SOFTWARE TOOL ARTICLE

Geniac: Automatic Configuration GENerator and Installer for nextflow pipelines [version 1; peer review: 1 approved, 1 approved with reservations]

Fabrice Allain\(^{1-4}\), Julien Roméjon\(^{1-4}\), Philippe La Rosa\(^{1-4}\), Frédéric Jarlier\(^{1-4}\), Nicolas Servant\(^{1-4}\), Philippe Hupé\(^{1-5}\)

\(^{1}\)Institut Curie, Paris, F-75005, France  
\(^{2}\)U900, Inserm, Paris, F-75005, France  
\(^{3}\)PSL Research University, Paris, F-75005, France  
\(^{4}\)Mines Paris Tech, Fontainebleau, F-77305, France  
\(^{5}\)UMR144, CNRS, Paris, F-75005, France

Abstract
With the advent of high-throughput biotechnological platforms and their ever-growing capacity, life science has turned into a digitized, computational and data-intensive discipline. As a consequence, standard analysis with a bioinformatics pipeline in the context of routine production has become a challenge such that the data can be processed in real-time and delivered to the end-users as fast as possible. The usage of workflow management systems along with packaging systems and containerization technologies offer an opportunity to tackle this challenge. While very powerful, they can be used and combined in multiple ways thus increasing their usage complexity. Therefore, guidelines and protocols are required in order to detail how the source code of the bioinformatics pipeline should be written and organized to ensure its usability, maintainability, interoperability, sustainability, portability, reproducibility, scalability and efficiency. Capitalizing on Nextflow, Conda, Docker, Singularity and the nf-core initiative, we propose a set of best practices along the development life cycle of the bioinformatics pipeline and deployment for production operations which address different expert communities including i) the bioinformaticians and statisticians ii) the software engineers and iii) the data managers and core facility engineers. We implemented Geniac (Automatic Configuration GENerator and Installer for nextflow pipelines) which consists of a toolbox with three components: i) a technical documentation available at https://geniac.readthedocs.io to detail coding guidelines for the bioinformatics pipeline with Nextflow, ii) a linter to check that the code respects the guidelines, and iii) an add-on to generate configuration files, build the containers and deploy the pipeline. The Geniac toolbox
Corresponding author: Philippe Hupé (philippe.hupe@curie.fr)

Author roles: Allain F: Methodology, Software, Writing – Review & Editing; Roméjon J: Conceptualization, Methodology, Software, Writing – Review & Editing; La Rosa P: Methodology, Software, Writing – Review & Editing; Jarlier F: Validation, Writing – Review & Editing; Servant N: Methodology, Software, Writing – Review & Editing; Hupé P: Conceptualization, Funding Acquisition, Methodology, Software, Supervision, Writing – Original Draft Preparation, Writing – Review & Editing

Competing interests: No competing interests were disclosed.

Grant information: This research was financially supported by the European Union's Horizon 2020 research and innovation programme under the grant agreement No 825835 (project EUCANCan). This work was also supported by the Institut Curie and the Centre national de la recherche scientifique.

Copyright: © 2021 Allain F et al. This is an open access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

How to cite this article: Allain F, Roméjon J, La Rosa P et al. Geniac: Automatic Configuration GENerator and Installer for nextflow pipelines [version 1; peer review: 1 approved, 1 approved with reservations] Open Research Europe 2021, 1:76 https://doi.org/10.12688/openreseurope.13861.1

First published: 02 Jul 2021, 1:76 https://doi.org/10.12688/openreseurope.13861.1

This article is included in the Societal Challenges gateway.

Geniac aims at the harmonization of development practices across developers and automation of the generation of configuration files and containers by parsing the source code of the Nextflow pipeline. The Geniac toolbox and two demo pipelines are available on GitHub. This article presents the main functionalities of Geniac.

Keywords
workflow management systems, containerization, reproducibility, high-performance computing, bioinformatics pipelines
Introduction

With the advent of high-throughput biotechnological platforms and their ever-growing capacity, life science has turned into a digitized, computational and data-intensive discipline. While genomics was the major driving force for high-throughput sequencing, other fields including proteomics, imaging and microscopy now contribute to this data explosion (Goh & Wong, 2020). Therefore, the bottleneck shifted from data generation to the ability to process and analyze data with efficient algorithms.

The data analysis actually encompasses two levels, corresponding to different time scales. The first level consists of the iterative research development of innovative and state-of-the-art analysis methods to address novel scientific questions. This level usually takes several months or years. In contrast, the second level consists of routine production analysis using mature and validated methods to apply standard analysis (including quality control) to the new samples processed by these high-throughput biotechnological platforms (Reiter et al., 2021). The production analysis must be performed within a short-delay (few hours up to few days) in order to process in real-time the huge data flow produced by these platforms, such that the relevant biological information can be extracted and provided to the end-users as fast as possible for their downstream analysis. This time constraint is even more important when the data are used in healthcare to support therapeutic decisions: in this case, quick parallel algorithms are required to process high-throughput sequencing data (Jarrier et al., 2020). Both levels feed each other in a virtuous circle.

Whatever the level, the analysis is generally a complex workflow (or pipeline) which involves many steps with different informatics languages and software, whether developed in-house or by third-parties. Coordinating the execution and allowing the scalability of these different steps to benefit from high-performance computing infrastructures to speed-up computation have been simplified with the development of workflow management systems (Leipzig, 2017; Strozzi et al., 2019) in combination with several technologies (Gruening et al., 2018; Tanjo et al., 2021) including package management systems with Conda or Bioconda (Grüning et al., 2018), and containers with Docker (Merkel, 2014) or Singularity (Kurtzer et al., 2017). Importantly, the use of workflow management systems also promotes reusable, reproducible and shareable analysis according to the FAIR principles (Wilkinson et al., 2016) which not only apply to data but also to workflows (Goble et al., 2020). Among the workflow management systems, Nextflow (Di Tommaso et al., 2017) became very popular and offers many interesting features (Jackson et al., 2021). It uses a dataflow programming model which implicitly defines the dependencies between the processes via their outputs and inputs such that different processes can be run in parallel and/or wait for each other when needed.

Whenever run for production analysis, the workflow has no choice but to deliver the results in time. If the workflow fails due to hardware issues or edge/corner cases due to the data themselves, then debugging, fixing, re-installing and resuming the workflow must be performed as fast as possible. This obviously requires harmonized guidelines and protocols across the different developers involved in the development life cycle of the pipeline. This way, it is easier to maintain many different pipelines on one hand and it simplifies the skill transfer from one person to another on the other hand. We developed the biogitflow protocols which cover the code versioning using git and GitLab (Kamoun et al., 2020). However, these protocols do not address how to write the code itself. While very valuable guidelines have been proposed to avoid pitfalls in the implementation of bioinformatics pipelines using the technological stack with workflows management systems, package management systems and containers (Gruening et al., 2018; Reiter et al., 2021; Tanjo et al., 2021), they remain very generic. Therefore, software engineering best practices and technical protocols for coding are necessary in order to promote software quality in bioinformatics to ensure the usability, maintainability, interoperability, sustainability, portability, reproducibility, scalability and efficiency of the workflows, as highlighted by several authors (da Veiga Leprevost et al., 2014; Georgeson et al., 2019; Lawlor & Walsh, 2015).

We acknowledge the nf-core initiative (Ewels et al., 2020) which plays a major role in promoting the software engineering best practices to implement bioinformatics pipelines with Nextflow including code template, linter, code reviewing and continuous integration for better quality. Our Bioinformatics Core Facility decided to capitalize on both Nextflow and the nf-core initiative to provide the end-users with bioinformatics pipelines in the context of production analysis. In this article, we describe additional guidelines on how to write the Nextflow code. We propose a set of best practices along the development life cycle of the pipeline and deployment for production operations which address different expert communities: i) the bioinformaticians and statisticians who prototype the pipeline with state-of-the-art methods in order to extract most of the hidden value from the data and provide the end-users with summary reports, ii) the software engineers who optimize the pipeline to reduce the amount of required informatics resources, to shorten...
the time to result delivery, iii) the data managers and core facility engineers who deploy and operate the pipeline for daily production analysis for the end-users. The guidelines were motivated by: i) being less invasive to the different expert communities, ii) reducing the overall development cycle from the prototyping stage to deployment in a production environment, iii) providing portable pipelines with containers (Docker and Singularity), and iv) automating (whenever possible) the building of containers from the source code of the pipeline. Therefore, we implemented Geniac (Automatic Configuration GENerator and Installer for nextflow pipelines) (Hupé et al., 2021a) to address these challenges. Geniac may be used alone for any pipeline built with Nextflow or as an extension of the nf-core guidelines to reduce the code complexity during the prototyping stage and standardize the deployment process.

Geniac actually consists of a toolbox with three components: technical documentation available at https://geniac.readthedocs.io, a linter to check that the code respects the guidelines, and an add-on to generate configuration files, build the containers and deploy the pipeline. We introduce here the main functionalities of the Geniac toolbox.

Methods
General principle
The ultimate goal of the proposed best practices of coding with Nextflow was to ensure the portability and reproducibility of the pipeline with containerization techniques (such as Docker and Singularity). Therefore, the rationale behind Geniac was motivated by the automation of the construction of the containers without explicitly writing the Dockerfile or the Singularity Definition File recipes (whenever possible). To do so, the pipeline source code is parsed by Geniac. The portability of the pipeline with containers can be achieved in two ways: either a single container including all the tools required by the pipeline is provided, or several containers (one container for each tool to be as granular as possible) are provided. We decided to retain this second way, as this one container - one tool strategy was strongly recommended by several experts (Gruening et al., 2018) who stated that Each container should encapsulate only one piece of software that performs a unique task with a well-defined goal (e.g. sequence aligner, mass spectra identification). Moreover, the one container - one tool strategy has the following advantages:

• most of the pipelines share tools in common meaning that, once a container is built for one tool, it can be easily reused in other pipelines,

• sometimes, different tools that might be incompatible with each other are required, making the usage of a single container including all the tools for the pipeline impossible,

• building a container with all the tools can be very long and possibly tedious. Each time the pipeline changes, the single container has to be rebuilt. With the one container - one tool strategy, only the container in which the tool has changed has to be updated (which is faster),

• the building of the containers can be parallelized to reduce the time in which the containers can be made available.

Generally, the building of the containers occurs in the very late stage of the development cycle of the pipeline, once it has been validated by the end-users and is ready to be deployed in production. Therefore, the guidelines also took into consideration that the pipeline could simply be run during its very early stage of prototyping by bioinformaticians and statisticians, without systematically building the containers whenever a new version of the tool is tested or a new tool is added. This challenge can be easily tackled with the Conda packaging system (Gruening et al., 2018) that offers great flexibility to bioinformaticians and statisticians in order to install and test new tools (when available in the different Conda channels, which is the case for most of the bioinformatics tools). For this reason, the Geniac toolbox allows for the possibility of running the pipeline with different Nextflow profiles (a profile is a set of configuration attributes that can be activated/chosen when launching a pipeline execution by using the -profile command line option with Nextflow) including:

• singularity to use the Singularity containers.

• docker to use the Docker containers.

• conda to use a single Conda environment on which all the tools are available.

• multiconda to use a dedicated Conda environment for each tool.
path to define the PATH environment variable for which all the tools are assumed to be installed in the
same directory.

multipath to define a dedicated PATH environment variable for which each tool is installed individually.

In addition to these profiles which set where the tools are accessible, we defined the cluster profile which
can be combined with the previous profiles so that Nextflow launches the computation on a high-performance
computing cluster.

As the Geniac toolbox automatically generates the configuration files used by these different profiles from
the source code, it ensures that they are consistent with each other. Moreover, it saves time for the devel-
oper who does not have to write them one by one. All these different profiles are made available with the
Geniac toolbox so that it covers pretty much all of the preferred work habits of a developer. However, we
strongly encourage the use of multiconda during prototyping and singularity for production. Of
note, both conda and path profiles may not work if the pipeline needs tools that are not compatible with
each other.

The one container - one tool strategy has to be translated in terms of coding best practices with Nextflow
so that the source code can be automatically parsed to generate these Nextflow configuration files and build
the containers. The coding relies mainly on the use of the label directive for each process which uses
only one tool and one container at a time, along with the withLabel process selector defined in each
configuration file for the different profiles which can be run with the -profile Nextflow option. Figure 1
illustrates this principle on two tools and three Nextflow processes.

Availability of the tool
When implementing a new tool in a pipeline, the main question to answer is Where is the tool availa-
ble?: Depending on the answer, the developer has to follow specific guidelines that are fully detailed in the
Geniac documentation. Whenever possible, it is recommended to use the tool from the Conda packaging
system (Gruening et al., 2018). However, not all tools are available in Conda. Therefore, we guide the
developer through a series of chronological questions that redirect the developer to the appropriate
section whenever an answer is Yes. The questions are the following: i) Is it just a standard Unix com-
mand?, ii) Is it available in Conda?, iii) Is it available only as a binary or as an executable script?,
v) Is the source code available?, v) Have you still not answered yes?. Briefly, the developer has to
perform some actions such as modify the conf/geniac.config file, copy the source code of the tool in

\[\text{Figure 1. one container - one tool strategy and coding of the Nextflow pipeline: each tool that is used by the
pipeline has its own container.} \text{ In the different configuration files which are specific to a Nextflow profile, each tool is
declared with the withLabel process selector. To use the appropriate tool in a Nextflow process, the use of the label
directive with the name of the tool makes the link. The same tool can be used in different processes when different
tasks have to be performed at different stages or with different options.}\]
the `modules` folder, write some recipes in the folder `recipes`, copy some files in the `bin` or `recipes/dependencies` folders, depending on the answer to the questions. In most of the cases, the tool should be available in Conda which is the most comfortable situation as it allows the automation of the creation of the containers and the configuration files for Nextflow profiles. However, the toolbox is able to tackle any other situations, although this requires more configuration on the developer’s side. Figure 2 describes the overall workflow with Geniac.

Implementation

It is important to note that the Geniac toolbox (Hupé et al., 2021a) relies on the structure of the Geniac Template (Servant & Hupé, 2021) while the latter could work without Geniac (provided that the missing configuration files for the different Nextflow profiles are added manually). It also means that the Geniac documentation explains how the Geniac Template works with the Geniac toolbox. Therefore, the source code must be organized as shown in Figure 3A with some mandatory files from the Geniac Template in blue and the Geniac toolbox itself in the eponymous folder in green. Based on this organization, the automatic generation of the Nextflow configuration files, container recipes and images are performed by some Nextflow scripts (`geniac/install/singularity.nf` and `geniac/install/docker.nf`) which are invoked by Cmake scripts. The file `nextflow.config` includes all the configuration files that are automatically generated by Geniac to define the different Nextflow profiles. The main Cmake script is the file `geniac/CMakeLists.txt` which invokes a series of functions from the `geniac/cmake` folder. The command-line interface for the linter corresponds to a standalone python package within the `geniac/src` folder. The linter offers the possibility for the developer to check the code’s compliance with the guidelines. The command includes the following tests: i) existence of mandatory or optional files and folders in the Nextflow project tree structure, ii) configuration and consistency of labels with process directives in the Nextflow pipeline, iii) validation of the geniac configuration file, iv) availability of tools and their dependencies. Similarly to the nf-core linter, the output can be exploited manually or automatically within a continuous integration pipeline since any critical error reported by the linter changes the exit code from 0 to 1.

As it will be illustrated in the Use Cases section, the file `conf/geniac.config` is essential as it allows registration of the tools available from Conda. This file also defines important options to ensure reproducibility with Singularity. Indeed, the default behavior of Singularity makes it possible to access the user’s $HOME directory from the host on which the process is executed. Some programming languages such as python or R preemptively load libraries from the user $HOME if they exist rather than the libraries from the canonical installation. This is therefore a major issue that can be tackled by defining in the file `conf/geniac.config`
the option `singularityRunOptions = "-B "$PWD":/tmp --containall"` which is passed on the Singularity command line when invoked by Nextflow. The `--containall` sets some directories as empty (including `$HOME` and `/tmp`) thus avoiding possible libraries installed in the user’s SHOME to be loaded by these programming language. The `-B` option allowing the binding of the `/tmp` (inside the container) to the work directory of the Nextflow process (i.e. in the folder like `work/2d/8202294` created by Nextflow during the execution) is also necessary. Indeed, the default overlay size of `/tmp` inside the Singularity image is few MB that makes the writing of temporary files required by many tools impossible due to space limitation. We preferred to bind `/tmp` to the work directory of the Nextflow process instead of on the actual `/tmp` of the host since it might use some RAM (tmpfs) that is not reported to the job scheduler (such as slurm or PBS). As we defined the `singularityRunOptions`, it isolates the SHOME from the host whatever the programming language used inside a Nextflow process that makes this solution universal. However, this solves this issue only for the singularity profile. In the case of the conda or multiconda profiles, we addressed this issue by adding ad-hoc environment variables using the `beforeScript` directive `beforeScript = "export R_LIBS_USER="/; export PYTHONNOUSERSITE=1"` in the files which are automatically generated by Geniac. If any other programming language suffers from the same behavior, ad-hoc solutions will have to be added. Note that the default options when using Docker containers within Nextflow avoid access to SHOME, this profile is therefore not concerned by the reproducibility issue.

Operation
The Geniac toolbox is intended to run on a Linux distribution with Nextflow (>= 20.01.0), git (>= 2.0), Cmake (>= 3.0), Make (>= 4.1) and Conda (>= 4.10.1). To use the containers, either Singularity (>= 3.2) or Docker (>= 18.0) are needed. The Geniac linter requires python (>= 3.9).
Use cases

In this section, several use cases are described. They are available in the bash script `data/useCases.bash` from the Geniac repository so that the reader can reproduce them step-by-step. The use cases depend on each other meaning that they must be sequentially performed from the beginning. They also require the installation of Nextflow (>= 20.01.0), git (>= 2.0), Cmake (>= 3.0), Make (>= 4.1), Conda (>= 4.10.1) and Singularity. The conducting line of this section is the deployment of a pipeline to be run in routine production with Singularity on a high-performance computing cluster having slurm as a job-scheduler. A comprehensive overview of the Geniac functionalities is available in the Geniac documentation.

Add a Nextflow process

We explain here how to add a new process in the pipeline when the tool is available in Conda. First, the source code from the Geniac Demo (Hupé et al., 2021b) repository can be downloaded as follows:

```
export WORK_DIR="${HOME}/tmp/myPipeline"
export SRC_DIR="${WORK_DIR}/src"
export INSTALL_DIR="${WORK_DIR}/install"
export BUILD_DIR="${WORK_DIR}/build"
export GIT_URL="https://github.com/bioinfo-pf-curie/geniac-demo.git"

mkdir -p ${INSTALL_DIR} ${BUILD_DIR}

# clone the repository
# the option --recursive is needed if you use geniac as a submodule
git clone --recursive $GIT_URL ${SRC_DIR}
```

Let’s assume that the tool `multiqc` (Ewels et al., 2016) must be added in the pipeline. The section `params.geniac.tools` of the file `${SRC_DIR}/conf/geniac.config` must contain the following line:

```
multiqc = "conda-forge::lzstring=1.0.4=py_1001
        conda-forge::matplotlib-base=3.1.1=py37h250f245_2
        conda-forge::spectra=0.0.11=py_1 bioconda::multiqc=1.8=py_2"
```

The syntax follows the pattern from the conda package naming convention, so that the version is explicit as recommended in some guidelines (Gruening et al., 2018):

```
toolName = "condaChannelName::softName=version=buildString"
```

If other Conda dependencies are required, they are added according to the same convention as shown for multiqc. In order to use the tool in a Nextflow process, use the label directive using the exact same name as given in the params.geniac.tools section. For example, the process `multiqc` in the file `${SRC_DIR}/main.nf` contains the label `multiqc` defined in the file `${SRC_DIR}/conf/geniac.config`:

```
process multiqc {
    label 'multiqc'

    // write your nextflow code.
}
```

Check the code with the linter

We recommend installing the Geniac command-line interface using the package installer for python (pip) in a separate environment with Conda as follows:

```
conda create -n geniac-cli python=3.9
conda activate geniac-cli
pip install git+https://github.com/bioinfo-pf-curie/geniac.git@release
```
Run the linter on your repository:

```bash
geniac lint ${SRC_DIR}
```

**Build the Singularity containers**

It consists of two main steps. First, the pipeline is configured with the required `cmake` options. The options with the pattern `-DCMAKE_<UPPER_CASE>=<value>` correspond to the variables available in the `cmake` language (CMAKE_INSTALL_PREFIX defines where the analysis pipeline will be deployed). The options with the pattern `-Dap_<lower_case>=<value>` correspond to the ad-hoc variables defined by the Geniac toolbox (see the Geniac documentation for the list of available options). In this use case, the option `-Dap_install_singularity_images=ON` tells Geniac to build and install the Singularity images and `-Dap_nf_executor=slurm` to set slurm as the executor (i.e. the job-scheduler) in the cluster profile used by Nextflow. Secondly, it starts the build of the pipeline with the `make` command. These two steps work as follows:

```bash
cd ${BUILD_DIR}

### configure the pipeline
`cmake` ${SRC_DIR}/geniac -DCMAKE_INSTALL_PREFIX=${INSTALL_DIR}
    ↪ -Dap_install_singularity_images=ON -Dap_nf_executor=slurm

### with sudo, the singularity and nextflow commands must be
### in the secure_path option declared in the file /etc/sudoers

### build the files needed by the pipeline
`sudo make`

### change file owner/group to the current user
`sudo chown -R $(id -gn):$(id -gn) ${BUILD_DIR}
```

During the build process, Geniac generates the different configuration files, the Singularity recipes and images in the `${BUILD_DIR}/workDir` folder as shown in Figure 3B. Note that these files are not present in the `${SRC_DIR}` as they are automatically generated. The root credentials are needed to build the Singularity images thus requiring the use of the `sudo` commands.

**Deploy the pipeline**

The deployment of the pipeline is performed with the `make` command:

```bash
make install
```

After the deployment, the pipeline is available in the `${INSTALL_DIR}/pipeline` folder in which the `main.nf` Nextflow file is available (see Figure 3C). Other folders are also created which contains the Singularity images, a folder (or a symlink depending on the option used during the configuration) for the genome annotations, and folders for the path and multipath profiles.

**Run the pipeline**

The pipeline can be run with the `singularity` profile locally on the computer as follows:

```bash
cd ${INSTALL_DIR}/pipeline
nextflow -c conf/test.config run main.nf -profile singularity
```

The pipeline can be run with the `singularity` and `cluster` profiles on a high-performance computing cluster with the slurm job-scheduler as follows:

```bash
nextflow -c conf/test.config run main.nf -profile singularity,cluster
```
Compatibility with Nextflow DSL2

Geniac is fully compatible with Nextflow DSL2 provided that the process, workflows and subworkflows are organized in the `${SRC_DIR}/nf-modules` folder as shown in red in Figure 3A. The Geniac Demo DSL2 (La Rosa et al., 2021) repository can be used instead of the Geniac Demo. To do so just set, at the very beginning of the Use cases section, the value of GIT_URL to:

```
export GIT_URL="https://github.com/bioinfo-pf-curie/geniac-demo-dsl2.git"
```

Conclusions

Geniac consists of a toolbox with three components: a technical documentation available at https://geniac.readthedocs.io, a linter to check that the code respects the guidelines, and an add-on to generate configuration files, build the container and deploy the pipeline. The Geniac Demo and Geniac Demo DSL2 pipelines are available so that the reader can practice the Geniac guidelines. The Geniac Template makes it possible to start a new pipeline from scratch. The Geniac toolbox is fully compatible with Nextflow DSL1 and DSL2. While it supports the automatic generation of Docker and Singularity containers, it is straightforward to add any other containerization framework such as podman or shifter. The development of the Geniac toolbox was strongly motivated by the context of production analysis. However, we strongly encourage the use of Geniac in the context of the research development analysis.

Data availability

All data underlying the results are available as part of the article and no additional source data are required.

Software availability

- Source code for geniac available from: https://github.com/bioinfo-pf-curie/geniac
- Archived source code at time of publication: https://doi.org/10.5281/zenodo.4883661 (Hupé et al., 2021a)
- Source code for geniac-demo available from: https://github.com/bioinfo-pf-curie/geniac-demo
- Archived source code at time of publication: https://doi.org/10.5281/zenodo.4884250 (Hupé et al., 2021b)
- Archived source code at time of publication: https://doi.org/10.5281/zenodo.4883720 (La Rosa et al., 2021)
- Source code for geniac-template available from: https://github.com/bioinfo-pf-curie/geniac-template
- Archived source code at time of publication: https://doi.org/10.5281/zenodo.4883775 (Servant & Hupé, 2021)
- All documentation is available at: https://geniac.readthedocs.io
- License: CeCILL Version 2.1

Author contributions

J.R. and P.H. conceived the coding best practices and automation principles proposed by Geniac. P.H. developed Geniac and Geniac Demo. P.L.R. developed Geniac Demo DSL2. N.S. and P.H. developed the Geniac Template. F.A. developed the Geniac linter. F.J. tested the different demo pipelines. P.H. wrote the manuscript and the documentation. P.H. supervised the study.

Acknowledgements

We are grateful to our colleagues from the Bioinformatics Core Facility for their feedback on the Geniac toolbox and from the ICT department.
References

PubMed Abstract | Publisher Full Text | Free Full Text
PubMed Abstract | Publisher Full Text
PubMed Abstract | Publisher Full Text | Free Full Text
PubMed Abstract | Publisher Full Text
PubMed Abstract | Publisher Full Text | Free Full Text
PubMed Abstract | Publisher Full Text | Free Full Text
PubMed Abstract | Publisher Full Text | Free Full Text
PubMed Abstract | Publisher Full Text | Free Full Text
PubMed Abstract | Publisher Full Text | Free Full Text
PubMed Abstract | Publisher Full Text | Free Full Text
PubMed Abstract | Publisher Full Text | Free Full Text
PubMed Abstract | Publisher Full Text | Free Full Text

Reference Source

PubMed Abstract | Publisher Full Text
PubMed Abstract | Publisher Full Text
PubMed Abstract | Publisher Full Text | Free Full Text
PubMed Abstract | Publisher Full Text | Free Full Text
Open Peer Review

Current Peer Review Status: ✔️ ❓

Version 1

Reviewer Report 31 August 2021

https://doi.org/10.21956/openreseurope.14944.r27224

© 2021 Lemoine F. This is an open access peer review report distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Frédéric Lemoine
Unité de Bioinformatique Evolutive, USR 3756 (DBC/C3BI), CNRS, Institut Pasteur, Paris, France

This manuscript describes Geniac, a toolbox aiming at increasing the reproducibility and reusability of Nextflow workflows. To do so, Geniac has the following features:

1. It defines a set of good practices for the development of Nextflow workflows, via a detailed documentation.
2. It provides a linter to check that the best coding practices are respected.
3. It provides a "add-on" that automatically builds dependencies, generates the singularity (or docker) containers needed to run the workflow, checks the configuration files, and installs the workflow in the specified directory ("deploy").

Geniac tackles an important issue with bioinformatic analyses: Pipelines are rarely homogeneous in terms of description and implementation, which leads to reproducibility, maintenance, and reusability difficulties. In terms of goals, Geniac present some similarities with nf-core, which aims at curating and harmonizing pipeline implementations. The solution proposed by Geniac is to define of a set of homogeneous practices for workflow development (e.g. how to describe process dependencies, how to name variables, channels and files, and provide tools to check workflows, build containers and install the workflow). The price to pay for more reproducible and homogeneous workflows is a more complex technological stack (with CMake specifically) and workflow architecture. Technically, Geniac is based on the following technologies: Nextflow, conda, Docker (or Singularity) and CMake. Although technical, the manuscript explains clearly the philosophy, the goals, the usage and the usefulness of Geniac. The tools are usable in the provided workflow example, I was able to run the geniac-demo and the linter works well and is easy to use.

I believe that Geniac is a useful tool that answers important questions. However, I think it would be easier to understand and use with a few clarifications in my opinion:

- I had some initial difficulties to run the geniac-demo workflow. I think the documentation (on readthedoc and on the git repositories) could be clearer about how to run the workflow in different situations, at least with docker or singularity. For example, I initially had an error...
when the install and the build directories were included in the geniac-demo directory. Before understanding the general philosophy of Geniac, I also had some difficulties to understand how to generate the singularity.config file needed to generate singularity images and run the workflow. A more visible step by step concrete quick-start example could be very useful.

○ Following the previous point, I was a bit confused by the README in the geniac-demo and geniac-template repositories, where I was looking for information on how to run the example. The documentation could be better adapted to the workflows. For example the geniac.md file refers to nf-CRISPR repository.

○ Testing the workflow is also generally a difficult part of the development and maintenance of workflows. It is not mentioned clearly in the introduction or in the discussion, though I think that tests can be performed using test profiles in Geniac. I think it would be great to give a little bit more insights on what tests consist of in geniac workflows.

○ The abstract states that workflow systems + containers increase usage complexity. I am not sure that Geniac tackles this problem, since it adds a new layer of complexity in the top of Nextflow, Docker, etc (which is unavoidable). If I understand correctly, Geniac is mainly solving the problem of workflow homogeneity in terms of structure, description, and documentation, and thus make the workflow more reproducible, maintainable and reusable.

○ A more detailed description of the specificities of Geniac compared to nf-core may be a great addition to the manuscript.

○ Are there already any public real-world Geniac compatible workflows in production? If so, it could be interesting to mention a few of them.

A few typos/suggestions:
○ Abstract+Introduction: "which address" => "which target"?

○ Introduction: "the workflow has no choice but deliver": Isn't it more the facility rather than the workflow that has no choice?

○ Introduction: "being less invasive to the different expert communities": This point is not very clear to me. Would it be possible to rephrase it?

○ Methods: "to be as granular as possible", maybe clearer with "to obtain the finest level of granularity"?

○ Methods: "The building of the containers can be parallelized to reduce the time in which...". It is not very clear to me. Would something like "to speed up the deployment of new containers" make sense?

○ The DOI link in the ref Servant & Hupé 2021 does not work (there is an additional space in the URL).

○ Methods: "Other folders are also created which contains"=>"contain".
Is the rationale for developing the new software tool clearly explained?
Yes

Is the description of the software tool technically sound?
Partly

Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?
Partly

Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?
Yes

**Competing Interests:** No competing interests were disclosed.

**Reviewer Expertise:** I work in the field of evolutionary bioinformatics where I develop tools and workflows. I am interested in the reproducibility of bioinformatics analyses, and I use Nextflow and container technologies on a daily basis.

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard, however I have significant reservations, as outlined above.

---

© 2021 Hermoso Pulido T. This is an open access peer review report distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

---

**Toni Hermoso Pulido**
Centre for Genomic Regulation, The Barcelona Institute of Science and Technology, Barcelona, Spain

In this article, the Geniac toolbox is presented. The software consists of several tools to address some common difficulties of deploying and adapting Nextflow pipelines to fit into different computing environments. Its major emphasis is ensuring both code reproducibility and production-level reliability. Technically speaking, it depends on Nextflow itself and CMake automation and packaging tool. A Python-based code analysis (linter) tool is also provided for checking pipeline setup in advance before the actual build and deployment process takes place.

Provided software for the pipelines relies primarily on Conda package manager, which Geniac facilitates to adapt to different approaches (e.g., single or multi), and converts to handy container
images such as Singularity-based ones.

Experienced practitioners familiar with Nextflow, Conda and container technologies can easily follow the instructions and customize the provided examples (which are based on both legacy and DSL2 Nextflow syntax). Maybe the final result and associated folders of the pipeline preparation (build, install), as depicted in Figure 3, could be more clearly stated.

Despite it being mentioned in the article and in the toolbox documentation as an actual possibility, a more explicit example case of a software step that does not rely on Conda could be a nice addition.

As a user experience suggestion for future releases, it could be interesting that the geniac executable could be already provided straight from the Python Package Index (PyPI). It would also help new adopters, specially if not so familiar with CMake, if geniac could wrap some of the execution calls that are included in the useCases.bash script.

**Is the rationale for developing the new software tool clearly explained?**
Yes

**Is the description of the software tool technically sound?**
Yes

**Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?**
Yes

**Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?**
Partly

**Competing Interests:** No competing interests were disclosed.

**Reviewer Expertise:** I work in Bioinformatics. I have experience developing and deploying Nextflow pipelines, and I regularly take care of infrastructure and devops-related tasks.

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.