TRAD2026

TREINAMENTO EM REPRODUTIBILIDADE PARA ANÁLISE DE DADOS

Módulo II: Containers

Controle de ambientes e reprodutibilidade

LNBio

lnbr

CNPEM Centro Nacional de Pesquisa em Energia e Materiais

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO

O que vamos ver

1. Ambientes contidos e reprodutibilidade
2. O que são containers e sua finalidade
3. O que é Singularity / Apptainer
4. Como um container funciona
5. Singularity × Docker
6. Construindo seu próprio container
7. Baixando containers prontos
8. Boas práticas

Objetivo de aprendizado

Ao final deste módulo, espera-se que os participantes sejam capazes de:

1. **Explicar reprodutibilidade computacional e por que ela é um problema científico, não apenas técnico.**
2. **Diferenciar container, máquina virtual e ambiente Conda**, sabendo quando cada um é a escolha adequada.
3. **Justificar o uso de Singularity/Apptainer em HPC**, entendendo as limitações do Docker em cluster (root, integração com escalonador).
4. **Obter imagens de repositórios públicos (Docker Hub, BioContainers, quay.io) e converter para o formato SIF.**
5. **Executar containers com `run`, `exec` e `shell`**, compreendendo a diferença prática entre os três.
6. **Controlar bind mounts e variáveis de ambiente**, incluindo `--bind`, `--cleanenv` e `--containall`.
7. **Escrever uma definition file completa**, com `%post`, `%environment`, `%runscript` e `%labels` (e `%apprun` quando usar SCIF).
8. **Construir a própria imagem**, escolhendo a estratégia viável no seu ambiente (root local, `--fakeroot` ou build remoto).

01

TÓPICO 01

Ambientes contidos e reprodutibilidade

Por que empacotar o ambiente — e o que significa uma análise reprodutível.

“...mas funciona na minha máquina.”

O mesmo script, resultados diferentes — ou erro que não roda em lugar nenhum.

Sistemas diferentes

Seu notebook, o cluster e o revisor rodam SOs e versões de kernel distintos.

conflito de dependências

Bibliotecas, compiladores e versões de pacotes que conflitam entre si.

Instalação frágil

Horas configurando ferramentas que quebram na próxima atualização.

Ciência não reproduzível

Sem o ambiente exato, ninguém consegue repetir sua análise anos depois.

CONCEITO CENTRAL

Reprodutibilidade e controle de ambiente

Ambiente empacotado

SO, bibliotecas, versões e configurações reunidos numa única imagem. Fim do "instalar tudo na mão".

Isolamento

O que roda dentro não interfere no host nem em outras ferramentas. Acabam os conflitos de versão.

Imutabilidade

O .sif é somente-leitura: a mesma imagem se comporta de forma idêntica em qualquer máquina e no futuro.

Proveniência

O .def registra exatamente como o ambiente foi construído. Legível, versionável no Git, auditável e citável.

A ideia central: mesmo ambiente → mesmos resultados. Hoje, no cluster e daqui a cinco anos. Isso é reprodutibilidade.

Níveis de reprodutibilidade

Cada nível empacota mais do ambiente — e deixa a sua análise mais fácil de repetir.

▲ mais robusto e reprodutível

README

Instruções manuais. Depende de instalar tudo na mão — quebra fácil.

Ambiente (conda)

conda/venv fixam pacotes, mas ainda dependem do SO do host.

Container .sif

Ambiente completo, isolado e portátil. O foco desta aula.

Container + workflow

.sif + Snakemake/Nextflow + Git + DOI. Reprodutibilidade total.

02

TÓPICO 02

O que são containers e sua finalidade?

A ideia central de um container e para que ele serve.

O que é um container?

Um pacote isolado e portátil

Um container empacota, em uma única unidade:

o código + as dependências + bibliotecas + as configurações do ambiente.

Esse pacote roda de forma **idêntica** em qualquer máquina que tenha o runtime — seu notebook, o cluster HPC ou o computador de outro pesquisador.

A analogia do contêiner de carga

- Navios, trens e caminhões movem a mesma caixa padronizada.
- Não importa o que há dentro — o transporte é sempre igual.
- Software em container é a mesma ideia: o "ambiente" viaja junto, empacotado e padronizado.

03

TÓPICO 03

O que é Singularity / Apptainer?

A ferramenta de containers feita para a computação científica.

Por que Singularity / Apptainer?

Feito para HPC

Roda como usuário comum, sem daemon e sem privilégios de root — o que o Docker exige e clusters compartilhados não permitem.

Imagem = um arquivo

Toda a imagem vira um único arquivo .sif: fácil de mover, versionar, arquivar e anexar a um artigo.

Integra com o cluster

Conversa naturalmente com SLURM, MPI e GPUs, encaixando nos fluxos de trabalho de bioinformática.

Singularity → Apptainer: em 2021 o projeto foi doado à Linux Foundation e renomeado para Apptainer. Os comandos são praticamente idênticos — troque `singularity` por `apptainer`.

04

TÓPICO 04

Como um container funciona?

Compartilhando o kernel do host — sem virtualizar o hardware inteiro.

Container × Máquina Virtual

Máquina Virtual

- ✗ **Virtualiza o hardware** — Cada VM carrega um sistema operacional completo.
- ✗ **Pesada** — Imagens de vários gigabytes; consome muita RAM/CPU.
- ✗ **Boot lento** — Iniciar leva minutos, como ligar outro computador.
- ✗ **Forte isolamento** — Ótima separação, mas com alto custo de recursos.

Container

- ✓ **Compartilha o kernel** — Usa o SO do host; empacota só o necessário.
- ✓ **Leve** — Imagens de megabytes; pouco overhead de recursos.
- ✓ **Início instantâneo** — Sobe em segundos, como abrir um programa.
- ✓ **Ideal p/ HPC & ciência** — Portátil, versionável e fácil de compartilhar.

05

TÓPICO 05

Singularity × Docker

Duas tecnologias de container com focos diferentes.

Singularity / Apptainer × Docker

Docker

padrão na indústria e em CI/CD

- **Daemon como root** — Um serviço em segundo plano com privilégios de root.
- **Precisa de admin** — Exige sudo — barrado em clusters HPC compartilhados.
- **Imagem em camadas** — Fica num store local; portar exige export/save.
- **Isolado por padrão** — Não vê seu $HOME e dados sem configuração extra.

Singularity

feito para HPC e ciência

- **Sem daemon, sem root** — Roda como você mesmo, como um programa comum.
- **Ideal para HPC** — Sem admin; integra com SLURM, MPI e GPUs.
- **Um único arquivo .sif** — Portátil: mover, versionar, arquivar e citar.
- **Acessa seus dados** — Monta $HOME, /tmp e a pasta atual automaticamente.

Sem rivalidade: o Singularity roda imagens Docker direto do registry (`singularity build bio.sif docker://…`). Docker brilha em microsserviços e CI; Singularity, em HPC e ciência reprodutível.

06

TÓPICO 06

Construindo seu próprio container

Da receita .def até a imagem .sif — e como executá-la.

Cinco palavras que você vai ouvir

Receita (.def)	Arquivo de texto com as instruções de como construir a imagem — a "lista de ingredientes".
Imagem (.sif)	O pacote final: imutável, executável e autocontido. É o que você roda e compartilha.
Build	O processo que transforma a receita .def na imagem .sif.
Bootstrap / base	A imagem de origem de onde você parte (ex.: Docker Hub, ubuntu:22.04).
Bind mount	Montar pastas do host dentro do container para ler e gravar seus dados.

Anatomia de um arquivo .def

Cabeçalho

```
Bootstrap: docker
From: ubuntu:22.04
```

Define a imagem base: de onde partir. "docker" busca no Docker Hub.

Seções começam com `%` e agrupam instruções por finalidade.

`%files`	Copia arquivos do host para dentro da imagem.
`%post`	Comandos executados na construção: instalar softwares e dependências.
`%environment`	Variáveis de ambiente definidas em tempo de execução.
`%runscript`	O que roda ao chamar singularity run.
`%labels`	Metadados: autor, versão, descrição.
`%help`	Texto de ajuda exibido por singularity run-help.

Uma receita .def completa

```
Bootstrap: docker
From: ubuntu:22.04

%labels
    Author   Joaquim
    Version  1.0

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

%environment
    export LC_ALL=C

%runscript
    exec samtools "$@"

%help
    Container com samtools + bcftools.
    Uso: singularity run bio.sif view in.bam
```

Lendo de cima para baixo

1. Parte da imagem base Ubuntu 22.04, obtida do Docker Hub.
2. Registra metadados de autor e versão em %labels.
3. Instala as ferramentas em %post (roda no build).
4. Fixa o locale em %environment para a execução.
5. Define samtools como comando padrão em %runscript.
6. Documenta o uso em %help.

Ambientes conda isolados no container

```
Bootstrap: docker
From: ubuntu:22.04

%files
    environment.yml /opt/env.yml

%post
    apt-get update && apt-get install -y wget
    wget -qO /tmp/mf.sh <URL-do-Miniforge>
    bash /tmp/mf.sh -b -p /opt/conda
    /opt/conda/bin/mamba env create -f /opt/env.yml
    /opt/conda/bin/mamba clean -a -y

%environment
    export PATH=/opt/conda/envs/bioenv/bin:$PATH
%runscript
    exec "$@"
```

1 · A lista de versões — environment.yml

```
name: bioenv
channels: [conda-forge, bioconda]
dependencies:
  - samtools=1.21
  - bcftools=1.21
```

2 · Executar a partir do container

```
$ apptainer exec bioenv.sif samtools --version
$ apptainer run  bioenv.sif view in.bam
```

O truque: o %environment coloca o env no PATH — nada de "conda activate". Cada exec/run já usa as versões fixadas.

Gerando a imagem .sif

`singularity build` `<saída.sif>` `<receita.def>`

A partir da receita (.def) *o caminho principal desta aula*

```
$ sudo singularity build bio.sif bio.def
```

Sem root, em cluster HPC (--fakeroot) *build como usuário comum, sem admin*

```
$ singularity build --fakeroot bio.sif bio.def
```

Direto de um registry, sem .def *puxa e converte uma imagem Docker pronta*

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

Rodando o container: run, exec, shell

run

Executa o %runscript — o comando padrão da imagem.

```
$ singularity run bio.sif view in.bam
```

exec

Roda um comando arbitrário dentro do container.

```
$ singularity exec bio.sif bcftools --version
```

shell

Abre um shell interativo dentro do container.

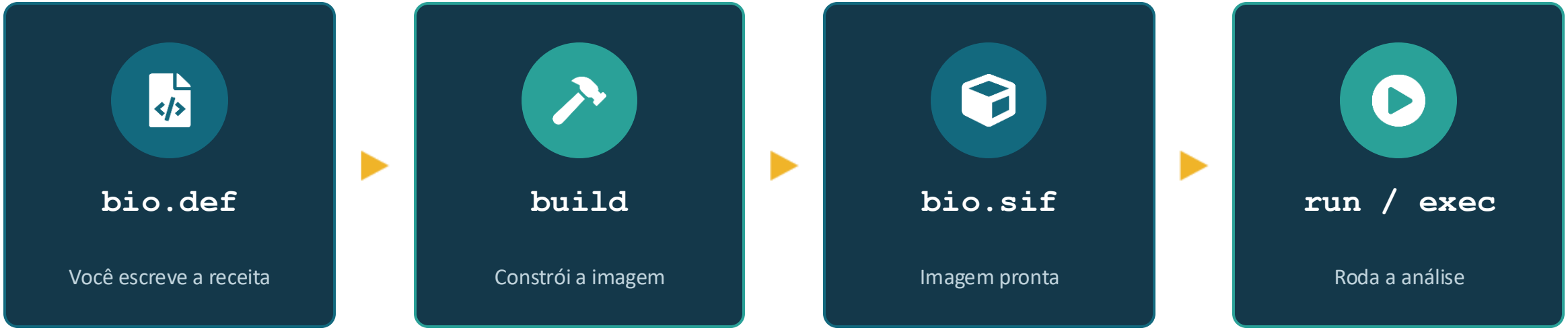
```
$ singularity shell bio.sif
```

--bind

Monta pastas do host para acessar seus dados.

```
$ singularity exec --bind /dados:/mnt bio.sif ...
```

O fluxo de trabalho completo



Versione o .def no Git. A receita é pequena e legível — reconstrói a imagem sempre que precisar, de forma idêntica.

07

TÓPICO 07

Baixando containers prontos

Reaproveite imagens já empacotadas por repositórios públicos.

Baixando containers prontos

Você quase nunca precisa construir do zero — a comunidade já empacotou milhares de ferramentas.

Docker Hub *imagens gerais e de organizações (ex.: StaPH-B)*

```
$ apptainer pull docker://staphb/samtools:1.21
```

BioContainers · Quay.io *todo pacote Bioconda vira uma imagem automaticamente*

```
$ apptainer pull docker://quay.io/biocontainers/bcftools:1.21--h8b25389_0
```

Galaxy Depot *.sif já convertido, ideal para HPC (sem conversão local)*

```
$ apptainer pull https://depot.galaxyproject.org/singularity/samtools:1.21--h50ea8bc_0
```

Dica: a tag exata (o "--hash") muda a cada build — copie-a da aba Tags no Quay.io ou da listagem do Galaxy Depot.

Gerar containers sob demanda

Escolha pacotes Conda/PyPI numa interface web e a plataforma constrói a imagem na hora (via Wave).

1. **Escolha os pacotes**
   ex.: samtools + bcftools — vão para a mesma imagem
2. **Selecione o formato**
   Singularity + linux/amd64 nas configurações
3. **Get Container**
   copie a URI da imagem que a plataforma devolve
4. **Puxe no seu computador**
   via oras:// (ou baixe o .sif direto pelo navegador)

No seu computador

```
$ apptainer pull \
    oras://community.wave
    .seqera.io/library/
    samtools:<id>
```

Sem escrever Dockerfile nem .def. Gratuito, com varredura de segurança. Devolve imagem Docker ou Singularity.

⚠ **Reprodutibilidade:** fixe a URI exata e arquive o .sif.

08

TÓPICO 08

Boas práticas

Recomendações para containers realmente reprodutíveis.

Boas práticas

Fixe as versões

Escreva ubuntu:22.04 e samtools=1.21, nunca latest — reprodutibilidade depende disso.

Documente a imagem

Preencha %labels e %help: seu "eu" do futuro e seus colegas vão agradecer.

Versione a receita

Guarde o .def no Git. O .sif é grande; a receita é pequena e reconstrói tudo.

Teste antes de usar

Rode singularity test / run-help e valide a ferramenta antes da produção.

Prefira bases enxutas

Imagens menores constroem mais rápido e têm menos vulnerabilidades.

Cuidado com bind mounts

Monte apenas as pastas necessárias; evite expor dados sensíveis.

Resumo em cinco pontos

1. Container = ambiente completo (código + dependências + configuração) num pacote portátil que roda igual em qualquer lugar.
2. Reprodutibilidade = imagem imutável + receita .def versionável → os mesmos resultados hoje e no futuro.
3. Singularity roda sem root e sem daemon; por isso funciona em HPC, onde o Docker esbarra — mas ainda roda imagens Docker.
4. Baixe imagens prontas (Docker Hub, BioContainers, Galaxy, Seqera) ou construa a sua com singularity build a partir do .def.
5. run, exec e shell executam; use conda dentro do container, fixe versões e versione o .def no Git.

OBRIGADO

Joaquim Martins Jr
joaquim.junior@lnbr.cnpem.br
@jmartinsjrbr

Inscreva-se para
receber comunicados
sobre o CNPEM
e suas unidades

cnpem.br

Laboratório Nacional de Biociências

João V. S. Guerra (EDB)
José G. C. Pereira (EDB)
Nilson A. R. Coimbra (MAS)
Rick H. Hokama (EDB)

Laboratório Nacional de Biorrenováveis

Joaquim Martins Jr (Ômicas Integrativas)
Monyque K. P. Silva (Ômicas Integrativas)

CNPEM
MINISTÉRIO DA
CIÊNCIA, TECNOLOGIA
E INOVAÇÃO

Responda sua auto-avaliação