Containerized Custom Software Setup: Spatiotemporal Point Process Benchmarks (BenchSTPP Project)

This document has been prepared by Ismail Drief under the supervision of Mr. Yahya Aalaila

Introduction

In large-scale and complex projects such as **BenchSTPP**, the ability to customize the execution environment is not just a convenience, it is a necessity. Scientific pipelines that involve GPU acceleration, multiple Python libraries, and specific system dependencies require an environment that matches the local development setup as closely as possible. Relying solely on the default cluster environment often leads to compatibility issues, version mismatches, and unstable runtime behavior.
To overcome these challenges, Pegasus allows users to build and deploy custom container images tailored to their project's exact requirements. This approach ensures reproducibility, portability, and seamless execution of complex workflows on the HPC infrastructure.

While the Pegasus documentation provides all the necessary details, new users often find it difficult to navigate, especially if they are unfamiliar with HPC systems, containerization, or tools like Enroot and Podman. However, the process becomes straightforward once you understand the basic steps and follow them carefully.

The customization workflow can be broken down into three main stages:

1. **Image Creation** : Build a Docker image locally that mirrors your development environment. This involves defining dependencies (Python version, CUDA runtime, libraries, etc.) in a Dockerfile, installing them inside the image, and testing it locally.
2. **Image Export and Transfer** : Once the image is built and tested, export it as a tarball using docker save and transfer it to the Pegasus cluster via scp.
3. **Deployment and Usage on Pegasus** : Load the image on the cluster, convert it into an Enroot-compatible **`.sqsh`** container, and use it directly to run your project with srun. This container can then serve as a consistent runtime environment for future experiments.

The entire process is simple and repeatable when the documented steps are followed precisely. Moreover, it significantly reduces the friction of deploying complex software stacks on HPC systems and provides a scalable path for future extensions of the STPPGC platform.

In the next sections, we will go through all the necessary commands and procedures to build and deploy a fully customized software environment for the BenchSTPP project, from local image creation to execution on the Pegasus cluster:

I. Initial tips before starting:

1. Identify what you need:

- List all the dependencies your software needs: Python version, libraries (with versions), system packages (via apt/yum), CUDA / GPU, etc.
- Find out what base images Pegasus offers (look for /enroot images with CUDA / Python etc).

2. Install container tooling locally

- Install Docker or Podman on your laptop.
- If possible, install Enroot if you can—though Enroot is more for HPC clusters; locally you might just test via Docker or Podman then convert.

3. Use same base image

- Pull the same base image they use; for example: **nvidia/cuda:12.2.0-runtime-ubuntu22.04** (if that's what your code requires). Pegasus Docs
- If they have custom base images in **/enroot** you can try pulling their images or analogous ones.

II. Image Creation:

1. Create Dockerfile or build script :

- Write a Dockerfile that starts FROM that base image.
- In it, install system dependencies (apt etc), Python / CUDA libs, etc. Remember to use exact versions.
- Possibly disable APT sandboxing if your container is rootless (this is often needed; see docs) Pegasus Docs.
- In the Dockerfile, set up virtualenv or conda environment with your Python dependencies (requirements.txt or environment.yml) inside (or copy them in and run pip/conda install). In this exampel we utilize **pyproject.toml** file with **pip install -e .** installation command:

dockerfile

```dockerfile
# ==========================

# Base image with CUDA + Python

# ==========================

FROM nvidia/cuda:12.2.0-runtime-ubuntu22.04


# Set environment variables

ENV DEBIAN_FRONTEND=noninteractive

ENV LANG=C.UTF-8

ENV LC_ALL=C.UTF-8

ENV PYTHONUNBUFFERED=1

ENV PYTHONDONTWRITEBYTECODE=1


# ==========================

# System dependencies

# ==========================

RUN apt-get update && apt-get install -y --no-install-recommends \

  python3.11 \

  python3.11-venv \

  python3-pip \

  git \

  build-essential \

  cmake \

  wget \

  curl \

  && apt-get clean && rm -rf /var/lib/apt/lists/*
```

```
# Make python3.11 default
RUN update-alternatives --install /usr/bin/python3 python3 /usr/bin/python3.11 1

# Upgrade pip
RUN python3 -m pip install --upgrade pip setuptools wheel Cython numpy

# =========================
# Create working directory
# =========================
WORKDIR /workspace

# =========================
# Copy your project
# =========================
COPY . /workspace/

# =========================
# Install Python dependencies
# =========================
# Install local project + dependencies
RUN pip install --upgrade pip
RUN pip install .

# =========================
# Optional: pre-install Git repos
# =========================
RUN pip install
git+https://github.com/YahyaAalaila/neural_stpp.git@main#egg=neural_stpp \
```

```
        git+https://github.com/YahyaAalaila/DeepSTPP.git@master#egg=deepstpp \
        git+https://github.com/YahyaAalaila/SMASH.git@main#egg=smash \
        git+https://github.com/YahyaAalaila/DiffSTPP.git@main#egg=diffstpp\
       git+https://github.com/IsmailDr13f/AutoSTPP.git@Update_imports#egg=autoint-stpp



# ========================
# Entrypoint (optional)
# ========================
# CMD ["python3", "-m", "scripts.cli"]
```

2. **Build the Docker image locally:**

Make sure your Dockerfile is in your project directory:

```
docker build -t lightning-stpp:latest .
```

- This builds your container with Python 3.11, CUDA runtime, and all your dependencies from pyproject.toml.
- Git dependencies (neural_stpp, deepstpp, smash, diffstpp, autoint-stpp) are installed inside the image.
- Optional: use pip install -e . in the Dockerfile for editable installs.

3. **Test the image locally (optionnal):**

Run it interactively to verify everything works:

```
docker run --rm -it --gpus all -v $(pwd):/workspace -w /workspace lightning-stpp:latest bash
```

Inside the container:

```
python -m scripts.cli  # test your CLI
python -c "import torch; print(torch.cuda.is_available())"  # check GPU
```

III. Image Export and Transfer:

1. Export the image to a tarball:

This tarball can be transferred to Pegasus and imported into Enroot:

```
docker save -o image.tar lightning-stpp
```

- This produces a single file (image.tar) containing your full environment.

2. Transfer to Pegasus:

Copy the tarball to a shared location on the cluster (e.g., /netscratch/drief/containers/):

```
scp image.tar drief@login1.pegasus.dfki.de:/netscratch/drief/images/
```

IV. Deployment and Usage on Pegasus (login to your prefered cluster)

1. Create target folder on Pegasus:

Once you can log in:

```
drief@login1:/netscratche/drief$ mkdir -p /netscratch/drief/images

drief@login1:/netscratche/drief$ cd /netscratch/drief/images
```

2. Load the image via Podman:

On the cluster, request an interactive job, so you have a compute node where you can run commands manually:

```
drief@login1:/netscratch/drief/images$ srun  --mem=64G  --time=01:00:00 --immediate=300  --container-image=/enroot/podman+enroot.sqsh  --container-mounts=/netscratch/$USER:/netscratch/$USER,"$(pwd)":"$(pwd)"  --pty bash
```

```
drief@login1:/netscratch/drief/images$ srun   --mem=64G   --time=01:00:00 --immediate=300   --container-image=/enroot/podman+enroot.s
qsh   --container-mounts=/netscratch/$USER:/netscratch/$USER,"$(pwd)":"$(pwd)"   --pty bash
srun: jobinfo: version v1.0.0
srun: job 2175488 queued and waiting for resources
srun: job 2175488 has been allocated resources
Job 2175488: Running on node(s) kusel
Job 2175488: Started at 2025-09-23 12:10:04+0200
Monitor this job here: http://monitoring.pegasus.kl.dfki.de/d/slurm-job-details/job-details?var-jobid=2175488&from=1758622204000
Job 2175488: creating container for /enroot/podman+enroot.sqsh
Job 2175488: creating container for /enroot/podman+enroot.sqsh took 0.9 seconds
[root@kusel /]# podman images
```

(Adjust --mem, --time as needed.) This gives you a shell inside the Podman+Enroot container with access to /netscratch/drief/images/image.tar.

3. Import the image via Podman:

Within that environment:

```
[root@kusel /]# cd /netscratch/drief/images

[root@kusel images]# podman load -i image.tar
Copying blob 91cf2c50d7c1 done   |
Copying blob e36a0d1ef72b done   |
Copying blob b86fd3d0126f done   |
Copying config d2cd92cfbd done   |
Writing manifest to image destination
Loaded image: docker.io/library/lightning-stpp:latest
```

This loads the Docker image into Podman's local image store. Then list images to find the image name (tag) assigned:

```
[root@kusel images]# podman images
REPOSITORY                        TAG       IMAGE ID      CREATED         SIZE
docker.io/library/lightning-stpp  latest    d2cd92cfbdd9  21 hours ago    19.8 GB
```

Convert / import into Enroot (.sqsh) format

Once you know the image name (say it's **lightning-stpp:latest**), run:

```
[root@kusel images]# enroot import -o /netscratch/drief/images/lightning-stpp.sqsh podman://docker.io/library/lightning-stpp:latest
[INFO] Extracting image content...
[INFO] Creating squashfs filesystem...

Parallel mksquashfs: Using 2 processors
Creating 4.0 filesystem on /netscratch/drief/images/lightning-stpp.sqsh, block size 131072.
[=============================================================================================================-] 299794/299794 100%

Exportable Squashfs 4.0 filesystem, lzo compressed, data block size 131072
        uncompressed data, compressed metadata, compressed fragments,
        compressed xattrs, compressed ids
        duplicates are removed
Filesystem size 12252283.18 Kbytes (11965.12 Mbytes)
        63.88% of uncompressed filesystem size (19179646.45 Kbytes)
Inode table size 2251814 bytes (2199.04 Kbytes)
        34.07% of uncompressed inode table size (6610017 bytes)
Directory table size 1991050 bytes (1944.38 Kbytes)
        38.47% of uncompressed directory table size (5175444 bytes)
Number of duplicate files found 50421
Number of inodes 186420
Number of files 163625
Number of fragments 9322
Number of symbolic links 974
Number of device nodes 0
Number of fifo nodes 0
Number of socket nodes 0
Number of directories 21821
Number of hard-links 121
Number of ids (unique uids + gids) 1
Number of uids 1
        root (0)
Number of gids 1
        root (0)
[root@kusel images]#
```

This creates **/netscratch/drief/images/lightning-stpp.sqsh**. The **.sqsh** file is the squashfs image you can later use for jobs.

Use the .sqsh image in your Slurm job

Now write a Slurm job script (or use srun) that uses this image. For example, if your script is run_my_job.sh (or python train.py or whatever), your Slurm script could look like:

```
drief@login1:/netscratch/drief/projects/STPPGC$ srun --partition=A100-80GB    --gres=gpu:1    --cpus-per-task=8    --mem=48G    --time=12:00:00    --container-image=/netscratch/drief/images/lightning-stpp.sqsh    --container-workdir="$(pwd)"    --container-mounts=/netscratch/$USER/projects/STPPGC:/netscratch/$USER/projects/STPPGC,/ds:/ds:ro,"$(pwd)":"$(pwd)"    python3 scripts/cli.py
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS                                  bash - STPPGC

drief@login1:/netscratch/drief/projects/STPPGC$ srun --partition=A100-80GB \
     --gres=gpu:1 \
     --cpus-per-task=8 \
     --mem=48G \
     --time=12:00:00 \ …
Trial status: 1 TERMINATED
Current time: 2025-09-24 18:43:21. Total running time: 4min 27s
Logical resource usage: 1.0/256 CPUs, 0/1 GPUs (0.0/1.0 accelerator_type:A100)
┌──────────────────────────────────────────────────────────────────────────────────────────┐
│ Trial name                      status              lr     iter     total time (s)     val_loss │
├──────────────────────────────────────────────────────────────────────────────────────────┤
│ _run_single_trial_eb3d0_00000   TERMINATED   0.000179394      1            260.474      500.359 │
└──────────────────────────────────────────────────────────────────────────────────────────┘

best_trial.last_result keys:
dict_keys(['val_loss', 'timestamp', 'checkpoint_dir_name', 'should_checkpoint', 'done', 'training_iteration', 'trial_id', 'date', 'time_this_iter_s', 'time_total_s', 'pid', 'hostname', 'node_ip', 'config', 'time_since_restore', 'iterations_since_restore', 'experiment_tag'])
full last_result dict:
{'checkpoint_dir_name': 'checkpoint_000000',
 'config': {'lr': 0.00017939383629581132},
 'date': '2025-09-24_18-43-21',
 'done': True,
 'experiment_tag': '0_lr=0.0002',
 'hostname': 'serv-3337',
 'iterations_since_restore': 1,
 'node_ip': '192.168.33.217',
 'pid': 3624925,
 'should_checkpoint': True,
 'time_since_restore': 260.4739091396332,
 'time_this_iter_s': 260.4739091396332,
 'time_total_s': 260.4739091396332,
 'timestamp': 1758732201,
 'training_iteration': 1,
 'trial_id': 'eb3d0_00000',
 'val_loss': 500.3594055175781}
drief@login1:/netscratch/drief/projects/STPPGC$
```

Important note: If you need to modify the image

If the existing image needs changes (e.g. you want to install extra packages), you can:

- ***Start a shell with srun using the .sqsh or using podman+enroot.sqsh, mount your scratch space, then inside build/modify via Podman. Then re-import to get a new .sqsh.***
- ***Or record the modification as install.sh, and use the "wrapper script" or "task prolog" approach to install needed bits at job startup. If modifications are lightweight.***

- ***Please visit the [Pegasus Docs] for more informations.***

Import into Enroot on Pegasus (not yet tested)

On Pegasus:

enroot import /netscratch/drief/containers/lightning-stpp.sqsh

- This creates lightning-stpp.sqsh that can be used by any cluster user with read access to the file.