A Framework for Scientific Workflow Reproducibility in the Cloud

Rawaa Qasha, Jacek Cała, Paul Watson

Newcastle University, Newcastle upon Tyne, UK

Email: {r.qasha, jacek.cala, paul.watson}@newcastle.ac.uk

In this paper

- A new framework for repeatability and reproducibility of scientific workflow

- Integrating logical and physical preservation approaches

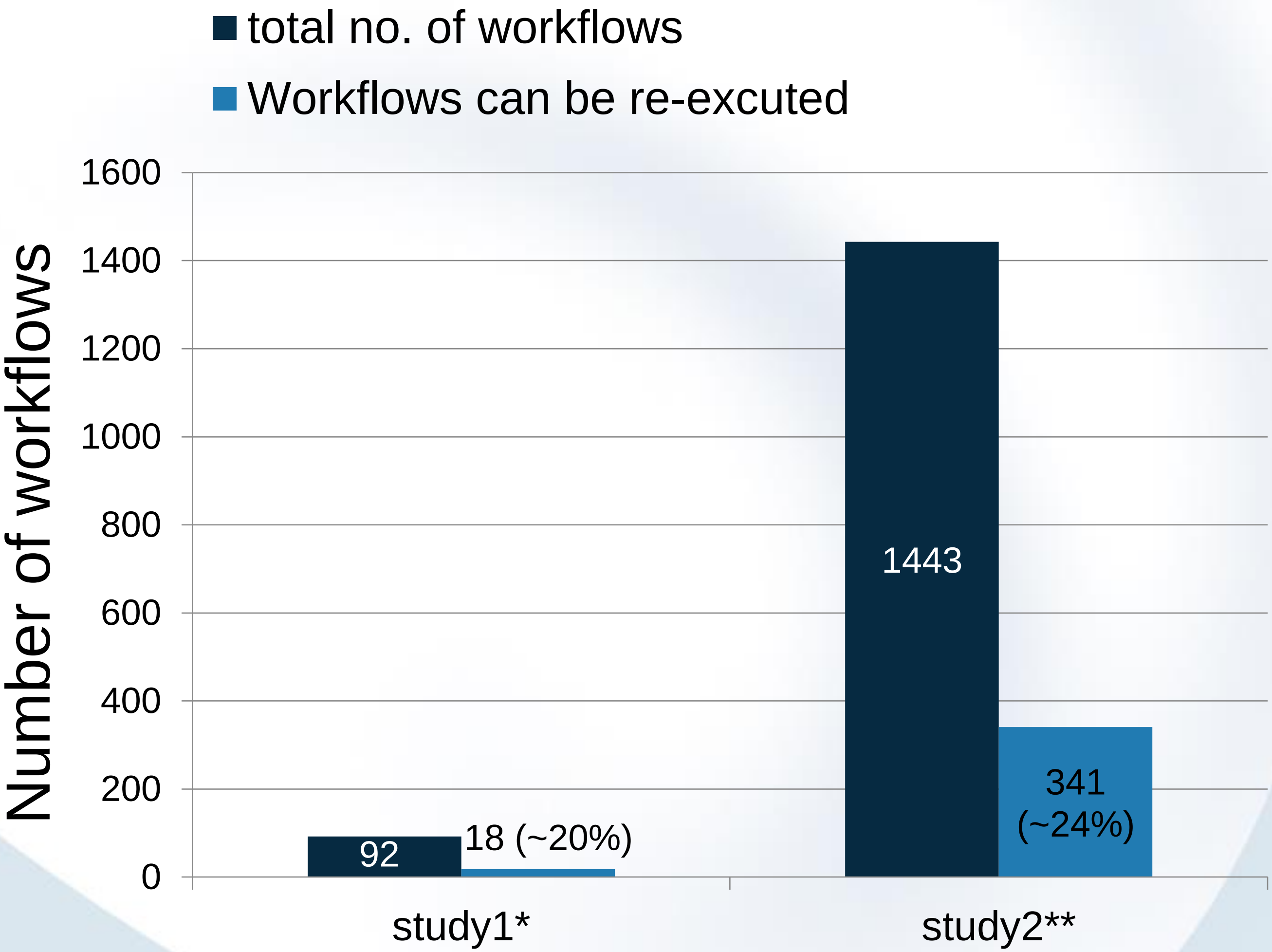
- Offering Workflow/tasks repositories with version control

- Supporting automatic deployment and image capture of workflows and tasks

Outline

- Background
- Challenges for workflow reproducibility
- Our solution for logical and physical preservations
- Overview of reproducibility framework
- Experiments and results
- Conclusions

Workflows & Reproducibility



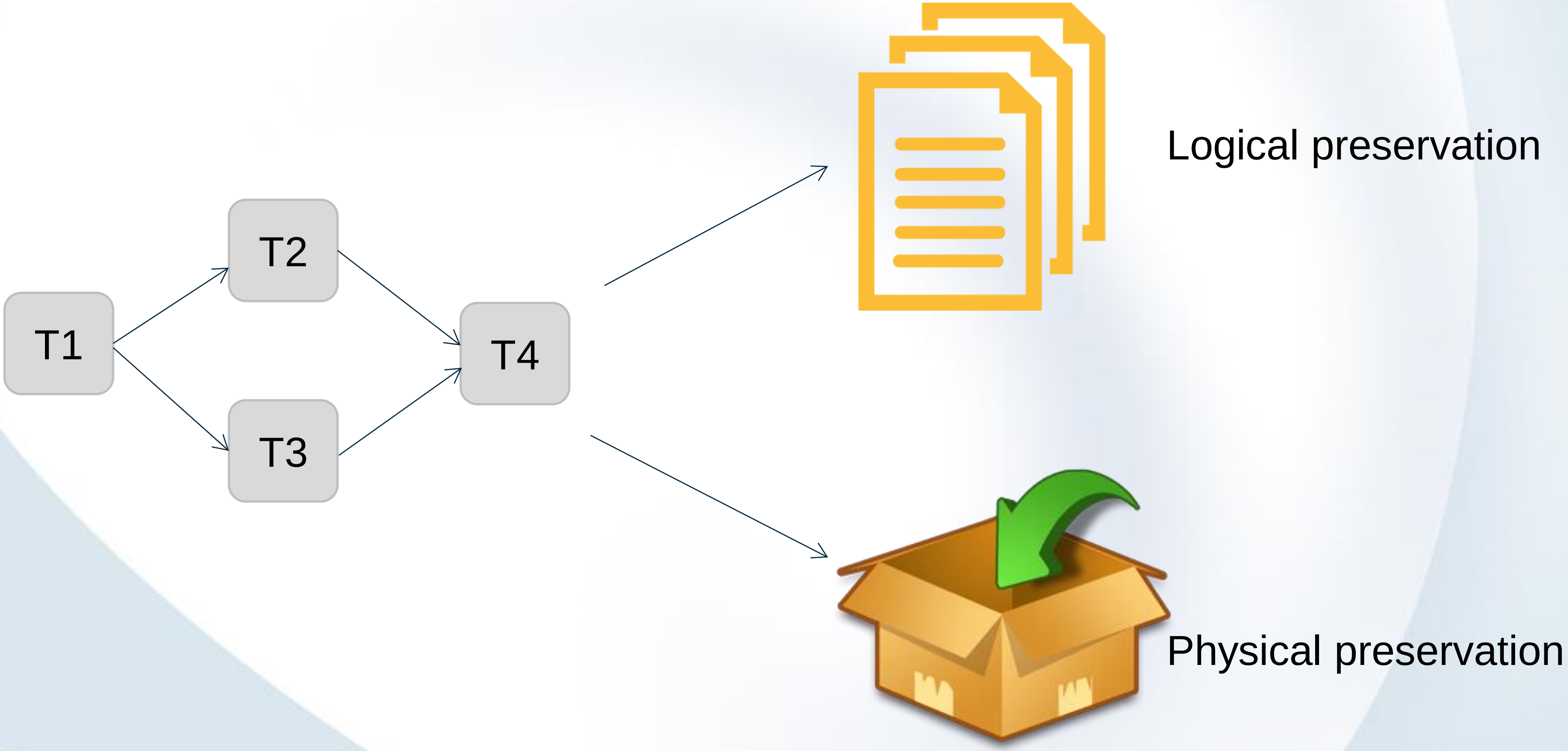
*Zhao et al, “Why workflows break Understanding and combating decay in Taverna workflows,” 2012

**Mayer et al, “A Quantitative Study on the Re-executability of Publicly Shared Scientific Workflows”, 2015

Challenges
for workflow reproducibility

- Insufficiently detailed workflow description
- Insufficient description of the execution environment
- Unavailable execution environments
- Absence of & changes in the external dependencies
- Missing input data

Common reproducibility approaches



T1

T2

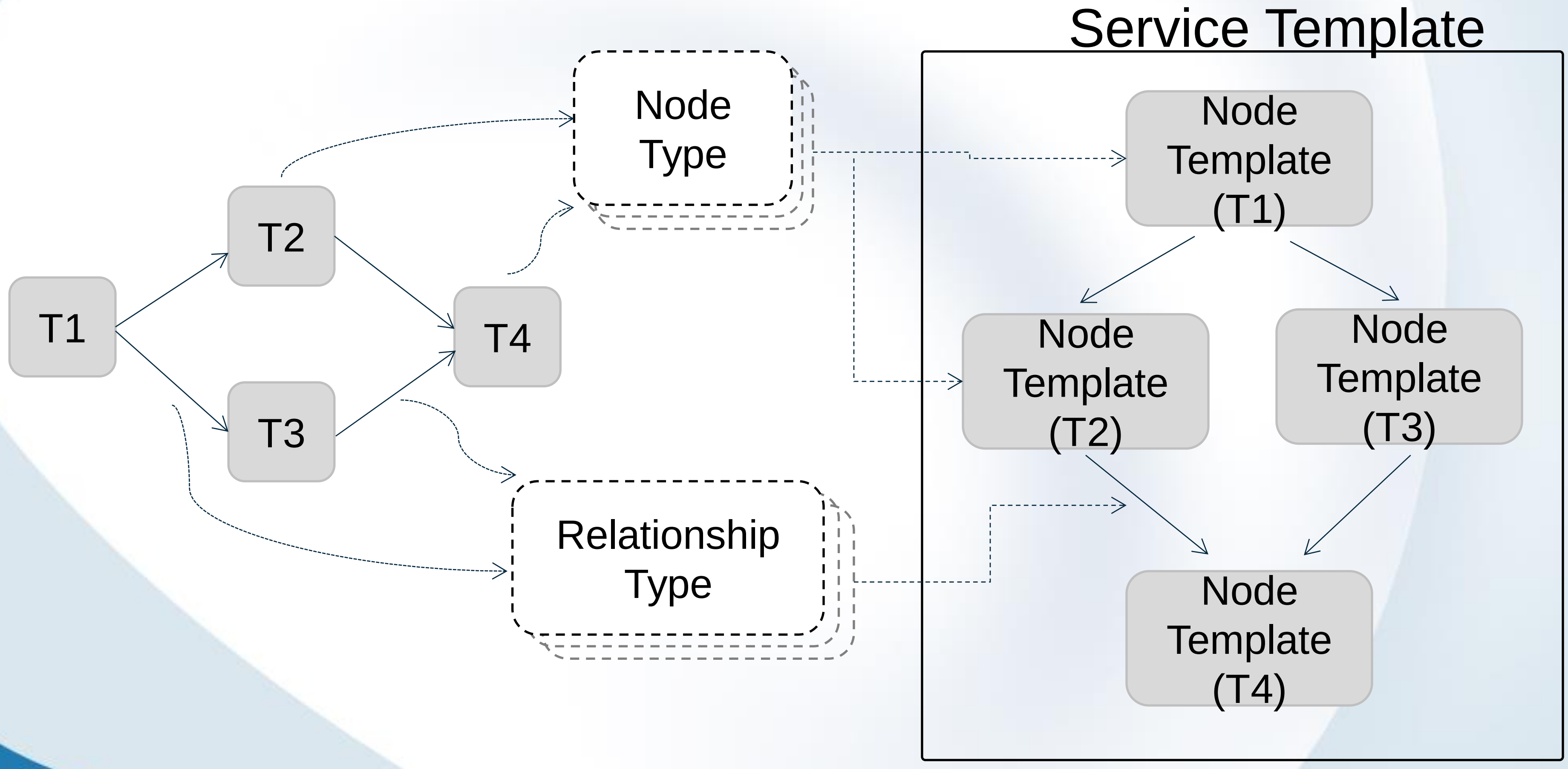
T3

T4

Logical preservation

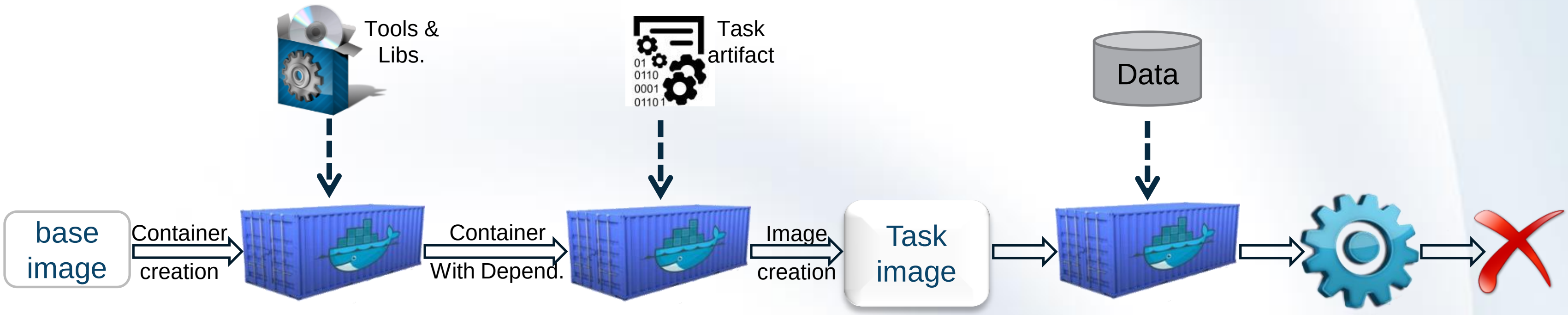
Physical preservation

Using TOSCA as a logical preservation



Workflow and execution environment description

Using Docker for physical preservation



Tools & Libs.

Task artifact

Data

base image

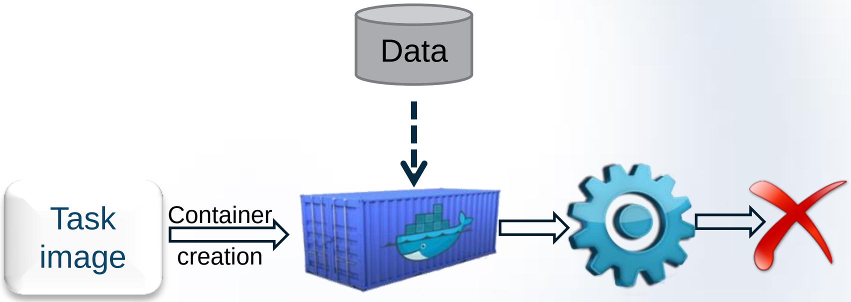
Container creation

Container With Depend.

Image creation

Task image

(a) Initial task deployment & execution



Data

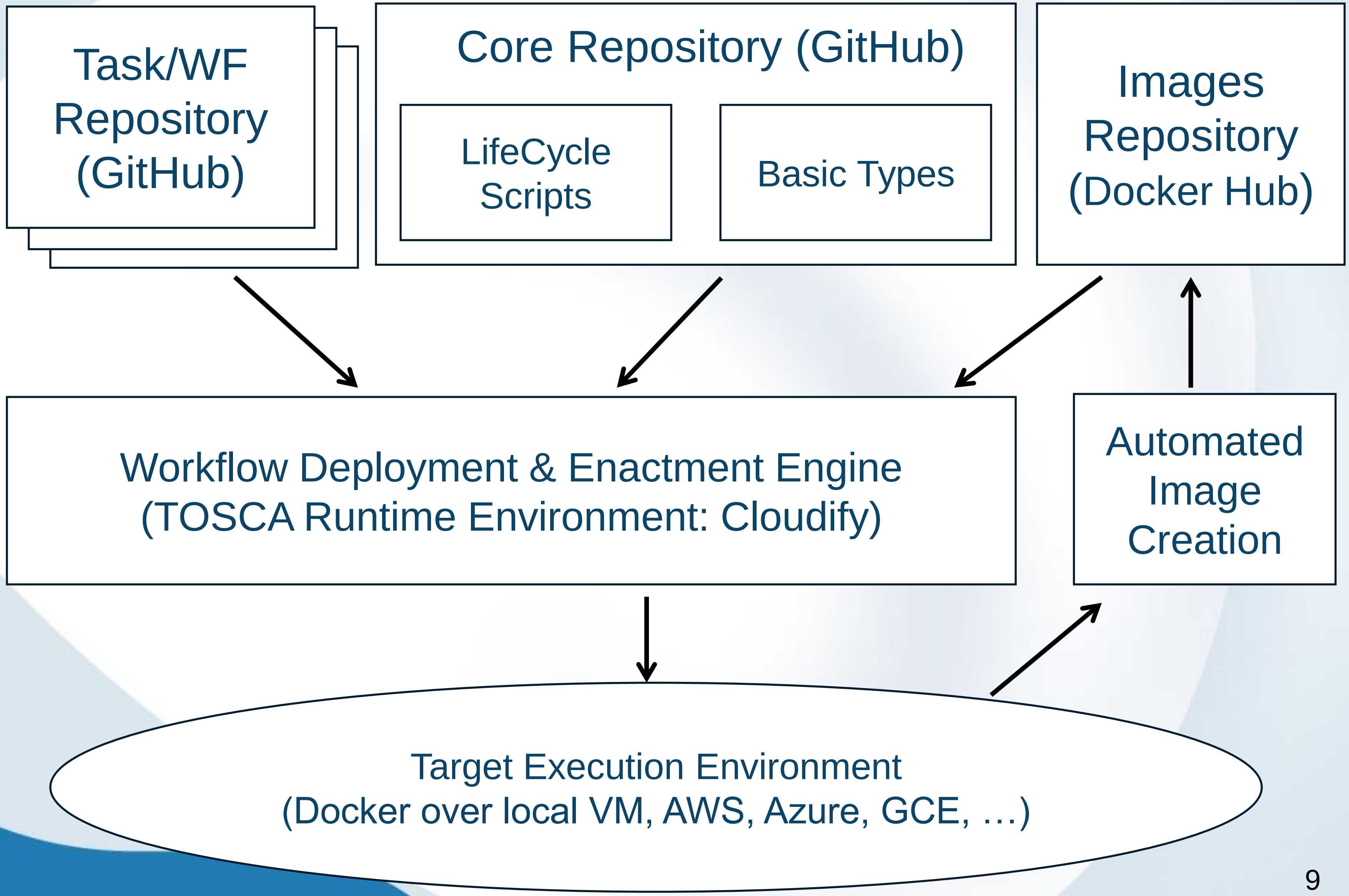
Task image

Container creation

(b) Task deployment & execution with task image

Preserving execution environment and dependencies, tracking changes

Reproducibility Framework



Task/WF Repository (GitHub)

Core Repository (GitHub)

LifeCycle Scripts

Basic Types

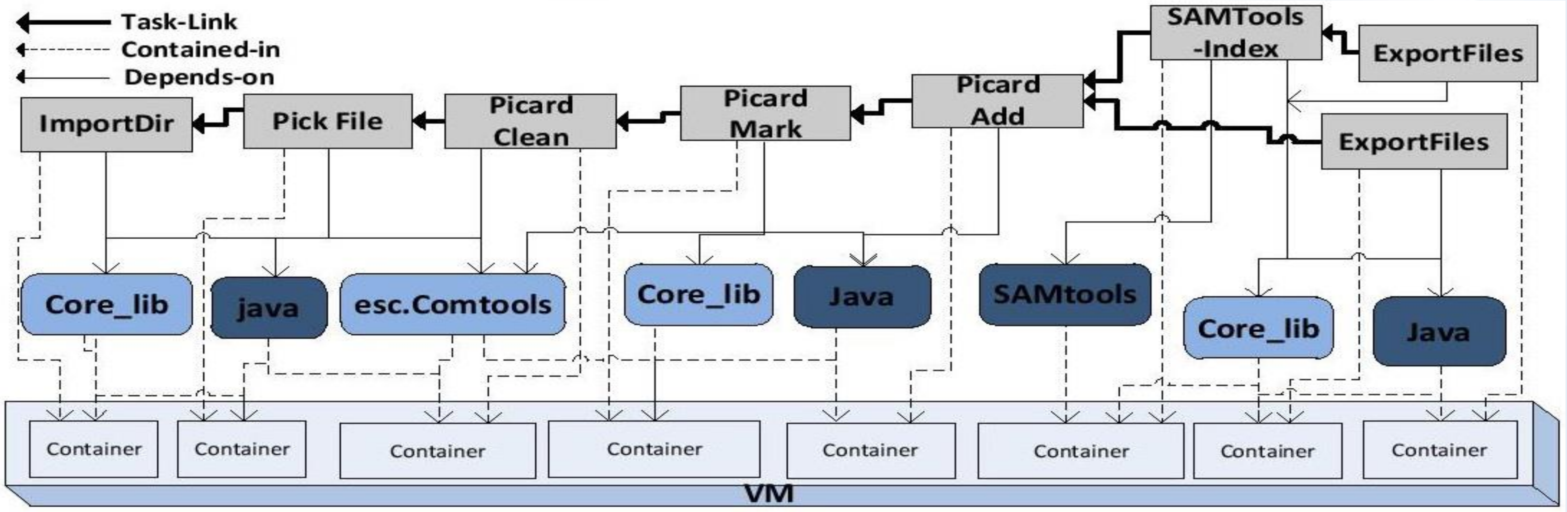
Images Repository (Docker Hub)

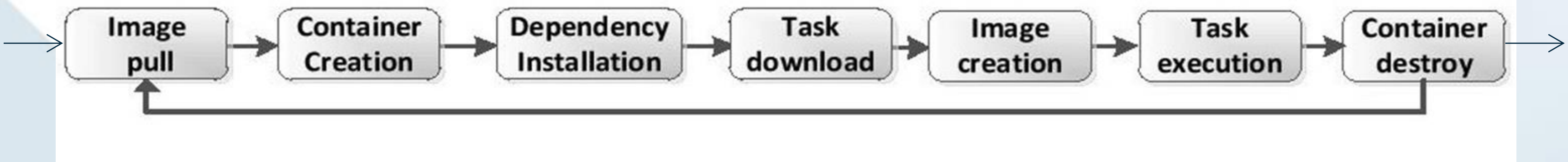
Workflow Deployment & Enactment Engine (TOSCA Runtime Environment: Cloudify)

Automated Image Creation

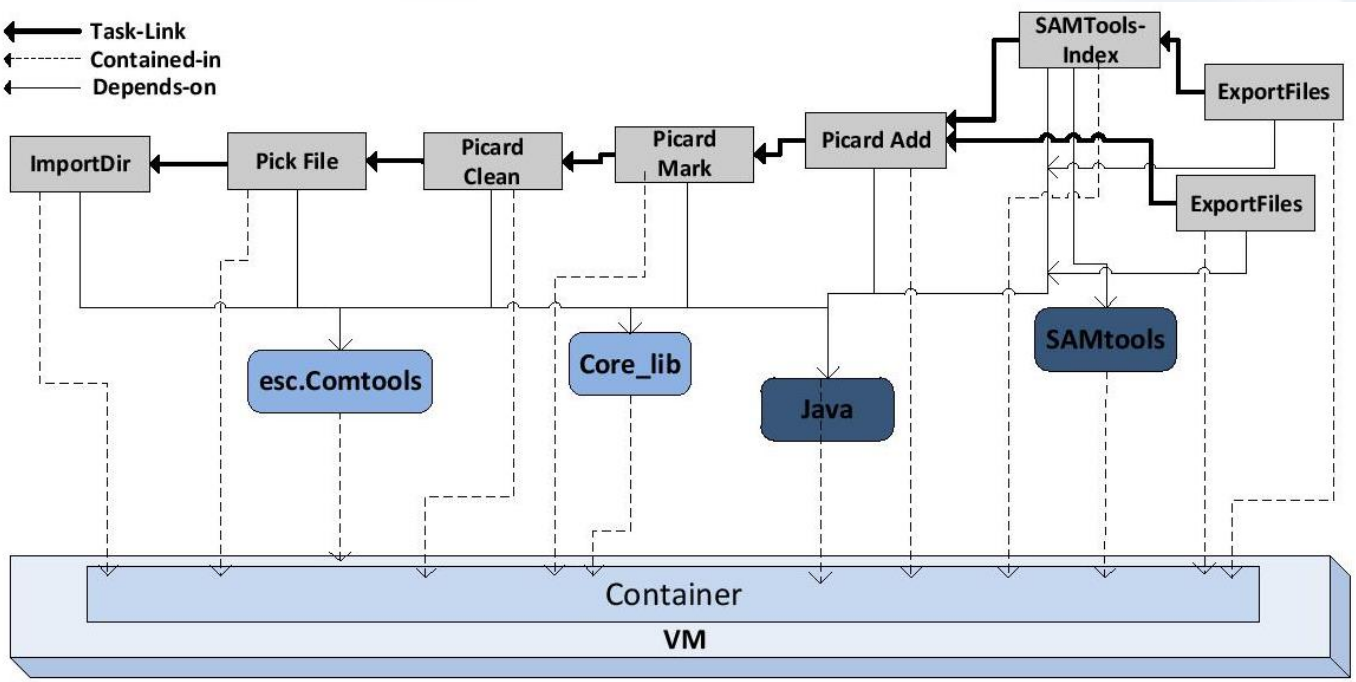
Target Execution Environment (Docker over local VM, AWS, Azure, GCE, …)

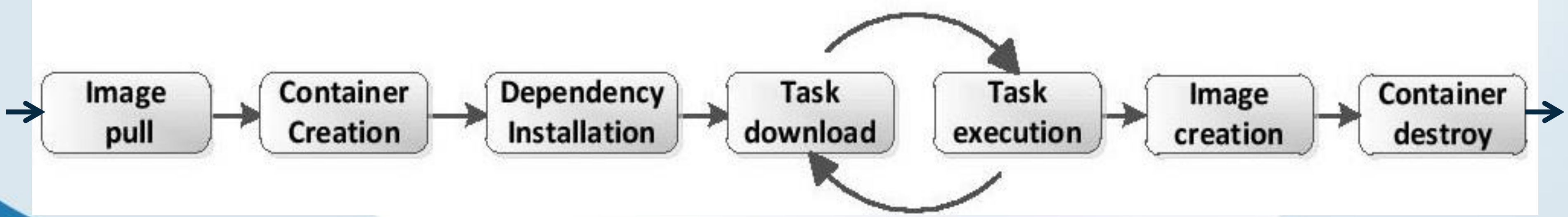
Multi-container deployment



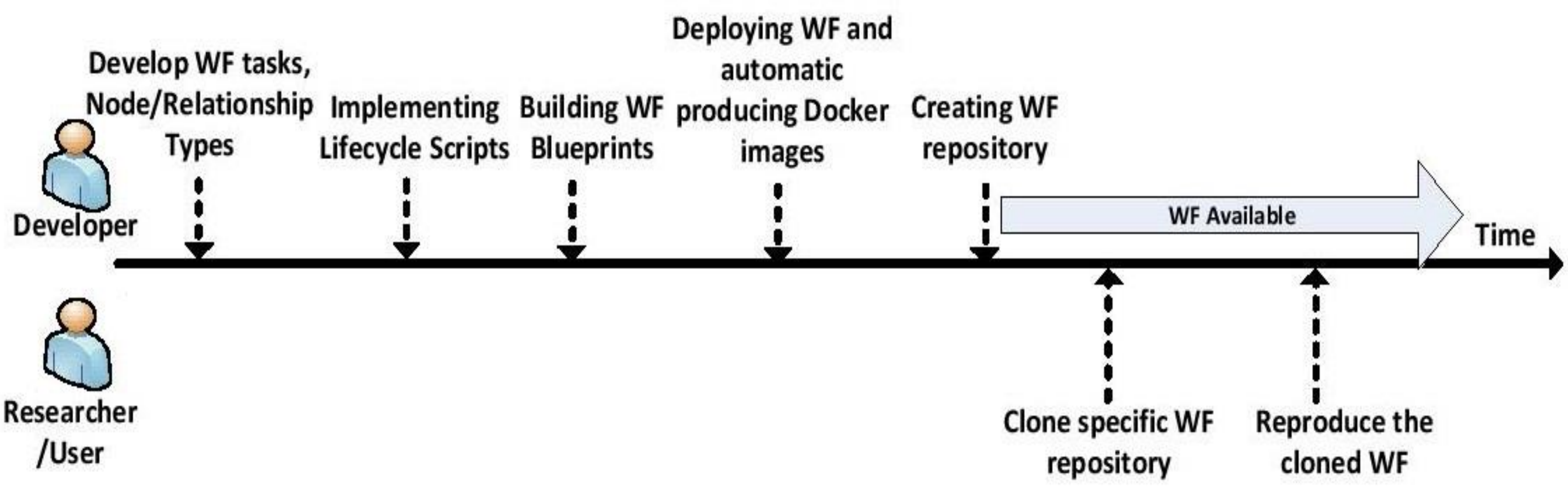


Single container deployment

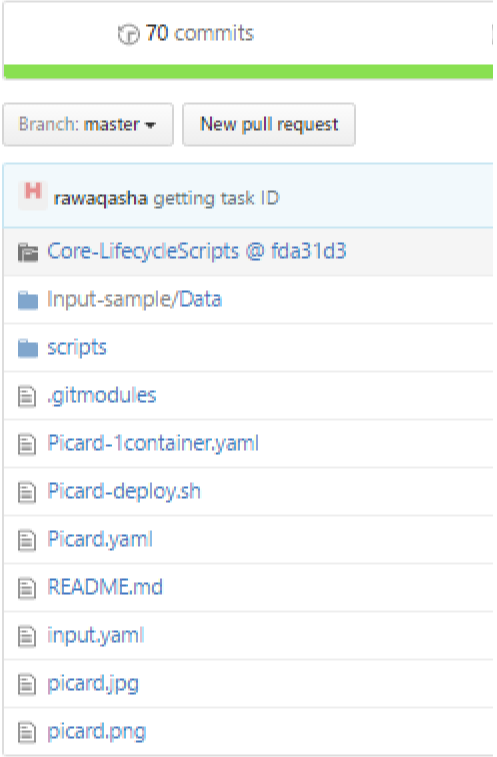




Time line of workflow devOps

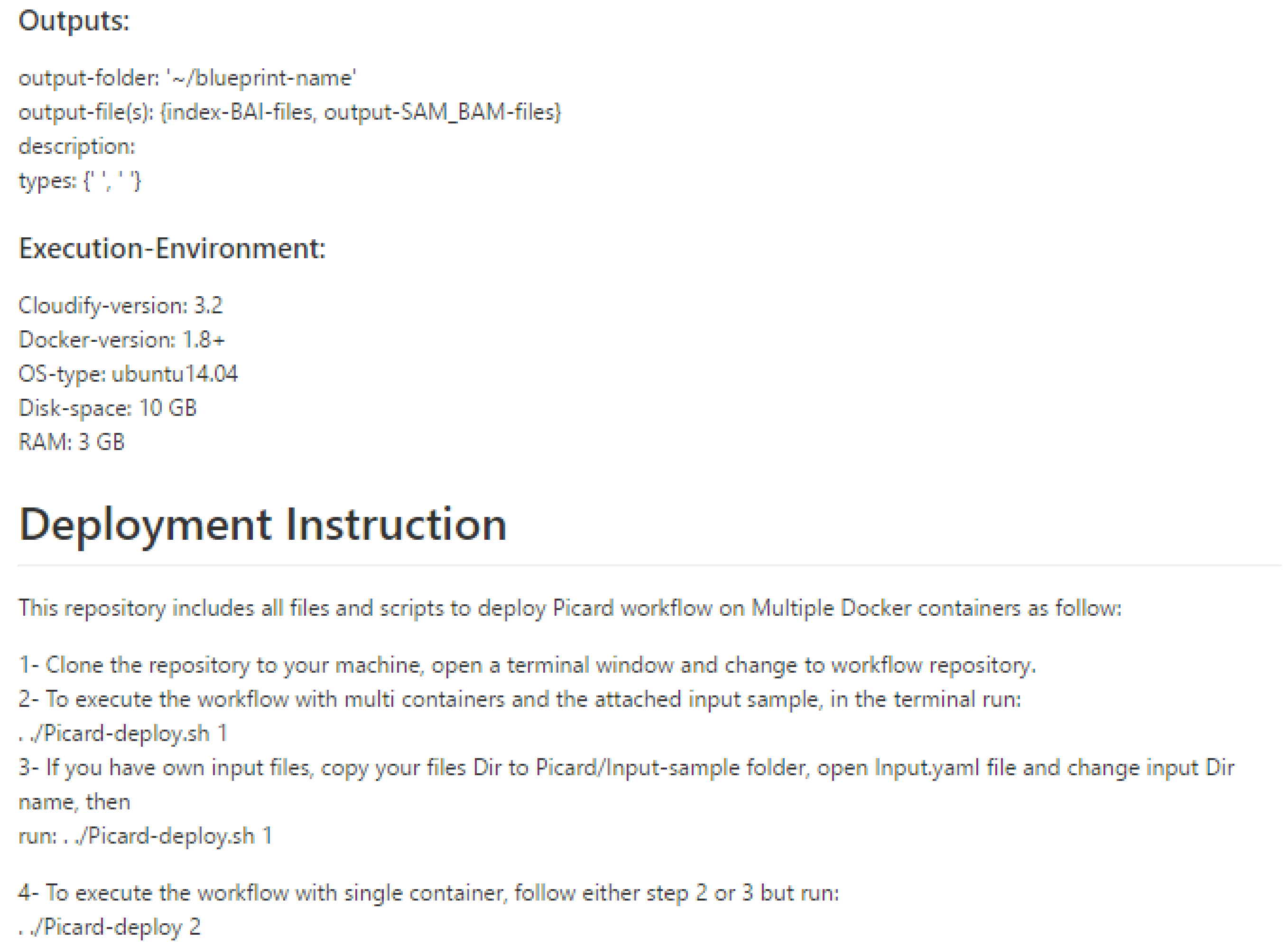

Developer
Develop WF tasks, Node/Relationship Types
Implementing Lifecycle Scripts
Building WF Blueprints
Deploying WF and automatic producing Docker images
Creating WF repository
WF Available
Time
Researcher /User
Clone specific WF repository
Reproduce the cloned WF

Workflow repository



rawaqasha getting task ID

- Core-LifecycleScripts @ fda31d3
- Input-sample/Data
- scripts
- .gitmodules
- Picard-1container.yaml
- Picard-deploy.sh
- Picard.yaml
- README.md
- input.yaml
- picard.jpg
- picard.png



Outputs:

output-folder: '~/blueprint-name'
output-file(s): {index-BAI-files, output-SAM_BAM-files}
description:
types: {' ', ' '}

Execution-Environment:

Cloudify-version: 3.2
Docker-version: 1.8+
OS-type: ubuntu14.04
Disk-space: 10 GB
RAM: 3 GB

Deployment Instruction

This repository includes all files and scripts to deploy Picard workflow on Multiple Docker containers as follow:

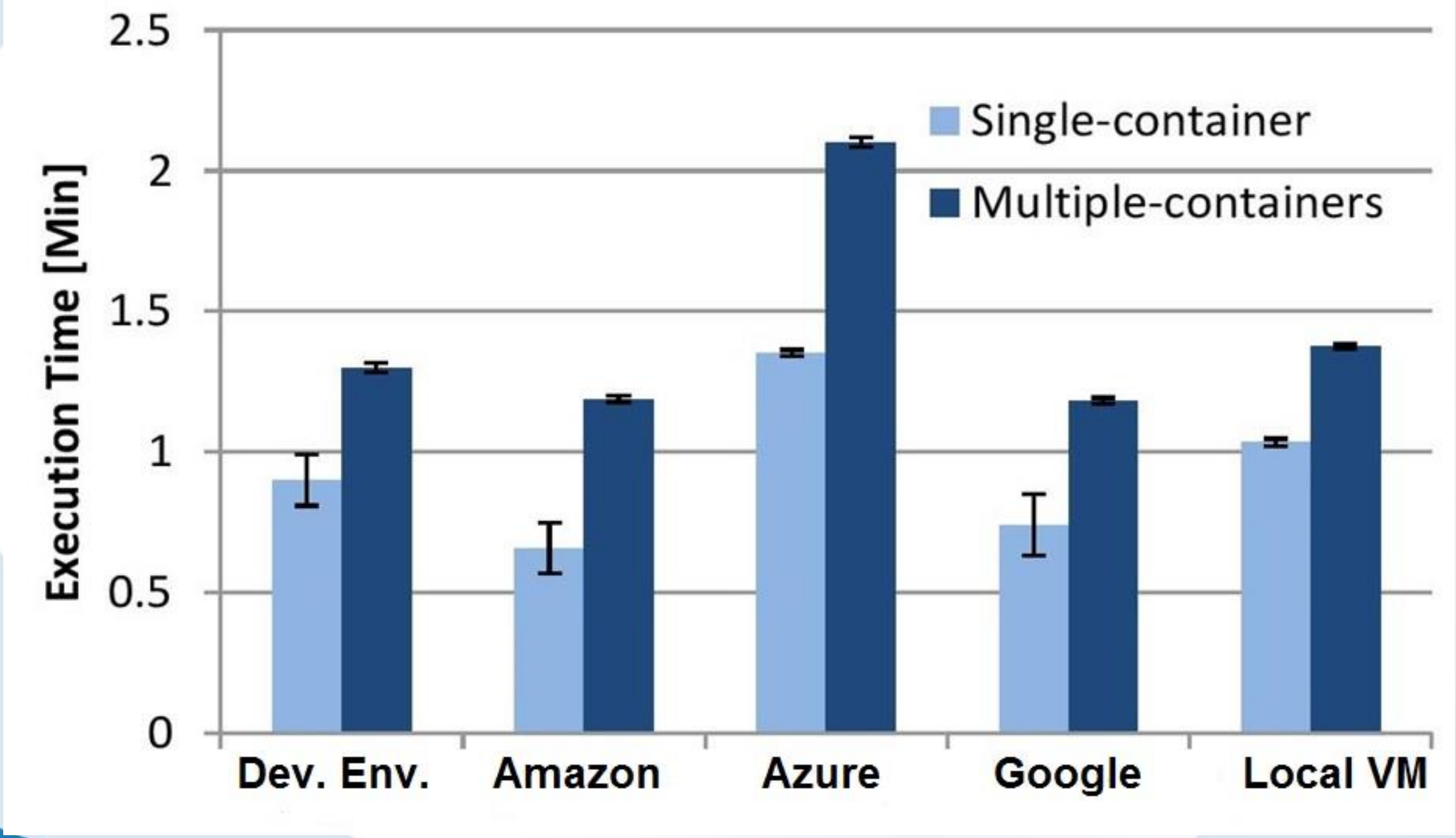
1- Clone the repository to your machine, open a terminal window and change to workflow repository.
2- To execute the workflow with multi containers and the attached input sample, in the terminal run:
. ./Picard-deploy.sh 1
3- If you have own input files, copy your files Dir to Picard/Input-sample folder, open Input.yaml file and change input Dir name, then
run: . ./Picard-deploy.sh 1

4- To execute the workflow with single container, follow either step 2 or 3 but run:
. ./Picard-deploy 2

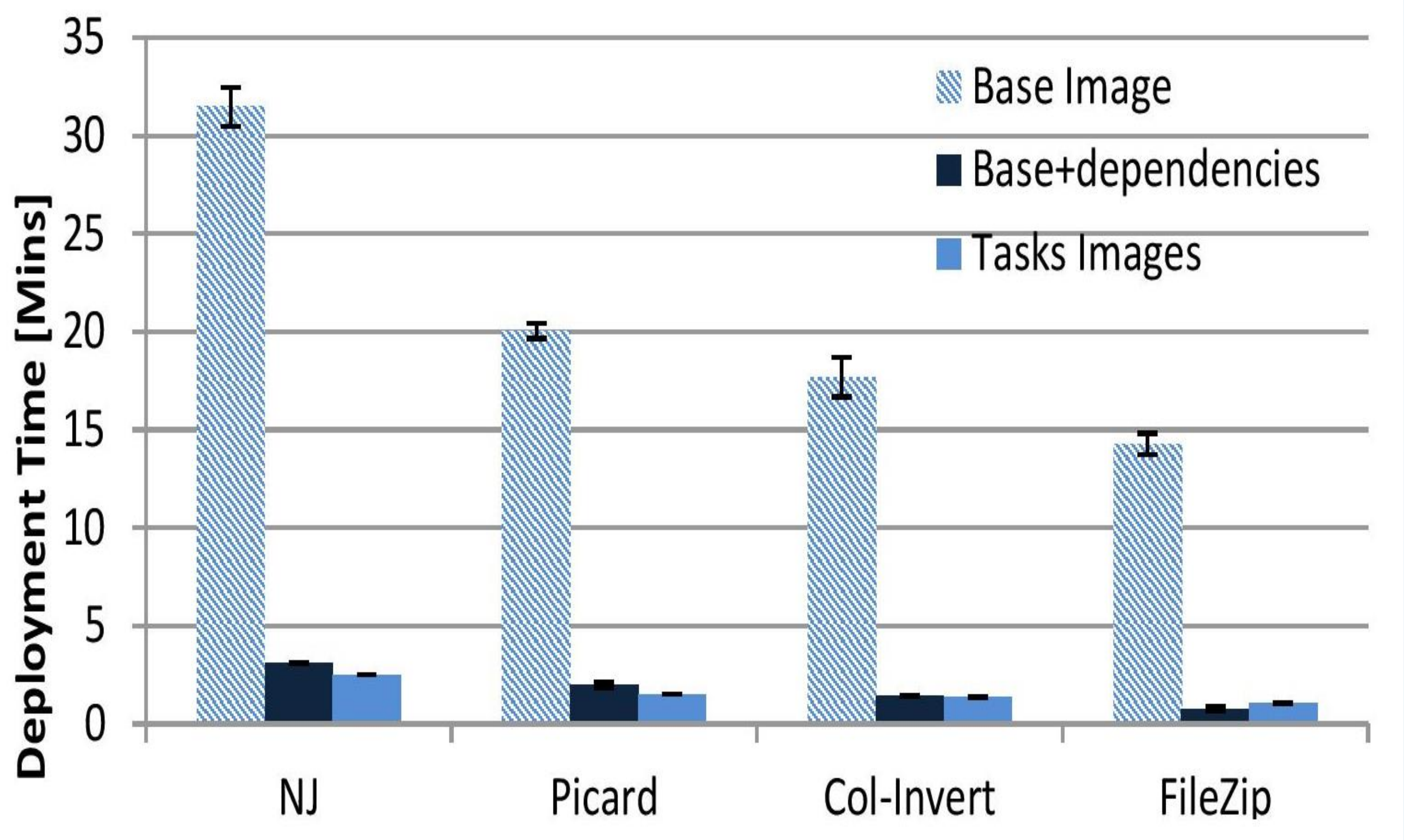
Preserving description, input data, tracking changes and deployment instructions

Experiments and Results

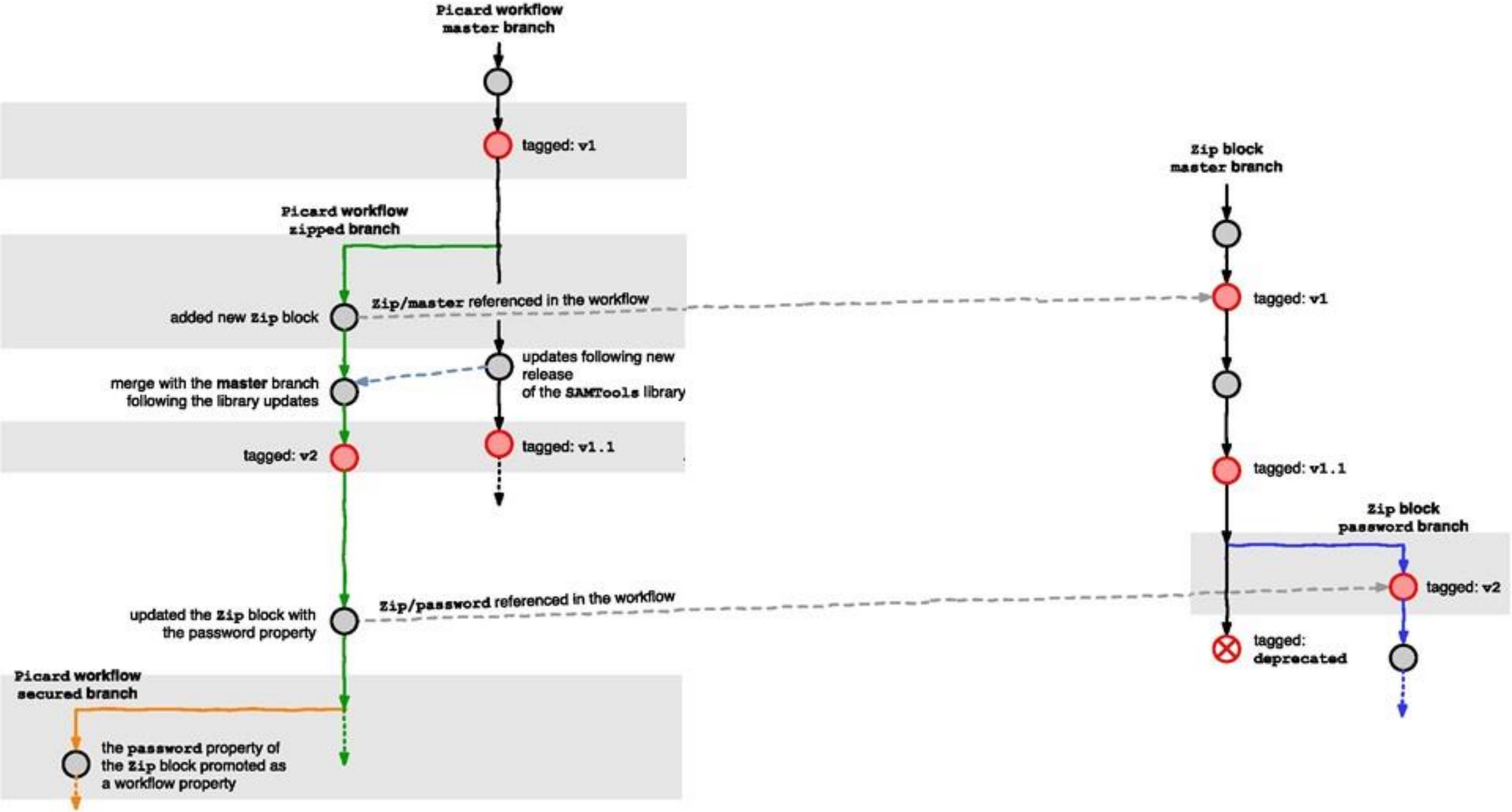
1- Repeatability of a workflow on different clouds



2- Automatic image capture for improved performance



3- Reproducibility in the face of development changes


Picard workflow master branch
tagged: v1
Zip block master branch
Picard workflow zipped branch
Zip/master referenced in the workflow
added new Zip block
tagged: v1
updates following new release of the SAMTools library
merge with the master branch following the library updates
tagged: v2
tagged: v1.1
tagged: v1.1
Zip block password branch
tagged: v2
Zip/password referenced in the workflow
updated the Zip block with the password property
tagged: deprecated
Picard workflow secured branch
the password property of the Zip block promoted as a workflow property

Conclusions

- Full workflow reproducibility is a long-standing issue
- TOSCA description is used for logical preservation
- Docker images for tasks/workflows support physical preservation
- Changes tracking and automatic deployment also contribute to a comprehensive solution of the problem
- Integration of these techniques addresses majority of the issues related to workflow decay

THANK YOU