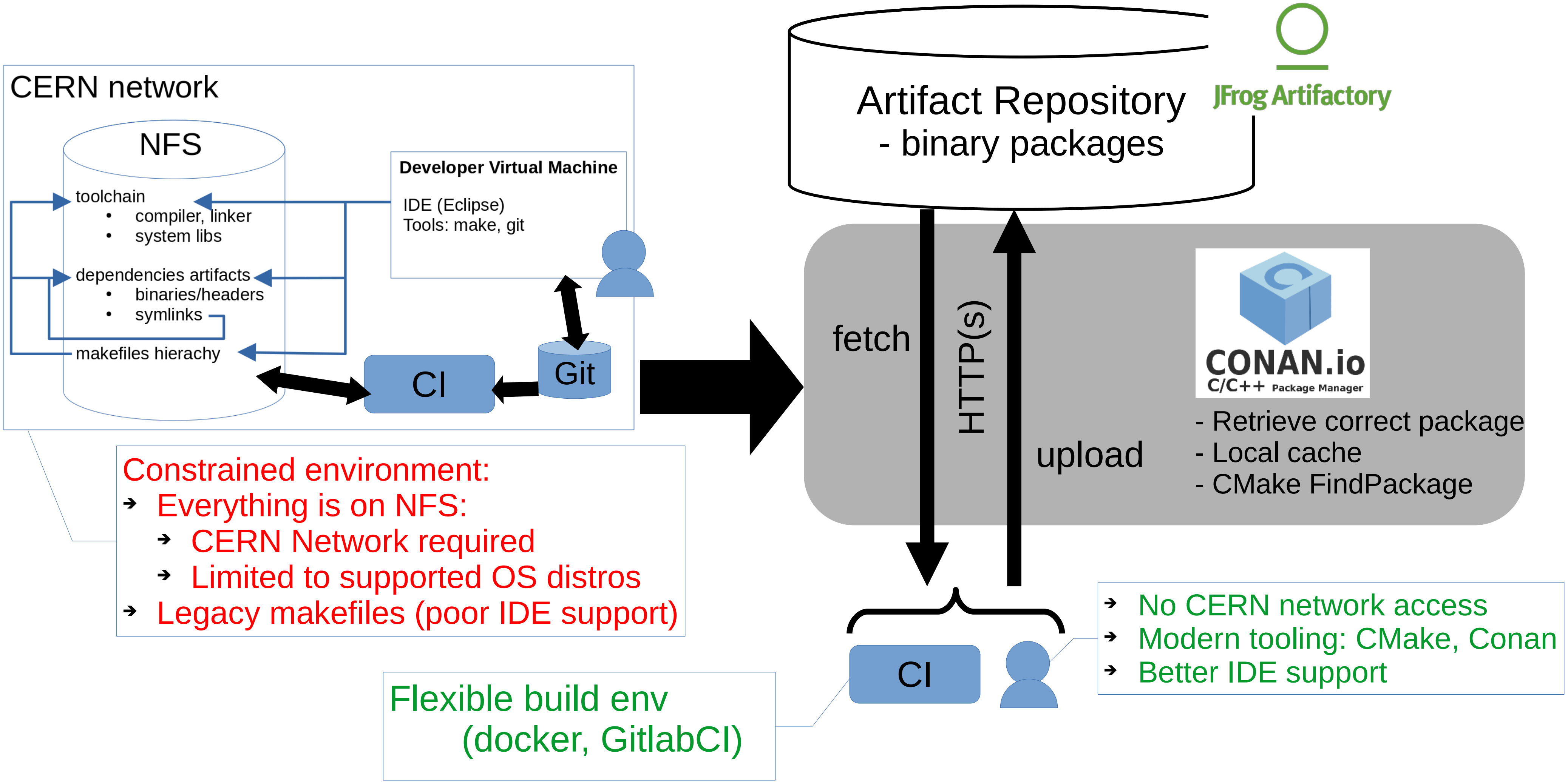
- Binary packages ID encapsulates the toolchain settings
- Conan will fails dependency resolution if no binary package built with the correct settings exist

➔ **Ensures artifact is built:**

➔ **using compatible binary dependencies**

➔ **for the correct toolchain**

More Flexible Development Environment



CERN network

NFS

toolchain

- compiler, linker
- system libs

dependencies artifacts

- binaries/headers
- symlinks

makefiles hierachy

Developer Virtual Machine

IDE (Eclipse)
Tools: make, git

- Retrieve correct package
- Local cache
- CMake FindPackage

Constrained environment:

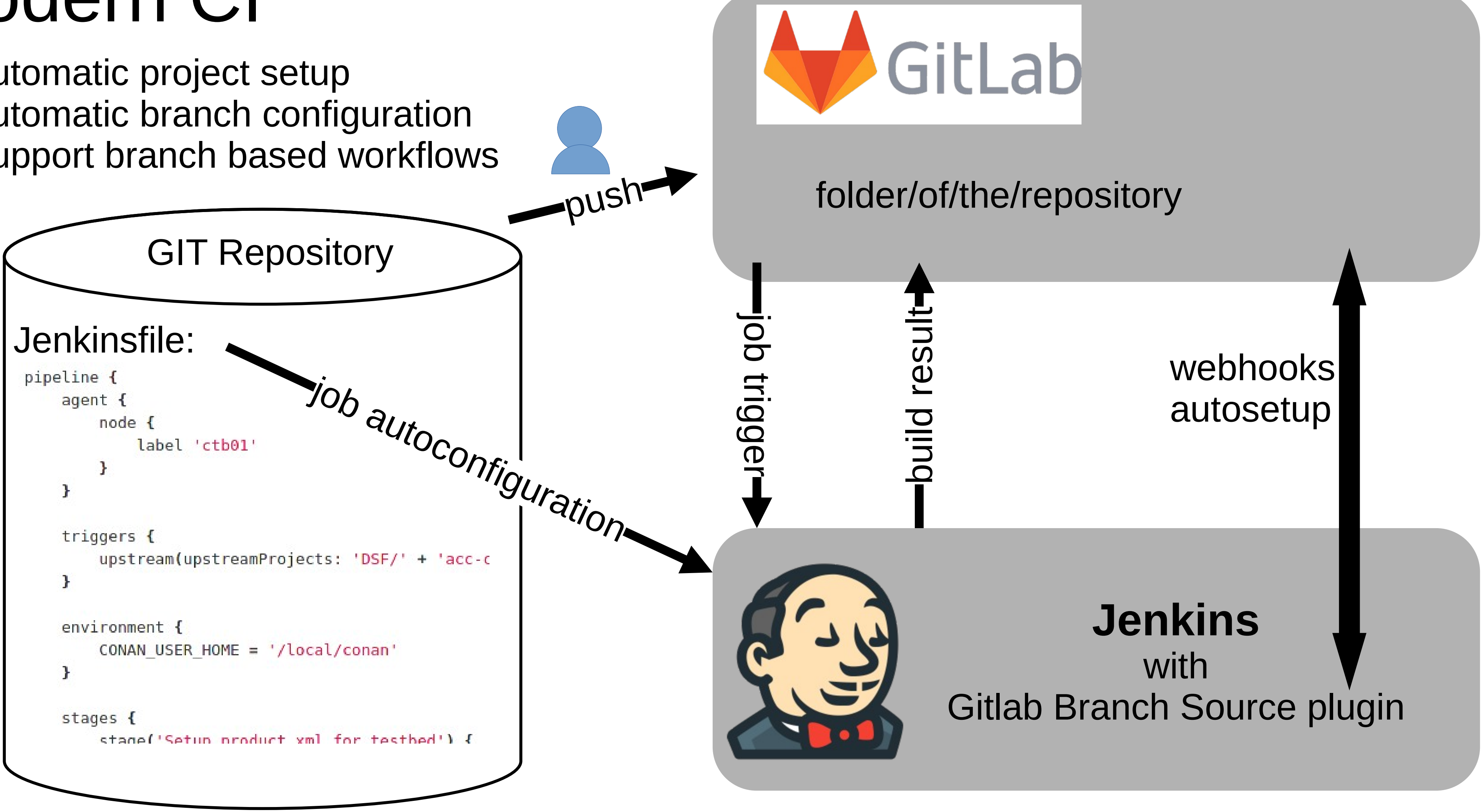
- Everything is on NFS:
  - CERN Network required
  - Limited to supported OS distros
- Legacy makefiles (poor IDE support)

Flexible build env
(docker, GitlabCI)

- No CERN network access
- Modern tooling: CMake, Conan
- Better IDE support

Modern CI

- Automatic project setup
- Automatic branch configuration
- Support branch based workflows



```
pipeline {
    agent {
        node {
            label 'ctb01'
        }
    }

    triggers {
        upstream(upstreamProjects: 'DSF/' + 'acc-c
    }

    environment {
        CONAN_USER_HOME = '/local/conan'
    }

    stages {
        stage('Setup product xml for testbed') {
```