A Guide to UCIS4EQ Deployment and Execution

-= A users' primer and hands-on quickstart tutorial =-

Author(s): Cedric Bhihe, Wave Phenomena Group, CASE Dept, BSC

Doc's current maintainer(s): Cedric Bhihe

Contributors:

Table of Contents

List of abbreviations

API: Application Programming Interface
AWP: Anelastic Wave Propagation (open-source numerical engine)
DC: Docker container
EQ: Earthquake
ETHZ: Eidgenössischer Technische Hochschule in Zürich
HPC: High Performance Computing
IO, i/o: Input & Output
MN4/5: Marenostrum 4 / 5 cluster
PSHA: Probabilistic Seismic Hazard Analysis
Repo: Repository
RTE: Runtime Execution
RW: (also ‘rw’) Read & Write
UCIS4EQ: Urgent Computing Integrated Services for EarthQuakes

1. Objectives of this document

This document was written with those intent upon learning how to install and run UCIS4EQ in mind.

First objective: to get you started quickly

This tiny tutorial is meant to get you (the unwary and still happy users) started fast. It will also explain (however rudimentarily) where some of the important moving parts of UCIS4EQ are located and roughly what they do. When putting this document together, I thought that the best way to keep computationally challenged or, if you prefer, non-HPC-oriented users happy might be to help them start executing UCIS4EQ fast so that results could be forthcoming without undue delay.

Second objective: to provide you with some background

Simultaneously the emphasis was also to prevent you (the now wary but still happy users) from either getting lost or becoming stranded at low tide while handling UCIS4EQ. Here "lost" mean uncomprehending when staring at error messages, whereas "low tide" is for instance when you are still launching that really important job at 20:54 on a Friday, everybody has left and you experience an issue. For that reason this tutorial will sometimes include small didactic notes, that the regular user can easily ignore. They are only meant to give new-comers some background and in the end impart them with a sense of what is happening.

Third objective: to collect options/perspectives to improve the user-friendliness and usability of the UCIS4EQ workflow.

This is a longer term goal and consists in improving and/or extending UCIS4EQ to the point that it becomes a turn-key package accessible to seismologists and others, whose main aspiration is to get their work done, not to read dry technical tutorials.

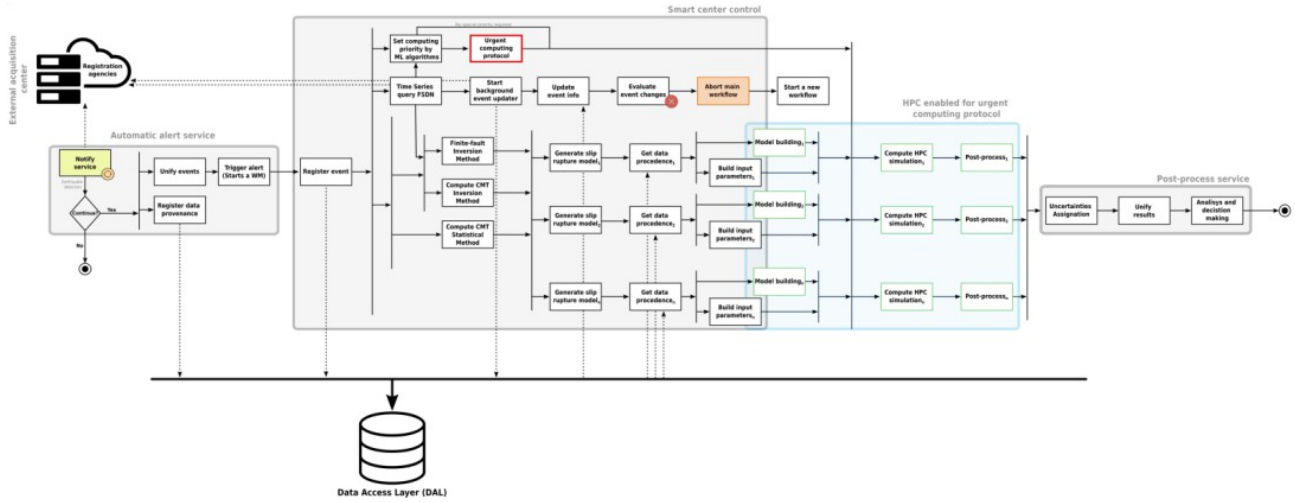
As it stands this tutorial should be considered work-in-progress. For that reason suggestions and critiques, isolated or by drove, are highly welcome and should be sent to the current document's maintainer.

2. UCIS4EQ: a high level view

UCIS4EQ is a workflow for the deterministic modelling of seismic waves in an urgent computing context. It consists of containerized microservices written in Python. One of those microservices is a wrapper that launches **Salvus**, a high-performance solver edited by Mondaic AG, on BSC's compute/storage infrastructure MN4/5. A high level view (shown below) was presented by Marta Pienkoska as early as 2021[1].

TODO

1 M. Pienkowska, J. E. Rodríguez, J. de la Puente, and A. Fichtner, “Deterministic modelling of seismic waves in the Urgent Computing context: progress towards a short-term assessment of seismic hazard,” presented at the EGU21, Harvard, Mar. 03, 2021. doi: 10.5194/egusphere-egu21-15516.



3. Key individuals

3.1- Author(s) of the code

Juan Esteban Rodríguez <jrodriguez@frontwave.io>, FrontWave imaging, SL (ex CASE Dept, BSC)

- The UCIS4EQ code was developped at BSC as part of the ChEESE project, funded from the European Union's Horizon 2020 research and innovation program, under the Grant Agreeme'nt No 823844.

3.2- Contributor(s) to the code

Jorge Ejarque <jorge.ejarque@bsc.es>, CS Dept, BSC
Marisol Monterrubio <marisol.monterrubio@bsc.es>, CASE Dept, BSC
Marta Pienkowska <marta.pienkowska@erdw.ethz.ch>, Earth Sciences Dept, ETHZ

3.3- Current Gitlab repository maintainer(s)

Cedric Bhihe <cedric.bhihe@bsc.es>, CASE Dept, BSC
Sergio Mendoza <sergio.mendoza@bsc.es>, CASE Dept, BSC
Josep de la Puente <josep.delapuente@bsc.es>, CASE Dept, BSC

- The UCIS4EQ code and Docker image registry are currently located at https://gitlab.bsc.es/wavephenomenagroup/ucis4eq

3.4- Advanced user(s)

Marisol Monterrubio <marisol.monterrubio@bsc.es>, CASE Dept, BSC
Marta Pienkowska <marta.pienkowska@erdw.ethz.ch>, Earth Sciences Dept, ETHZ
Jorge Ejarque <jorge.ejarque@bsc.es>, CASE Dept, BSC
Cedric Bhihe <cedric.bhihe@bsc.es>, CASE Dept, BSC

4. Licenses

5. Solvers

5.1- Salvus

Salvus is a high-performance solver for spectral-element wave propagation (forward compute), full waveform modeling and inversion (backward compute). Salvus is published by Mondaic. Its executable binary is currently located (in MN4) at:

`/gpfs/projects/bsc44/earthquake/UCIS4EQ/Salvus/bin/salvus`

Salvus covers meshing, model building, data management and full waveform inversions in a scalable manner, taking advantage of NVIDIA GPU support using CUDA. It runs on ARM and POWER PC architectures as well as on CPU based on the x86-64 instruction set.

Marenostrum 4 (MN4) users can run Salvus in two ways:

- as part of a microservice with deported execution, e.g. as part of the UCIS4EQ workflow,
- as a standalone binary.

To execute the stand-alone binary of Salvus, the Marenostrum user places the job in a queue, using Slurm, the BSC queue manager used for batch processing:

```
> sbatch /gpfs/projects/bsc44/earthquake/UCIS4EQ/runs/.../<my_job>.slurm
```

The `<my_job>.slurm` executable is a custom file with a specific structure and syntax:

```
> cat /gpfs/projects/bsc44/earthquake/UCIS4EQ/runs/.../<my_job>.slurm
#!/bin/bash
#SBATCH --jobname=<my_jobname>
#SBATCH --time=12:00:00
#SBATCH --nodes=50
#SBATCH --tasks-per-node=48
#SBATCH --ntasks=2400
#SBATCH --error=<my_jobname_SalvusRun.e>
#SBATCH --output=<my_jobname_SalvusRun.o>
#SBATCH --qos=<my_picked_queue>
#SBATCH --mail-type=start|end|all|none
#SBATCH --mail-user=<my_email_address>

cd $SLURM_SUBMIT_DIR
module load fabric
export I_MPI_EXTRA_FILESYSTEM_LIST=gpfs
export I_MPI_EXTRA_FILESYSTEM=on

/usr/bin/srun --overlap --ntasks=$SLURM_NTASKS --ntasks-pernode=\
$SLURM_NTASKS_PER_NODE /gpfs/projects/bsc44/earthquake/UCIS4EQ/Salvus/bin/salvus \
compute /gpfs/projects/bsc44/earthquake/UCIS4EQ/runs/.../salvus_wrapper/\
salvus_input_rupture.toml
```

where:

`/gpfs/projects/bsc44/earthquake/UCIS4EQ/runs/.../salvus_wrapper/salvus_input_rupture.toml` is a required `toml` file. It describes a specific event's compute grid geometry, domain meshing, rupture model and as well as the ground velocities along Z (vertical), N and E (horizontal plane), at each recording seismic station location.

The `toml` file is obtained based on the contents of `/gpfs/projects/bsc44/earthquake/UCIS4EQ/runs/[……]/salvus_wrapper/salvus_input.yaml` which contains descriptive parameters for:

- the seismic event of interest,

- the receiving stations' locations,

- the precomputed mesh's file paths if the mesh is not computed on the fly:

The mesh, whether pre-computed or computed on-the-fly, depends on the wave's highest resolved frequency of interest. Accordingly Salvus's treatment of frequency largely governs mesh and memory requiremens at time of execution. "The highest resolvable frequency - or the shortest period, respectively - is the main external driver governing the mesh generation. It influences the wavelengths of the wavefield (or "polynomial_degree"), the effective medium which results in a homogenized model, as well as the resolution of topography and bathymetry which yield the effective domain. The computational costs are proportional to the number of elements and the number of time steps." (cit. Mondaic's documentation pages). Resolving higher frequencies will obviously increase the overall computational cost incurred by users.

- the bathymetry and topography (if applicable),

- the kinematic rupture model's Standard Rupture Format (`*.srf`) file's fully qualified path.

In the latter `*.srf` file, the general fault surface is represented by a distribution of point sources (each pont source being reffered to as a subfault). Each subfault's point-source description contains the kinematic (displacement) information necessary to compute the contribution of that subfault to the total response to the fault rupture.

5.2- AWP

The Anelastic Wave Propagation (AWP) numerical engine is an open-source solver based on finite differences. It powers the CyberShake HPC workflow with planar topography (squashed mesh model), and was developed to perform PSHA studies for Southern California by SCEC (the Southern California Earthquake Center). It is instrumental in providing Physics-Based Probabilistic Seismic Hazard Analysis (PB-PSHA) by computing a large set of earthquake synthetic ground-motion time histories from kinematic rupture scenarios on 3D finite faults.

...Versión CPU (versión de Juane) corre en Power9
Versión CPU (versión de Edu) corre en MN4

With non-planar topography on GPU (Power9) -- TODO

6. Installing UCIS4EQ

6.1- Preliminary steps

a) Install the following resources on your host:

- `docker` (Community Edition), `docker-compose`
  If you are unfamiliar with that technology, follow this tutorial to get started. Supplement it with those official docs if necessary.

- `git`
  If you are unfamiliar with that technology, and its GitHub or GitLab incarnation, follow this tutorial to get started.

- `Salvus`
  Salvus currently requires an installation of Python 3.7, 3.9 or 3.11. There are different ways to set this up and most should work fine. We recommend to install the latest Miniconda distribution and subsequent dependencies using the package manager Mamba. If you follow Mondaic's recommended installation procedure you will first install Miniconda and then create a new python environment using `mamba` in which to finally install Salvus and all dependencies. In case this is new to you there are a lot of guides and

instructions around, e.g. this one.
You may also choose not to resort to Conda or to Miniconda, but to Pyenv instead, to create a virtual Python environment with all packages expected by Salvus in order to run.

b) Secure at least read-access permissions to the current repo https://gitlab.bsc.es/wavephenomenagroup/ucis4eq/

c) Pick `<basedir>` as your working directory on your local host's filesystem to clone the above git repository. This will bring the complete code suite onto your local host.

```
> cd <basedir>
> git clone https://gitlab.bsc.es/wavephenomenagroup/ucis4eq.git
```

d) Secure access to the git repo's registry.
The `https://gitlab.bsc.es/wavephenomenagroup/ucis4eq.git` repository includes a registry holding Docker images. In order for UCIS4EQ to operate normally, ensure programmatic access to the repo's Docker image registry via the SSH protocol. To that end further steps are required. Access to the registry is mediated by your local host's environment variable SSH_AUTH_SOCK. That environment variable must be set so that the host's SSH agent points to the Unix domain socket it will use for SSH communication between processes running in your host and the Gitlab's instance's registry.

The sequence of actions required to complete this specific setup consists in:

- creating an asymmetric key pair
- start an SSH agent: '`ssh-agent`' (most probably)
- adding one or more keys to the SSH agent's keyring
- making sure that your SSH agent is automatically started at the beginning of your session
- ensuring that your SSH agent's keyring is populated with keys so it may provide them on demand.

In case '`ssh-agent`' is not available and GNOME is your Desktop Manager, activate the GNOME keyring agent '`grc-ssh-agent.socket`' (part of the '`grc`' package), per this link.

With '`ssh-agent`' the detailed actions are:

- Create either an RSA (Rivest–Shamir–Adleman), or an Ed25519, or an ECDSA (the latter two being Elliptic Curve Digital Signature algorithms) asymmetric key pair and save it to the specified path. For ECDSA only 1 of 3 elliptic curve sizes (in bits) can be specified with the `-b' flag: 256, 384, or 521. When prompted, it is a good idea to differentiate the key pair 's name from others by calling it for instance: '`gitbsc_id_rsa`' for an RSA key pair or '`gitbsc_id_ecdsa` for an ECDSA key pair.

```
> ssh-keygen -t rsa -b 4096 -C <my_email> -f ~/.ssh/gitbsc_id_rsa \
       -P "my_secret_passphrase"
```

or, for lighter-weight but equally secure alternatives:

```
> ssh-keygen -t ecdsa -b 521 -C <my_email> -f ~/.ssh/gitbsc_id_ecdsa \
      -P "my_secret_passphrase"
```

or:

```
> ssh-keygen -t ed25519 -C <my_email> -f ~/.ssh/gitbsc_id_ecdsa \
      -P "my_secret_passphrase"
```

Note: `ssh-keygen` defaults to creating an RSA key of length 3072 bits.

- Start the SSH Agent for it to manage keys properly, this will set the environment variable SSH_AUTH_SOCK automatically.

```
> eval "$(ssh-agent -s)"
```

- Add the private key to SSH Agent. (You will be prompted for the passphrase you gave for the key pair.)

```
> ssh-add ~/.ssh/gitbsc_id_rsa    # for RSA keys
```

or

```
> ssh-add ~/.ssh/gitbsc_id_ecdsa  # for ECDSA keys
```

- Verify that keys are correctly created andd that the SSH_AUTH_SOCK environment variable is correctly set.

```
> echo $SSH_AUTH_SOCK
 ...
> ls -lrt ~/.ssh      # lists the key files in their default location on your host.
 ...
> ssh-add -l          # lists the fingerprints of all identities represented by the Agent.
 ...
> cat ~/.ssh/config   # look for lines containing `IdentityFile' with the path to your private key.
 ...
```

- Finally check the remote server that hosts the Gitlab instance for the presence of the public key you just created (e.g. `gitbsc_id_rsa.pub'). The private key (e.g. `gitbsc_id_rsa') you just created should not appear on the remote server running the GitLab instance; only the public key should. In other words that public key and only it must be stored on the distant host's `~/.ssh/authorized_keys' file, or in your Gitlab repo's profile's setting in the *SSH Keys* section.

- Checking that your 'ssh-agent' is automatically started at the beginning of your session and populated with keys so it may provide them on demand when SSH'ing to a distant server is essential. If it is not the case refer to this post and this wiki to setup the automatic launch of a Systemd *user service*.

To run UCIS4EQ on your host while running Salvus on Marenostrum successfully, a number of supplementary steps are necessary. Those steps are described in the following sections.

6.2- Secure an account on Marenostrum 4 (MN4)

That should include zero-login access to the MN4 supercomputing resources:

- Marenostrum 4 (x86_64 CPU CPU cluster),
- Minotauro (Bull CUDA cluster),
- /gpfs/ (MN4 distributed storage system).

Create your SSH asymmetric key pair (e.g. 4096 bit RSA or equivalent) to access the 4 nodes {dt01,mn1,mn2,mn3}.bsc.es. This key should include a passphrase for security.

```
> cd ~/.ssh
> ssh-keygen -t rsa -b 4096 -C "<your_bsc_email>" -f ~/.ssh/bsc_id_rsa \
      -P "<your_passphrase>"
> chmod go-r ~/.ssh/bsc_id_rsa.pub
> ssh-copy-id <your_bsc_hpc_username>@dt01.bsc.es
```

By default ssh-copy-id should add the last created public key to your remote hosts ~/.ssh/authorized_keys file. If successful you should now be able to log in onto any of the above four MN4 nodes without the need to provide a password. Try it for instance with:

```
> ssh -l <your_bsc_hpc_username> dt01.bsc.es
> cat ~/.ssh/authorized_keys
```

6.3- Set up your HPC user to run Salvus and to access events' static data

Salvus requires a quick access to event-related static data to avoid solver execution delays induced by lengthy data transfer to compute nodes. For this reason the weightier part of that data needs to be stored where accessing it will not incur delays, i.e. on the dedicated MN4 file system /gpfs/projects/bsc44/....

To meet those requirements, the HPC user should be a member of the bsc44 Unix group on MN4, with the following permissions:

```
/gpfs/projects/bsc44       drwxrws---
```

The above is enough to give you access to the Salvus binary and to the bulk of static data needed to run the solver for specific faults and fault geometries related to events. There are additional static data access requirements as detailed in the following sub-section .

6.4- Gain access to https://b2drop.bsc.es

That should include in particular the B2DROP directory `UCIS4EQ' owned at time of writing by user Marisol Monterrubio <marisol.monterrubio@bsc.es>. It contains:

- UCIS4EQ/Data/Events/SlipDistributions/
- UCIS4EQ/Data/Regions/
- UCIS4EQ/Setup/AAS/config_{stations,events,dataselect}.json
- UCIS4EQ/Setup/DALv2/
- UCIS4EQ/Setup/SCC/indexPriority/PGA_database **** in disuse**
- UCIS4EQ/Runs/

To secure access to b2drop.bsc.es, send an email from BSC to <datamanagement@bsc.es> with your request. After receiving a username and setting up your credentials, ask any known user of directory UCIS4EQ to share its contents with you.

At time of writing, users with access to the UCIS4EQ folder are: Cedric Bhihe, Eduardo Cabrera, Jorge Ejarque, Sergio Mendoza, Marisol Monterrubio, Natalia Zamora at BSC, and Marta Pienkowska at ETH.

The contents of directory UCIS4EQ en https://b2drop.bsc.es include static data needed to specify events to be submitted to Salvus for execution. The UCIS4EQ filetree, i.e. the organization of its contents in a tree made of sub-folders and individual files, must correspond to the output of:

```
> cd <basedir>
> find . -type f -exec grep -A2 -i B2DROP {} \;
```

- At time of writing an exception to the previous comment is the UCIS4EQ/Setup/SCC/indexPriority/* path pattern currently in disuse inside https://b2drop.bsc.es. Paths identifying with that pattern and referenced in the code can be disregarded.

6.5- Create Salvus output directory

The solver, Salvus, persists some of its output files. For that reason you need to create:

```
/gpfs/scratch/<your_hpc_group>/<your_hpc_username>/Eflows4hpc/Ucis4eq/
```

in MN4's GPFS scratch storage volume.

6.6- Modify setup files in <basedir>/Ucis4eq/data/

File: **DAL.json**

```
{"BSC_B2DROP": {
    "user": "<your_bsc_b2drop_username>",
    "pass": "<your_bsc_b2drop_passwd>",
    "url": "https://b2drop.bsc.es/remote.php/webdav/",
    "datamapping": "UCIS4EQ/StaticDataMappingV2_Jorge.json",
    "output": "UCIS4EQ/Runs"
  },
  "local": {
     "path": "/workspace/"
```

```
  },
  "BSC_DT":{
    "user": "<your_bsc_hpc_username>",
    "url": "dt01.bsc.es",
    "path": "/gpfs/scratch/<your_bsc_hpc_group>/<your_bsc_hpc_username>/Eflows4hpc/
Ucis4eq"
  },
  "ETH_DAINT":{
      "user": "your_pd_hpc_username",
      "url": "daint.cscs.ch",
      "proxy": "your_pd_hpc_username@ela.cscs.ch",
      "path": "/path/on/Piz-Daint/to/UCIS4EQ/"
  }
}
```

File: **Sites.json**

```
{
  "MN4": {
    "user": "<your_bsc_hpc_username>",
    "url": "mn1.bsc.es",
    "path": "/gpfs/scratch/<your_bsc_hpc_group>/<your_bsc_hpc_username>/Eflows4hpc/
Ucis4eq/",
    "resources": {
      ....
```

- The rest of the `Sites.json` file, specifically the section identified by the "SalvusRun" key, requires modification to conform to the resources you can commandeer on the computing infrastructure. Your resource privileges are partially controlled by Slurm, and can be visualized with:

```
> ssh -l <your_bsc_hpc_username> dt01.bsc.es
> bsc_queues
QUEUE NAME          MAX TIME     MAX PROC
bsc_case            2-00:00:00   2400       <- maximum duration: 48h, and 50 nodes
debug               02:00:00     768        <- maximum duration: 2h, and 16 nodes
interactive         02:00:00     4          <- maximum duration: 2h, and 2 nodes
```

for an BSC HPC user from the CASE dept.

In debugging mode or when beginning as a UCIS4EQ user, choosing "debug" will afford you maximum run-times of 7200s and a maximum of 16 nodes for 768 processes (16 nodes x 48 CPU/node). You can modify the rest of the `Sites.json` file accordingly, leaving `"1"` for the number of nodes to be used whenever the associated task is sequential (not parallelized).

6.7- [Optional] Create a second SSH asymmetric 4096 bit RSA key pair w/ a blank passphrase on your local host

- This second key pair may be needed for the local containers executing microservices locally to execute and monitor Salvus on MN4. The key pair's private key is embedded in secondary Docker images built from the base Ubuntu 20.4 Docker image used by UCIS4EQ. In a second step, as `docker-compose` prepares to launch services, docker containers are built and run from the secondary images.

```
> cd ~/.ssh
> ssh-keygen -t rsa -b 4096 -C <your_bsc_email> -f ~/.ssh/docker_mn_id_rsa -P ""
> chmod go -r ~/.ssh/docker_mn_id_rsa.pub
> ssh-copy-id <your_bsc_hpc_username>@dt01.bsc.es
```

The last command will actually copy your public key to the remote user's `~/.ssh/authorized_keys` (creating the file, and directory, if necessary).

7. Running UCIS4EQ

7.1- The runtime-execution (RTE) landscape (Part 1)

Before raising Docker containers for each microservice, you will in general need to login to the Gitlab's registry to have access to the Docker images stored there:

```
> docker login registry.gitlab.bsc.es/wavephenomenagroup/ucis4eq -u \
      <your_gitlab_username> --password-stdin <<< <your_gitlab_access_token>
```

In case of success "Login succeeded" is displayed in terminal.

To raise containers, simply issue:

```
> HPC_RUN_PYCOMPSS="<True|False>" SSH_PRV="$(\cat ~/.ssh/docker_mn_id_rsa)" \
      LOCATION="<location>" PD_USER="<username>" docker-compose up
```

or its simplified equivalent version:

```
> HPC_RUN_PYCOMPSS="<True|False>" SSH_PRV="$(\cat ~/.ssh/docker_mn_id_rsa)" \
      docker-compose up
```

In the above `HPC_RUN_PYCOMPSS="False"` is the default, whereby each task in the UCIS4EQ workflow is submitted seperately to the SLURM queue. By contrast when `HPC_RUN_PYCOMPSS="True"` is specified all successive tasks of the workflow are submitted as only one job in the SLURM queue. The latter makes possible the cancellation of a complete workflow execution instance and confers malleability to UCIS4EQ.

A loose enumeration of workflow tasks is:

- Slipgen
- SalvusPrepare
- SalvusRun
- SalvusPost
- SalvusPostSwarm
- SalvusPlots

In the second command above two RTE variables are left undefined:

- LOCATION specifies the cluster on which Salvus is run.
- PD_USER is required to launch Salvus on the ETH cluster named "Piz Daint".

When **not** executing on the ETHZ cluster, Piz Daint, you can ignore those two variables, provided you also edit the file <basedir>`/Ucis4eq/deployment/dockers/Dockerfile-credentials`, commenting out the following line:

```
#RUN ssh $PD_USER@ela.cscs.ch ssh-keyscan daint.cscs.ch >> \
      /root/.ssh/known_hosts;\ exit 0
```

When run, the code creates a Docker network (*`Network ucis4eq_default`*) and launches 7 Docker containers (DCs) as decribed in Table 1 below. Those constitutes your RTE landscape.

Irrespectively of whether services are launched automatically or manually, as soon as all them are deployed you will need to feed your idling services data for them to run an actual simulation instance. This is done in two ways (with or without a listener service) as described in the next section (Section 7.2).

	Containers	Images at registry.gitlab.bsc.es/wavephenomenagroup/ucis4eq/	Services	Deployment
1	ucis4eq-DAL-1	mongo:4.2	DAL	Auto
2	ucis4eq-workflowmanager-1	workflowmanager	workflowmanager	Auto/Manual
3	ucis4eq-dashboard-1	dashboard	dashboard	Auto/Manual
4	ucis4eq-salvus-1	salvus	salvus	Auto/Manual
5	ucis4eq-microservices-1	microservices	microservices	Auto/Manual
6	ucis4eq-slipgen-1	slipgen	slipgen	Auto/Manual
7	ucis4eq-listener-1	listener	listener	Auto/Manual

***Table 1**:From their respective images, respectively containers and services can be run and deployed automatically (Auto) and/or manually (Manual).*

7.2- Create a job ID and feed a seismic event to UCIS4EQ

7.2.1 Active "*listener*" service

When the "*listener*" service is active, UCIS4EQ sends periodic pull request every 60s to one or more of the major seismological agencies and FDSN data centers online, e.g.:

"IRIS": "http://service.iris.edu/fdsnws/"
"INGV": "http://webservices.ingv.it/fdsnws/"
"IMO": "http://www.orfeus-eu.org/fdsnws/"
"GEOFON": "https://geofon.gfz-potsdam.de/fdsnws/"
"SCEDC": "http://service.scedc.caltech.edu/fdsnws/"
"NCEDC": "http://service.ncedc.org/fdsnws/"
"NOA": "http://eida.gein.noa.gr/fdsnws/"
etc...

The "*Listener*" service pulls new events, whose specifications format complies with the FDSN RESTful web service characteristics. Those selected for urgent computing following specific criteria (including Moment magnitude, etc.) are automatically included in the UCIS4EQ computing workflow.

7.2.2 Inactive "*listener*" service

When the "*listener*" microservice is not used:

- compute jobs are not automatically meted a unique job id (UUID),
- seismic events are not automatically pulled from various seismological agencies across the world and fed to the UCIS4EQ flowmanager.

In that latter case, UCIS4EQ users need first to specify a job ID value for the key "`uuid`", in a specific JSON file. This can be done manually with any text editor, or via CLI like so:

```
> jq '.uuid = "<new_job_id>"' <my_data.json> >| tmpfile
> mv tmpfile >| <my_data.json>
```

Followed by:

```
> curl -X POST -H 'Content-Type: application/json' -d \
      @<my_data.json> http://127.0.0.1:5001/PyCOMPSsWM
```

The latter command, `` `curl -X ...` ``, specifies a custom request method (`POST`) to use when communicating with the UCIS4EQ Workflow Manager (WM) service based on PyCOMPSs, and reachable via the HTTP server client at `` `127.0.0.1:5001` `` on port 5001.

As an example the instruction may submit the file content of <basedir>/data/SAMOS_EQ_Event_DEMO.json (located in your local UCIS4EQ repo) to the Workflow Manager (WM). The set of complete instructions becomes:

```
> cd <basedir>/Ucis4eq
> HPC_RUN_PYCOMPSS="<True|False>" SSH_PRV="$(\cat ~/.ssh/docker_mn_id_rsa)" \
      docker-compose up
> jq '.uuid = "<new_job_id>" | .ensemble = "<exec_mode>"' \
      data/MED_SAMOS_IZMIR_IRIS_test.json >| tmpfile
> mv tmpfile >| data/MED_SAMOS_IZMIR_IRIS_test.json
> curl -X POST -H 'Content-Type: application/json' -d \
      @data/MED_SAMOS_IZMIR_IRIS_test.json http://127.0.0.1:5001/PyCOMPSsWM
```

In the above the event definition file, <basedir>/Ucis4eq/data/MED_SAMOS_IZMIR_IRIS_test.json, has the following structure:

```
{
 "sources": {
      "INGV": {
            "data": "xx",
            "timestamp": 0,
            "query": "xx"
            }
      },
 "uuid": "<new_job_id>",
 "alerts": [{
      "agency": "USER EVENT",
      "time": "2020/10/30, 11:51:27",
      "latitude": 37.918,
      "longitude": 26.79,
      "magnitude": 7.0,
      "depth": 21000,
      "elapsedtime": 2456,
      "description": "Samos Izmir manual trigger",
      "cmt": {"GCMT": {"strike": 270,
                       "dip": 37,
                       "rake": -95
                       }
            }
      }],
 "ensemble": <exec_mode>
}
```

where the key-value pair "ensemble": <exec_mode> is optional and <exec_mode> may have one of three values :

"statistical" (default)
"onlyOne"
"seisEnsMan"

7.2.3 Manual service deployment

Each microservice detailed in Table 1 can be deployed manually and run independently from others. This is useful in debug mode to isolate outputs from each service, i.e. to avoid stdout being flooded by mixed outputs coming from all microservices. The best display configuration to adopt for that purpose is to open as many terminal windows or PTSs as you have services. In each PTS window or tile, issue one of the following commands taken from the sequence hereafter:

```
> HPC_RUN_PYCOMPSS="<True|False>" SSH_PRV="$(\cat ~/.ssh/docker_mn_id_rsa)" \
docker-compose -f docker-compose-debug.yml up workflowmanager

> HPC_RUN_PYCOMPSS="<True|False>" SSH_PRV="$(\cat ~/.ssh/docker_mn_id_rsa)" \
docker-compose -f docker-compose-debug.yml up dashboard

> HPC_RUN_PYCOMPSS="<True|False>" SSH_PRV="$(\cat ~/.ssh/docker_mn_id_rsa)" \
docker-compose -f docker-compose-debug.yml up microservices

> HPC_RUN_PYCOMPSS="<True|False>" SSH_PRV="$(\cat ~/.ssh/docker_mn_id_rsa)" \
docker-compose -f docker-compose-debug.yml up slipgen

> HPC_RUN_PYCOMPSS="<True|False>" SSH_PRV="$(\cat ~/.ssh/docker_mn_id_rsa)" \
docker-compose -f docker-compose-debug.yml up salvus

> HPC_RUN_PYCOMPSS="<True|False>" SSH_PRV="$(\cat ~/.ssh/docker_mn_id_rsa)" \
docker-compose -f docker-compose-debug.yml up listener
```

As the MongoDB service is alway automatically launched, no command is needed to launch it. When the "*listener*" or "*dashboard*" services are not required, simply do not issue the corresponding "`docker-compose up`" command. Once issued the above commands will trigger the deployments of each corresponding service. Observe the outputs in each terminal to ensure that the corresponding service is satisfactorily deployed . Finally, in yet another terminal window, issue:

```
> jq '.uuid = "<new_job_id>"' <my_data.json> >| <my_data.json>
> curl -X POST -H 'Content-Type: application/json' -d \
      @<my_data.json> http://127.0.0.1:5001/PyCOMPSsWM
```

where, as noted previously, `<my_data.json>` is the relative path to the static input file identifying a seismic event. So Salvus may perform its magic, recall that paths when relative are based on <basedir>/ as defined earlier.

7.3- Test UCIS4EQ with a dry run

You may decide to launch dry runs. They are essentially attempts at real execution launches, without executing the compute-job. This is generally done to ensure that all input files and the general compute setup is correct without committing jobs to the infrastructure.

To do so simply comment out job submission instructions:

```
submission.run ...
```

in `salvus.py` and `SlipGen.py`.

7.4- Monitor the current UCIS4EQ job

You can carry out routine checks on your current jobs' executions.

7.4.1 Monitoring with Docker

`> docker ps [-a|-l]` # lists container(s) on the local HOST

`-a` = `all` (active and inactive)

`-l` = `latest` (latest created container)

7.4.2 Monitoring via Flask server

Visit the web page: http://127.0.0.1:8050 to visualize bottlenecks and errors on compute jobs.

7.4.4 Monitoring via CLI on MN4

```
MN4> watch squeue                    # type CTRL-C to quit.
```

7.5- Stop/quit a UCIS4EQ job

Issue `CTRL-C` twice on your local host tty for each microservice launched independently in order to end their respective container execution, followed by:

```
> docker compose down
```

The latter command will erase all current UCIS4EQ docker containers from your local host.

Additionally you can verify that your Marenostrum file tree was correctly populated with result files. This however does not guarantee a correct job completion as no error control is performed on the correct execution of Salvus from UCIS4EQ. The file tree lookup command is:

```
> tree -L 4 /gpfs/scratch/<your_hpc_group>/<your_hpc_username>/Eflows4hpc/Ucis4eq/
```

Take note of the `<code_name>` given to the execution of interest and as shown in the JSON file:

```
<basedir>/Ucis4eq/data/<event_ref>.json
```

at key `"uuid"`. This `<code_name>` is used by UCIS4EQ to generate a new tar ball that contains all result plots:

```
/gpfs/scratch/<your_hpc_group>/<your_hpc_username>/Eflows4hpc/Ucis4eq/event_<code_name>/
salvus_plots/event_<code_name>.tar.gz
```

7.6- The runtime-execution (RTE) landscape (Part 2)

UCIS4EQ is a realtively complex workflow and uses 3 distinct store and compute systems to run:

A) your local system where the UCIS4EQ code resides and Docker containers will be launched, `<basedir>/Ucis4eq/data/*`

B) the B2DROP instance at BSC, to store both initialization files and ultimately result files (e.g. plots of shake maps),

C) the HPC infrastructure where:

1. SlipGen, the kinematic rupture engine from SCEC[2], and the solver, Salvus, are installed and run, controlled by the corresponding Docker containers (DC-4 and DC-6 in Table 1).

   - For given kinematic rupture (event bound) and velocity models (geology bound), Salvus expects the following `.yaml' file:

   ```
   /gpfs/projects/bsc44/earthquake/UCIS4EQ/salvus_urgent_simulations_setup/data/
   salvus_parameters_template/salvus_parameters.yaml
   ```

   to be present and correctly informed. Unexperienced users should NOT modify this file without a thorough knowledge of what each of its parameters does.

2. Large datasets are stored for computational convenience.
   In this way the workflow, when launched, need not upload large data chunks to the HPC infrastructure filesystem, with a large wait time.

4. Intermediate results are stored.
   For instance the rupture model produced by SlipGen are stored in the directories:

   ```
   /gpfs/scratch/bsc21/<your_hpc_username>/Eflows4hpc/Ucis4eq/event_<code_name>/
   ```

2 SCEC: Southern California Earthquake Center, University of Southern California
3651 Trousdale Parkway #169, Los Angeles, CA 90089-0742 - USA

```
trial_<fault_plane_ID>.slip1/slipgen/scratch/
```

For a given velocity model and event, restarting UCIS4EQ will only recalculate the kinematic rupture model if the corresponding `scratch/' directory is absent. Otherwise UCIS4EQ assumes that the existing rupture model is already available and skips that compute step for that event computation.

7.7- Manual SlipGen DC execution as standalone code

[This section requires further work and verifications]

It is possible to run the Graves-Pitharka Slip Generator DC, `slipGen`, in interactive mode as standalone code, for instance to produce kinematic rupture maps/models on demand for a specific seismic event.
To do so, one method consists in raising the DC and initiating code execution via the specified entry-point, as in:

```
> docker run -it -v "$PWD:/workspace" -v \
      "<basedir>/Ucis4eq/data/Sites.json:/opt/Sites.json" \
      --entrypoint="/opt/scripts/launcher.sh" \
      registry.gitlab.bsc.es/wavephenomena_group/ucis4eq/slipgen \
      -v crust1p0.1d -s GP_inputs.src -t 0.010000 -x 2.500000 -y 1.500000 \
      -o <filename>
```

where every option listed after the container name is parsed by the code executing inside the Docker container. In this case:

Option flag	Option parameter	Description
-v	crust1p0.1d	Velocity model
-s	GP_inputs.src	Graves-Pitarka kinematic rupture stochastic source
-t	0.010000	
-x	2.500000	
-y	1.500000	
-o	<filename>	Output file with kinematic rutpure model produced.

As specified in `<basedir>/Ucis4eq/deployment/dockers/Dockerfile-slipgen`, port 127.0.0.1:5002 becomes exposed, while `<basedir>/Ucis4eq/code/services/slipGenService.py` executes and arguments are passed to the DC's entry point.

Another method consists in raising the container and delaying execution until an execution payload is POSTed through one of the available endpoints, as in:

```
> docker run -it -v "$PWD:/workspace" -v \
      "<basedir>/Ucis4eq/data/Sites.json:/opt/Sites.json" \
      --entrypoint="" \
      registry.gitlab.bsc.es/wavephenomena_group/ucis4eq/slipgen

> curl -X POST -H 'Content-Type: application/json' -d @<data.json> \
      http://127.0.0.1:5002/<endpoint_label>
```

where <endpoint_label> can be either "preGraves-Pitarka" or "Graves-Pitarka", and `<data.json>` must be correctly specified. File `<data.json>` has the following structure:

```
{'resources': "...",
 'region': {'file_structure': "...",
            'id': "...",
            'available_fmax': [....]},
 'trial': "...",
 'CMT': "...",
 'event': "...",
```

```
'ensemble': "...",
'setup': {'fmax_policy':....,
          'dwid': ....,
          'dlen': ....,
          'corner_freq': ...,
          'model': ...,
          'dt': ...,
          'path': ...,},
}
```

8. Miscellanea

8.1- Clean-up and garbage collection

After each run, containers normally keep on running. To exit them, issue CRTL+C twice in the corresponding terminal and further issue:

```
> docker compose down
```

or, in case other docker containers, remnants of other code executions, are up and running:

```
> docker [container] ps -q | cut -d' ' -f1 | xargs docker [container] stop
> docker [container] ps -q | cut -d' ' -f1 | xargs docker [container] rm
```

You can also choose to remove all Docker containers (including those currently running) forcibly:

```
> docker [container] rm -f $(docker [container] ps -a -q)
```

To list locally available Docker images:

```
> docker images
```

or, equivalently

```
> docker image ls
```

To remove selected Docker images, find a <pattern> common to all of them and place it in the pattern matching condition in the `awk' cmd below:

```
> docker rmi -f < <(docker images | awk '/pattern/ {print $3}')
```

8.2- (Re-)Build Docker images

This may justified by the creation of a new microservice or by the modification of existing code.

- login into the GitLab repository that holds the Docker registry:

```
> docker login [registry.]gitlab.bsc.es/wavephenomenagroup/ucis4eq
```

- if the Docker CLI plugin for extended build capabilities, `docker-buildx' is installed, create and run a container with the extended builder BuildKit:

```
> docker buildx create –user
```

- create and push images to Git's Docker image registry, e.g. for the ***simulation*** microservice:

```
> docker buildx build --no-cache \
      -f deployment/dockers/Dockerfile-simulationServices> \
      -t registry.gitlab.bsc.es/wavephenomenagroup/ucis4eq/simulation:latest \
      --load --push \
      .
```

or more simply, if the Docker CLI plugin for extended build capabilities, `docker-buildx' is NOT installed:

```
> docker build --no-cache \
      -f deployment/dockers/Dockerfile-simulationServices> \
      -t registry.gitlab.bsc.es/wavephenomenagroup/ucis4eq/simulation:latest \
      --load \
      .
> docker push registry.gitlab.bsc.es/wavephenomenagroup/ucis4eq/simulation:latest
```

- logout and optionally kill and remove the *BuildKit* running container until needed again.

```
> docker logout
> ps -i buildkit < <(docker container ls)
[...]
> docker container kill <container_id>
> docker container rm <container_id>
```