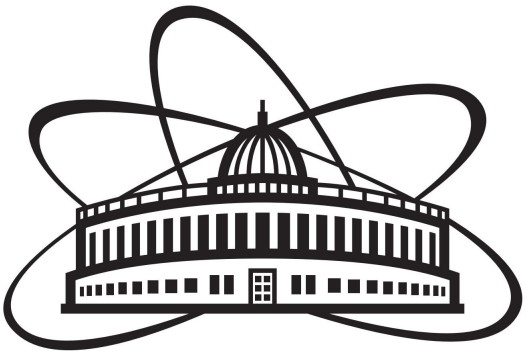
Joint Institute for Nuclear Research
Veksler and Baldin laboratory of High Energy Physics



Final report of the START programme

Application of containerization in the BM@N NICA experiment

Student:
Nizamov Rinat
Saint Petersburg State University, Russia

Supervisor:
Dr. Sergei Pavlovich Merts

Participation period:
July 9 - August 19

Dubna, 2023

Contents

1 Abstract

Software development for problems of modern physics is carried out in accordance with an increasing number of best practices developed in the IT industry. These methods help specialists to properly control code versions, ensure the reproducibility of the assembly and restrain the unrestrained growth of external dependencies. Tools developed in the software industry to solve these everyday problems have found application in NICA experiments.

One of these tools is containerization technology, which allows packaging software code with just the operating system (OS) libraries and dependencies required to run the code to create a single lightweight executable — called a **container** — that runs consistently on any infrastructure. This report describes: implementation of the image of the BmnRoot software package using local installation and CernVM File System service;

2 Introduction

NICA (Nuclotron based Ion Collider fAcility) is an accelerator complex that is being built on the basis of the Joint Institute for Nuclear Research (Dubna, Russia). The main goal of the project is to create a multitasking experimental complex for studying the properties of matter in conditions of heavy ion collisions. In Fig. 1 a schematic representation of the NICA accelerator complex is presented.

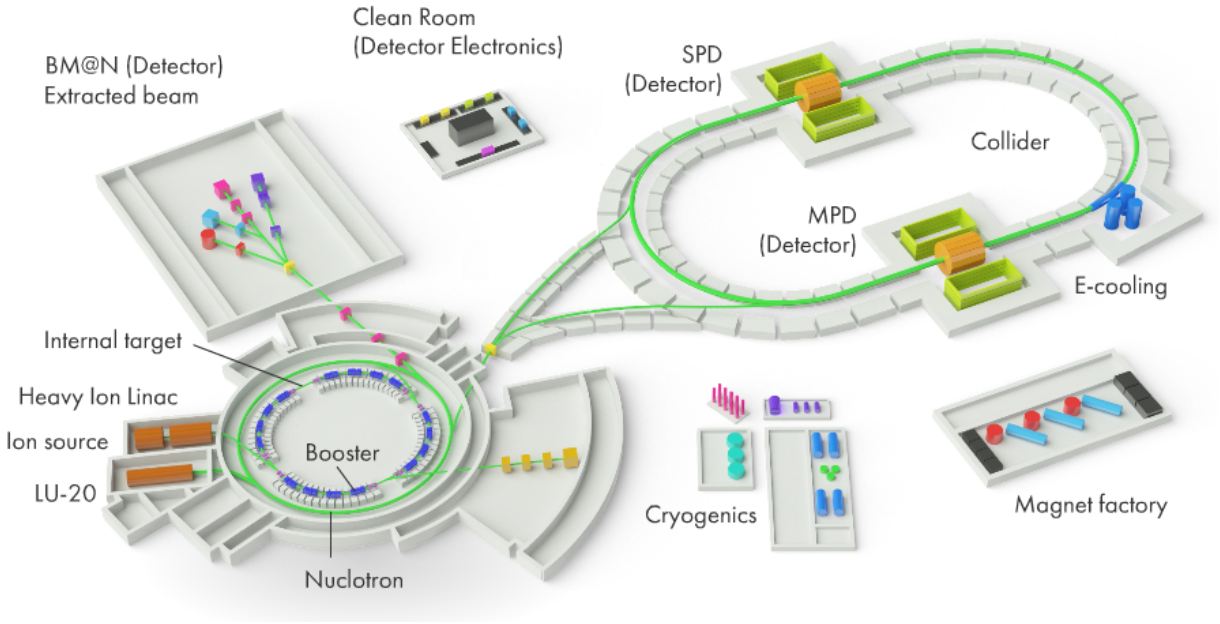


Figure 1: A schematic view of the NICA accelerator complex.

Heavy-ion collisions at high energies provide a unique opportunity to study nuclear matter under extreme density and temperature. These conditions are well suited to investigate the compressibility of nuclear matter, in particular, the stiffness of the nuclear equation-of-state (EoS) [1]. Theoretical models, however, suggest different possible scenarios for these modifications, – so that new experimental data with high resolution and statistics are needed in order to disentangle the different theoretical predictions.

BM@N (Baryonic Matter at Nucleotron) is the first experiment at the NICA accelerator complex to study collisions of particles and ions with a fixed target at the energies up to 6 AGeV for different beam species (up to krypton, at the moment). It covers studies of elementary reactions (pp, pA) and cold nuclear matter, the properties of dense baryonic matter formed in course of heavy-ion collisions, in-medium effects, hypermatter and strangeness production, as well as hadron femtoscopy[2].

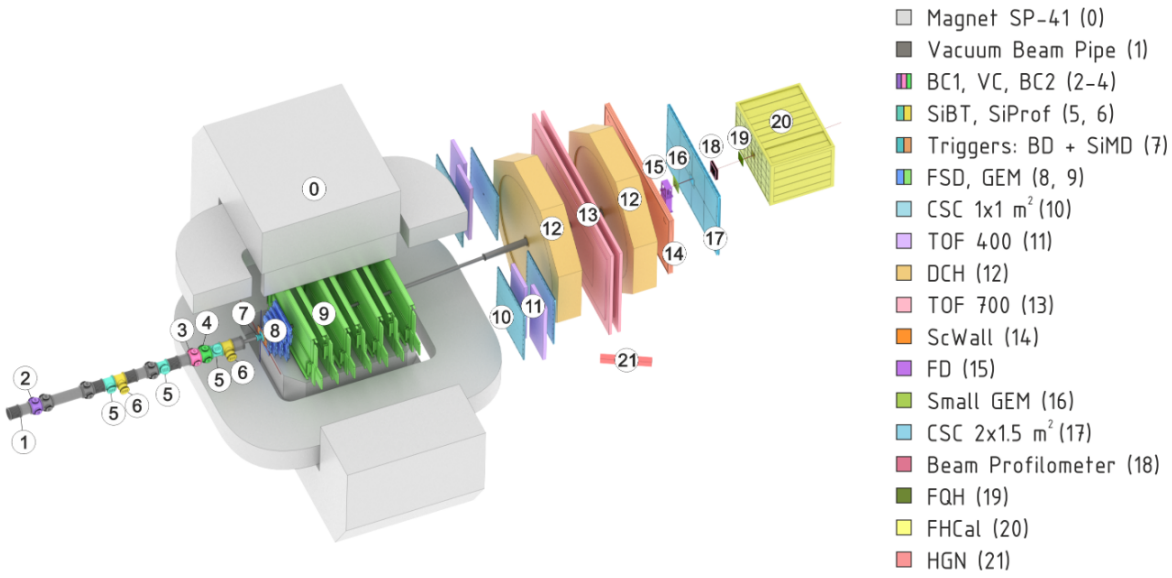


Figure 2: Schematic view of BM@N in the last Run

3 Theory overview

Definitely, the development of suitable software is a key factor in the success of the BM@N experiment. The software should allow simulating the passage of particles through the detectors and to reconstruct events taking into account the specific features of the detectors in which the reconstruction is performed [3].

3.1 BmnRoot framework

BmnRoot framework [4] provides a powerful tool for studying the effectiveness of detectors, modeling events and developing algorithms that will be used for reconstruction and physical analysis of events registered by the BM@N installation. The software is based on the FairRoot[5] framework, originally used for the CBM[6] experiment. The FairRoot package includes common core services covering and simplifying detector modeling, event reconstruction, and data analysis. In Fig. 3 shows the sequence of experimental data processing available using

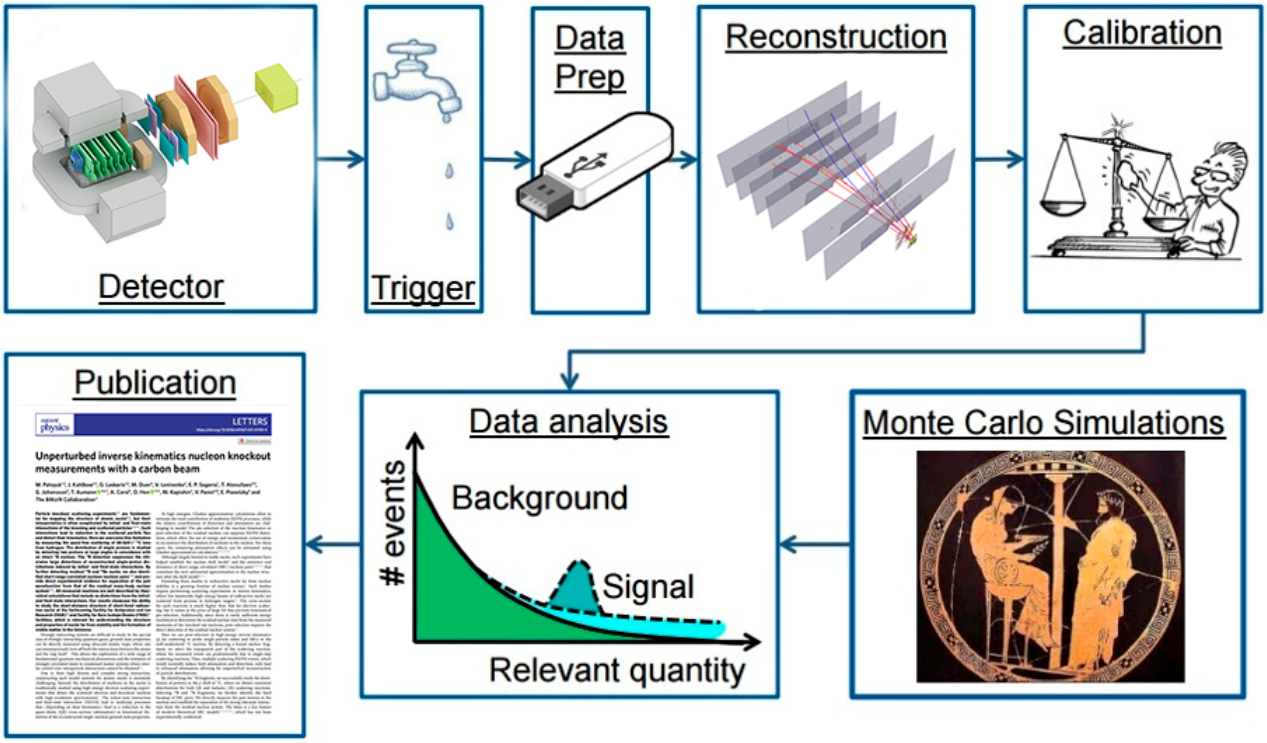


Figure 3: Experimental data processing chain in the BmnRoot framework

the BmnRoot software. It can be seen that data processing has a multi-level structure. First, the raw data from the detectors recorded by the data acquisition system (DAQ) is converted to *root* format [7] and digitized. Then a procedure, aimed at restoring the parameters of the particles in the detectors, is applied. As the final stage of the data transmission chain, a physical analysis is performed to study the physical properties of the nuclear substance formed during collisions.

3.2 Docker

Docker is an open platform for developing, shipping, and running applications. Docker[8] was based on the existing Linux container technology with a variety of wrappers and extensions, mainly using portable images and a user–friendly interface to create a completely ready-to-use solution that ensures the creation and distribution of containers. The Docker platform consists of two separate components: Docker Engine, the mechanism responsible for the creation and operation of containers, and Docker Hub, a cloud service for the distribution of containers.

The Docker Engine provides an efficient and user-friendly interface for launching containers. Prior to that, launching containers using such technology as, for example, LXC required a fair amount of special knowledge in this area and a large amount of manual work. Docker Hub provides numerous open access container images for downloading, allowing users to quickly get started with them and avoid routine work previously done by other people.

3.2.1 Docker Architecture

In Fig. 4 shows the main components of the Docker platform[9]

- **docker daemon** located in the center of the diagram, is responsible for creating, launching and controlling the operation of containers, as well as for creating and storing images. Containers and images are shown on the right side of the diagram. The Docker daemon is started by the *docker daemon* command, but usually the host operating system takes care of this;

- **the Docker client**, located on the left side of the diagram, is used for a dialog with the Docker daemon over the HTTP protocol. By default, this connection is established via a Unix domain socket, but a TCP socket can also be used to support connections to remote clients or a file descriptor for sockets managed by systemd. The application programming interfaces used to organize data exchange with the daemon are clearly defined and documented in detail, which allows developers to write programs that interact directly with the daemon without using the Docker client. The Docker client and daemon are distributed as separate, independent binaries;

- **docker registries** are used for storing and distributing images. The registry selected by default is Docker Hub[10], which stores thousands of public images, as well as managed "official" images. Many organizations create their own registries, which are used to store commercial and private images and to eliminate the overhead associated with downloading images over the Internet. The Docker daemon loads images from registries at the request of docker pull. In addition, it performs automatic loading of images specified in the docker run request and in the instructions FROM the Dockerfile file, if these images are not available on the local system.

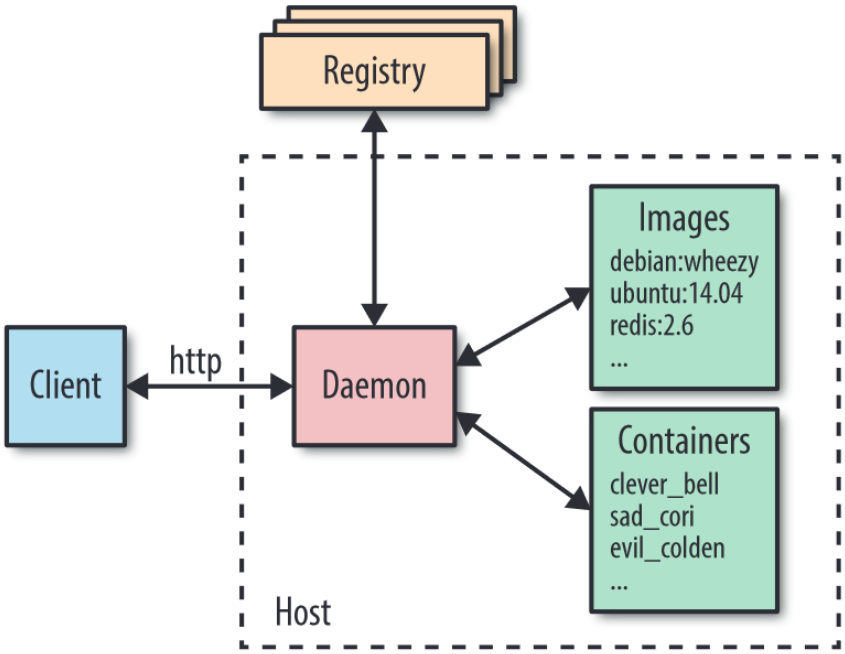


Figure 4: The general scheme of interaction of the main components of Docker

3.3 Apptainer

If consider the architecture of Docker at the operating system level, it can be seen that the basis of Docker is a service daemon called *dockerd*. The main purpose of this daemon is container management. This daemon can be accessed via the REST API via the Docker client using the command line interface. A frequently mentioned and important detail of the Docker daemon is that it only works if you have **superuser** privileges. It is safe to say that this fact is the biggest and main obstacle, because of which the Docker containerization system is not suitable and cannot be deployed within the framework of a high-performance cluster infrastructure.

Apptainer[11] is a containerization tool created for scientific software solutions. Apptainer is build in the way, that ordinary users can safely run their containers on a supercomputer system without the possibility and need to upgrade user privileges to the superuser level. Internally, Apptainer uses a simpler approach that does not require a daemon, unlike Doker. Images are executed with user rights using the UNIX SUID functionality. Apptainer uses the "namespace" function of the Linux kernel to isolate processes. However, Apptainer does not use the "cgroups" kernel functions or any other resource management methods and methods.

As mentioned earlier, Apptainer allows you to use Docker images both directly for execution and as a base image in the Apptainer build instructions. As in the case of Docker Hub, Apptainer has its own public registry of images. The main feature of Apptainer containers is that users can work without root privileges, which is required within the framework of the supercomputer infrastructure. That is why most of the solutions that are implemented today

within the framework of the supercomputer infrastructure are made using Apptainer

3.4 CernVM File System

The CernVM File System [12] provides a scalable, reliable and low-maintenance software distribution service. It was developed to assist High Energy Physics (HEP) collaborations to deploy software on the worldwide-distributed computing infrastructure used to run data processing applications. CernVM-FS is implemented as a POSIX read-only file system in user space (a FUSE module). Files and directories are hosted on standard web servers and mounted in the universal namespace /cvmfs.

Internally, CernVM-FS uses content-addressable storage and Merkle trees in order to store file data and meta-data. CernVM-FS uses outgoing HTTP connections only, thereby it avoids most of the firewall issues of other network file systems. It transfers data and meta-data on demand and verifies data integrity by cryptographic hashes.

4 Results

As a result, several images were created using Docker:

1. with a full local installation of FairSoft, FairRoot and BmnRoot inside the container;
2. with the installation of the CernVM-FS client and only BmnRoot inside the container.

Along with README.md instruction, all files can be found at **https://git.jinr.ru/nica/Docker-Images**.

4.1 Full local installation

The image was created using Docker and this Dockerfile 1:

```
1  FROM git.jinr.ru:5005/nica/docker-images/alma9/fair:apr22_v18.6.8
2
3  SHELL ["/bin/bash", "-c"]
4  #Env vars for the nvidia-container-runtime.
5  ENV NVIDIA_VISIBLE_DEVICES all
6  ENV NVIDIA_DRIVER_CAPABILITIES graphics,utility,compute
7  #Installing libs for graphics
8  RUN yum install -y xdg-utils xterm mesa-libGL mesa-libGL-devel libXext libX11 libX11-devel mesa
       -libGLw-devel.x86_64 firefox && dnf install -y mesa-dri-drivers
9  RUN git clone -b dev --recurse-submodules https://git.jinr.ru/nica/bmnroot.git
10 RUN mkdir bmnroot/build
11 RUN cd bmnroot && source SetEnv.sh && cd build && cmake -DCMAKE_INSTALL_PREFIX=/opt/
       bmnroot/install .. && make -j4 && make install
12 COPY entrypoint.sh /entrypoint.sh
13 RUN chmod +x /bmnroot/build/config.sh && chmod +x /entrypoint.sh
14
15 ENTRYPOINT ["/entrypoint.sh"]
```

Listing 1: Docker for full local install

```
1  #!/bin/bash
2  source /bmnroot/build/config.sh
3  bash
```

Listing 2: Simple enrtypoint script

This Dockerfile uses an image with FairSoft and FairRoot as the base image (line 1), which was preloaded to GitLab into the special container registry. Next, using *bash* shell, calling a sequence of commands to install all the necessary packages and OpenGL libraries to enable graphic output inside the container (line 5-8).

Line 10-12 – BmnRoot installation.

Line 14-15 – copying the entrypoint 2 script into the container and making it executable. Each launch of the container is accompanied by the launch of this script, which in turn launches the configuration file of the BmnRoot and open the *bash* shell.

The same image was made on the basis of Ubuntu 22.04. Using the premade script, user can choose a suitable base to work with.

4.2 Installation to use BmnRoot with CernVM-FS

A slightly different Dockerfile and Entrypoint file to build the CernVM-FS image, only the new entrypoint is presented here:

```
1  #!/bin/bash
2  sleep 2
3  export SIMPATH=/cvmfs/nica.jinr.ru/ubuntu2004/fairsoft/bmn
4  export FAIRROOTPATH=/cvmfs/nica.jinr.ru/ubuntu2004/fairroot/bmn
5  if ! [ -e /bmnroot/build/config.sh ];
6  then
7      THREADS=$(grep -c ^processor /proc/cpuinfo 2>/dev/null || sysctl -n hw.ncpu || echo "
       $NUMBER_OF_PROCESSORS")
8      if [[ $# -gt 3 ]]; then
9          THREADS=$((THREADS-1))
10     fi
11     cd /bmnroot && source SetEnv.sh && cd build && cmake .. && make -j$THREADS && cd /
12 else
13     source /bmnroot/build/config.sh
14 fi
15 bash
```

Listing 3: Enrtypoint script

A slightly more complex enrtypoint 3 script used here. Due to the lack of access rights inside the container, we can't use usual autoFS with CernVM-FS. In this case, we had to use pre-mounting option inside the container to enable needed repositories. **Pre-mounting** option is available in apptainer with *–fusemount* option. Example:

```
1  apptainer run --fakeroot --writable --fusemount "container:cvmfs2 nica.jinr.ru /cvmfs/nica.jinr.ru"
       cvmfs_ubuntu
```

Listing 4: Line to launch the container

How it works:

1. Only source files for BmnRoot and installed CernVM-FS client are located inside the container, which is based on Ubuntu 20.04 and does not contain any FairSoft or FairRoot files.

2. Due to the absence of FairSoft and FairRoot inside the container, we can't build BmnRoot when building the container.

3. To fix this problem, the BmnRoot installation launches in the first run of the container (line 7-18, entrypoint 3). At this moment, we have access to FairSoft and FairRoot, due to pre-mounting of CernVM-FS. With correctly initialized environment variables SIMPATH and FAIRROOTPATH (line 3-5, entrypoint 3), the build is successful.

4. In the following container run, only the config.sh script is started.

5. And the container is ready to work.

4.3 Using the installation script

To simplify the installation process, a special installation script was created. *install_ bmnroot_ container.sh* script installs an apptainer package, builds BmnRoot container and adds aliases to operate the container. By default, the script bypasses the apptainer installation if it is already present on the machine.

Three options provide configuring the installation:

- *–force/-f*: installs/reinstalls the apptainer package, even if it is already installed (It removes any version of the Apptainer or Singularity and installs Apptainer)

- *–os=preferredOS*: ubuntu OR almalinux. Allows you to choose the preferred operating system on which BmnRoot is based (AlmaLinux 9 chosen by default)

- *–cvmfs/-c*: installing BmnRoot to use it with CernVM-FS instead of Full local installation.

The script verifies, that the following packages are installed, such as *wget, xhost, xauth* and installs them in case of the absence. After that, the Apptainer is installed, the selected container on the chosen base is built and necessary aliases are added to start and reinstall the container. Usage example of the installation script and the following output provided at the Fig. 5.

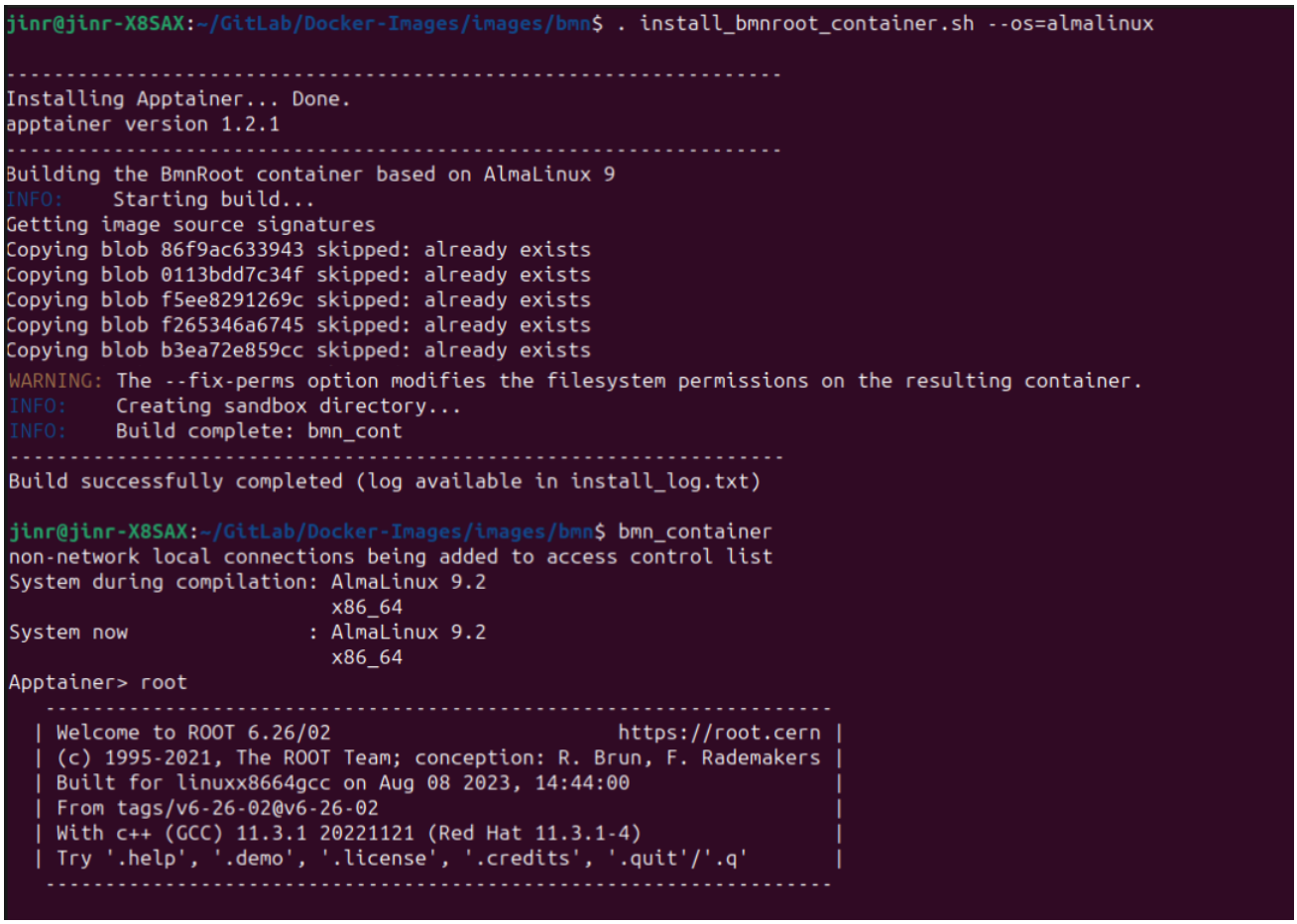


Figure 5: Using the installation script

5 Conclusion

This paper in detail describes the process of creating containers for the main software package of the BM@N experiment – BmnRoot. Four different containers were created for different user needs: containers with full local installation and containers that use CernVM-FS on AlmaLinux 9 and Ubuntu 22.04 bases. Some Dockerfiles are provided with detailed comments. The installation script were written to simplify the process of building and running the BmnRoot container. All mentioned files can be found at official NICA GitLab repository, *https://git.jinr.ru/nica/Docker-Images*.

6 Acknowledgments

On my own behalf, I would like to express my gratitude to Konstantin Gertsenberger and Nikita Balashov, for all their patience to my mistakes, immense knowledge and constant support during the whole program;

to Sergey Merts and all the colleagues I met here, for being good people and great professionals

References

[1] Mikhail Kapishin. Studies of baryonic matter at the BM@n experiment (JINR). *Nuclear Physics A*, 982:967–970, feb 2019, p. 1.

[2] Pavel Batyuk, Konstantin Gertsenberger, Sergey Merts, and Oleg Rogachevsky. The BmnRoot framework for experimental data processing in the BM@n experiment at NICA. *EPJ Web of Conferences*, 2019, pp. 1-2.

[3] P. Batyuk, D. Baranov, S. Merts, and O. Rogachevsky. Event reconstruction in the BM@n experiment. *EPJ Web of Conferences*, 204:07012, 2019.

[4] "BmnRoot" [software] BM@N project. Url: https://git.jinr.ru/nica/bmnroot/tree/dev, Дата обращения 13.08.2023.

[5] "FairRoot" [software] FAIR project. Url: https://github.com/fairrootgroup/fairroot, accessed: 14.08.2023.

[6] "CbmRoot" [software] CBM experiment. Url: https://git.cbm.gsi.de/computing/cbmroot, accessed: 14.08.2023.

[7] "Root" [software] ROOT project. Url: https://github.com/root-project/root, accessed: 14.08.2023.

[8] Ian Miell and Aidan Hobson Sayers. *Docker in Practice*. Manning Publications, 2016, pp. 3-18.

[9] Adrian Mouat. *Using Docker Developing and Deploying Software with Containers*. O'Reilly Media, Incorporated, 2015, pp. 3-10.

[10] Docker Hub Registry. Url: https://hub.docker.com/, accessed: 14.08.2023.

[11] "Apptainer" Apptainer project. Url: https://docs.sylabs.io/guides/2.6/user-guide/introduction.html, accessed: 10.08.2023.

[12] CernVM-FS docs. Url: https://cernvm.cern.ch/fs/, accessed 14.08.2023.