Modern Software Development for JUNO offline software

Xingtao Huang, Teng Li, Weidong Li, Tao Lin, Jiaheng Zou

lintao@ihep.ac.cn

CHEP 2023

Norfolk, Virginia, USA

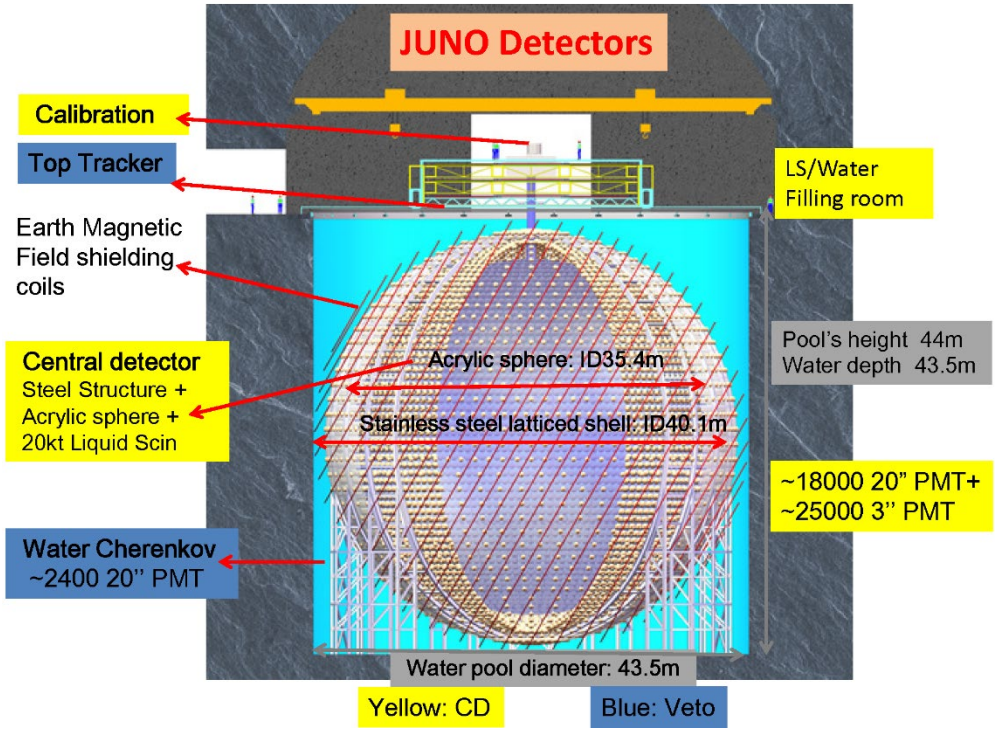
May 8-12, 2023

Outline

- Introduction
- Modern software development for JUNOSW
  - Development using CMake and Git
  - Deployment using junoenv, container, and Gitlab runner
- Summary and plan

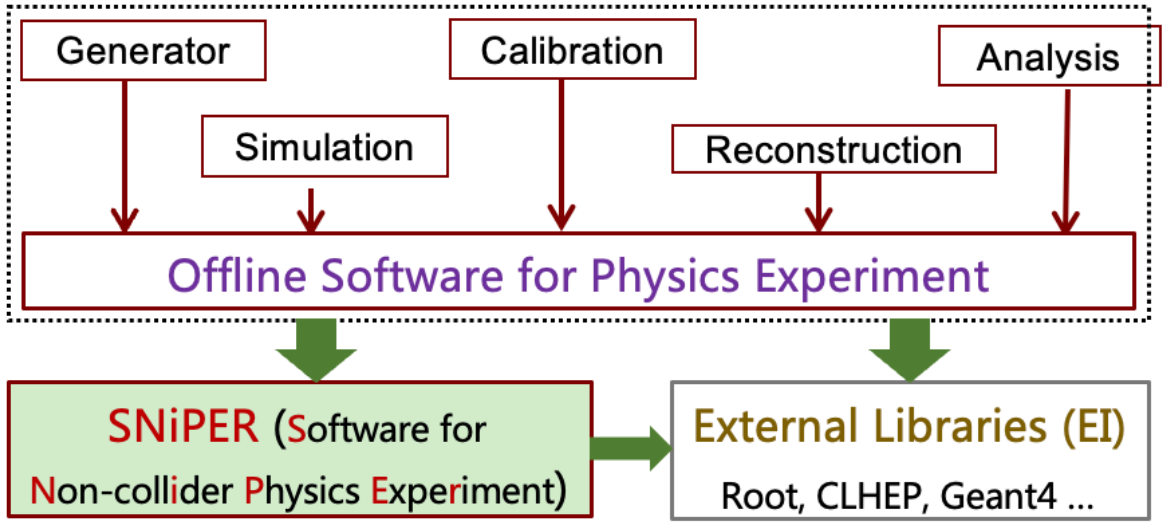
Introduction

- JUNO: Jiangmen Underground Neutrino Observatory
  - Rich physics program [1]
    - Neutrino Mass Ordering and Precise measurement of 3 oscillation parameters
    - Reactor neutrinos, Supernova burst neutrinos, Geo neutrinos, Atmospheric neutrinos, and Solar neutrinos
  - JUNO detector
    - 700 m deep underground
    - Central detector: 20 kton LS with 3%@1MeV of energy resolution.
    - Water Cerenkov detector and Top Tracker
  - Lifetime: 20+ years



[1] JUNO Collaboration, Prog.Part.Nucl.Phys. 123 (2022) 103927

JUNOSW: JUNO offline software



Applications: SNiPER Algorithms
Core Software: SNiPER framework, ROOT I/O, Database, etc.
External Libraries: ROOT, Geant4, Boost, Python, etc.

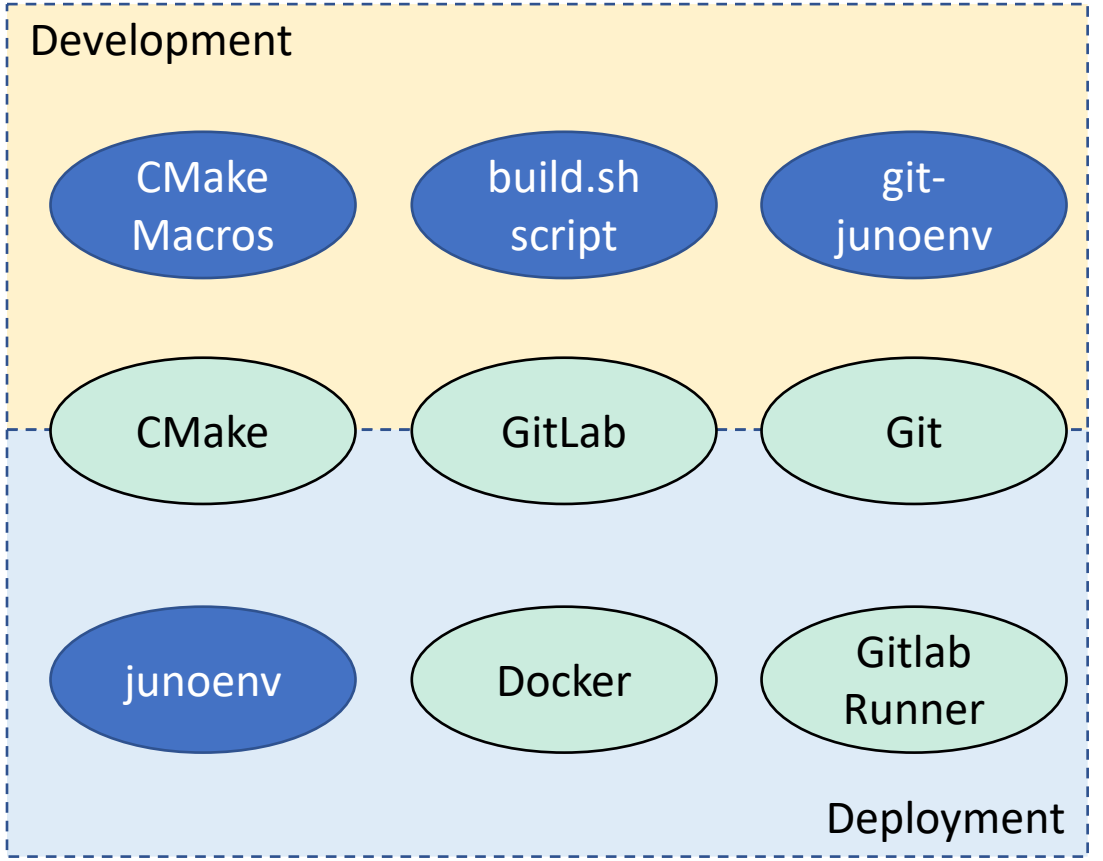
For details, see Xingtao's talk.

The development of JUNOSW (originally called JUNO offline) starts in 2012.

- Operating System
  - SL5 -> SL6 -> CentOS 7
- Compiler
  - GCC 4.1.2 -> 4.4.7 -> 4.9.4 -> 8.3.0 -> 11.2.0
- Build tool and version control
  - CMT -> CMake
  - SVN/Trac -> Git/Gitlab

Modern software development in JUNOSW

- Following the best practices from HSF [1], the migration to CMake/Git had been done for JUNOSW.
  - Use CMake to reduce the building time and support the different use cases for online and offline environments.
  - Adopt the git-based workflow to improve the code quality with code review and CI.
- To migrate smoothly, helper scripts have been developed.



[1] HSF-CWP-2017-13: Software Development, Deployment and Validation. arXiv:1712.07959

CMake (1)

- When moving from CMT to CMake, follow the instructions in Modern CMake [1].
- CMake macros and functions are developed to simplify the usage of CMake.
  - All the packages in JUNOSW are built in the same manner.
  - The source code files under “src” are automatically used to build the library.
  - Targets are used to describe the dependencies.
  - Extra environment variables could be set.

CMT

```
package Geometry
use Identifier      v* Detector
use ROOT            v* Externals
macro_append ROOT_linkopts " `root-config --evelibs` "
macro_append Boost_linkopts " -lboost_filesystem -lboost_system "
set JUNO_GEOMETRY_PATH "${GEOMETRYROOT}"
apply_pattern install_more_includes more="Geometry"
library Geometry *.cc
apply_pattern linker_library library=Geometry
apply_pattern install_python_modules
```

cmake

```
include (PKG)
PKG(Geometry
    DEPENDS
        Identifier
        boost_filesystem boost_system
    SETENV
        JUNO_GEOMETRY_PATH="${CMAKE_CURRENT_SOURCE_DIR}"
)
```

[1] Modern CMake: https://cliutils.gitlab.io/modern-cmake/

CMake (2)

In order to support both online and offline environments, the following strategies are used:

- A script called “build.sh” is used for both cases. Use the environment variable to switch the build mode.
- When the online mode is detected, the main CMakeLists.txt will load CMakeLists.online.txt instead of CMakeLists.default.txt.
- Generator expressions are used to control the build flags.

```
function run-build() {
    local installdir=$(install-dir)
    local blddir=$(build-dir)
    check-build-dir
    check-install-dir

    pushd $blddir

    cmake .. $(check-var-enabled graphviz) \
             $(check-var-enabled withoec) \
             $(check-var-enabled online) \
             $(check-var-enabled PerformanceCheck) \
             -DCMAKE_CXX_STANDARD=17 \
             -DCMAKE_BUILD_TYPE=$(cmake-build-type) \
             -DCMAKE_INSTALL_PREFIX=$installdir \
                     || error: "ERROR Found during cmake stage. "

    local njobs=-j$(nproc)
    cmake --build . $njobs || error: "ERROR Found during make stage. "
    cmake --install . || error: "ERROR Found during make install stage. "

    popd
}
```

```
if(BUILD_ONLINE)
  message(STATUS "Using online OEC packages lists")
  include (${CMAKE_SOURCE_DIR}/CMakeLists.online.txt)
elseif (EXISTS "${CMAKE_SOURCE_DIR}/CMakeLists.user.txt")
  message(STATUS "Using user customized package lists")
  find_package(junosw)
  include ("${CMAKE_SOURCE_DIR}/CMakeLists.user.txt")
else()
  message(STATUS "Using default package lists")
  include ("${CMAKE_SOURCE_DIR}/CMakeLists.default.txt")
endif()
```

```
$<$<NOT:$<BOOL:${BUILD_ONLINE}>>:Parameter>
```

git-junoenv

- When using SVN and CMT, developers could check out and build some packages instead of the whole project.
- In order to support this feature, a shell script called git-junoenv is developed to extend git, managing the packages to be checkout and built.
  - The core part is still the sparse checkout in git. The git ls-files is used to list the available packages.
  - When a package is checked out, it will be also registered in CMakeLists.user.txt. When building the project, CMake will use CMakeLists.user.txt.

```
$ git junoenv init-project junosw && cd junosw       # get the junosw without packages
$ git junoenv list-pkgs                              # list all the available packages
$ git junoenv add-pkg Reconstruction/OMILREC         # checkout a package
```

junoenv (1)

- It is a collection of shell scripts to deploy external libraries and JUNOSW.
  - Modularized by bash functions, which are inspired by ENV [1].
  - About 50+ external libraries are supported. About 30+ libraries are deployed.
  - Patches for the external libraries are also maintained in the same project.
  - The versions are collected for each release.

```
function juno-ext-libs-git-version- { echo 2.37.3 ; }
function juno-ext-libs-cmake-version- { echo 3.24.1 ; }
function juno-ext-libs-python-version- { echo 3.9.14 ; }
function juno-ext-libs-python-setuptools-version- { echo 58.1.0 ; }
function juno-ext-libs-python-pip-version- { echo 22.2.2 ; }
function juno-ext-libs-python-cython-version- { echo 0.29.28 ; }
function juno-ext-libs-python-numpy-version- { echo 1.22.3 ; }
```

[1] ENV, developed by Simon Blyth, https://juno.ihep.ac.cn/~blyth/env/notes/

junoenv (2)

- When deploying the software into CVMFS, the prefix is different from the original prefix when building software.
  - However, we can't run "make install" again on the CVMFS server.
- Use the paths with the same length and replace the prefix path
  - Inspired spack [1]
  - Solves the problems when the libraries or files are hardcoded with full paths.
  - Use "sed" to replace the paths for both libraries and software links.

```
# 2023.04.19
srcdir()        { echo /tmp/ihep.ac.cn/juno/b/centos7_amd64_gcc1120/Pre-Release/J23.1.0-rc0 ; }
dstdir()        { echo /cvmfs/juno.ihep.ac.cn/centos7_amd64_gcc1120/Pre-Release/J23.1.0-rc0 ; }
```

[1] spack: https://github.com/spack/spack

Docker image

- Two types of Docker images are prepared
  - Lightweight image with CVMFS clients installed.
  - Full image with all the external libraries installed.

	Lightweight image	Full image
Use cases	CI; Develop in users' computer (need network)	Develop in users' computer (offline)
Image size (compressed)	372.87 MB (CentOS 7)	17.29 GB (CentOS 7) 20.37 GB (Ubuntu 22.04)
Privileged in Docker	Needed (required by CVMFS)	Not necessary

.gitlab-ci.yml 1.97 KiB

```yaml
variables:
  JUNOTOP: /cvmfs/juno.ihep.ac.cn/centos7_amd64_gcc1120/Pre-Release/J23.1.x
  JUNO_CLANG_PREFIX: /cvmfs/juno.ihep.ac.cn/centos7_amd64_gcc1120/contrib/clang/12.0.0

default:
  image: mirguest/juno-cvmfs

stages:          # List of stages for jobs, and their order of execution
  - build
  - test

build-job-gcc:       # This job runs in the build stage, which runs first.
  stage: build
  script:
    - sudo mount -t cvmfs juno.ihep.ac.cn /cvmfs/juno.ihep.ac.cn
    - source $JUNOTOP/setup.sh
    - ./build.sh
```

The lightweight image is used in the Gitlab CI. CVMFS is mounted first. Then the runtime environment is setup.

When the external libraries are upgraded, the CI could access them by using the latest release.

Gitlab Runner

- Gitlab Group Runners are deployed in a self-hosted Kubernetes cluster.
  - Gitlab agents are used to connect Gitlab and Kubernetes. Then the Gitlab runners are managed by the Gitlab agents.
  - All the configurations are managed in GitLab repositories.

main / gitlab-agent / .gitlab / agents / juno-offline-k8s / config.yaml

config.yaml 222 bytes

```yaml
gitops:
  manifest_projects:
  - id: JUNO/offline/gitlab-agent
    default_namespace: junooffline
    paths:
    - glob: 'manifests/*.{yaml,json}'
    - glob: '/**/*.{yaml,json}'

ci_access:
  groups:
  - id: JUNO/offline
```

master / cluster-management / applications / gitlab-runner / helmfile.yaml

helmfile.yaml 221 bytes

```yaml
repositories:
- name: gitlab
  url: https://charts.gitlab.io

releases:
- name: runner
  namespace: gitlab-managed-apps
  chart: gitlab/gitlab-runner
  version: 0.44.0
  installed: true
  values:
    - values.yaml.gotmpl
```

Summary and plan

- Modern development and development for JUNOSW
  - CMake macros are developed and a high-level script is used by users.
  - git-junoenv is developed to partially check out and build packages.
  - junoenv is used to deploy JUNOSW and external libraries.
  - Docker and Kubernetes are also used to support CI.
- Plan: evaluate the migration from junoenv to spack
  - Spack is a package management tool, which has already been adopted in the Key4hep [1].
  - We hope to benefit from spack, including the reuse of the existing libraries and validation of the software stack by other experiments. This could be important when moving to new operating systems or architectures in the future.

[1] Valentin Volkl, vCHEP 2021, Building HEP Software with Spack

Abstract

- The Jiangmen Underground Neutrino Observatory (JUNO), under construction in South China, primarily aims to determine the neutrino mass hierarchy and the precise measure oscillation parameters. The data-taking is expected to start in 2024 and plans to run for more than 20 years. The development of JUNO offline software (JUNOSW) started in 2012, and it is quite challenging to maintain the JUNOSW for such a long time. In the last ten years, tools such as Subversion, Trac, and CMT had been adopted for software development. However, there are some new requirements, such as how to reduce the building time for the whole project, how to deploy offline algorithms to an online environment, and how to improve the code quality with code review and continuous integration. To meet the further requirements of software development, modern development tools are evaluated for JUNOSW, such as Git, GitLab, CMake, Docker, and Kubernetes. This contribution will present the software development system based on these modern tools for JUNOSW and the functionalities we have achieved: CMake macros are developed to simplify the build instructions for users; CMake generator expressions are used to control the build flags for the online and offline environments; a tool named git-junoenv is developed to help users partially checkout and build the software; a script is used to build and deploy the software on the CVMFS server; a Docker image with CVMFS client installed is created for continuous integration; a GitLab agent is set up to manage GitLab runners in Kubernetes with all the configurations in a GitLab repository. In late 2022, the migration had been done.