Apptainer

Secure, portable, and easy-to-use container system

What it does?

- simplifies the creation and execution of containers, ensuring software components are encapsulated for portability and reproducibility.
- allows unprivileged users to use containers and prohibits privilege escalation within the container; users are the same inside and outside the container.
- can encrypt containers and integrates with Vault and other secret management platforms to secure applications, models, and data.
- can import any container from Open Containers Initiative registries. It aims for maximum compatibility with Docker, allowing you to pull, run, and build from most containers on Docker Hub without changes.

The single-file SIF container format allows you to reproducibly build, share, and archive your workload from workstations to HPC to the edge.

Most useful commands

cache	Manage the local cache
capability	Manage Linux capabilities for users and groups
config	Manage various Apptainer configuration (root user only)

exec	Run a command within a container
instance	Manage containers running as services
run	Run the user-defined default command within a container
shell	Run a shell within a container
test	Run the user-defined tests within a container

build	Build an Apptainer image
delete	Deletes image from the library
pull	Pull an image from a URI
push	Upload image to the provided URI
sif	Manipulate Singularity Image Format (SIF) images

How to run Apptainer on PLGRID?

Very easy, it’s installed on all computing nodes :))
(It is not available on login node)

```
srun --time=2:00:00 --mem=1G --ntasks 1 --partition=plgrid-now
--account=plglscclass24-cpu --pty /bin/bash
apptainer version
1.2.3-1.el8
```

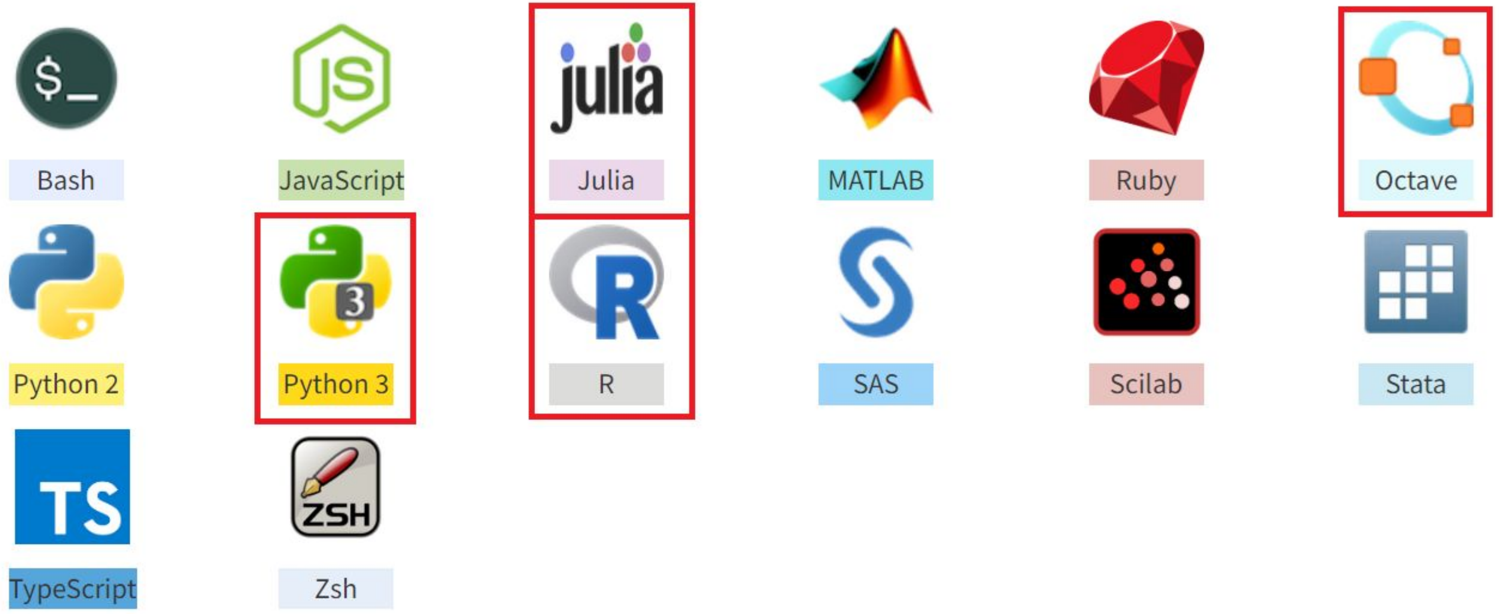
We can even run container with Jupyter Notebook and SOS kernel:

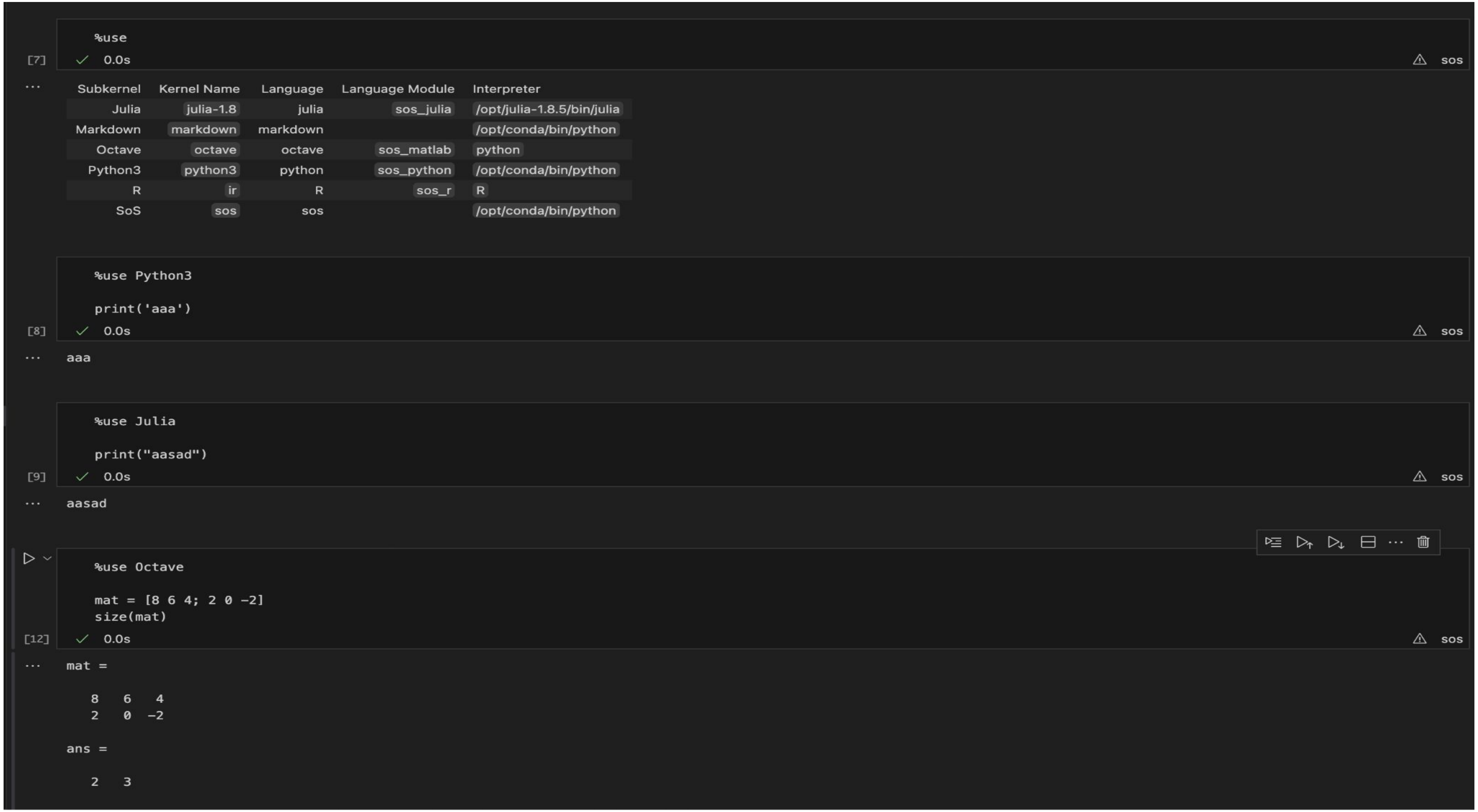
```
apptainer pull docker://vatlab/sos-notebook
apptainer run sos-notebook_latest.sif
```

It may take some time unfortunately to build the image (about half an hour) :(((

How Script of Scripts works?

It enables running a single notebook with multiple language cells inside. SOS kernel contains all known Jupyter kernels as subkernels:





```
%use
```
[7] ✓ 0.0s ⚠ sos

Subkernel	Kernel Name	Language	Language Module	Interpreter
Julia	julia-1.8	julia	sos_julia	/opt/julia-1.8.5/bin/julia
Markdown	markdown	markdown		/opt/conda/bin/python
Octave	octave	octave	sos_matlab	python
Python3	python3	python	sos_python	/opt/conda/bin/python
R	ir	R	sos_r	R
SoS	sos	sos		/opt/conda/bin/python

```
%use Python3

print('aaa')
```
[8] ✓ 0.0s ⚠ sos

```
aaa
```

```
%use Julia

print("aasad")
```
[9] ✓ 0.0s ⚠ sos

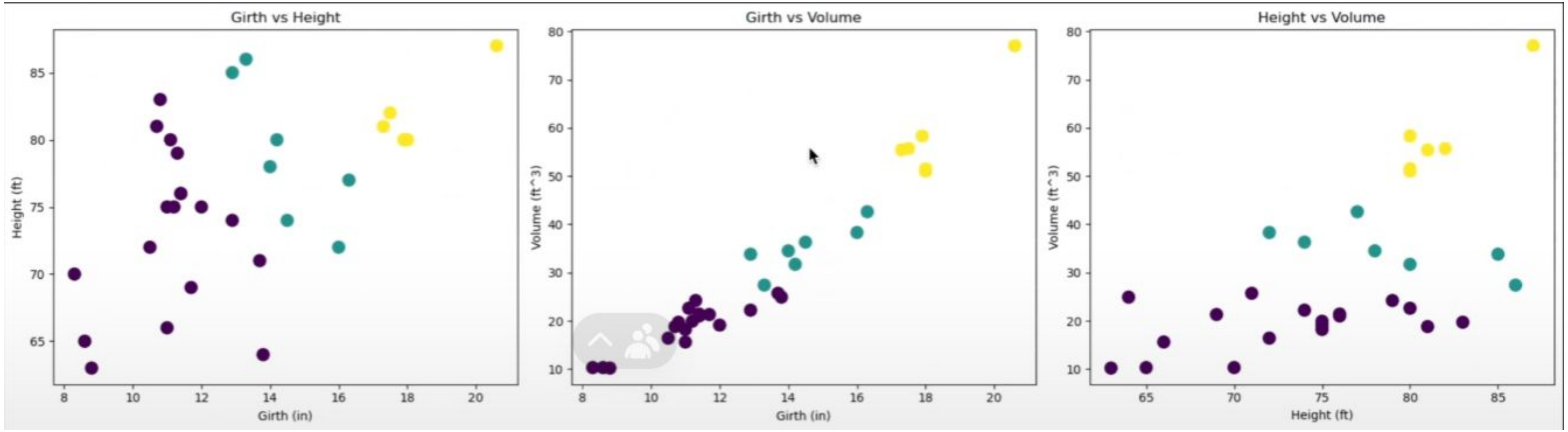
```
aasad
```

```
%use Octave

mat = [8 6 4; 2 0 -2]
size(mat)
```
[12] ✓ 0.0s ⚠ sos

```
mat =

   8   6   4
   2   0  -2

ans =

   2   3
```

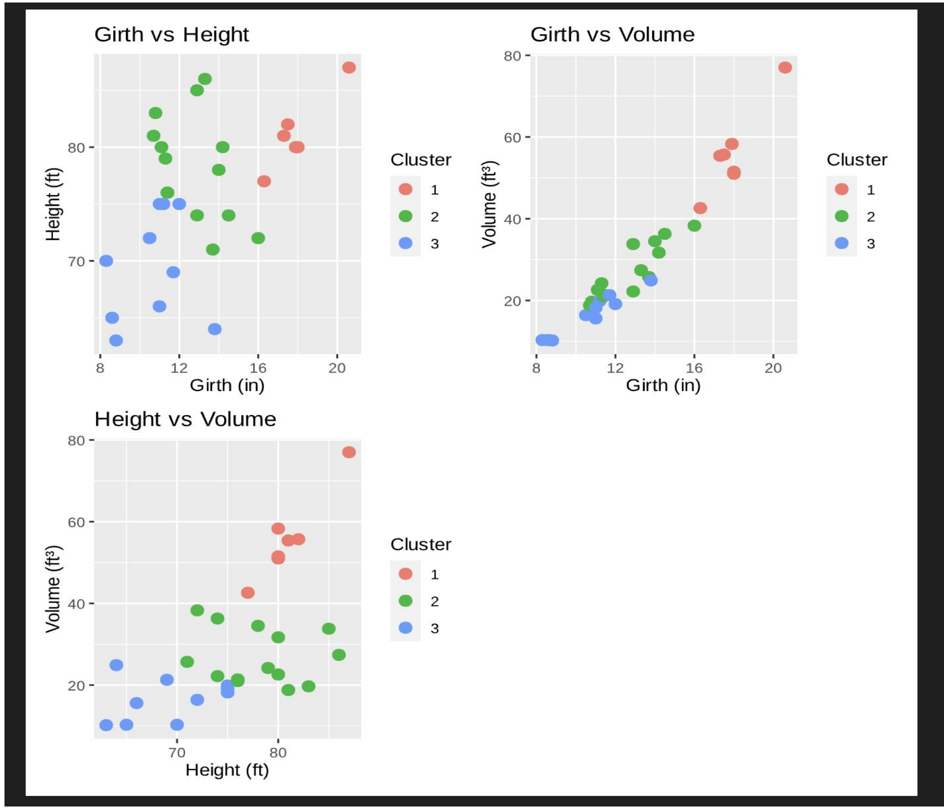
Maybe some more advanced computations - Python

```
df = pd.read_csv("trees.csv")
features = df[["Girth.in.", "Height.ft.", "Volume.ft3."]]
kmeans = KMeans(n_clusters=3, random_state=42)
df["Cluster"] = kmeans.fit_predict(features)
fig, axes = plt.subplots(1, 3, figsize=(18, 5), sharey=False)

axes[0].scatter(df["Girth.in."], df["Height.ft."], c=df["Cluster"], cmap="viridis", s=100)
axes[1].scatter(df["Girth.in."], df["Volume.ft3."], c=df["Cluster"], cmap="viridis", s=100)
axes[2].scatter(df["Height.ft."], df["Volume.ft3."], c=df["Cluster"], cmap="viridis", s=100)

plt.show()
```



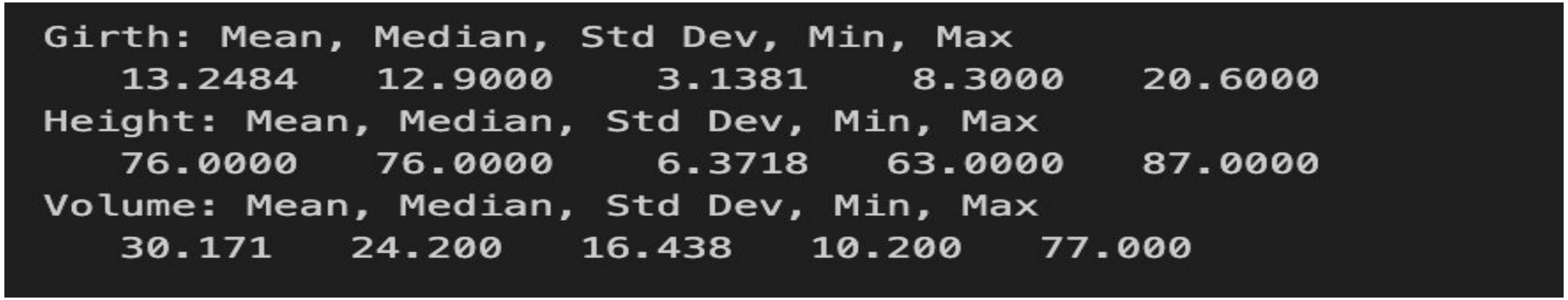
Maybe some more advanced computations - R

```
df <- read.csv("trees.csv")

features <- df[, c("Girth.in.", "Height.ft.",
"Volume.ft3.")]
set.seed(42)

clusters <- kmeans(features, centers = 3)

df$Cluster <- as.factor(clusters$cluster)

ggplot(df,
aes(x = Girth.in.,y = Height.ft., color = Cluster))
+ geom_point(size = 3) + ggtitle("Girth vs Height")
+ xlab("Girth (in)") + ylab("Height (ft)")
```



Maybe some more advanced computations - Octave

```
data = csvread(trees.csv', 1, 0);

girth = data(:, 2);  % Girth (inches)
height = data(:, 3);  % Height (feet)
volume = data(:, 4);  % Volume (cubic feet)

mean_girth = mean(girth);
median_girth = median(girth);
std_girth = std(girth);
min_girth = min(girth);
max_girth = max(girth);

disp("Girth: Mean, Median, Std Dev, Min, Max");
disp([mean_girth, median_girth, std_girth, min_girth, max_girth]);
```



```
Girth: Mean, Median, Std Dev, Min, Max
   13.2484   12.9000    3.1381    8.3000   20.6000
Height: Mean, Median, Std Dev, Min, Max
   76.0000   76.0000    6.3718   63.0000   87.0000
Volume: Mean, Median, Std Dev, Min, Max
   30.171   24.200   16.438   10.200   77.000
```

How to run our example? Very simple:

1. Connect to your favourite LSC grid:

```
srun --time=2:00:00 --mem=1G --ntasks 1 --partition=plgrid-now
--account=plglscclass24-cpu --pty /bin/bash
```

2. Run container using SIF image:

```
mkdir lsc-proj && cd lsc-proj
apptainer pull docker://vatlab/sos-notebook
apptainer run sos-notebook_latest.sif
```

3. Copy CSV and Notebook files to directory from which you run container:

```
scp ./trees.csv ARES_ADDRESS:~/lsc-proj/trees.csv
scp ./demo-notebook.ipynb ARES_ADDRESS:~/lsc-proj/demo-notebook.ipynb
```

4. Connect to notebook using URL printed by container logs and choose SoS kernel in VSC

Apptainer — summary

Pros:

- rootless execution
- isolation mechanism (cluster safety)
- natural compatibility with SLURM
- easy to use: portable image format (SIF)
- direct access to resources (as opposed to Docker)

Cons:

- lacks support for microservice models and large-scale orchestration (e.g., Kubernetes)
- little technical support and community
- no network isolation

Thanks for your attention

Bartłomiej Chwast, Jakub Strojewski, Michał Wójcik