CNPEM
Centro Nacional de Pesquisa em Energia e Materiais

TRAD2026

TREINAMENTO EM REPRODUTIBILIDADE PARA ANÁLISE DE DADOS

Módulo II: CONTAINERS

Controle de ambientes e reprodutibilidade

LNBio

lnbr

Sumário

1. Ambientes contidos e reprodutibilidade

O problema. O mesmo script pode gerar resultados diferentes — ou nem rodar — em outra máquina. As causas são recorrentes: sistemas operacionais e kernels distintos, conflitos de bibliotecas e versões (o *"inferno de dependências"*), instalações manuais frágeis e, no fim, ciência que não se reproduz. Em bioinformática isso é crítico: pipelines encadeiam dezenas de ferramentas, e um resultado só vale se puder ser repetido.

Ambiente contido. É empacotar tudo o que a análise precisa — SO base, bibliotecas, ferramentas e configurações — numa unidade isolada e portátil, carregada junto com o código. A ideia-síntese: **mesmo ambiente → mesmos resultados**, hoje, no cluster e daqui a cinco anos.

Os quatro pilares

Pilar	O que significa
Ambiente empacotado	SO, bibliotecas, versões e configurações numa única imagem.
Isolamento	O que roda dentro não interfere no host nem em outras ferramentas.
Imutabilidade	A imagem é somente-leitura: comporta-se de forma idêntica em qualquer lugar.
Proveniência	A receita da imagem é legível, versionável e auditável.

Níveis de reprodutibilidade

- **Nível 0 — README:** instruções manuais; quebram com facilidade.
- **Nível 1 — Gerenciador de ambiente:** conda/venv fixam versões de pacotes, mas dependem do SO do host.
- **Nível 2 — Container (.sif):** ambiente completo, isolado e portátil. Foco desta aula.
- **Nível 3 — Container + workflow versionado:** .sif com Snakemake/Nextflow, Git e arquivamento com DOI. Reprodutibilidade total.

2. O que são containers e sua finalidade

Definição. Um container empacota, numa única unidade, o código somado às dependências, bibliotecas e configurações do ambiente. Esse pacote roda de forma idêntica em qualquer máquina que tenha o runtime instalado — notebook, cluster HPC ou o computador de outro pesquisador.

> **Analogia do contêiner de carga:** navios, trens e caminhões transportam a mesma caixa metálica padronizada. Não importa o conteúdo — o transporte é sempre igual, porque a interface externa é padronizada. O software em container faz o mesmo: o ambiente viaja junto, empacotado.

Para que servem

- **Portabilidade —** mover a análise entre máquinas sem reinstalar nada.
- **Reprodutibilidade —** a mesma imagem gera os mesmos resultados ao longo do tempo.
- **Isolamento —** rodar ferramentas com dependências conflitantes lado a lado.
- **Compartilhamento —** distribuir um ambiente pronto a colegas, revisores ou como material suplementar.
- **Escalabilidade —** executar a mesma imagem em centenas de nós, com o mesmo comportamento.

3. O que é Singularity / Apptainer

O Singularity (e o sucessor **Apptainer**) é um sistema de containers projetado para computação científica e ambientes de HPC, onde o Docker tradicionalmente não se encaixa. Três características o tornam adequado à ciência:

- **Feito para HPC —** roda como usuário comum, sem daemon e sem privilégios de root.
- **Imagem = um único arquivo —** toda a imagem vira um arquivo .sif, fácil de mover, versionar e anexar a um artigo.
- **Integra com o cluster —** conversa naturalmente com SLURM, MPI e GPUs.

Singularity → Apptainer. Em 2021 o projeto de código aberto foi doado à Linux Foundation e renomeado para Apptainer. Na prática os comandos são quase idênticos: onde havia `singularity`, pode-se usar `apptainer`. Existe ainda a versão comercial SingularityCE (Sylabs), mas os conceitos valem para todos.

> **Em uma frase:** é a ferramenta de containers que roda com segurança em máquinas compartilhadas e produz uma imagem que é um único arquivo portátil.

4. Como um container funciona

Compartilhando o kernel. A forma mais clara de entender é comparar com uma máquina virtual (VM). A VM emula o hardware inteiro e carrega um SO completo; o container não virtualiza nada — compartilha o kernel do host e isola apenas um processo. Por isso é leve e rápido.

Aspecto	Máquina Virtual	Container
O que virtualiza	Hardware inteiro; SO completo por VM.	Nada — compartilha o kernel do host.
Peso	Vários GB; muita RAM/CPU.	Megabytes; pouco overhead.
Inicialização	Minutos.	Segundos.
Isolamento	Muito forte, porém caro.	Suficiente para uso científico, com custo baixo.

Imagem × instância

A **imagem** (.sif) é o pacote estático, somente-leitura e imutável. Ao executá-la, o runtime cria uma **instância** — um processo isolado. Como a imagem não muda, toda execução parte exatamente do mesmo estado.

Isolamento, dados e segurança

O container tem seu próprio sistema de arquivos, mas o Singularity/Apptainer **monta automaticamente** o `$HOME`, o `/tmp` e o diretório de trabalho atual — para outras pastas usa-se o **bind mount**. E o principal: um comando dentro do container roda **com as permissões do próprio usuário** que o iniciou, sem escalar privilégios. É isso que o torna seguro em clusters multiusuário.

5. Singularity × Docker

Ambos empacotam ambientes, mas com focos diferentes — não é rivalidade. O Docker brilha em microsserviços, aplicações web e CI/CD; o Singularity, em HPC e ciência reprodutível.

Aspecto	Docker	Singularity / Apptainer
Execução	Daemon em segundo plano, como root.	Sem daemon; roda como programa comum.
Privilégios	Exige sudo/admin — barrado em clusters.	Sem root; ideal para HPC.
Imagem	Camadas num store local.	Um único arquivo .sif.
Acesso a dados	Isolado por padrão.	Monta $HOME, /tmp e a pasta atual.

Compatibilidade. O mais importante: os dois se complementam. O Singularity roda imagens Docker diretamente de um registro, convertendo-as para .sif no ato:

```
singularity build bio.sif docker://ubuntu:22.04
```

Assim, todo o ecossistema Docker (incluindo os BioContainers) segue acessível. Na prática: Docker para desenvolvimento e CI; Singularity para executar em HPC.

6. Construindo seu próprio container

Vocabulário essencial

Termo	Significado
Receita (.def)	Arquivo de texto com as instruções de build — a "lista de ingredientes".
Imagem (.sif)	O pacote final: imutável, executável e autocontido.
Build	O processo que transforma a receita .def na imagem .sif.
Bootstrap / base	A imagem de origem de onde se parte (ex.: docker://ubuntu:22.04).
Bind mount	Montar pastas do host dentro do container para acessar dados.

Anatomia de um arquivo .def

- **Cabeçalho** (`Bootstrap: / From:`) — define a imagem base.
- `%files` — copia arquivos do host para a imagem, na construção.
- `%post` — comandos durante o build: instalar softwares e dependências.
- `%environment` — variáveis de ambiente definidas em tempo de execução.
- `%runscript` — o que roda ao chamar singularity run.
- `%labels` / `%help` — metadados e texto de ajuda.

Exemplo (samtools + bcftools):

```
Bootstrap: docker
From: ubuntu:22.04

%labels
    Author   SeuNome
    Version  1.0

%post
    apt-get update
    apt-get install -y samtools bcftools

%environment
    export LC_ALL=C

%runscript
    exec samtools "$@"
```

Conda isolado dentro do container

Padrão poderoso em bioinformática: o **container** garante o SO e a portabilidade; o **Conda** (canal Bioconda) instala centenas de ferramentas com versões exatas, sem compilar. O ambiente é descrito num `environment.yml` versionável e instalado na seção `%post`. O truque de "ativação": em vez de `conda`

`activate`, coloca-se o `bin` do ambiente no `PATH`, dentro de `%environment` — assim o ambiente já vem ativo.

Do .def ao .sif e execução

- **A partir da receita:** `sudo singularity build bio.sif bio.def`
- **Sem root, em HPC:** `singularity build --fakeroot bio.sif bio.def`
- **Direto de um registro:** `singularity build bio.sif docker://ubuntu:22.04`

Comando	O que faz
run	Executa o %runscript — o comando padrão da imagem.
exec	Roda um comando arbitrário dentro do container.
shell	Abre um shell interativo dentro do container.
--bind	Monta pastas do host (ex.: --bind /dados:/mnt).

> **Fluxo completo:** bio.def (você escreve) → build (constrói) → bio.sif (imagem pronta) → run/exec (roda a análise). Como o .def é pequeno e legível, versione-o no Git: a imagem pode ser reconstruída de forma idêntica sempre que necessário.

7. Baixando containers prontos

Quase nunca é preciso construir do zero — a comunidade já empacotou milhares de ferramentas. Principais fontes:

- **Docker Hub —** imagens gerais e de organizações (ex.: StaPH-B): `apptainer pull docker://staphb/samtools:1.21`
- **BioContainers (Quay.io) —** empacota automaticamente cada pacote do Bioconda. A tag (hash de build) muda a cada reconstrução; copie-a da aba Tags.
- **Galaxy Depot —** mantém os .sif já convertidos dos BioContainers, ideais para HPC (sem conversão local).

Seqera Containers — gerar sob demanda

A plataforma `seqera.io/containers` inverte a lógica: em vez de procurar imagens prontas, você escolhe pacotes Conda/PyPI numa interface web e ela constrói a imagem na hora (via serviço **Wave**), devolvendo uma URI Docker ou Singularity. É gratuito, dispensa escrever Dockerfile/.def, faz varredura de segurança (Trivy) e permite baixar o .sif pelo navegador. Para Singularity, a URI usa o protocolo **ORAS**.

> **Reprodutibilidade:** builds sob demanda são convenientes, mas para proveniência de longo prazo fixe a URI exata que usou e arquive o .sif.

pull × build. O `pull` baixa uma imagem pronta (convertendo de Docker para .sif quando necessário); o `build` também puxa de um registro, mas permite ir além, aplicando uma receita .def.

8. Boas práticas

- ☐ **Fixe as versões.** Escreva ubuntu:22.04 e samtools=1.21, nunca latest.
- ☐ **Documente a imagem.** Preencha %labels e %help.
- ☐ **Versione a receita.** Guarde o .def (e o environment.yml) no Git; o .sif é grande e não precisa ir ao repositório.
- ☐ **Teste antes de usar.** Rode singularity test / run-help e valide a ferramenta.
- ☐ **Prefira bases enxutas.** Imagens menores constroem mais rápido e têm menor superfície de vulnerabilidades.
- ☐ **Cuidado com bind mounts.** Monte só o necessário e, se possível, em somente-leitura (:ro).
- ☐ **Arquive para reprodutibilidade real.** Para publicação, deposite o .sif (ou a URI exata) num local persistente.

Glossário

Apptainer — nome atual do projeto Singularity de código aberto, sob a Linux Foundation desde 2021.

Bind mount — montagem de uma pasta do host em um caminho dentro do container.

Bioconda — canal do Conda especializado em ferramentas de bioinformática.

BioContainers — projeto que empacota pacotes Bioconda como imagens de container.

Bootstrap — diretiva do .def que define a origem da imagem base.

Container — pacote isolado e portátil com código, dependências e ambiente.

.def — arquivo-receita que descreve como construir uma imagem.

Daemon — serviço em segundo plano; o Docker usa um, o Singularity não.

HPC — High-Performance Computing; clusters de alto desempenho, tipicamente compartilhados.

Imagem — o pacote estático e imutável a partir do qual containers são executados.

Kernel — núcleo do sistema operacional; compartilhado entre host e container.

ORAS — protocolo para distribuir artefatos (como imagens .sif) em registros OCI.

.sif — Singularity Image Format; a imagem como um único arquivo.

Wave — serviço da Seqera que constrói imagens de container sob demanda.

Referências

Apptainer — Documentação: https://apptainer.org/docs/

Apptainer — Guia rápido: https://apptainer.org/docs/user/main/quick_start.html

Singularity - Documentação: https://docs.sylabs.io/guides/4.3/user-guide/

Singularity – Guia rápido: https://docs.sylabs.io/guides/4.3/user-guide/quick_start.html

BioContainers: https://biocontainers.pro/

Galaxy Project — depósito de imagens Singularity: https://depot.galaxyproject.org/singularity/

Seqera Containers: https://seqera.io/containers/

Conda / Bioconda: https://bioconda.github.io/

Este resumo condensa a teoria. Para a prática — instalar, baixar, construir e executar containers — siga o Roteiro Teórico-