


INCREASING THE REPRODUCIBILITY OF SCIENTIFIC RESEARCH WORKS: A CASE STUDY USING THE ENVIRONMENT CODE-FIRST FRAMEWORK

Daniel Adorno Gomes^A, Pedro Mestre^B, Carlos Seródio^C



ARTICLE INFO	ABSTRACT
<p>Article history: Received: February, 13th 2024 Accepted: May, 03rd 2024</p>	<p>Objective: The purpose of this paper is to present a case study on how a recently proposed reproducibility framework named Environment Code-First (ECF) based on the Infrastructure-as-Code approach can improve the implementation and reproduction of computing environments by reducing complexity and manual intervention.</p>
<p>Keywords: Infrastructure-as-Code; Reproducibility; Virtualization; Containerization; Open Science.</p> 	<p>Methodology: The study compares the manual way of implementing a pipeline and the automated method proposed by the ECF framework, showing real metrics regarding time consumption, efforts, manual intervention, and platform agnosticism. It details the steps needed to implement the computational environment of a bioinformatics pipeline named MetaWorks from the perspective of the scientist who owns the research work. Also, we present the steps taken to recreate the environment from the point of view of one who wants to reproduce the published results of a research work.</p> <p>Findings and Conclusion: The results demonstrate considerable benefits in adopting the ECF framework, particularly in maintaining the same applicational behavior across different machines. Such empirical evidence underscores the significance of reducing manual intervention, as it ensures the consistent recreation of the environment as many times as needed, especially by non-original researchers.</p> <p>Originality/Value: Verifying published findings in bioinformatics through independent validation is challenging, mainly when accounting for differences in software and hardware to recreate computational environments. Reproducing a computational environment that closely mimics the original proves intricate and demands a significant investment of time. This study contributes to educate and assist researchers in enhancing the reproducibility of their work by creating self-contained computational environments that are highly reproducible, isolated, portable, and platform-agnostic.</p> <p>Doi: https://doi.org/10.26668/businessreview/2024.v9i5.4662</p>

^A Master in Network Telecommunication Management. University of Trás-os-Montes and Alto Douro. Vila Real, Vila Real, Portugal. E-mail: adornogomes@gmail.com Orcid: <https://orcid.org/0000-0002-1306-6733>

^B PhD in Electrical & Electronic Engineering. University of Trás-os-Montes and Alto Douro and Centre for the Research and Technology of Agro-Environmental and Biological Sciences. Vila Real, Vila Real, Portugal. E-mail: pmestre@utad.pt Orcid: <https://orcid.org/0000-0002-0445-7935>

^C PhD in Electrical & Electronic Engineering. University of Trás-os-Montes and Alto Douro. Vila Real, Vila Real, Portugal, and Centro Algoritmi - University of Minho, Guimarães, Braga, Portugal. E-mail: cserodio@utad.pt Orcid: <https://orcid.org/0000-0003-4632-9664>

AUMENTANDO A REPRODUTIBILIDADE DE TRABALHOS DE PESQUISA CIENTÍFICA: UM ESTUDO DE CASO UTILIZANDO O FRAMEWORK ENVIRONMENT CODE-FIRST**RESUMO**

Objetivo: O objetivo deste artigo é apresentar um estudo de caso sobre como um framework de reprodutibilidade proposto recentemente, denominado Environment Code-First (ECF) e baseado na abordagem Infraestrutura-como-Código, pode melhorar a implementação e reprodução de ambientes computacionais, reduzindo a complexidade e a intervenção manual.

Metodologia: O estudo compara a forma manual de implementação de um pipeline e o método automatizado proposto pelo framework ECF, mostrando métricas reais quanto ao consumo de tempo, esforços, intervenção manual e agnosticismo da plataforma. São detalhadas as etapas necessárias para implementar o ambiente computacional de um pipeline de bioinformática denominado MetaWorks na perspectiva do cientista proprietário do trabalho de pesquisa. Além disso, apresentamos os passos necessários para recriar o ambiente do ponto-de-vista de quem deseja reproduzir os resultados publicados de um trabalho de pesquisa, ou seja, dos cientistas não-originais.

Resultados e Discussão: Os resultados demonstram benefícios consideráveis na adoção do framework ECF, particularmente na manutenção do mesmo comportamento aplicativo em diferentes máquinas. Tais evidências empíricas ressaltam a importância da redução da intervenção manual, pois garantem a recriação consistente do ambiente quantas vezes forem necessárias, especialmente por pesquisadores não-originais.

Originalidade/Valor: Verificar as descobertas publicadas em bioinformática por meio de validação independente é um desafio, principalmente quando se leva em conta diferenças em software e hardware para recriar ambientes computacionais. Reproduzir um ambiente computacional que se assemelhe de perto com o original é complexo e exige um investimento significativo de tempo. Este estudo contribui para educar e auxiliar os pesquisadores a melhorarem a reprodutibilidade de seus trabalhos, criando ambientes computacionais independentes que são altamente reprodutíveis, isolados, portáteis e independentes de plataforma.

Keywords: Infraestrutura-Como-Código, Reprodutibilidade, Virtualização, Containerização, Ciência Aberta.

AUMENTAR LA REPRODUCIBILIDAD DEL TRABAJO DE INVESTIGACIÓN CIENTÍFICA: UN ESTUDIO DE CASO UTILIZANDO EL FRAMEWORK ENVIRONMENT CODE-FIRST**RESUMEN**

Objetivo: El objetivo de este artículo es presentar un estudio de caso sobre cómo un framework de reprodutibilidad propuesto recientemente, llamado Environment Code-First (ECF) y basado en el enfoque Infraestructura-como-código, puede mejorar la implementación y reproducción de entornos informáticos, reduciendo la complejidad, e intervención manual.

Metodología: El estudio compara la forma manual de implementar el pipeline y el método automatizado propuesto por el framework ECF, mostrando métricas reales en cuanto a consumo de tiempo, esfuerzos, intervención manual y agnosticismo de la plataforma. Además, detalla los pasos necesarios para implementar el entorno computacional de un pipeline bioinformático llamado MetaWorks desde la perspectiva del científico propietario del trabajo de investigación. Además, presentamos los pasos dados para recrear el entorno desde el punto de vista de alguien que quiere reproducir los resultados publicados de un trabajo de investigación.

Resultados y Discusión: El estudio compara la forma manual de implementar un pipeline y el método automatizado propuesto por el framework ECF, mostrando métricas reales en cuanto a consumo de tiempo, energía, intervención manual y agnosticismo de la plataforma. Los pasos necesarios para implementar el entorno computacional de un pipeline bioinformático llamado MetaWorks se detallan desde la perspectiva del científico propietario del trabajo de investigación. Además, presentamos los pasos necesarios para recrear el entorno desde el punto de vista de quienes deseen reproducir los resultados publicados de trabajos de investigación, es decir, de científicos no originales.

Originalidad/Valor: Verificar los descubrimientos publicados en bioinformática mediante validación independiente es un desafío, especialmente cuando se tienen en cuenta las diferencias en el software y el hardware para recrear entornos informáticos. Reproducir un entorno informático que se parezca mucho al original es complejo y requiere una importante inversión de tiempo. Este estudio contribuye a educar y ayudar a los investigadores a mejorar la reprodutibilidad de su trabajo mediante la creación de entornos informáticos independientes que sean altamente reproducibles, aislados, portátiles e independientes de la plataforma.

Palabras clave: Infraestructura-Como-Código, Reprodutibilidad, Virtualización, Contenerización, Ciencia Abierta.

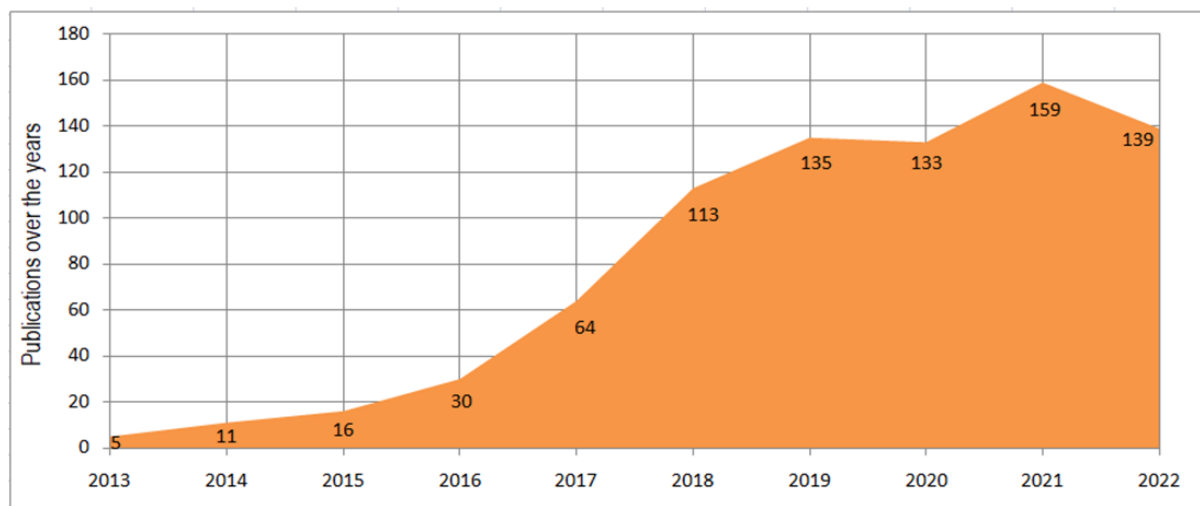
1 INTRODUCTION

The use of computation is essential for modern scientific research. Simulators, for example, play a crucial role in various fields such as chemistry, physics, biology, and numerous other domains, enabling silico experimentation. Simulations offer significant advantages in terms of cost and speed compared to conducting experiments on actual molecules, for example. In numerous endeavors, computational resources are valuable and essential, particularly when the volume of data surpasses human capacity for timely processing (Coveney et al., 2021; de Bayser et al., 2015).

However, the more science depends on computational means, the more the need to create scientific works that are more easily reproducible increases, mainly, from the point of view of one that is trying to recreate the final results published by others. Also, a mounting apprehension has emerged within the scientific community regarding unverifiable results that lack reproducibility (Reinecke et al., 2022). Currently, specialists consider this dependency to be one of the primary factors contributing to the crisis of scientific reproducibility. Figure 1 illustrates a noticeable surge in interest regarding the reproducibility crisis in science, primarily over the past five years. The chart, sourced from the Web of Science, exhibits the number of research works related to this subject published in the last decade.

Figure 1

Number of Web of Science publications that contain in the title, abstract, or keywords one of the following terms: "reproducibility crisis", "scientific crisis", "science in crisis", "crisis in science", "replication crisis", "replicability crisis". The search was executed considering the period from 2013 and 2022.



A growing consensus underscores the significance of reproducing research findings to enhance comprehension of conveyed concepts and facilitate their continual advancement (Cacho & Taghva, 2018).

Computational reproducibility is essential to validate the credibility of scientific papers and their results. The ability to revisit and reproduce past experiments plays a vital role in the scientific method. Scientists depend on effectively handling experiment-related data to interpret outcomes, maintain adherence to accepted protocols, and verify findings (Liu & Salganik, 2019). In education, reproducibility holds immense value, as the swift advancement of scientific knowledge increases the amount of information students need to comprehend. By repeating experiments, students can learn by scrutinizing the origin details of the original investigation, reassessing its inquiries, and expanding upon the results obtained in the initial study (Cacho & Taghva, 2020).

Reproducibility depends on open data, code, and extensive documentation that will permit to recreate of the entire software development environment (Barba & Thiruvathukal, 2017). However, regarding the environment, the most important item is the documentation. Rebuilding the same computational environment in which the original experiment was conducted is a challenging and time-consuming task, when documentation is available. The absence of the documentation makes the reproducibility of a research work almost impossible. Other important issues that can be highlighted are the differences in software and hardware platforms and lack of the correct version of dependencies (e.g., libraries, packages, third-party software) or even its absence (Grüning et al., 2019).

The use of researchers made their personal computers when installing and configuring the environment, and the practice of provisioning the resources manually, are factors contributing to and aggravating the scenario of irreproducibility (Segal & Morris, 2012), most of the time producing heterogeneous environments.

In the software industry various environments are commonly used during the development of systems information namely development, testing, staging and production. According to the Twelve-Factor App methodology, one best practice in software engineering is to keep the different environments where the application will be developed and run as similar as possible regarding the technical aspects. This similarity between the environments ensures applications have the same behavior in any of them generating always the same results (Wiggins, 2017). Essentially, the techniques used to provisioning homogeneous environments must be identical (Humble & Farley,

2010). This best practice also must be applied for scientific applications, mainly when considering the reproducibility of research works by non-original researchers.

Nowadays, especially for researchers starting their research works using their machines alone or in small teams, the Linux containers technology is one of the essential reproducible tools helping provision more homogeneous computational environments (Marwick, 2017; Wiebels & Moreau, 2021). This technology allows all dependencies of the computational application, such as libraries, packages, compilers, interpreters, databases, and their respective versions, to be specified and configured programmatically so that the environment is reproduced exactly as specified. However, this technology only allows us to programmatically specify the dependencies that support the scientific application, in other words, the applicational environment. Therefore, the infrastructure necessary to support the containerized environment (e.g., container engine) must be installed and configured manually. Manual intervention leads researchers to face issues when provisioning their environments, mainly in non-Linux platforms such as Microsoft Windows and Apple MacOS (Docker, 2024a; Docker, 2024b; Docker, 2024c).

The infrastructure-as-code approach has been used to address this kind of solution to provisioning the entire computational environment, infrastructure and containerized application, programmatically as source code through tools such as Terraform, vagrant, Ansible, Chef, and Puppet. Defining the entire environment as code permit us to produce more homogeneous computational environments reducing the level of manual intervention to a minimal and, consequently, to increase the reproducibility of the research work. Besides of these factors, treating the environment as a software system provides us the possibility to store and versioning it in a code repository (e.g., Github or Gitlab), to test it improving its quality, reproducing an identical, consistent and reliable environment every time, as many times as needed.

In this paper, the authors present a case study of implementing the computational environment of a bioinformatics pipeline named MetaWorks (Porter & Hajibabaei, 2022) following the Environment Code-First (ECF) framework (Gomes et al., 2022). Currently, MetaWorks is limited by running only on Linux platforms and its implementation requires a high level of manual intervention. The use case presents details on how to follow the framework. It demonstrates how the ECF framework can enhance the reproducibility and transparency of scientific research, making the computational environment readily available. It reduces the obstacles for others to effortlessly replicate published experiments across multiple platforms with minimal manual intervention, eliminating the need to speculate about the processes followed by the original authors.

2 THEORETICAL BASIS AND RESEARCH OVERVIEW

2.1 THE METAWORKS PIPELINE

Metaworks (Porter & Hajibabaei, 2022) is a versatile bioinformatics pipeline designed to process demultiplexed Illumina paired-end reads such as SeqPrep (St John, 2016), CutAdapt (Martin, 2011), VSEARCH (Edgar, 2016), and the RDP Classifier (Wang et al., 2007). It uses Conda (Conda, 2024) package manager to control the programs and the dependencies that compose the environment and Snakemake (Snakemake, 2024) workflow manager to automate pipelines and utilize computational resources efficiently. It is free and open-source software licensed under GPLv3. The third-party software packages used to it are open-source as well. The source code is available on Github (MetaWorks, 2024a). The software comes with a small set of raw data, and the step-by-step tutorial (MetaWorks, 2024b) can guide new users on gain experience.

It was designed to be more reproducible, automatizing the creation of the pipelines, and more scalable, permitting improved performance by increasing the processing power, for example, moving from a PC to the cloud. However, a high level of manual intervention is necessary when installing and configuring it. Another limitation of the pipeline, according to the official documentation, is the fact that MetaWorks runs at the command line only on the Linux-64 platform.

2.2 THE ENVIRONMENT CODE-FIRST FRAMEWORK

The Environment Code-First framework (ECF) has its foundation on the Infra-structure-as-Code (IaC) approach (Gomes et al., 2019; Morris, 2020), and its main goal is to guide researchers on implementing self-contained computational environments more reproducible, isolated, portable, and independent of any platform of software and hardware. The environments are programmatically defined as source code, permitting them to be treated as software systems and recreated as often as needed.

The framework's objective is to steer the creation of an environment that embodies the following characteristics:

- a) independence from specific hardware and software platforms, irrespective of operating systems;

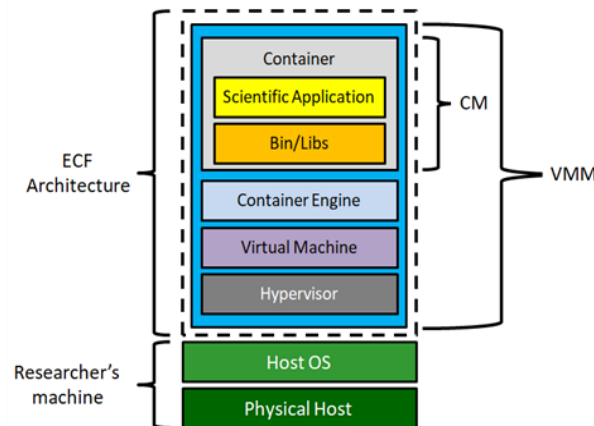
- b) versatility to operate on-premise, whether on personal computers, robust servers, or even within cloud setups;
- c) full programmable provisioning, eliminating manual installations and configurations. Source code resides in a repository (e.g., Github or GitLab);
- d) dynamic software resource management, enabling swift addition or removal directly from the environment's source code.

The framework comprises two components. The initial component outlines the architecture of the computational environment, while the second part offers a systematic guide delineating the researcher's step-by-step procedure for establishing and upholding the infrastructure.

2.2.1 The Environment Code-First Architecture

The primary objective of the architecture established by the ECF is to establish a consistent environment that remains unaffected by the hardware and software platforms employed by researchers.

The ECF introduces an architecture comprising two modules, designed to ensure consistent computational environments when replicating research endeavors. The first module, known as the Container Module (CM), is a Linux container encompassing all the necessary software, libraries, and packages necessary to develop and run a scientific application. The second module, referred to as the Virtual Machine Module (VMM), includes a hypervisor (e.g., Oracle Virtualbox, VMware), a lightweight virtual machine based on a Linux distribution, and a container engine (e.g., Docker, Podman). During the development phase, these modules are developed independently. However, in the execution phase, the CM operates within the VMM, overlaying the container engine layer. In essence, the CM functions as an additional layer of the VMM. As depicted in Figure 2, the green layers represent the physical machine and its installed operating system, while the other layers enclosed by dotted lines constitute the architecture specified by the ECF framework.

Figure 2*The ECF architecture.*

Source: Extracted from (Gomes et al., 2022).

Provisioning of both modules must be carried out programmatically using Infra-structure-as-Code (IaC) resources. Setting up the computational environment in alignment with the ECF architecture guarantees that the container consistently runs always on the same operating system, irrespective of the software platform used by researchers on their physical machines.

2.2.2 The Environment Code-First Guidance

The primary aim of ECF guidance is to assist researchers in implementing the two modules specified within the ECF architecture: the CM and the VMM. The CM takes precedence as it constitutes one of the four layers within the VMM. For both modules, the framework delineates a structured series of steps for researchers to follow, ensuring the successful implementation of each.

The initial stage in establishing a computational environment involves constructing the Container Module (CM) by executing the following steps:

- a) requirements identification;
- b) development of the CM source code;
- c) source code storage;
- d) container image generation;
- e) container image storage.

During the requirements identification step, researchers are tasked with pinpointing all the necessary software, libraries, and packages for developing and running the scientific application. The first imperative requirement in this phase is the specification of the container

engine. Determining the container engine is a prerequisite that precedes all other requirements, as it dictates the source code structure that researchers will create to define installations and configurations for environment creation. This source code must adhere to the patterns and syntax prescribed by the chosen engine. Other requirements may vary depending on the specific needs of each experiment's environment. The source code files must comprehensively encompass all instructions and explanations to document the commands and configurations. To facilitate this process, the ECF framework offers a form model comprising questions designed around standard software components used in scientific environments. Researchers must respond to these questions while analyzing the prerequisites for constructing the CM. The subsequent phase involves crafting the source code for the container image, utilizing the identified requirements. Once the container image's source code has been written, it should be securely stored within a version control system, such as Github or Gitlab. Moving forward, the source code must be compiled to produce the image that underpins the development and execution of the scientific application. Additionally, comprehensive testing of the image is imperative. The final stride entails storing the image in a container repository like Docker Hub.

Having the CM concluded, it is necessary to implement the Virtual Machine Module (VMM). For this, the framework defines the following steps:

- a) requirements identification;
- b) development of the VMM source code;
- c) source code storage.

The VMM consists of four fundamental layers: the hypervisor, the virtual machine, the container engine, and the CM itself. This predefined structure obviates the need for adding or removing layers. During the requirements identification stage, researchers are responsible for specifying the hypervisor supporting the virtual machine, determining its memory and CPU allocation, and selecting the operating system for installation. The container engine is already established at this stage, and the CM is ready for deployment. The remaining decisions relate to the Infrastructure-as-Code (IaC) tools the researcher intends to employ for VMM development. Additionally, the ECF provides a structured questionnaire to facilitate researchers' effective navigation through this phase. With all the information necessary to create the VMM, researchers should start developing the source code. The source code must handle the hypervisor installation on the physical machine, as well as the setup of the virtual machine containing both the container engine and the CM. Thorough documentation outlining the VMM steps should be integrated into the source code files. While the source code

generated in this phase should be stored in a version control system, keeping the virtual machine image is unnecessary. This image's purpose is solely to launch a container image representing the scientific environment, making it suitable for local storage facilitating its convenient destruction and recreation. During initialization, the VMM should check for updates to the CM in the image repository. If a newer version is available, it should be downloaded before instantiation.

3 MATERIAL AND METHODS

In this section, we describe our experience in recreating a computational environment for the bioinformatic pipeline named MetaWorks presented by Porter and Hajibabaei in (Porter & Hajibabaei, 2022). The first part of our experiment implements the computational environment of the MetaWorks pipeline following the original documentation provided by the authors (MetaWorks, 2024b). The second part implements the MetaWorks environment following the ECF framework's architecture and guidelines. This part of the experiment explores two points of view. The first analyzes the implementation from the point of view of the owner of the research, and the second from the point of view of those who want to reproduce the results published by third parties. The source code produced during our experiment is available in (Gomes, 2024).

Our experiment was performed using three different physical machines using different operating systems to measure the independence of platforms. The machine one (M1) is a PC notebook configured with Ubuntu Linux v22.04 64-bit operating system, an Intel(R) Core(TM) i7-5500U CPU @ 2.40GHz processor, 16 GB of RAM and a hard disk 512 GB SSD. The machine two (M2) is a PC notebook configured with Microsoft Windows 10 Home Edition 64-bit operating system, an Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz processor, 16 GB of RAM and a hard disk 512 GB SSD. The last one, machine three (M3), is a PC notebook configured with Fedora Linux v36 64-bit operating system, an Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz processor, 16 GB of RAM and a hard disk 512 GB SSD. It is essential to highlight that the M1 PC's operating system is a Debian-based Linux distribution, Ubuntu, and the M3 PC is Redhat-based, Fedora.

3.1 IMPLEMENTING THE ORIGINAL METAWORKS ENVIRONMENT

As mentioned earlier, all the steps needed to install, configure and test the MetaWorks environment must be executed manually.

The official tutorial of installation and configuration defines the following steps that have to be executed by a researcher when provisioning the environment:

- a) install MetaWorks;
- b) install and initialize Conda;
- c) activate the MetaWorks environment;
- d) install a custom-trained classifier;
- e) install ORFfinder.

Also, the tutorial provides a step to test the pipeline and certify the environment was correctly configured. If the test is performed with success a final output file named results.csv is generated, and it can be analyzed by importing it into R, for example, for bootstrap support filtering, pivot table creation, and normalization.

On M1 PC, the steps were successfully executed, and it was necessary 91 minutes to perform the entire procedure. Regarding the test step, it was performed in 23 minutes. On M3 PC, the installation and configuration were successfully performed in 97 minutes, and the test step was completed in 21 minutes. On M2 PC it was not possible to install the MetaWorks because the pipeline is only available for Linux platform.

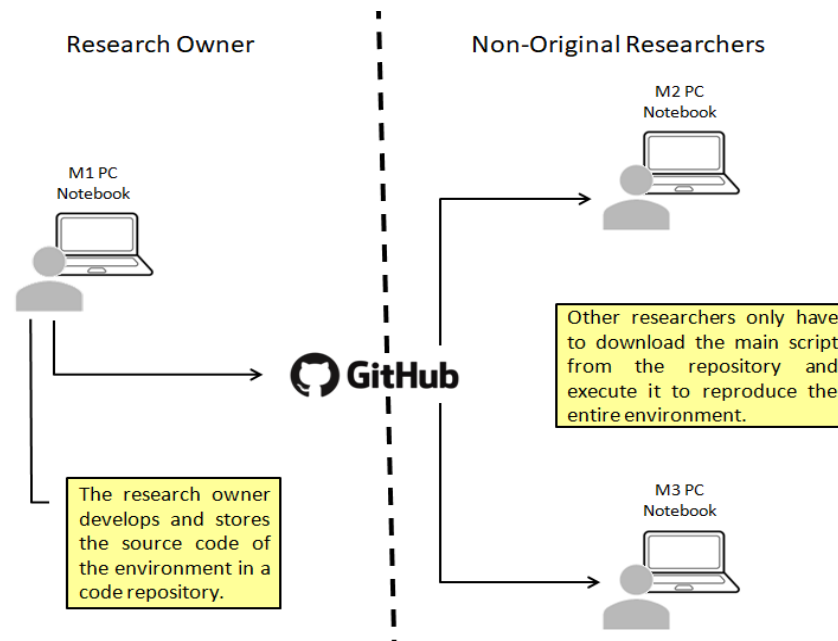
3.2 IMPLEMENTING THE METAWORKS ENVIRONMENT BY FOLLOWING THE ECF FRAMEWORK

This topic of the experiment has two parts. The first part presents the provisioning of the MetaWorks computational environment following the guidelines defined by the ECF framework (Gomes et al., 2022). It shows which steps are needed to create an environment, in a programmatic way, that will support the development and execution of a scientific application. The result must be a self-contained infrastructure, self-documented, that can be stored in a source control and easily reproducible. It consists of the work performed by the owner of the research. The second part presents a point of view from someone who wants to reproduce the environment exactly as the research owner developed it. According to the ECF framework's guidelines, it must occur simply, with minimal effort and manual intervention.

As shown in the Figure 3, to create the environment from the research owner point of view it was used the M1 PC notebook. The M2 and M3 PC notebooks were used to reproduce the same environment from the point of view of non-original researchers.

Figure 3

Graphical representation of the experiment.



Our experiment demonstrates a situation where the entire computational environment, composed by infrastructure and applicational environment, is totally based on source code being stored in a Github repository. In this way, we decided to do not use prebuilt container images. The container images are generated automatically, during the installation and configuration of the environment.

3.2.1 The Research Owner Point of View

We started to build the Container Module (CM) by following the steps described earlier. During the analysis step, approximately 60 minutes were dedicated to reviewing and completing the form containing the environmental requirements, with many of them already outlined in (Porter & Hajibabaei, 2022).

Table 1 displays the form containing the questions and answers utilized in the creation of the container module (CM) for the MetaWorks environment.

Tabela 1*MetaWorks' Container Module form.*

Question	Answer	Version
Which container engine will be used?	Docker and dependencies	24.0.2
Which will be the base image of the containers?	Ubuntu Linux	22.04
Which libraries, packages and third-party software have to be installed?	Miniconda and dependencies	3
	wget	1.19.4-1ubuntu2.2
	libuv1	1.18.0-3
	libdw1	0.170-0.4ubuntu0.1
	unzip	6.0-21ubuntu1.2
Is it necessary to perform any configuration?	libnghttp2-14	1.30.0-1ubuntu1
Is it necessary to perform any configuration?	Yes. It is necessary to change the \$PATH environment variable using the command: export PATH="/root/miniconda3/bin:\$PATH"	N/A
Is it necessary to copy any files into the container? Which files?	Yes. It is necessary to copy the following file to root directory: Runme.sh	N/A

As we defined Docker as the container engine to be used in the CM, we started to write the Dockerfile specifying the installation and configuration of all the software, libraries, packages, and dependencies that would compose the environment, according to the specification shown in Table 1. Also, we used the Dockerfile to document the parts of the environment that were being installed and configured inside the container module (CM). It took around 125 minutes to create the code and the documentation of the CM. The developed source code was stored in a Github repository.

The subsequent step involved creating a container image from the Dockerfile definitions. Docker completed this task in 35 minutes. To validate the MetaWorks applicational environment we instantiated a container from the generated image, and ran a test based on the official documentation, the same described in the previous topic. All the steps of this process took around 24 minutes.

In total, when considering all phases of CM development, achieving a successful outcome required approximately 244 minutes as shown in Table 2.

Tabela 2*Steps to create the Container Module (CM).*

Step	Time stent in minutes
Analyzis of the container module's requirements	60
Development and storage of the Dockerfile in a Github repository	125
Container image generation by Docker	35
Tests to validate the environment	24

Once the container module (CM) was functioning correctly, our attention turned to the virtual machine module (VMM), where we followed the VMM guide. To begin, we conducted an assessment of the essential prerequisites needed to build a virtual machine capable of supporting the MetaWorks. Regarding the operating system, we decided to use the same as the CM to maintain compatibility as outlined in Table 3. To perform this analysis we took around 40 minutes.

Table 3

MetaWorks' Virtual Machine Module form.

Question	Answer	Version
Which hypervisor will be used?	Oracle Virtualbox	7.0.10
How much memory will be allocated for the virtual machine?	11 GB	N/A
How much CPUs will be dedicated to the virtual machine?	1 CPU	N/A
Which operating system will be installed on the virtual machine?	Ubuntu Linux	22.04
Which container engine will support the containers?	Docker	24.0.2
Which IaC tools will be used to automate the provisioning of the environment?	Vagrant and dependencies	2.3.7
	Ansible and dependencies	2.12.2
Other resources	Shell-scripts Linux (main script)	N/A
	Shell-scripts Windows (main script)	N/A

The initial step involved implementing a main script responsible for launching the MetaWorks environment. Since our objective was to conduct tests on both Linux and MS-Windows machines, we needed to develop this primary script to accommodate both operating systems. The script's functionality encompasses several checks and actions. Initially, it verifies whether Virtualbox is installed on the PC; if not, the hypervisor is downloaded and installed on the machine. Subsequently, it checks for the presence of Vagrant and Ansible, initiating their installation if they are absent. Following this, the script proceeds to provision a virtual machine configured with Ubuntu Linux. To enable Vagrant to create the virtual machine as per our specifications, we defined the necessary configurations and installations within a file named Vagrantfile. Within this file, we specified the allocation of RAM and the number of CPUs for the virtual machine, described in Table 3. Additionally, we requested the installation of the Docker engine using Ansible. Once the virtual machine is up and running, the script handles the downloading of the container module (CM) from the Docker Hub, if necessary, and initiates the launch of a container encapsulating the MetaWorks environment.

Coding all components that make up the VMM, including Linux and MS-Windows scripts and configuration files for Vagrant and Ansible, required approximately 295 minutes to complete. In this phase, the scripts had to undergo individual and combined testing, which consumed about 315 minutes. This testing duration encompasses the assessments conducted with the CM and the VMM working in tandem. When factoring in the time required for implementing both modules, the CM and VMM, the total time expended amounted to 894 minutes. It is essential to highlight that all the source code developed to create both modules were stored in a Github repository.

3.2.2 The Point of View From Non-Original Researchers

This part of the experiment exposes the point of view of a scientist who wants to reproduce the results of research published by others. According to the ECF framework specification, the environment must be provisioned with minimal effort and manual intervention as possible, in this case, by running only one script. Actually, this is the consolidation of all the work developed by the owner of the research that was described previously. This test was performed on the M2 and M3 PCs.

The operation was initiated by downloading the main script from the repository on both machines, for MS Windows on the M2 PC and Linux on the M3 PC. This script is the only file requiring manual intervention by a researcher seeking to replicate the environment. Running this script on the two PCs seamlessly and successfully provisioned the MetaWorks environment.

This entire process consumed 65 minutes on the M2 PC and 57 minutes on the M3 PC, considering a scenario where all the software necessary to establish the infrastructure of the environment, such as Virtualbox, Ansible, and Vagrant, had to be downloaded from the internet. Also, as mentioned before, we opted to generate the Container Module (CM) during the provisioning of the environment consuming, in average, around 30 minutes.

After provisioning the environment automatically, the manual test suggested in the official documentation of MetaWorks was performed on both machines to certify that it was working correctly. On M2 PC, the test ran in 22 minutes and, on M3 PC, in 21 minutes.

3.2.3 The output validation

As mentioned earlier, the official documentation of the MetaWorks pipeline provides a guide that permits us to validate if the provisioned environment is working properly. When the test is executed with success a file named results.csv is created (MetaWorks, 2024b).

Our experiment successfully performed the test in all environments provisioned manually and automatically. The same results.csv file was generated with 129 KB of size in all tests. Also, the content of the files was compared programmatically through a script developed in Python. No differences were found between them.

Having the same output in the environments provisioned by following the ECF framework and those installed and configured according to the official documentation validates our proposal and demonstrates that the MetaWorks pipeline based on the ECF framework is trustable.

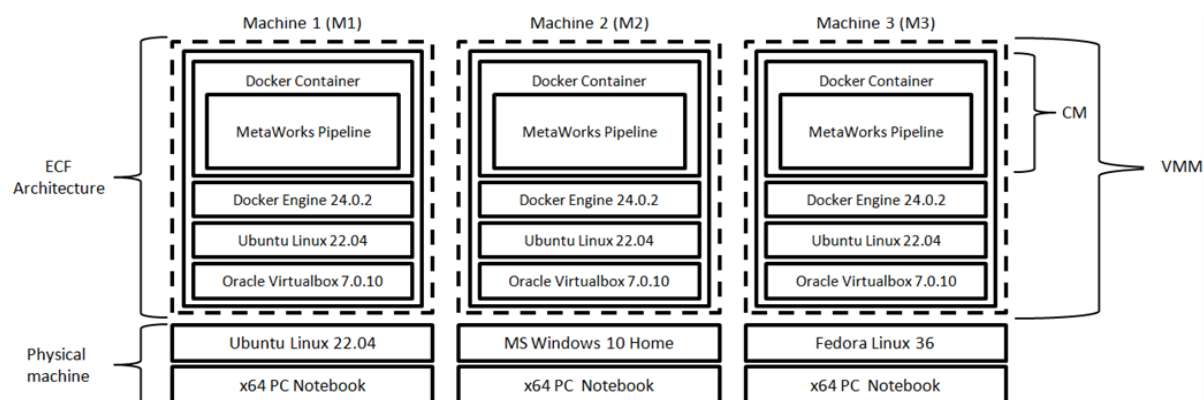
3.2.4 Technical aspects of the proposed method

With the container module (CM) and the virtual machine module (VMM) adequately implemented, the MetaWorks computational environment can be provisioned as many times as needed only by running a script with minimal manual intervention. The environment can be provisioned on many computers as necessary, and it will always be the same independently if it is being created by a research team member or by a third person. Having the same environment, the results produced by the scientific application will also be the same.

The virtual machine module (VMM) creates a standard layer that guarantees the homogeneity necessary to run the container module (CM) on the same operating system independently of the platform. Our experiment represents it by the Oracle Virtualbox 7.0.10 running a lightweight virtual machine with Ubuntu Linux 22.04 and the Docker engine 24.0.6 within it. The main script, developed for MS Windows and Linux platforms, uses Vagrant and Ansible to programmatically install and configure this layer, corresponding to the infrastructure. Figure 4 details the homogeneous environment provisioned on the three machines used in the experiment, each using a different operating system.

Figure 4

MetaWorks pipeline environment provisioned on the three machines used in the experiment by following the ECF framework.



We can observe in Figure 4 that the Docker container that encapsulates the MetaWorks applicational environment will run on the same operating system in every machine where the environment is provisioned. Also, the ECF framework guarantees that the same infrastructure will support the entire environment, namely the hypervisor, operating system, and container engine.

4 RESULTS AND DISCUSSION

The primary objective of the ECF framework is to improve reproducibility through the Infrastructure-as-Code (IaC) approach, assisting original researchers in creating a computational environment that can be easily reproduced by them and other researchers. Naturally, to improve reproducibility and develop mechanisms that enable the efficient recreation of computational environments, it will be imperative for the researchers responsible for provisioning to invest significant effort in acquiring the knowledge and skills required for working with IaC tools.

The presented discussion should be considered from both of the previously mentioned perspectives: one from the original researcher creating the environment through programmatic methods, and the other from the researcher tasked with reproducing it. The comparison between provisioning methods, one based on the official MetaWorks documentation and the other based on the ECF framework, took into account the following parameters: time consumption, efforts, manual intervention, and platform agnosticism.

By adhering to the method described in (MetaWorks, 2024b), our environment provisioning process averaged 94 minutes. Considering the test to verify the proper functioning

of the environment the average is 116 minutes. The level of manual intervention is high, requiring that all stages of installation and configuration of the environment be carried out manually. Anyone needing to recreate the MetaWorks computing environment always will perform the same steps in a manual way. Be it the owner of the research, a member of the research team, or even someone who wants to reproduce published results by others. Despite the rich and detailed documentation, manual execution of the instructions may result in errors, incorrect configurations, or even a different environment than expected increasing the risk to generate different results from the original.

By following the ECF framework from the point of view of the original researcher, it was necessary approximately sixteen hours to build the entire computational environment that supports the MetaWorks using the M1 PC. It is essential to highlight that it was designed to assist those directly involved in research development and anyone who wants only to run an application and verify published results. From this second point of view, the effort necessary would be simply downloading and executing only one script to run a non-interactively installation and to get the computational environment ready to use. For this, we used exclusively the M2 and M3 PCs achieving, on average, 61 minutes only to provisioning the environment automatically. Considering the test executed after provisioning it, the average is 82.5 minutes.

Regarding platform-agnosticism, the original MetaWorks is limited to running on Linux operating systems. In our experience, it could be provisioned only in two of three available machines because one has MS-Windows as the operating system. On the other hand, the MetaWorks environment based on the ECF framework could be provisioned on all machines. The ECF framework opens new possibilities, extending the MetaWorks to run on any platform that supports the hypervisor of the VMM, in this case, the Oracle Virtualbox. Besides Linux and MS-Windows, the pipeline could be provisioned on these operating systems: Mac OS X and Solaris. Another advantage of creating the environment by programmatic means is the possibility to create and destroy at any time, recreating it as often as needed.

The documentation stands as another advantageous aspect of employing the ECF framework for MetaWorks environment development. Utilizing the Infrastructure as Code (IaC) approach, all environment components are defined through code. Consequently, we meticulously detailed and elucidated the environment, installations, and configurations within the source-code files we generated for Vagrant, Docker, Ansible, and the shell scripts. This documentation serves as a critical guide not only for those aiming to reproduce the environment but also for anyone seeking to comprehend the intricacies of its development process.

5 CONCLUSION

In Porter and Hajibabaei (2022), the authors present a flexible pipeline called MetaWorks that supports the bioinformatic processing of multiple popular markers such as rRNA genes, spacers, and protein-coding genes. To enhance the reproducibility, scalability, and shareability of the workflows, MetaWorks employs the Conda package manager for seamless program and dependency acquisition, along with the Snakemake workflow manager to automate pipelines and optimize computational resource utilization. As mentioned in the paper, the instructions on how to install and configure the pipeline are provided in an online documentation (MetaWorks, 2024b). Besides the MetaWorks encapsulates all the software, libraries, and packages necessary to create the computational environment for different types of bioinformatic pipelines, its installation and configuration process involves a high level of manual intervention.

In the illustrated case study, we demonstrated how to enhance the MetaWorks bioinformatic pipeline provisioning its computational environment by following the guidelines of the ECF framework. In fact, the ECF framework demonstrated its practical ability to programmatically establish a comprehensive computational environment with minimal researcher intervention required for its replication. The results clearly attest to its advantages, particularly in maintaining consistent environment behavior across the three machines employed in our experiment. Such empirical evidence underscores the significance of reducing manual intervention, as it ensures the consistent recreation of the environment for numerous iterations.

Indeed, we cannot solely highlight the advantages of the ECF framework because its adoption has associated costs. First and foremost, it's crucial to underscore the time commitment that original researchers must allocate to programming the various components of the environment. As previously mentioned, given our substantial expertise in the programming languages and tools utilized in the experiment and our familiarity with the Infrastructure-as-Code (IaC) approach, it took approximately fifteen hours to develop and thoroughly test the source code. However, it's worth noting that the development and testing phases can be more demanding when researchers do not possess an IT background. This leads to the second most significant cost: the time and effort required to gain proficiency in programming languages, IaC tools, and software engineering practices.

We strongly advocate the adoption of open-source software for researchers seeking to adhere to ECF guidelines when establishing their computational environments. These open-source tools align with the principles of open science and benefit from extensive user communities, expediting the learning curve and offering robust technical support. Furthermore, we recommend using established and widely accepted tools within the scientific community, such as Docker, Virtualbox, and Python. These mature tools exhibit fewer technical challenges than newer alternatives, and their abundance of documentation and active forums ensure comprehensive guidance for newcomers.

One noteworthy aspect of the ECF framework that we emphasize is its valuable educational potential in shaping the next generation of researchers. Facilitating the creation of transparent and reproducible research adds value and enriches the scientific community. As part of our future endeavors, we propose expanding the implementation of computational environments across various scientific domains to enhance further and refine the ECF framework.

REFERENCES

- Barba, L. A., & Thiruvathukal, G. K. (2017). Reproducible Research for Computing in Science Engineering. *Computing in Science Engineering*, 19(6), 85–87.
- Cacho, J. R. F., & Taghva, K. (2018). Reproducible research in document analysis and recognition. In *Information Technology-New Generations* (pp. 389–395). Springer.
- Cacho, J. R. F., & Taghva, K. (2020). The State of Reproducible Research. In *Computer Science, 17th International Conference on Information Technology – New Generations (ITNG 2020), Advances in Intelligent Systems and Computing* (Vol. 1134, pp. 519-524). Springer.
- Conda. (2024). *Conda's official website*. Retrieved from <https://docs.conda.io/en/latest>
- Coveney, P. V., Groen, D., & Hoekstra, A. G. (2021). Reliability and reproducibility in computational science: implementing validation, verification and uncertainty quantification in silico. *Philosophical Transactions of the Royal Society A*, 379, 1-5. <https://doi.org/10.1098/rsta.2020.0409>
- de Bayser, M., Azevedo, L. G., & Cerqueira, R. (2015). ResearchOps: The case for DevOps in scientific applications. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)* (pp. 1398–1404). IEEE.
- Docker. (2024a). *Troubleshoot topics for Docker Desktop*. Retrieved from <https://docs.docker.com/desktop/troubleshoot/topics/>

- Docker. (2024b). Workarounds for common problems. Retrieved from <https://docs.docker.com/desktop/troubleshoot/workarounds/>
- Docker. (2024c). Known issues. Retrieved from <https://docs.docker.com/desktop/troubleshoot/known-issues/>
- Edgar, R. C. (2016). *UNOISE2: improved error-correction for Illumina 16S and ITS amplicon sequencing*. bioRxiv. <https://doi.org/10.1101/081257>
- Gomes, D. A., Mestre, P., & Serôdio, C. (2019). Infrastructure-as-Code for Scientific Computing Environments. In *CENTRIC 2019: The Twelfth International Conference on Advances in Human-oriented and Personalized Mechanisms, Technologies, and Services* (pp. 7-10).
- Gomes, D. A., Mestre, P., & Serôdio, C. (2022). Environment Code-First Framework: Provisioning Scientific Computational Environments Using the Infrastructure-as-Code Approach. *International Journal on Advances in Software*, 15(1 & 2), 1-13.
- Gomes, D. A. (2024). *Metaworks based on ECF Framework* [Github repository]. Retrieved from https://github.com/adornogomes/MetaWorks_Based_On_ECF_Framework
- Grüning, B. A., Lampa, S., Vaudel, M., & Blankenberg, D. (2019). Software engineering for scientific big data analysis. *Gigascience*, 8(5). <https://doi.org/10.1093/gigascience/giz054>
- Humble, J., & Farley, D. (2010). *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education.
- Liu, D. M., & Salganik, M. J. (2019). Successes and Struggles with Computational Reproducibility: Lessons from the Fragile Families Challenge. *Socius*, 5, 1-21. <https://doi.org/10.1177/2378023119849803>
- Marwick, B. (2017). Computational Reproducibility in Archaeological Research: Basic Principles and a Case Study of Their Implementation. *Journal of Archaeological Method and Theory*, 24, 424–450. <https://doi.org/10.1007/s10816-015-9272-9>
- Martin, M. (2011). CutAdapt removes adapter sequences from high-throughput sequencing reads. *EMBnet journal*, 17, 10.
- MetaWorks. (2024a). *MetaWorks' official page on Github*. Retrieved from <https://github.com/terrimporter/MetaWorks>
- MetaWorks. (2024b). *MetaWorks' official implementation tutorial*. Retrieved from <https://terrimporter.github.io/MetaWorksSite/tutorial>
- Morris, K. (2020). *Infrastructure as Code: Dynamic Systems for the Cloud Age* (2nd ed.). O'Reilly Media, Inc.
- Porter, T. M., & Hajibabaei, M. (2022). MetaWorks: A flexible, scalable bioinformatic pipeline for high-throughput multi-marker biodiversity assessments. *PLoS ONE*, 17(9), 1-11. <https://doi.org/10.1371/journal.pone.0274260>

- Reinecke, R., Trautmann, T., Wagener, T., & Schüler, K. (2022). The critical need to foster computational reproducibility. *Environmental Research Letters*, 17. <https://doi.org/10.1088/1748-9326/ac5cf8>
- Segal, J., & Morris, C. (2012). Developing Software for a Scientific Community: Some Challenges and Solutions. In J. Leng & W. Sharrock (Eds.). *Handbook of Research on Computational Science and Engineering: Theory and Practice* (pp. 177-196). IGI Global. <https://doi.org/10.4018/978-1-61350-116-0.ch008>
- Snakemake. (2024). *Snakemake's official website*. Retrieved from <https://snakemake.github.io>
- St John, J. (2016). *SeqPrep's official page on Github*. Retrieved from <https://github.com/jstjohn/SeqPrep/releases>
- Wang, Q., Garrity, G. M., Tiedje, J. M., & Cole, J. R. (2007). Naive Bayesian Classifier for Rapid Assignment of rRNA Sequences into the New Bacterial Taxonomy. *Applied and Environmental Microbiology*, 73, 5261–5267. <https://doi.org/10.1128/AEM.00062-07>
- Wiebels, K., & Moreau, D. (2021). Leveraging containers for reproducible psychological research. *Advances in Methods and Practices in Psychological Science*, 4(2), Article 25152459211017853. <https://doi.org/10.1177/25152459211017853>
- Wiggins, A. (2017). *The Twelve-Factor App Official Website*. Retrieved from <http://12factor.net>