Deliverable D4.1

Project Title:	World-wide E-infrastructure for structural biology	
Project Acronym:	West-Life	
Grant agreement no.:	**675858**	
Deliverable title:	Consolidated architecture of job submission and interaction with infrastructure	
WP No.	4	
Lead Beneficiary:	Masarykova univerzita	
WP Title	Operation and maintenance of the computing and data infrastructure	
Contractual delivery date:	Month 9	
Actual delivery date:	Month 9	
WP leader:	Aleš Křenek	Partner: MU
Contributing partners:	Luna, MU, STFC, UU, CRIMMP, INFN, CSIC	

Deliverable written by Aleš Křenek

Contents

1 Executive summary

Current West-Life portals use different architectures for backend management (grid, cloud, job dispatch and submission…). In order to increase sustainability and interoperability of those portals, one of West-Life core objectives is to standardize the backend.

The expected benefits are:

1. Greater maintainability (same technology used everywhere)

2. Greater interoperability

3. Greater interactions with EGI infrastructure, EUDAT services

One of the difficulties is that West-Life portals have different requirements in terms of file sizes, compute time etc. Neither grid nor cloud computing alone meets the needs of this Many-Task Computing. This document identifies common features in the portal architectures, and it defines principal functional requirements on the components of the common architecture.

Additionally, various candidate technologies are available, and each of them comes with different underlying trade-offs. The main objective of this deliverable is to expose those trade-offs and make the right choice, given the constraints extracted from West-Life portals requirements. The chosen solutions will be deployed on the project testbed, and existing portals will be migrated gradually.

2 Project objectives

With this deliverable, the project has reached or the deliverable has contributed to the following objectives:

No.	Objective	Yes	No
1	**Provide analysis solutions for the different Structural Biology approaches**		**X**
2	**Provide automated pipelines to handle multi-technique datasets in an integrative manner**		**X**
3	**Provide integrated data management for single and multi-technique projects, based on existing e-infrastructure**	**X**	
4	**Foster best practices, collaboration and training of end users**		**X**

3 Current Status and Goals

This document is a report on activities done towards achieving the work package objective "*O4.2: Define, implement, and deploy consolidated architecture for job submission and data access*." Specifically, the designed architecture aims at the following goals:

1. There is a uniform "West-Life" recipe to deploy portal on the cloud + grid infrastructure.
2. Multiple instances of the same portal may coexist (for better load balancing, isolation of user groups, ...), and their deployment is straightforward.
3. The approach is homogeneous for different portals.
4. Maintenance, including migration due to e.g. hardware of software upgrade becomes easier.

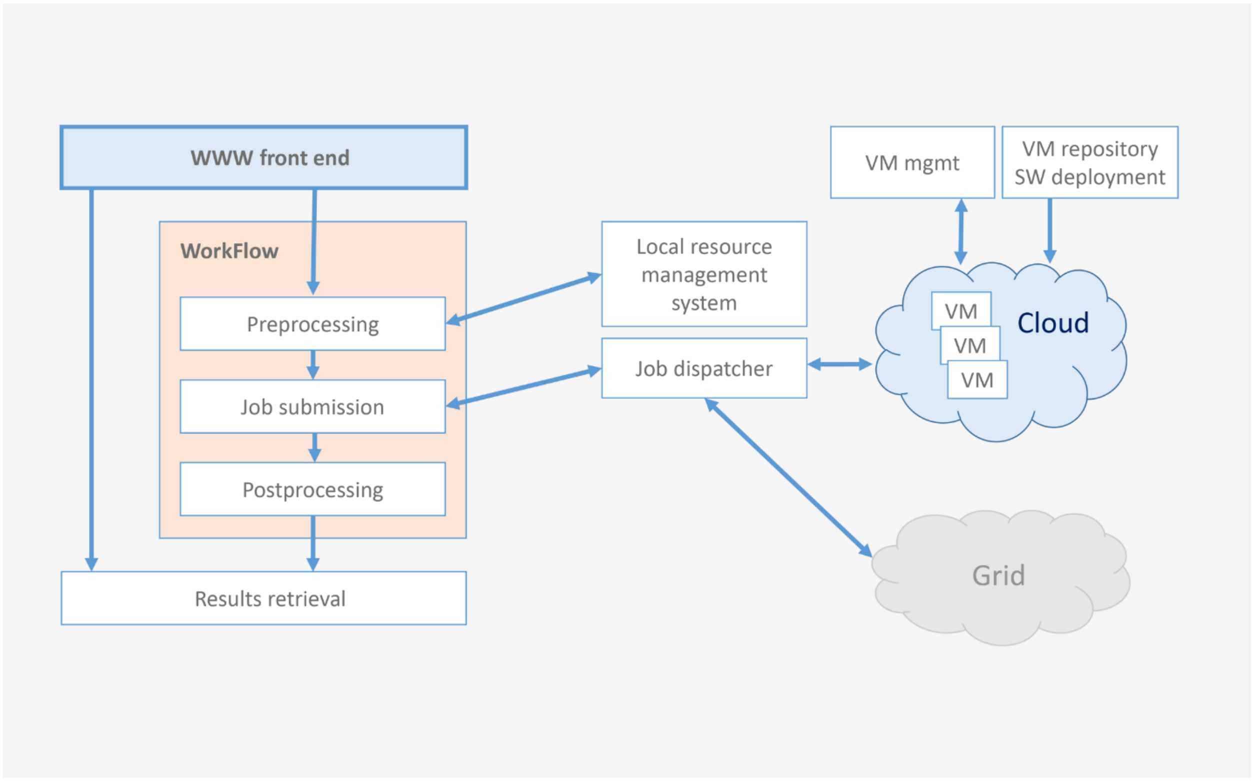
A survey was carried out with West-Life portals to gather requirements and infer constraints. Based on portal operators' answers [1], we could draw a common architecture target defined in detail in Sect. 4. Several high-level conclusions were drawn from the surveys:

1. All portal operators can be mapped to the target architecture without too much effort
2. Local resources and grid/cloud do not tackle the same job scales and should be treated separately
3. It is probably "overkill" to try to standardize local resources management since they are heavily dependent on each portal workflow. This deliverable should thus focus on grid/cloud dispatcher.
4. The target architecture must enable cloud bursts without abandoning the grid legacy that has worked well so far.

The survey is complemented with a set of use cases, described in detail, with discussion of the applicability of the common architecture in Appendix A. Available technical solutions were reviewed thoroughly as recorded in Appendix B.

4 Target Architecture

Despite the existing portal solutions being run by the project partners use rather different technology, there is a common design pattern captured in the diagram bellow:



All the inputs required for the application run are gathered from the user via a WWW front end interface. The request is processed as a workflow of varying complexity (it may consist of quite many distinct steps). Typically, there is a preprocessing stage, which may require moderate computing resources but it does not take long time, therefore it is not feasible to send the computation to remote resources; local computer clusters are used instead. The main payload of the request, once its inputs are ready from the preprocessing stage, is delegated to the remote computing resources. The standard way uses grid interfaces (most of resources are available in this way nowadays), however, there have been successful pilots to use cloud IaaS as well.

The work of West-Life aims at general consolidation of the heterogeneous implementations of the multiple application portals. This is going to be done at two levels:

- *Portal virtualization*: each portal will be described as a set of mutually interacting nodes (e.g. the web front end, auxiliary database server, local batch system server and several local worker nodes) that can be, as a whole, deployed in the cloud environment in multiple instances (for different purposes, e.g. isolate user groups, run multiple versions simultaneously, balance load etc.).
- *Common access to resources*: the project will choose technical means and it will define recipes to use remote computing resources (both grid and cloud) in a uniform way.

The existing portals will be migrated gradually to this target architecture at both levels.

4.1 Virtual Cluster Management

The first requirement to achieve the portal virtualization is a standardized description of the virtual portal, i.e. listing the nodes (virtual machines) forming the portal, defining their mutual dependencies, means of deployment (e.g. VM images, contextualization recipes etc.). The emerging industry standard in this field is TOSCA [2]. (which is used by all the candidate implementations described in Sect. 6).

Further, the implementation of this component (commonly referred to as *cloud orchestrator*) is expected to parse the virtual cluster description, to deploy it on the configured IaaS cloud infrastructure, and to manage its life-cycle. In particular, failures of the nodes should be handled transparently in most cases, and the framework should provide elasticity (e.g. when a batch system reports a long queue, new worker nodes should be spawned). The operation must be complemented with appropriate monitoring, which generates alerts to the portal maintainers if a human action is required (e.g. a failure which is unsafe to recover automatically).

4.2 Templating System and Software Deployment

This section is about the technology West-Life shall use to manage computing nodes configuration, and dynamically configure them at run time for the case which are not relying on the classical grid resources provided by EGI.

We want to pool partners' resources so this means each node needs to be used for several different jobs. We thus need a configuration templating system (like Puppet, Ansible, or an alternative), each node must be reconfigured dynamically each time a job starts, to ease maintenance and configurations management. We can use VM Images (ex: Gromacs VM Image), or Docker Images (that would run in a generic Docker VM image, or bare metal grid resources, in some cases even if Docker engine is not installed on the node). Docker containers are more lightweight and easier to use. Moreover, it doesn't take much to set up Docker on a laptop and prototype with containers, so developers can easily get started. Making VM images is much more complicated.

Many cloud-enabled job dispatcher solutions exist, and even more if we include Docker technology in our analysis. In order to find the right technology for West-Life portals, we performed analysis on several dimensions which are described below:

A) Compatibility with existing grid systems
The path of least resistance would be to use classic bare-metal node configuration systems such as Puppet, Chef or Ansible. Some of those systems are able to scale to thousands of nodes, but with quickly increasing complexity. Moreover, some of those tools are difficult to learn.
VM Images are a good solution too, but require a cloud framework, or at the very least hypervisor, this would require modifying the existing grids, which are bare-metal. However, VM images are simpler to understand: you configure a VM and make a frozen image.
Docker containers are a kind of in-between: you can use containers instead of VM, and a container is basically a “frozen” configuration, just like a VM image. BUT containers frameworks are easy to install on a bare metal grid, much easier than cloud/virtualization frameworks.

B) Compatibility with cloud bursts
For cloud bursts, one of the key requirements is to be able to spin up new nodes quickly. From that point of view, VM images and Docker containers are a good fit since they can start up quickly. On the opposite side, if the required software for a compute job is long to install, bare-metal tools will be long to run since they will run the install each time, unless CVMFS is used.

C) No lock-in with non-desirable dispatchers
Bare-metal tools and VM images come with few string attached from the dispatcher point of view. For Docker containers however, you have to use existing Docker framework dispatchers (Mesos, Kubernetes etc).

4.3 Job dispatcher

The principal role of job dispatcher is taking the heavy payload of the job from the workflow logic of a specific portal application, finding suitable remote computing resource to execute the payload, to submit the job to those resource (using the portal robot credentials typically), and to “babysit” the job execution, i.e. to react to monitoring events like job failure or timeout (resubmit eventually) or successful completion.

The ideal job dispatcher should be able to submit job to both grid and cloud infrastructure. The grid infrastructure has to be supported because much more powerful computing resources are still available in grids than in EGI Federated Cloud infrastructure [3]. On the other hand, cloud offers flexibility, which might be required in several of our use cases.

Another important criteria is coupling with templating system: we need a dispatcher which works with a templating system we accept to work with.

4.4 Remote VM Pool Management

Virtual machines enable easy setup of software with complex requirements on any site, making them attractive to run the portal payload in a flexible way. However, unlike the grid case, when the pool of resources is more or less static, submission to cloud requires a service which maintains pool(s) of VMs running, configured with particular software, and ready to accept jobs. The component responsible is commonly referred to as *infrastructure manager* (IM).
In collaboration with DIRAC developers, in particular Victor Mendez, a VM mounting the software partition of the enmr.eu VO via CVMFS has been successfully tested. It is automatically launched by the DIRAC machinery on EGI Federated Cloud sites supporting the enmr.eu VO, hence using DIRAC to take the IM role. So far molecular dynamics simulations with Gromacs have been successfully submitted via DIRAC to those cloud VMs. The jobs were run in VMs with up to 8 processors per node. The DIRAC brokering system has been configured to send jobs to these EGI Federated Cloud VMs if the submitted job (JDL) contains a tag for multiprocessor, e.g. `Tags = {"4Processors"}`. Jobs not containing this tag (e.g. the ones submitted to DIRAC by the HADDOCK portals) will be directed to standard grid resources.

4.5 Data Storage and Transfer

Data storage provided by scientific as well as commercial vendors can be integrated in different levels.

- File system integration: Online, files transparently transferred via e.g. WEBDAV protocol.
- File system integration: Online/Offline, heavy caching mechanism e.g. CernVM-FS.
- File system integration: Offline, synchronization on background when online, files accessible when offline via OwnCloud client or Dropbox client or similar.
- Programming API level integration: Amazon S3, Google Drive, Dropbox, EUDAT allows API (REST based on HTTP) to access the files and it's metadata. This introduces integration (programming) effort to implement with the existing tools.

Current WP6 prototype [4] shows File System online integration via WEBDAV protocol and offline using Owncloud client to be manageable. The offline solution needs additional space in the local instance of WP6 dedicated to user, thus an intelligent policy is needed on which data to integrate with online and which also in offline. The strategy might be introduced to user and some default clever can be achieved by a dedicated directory. The aggregated view to the mounted storages are additionally available via an instance’s WEBDAV endpoint which can be mounted into user’s file system. WP5 prototype integrates some storage providers on API level giving direct access to the data and metadata, presenting them in a UI.

5 Hardware Provisioning

Some project partners contribute hardware resources to West-Life project. What we would like is to have one unique pool of resources for the project, a "federation" between project members. This pool will be organized as an EGI VO (virtual organization). Resources will be merged in “enmr.eu” existing VO if not already integrated.

In terms of architecture, the web portals will submit jobs to a static grid registered with EGI, and the grid would scale elastically with cloud provisioning. --> static grids with elastic scaling

To sum up: we choose a static grid with elastic scaling as the target architecture, and for grid management we use EGI services.

6 Proposed Implementation version 1

We follow the common architecture and requirements on its components defined in Sect. 4. Candidate implementations (existing software packages) are described in Appendix B. In this section we describe decisions, supported by appropriate reasoning, taken for the first generation of the project testbed.

6.1 Virtual Cluster Management

The technology choice was restricted to two options -- Cloudify (Sect. B.5.5) and Infrastructure Manager (Sect. B.5.1), which will be used and evaluated simultaneously.
Cloudify is a relatively simple and compact tool, reasonably stable, with good level of maintenance and clear mid-term roadmap. It implements the TOSCA standard in the cloud definition, therefore, even if we chose other tools later, the migration would not be too difficult.
In the first stage we use the simpler of two Cloudify modes, the stateless command-line tool “cfy”, suitable for quick prototyping. Later we will migrate to the Cloudify Manager, a service for complex deployment with monitoring of the deployed virtual clusters.
We implemented a prototype of OCCI plugin to Cloudify [5], which enables use of EGI Federated Cloud sites (lack of OCCI support was a showstopper, on the contrary to other welcome features of Cloudify). This prototype will be developed further to meet the needs of the project.
Simultaneously, in collaboration with Indigo project, the early adoption of its solutions (based on Infrastructure Manager) will continue. Outcomes will be evaluated in D4.3, when a decision on the software to be used in the 2nd generation of testbed will be also taken.

6.2 Templating system

This part addresses the cloud vs Docker decision. Basically, cloud solutions (with hardware virtualization) and Docker solutions (container-based virtualization) can be seen as 2 different templating system = 2 different ways to run predefined image (VM images vs containers).

From the point of view of compatibility with bare-metal grids and cloud burst, Docker would be the best option (see section 4.2) as it plays nice with bare-metal server (required grid updates would be minimal), and enables fast start-up for cloud bursts.

However, Docker can be problematic in 3 cases:

1. when compute jobs run on multiple nodes (Docker and MPI might not play well together)

2. when compute jobs require shared drive mount (Docker does not like that)
3. on older operating systems still existing on the grid infrastructure

There is ongoing work to make Docker play well with MPI and shared storage, and the Indigo projects builds its solution on those technologies. However, relying completely on the ongoing development would present unacceptable risk to West-Life, which is focused on production services. Therefore, we stick with conservative solutions for the time being.

Several West-Life portals rely on MPI and shared storage, so Docker technology and the dispatchers that come with it cannot be used for our project.

Note 1: Docker and cloud frameworks (Openstack etc) ecosystems do not draw boundaries the same way. Cloud frameworks typically take care of resources provisioning, network topology and storage abstraction, and leave job dispatch to external solution. Docker technology on the contrary tends to merge both. For example, in Docker ecosystem, for resources management, you can use Mesos and Kubernetes. Both frameworks provision resources AND manage job dispatching. But those job dispatchers are thus embedded with Docker technology.

So, even if West-Life portals did not use MPI nor shared storage, it would have been hard to consider Docker ecosystem dispatchers, without throwing to the bin all the EGI grid legacy, which we want to keep.

All this means that we will choose VM images as the primary templating system, which require cloud frameworks. Since the required grid updates (install locally everywhere a cloud/virtualization framework such as OpenStack) would be significant, it is possible that each portal keeps a bare-metal setup, and VM images will be considered only for cloud bursts. Then we will have to find a way to make sure VM images are in sync with bare-metal nodes. This part is to be discussed later once we have more experience in the convergence implementation.

6.3 Job Dispatcher

Drawing from previous considerations, it’s clear we have to use a non-Docker ecosystem dispatcher for the time being. This decision might be revised later in the project according to new developments done elsewhere, INDIGO project in particular (cf. next section). Since we want to keep using existing EGI grids, we have to find a grid dispatcher with elastic cloud scaling capabilities (capable of spinning up VM and using VM images in conjunction with bare-metal nodes).

Several options have been considered: DIRAC, EC3, Condor…

Dirac and HTCondor represent traditional batch queuing system with long history of development and usage primarily in academic community. Both has its roots in times of grid infrastructures, but both systems are under continuous development and new features have been added or can be expected to be developed in the future.

Mesos together with the frameworks like Aurora, Marathon and Chronos represent quite a new approach when comparing to Dirac and HTCondor. Mesos seems like a promising technology with great potential in the future. The project is supported by large community of users and developers a deployed in many instances both in academic and commercial sector.

Unfortunately, Mesos do not offer any support for submitting jobs to grid infrastructures. When considering the use in context of West-Life project an interface to any job dispatching systems widely used in grid (and cloud) infrastructure or substitution for such functionality would be required. Creating such layer and API is out of scope of West-Life project.

For role of West-Life job dispatcher should be chosen one of two candidates: Dirac and HTCondor which both offer similar functionality. The DIRAC framework has been already used by one of the West-Life portal with good results and it seems like a good candidate for the first deployment of consolidated architecture.

We decided to settle on DIRAC since it seems to be the solution with the biggest community, which means support and training material in the foreseeable future.

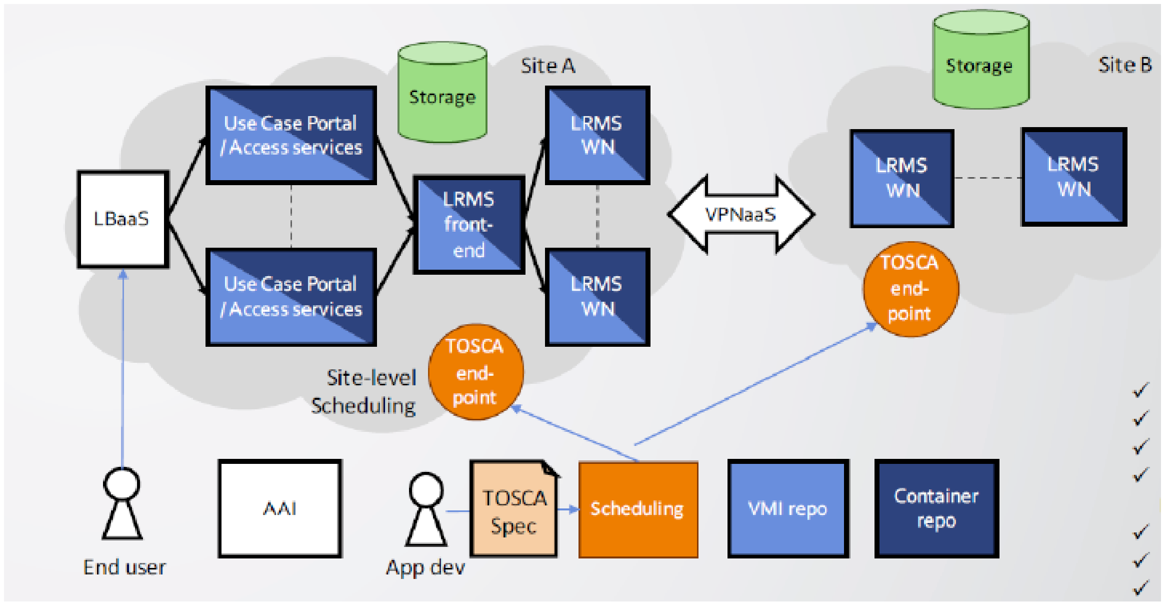
6.4 INDIGO solutions

Section B.6 described a list of key components that will be available in the first release of INDIGO. Some of their first concrete applications in real use cases have been however already piloted. In particular, UU and CIRMMP partners of West-Life, who are also partners of INDIGO project, already experimented a couple of prototypes where a web portal exploits a batch system or a Docker container executed within a VM provisioned from a IaaS cloud to run applications. The prototypes were tested with the WeNMR/West-Life applications HADDOCK, DisVis, PowerFit, and AMPS-NMR [6] [7].

6.4.1 Haddock:

The case study for HADDOCK Portal involves the virtualization of the HADDOCK web portal and the required computational infrastructure underneath it using INDIGO solutions. The aim is to be less dependent on local hardware and to facilitate the deployment of the software at other sites (possibly within a company, or usage for teaching purposes).

A possible solution can be based on the architecture proposed in INDIGO, as seen in the figure below. With the help of INDIGO solutions, HADDOCK Portal can be started as a VM with a web portal front end, having a batch system installed and preconfigured with the number of cores per node, as number of queue slots per node to run HADDOCK jobs. A typical HADDOCK run on 100 CPU cores might take between 1 and 10 hours, so a virtual cluster with different configuration may need to be up and run for several days or weeks to complete the job. Both ssh and web access should be supported. There should be enough storage space for temporary files on the compute nodes and enough storage (e.g. 500 GB) on the master node to store the end results obtained from HADDOCK. Currently, the results are stored on the server for a maximum of two weeks after which they are automatically deleted. So, it is the end user's responsibility to download the data within this period. Models obtained with HADDOCK may be deposited in open repositories.

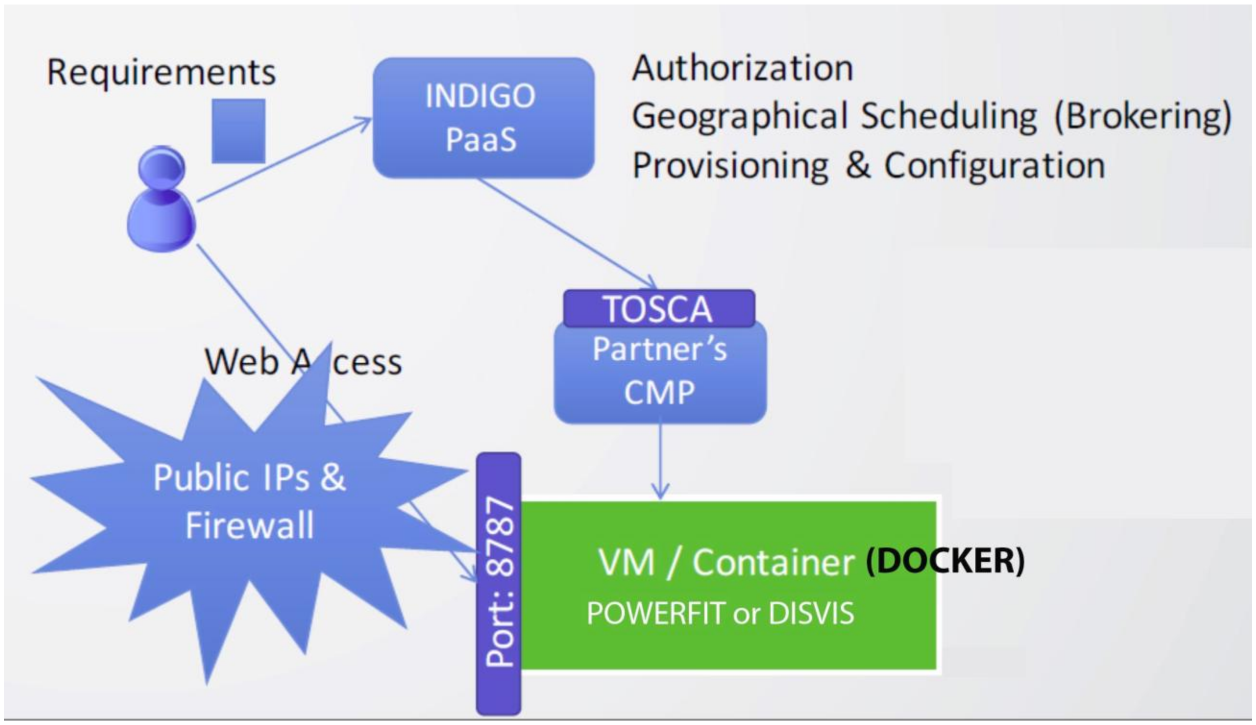


The developers and operators of HADDOCK have the root level access to the portal and back-end to perform operation maintenance and configuration. They need also provide access for HADDOCK Portal VM only to users with specific credentials (e.g. WeNMR Virtual Research Community credentials, West-Life credentials, or HADDOCK portal credentials).

The end user is expected to interact with web portal front end and submits the jobs using his/her credentials. The input files for the jobs are mainly text files. The typical way of working for the end user is via laptop/desktop. However access from mobile phones/tablets can be a plus (but simple since it only requires a web browser). The computational process in the background is not of the concern of the end user. He/she is only interested to obtain the results in a reasonable amount of time, and download them via web interface when the job is complete. The user is also able to monitor the status of the running jobs. When the job is complete, the user is notifying by email (the VM should thus allow emailing) and can access his results via a web page presenting various statistics and graphical analysis. Optionally, models can be visualized online via plugins such as jmol.

6.4.2 DisVis and PowerFit:

DisVis case study involves the deployment of DisVis into a VM with all its dependencies (e.g. FFTW3, pyFFT, opencl libraries…). The software is able to run using single/multiple CPUs or GPGPU. The scaling performance for an increasing number of CPUs should be similar, as when running directly on hardware. Docker containers may be a possible solution for this case. The same consideration apply for PowerFit case study.
A similar INDIGO solution as their R-Studio server example can be applied for both DisVis and PowerFit, as seen in figure below. There should be enough space for storage of temporary and end results. The end user can start a VM that provides a web-interface to run DisVis (or PowerFit) and presents the results, which still has to be developed. Simple ssh access should also be offered. The end user can upload input files and download the output. A typical way of working for the end user is via laptop/desktop. However, access from mobile phones/tablets can be a plus.

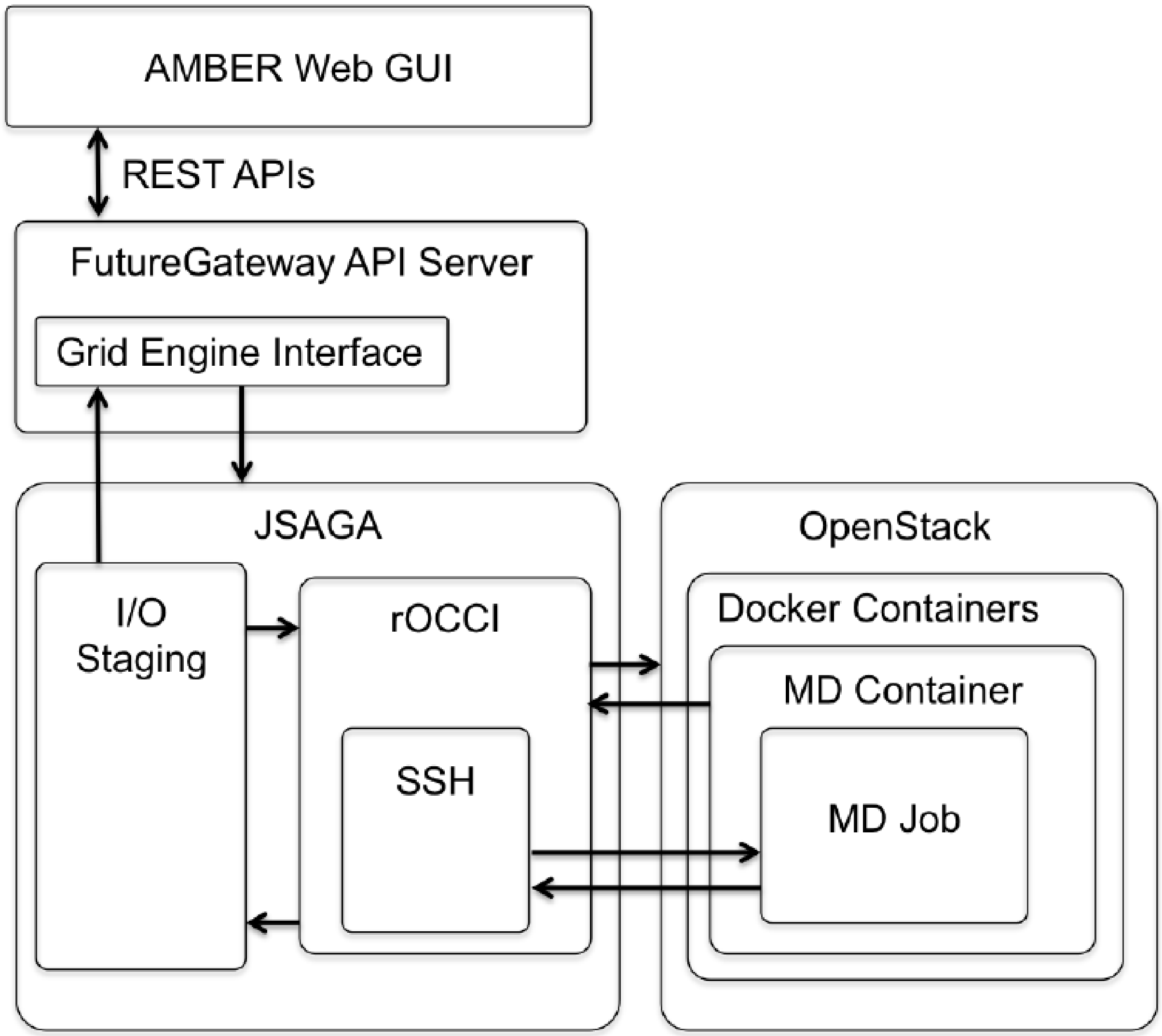


6.4.3 AMPS-NMR:

The AMPS-NMR portal has been selected to exploit the capabilities of the INDIGO FutureGateway component. Running e.g. a restrained MD calculation in AMBER through AMPS-NMR is a process logically split in four macro steps:

1. input data preparation
2. set MD parameters, by selecting a redefined protocol
3. send the calculation to the computational infrastructure
4. retrieve the output from the computational Infrastructure and store it in the local virtual folder of the user.

To enable the usage of INDIGO FutureGateway by the AMPS-NMR portal, it was enough to adjust the actions 3 and 4, leaving the user interface completely unchanged. So, enabling the job submission via FutureGateway did not affect the user experience at all, with respect the current production portal. The only modification involved adding a radio button in the portal to allow the user to select “FutureGateway” as computational infrastructure. This use case made use of an existing Docker container which includes the necessary software packages to run different MD simulations via AMBER. The MD container is served by an OpenStack IaaS capable to run Docker instances. The IaaS supports OCCI standard to manage containers in the same way as done in the EGI Federated Cloud. In this scenario the FutureGateway API Server can instantiate an MD Docker container and then execute on it a job, retrieving at the end the output via the rOCCI JSAGA adaptor, as described in the architectural scheme below.



In principle, the same functionality can be achieved by using the existing grid infrastructure using exactly the same FutureGateway API calls. In this case the figure above will be the same, except the for the rOCCI box replaced e.g. by the EMI-WMS, in charge to send the MD job to the grid. After successful submission, the user can monitor his job within the AMPS-NMR portal. For completed jobs, the output is automatically retrieved and copied to the user folder, which is maintained in the local file system. The output can then be downloaded by the user on his own storage. At the time of writing the rOCCI adaptor was used to connect with the computational backed. In the first official INDIGO release a TOSCA adaptor will be available to connect with the PaaS Orchestrator component. However, with the above architecture the end application will not require any code changes. This prototype demonstrated that the implementation of the submission mechanism using the INDIGO FutureGateway API server did not involve any changes in the way the user interacts with the current portal, nor in the way the portal validates and processes the input data and the job requests submitted by the user. In particular, the management of the VM using the appropriate contained with the AMBER software was entirely taken care by the FutureGateway component and did not require any specific change on the portal side. Therefore, the prototype can represent a straightforward approach to update or extend the computational backend of existing WeNMR/West-Life portals, and also support the implementation of new portals in the West-Life perspective.

7 Conclusion

The West-Life project started with many legacy portal solutions, while some of them require considerable upgrade (e.g. migration to new versions, support for GPU), and new ones are taken on board. In this document we identified a common pattern in the portal architecture, including functional requirements on its components. In parallel Deliverable 4.2 is being submitted, which addresses broad security aspects of the whole architecture. Therefore, security is omitted in this document deliberately.

Simultaneously, available technology – existing candidate implementations for the architecture components were reviewed, and design decisions were taken for the first generation of the testbed. Two frameworks (Cloudify and Infrastructure Manager) will be evaluated for portal virtualization, DIRAC will be used as the common dispatcher for both grid and cloud environment, and EUDAT solutions will be leveraged for data handling.

In the upcoming months candidate portals for migration to this new platform will be selected. The experience from this deployment will be reported in Deliverable 4.3, and appropriate adjustments to the project plan will be taken.

Appendix A: Use cases

A.1 Scipion

A.1.1 Current status

Scipion covers the whole workflow of about a dozen of steps of processing raw CryoEM data to refined electron density map. Some of the workflow steps are automatic, some require user interaction. The individual steps are implemented by various software tools, including third party software, and providing alternatives (there is no one-size fits all choice, some approaches work better for some inputs and vice versa). Scipion framework integrates those tools to work together smoothly.

In general, a regular Cryo-EM image processing workflow starts with rather huge raw data (CryoEM "movies", up to terabytes) on which fairly lightweight calculation (mostly I/O bound) is performed. In subsequent workflow steps the amount of data gets reduced while the computational complexity increases, ending with many days of CPU time.

The implementation may use multiple computing nodes to distribute the load but it relies on a shared POSIX file system. The file system is organized in projects -- self-contained folders where all input data, intermediate results, and status of the workflow processing is stored.

Computations can be spawned through traditional batch systems (Oracle Grid Engine, Torque, Slurm, ...).

Some computationally demanding steps of the workflow support parallel execution with either threading or MPI, some of the tools leverage GPU acceleration.

A.1.2 Scipion desktop

Scipion desktop is the main package, offering a desktop interface for user interaction. It is the complete solution with more than a 100 of protocols/process from 16 different packages available. Typically, Scipion is installed on fat nodes or even clusters ready to make use of as much computational power as is available. Under this scenario, Scipion is not installed in the end user machine but on servers, requiring the user to use any of the available remote desktop presentation solutions (ssh, x2go, VNC,...).

A.1.3 Scipion web tools

Web tools (SWT) represent a set of small workflows a user might want to try over a web interface, with the only requirement being a modern web browser, and therefore no need to install software at the user's machine. Currently those are:

My movie alignment

Corrects for global frame movements, as well as local within-frame movements of your movies. Data requirements: upload 1-2TB (resumable upload implemented, for other web tools too) Computational requirements: 3-5 minutes with 1 CPU.

NOTE: one of the methods uses GPU, so a GPU queue is in place in production.

My first map

Generates a first volume using particle averages to be use a a latter volume reference for refinement. Data requirements: upload 0.5 MB. Computational requirements: 5 hours with 1 CPU.

My resolution map

Computes the local resolution of 3D density map using Resmap. Data requirements: upload 55 MB Computational requirements: 1 minute with 1 CPU

So far, the web tools run on a single server. They are integrated with queue managers, so it is possible to perform the processing in a cluster. They have been successfully deployed to virtual machines in the cloud (the process is highly automated, but there is still room for polishing).

A.1.4 SWT with proposed architecture

Storage site

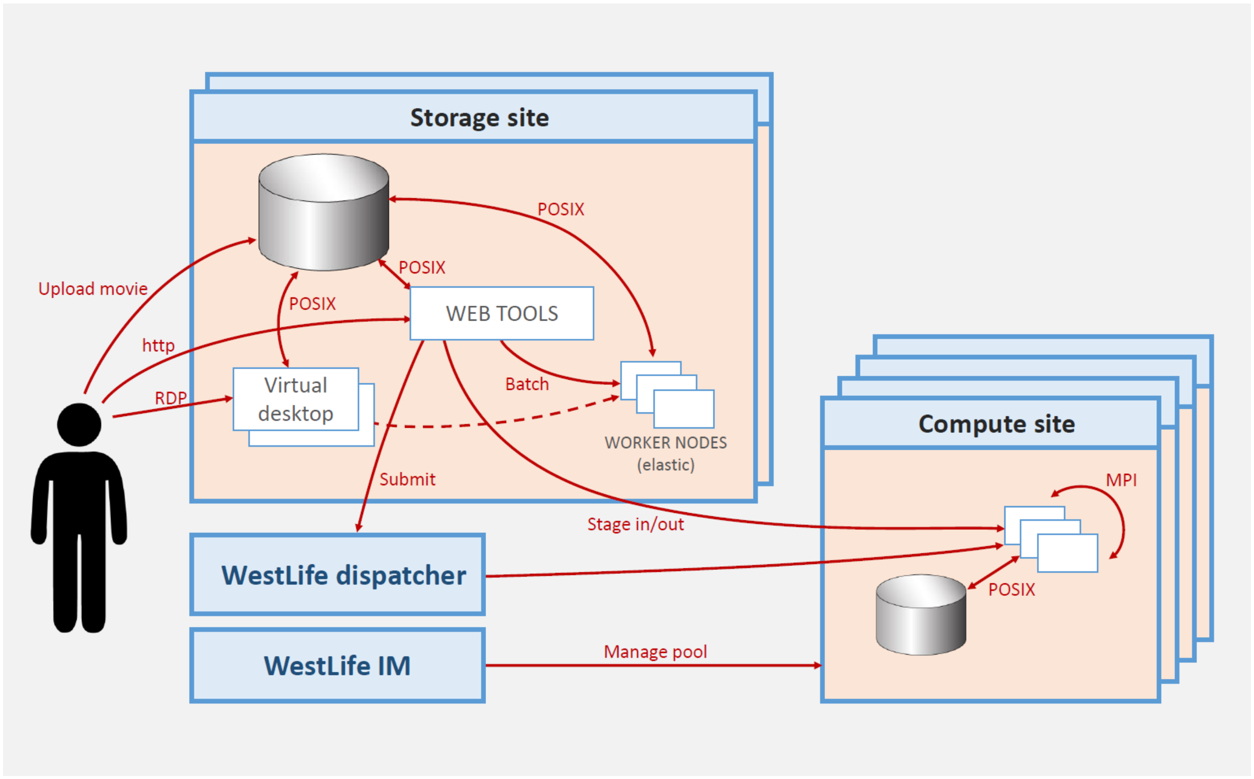
Main SWT server will run on the storage site being able to resolve http request (maybe behind a load balancer). Data from other sources like EUDAT could be mounted on the store site and accessible to the SWT as a source of input data and or destination of outcomes. There must be a shared file system (nfs), where all the SWT projects will be stored, mounted across all worker nodes within the storage site to be able to access same data from different machines that will run SWT processes. Apart from sharing the data, all the software packages responsible for running all the CryoEM workflows should be present in all machines enabling them to run any POSIX command issue from the main SWT server and distributed by a Scipion compatible queue system (Slurm, TORQUE, …). If a certain process requires a large computational node, then this process will need to be carried out in the Compute site.

Compute site

In case of high computational demand, a certain process within scipion workflow should be “forwarded” to a compute site. These compute sites, do not share the data with the storage site, therefore required data should be moved/copy across before any processing takes place. Compute sites, as is the case for the worker nodes in the storage site, need to have installed all the software needed to run any process. Once the process has finished, output of the process need to be copied back to the storage site for later use or visualization of the output.

A.1.5 Impact on Scipion development

Currently the web tools run as a standard web application run by a single OS account. There is no identification or authorization process in place. If resources access (data as well as CPU) is granted based on a SSO identity, changes in the way Scipion web tools, and Scipion itself access those resources might need to be adapted. Moving data back and forth to the computing site should be also implemented.



A.2 Grid-enabled web portals hosted by UU

A.2.1 DisVis

DisVis is a software tool developed for the visualization and quantification of the accessible interaction space of distance restrained binary biomolecular complexes (Figure 4). It is implemented as a Python package and a simple command-line program, with the ability of harnessing multiple CPUs or running on a GPU. The runtime on a single standard GPU is thereby comparable to the runtime on 16 CPUs (Zundert and Bonvin, Bioinformatics, 2015). A webserver submitting DisVis jobs to GPU resources of the grid is currently accessible at : http://milou.science.uu.nl/enmr/services/DISVIS.

A.2.2 HADDOCK

HADDOCK2.2 (High Ambiguity Driven protein-protein DOCKing) is an integrative, information-driven flexible docking approach for the modeling of biomolecular complexes. The HADDOCK portal is accessible from: http://haddock.science.uu.nl/enmr/services/HADDOCK2.2

A.2.3 GROMACS

GROMACS (www.gromacs.org) is a versatile package to perform molecular dynamics. GROMACS is able to work with many biochemical molecules like proteins, lipids and nucleic acids. The GROMACS grid-enabled web portal combines the versatility of this molecular dynamics package with the calculation power of the grid. The GROMACS portal is accessible from: http://haddock.science.uu.nl/enmr/services/GROMACS

A.2.4 CS-Rosetta

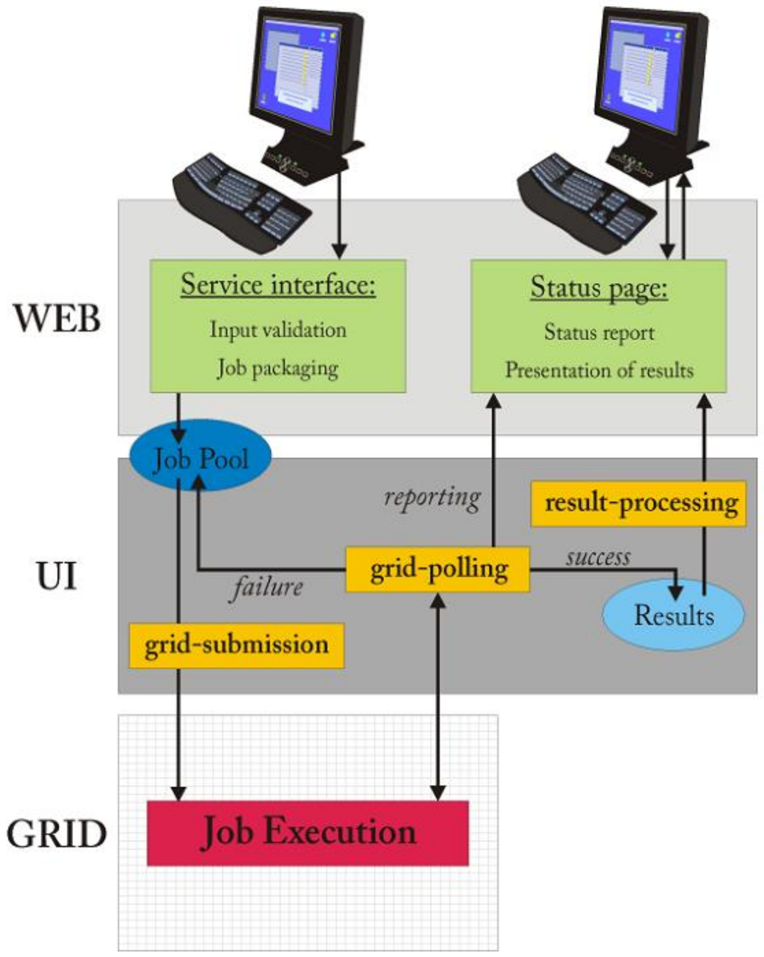
CS-Rosetta is a protocol which generates 3D models of proteins, using only the 13CA, 13CB, 13C', 15N, 1HA and 1HN NMR chemical shifts as input. Based on these parameters, CS ROSETTA uses a SPARTA-based selection procedure to select a set of fragments from a fragment-library (where the chemical shifts and the 3D structure of the fragments are known). The fragments are assembled using the Rosetta protocol. The CS-Rosetta portal is accessible from: http://haddock.science.uu.nl/enmr/services/CS-ROSETTA3/

A.2.5 UNIO

UNIO program enables users to perform automated NMR data analysis for 3D protein structure determination. The UNIO program includes data analysis algorithms for all parts of an NMR structure determination process ranging from backbone and side-chain assignment to NOE assignment and structure calculation. The UNIO portal is accessible from http://haddock.science.uu.nl/enmr/services/UNIO

A.2.6 General architecture of the UU portals

The implementation of UU web servers can be generalized as follows.



Briefly, a web interface collects and validates all necessary input. Upon successful user authentication the job is prepared for submission to the grid and added to the job pool. From there the job is submitted to the grid via DIRAC or gLite (CPU or GPU resources based on requirements). A polling daemon queries for the status of the job and updates the status page of the web interface. Upon job completion the results are retrieved, post-processed and returned to the user via the web interface.

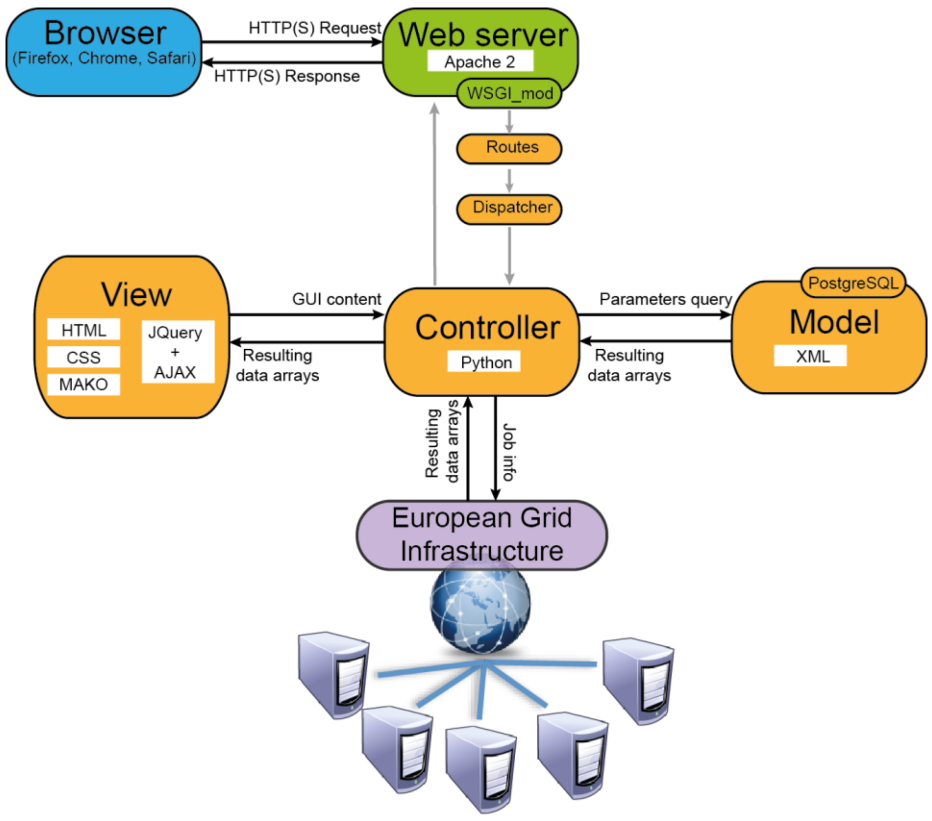
We have already investigated and successfully tested the possibility to use DIRAC to submit job to cloud VMs. This is a completely transparent process for the current submission machinery of the UU portals. Adapting to other solutions will have to be investigated once those are in place, with the important requirement that these should not disrupt our production portals (like the HADDOCK portal currently submitting >8 million jobs a year to the grid).

A.3 AMPS-NMR

A key computational technique in Structural Biology is Molecular dynamics (MD), a computer simulation of the physical movements of atoms and molecules as a function of time. MD simulations capture the behavior of biological macromolecules in full atomic detail using statistical thermodynamics laws. Such simulation may serve as a computational microscope, revealing biomolecular mechanisms at spatial and temporal scales that are difficult to observe experimentally. It has extensive application to biological systems, from protein folding to enzymatic catalysis and the comprehension of signaling cascades. MD-derived models allow researchers to interpret the experimental data and make testable predictions; the experimental data provide a quality assessment for the MD simulations as well as restraints for model refinement.

The AMBER- based Portal Server for NMR structures (AMPS-NMR) has been developed to provide an user-friendly entry point mainly designed for the energy refinement of NMR structures based on restrained molecular dynamics (rMD), but permitting also free MD simulations with the MD package AMBER. Users can select different predefined calculation protocols for rMD and free MD, depending also on the type of experimental data available.

The portal is based on an ad hoc mixture of HTML (http://www.w3.org/), CSS (http://www.w3.org/) and Mako (http://www.makotemplates.org/); it enables the users to create registered accounts through their own Digital Certificate. AMPS-NMR makes use of specific python-based utilities that automatically convert/adapt all input data, in the native format of various commonly used programs or databases (DYANA, CYANA, XPLOR, CNS, TALOS, PDB), to the AMBER format. This process is completely transparent to the user. Furthermore, AMPS-NMR exploits a series of python routines to collect all information about the new calculation, build all batch commands necessary to submit it, thereby creating a so-called job, and submit the calculation to the Grid Computing Infrastructure using gLite. At present, CPU or GPU resources are automatically selected by the portal, depending on the predefined calculation protocol chosen by the user. Finally, another relevant feature of AMPS-NMR is its persistent user workspace, where users' data and jobs are maintained and managed across multiple work sessions. This functionality has been implemented through the combined use of the HTTPS authentication protocol and of the SQLAlchemy (http://www.sqlalchemy.org/) Python Database Abstraction Library. A scheme of the AMPS-NMR portal is shown in the Figure below.



Recently, we have demonstrated the use of the Future Gateway, which is being developed in the INDIGO-DatCloud project, to submit AMBER jobs both to cloud and grid infrastructures. For this, the management of tasks within the web portal is done via ssh connections to the API server. The directories containing the data are shared using NFS over a local private network. More details are available at https://www.indigo-datacloud.eu/documents/first-toolkits-prototype-evaluation-report-d62.

Appendix B: Available Software Technology

B.1 gLite

gLite middleware was developed in the series of FP7 EGEE projects, and covered also by the follow-up EMI project. Currently, it is probably the prevailing middleware stack used by EGI sites. However, the number of components of original gLite and EMI distributions that are actively developed nowadays is reduced. In particular, support for EMI-WMS Workload Manager (the job dispatcher role in West-life architecture) is being phasing out, and alternatives for the job dispatcher component should be considered seriously when a new architecture is being designed.

On the contrary, CREAM computing element is being developed, therefore it can be relied on as the submission mechanism to the grid sites, as performed by e.g. DIRAC.

Suitability for West-Life: gLite has a wide existing user base, but it is not actively developed enough: after the end of EMI project in 2013 some of its components are in fact not supported anymore, while others are still supported by the individual development teams. Moreover, there are no plans to migrate towards cloud technologies.

B.2 DIRAC

Dirac (http://diracgrid.org/) is a software framework for distributed computing. Dirac builds software layer which is providing common interface to user for accessing heterogeneous resources. Dirac can be used for submitting jobs to both grid and cloud infrastructure. Support for cloud infrastructure is implemented in extension VMDIRAC, which supports among others OCCI and OpenStack APIs. Dirac is licenced under GNU General Public Licence Version 3. Dirac framework has been already used by one of West-Life portals (Haddock portal) for submitting its jobs to EGI grid infrastructure.

DIRAC is based on the pilot job approach [8]. The pilots are empty containers – jobs, which are submitted to the traditional grid resources, they pass through batch system queue, and as soon as they are executed, the make sanity checks (availability of disk space, memory, network connectivity, installed software etc.). If the check is passed, the pilot “calls home” the DIRAC server to retrieve actual payload. Certain number of pilots may fail to pass the sanity check, however, the failure due to instability of infrastructure etc. does not imply failure the payload.

Suitability for West-Life: Dirac is actively maintained and is compatible with hybrid grid/cloud setups. One of West-Life portals has experience with it, which makes Dirac one of the lead candidates.

B.3 HTCondor

HTCondor (https://research.cs.wisc.edu/htcondor/) is a batch queuing system, that has been is traditionally deployed in grid computing environments. The HTCondor software has been under development since 1988 at the University of Wisconsin-Madison. It is licensed under the Apache Licence 2.0. Before 2012 it was known as project Condor. HTCondor is mature and still actively developed software. New features including support for Docker and Cloud environments have been added.

Suitability for West-Life: HTCondor is actively maintained and very stable. From a pure technical point of view it is a good candidate, but the decisions related to its development are taken in the US, so compatibility with EU cloud infrastructure is not a priority.

B.4 Apache Cloud Software Family

B.4.1 Apache Mesos

Apache Mesos (http://mesos.apache.org/) is a scheduling and orchestration platform for managing cluster resources and submitting jobs. Apache Mesos clusters are based on usage of containerization technology, such as Docker and Linux Containers (LXC).

Mesos was presented for the first time in 2009. It is open source project licensed under Apache License 2.0. The project is under active development by Apache Software Foundation with large community of users and developers.

Apache Mesos offers functionality that crosses between Infrastructure as s Service (IaaS) and Platform as a Service. Application build on Mesos API are called frameworks. Apache Aurora, Marathon and Chronos are Mesos frameworks, which provides services of jobs and services scheduling.

Suitability for West-Life: Mesos is a great framework but is totally tied with Docker technology. This means that it will not be possible to achieve out of the box compatibility with existing grid systems, or even with existing EGI Federated Cloud systems. On the other hand, the Indigo services are based on Mesos, and the ambition of the project is to overcome those problems. Therefore, Mesos becomes a possible candidate for later stages of West-Life. CIRMMP has been testing the use of Chronos and Mesos within the INDIGO-Datacloud project for small-molecule docking based on free MD.

B.4.2 Apache Aurora

Apache Aurora (http://aurora.apache.org/) is a service scheduler, that runs on top of Apache Mesos. It is Mesos framework engineered for long running services, cron jobs and other ad-hoc jobs, that take advantage of Mesos platform. Aurora component is responsible for running the jobs. When any worker node experience failure, Aurora take care of rescheduling those jobs onto healthy worker nodes. For description of services is used Domain-Specific Language (DSL) which is highly configurable and supports templates.

Aurora was originally developed by Twitter (since 2010) and open sources in 2013. Currently it is project under active development by Apache Software Foundation. It is licensed under Apache License 2.0

B.4.3 Marathon

Marathon (https://mesosphere.github.io/marathon/) is a service scheduler, that runs on top of Apache Mesos. It provides functionality similar to Apache Aurora.

Marathon is developed by Mesosphere under Apache License 2.0.

B.4.4 Chronos

Chronos is another framework for Mesos. It was designed as fully-featured, distributed and fault tolerant scheduler, which eases the orchestration of jobs.

Chronos was originally developed by Airbnb. It is developed under Apache License 2.0

B.5 Virtual cluster management

B.5.1 Infrastructure Manager

Infrastructure Manager(www.grycap.upv.es/im/) has been developed by Polytechnic University of Valencia as s tool for virtual clusters deployment and orchestration. Both public (AWS, Google Cloud or Microsoft Azure) and on-premises (OpenNebula, OpenStack) IaaS providers are supported. OCCI interface and container technologies are also supported. It is licensed under GNU General Public Licence. Infrastructure definitions are written in proprietary RADL files. As part of the INDIGO-Datacloud project support for TOSCA documents is going to be added. For software deployments is IM integrated with Ansible.

B.5.2 Elastic Cloud Computing Cluster (EC3)

Elastic Cloud Computing Cluster (www.grycap.upv.es/ec3) is a tool developed to create elastic virtual clusters. Both public (AWS, Google Cloud or Microsoft Azure) and on-premises (OpenNebula, OpenStack) IaaS providers are supported. OCCI interface is also supported. EC3 is built on features of Infrastructure Manager for infrastructure deployment and CLUES (http://www.grycap.upv.es/clues/eng/index.php) for cluster elasticity.

B.5.3 Venus-C PMES-COMPSS

Venus-C platform was developed as a project funded under the European Commission's 7th Framework Programme between 2010 and 2012. It is not evolving since 2013.

B.5.4 Karamel

Karamel (http://www.karamel.io) is a set of management tools for provisioning, orchestration, monitoring and debugging of cloud systems and experiments. It is developed under Apache License 2.0. Public clouds like Amazon EC2 and Google Compute Engine and private clouds (Openstack) are supported. Support for OCCI has been recently added. For definitions are used Karmelfiles written in YAML. For orchestration of software packages Chef cookbooks are used. The mail aim of Karamel is reproducibility of distributed systems and experiments in cloud.
In contrast, the tool for West-Life use case should be able not only to start defined cloud environment, but also managed it during its whole lifecycle.

B.5.5 Cloudify

Cloudify (http://getcloudify.org/) is an orchestration developed by GigaSpaces Technologies.
It is licensed under Apache License 2.0.
Cloudify templates (called blueprints) are based on TOSCA (Topology and Orchestration Specification for Cloud Applications), which is an OASIS standard language for description of cloud based services. TOSCA

B.5.6 Terraform

Terraform (https://www.terraform.io/) is an orchestration or infrastructure management tool developed by HarshiCorp (www.hashicorp.com), the company well known for its widely used vagrant tool. Terraform is licenced under Mozilla Public Licence, version 2.0. It can be used to create, manage and manipulate resources provided by several IaaS providers. AWS, Google Cloud, Azure, Openstack and others are supported. OCCI interface is not currently supported, but is expected to be added in a few months by EGI Federated Cloud team.

B.5.7 Heat

Heat (https://wiki.openstack.org/wiki/Heat) is part of Openstack project. Heat is licensed under Apache License Version 2.0. It is native orchestration tool for cloud environments based on OpenStack infrastructure. Main advantage is integration with Openstack environment and resources and strong support from OpenStack. Heat templates use proprietary format based on YAML. Convertor for TOSCA documents is also available. All infrastructure resources like server, floating IPs, volumes, security groups, user and others can be described. Elasticity and autoscalling service is also available. Main disadvantage is lack of support for any other IaaS provider.

B.6 INDIGO-DataCloud solutions

INDIGO-DataCloud: Integrating Distributed Data Infrastructures for Global Exploitation (INDIGO from now on), is a HORIZON 2020 European Project aiming at addressing the challenge of developing advanced software layers deployable in the form of data/computing platforms targeted at scientific communities. In fact, new technological advancements such as virtualization and cloud computing pose new important challenges when it comes to exploit scientific computing resources.

B.6.1 Architecture

INDIGO has designed an architecture [9] that fill some existing gaps in the current solutions and provides progresses beyond the state-of-art at the three basic layers that compose a scientific computing infrastructure: the Infrastructure (or Resource Provider, or IaaS) layer, the Platform layer (PaaS) and the User interface layer.

The INDIGO approach to the application distribution and execution is based on Docker containers, and exploits Mesos [10] to manage cluster resources (cpu, memory) providing isolation and sharing across distributed applications. It also exploits Marathon [11] and Chronos [12], which are two powerful frameworks that can be deployed on top of a Mesos Cluster. In particular, Marathon is used to deploy, monitor and scale long-running services such as web portals, ensuring that they are always up and running. Chronos instead is used to run user applications (jobs), taking care of fetching input data, handling dependencies among jobs, and rescheduling failed jobs. All the application executions are described exploiting a TOSCA Template via simple APIs or Portlets. The input/output are automatically managed by the PaaS layer (via OneData and external endpoints), and the geographical data-aware scheduling is provided by INDIGO PaaS Orchestrator

B.6.2 Components and Releases

The first public release of INDIGO software components is scheduled by the end of July 2016, while a second major release is planned by the end of March 2017, six months before the official end of the project.

The first release will provide solutions that can be grouped in the following way:

1. Site level solutions:

    1.1. New scheduling algorithms for allocation of resources by open source Cloud frameworks: both fair-share scheduling and spot instances.

    1.2. Support for dynamic partitioning of resources among “traditional batch systems” and Cloud infrastructures.

    1.3. Full and transparent support for containers, with or without Docker, by providing extensions to OCCI functionalities

    1.4. Improved QoS capabilities of storage resources and data lifecycle support.

    1.5. Improved automation at IaaS level, based on TOSCA.

2. Data management solutions: a set of components called Unified Data Access (UDA) layer that includes OneData [13], FTS-3 [14] and Dynafed [15]). The UDA layer will provide a PaaS level interface for end users, including REST APIs and the CDMI (Cloud Data Management Interface) standard protocol, offering the following functionalities:

    2.1. Distributed data federations supporting legacy applications as well as high level capabilities for distributed QoS and Data Lifecycle Management. This includes e.g. integrated local and remote Posix access for all types of resources (bare metal, virtual machines, containers).

2.2. Support for distributed data caching mechanisms and integration with existing storage infrastructures. INDIGO storage solutions are capable of providing efficient access to data and of transparently connecting to Posix file systems already available in data centers.

2.3. Transparent client-side import/export of distributed Cloud data. This supports dropbox-like mechanisms for importing and exporting data from/to the Cloud. That data can then be easily ingested by Cloud applications through the INDIGO UDA tools.

2.4. Linux, Mac OS, Windows desktop support.

3. Automated solutions (PaaS Orchestrator):

3.1. Standard interface to access PaaS services. Currently, each PaaS solution available on the market is using a different set of APIs, languages, etc. INDIGO will use the TOSCA standard to hide these differences.

3.2. Selection of resources across multiple Cloud providers (e.g. depending on data location or resource requirements), as well as support for application requirements in Cloud resource allocations (e.g. for what regards InfiniBand or GPUs). TOSCA templates used to specify resource requirements, dependencies, and configuration of the services/applications.

3.3. Support for dynamic and elastic clusters of resources. Resources and applications can be clustered through the INDIGO APIs. This includes e.g. batch systems on-demand (such as HTCondor or Torque) and extensible application platforms (such as Apache Mesos) capable of supporting both application execution and instantiation of long-running services.

3.4. Support for custom frameworks for porting arbitrary applications to the Cloud, with automated monitoring and scalability.

3.5. Integrated support for high-performance Big Data analytics. This includes e.g. general purpose engines for large-scale data processing such as Spark.

4. User level solutions (FutureGateway APIs and Portal):

4.1. Customizable / programmable portal (gateway) engine integrated with the features mentioned in the previous solutions.

4.2. Sample portals delivered for selected applications ("all-in-one", “plug and play” bundles).

4.3. Mobile toolkit to access INDIGO features on mobile devices.

5. AAI advanced interoperable solutions (IAM, Identity and Access Management Service):

5.1. Support for SAML, OpenID Connect and X509 user authentication.

5.2. Identity harmonization by linking heterogeneous authentication mechanisms to a single VO identity.

5.3. Management of VO membership (i.e., groups and other attributes), provisioning of VO structure and membership information to services.

5.4. Management, distribution and enforcement of authorization policies.

5.5. Support for controlled delegation of privileges across services and credential translation, through the integration with the INDIGO Token Translation Service (TTS).

.

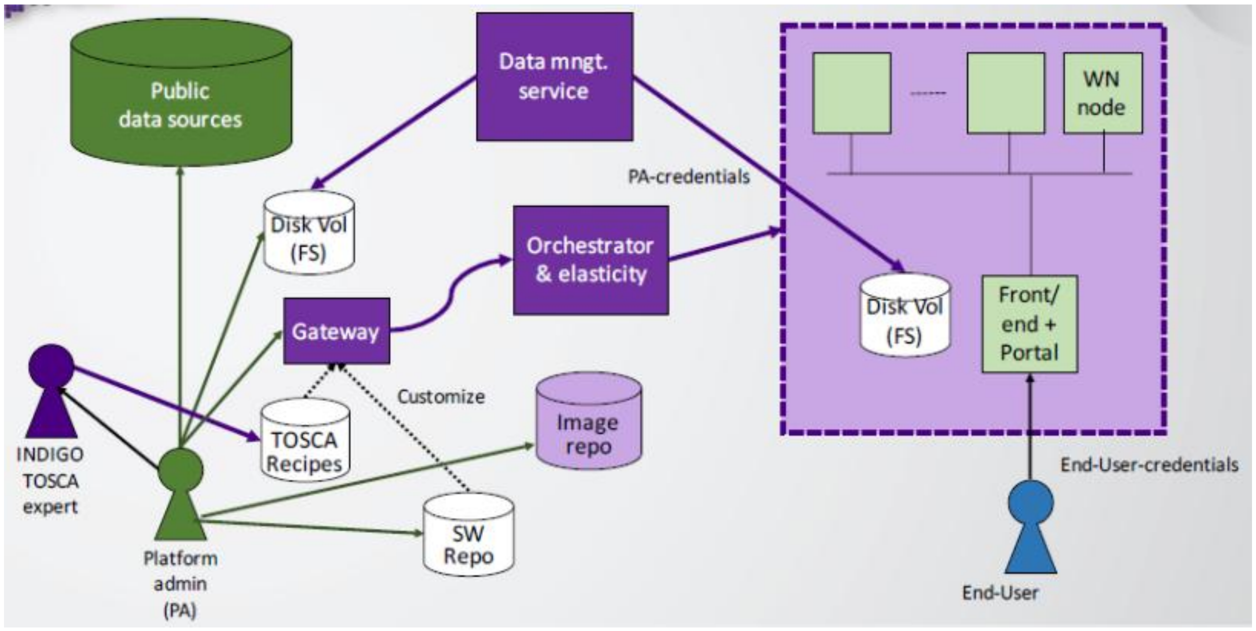
The figure below shows the architecture guiding the use case:

- The INDIGO TOSCA expert helps the Platform Administrator (PA) to write the TOSCA template to automatic deploy and configure the virtual infrastructure on the Cloud suited for the portal application

- The PA create and publish the VM and Docker images needed for the portal and for the application execution, and submit the TOSCA template to the INDIGO Gateway API component, which in turn calls the PaaS Orchestrator component.

- When the virtual infrastructure is up and running, the end user access the application portal which sits on top of the batch system front-end, and submits his task. The scalability of the entire system is ensured by load balancing and elasticity capabilities implemented in the Orchestrator PaaS component. E.g. the number of nodes available for computing should increase (scale out) and decrease (scale in) according to the workload. The system should also be able to do Cloud-bursting to external infrastructures when the workload demands it.

- Input data can be either supplied by user or extracted from public repositories (Protein DataBank, EM DataBank, etc.). Output data (order of GB) can be kept for two weeks for user to download it and deleted after that. Moreover, INDIGO data management services and tools can be used for allowing remote Posix access to data, or providing dropbox-like mechanisms for importing and exporting data from/to the Cloud.



At the time of writing, that is before the first official release, the first prototype of the INDIGO architecture is ready for testing at [2].

Several components of INDIGO architecture, once demonstrated their maturity, can be suited for West-Life:

- The IAM service, for harmonizing the AAI infrastructure by mapping different digital identities (e.g. institutional credentials, social logins, X509 certificates) to the same individual so that consistent authorization and accounting can be performed at various levels of the infrastructure.

- The PaaS Orchestrator, to coordinate the deployment process of services and applications on a distributed infrastructure.

- The Data Management services, that provide an abstraction layer for accessing data storage in a unified and federated way, also providing capabilities for importing data and scheduling transfers of data.

- FutureGateway APIs and portlets, for helping West-Life developers to build portals which easily integrates with INDIGO IAM, PaaS and Data services.

B.7 EUDAT services

EUDAT [16] is the European e-infrastructure aiming to provide data storage, consultancy and related services for research communities. It provides a mature B2DROP service - Dropbox like service giving access to the virtual storage via Owncloud client (online/offline synchronization), over the WEBDAV protocol. The other services B2SHARE can make the data available publicly with an integration with annotation and metadata information (B2NOTE in roadmap), they heavily utilize existing open source solution for it (Owncloud, Virtuozo, …). B2ACCESS is recommended for federated authentication and authorization management. Instruct has worked with EUDAT to integrate their Authorization services.

References cited

[1] West-Life Deliverable D5.2, http://internal-wiki.west-life.eu/w/index.php?title=File:Overview_(baseline)_of_services_and_portals_to_be_integrated_into_the_new_VRE.docx

[2] Topology and Orchestration Specification for Cloud Applications (TOSCA), OASIS Standard, https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca

[3] EGI Federated Cloud, https://www.egi.eu/infrastructure/cloud/

[4] West-Life Deliverable D6.1, http://internal-wiki.west-life.eu/w/index.php?title=D6.1:System_Design

[5] Cloudify OCCI plugin prototype, https://github.com/vholer/cloudify-occi-plugin-experimental

[6] INDIGO Deliverable D2.4: https://www.indigo-datacloud.eu/documents/confirmation-support-initial-requirements-and-extended-list-d24

[7] INDIGO Deliverable D6.2: https://www.indigo-datacloud.eu/documents/first-toolkits-prototype-evaluation-report-d62

[8] A.Casajus et al., DIRAC pilot framework and the DIRAC Workload Management System, J. Phys. Conf. Series, Volume 219, Part 6, 2010.

[9] INDIGO-Datacloud: foundation and architectural description of a Platform as a Service oriented to scientific computing, http://arxiv.org/abs/1603.09536

[10] Apache Mesos, http://mesos.apache.org/

[11] Marathon, https://mesosphere.github.io/marathon/

[12] Chronos, http://mesos.github.io/chronos/

[13] OneData, https://onedata.org

[14] FTS-3, http://fts3-service.web.cern.ch/

[15] Dynafed, http://lcgdm.web.cern.ch/dynafed-dynamic-federation-project

[16] EUDAT, https://eudat.eu/