Testing weather code on multiple HPC systems

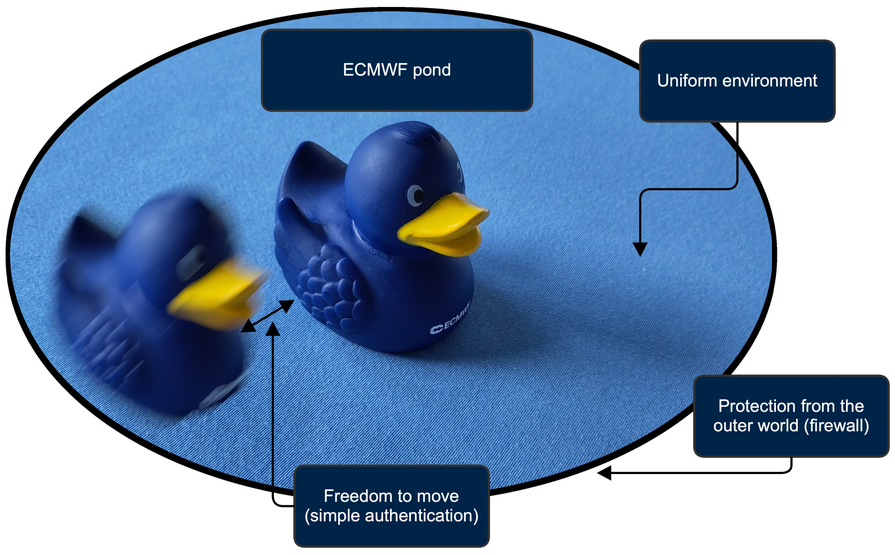
Johannes Bulin[1*]

(1) ECMWF; (*) johannes.bulin@ecmwf.int

1. Single-system bliss

You should test your code - for obvious reasons! If you only have to do it on your organisation's HPC, you can usually rely on

- shared file systems
- common authentication procedures
- certain control over the used hardware.



Now try to run it on a different HPC. You most likely

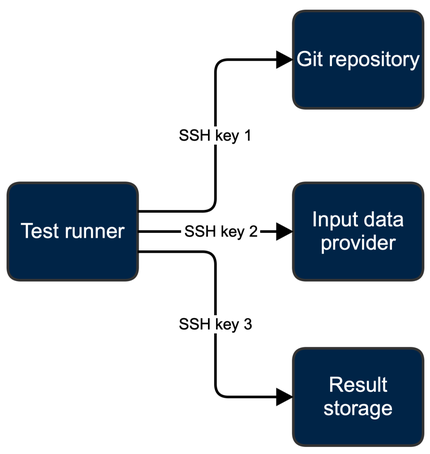
- need to login via SSH - using a designated SSH key
- can't change the system configuration
- have different hardware/compilers/accelerators
- don't have direct access to all your files.

2. SSH key handling

You want to automatically test your weather code on a different HPC system now? You may need

- the actual source code
- initial data
- some storage for the results.

All these components may be placed on different servers – and access usually works with SSH keys (unless you stay organisation-internal!).



In this case, it would be great to

- stop the "diffusion" of SSH keys to different systems
- keep private service account SSH keys (CI keys) invisible to users
- make it easier to manage the different SSH keys.

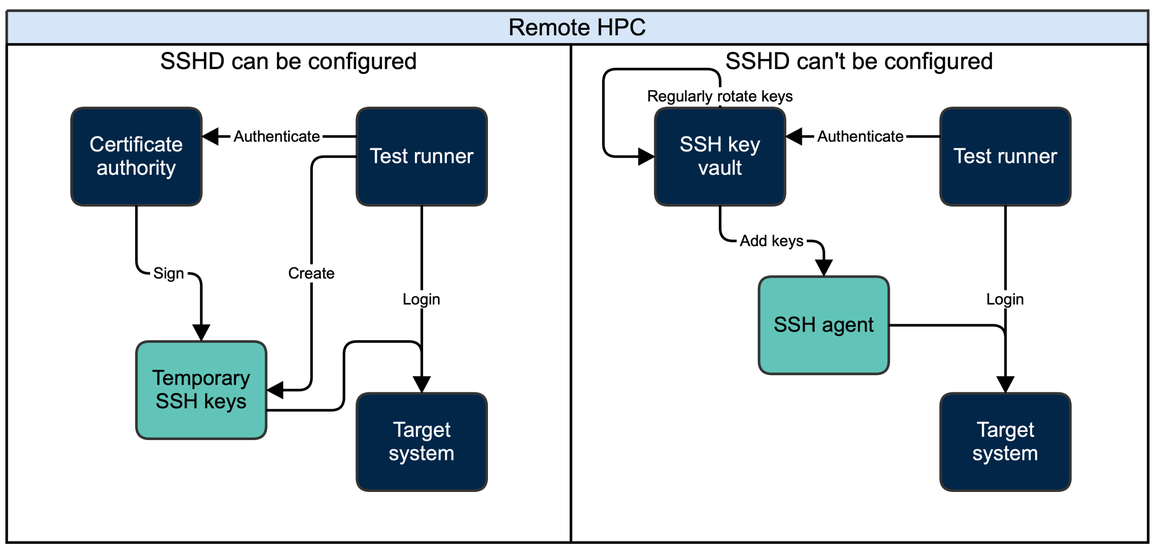
The approaches that we've been using differ, depending on whether we can change the SSH configuration on the target HPC system or not.

Remote HPC (SSHD) can be configured

1. Enable certificate-based login on remote HPC.
2. Setup certification authority (for example Teleport or Hashicorp Vault).
3. Test runner creates temporary SSH keys, gets them signed by CA.
4. Login to target HPC with signed SSH keys.

Remote HPC (SSHD) can't be configured

1. SSH keys are stored in central vault (Hashicorp Vault).
2. Test runner uses token to obtain SSH keys.
3. Keys are added to SSH agent and are forwarded (no spread to other systems).
4. SSH keys are rotated frequently and are used for no other purpose.



3. Result verification

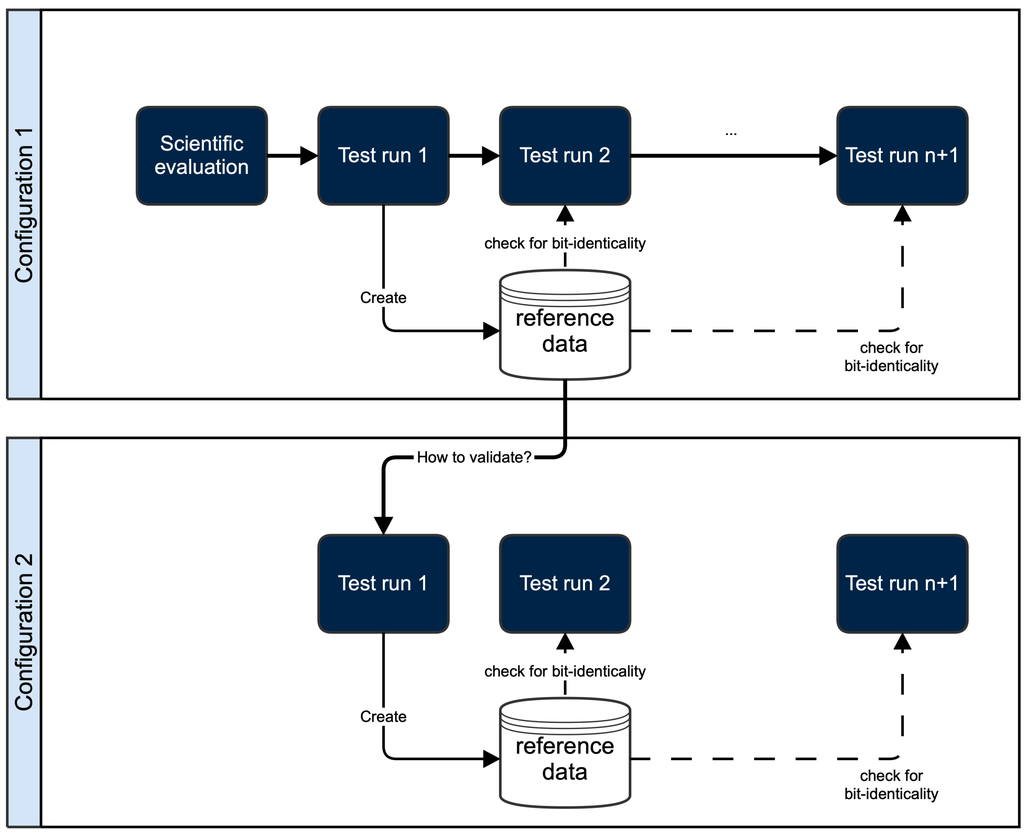
When running integration tests, results must be verified of course. After model changes, a full scientific evaluation may be necessary. Subsequent technical changes can in most cases be verified by checking for bit-identicality.

Bit-identicality is usually only guaranteed when exactly the same configuration is used which means that

- the hardware
- the compiler
- the compiler flags
- the parallel setup

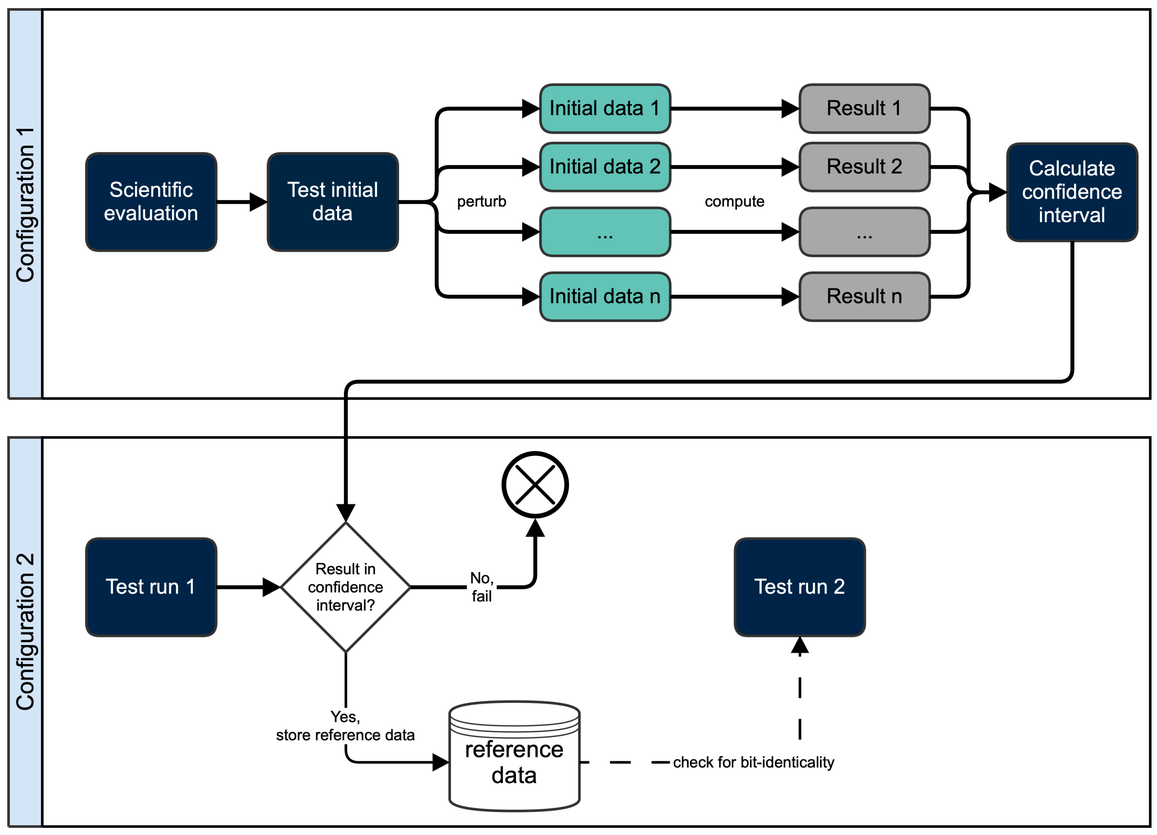
must be the same (plus some additional requirements).

What should be done if you want to verify your results on a wide range of systems but don't have a basement full of scientists to do a complete scientific evaluation for each system?



Differences in the results should only stem from floating point inaccuracies. Everything else is most likely a bug! Therefore, we use a simplified version of the approach in *"An ensemble-based statistical methodology to detect differences in weather and climate model executables"* (C. Zeman and C. Schär, Geoscientific Model Development, doi: 10.5194/gmd-15-3183-2022):

1. Run a full (scientific) code validation on a system of your choice.
2. On this system, run an ensemble of simulations, by disturbing the initial data (magnitude similar to floating point accuracy).
3. Retrieve the results of each ensemble member.
4. For each result type, determine the expected range/confidence interval using the ensemble results.
5. Run simulation on another configuration. Check that it satisfies the ranges.



Funded by the European Union

Destination Earth implemented by ECMWF esa EUMETSAT