WMS and Computing Resources

Alexandre F. Boyer
2nd Virtual DIRAC User Workshop
May 10th 2022

DIRAC
THE INTERWARE

Introduction

The **DIRAC WMS** can:

- Interact with a variety of distributed components (Batch Systems, CEs...).
- Federate a large number of computing resources (mostly from Grids).

But:

- Software always requires more complex hardware.
- Computing infrastructure and funding models are changing.
- National science programs are consolidating computing resources and encourage using **Cloud systems** as well as **High-Performance Computers.**

Objectives

- **Support relevant distributed and heterogeneous computing resources** for one or more communities.
- **Get as many allocations as needed**, as fast as possible if necessary.
- **Use these allocations efficiently.** Tasks should be adapted to the underlying resources and respect the allocation conditions.

Table of Contents

- DIRAC Jobs
- Computing Resources
- Supplying Computing Resources with Jobs
- Efficient executions

DIRAC Jobs:
Definition, Structure & Requirements

Interfaces: link

Job: a type of container to acquire resources on a computing system.

- Input: executable, arguments, input data
- Output: stdout/stderr, output data

Various ways to manage jobs in DIRAC:

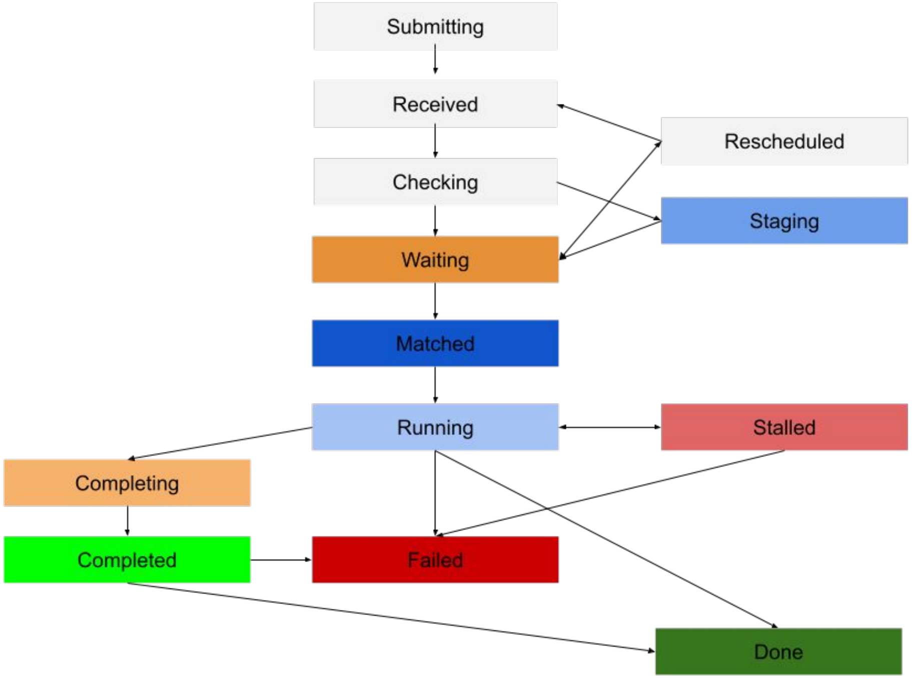
- CLI
- Python API
- WebApp

See the Interfaces presentation for further details.

Status: link

WMS jobs are proceeding through a chain of states from Submitting to *Done/Failed.*

v7r3 Introduction of a strict **JobState machine**: Forbid state transitions which are not allowed in the state machine definition.



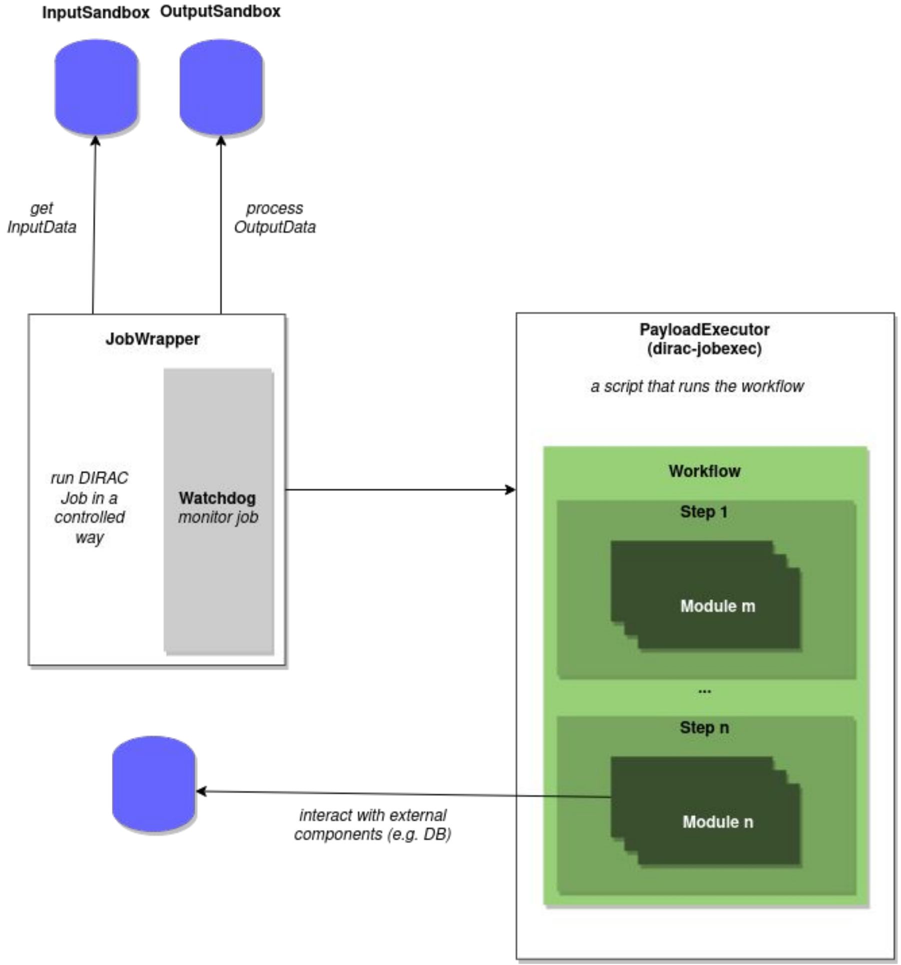
(Default) Structure: link

Executable, arguments and inputs are part of the workflow (modules to execute Scripts, Applications).

A workflow is defined in XML.

Executed within an allocation on a same worker node (not always true…).

Default behaviour: components can be set differently.

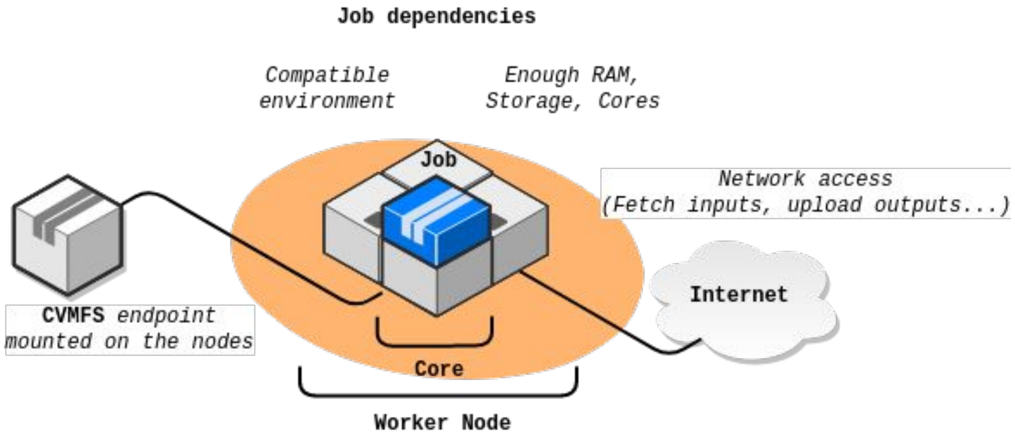

InputSandbox
OutputSandbox
get InputData
process OutputData
JobWrapper
run DIRAC Job in a controlled way
Watchdog
monitor job
PayloadExecutor (dirac-jobexec)
a script that runs the workflow
Workflow
Step 1
Module m
...
Step n
Module n
interact with external components (e.g. DB)

Requirements

CVMFS (link): getting complex software stack on distributed computing infrastructures. Not a strict requirement.

Getting a compatible environment: Singularity is used.

- Documentation here
- Within DIRAC here


Job dependencies
Compatible environment
Enough RAM, Storage, Cores
Job
Network access (Fetch inputs, upload outputs...)
Internet
CVMFS endpoint mounted on the nodes
Core
Worker Node

Computing Resources: Various computing paradigms and components

Declaring Computing Resources: link

Computing resources should be declared in the DIRAC CS Section */Resources/Sites* to be reachable.

- Sites: a virtual administrative domain
- Site: a physical domain
- CE: an entrypoint within a physical domain
- Queue: a logical partition within a Site to split resources according to their features

```
/Resources/Sites
LCG { #Sites
 LCG.CERN.cern { # Site
  CEs  {
   cern120.cern.ch { # CE
    # CE params
    Queues {
     normal # Queue
     {
      # Queue params
      ...
```

Computing Elements (CEs): link

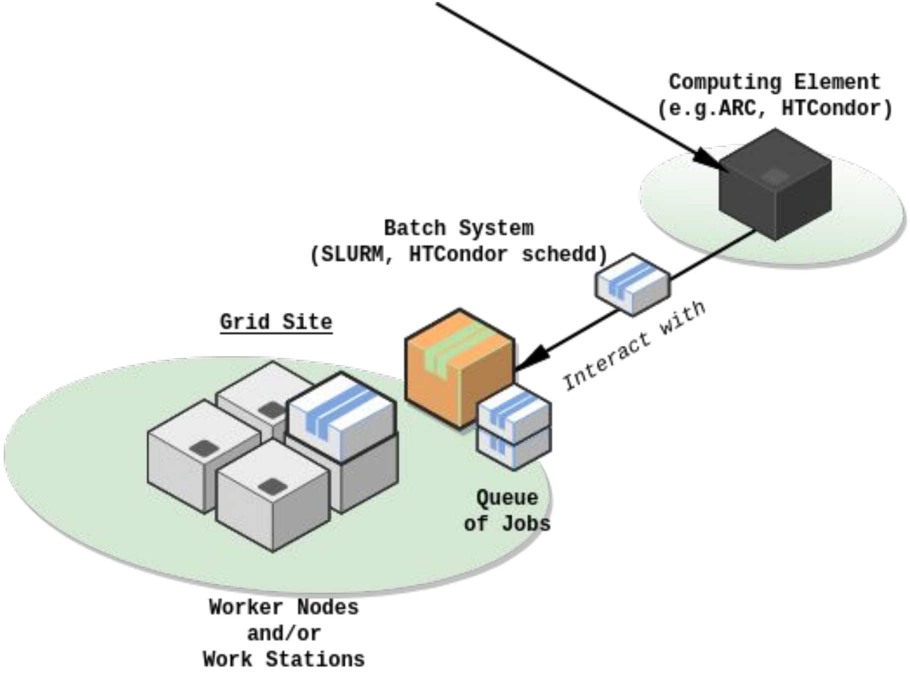
Intermediary elements between WMS and computing resources, aiming to ease the interactions with various and heterogeneous computing resources (Batch Systems, Cloud Providers).

DIRAC implements **interfaces to manage jobs via CEs** (submit, monitor jobs, get their status). They are named *<CEName>CE*.

Grid Computing

Distributed computing infrastructure for advanced science and engineering:

- "Coordinated resource sharing and problem-solving in dynamic, multi-institutional virtual organizations"
- Virtual Organizations (VO) are collaborating.
- Ideal to work with HTC workloads.



Grid Computing: HTCondor

Manage a cluster of dedicated compute nodes, harness wasted CPU power from idle desktop workstations. v10 is coming in May 2022, v9 support will end on February 1st 2023.

HTCondorCE is implemented using the **condor** CLI, supports local and remote schedd. Since last year:

- Now supports MP slots v7r2
- When coupled with remote schedd, automatically clean outputs v7r2

From v10, **GSI support will end**: token-based authn will replace X509-based authn (see here). **DIRAC is not ready yet**.

Grid Computing: ARC

Manage jobs on various Batch Systems. LTS version: ARC6

ARCCE is implemented using the **Python binding library**. Since last year:

- Support of AREX services (via HTTP) via the EMI-ES interface v7r3
- Can submit jobs with inputs, outputs in parameters (useful when used as a standalone) v7r3

ARC is slowly dropping support of gridftp services, they provide a REST interface to interact with AREX services. We are on it: AREXCE PR v7r3

Grid Computing: no CE

No worries: there is an *SSHCE*, which can be combined with various Batch System interfaces: SLURM, Condor, Torque, LSF, GE, OAR.

SSHCE deploys a batch system interface to remote computing infrastructures and performs remote calls to interact with a given Batch System.

Since last year, efforts have been devoted to SLURM:

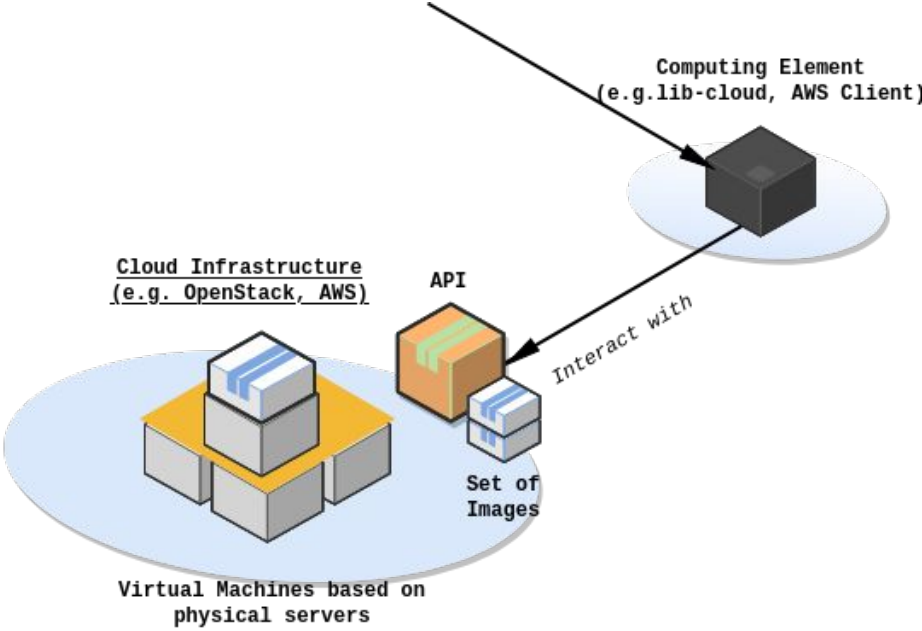
- Support Multi-Node allocations v7r2
- Minimal support for GPUs v7r3

Cloud Computing

On-demand access to a shared pool of configurable computing resources.

- **IaaS**, PaaS, SaaS
- Private, **Community (OpenStack)**, Public (AWS, Google Cloud, Microsoft Azure), Hybrid

No interoperability between clouds, not standard



Cloud Computing: VMDIRAC & CloudCE

VMDIRAC:

- VMDIRAC extension now merged into core DIRAC v7r3
- Updated for py3 compatibility.

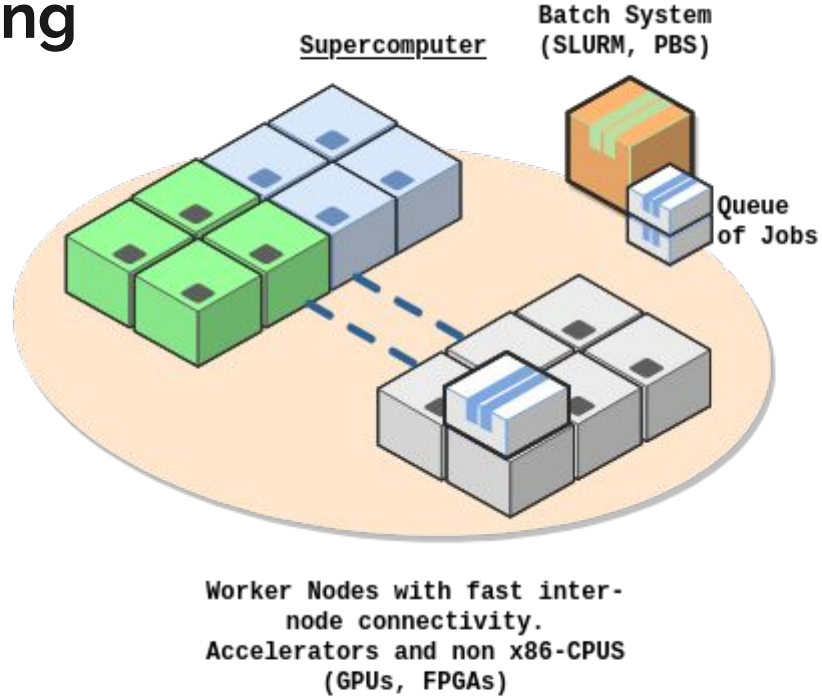
Resources/CloudComputingElement added from v7r3

- Provides direct submission to clouds (allocation = one VM instance).
- Based on apache-libcloud, a python library for generalising cloud interfaces: Provides inbuilt classes for common cloud types, we can add more classes if needed.
- Uses cloud-init for VM contextualisation.

High-Performance Computing

Cluster-based systems: computing resources networked together (many-core architectures). Characterized by fast internode connectivity. Supercomputers are the largest HPC of the world.

Much more constrained than a traditional Grid site (no external connectivity, no CVMFS, external access to a Batch System). Each HPC is unique and requires [a specific attention](#).


Supercomputer
Batch System (SLURM, PBS)
Queue of Jobs
Worker Nodes with fast inter-node connectivity.
Accelerators and non x86-CPUS (GPUs, FPGAs)

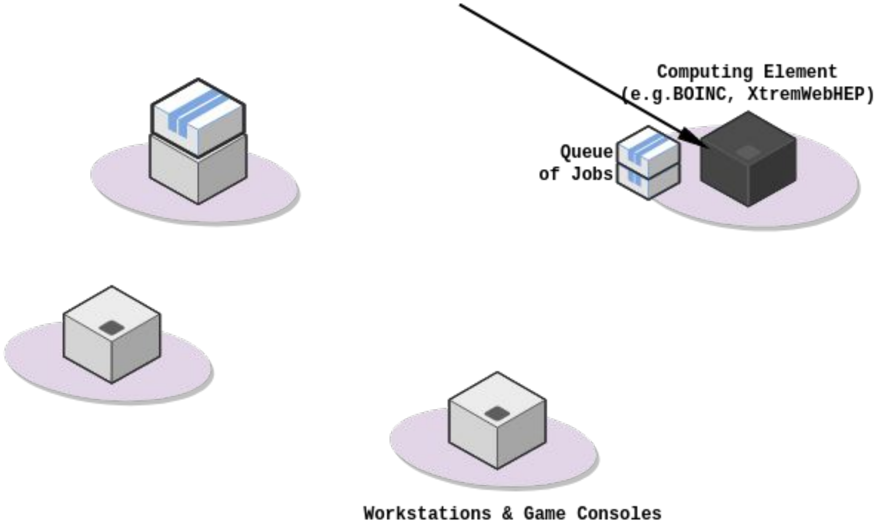
Volunteering Computing

Large computing power distributed in hundreds of thousands of personal computers belonging to the general public willing to share their resources.

Trustless environments, mechanisms to check the fidelity of the results.

Works well with preemptible HTC workloads.

Decentralized cloud concepts inherit from Volunteering Computing.



Volunteering Computing: BOINC

BOINCCE allows to interact with BOINC resources via SOAP.

No progress since 2013.

“Inner” CEs: basic components

Interfaces to manage jobs within an allocation. “Inner” CEs are independent from the computing paradigm.

- *InProcessCE*: execution of the job in the same process as the caller
- *SudoCE*: execution in a spawned process with a different user ID (used on VMs to isolate caller environment from the user job)
- *SingularityCE*: execution inside a Singularity container (isolation of the environment, possibility to update the environment for user job execution) e.g. reinstall DIRAC client with different options.

Configuration (in CE section of the CS): `LocalCEType = <InnerCE>`

“Inner” CEs: PoolCE

Run several jobs simultaneously in separate processes, managed by a ProcessPool.

The *PoolCE* transfers jobs to a basic “inner” CE (InProcess, Sudo, Singularity).

Mostly used to partition many-core nodes with jobs having different CPU requirements:

- Getting a whole node (48 cores)
- Getting available cores (4-16 cores)
- Getting a fixed number of cores (1 core; 5 cores)

Configuration (in CE section of the CS): `LocalCEType = Pool/<InnerCE>`

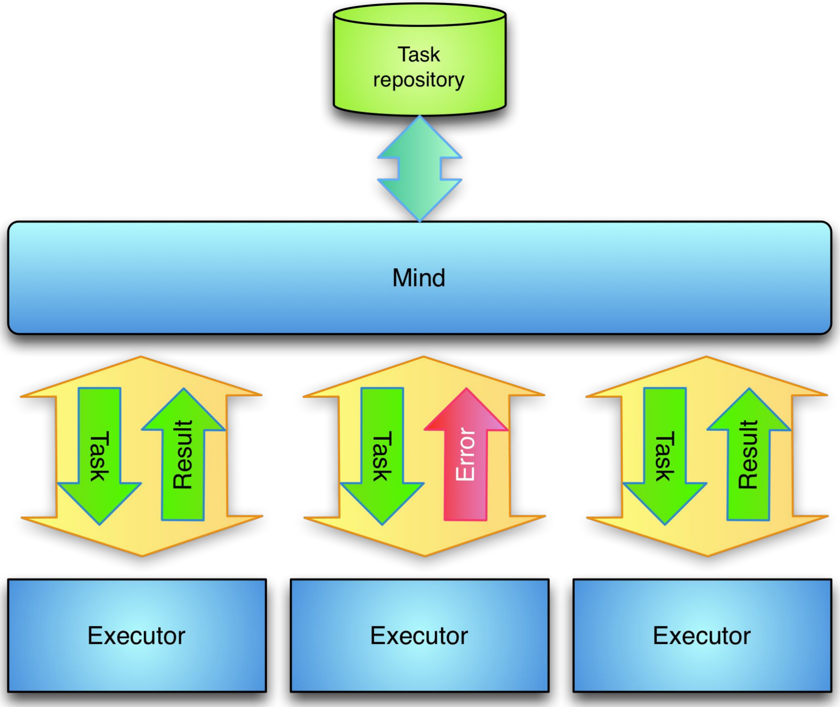
Supplying Computing Resources with DIRAC Jobs

Pre-Processing Steps: link

Currently rely on the Executor architecture:

- Tasks need to be executed before processing a job (checking InputData, assigning a job to a task queue…)
- Executors get tasks once they are available and process them

v8.0 The architecture is going to be replaced by Celery + Message Queue.



Resources and Jobs Tag

Tags were introduced to declare special capabilities of computing resources

- e.g. `Tags = GPU` to declare GPU applications support

Tags can be requested in job's JDLs in order to limit them only to sites with special capabilities. Site queues can also define jobs with which tags they accept

Tags are not statically predefined and can be used for flexible tuning:

- e.g. site queues offering resources for the biomed VO but only for COVID19 related jobs define:

  ```
  RequiredTag = COVID19

  VO = biomed
  ```

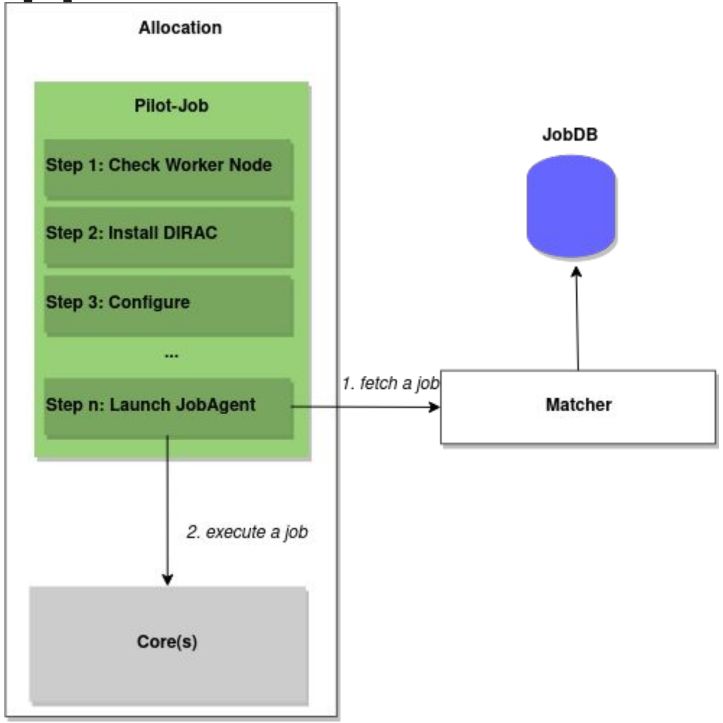
Getting an allocation: pull-based approach

Pilot-Job: resource reservation container (code here).

- Once installed on a computing resource, executes configurable steps.

JobAgent: submits environment information to the Matcher service, and gets an appropriate job.

- Job is then submitted to an "Inner" CE



Getting an allocation: pull-based approach

Pilot-Job implementation has its repository, independent from DIRAC (Pilot3). Since last year:

- Various minor features (support for GPUs…)
- **Pilot logging** is coming: pushing logs to LocalFile, REST or MQ v8.0
    - DIRAC is then able to get pilot logs and store them in a SE.
    - No need to get Pilot outputs from the CEs, they are automatically transferred by the Pilot itself.
    - *CloudCE* is expecting this feature (no unified way to get outputs from VM via lib-cloud)

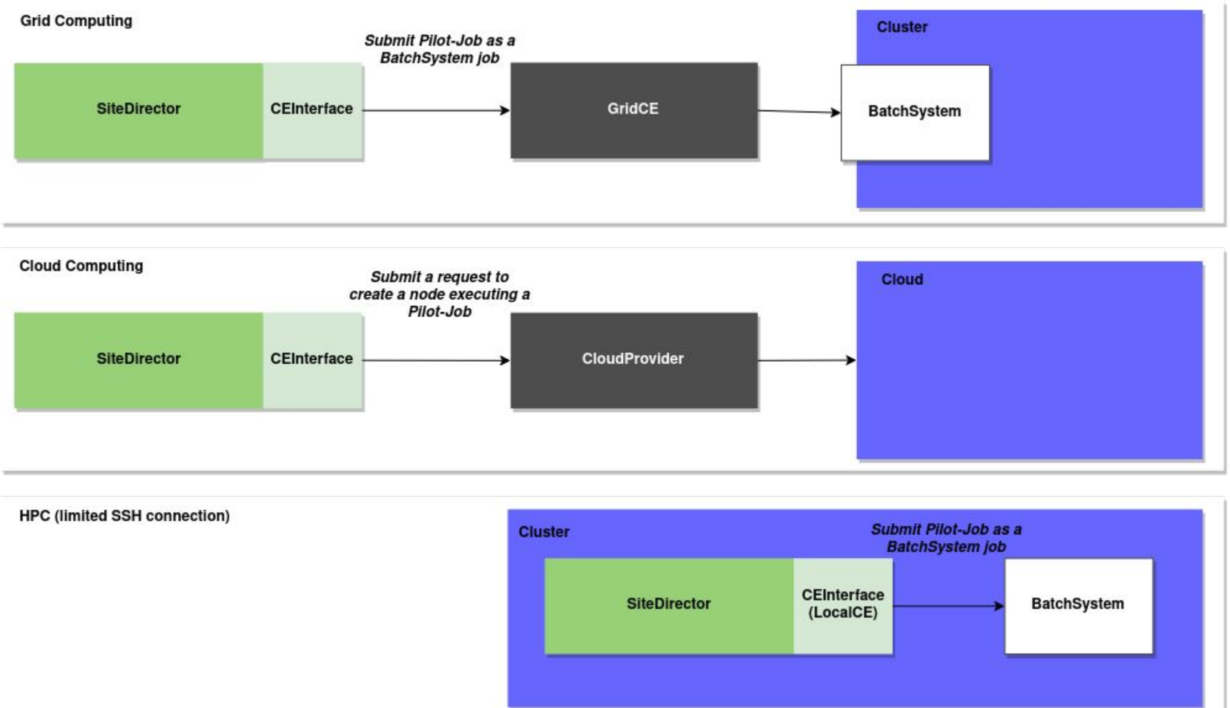
Getting an allocation: pull-based approach

SiteDirector: Manage Pilot-Job instantiations.

- Checks waiting jobs, generates Pilot-Jobs if necessary.
- Pushes them to CEs.
- Monitors them.
- Reports data to the Accounting system.

CloudDirector deprecated: SiteDirector + CloudCE as replacement. **v8.0**
CloudDirector will be removed once communities have tested CloudCE.

Getting an allocation: pull-based approach



Getting an allocation: push-based approach

A pull-based approach does work if the remote computing resources have an outbound connectivity. This is not always the case, especially for HPC sites.

PushJobAgent: a mix between a SiteDirector and a JobAgent. Run outside a Site, fetch jobs, tag them and submits them to an inner PoolCE. v7r3

- Workflows run outside the Site, only the Script/Application/Executable is sent to the Site

The solution remains simple but **consumes a lot of memory**: cannot scale.

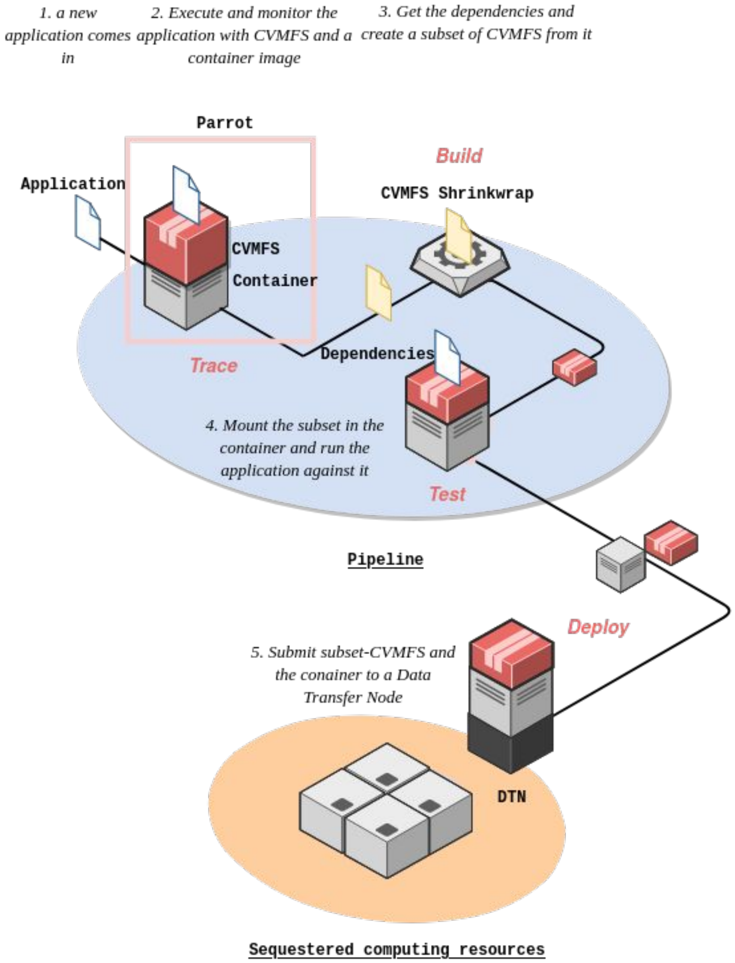
Will come up with a revised version/architecture in a future DIRAC version.

Getting an allocation: push-based approach

CVMFS cannot work if no outbound connectivity is available.

SubCVMFS, utility to:

- trace applications of interest
- build a subset of CVMFS
- test it with applications of interest
- deploy it to a computing infrastructure


1. a new application comes in
2. Execute and monitor the application with CVMFS and a container image
3. Get the dependencies and create a subset of CVMFS from it
Parrot
Application
CVMFS
Container
Trace
Build
CVMFS Shrinkwrap
Dependencies
4. Mount the subset in the container and run the application against it
Test
Pipeline
Deploy
5. Submit subset-CVMFS and the conainer to a Data Transfer Node
DTN
Sequestered computing resources

Efficient Executions

DIRAC Benchmark: definition

A given CPU-intensive task can perform differently according to the underlying CPU used:

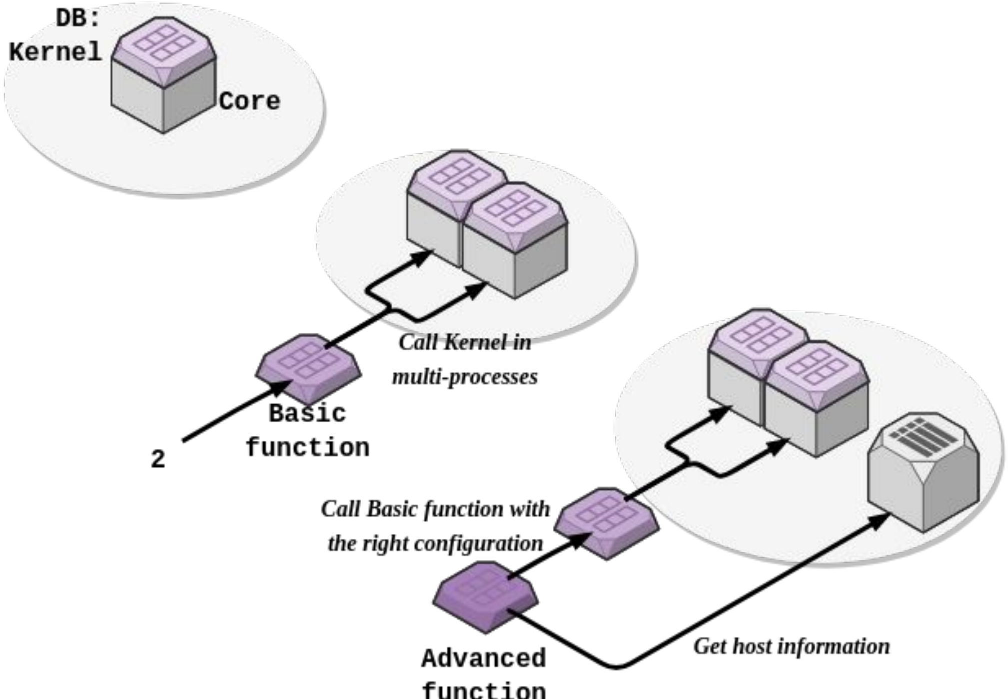
- SiteA: 5 secs, SiteB: 15 secs

The DIRAC Benchmark (DB12) is a fast CPU benchmarking solution aiming to compute the power of a CPU.

DB12 replicates a Monte-Carlo simulation execution (mainly used in the HEP context).

Provide ~accurate information in less than a minute.

DIRAC Benchmark: structure



DIRAC Benchmark: progress v1.0.4

Since last year:

- **DB12 was ported to Python3.9**: noticed discrepancies with the Python2 versions (see further details here)
- Introduced correction factors to provide scores close to the Python2 implementation
- DB12 was uploaded to Pipy and Conda-forge: added a CI and tests
- DB12 (the repository) is now properly used within DIRAC (not copy pasted)

Conclusion

- **Keep integrating new kind of computing resources** to DIRAC (especially HPC ones)
- **Keep following recent computing resources developments** (tokens, end of GSI support)
- **Keep maintaining the DIRAC WMS** to efficiently exploit resources (Pilot-Logging, DB12, Executors)

- While providing **a uniform and simple interface to users**