Benchmarking WLCG resources using HEP experiment workloads: infrastructure for container builds

Andrea Valassi (CERN IT-DI)
On behalf of the WG

Benchmarking Pre-GDB, 8 October 2019 – CERN
https://indico.cern.ch/event/739897

Recap: project packages and repositories

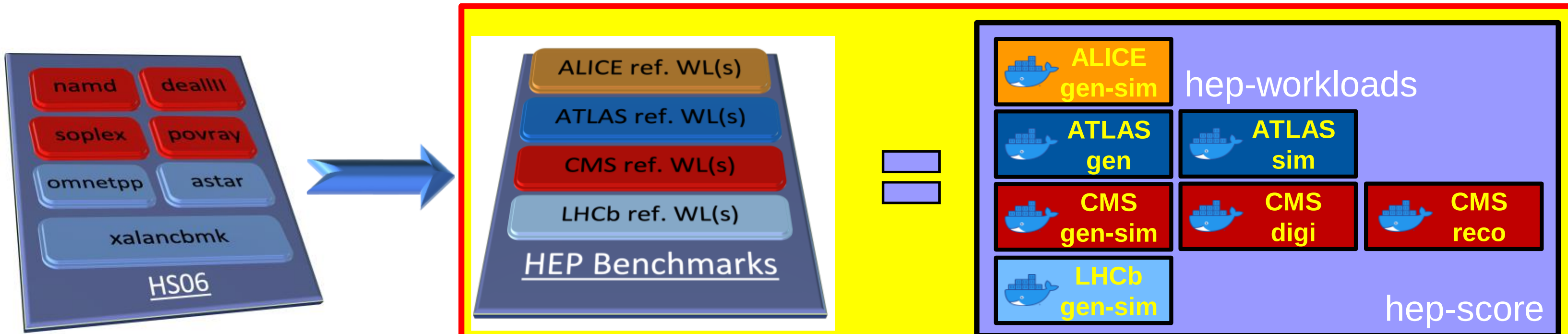
- Three repositories under https://gitlab.cern.ch/hep-benchmarks:
  - *hep-workloads* (the common infrastructure and individual workloads)
  - *hep-score* (single-number benchmark aggregator from several WLs)
  - *hep-benchmark-suite* (automate execution, collect results in a database)

- *Outline: in this talk I will essentially describe the internals of hep-workloads*
  - The other two packages will be described in subsequent talks

Recap: an alternative to HEP-SPEC06, benchmarking CPUs using HEP workloads

- For comparison, how did we converge on HEP-SPEC06 in the past?
  - We analysed several subsets of the SPEC benchmark suite
  - We chose one with high correlation and similar patterns as HEP WLs

- *By construction*, using HEP workloads directly is guaranteed to give
  - A score with high correlation to the throughput of HEP workloads
  - A CPU usage pattern that is similar to that of HEP workloads
  - For event generation, detector simulation, digitization, reconstruction….



What enables doing this now is the availability of *container technology!*

docker

Why many independent containers in hep-workloads ?

1. Technical: encapsulation
   – Each container includes all that is needed to run one workload, and only that

2. More fundamental: preserve the many degrees of freedom in the problem
   – "A computer system's performance cannot be characterized by a single number or a single benchmark. […] Many users (decision makers), however, are looking for a single-number performance characterization. […] There are no simple answers. *Both the press and the customer, however, must be informed about the danger and the folly of relying on either a single performance number or a single benchmark.*"

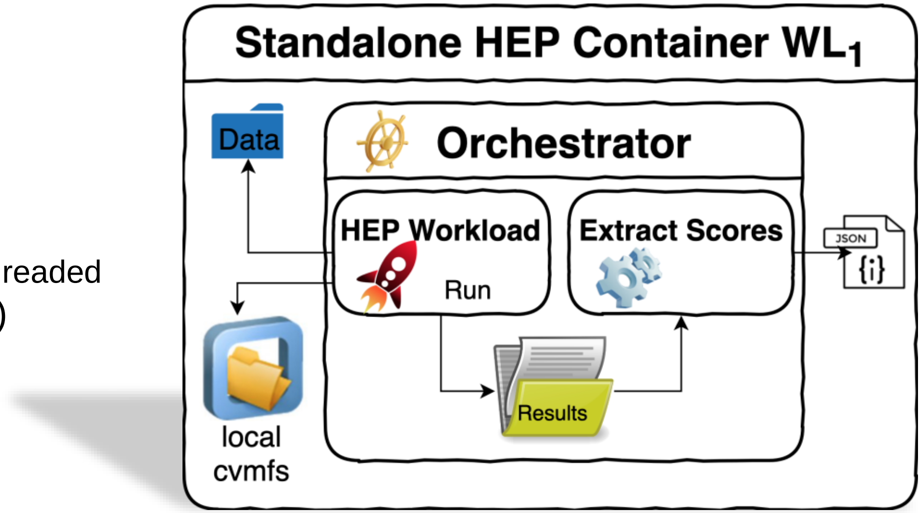
   Kaivalya M. Dixit, Overview of the SPEC Benchmarks,
   in J. Gray (Ed.), The Benchmark Handbook for Database and Transaction Systems, 1993.

   – Each HEP workload stresses different components of a computer system
     - Some are I/O intensive, others not; some are vectorized, others not…
     - *Our infrastructure provides a single benchmark number (hep-score), but keeps the possibility to separately measure and record different WL benchmarks independently*

The hep-workloads containers

- Encapsulation: for each HEP workload, build a standalone benchmark container
  - Portable and self-contained (no need for network connectivity)
  - As small as possible (include all dependencies needed to run the workload, and only those)
  - Doing always the same thing (results should be as reproducible as possible)

- Components of each HEP workload container
  - Software repository (O/S and /cvmfs)
  - Input data (event and conditions data)
  - An orchestrator script (benchmark driver)
    - Sets the environment
    - Runs (many copies of) the application
      - Each copy may be multi-process or multi-threaded
    - Parses the output to generate scores (json)



The hep-workloads CI and registry

- Individual HEP workload container images are built and distributed via gitlab
  - The **gitlab CI** (continuous integration) builds and tests new images on commit
  - If the build is successful, the CI pushes these (versioned) images to the **gitlab registry**

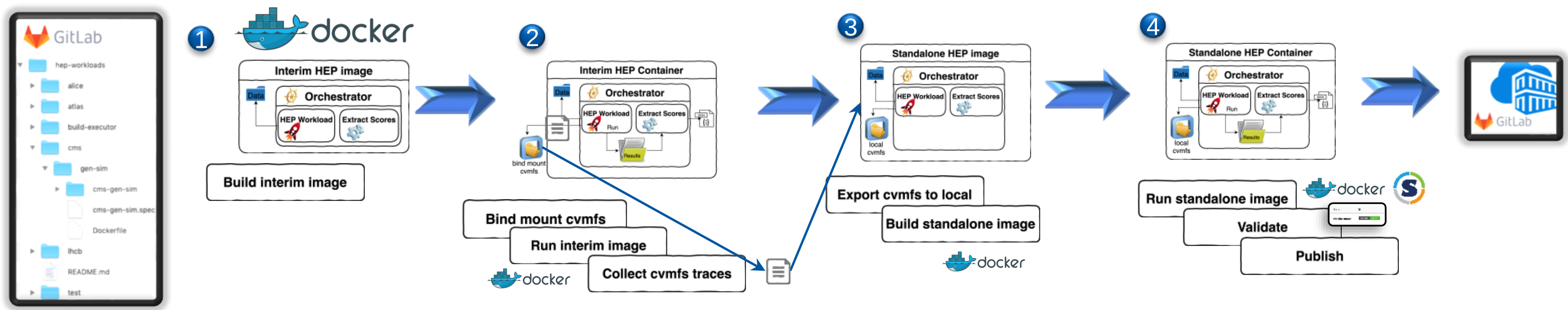
- The images are built as Docker containers
  - But they can be executed both via **Docker** and **Singularity**

```
$ docker run -v /my_host_path:/results $IMAGE
$ singularity run -B /my_host_path:/results docker://$IMAGE
```

  - A json summary and detailed logs are then found in /my_host_path on the host system

Build procedure in the hep-workloads CI

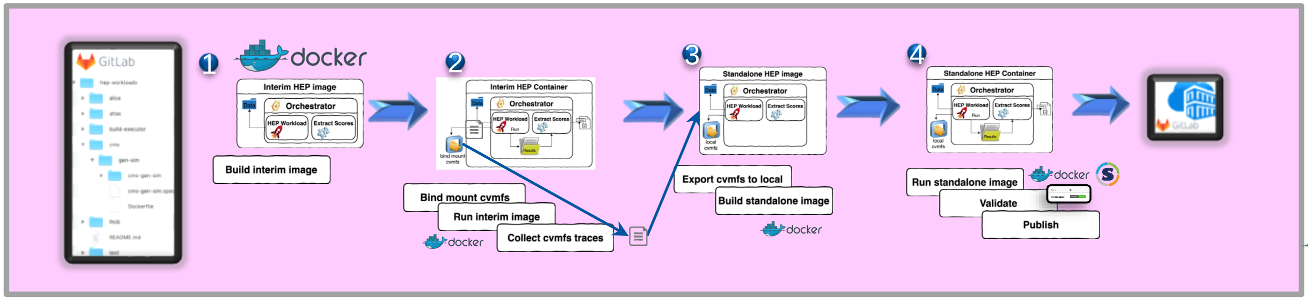
- *Main idea: experiment software is on /cvmfs, discover what is needed in a dry run*
- *Enabling technology: cvmfs tracing mechanism*



- Starting from a gitlab repo containing only the CI and WL orchestrator scripts:
  1. Build an interim image, where /cvmfs is the standard network-connected service
  2. Run the WL from that image, generating cvmfs traces listing which files were accessed
  3. Build the final standalone image, where /cvmfs is a local folder, copying all relevant files
  4. Test the WL from that image (both in Docker and Singularity), push it to the gitlab registry

The hep-workloads builder image

- Individual WL images (interim and final) are built within a builder image
  - This is common for all workloads and changes very infrequently
    - Consistently check error, save logs and cvmfs traces for all workloads
  - It contains docker, singularity and all other build and tests packages
  - It is itself built by the CI, using standard (shared) runners at CERN



WL image build process within the builder image

- Technicality: actually the builder image starts another builder image
  - And within that internal builder image it builds individual workloads
  - This additional “inception” is needed to manage docker volumes and privileges…

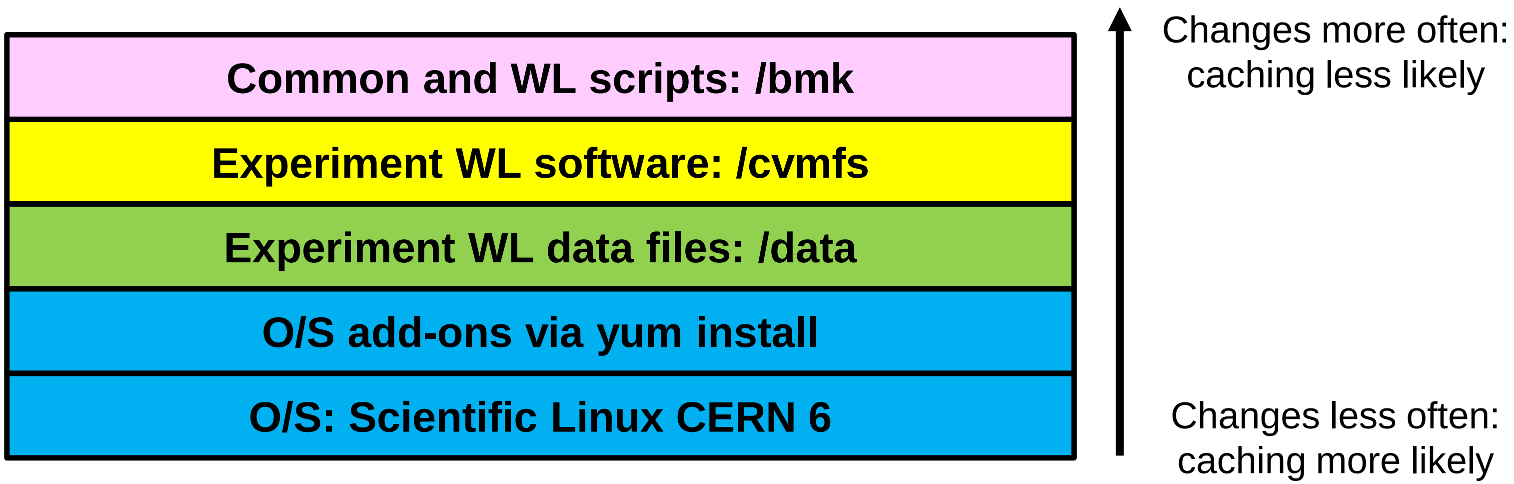
The hep-workloads gitlab CI workers cluster

- We chose to set up our own (non-shared) CI workers
  - Docker privileged mode, keep large logs, define custom host volumes…
  - Individual WLs are built within the builder image running on these nodes
- In time we moved from two standalone nodes to a Puppet cluster (4 nodes)
  - More consistent setup, easier maintenance
  - The standalone nodes were often getting full disks, or docker network issues
- Each node is a VM with four logical processors (`nproc` equals 4)
  - Each node can only build one workload at a time
    - Because the build of MP and MT workloads may need all four virtual processors
  - We now do not always run many WL copies at build time to saturate the node
- Storage management involves caching and cleanup
  - Docker and singularity caches make the build process less storage hungry
  - Regular cleanups after each build seem ok to keep space under control so far

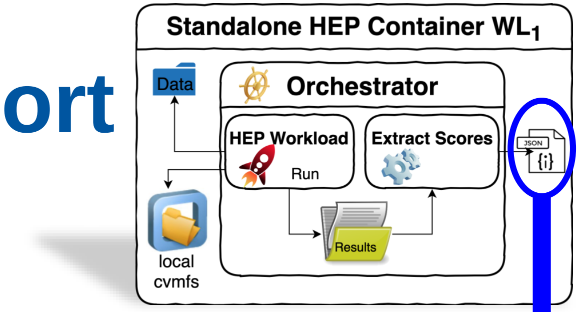
Docker layers in hep-workloads images

- Docker container images are always made up of *layers*
  - Translating Docker images to Singularity also keeps this layer structure unchanged
  - From the bottom up, these layers can be *cached* until the first difference is found

- The hep-workloads CI builds these layers to make them as cacheable as possible
  - The bottom layers contain what is expected to change least often
  - The top layers may change more frequently (across different workloads or versions)
  - Advantage in the CI: faster builds/tests, save storage space (both Docker and Singularity)
    - Advantage for users: faster tests, save storage space (if Docker and Singularity caches are set up)



The hep-workloads output report



JSON document with the essential information

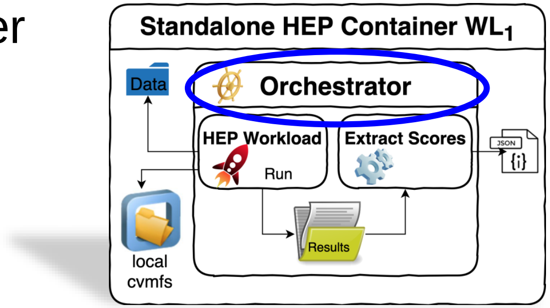
- Configuration parameters
  - #copies, #threads, #events, status
- Benchmark score: **total node throughput**
  - **Events per wall second** (sum over all copies)
    - Or events per CPU second in some cases
  - Details for each application copy
    - Statistics: mean, median, max, min…
- Additional metrics for performance studies:
  - Memory and CPU utilization
- Workload metadata
  - Description, version, checksum


```
"report": {
  "wl-scores": {
    "gen-sim": 0.4438
  },
  "wl-stats": {
    "CPU_score": {
      "max": 0.0226,
      "score": 0.1123,
      "median": 0.0225,
      "avg": 0.0225,
      "min": 0.0222
    },
    "throughput_score": {
      "max": 0.0892,
      "score": 0.4438,
      "median": 0.089,
      "avg": 0.0888,
      "min": 0.088
    }
  },
  "log": "ok",
  "app": {
    "bmkdata_checksum": "e57b3ad19144b7e9574b97056fb35d11",
    "cvmfs_checksum": "b2ab0e3bd4ba1333ebfc7dc49a024536",
    "bmk_checksum": "fc73ae9f18c4ef90791f097cd31b45dc",
    "version": "v1.0",
    "description": "CMS GEN-SIM of ttbar events, based on CMSSW_10_2_9"
  },
  "threads_per_copy": 4,
  "copies": 5,
  "events_per_thread": 100
},
```


The JSON produced by each individual WL are then embedded in the overall JSON produced by the hep-score aggregator

Many hep-workloads flavors, one common policy and one common driver

- General idea: enforce some commonality between different WLs
  - Common naming convention on scripts, images, directories…
  - Derive each WL orchestrator from a common bmk driver



- Consistent approach within the common bmk driver
  - Command line argument interpretation
  - Spawn several WL copies and check status code
  - Parse results and produce consistent json summaries
    - Work in progress in finalizing the json schema
  - Validate json summaries (e.g. linting)

Within hep-workloads containers: non-root user, read-only /cvmfs

- Within each container, the entry-point driver script is executed as root
  - For simplicity (access to /var, /tmp…), but this could be changed
- This led to a peculiar issue: the local /cvmfs could be overwritten
  - The ATLAS sim workload overwrote conditions on /cvmfs and failed
    - Not clear why the ATLAS software does that… but a fix was needed
- Workaround: make /cvmfs read-only when executing the workloads
  - Create a new user ‘bmkuser’ in the Dockerfile
  - Run all WLs as bmkuser via ‘su bmkuser’ in the common bmk driver

The hep-workloads containers: available images and work in progress

- **GEN and SIM workloads are available for all four LHC experiments**
- DIGI and RECO workloads are available for CMS, work in progress for ATLAS

- Download the container images from the gitlab registry:
  - gitlab-registry.cern.ch/hep-benchmarks/hep-workloads/alice-gen-sim-bmk:latest
  - gitlab-registry.cern.ch/hep-benchmarks/hep-workloads/atlas-gen-bmk:latest
  - gitlab-registry.cern.ch/hep-benchmarks/hep-workloads/atlas-sim-bmk:latest
  - gitlab-registry.cern.ch/hep-benchmarks/hep-workloads/cms-gen-sim-bmk:latest
  - gitlab-registry.cern.ch/hep-benchmarks/hep-workloads/cms-digi-bmk:latest
  - gitlab-registry.cern.ch/hep-benchmarks/hep-workloads/cms-reco-bmk:latest
  - gitlab-registry.cern.ch/hep-benchmarks/hep-workloads/lhcb-gen-sim-bmk:latest

Outlook: hep-workloads containers on heterogeneous resources (HPCs, GPUs…)

- All of the work on hep-workloads described so far refers to x86 architectures
- WLCG computing is expected to go well beyond x86 in the medium term future
  - Non-x86 HPC supercomputers (ARM, Power9, GPUs…) will probably play a large role
  - *The container build approach I described applies also in these cases with a few changes*
- By and large, the software of the experiments is not yet production-ready for this
  - Porting and validating it (and having the people to do that) is one of the first priorities
  - But our new benchmarks must be ready in time to do the accounting for these resources!
- Specifically: work is in progress on a HEP workload container involving GPUs
  - CMS event reconstruction, with optional GPU offload of pixel tracking (see Patatrack talk)

Conclusions

- After 10 years, HEP-SPEC06 no longer describes well enough HEP workloads
- Our solution: build a new benchmark directly from HEP workload throughputs
  - *Enabling technologies: Docker containers and cvmfs tracing mechanism*
- Implementation of image builds is based on gitlab CI runners
- Status: individual containers exist for GEN-SIM workloads of all four experiments
  - And for the DIGI and RECO of CMS (and soon for ATLAS too)
- Outlook: can extend the idea and implementation to HPCs and non-x86 resources
  - A container for a workload with optional GPU offload (CMS Patatrack) is being prepared