Developing effective testing pipelines for HPC applications

Jason C. Nucciarone | @nuccitheboss | Associate Software Engineer | HPC

CANONICAL ubuntu

Who am I?

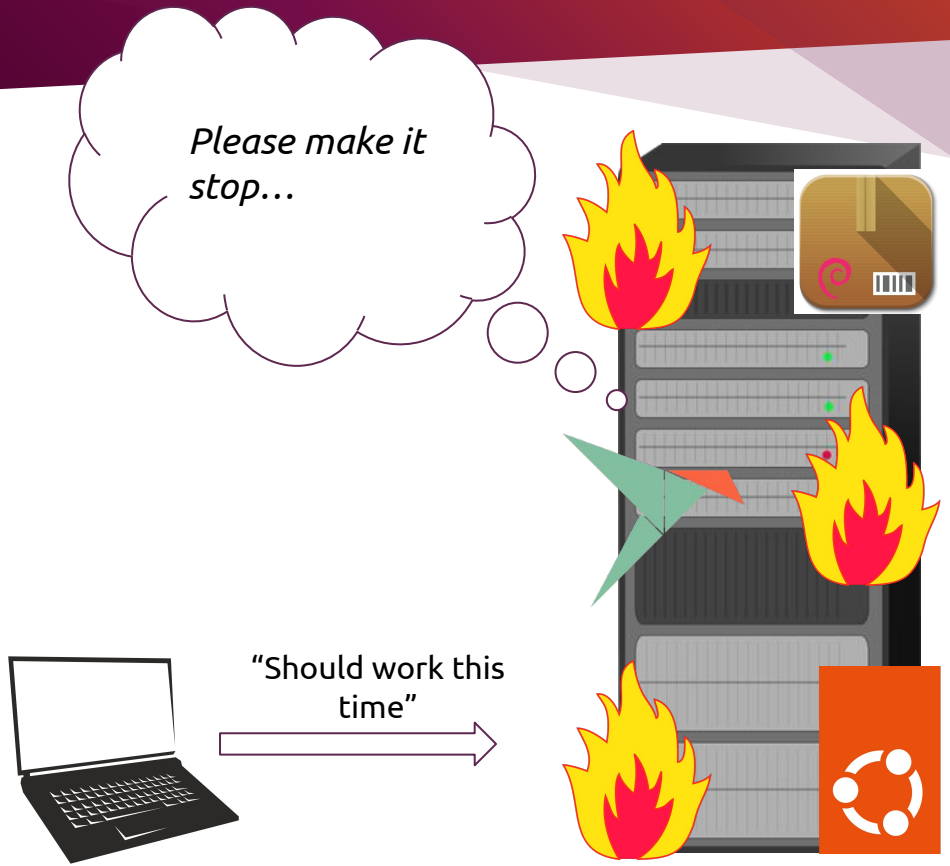
Who's this guy talking to you up here?

- First started in HPC industry with my university as an HPC software consultant with my university solving researchers issues with Singularity, Fortran, Jupyter, etc.
- Left for a bit to work adversarial machine learning workflow orchestration framework.
- Came back to my university's HPC site as an engineer to work on Singularity containers and cluster debugging tools.
- After graduation, I joined Canonical's new HPC team.

Now what sent me down this path of wanting to develop effective testing pipelines for HPC applications?

How it started…

- Began writing “destructive” code. Dangerous to test locally - what if I accidentally uninstall a package on my workstation that I should not have.
- Wanted to test machine provisioning scripts before moving onto actual deployments.
- Desire for reproducible tests. Enable others to run unit, functional, and integration tests without needing to adapt my workflow to their infrastructure.

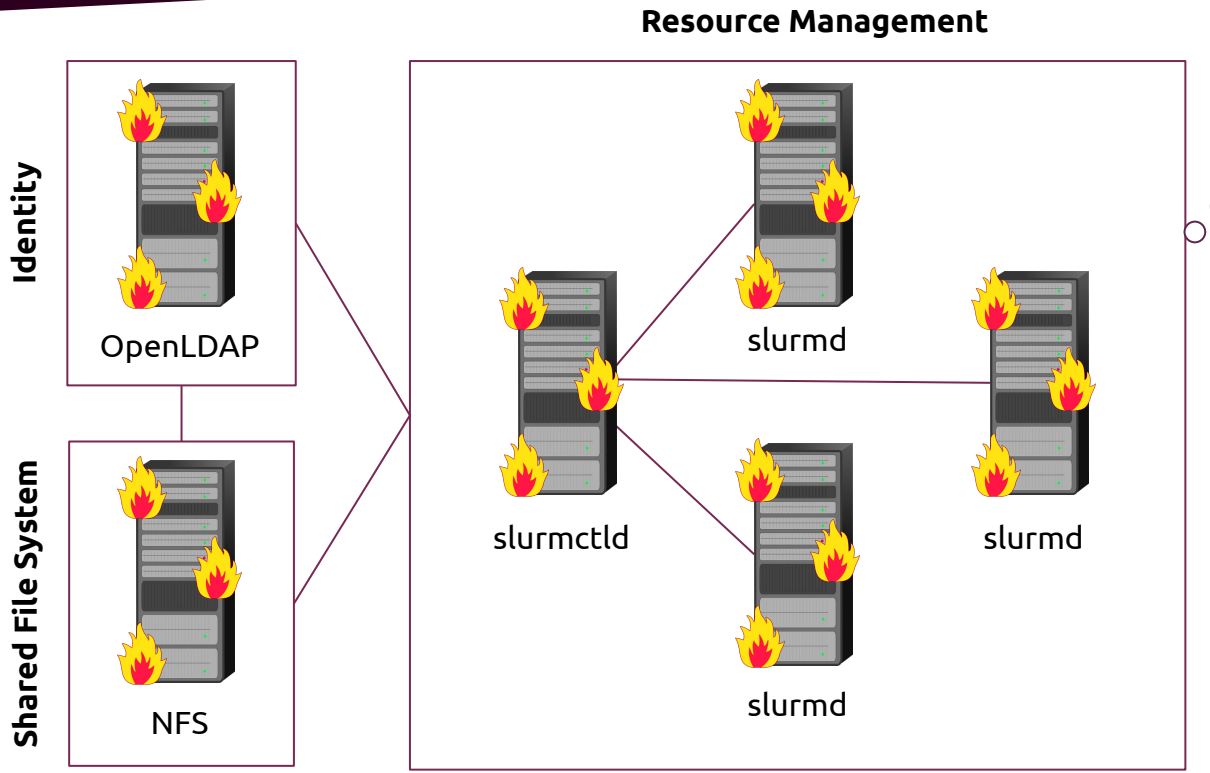


This gave me an idea…

What if I could take a test written here…

…and run it using any hypervisor I want, on whatever operating system I need, without any extra hassle?

How it is going…



"If you keep the thermostat set to 8°C, you might be able to afford the cloud bill this month"

Why waste precious compute time on your HPC cluster when you could *emulate* your cluster on a smaller scale locally instead?

What if you could test your applications, jobs, or simulations on a *mini-HPC cluster?* Test before using your compute allocation.

Let me introduce you to **cleantest** - a testing framework that brings up clean environments and mini-HPC clusters for developers in a hurry

What exactly is a “cleantest”?

cleantest’s are composed of three parts

Bootstrapping/Configuration

cleantest can be configured by registering hooks before the testlet is executed.

cleantest also has built-in utilities for creating mini-HPC clusters using your “test environment provider” of choice.

Testlets

Testlets are an entire Python program encapsulated inside of a regular function.

They contain the test that you want to run inside of the containerized/virtual environment.

Evaluation/Reporting

The result of the teslet is returned to the launching process.

cleantest is testing framework “agnostic” which means that you can use it with your testing framework of choice (pytest, unittest, etc.)

Bootstrapping/Configuration

Nodes can be provisioned, test environment instance configurations can be created, and hooks can be registered before the testlet is injected into the test environment instance.

The process for configuring cleantest usually is:

1. Instantiate a Configurer instance
2. Bring up nodes
3. Define hooks
4. Register hooks

Right: *LXDArchon registering a StopEnvHook (run after testlet has completed) and creating nodes.*

NucciTheBoss +1

```python
def test_lxd_archon_local() -> None:
    """Test LXDArchon against local LXD cluster."""
    archon = LXDArchon()
    archon.config.register_hook(
        StopEnvHook(name="get_result", download=[File("/tmp/result", root / "result")])
    )
    _ = archon.config.get_instance_config("ubuntu-jammy-amd64").dict()
    _["name"] = "mini-hpc-sm"
    archon.config.add_instance_config(
        InstanceConfig(
            config={
                "limits.cpu": "1",
                "limits.memory": "8GB",
                "security.privileged": "true",
                "raw.apparmor": "mount fstype=nfs*, mount fstype=rpc_pipefs,",
            },
            **_,
        )
    )
    archon.add(
        "ldap-0",
        image="mini-hpc-sm",
        provision_script=root / "ldap_provision_script.py",
    )
    sssd_conf = StringIO(
        templates.get_template("sssd.conf.tmpl").render(
            ldap_server_address=archon.get_public_address("ldap-0")
        )
    )
    archon.add(
        "nfs-0",
        image="mini-hpc-sm",
        provision_script=root / "nfs_provision_script.py",
        resources=[File(sssd_conf, "/root/.init/sssd.conf")],
    )
```

Testlets

Using Python decorators and metaprogramming utilities, the testlet is taken out of the Python process and instead run using the Python interpreter bundled within the test environment.

Middle-top: *Spread test on three different Ubuntu distributions.*

Middle-bottom: *Test that snaps were successfully installed.*

Right: *Submitting test SLURM job to mini-HPC cluster.*

NucciTheBoss

```
@lxd(
    image=["ubuntu-jammy-amd64", "ubuntu-focal-amd64", "ubuntu-bionic-amd64"],
    preserve=False,
    parallel=True,
    num_threads=2,
)
def install_tabulate():
    import sys

    try:
        from tabulate import tabulate

        print("tabulate is installed.", file=sys.stdout)
    except ImportError:
        print("Failed to import tabulate package.", file=sys.stderr)
        sys.exit(1)

    sys.exit(0)
```

NucciTheBoss

```
@lxd(image="ubuntu-jammy-amd64", preserve=False)
def functional_snaps():
    import sys
    from shutil import which

    if which("pypi-server") is None:
        sys.exit(1)
    elif which("marktext") is None:
        sys.exit(1)
    else:
        sys.exit(0)
```

NucciTheBoss +1

```
@lxd.target("slurmctld-0")
def run_job():
    import os
    import pathlib
    import shutil
    import textwrap
    from time import sleep

    from cleantest.utils import run

    tmp_dir = pathlib.Path("/tmp")
    (tmp_dir / "research.submit").write_text(
        textwrap.dedent(
            """
            #!/bin/bash
            #SBATCH --job-name=research
            #SBATCH --partition=all
            #SBATCH --nodes=1
            #SBATCH --ntasks-per-node=1
            #SBATCH --cpus-per-task=1
            #SBATCH --mem=500mb
            #SBATCH --time=00:00:30
            #SBATCH --error=research.err
            #SBATCH --output=research.out

            echo "I love doing research!"
            """
        ).strip("\n")
    )

    # Set user to test cluster user nucci.
    os.setuid(10000)
    os.chdir("/home/nucci")
    for result in run(
        f"cp {(tmp_dir / 'research.submit')} .",
        "sbatch research.submit",
    ):
        assert result.exit_code == 0
    sleep(60)
    shutil.copy("research.out", (tmp_dir / "result"))
```

Evaluation/Reporting

Testlet results from the test environment instances are returned as a generator. This is because the same testlet can be executed on multiple test environments instances.

Right: Various ways of evaluating the result of a testlet/testlets.

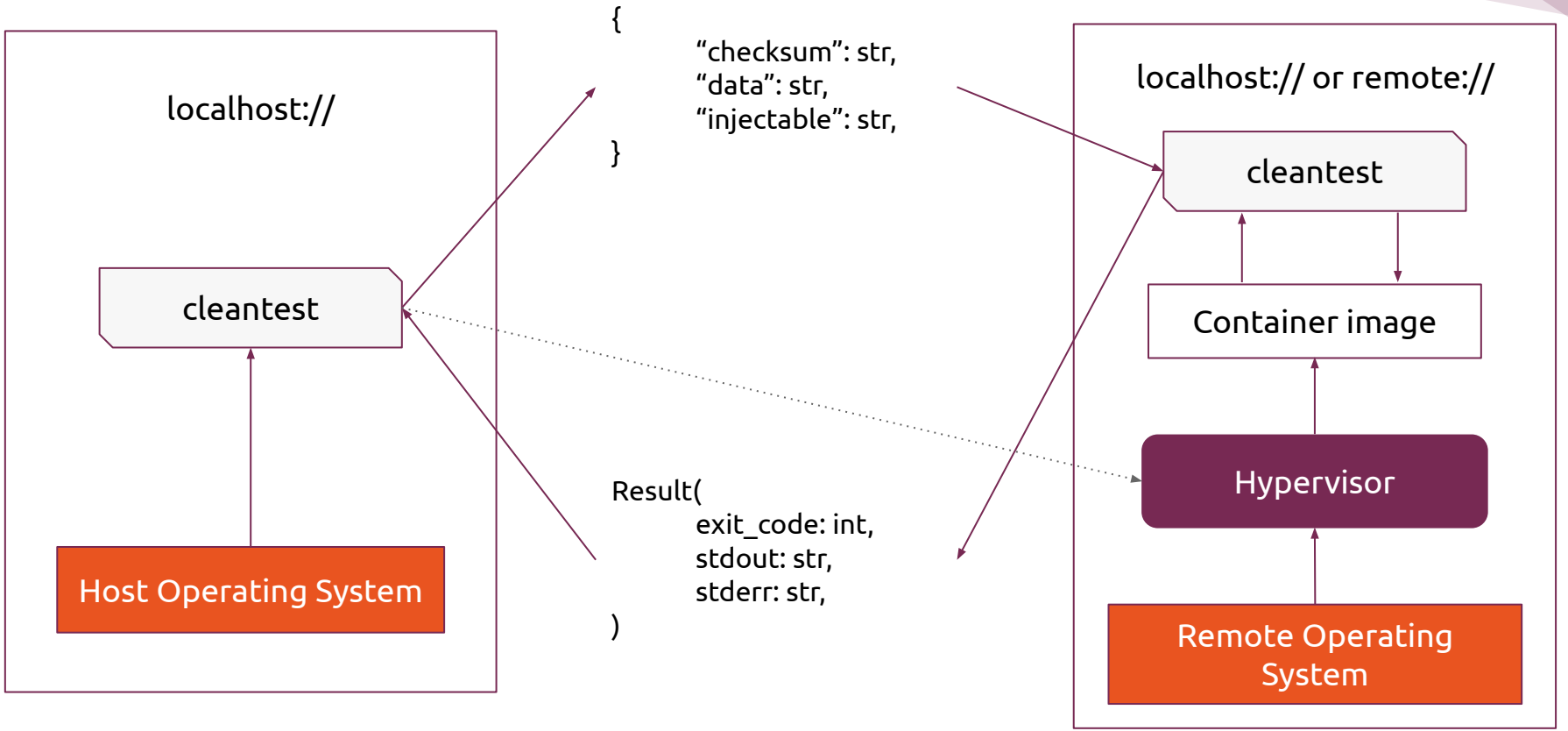
```
for name, result in work_on_artifacts():
    assert (
        pathlib.Path(tempfile.gettempdir()).joinpath("dump.txt").is_file() is True
    )
    assert pathlib.Path(tempfile.gettempdir()).joinpath("dump").is_dir() is True
```

```
for name, result in functional_snaps():
    assert result.exit_code == 0
```

```
for name, result in install_tabulate():
    try:
        assert result.exit_code == 0
    except AssertionError:
        raise Exception(f"{name} failed. Result: {result}")
```

```
for name, result in install_snapd():
    assert result.exit_code == 0
```

How does it work?



How does it work? (cont.)

Archon

An Archon (director) class is used for explicitly controlling the test environment provider.

Archon injects cleantest and its dependencies into every new node that it is directed to add.

Example of Archon being used:

```
archon.pull(
    "slurmctld-0", data_obj=[File("/etc/munge/munge.key", root / "munge.key")]
)
archon.add(
    ["slurmd-0", "slurmd-1", "slurmd-2"],
    image="mini-hpc-sm",
    provision_script=root / "slurmd_provision_script.py",
    resources=[
        File(sssd_conf, "/root/.init/sssd.conf"),
        File(StringIO(nfs_ip), "/root/.init/nfs-0"),
        File(root / "munge.key", "/root/.init/munge.key"),
    ],
)
```

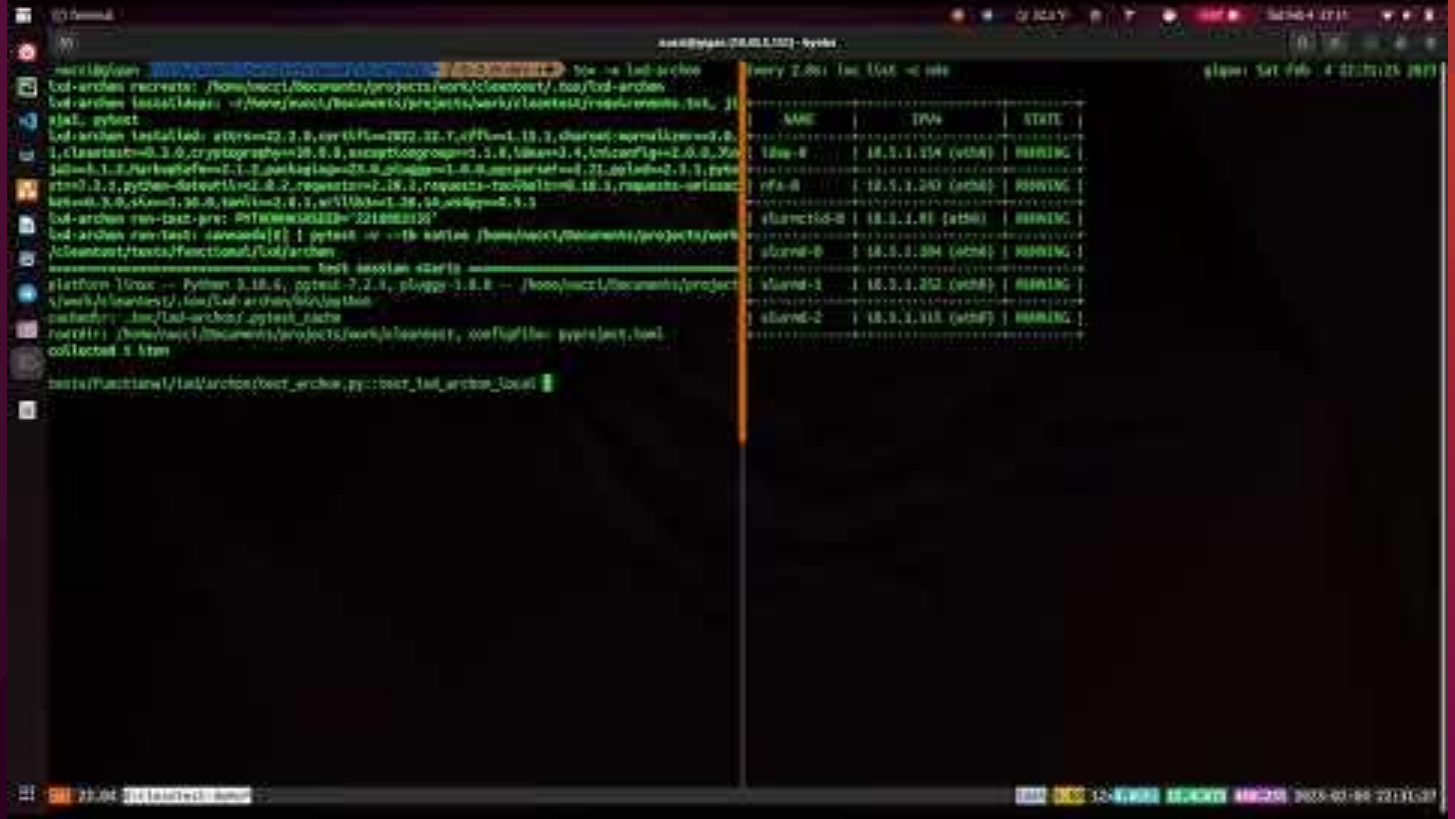
Harness

Harness wraps around a testlet to initialize, provision, manage, and destroy a test environment instance.

Harness injects cleantest and its dependencies into the test environment instance when a unique instance is created.

Example of Harness being used:

```
NucciTheBoss
@lxd(
    image=["ubuntu-jammy-amd64", "ubuntu-focal-amd64", "ubuntu-bionic-amd64"],
    preserve=False,
    parallel=True,
    num_threads=2,
)
def install_tabulate():
```



Current limitations :-(

- **Lack of robust multi-distribution support.**
  - You can launch Alma, Rocky, CentOS, Arch, etc instances, but the package macros and utilities do not yet support them fully.
- **Public documentation is behind.**
  - cleantest 0.3.0 -> 0.4.0 has seen some major API changes based on identified limitations in 0.3.0.
- **Lack of package (manager) integrations.**
  - Support for packages has been added ad hoc. In 0.5.0 I would to see the inclusion of support for Debs, Rpms, Pacs, Spack, and EasyBuild.
- **I am the only developer currently.**
  - I like to think that I am a good programmer, but we all know the truth.

cleantest source code (if you are interested!)

https://github.com/NucciTheBoss/cleantest

Ubuntu & HPC

Powering the next generation of research with the best of FOSS

We bring the best of…

TensorFlow

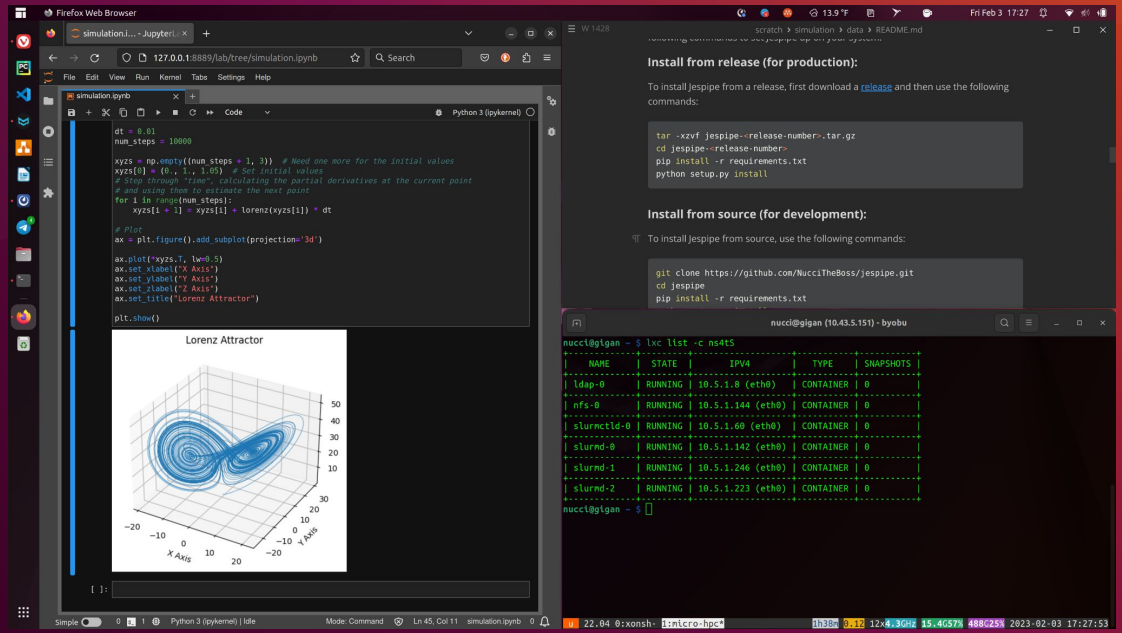
jupyter

slurm workload manager

Keras

MPICH

and more!



Want to improve HPC on Ubuntu? Scan the QR code above to join our public Mattermost channel!

Thank you for listening!
Questions, Comments, Concerns?