Running Parallel, Distributed ROOT Analysis with PyRDF on Public Cloud

AWS Lambda Case Study

26 Jan 2021

Jacek Kuśnierz
Piotr Pasternak
Maciej Malawski

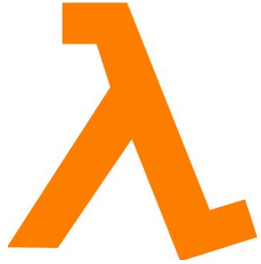
AGH University of Science and Technology
Kraków, Poland

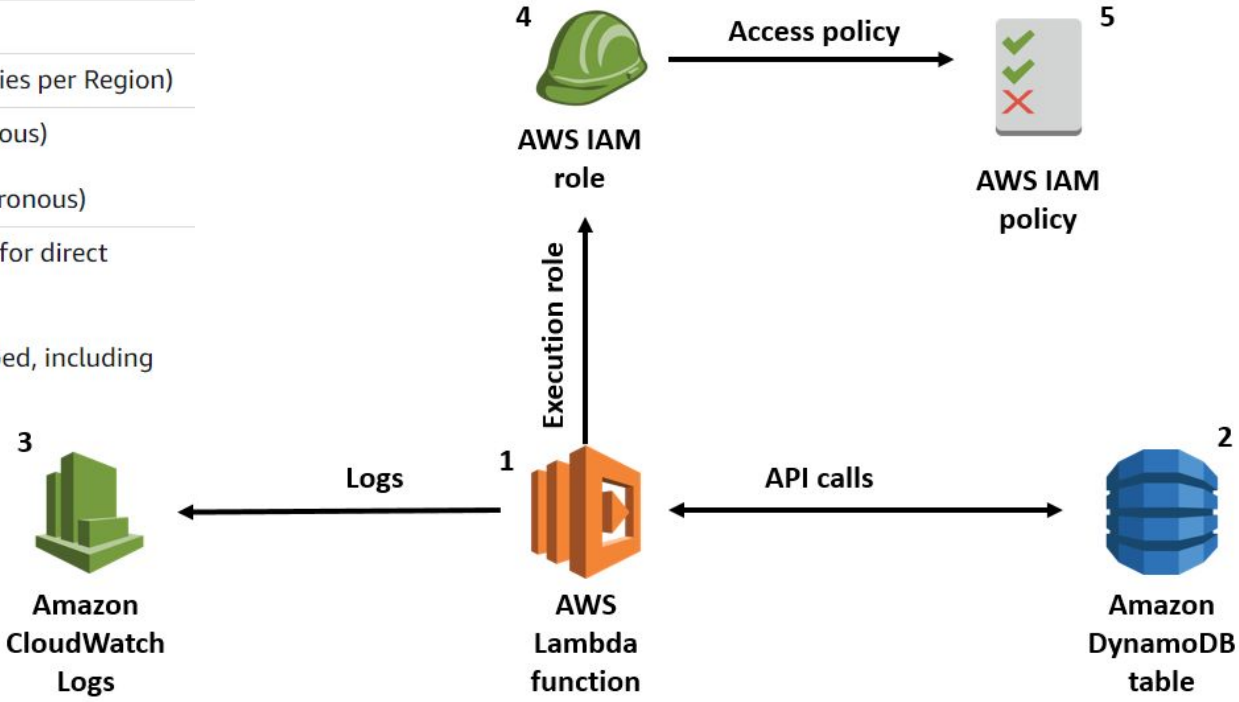
Presentation plan

1. Serverless, Lambda, ROOT and PyRDF
2. Project Structure Overview
3. Using the System
4. Technical Highlights

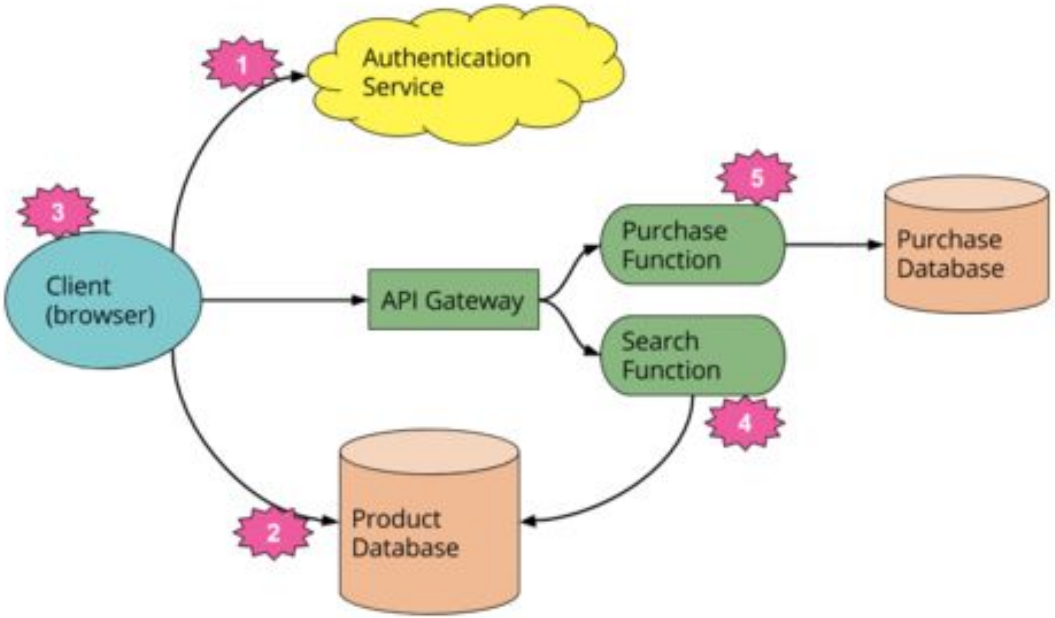
Function memory allocation	128 MB to 10,240 MB, in 1-MB increments.
Function timeout	900 seconds (15 minutes)
Function environment variables	4 KB
Function resource-based policy	20 KB
Function layers	5 layers
Function burst concurrency	500 - 3000 (varies per Region)
Invocation payload (request and response)	6 MB (synchronous) 256 KB (asynchronous)
Deployment package (.zip file archive) size	50 MB (zipped, for direct upload) 250 MB (unzipped, including layers)

AWS Lambda



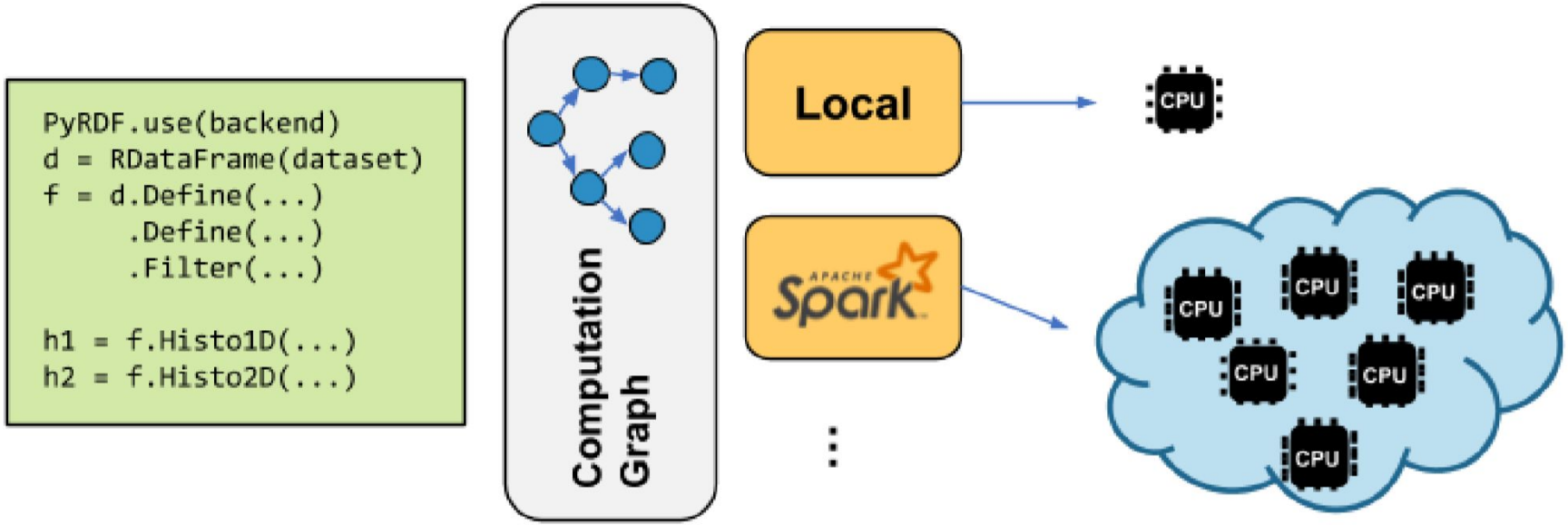


Serverless


1
Authentication Service
3
Client (browser)
API Gateway
5
Purchase Function
Purchase Database
Search Function
4
2
Product Database

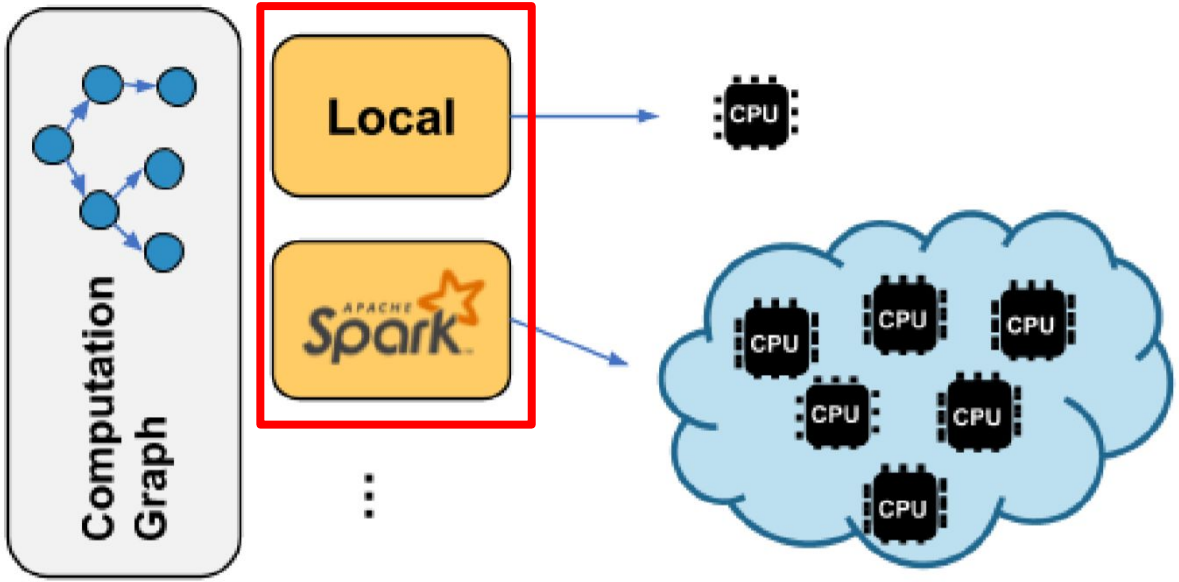
ROOT

PyRDF

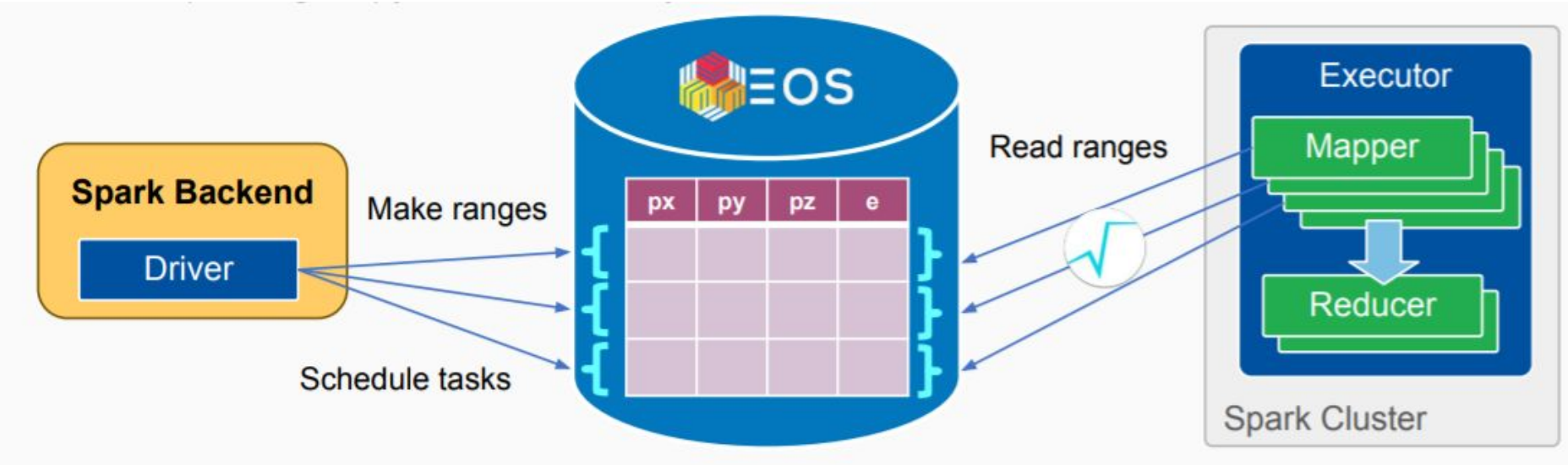

PyRDF.use(backend)
d = RDataFrame(dataset)
f = d.Define(...)
.Define(...)
.Filter(...)
h1 = f.Histo1D(...)
h2 = f.Histo2D(...)
Computation Graph
Local
APACHE Spark
...
CPU
CPU
CPU
CPU
CPU
CPU
CPU

PyRDF

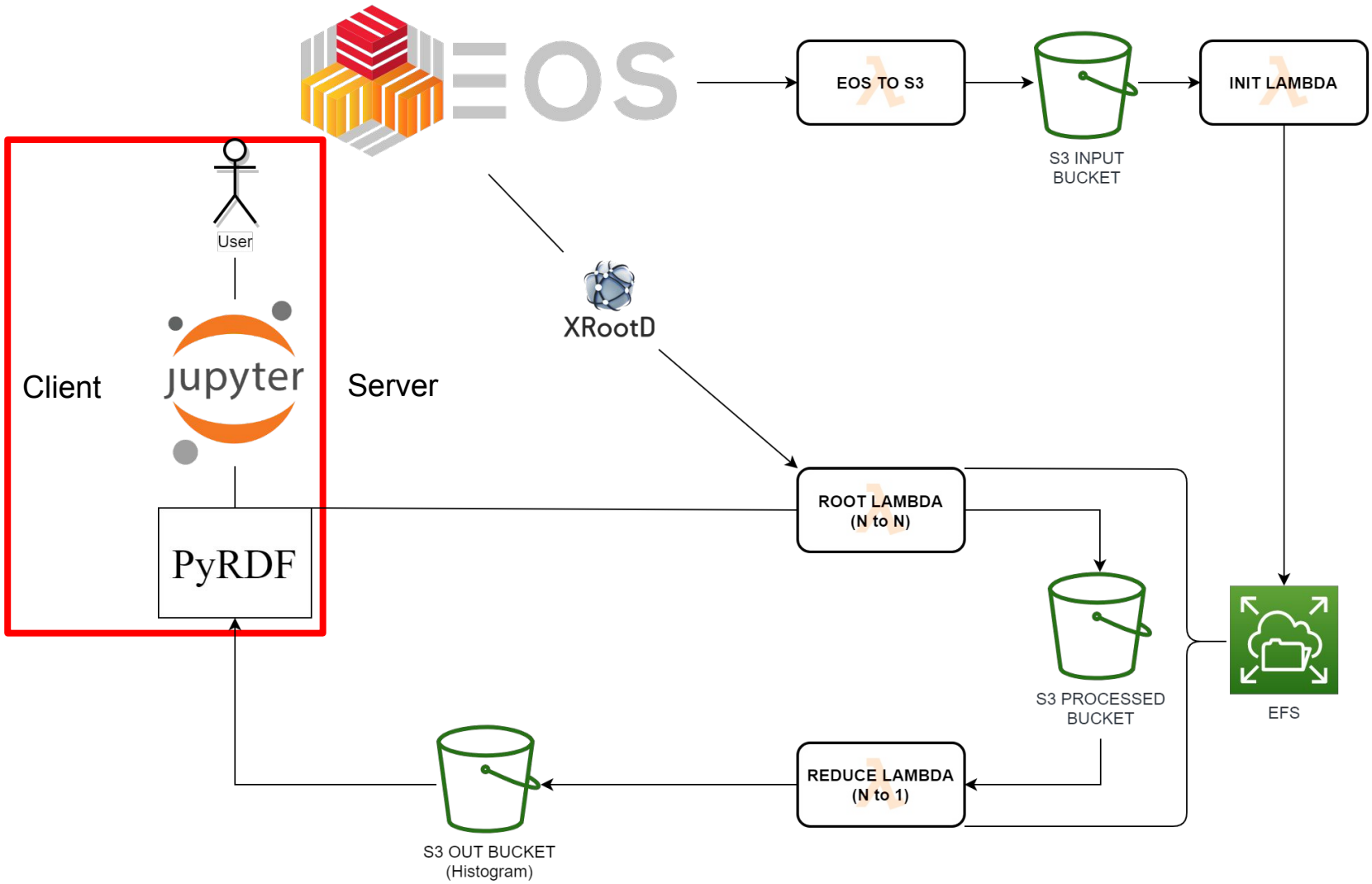
```
PyRDF.use(backend)
d = RDataFrame(dataset)
f = d.Define(...)
      .Define(...)
      .Filter(...)

h1 = f.Histo1D(...)
h2 = f.Histo2D(...)
```



PyRDF


Spark Backend
Driver
Make ranges
Schedule tasks
EOS
px
py
pz
e
Read ranges
Executor
Mapper
Reducer
Spark Cluster

Structure Overview


EOS
EOS TO S3
S3 INPUT BUCKET
INIT LAMBDA
User
XRootD
Client
jupyter
Server
PyRDF
ROOT LAMBDA (N to N)
S3 PROCESSED BUCKET
EFS
REDUCE LAMBDA (N to 1)
S3 OUT BUCKET (Histogram)

Using the System

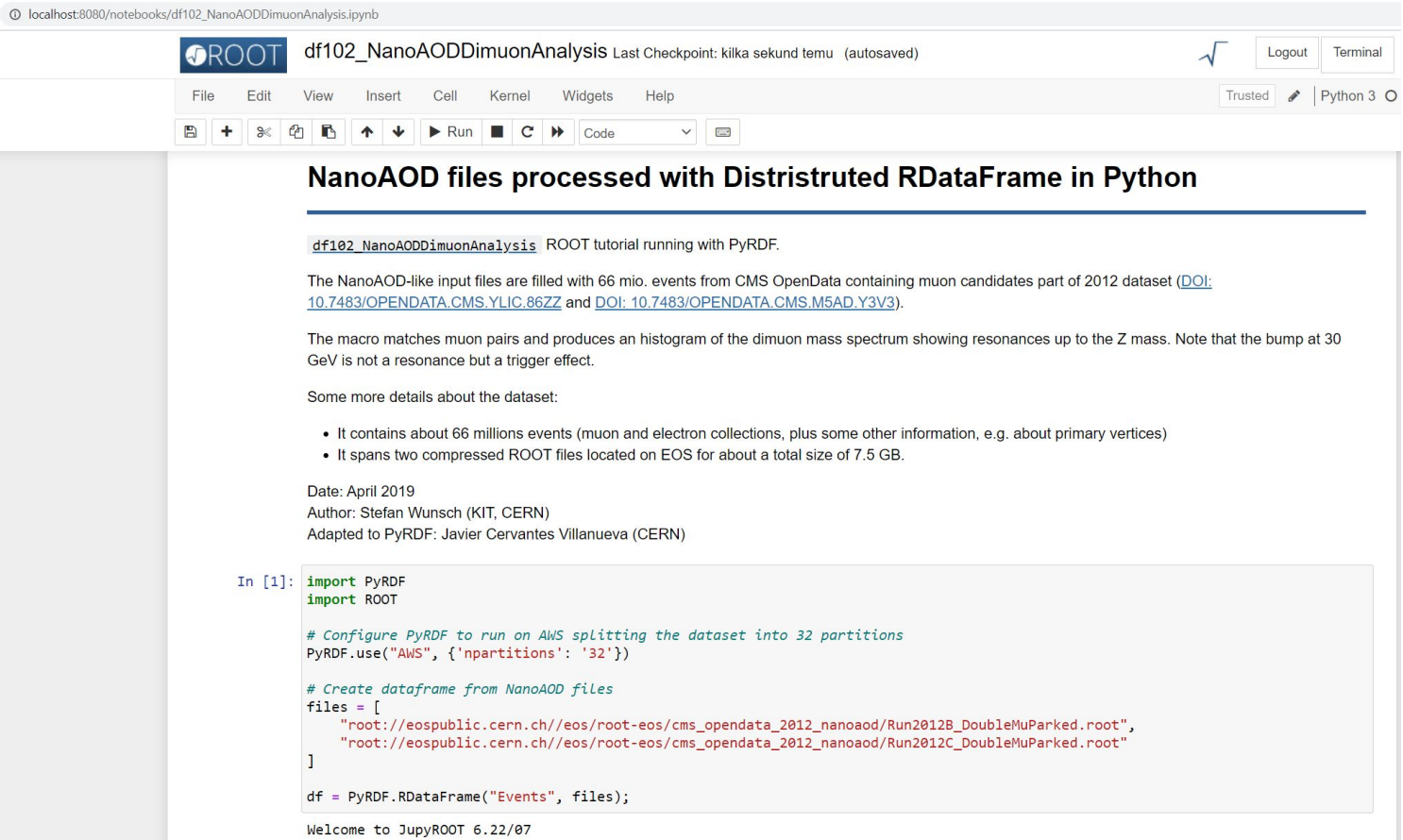
Using the system - create infrastructure

```
bash-4.2# cat ~/.aws/credentials
[default]
aws_access_key_id=ASIAUAHWLMWXWGET2WOG
aws_secret_access_key=TaN2q0ZoKrIJMxiie9fBcdOlyLheohb+q3Mhy9jf
aws_session_token=FwoGZXIvYXdzEE0aDN2AjoxtMCvWnIR6XyLLATNmfwVT/HvTydT/H4crFiFNZ4/7B7SGU91uksy7jwCSY5MBq1Ku9p3+tmH1VX+fv5
VuRWvDjL9tiTOkbwR3Db7c4AMFLimPm+D4ccg6Qd/mGsDzKRX8t/fyuhPEAcx6HueaxoRTsl7N+WUczGA3UtjxeNvcafw6UsIWWZldBDSCqnqth45TbYNakJ
KKbaMldYTBbR/IU7if5F4O2W4hUHsWw278JkZrMYEf41sAN1VfgZrRisoRWbxq1KVmEnIEd5y8cjSZ2GFyeigjKLmmvoAGMi2x5XJtGY8xrNsz5o6BbwZwpJ
x7PkTpwdK1VPJky4xToNspZ4rSrFogS2KH9WE=
bash-4.2#
```

```
bash-4.2# terraform apply -var-file="logpass.tfvars" -auto-approve
null_resource.prepare_lambda_sources: Creating...
null_resource.prepare_lambda_sources: Provisioning with 'local-exec'...
null_resource.prepare_lambda_sources (local-exec): Executing: ["/bin/sh" "-c" "./build.sh"]
null_resource.prepare_lambda_sources (local-exec): Cloning into 'curl-to-s3'...
aws_eip.nat: Creating...
aws_vpc.main: Creating...
aws_efs_file_system.root_efs: Creating...
aws_s3_bucket.output_bucket: Creating...
aws_s3_bucket.input_bucket: Creating...
aws_s3_bucket.processing_bucket: Creating...
aws_eip.nat: Creation complete after 2s [id=eipalloc-0c58b9d9bb470172e]
```

```
- PyRDF.use("spark", {'npartitions': 32})
+ PyRDF.use("AWS", {'npartitions': 32})
```

Using the system - analysis



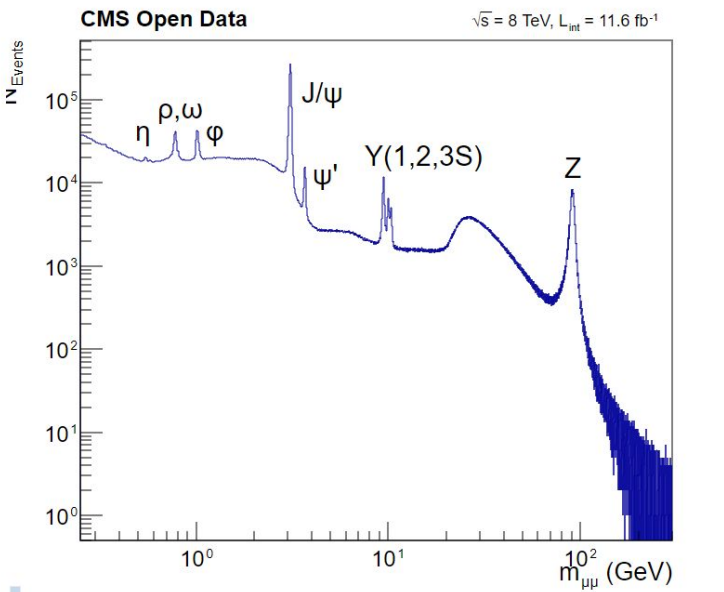
Using the system - analysis

In [6]:
```
# Produce pdf of the plot
ROOT.gStyle.SetOptStat(0)
ROOT.gStyle.SetTextFont(42)

c = ROOT.TCanvas("c", "", 800, 700);
c.SetLogx(); c.SetLogy();
h.SetTitle("");
h.GetXaxis().SetTitle("m_{#mu#mu} (GeV)"); h.GetXaxis().SetTitleSize(0.04);
h.GetYaxis().SetTitle("N_{Events}"); h.GetYaxis().SetTitleSize(0.04);
h.Draw();

label = ROOT.TLatex()
label.SetNDC(True);

label.DrawLatex(0.175, 0.740, "#eta");
label.DrawLatex(0.205, 0.775, "#rho,#omega");
label.DrawLatex(0.270, 0.740, "#phi");
label.DrawLatex(0.400, 0.800, "J/#psi");
label.DrawLatex(0.415, 0.670, "#psi'");
label.DrawLatex(0.485, 0.700, "Y(1,2,3S)");
label.DrawLatex(0.755, 0.680, "Z");
label.SetTextSize(0.040); label.DrawLatex(0.100, 0.920, "#bf{CMS Open Data}");
label.SetTextSize(0.030); label.DrawLatex(0.630, 0.920, "#sqrt{s} = 8 TeV, L_{int
```

```
c.SaveAs("dimuon_spectrum.pdf");
```

```
Info in <TCanvas::Print>: pdf file dimuon_spectrum.pdf has been created
```

In [7]:
```
%jsroot on

ROOT.gStyle.SetTextFont(42)

c2 = ROOT.TCanvas("c2", "", 800, 700);
c2.SetLogx(); c2.SetLogy();

h.SetTitle("");
h.GetXaxis().SetTitle("m_{#mu#mu} (GeV)"); h.GetXaxis().SetTitleSize(0.04);
h.GetYaxis().SetTitle("N_{Events}"); h.GetYaxis().SetTitleSize(0.04);
h.SetStats(False)
h.Draw();

label = ROOT.TLatex()
label.SetNDC(True);

label.DrawLatex(0.175, 0.740, "#eta");
label.DrawLatex(0.205, 0.775, "#rho,#omega");
label.DrawLatex(0.270, 0.740, "#phi");
label.DrawLatex(0.400, 0.800, "J/#psi");
label.DrawLatex(0.415, 0.670, "#psi'");
label.DrawLatex(0.485, 0.700, "Y(1,2,3S)");
label.DrawLatex(0.755, 0.680, "Z");
label.SetTextSize(0.040); label.DrawLatex(0.100, 0.920, "#bf{CMS Open Data}");
label.SetTextSize(0.030); label.DrawLatex(0.630, 0.920, "#sqrt{s} = 8 TeV, L_{int} = 11.6 fb^{-1}");

label.DrawClone("Same")
c2.Draw()
```

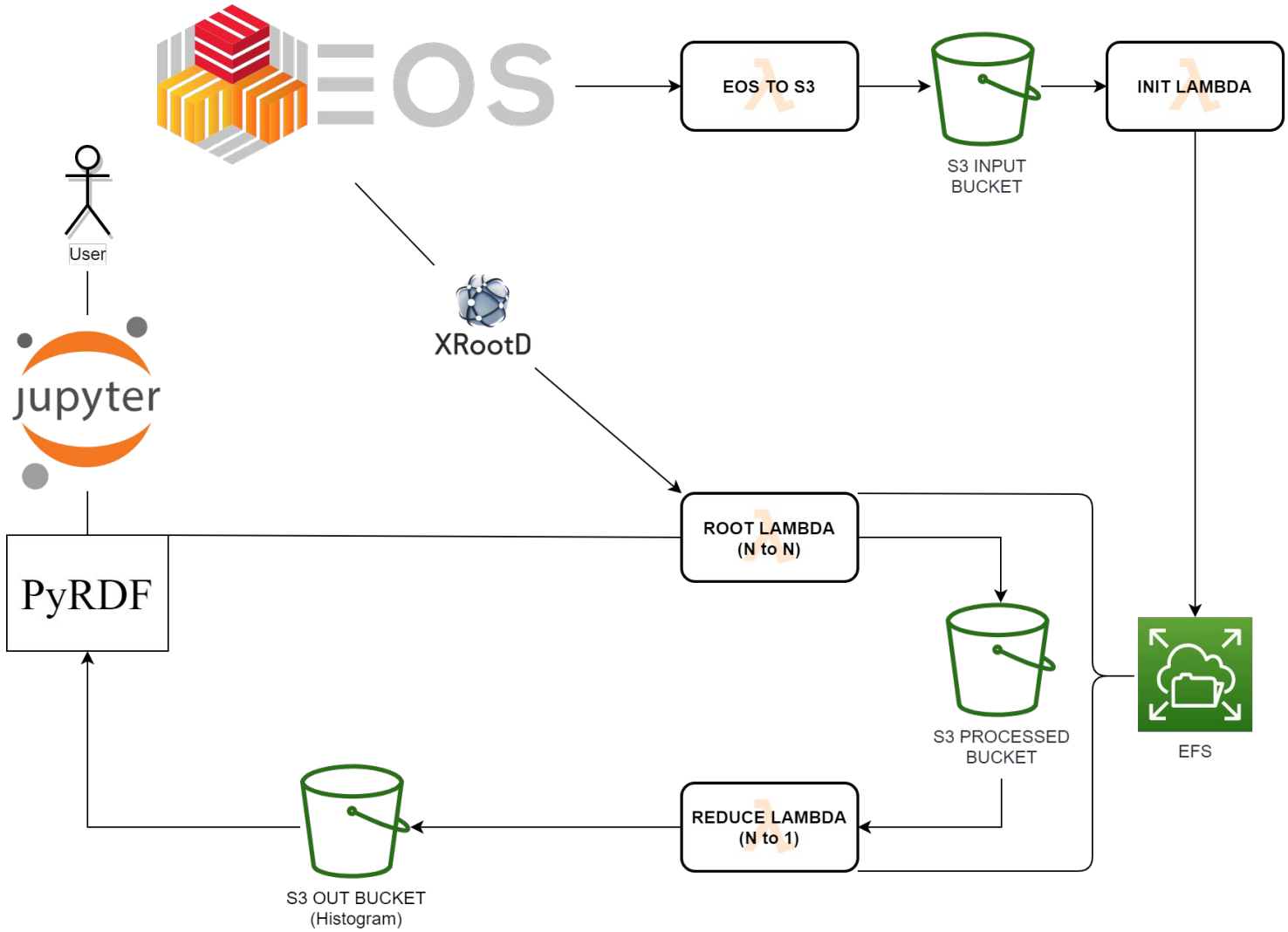


Some Remarks

- Computation is quite expensive as it stands now
- There’s not really much of exception handling at this point
- Remember to clean up after you’re done:

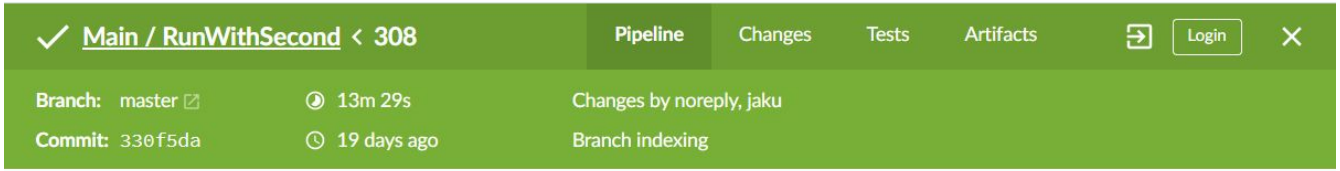
```
bash-4.2# terraform destroy -auto-approve
```

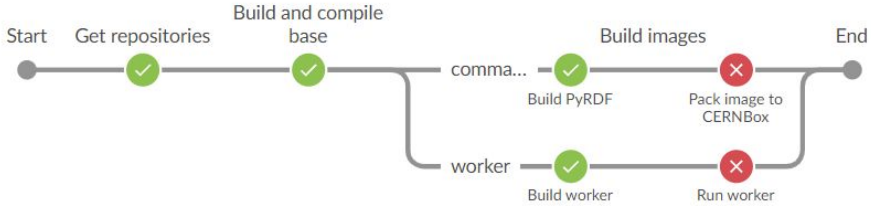
Technical Highlights


EOS
EOS TO S3
S3 INPUT BUCKET
INIT LAMBDA
User
XRootD
jupyter
PyRDF
ROOT LAMBDA (N to N)
S3 PROCESSED BUCKET
EFS
REDUCE LAMBDA (N to 1)
S3 OUT BUCKET (Histogram)

```
bash-4.2# du -h --max-depth=1 /mnt/cern_root/
394M    /mnt/cern_root/chroot
368M    /mnt/cern_root/root_install
762M    /mnt/cern_root/
bash-4.2#
```

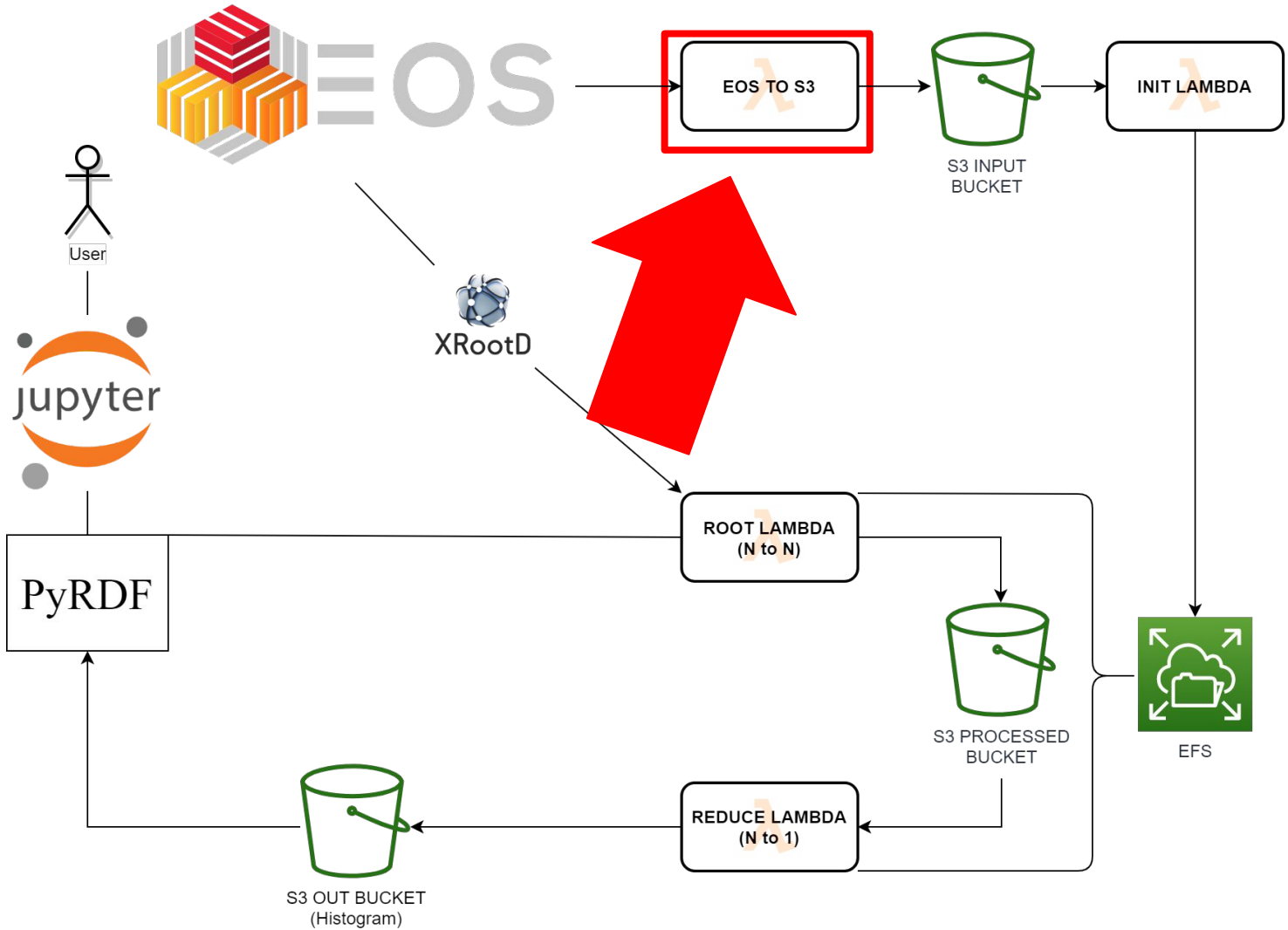
CI/CD - Jenkins



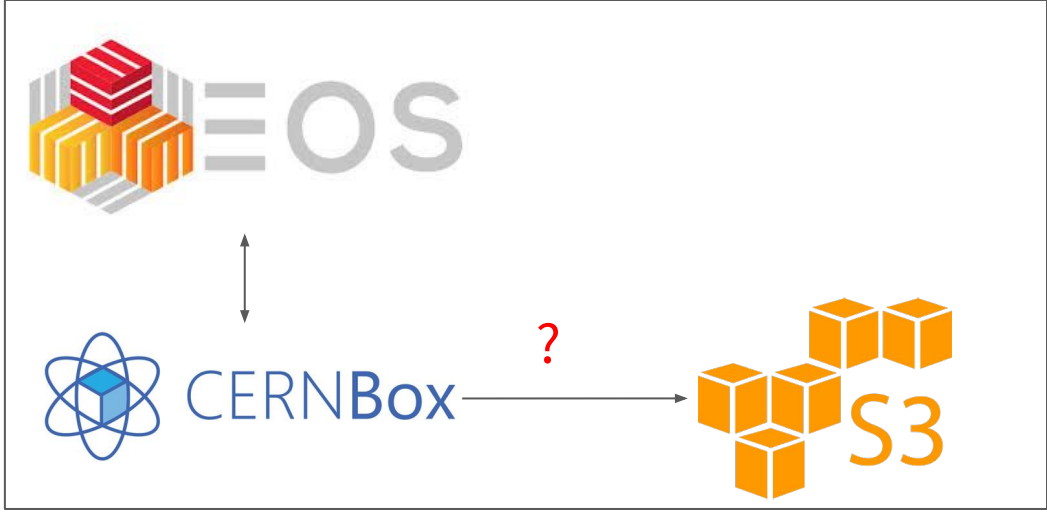


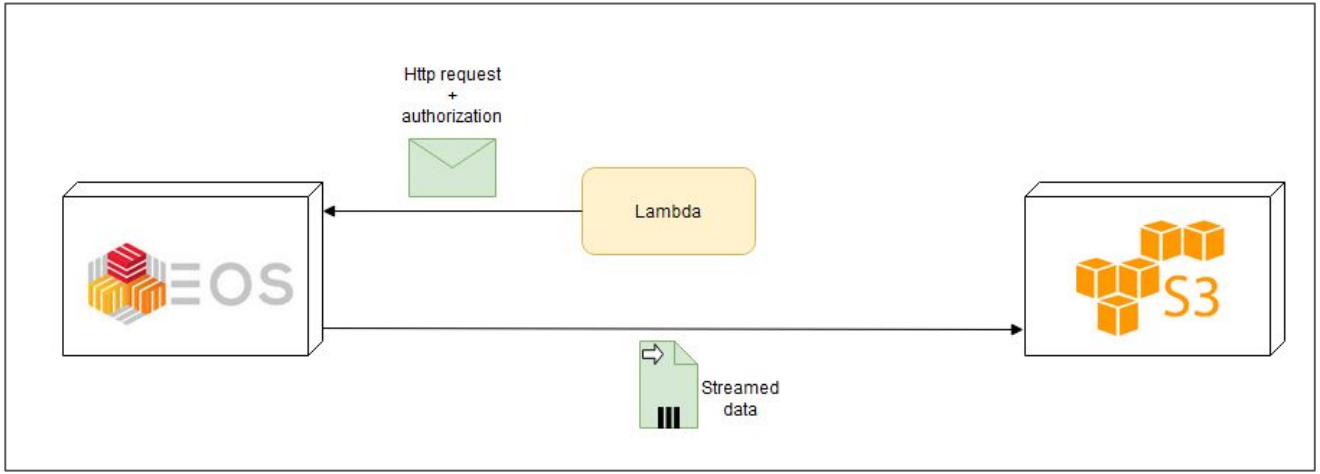
Build images / command - <1s

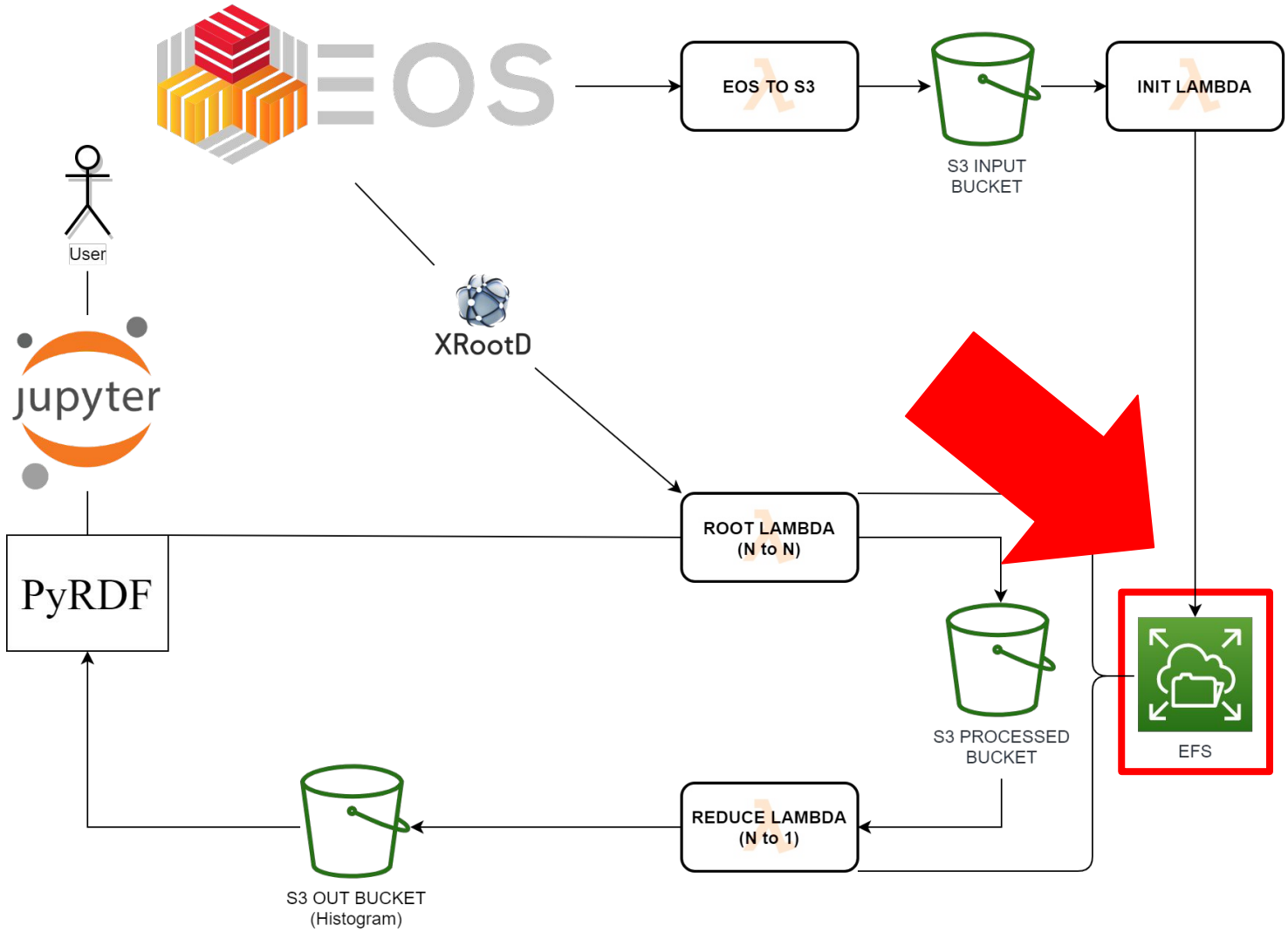
✓	Checks if running on a Unix-like node	<1s
✓	docker build -t pyrdf_terraform --network='host' commandRING — Shell Script	7m 9s
✓	Checks if running on a Unix-like node	<1s
✓	docker inspect -f . pyrdf_terraform — Shell Script	<1s
✗	mkdir -p /mnt/dav/AWS_ROOT cp /*.pdf /mnt/dav/AWS_ROOT cd /mnt/cern_root zip -r /mnt/dav/AWS_ROOT/... — Shell Script	5m 59s

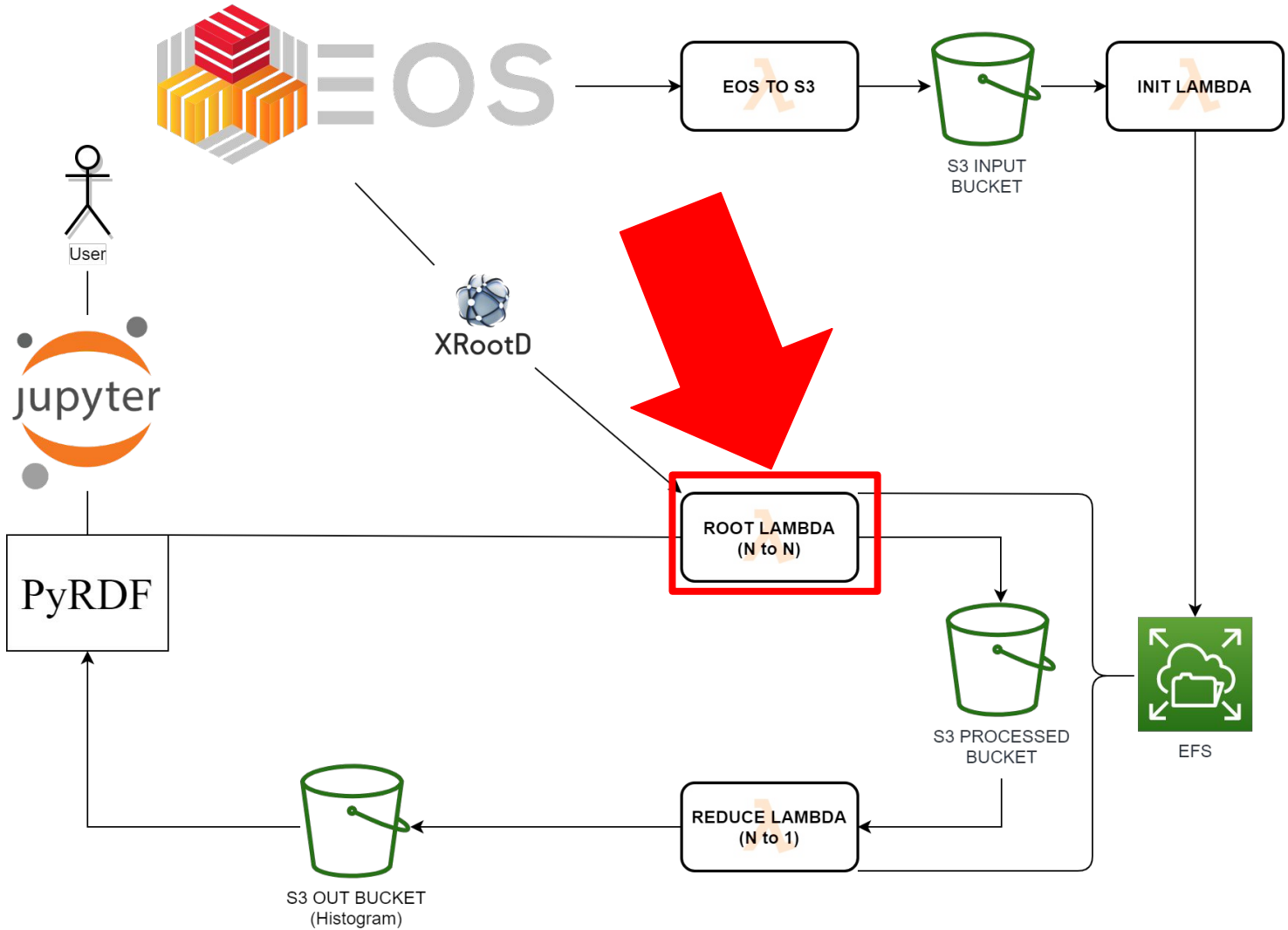

EOS
EOS TO S3
S3 INPUT BUCKET
INIT LAMBDA
User
XRootD
jupyter
PyRDF
ROOT LAMBDA (N to N)
S3 PROCESSED BUCKET
EFS
REDUCE LAMBDA (N to 1)
S3 OUT BUCKET (Histogram)

EOS Lambda

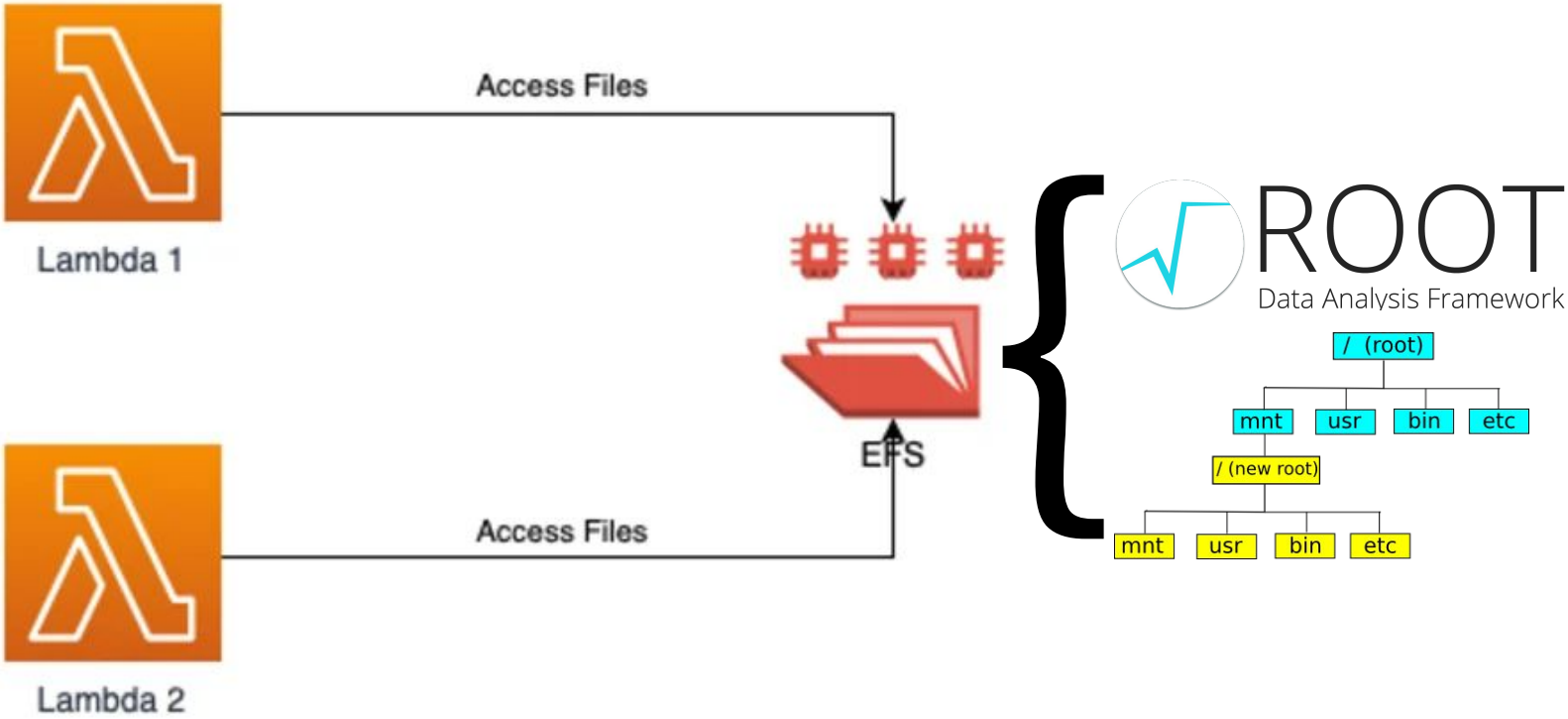

EOS
CERNBox
?
S3


Http request
+
authorization
Lambda
EOS
S3
Streamed
data


EOS
EOS TO S3
S3 INPUT BUCKET
INIT LAMBDA
User
XRootD
jupyter
PyRDF
ROOT LAMBDA (N to N)
S3 PROCESSED BUCKET
EFS
REDUCE LAMBDA (N to 1)
S3 OUT BUCKET (Histogram)

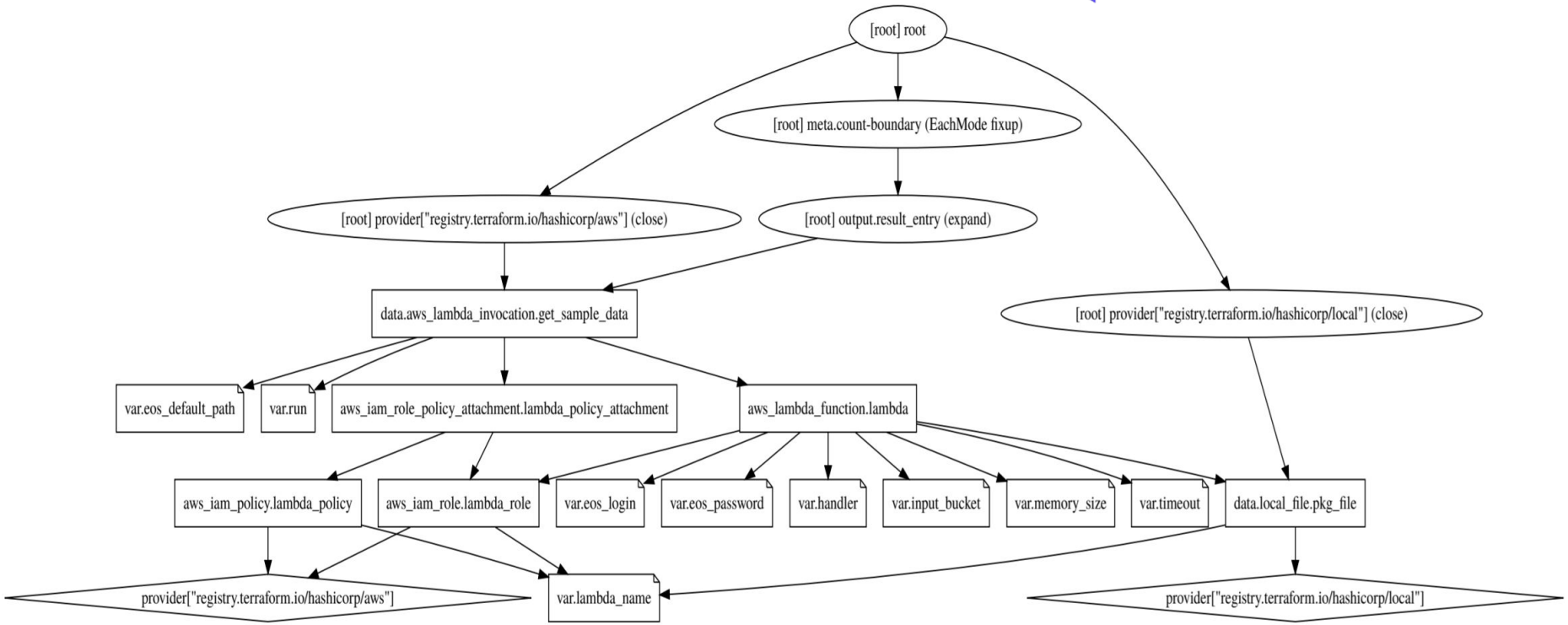

EOS
EOS TO S3
S3 INPUT BUCKET
INIT LAMBDA
User
XRootD
jupyter
PyRDF
ROOT LAMBDA (N to N)
S3 PROCESSED BUCKET
EFS
REDUCE LAMBDA (N to 1)
S3 OUT BUCKET (Histogram)

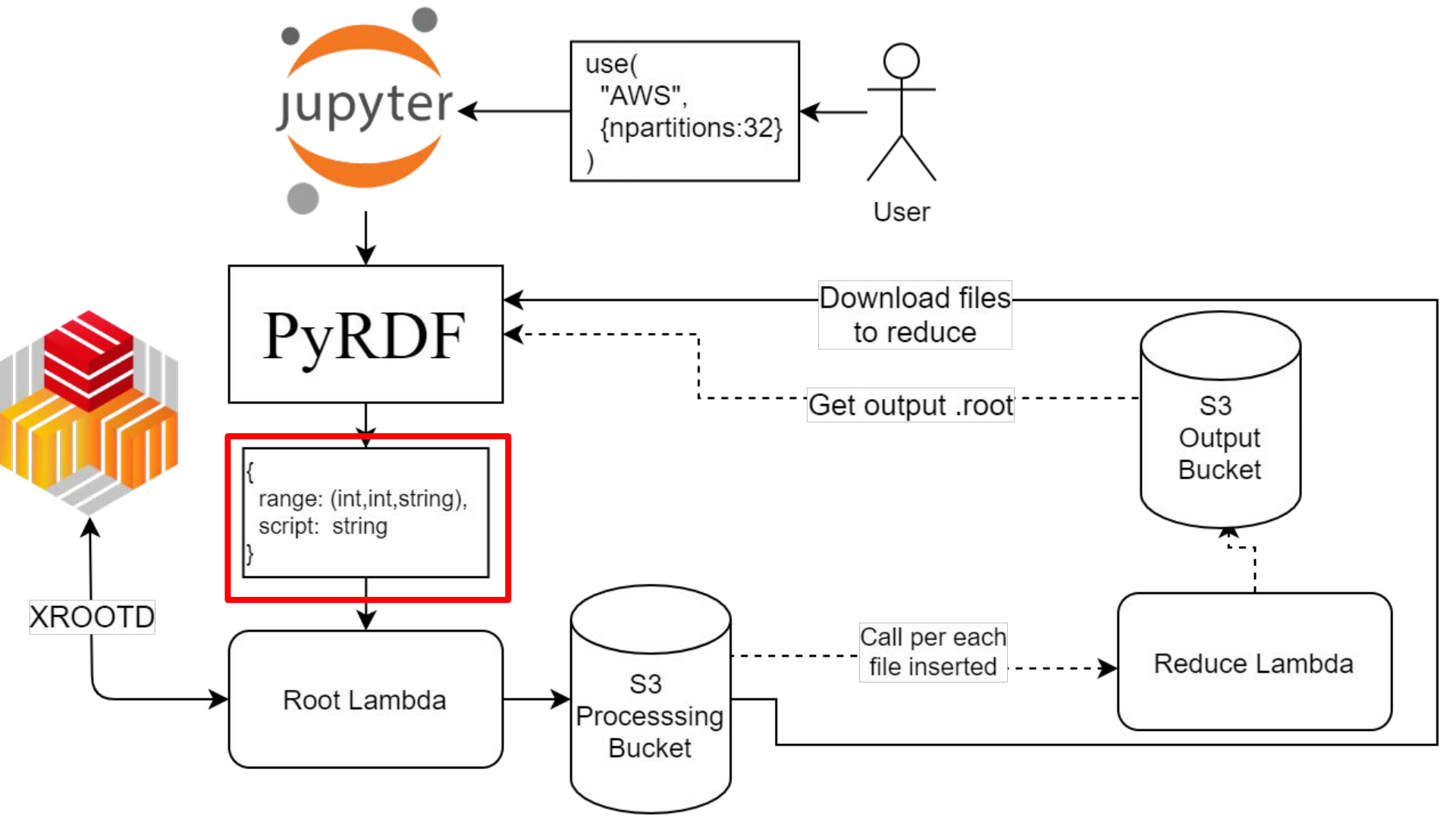
ROOT Lambda


Access Files
Lambda 1
EFS
ROOT
Data Analysis Framework
/ (root)
mnt
usr
bin
etc
/ (new root)
mnt
usr
bin
etc
Access Files
Lambda 2

Terraform

HashiCorp Terraform




jupyter
use(
"AWS",
{npartitions:32}
)
User
PyRDF
Download files
to reduce
Get output .root
S3
Output
Bucket
{
range: (int,int,string),
script: string
}
XROOTD
Root Lambda
S3
Processsing
Bucket
Call per each
file inserted
Reduce Lambda

Results

- Proof of Concept works
- Looks promising despite 17s init time per lambda
- Shared file system on EFS - bottleneck
- High cost - new Docker Lambda implementation should solve it

Some complaints

Summary and future work

- set up ROOT C++ JIT environment on AWS

Summary and future work

- set up ROOT C++ JIT environment on AWS
- integrated solution for PyRDF

Summary and future work

- set up ROOT C++ JIT environment on AWS
- integrated solution for PyRDF
- under certain conditions obtain not worse results than the current solution

Summary and future work

- set up ROOT C++ JIT environment on AWS
- integrated solution for PyRDF
- under certain conditions obtain not worse results than the current solution
- run arbitrary software with own environment on Serverless Platform

Summary and future work

- set up ROOT C++ JIT environment on AWS
- integrated solution for PyRDF
- under certain conditions obtain not worse results than the current solution
- run arbitrary software with own environment on Serverless Platform
- next step: move to Docker Lambda and test further

Summary and future work

- set up ROOT C++ JIT environment on AWS
- integrated solution for PyRDF
- under certain conditions obtain not worse results than the current solution
- run arbitrary software with own environment on Serverless Platform
- next step: move to Docker Lambda and test further

Thank you. Questions?

Special thanks to:

- Maciej Malawski, Associate Professor of AGH - Supervisor of the Project
- Vincenzo Eduardo Padulano - PyRDF expert

You can view code at https://github.com/CloudPyRDF

This work was supported by the Polish Ministry of Science and Higher Education, grant DIR/WK/2018/13

The project was realised as part of Bachelor Thesis at AGH University of Science and Technology in Kraków.