Research and Innovation Action (RIA) H2020 – 957017

Stream Learning for Multilingual Knowledge Transfer

https://selma-project.eu/

D4.3 Intermediate platform with continuous massive stream learning NLP capabilities

Work Package	4
Responsible Partner	IMCS
Author(s)	Guntis Barzdins, Sebastião Miranda, Didzis Gosko
Contributors	Afonso Mendes, Arturs Znotins, Mikus Grasmanis, Paulis Barzdins, Roberts Dargis, Normunds Gruzitis
Reviewer	Yannick Esteve
Version	1.0
Contractual Date	31 December 2022
Delivery Date	22 December 2022
Dissemination Level	Public

Version History

Version	Date	Description
0.1	30/10/2022	Initial Table of Contents (ToC)
0.2	18/11/2022	Section on DockerSpaces drafted
0.3	20/11/2022	Overall deliverable structure drafted
0.4	23/11/2022	Section on integration with Monito and Plain X added
0.5	08/12/2022	Ready for internal review
0.6	21/12/2022	Internal review feedback received, final updates
1.0	22/12/2022	Publishable version

Executive Summary

The Intermediate platform release with continuous massive stream learning capabilities covers two main Use Cases UC1, UC2 of the SELMA project, along with internal testing Use Case UC0. Focus is on the common Maestro Orchestrator streaming NLP backend enhanced by the DockerSpaces technology recently developed within the SELMA project. The novel DockerSpaces technology enables massive scaling of above mentioned two demonstrators with vastly different frontend GUIs and NLP pipelines tailored to the specific Use Cases.

This document provides an overview of the intermediate SELMA platform release to be followed by the final release later in the project. The document also discusses the "continuous" aspect of long-term software support in SELMA project with the focus on DockerSpaces as a potential solution.

Table of Contents

Table of Figures

1. Introduction

Continuous massive stream processing (including *continuous massive stream learning* described in the deliverables D2.1, D2.2, D2.4, and D2.5) in SELMA platform is enabled via SELMA DockerSpaces technology or via external commercial services like IBM Watson, MS Azure, AWS, etc. as shown in Figure 1.

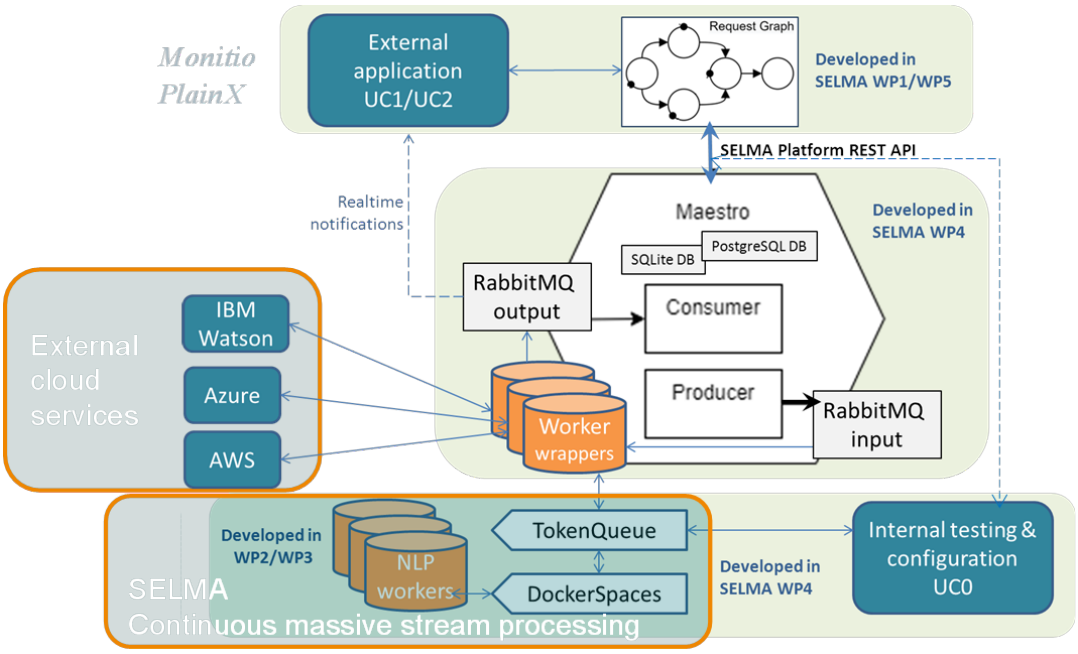

Figure 1 *SELMA Platform architecture: continuous massive stream processing via DockerSpaces load-balancer or via external cloud services*

DockerSpaces approach enhances the existing SELMA platform as described in the previous deliverables D4.1 and D4.2 with the highly scalable yet simple continuous massive stream processing capability. This new capability is essential for delivering continuous massive stream learning technologies developed in WP2 and WP3 to the media monitoring Use Case 1 (Monitio), media production Use Case 2 (Plain X), and other use cases (such as Use Case 0 for testing and configuration, Podcast creation of Use Case 2, external LETA Use Case) described

in deliverables D1.2 and D1.3. DockerSpaces will be released as open-source software with potential applications also outside SELMA project.

2. DockerSpaces approach to continuous massive stream processing

DockerSpaces is an original compute-cluster load-balancing technology developed within the SELMA project and improves upon the original TokenQueue technology "side-car" implementation introduced in the earlier SELMA deliverables D4.1 and D4.2. DockerSpaces was developed in the IMCS Docker development lab to provide support for both x86 and ARM architectures along with GPU acceleration. SELMA DockerSpaces[1] is an alternative to the popular cloud infrastructure scaling and management platforms such as Kubernetes, NGINX, Docker-Swarm, HashiCorp Vagrant, etc. The key innovation of DockerSpaces compared to the above-mentioned platforms is "zero management" – as shown in Figure 2, DockerSpaces is a state-less service launched on top of standard Docker installation (on a single server or in a cluster of servers), to which all state-configuration parameters are passed along with every incoming REST API call as part of the URL path.

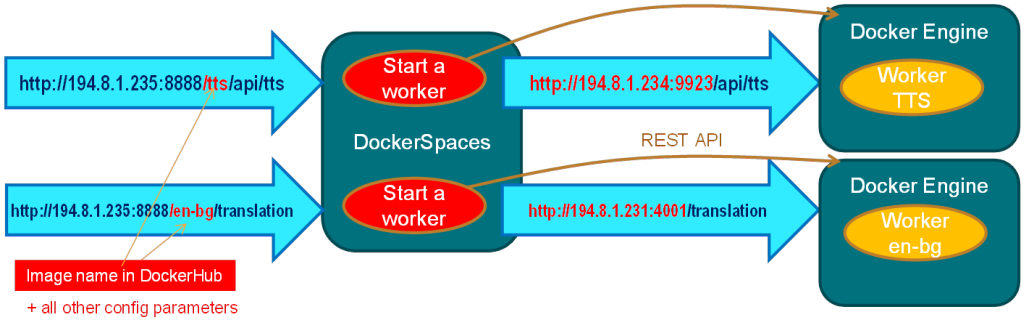

Figure 2 *A simplified diagram of SELMA DockerSpaces "zero management" principle*

[1] SELMA DockerSpaces will be open-sourced at https://github.com/SELMA-project/docker-spaces

The “zero management” implies not only simplicity but also universality of the DockerSpaces approach. A DockerSpaces compute-cluster is not only completely agnostic to the kinds of jobs to be executed (any DockerHub container can be executed via REST API call), but services also scale automatically with the influx of the various kinds of REST API calls.

The state-less architecture allows DockerSpaces replication for extreme scalability via round-robin Domain Name System (DNS) configuration – a globally distributed farm of DockerSpaces installations on different IP addresses can reside under the same DNS name used by the incoming REST APIs, thus enabling virtually unlimited scalability in default round-robin or more intelligent Akamai-DNS approach. DockerSpaces scalability is further enhanced by the support of “serverless” frontend NLP pipelines – it means that all logic of an NLP application can reside inside the web page JavaScript loaded in the users' web browser – JavaScript from the web page then directly calls REST API services in the DockerSpaces compute-cloud thanks to the Cross-Origin Resource Sharing (CORS) support in DockerSpaces. Besides simplicity, this enables easy NLP application scaling by foregoing the need for developing and running a (potentially bottlenecking) custom backend services. SELMA Testing and Configuration UseCase0 (UC0) at https://selma-project.github.io serves as a demo of such massively scalable frontend NLP pipeline.

It should be noted that DockerSpaces is well suited for massive scaling of simple pipelines (such as NLP processing pipelines) but is not a complete replacement of NGINX, Kubernetes or Docker-Swarm. The main limitation of DockerSpaces is support for execution of only isolated non-persistent Docker containers, rather than internally networked Kubernetes-pods or Docker-Compose applications with potentially persistent data storage. This intentional limitation is the key enabler for the DockerSpaces unparalleled simplicity and efficiency. Another limitation of DockerSpaces is the lack of built-in authentication, accounting, and web server functionality – they are omitted for the state-less simplicity and efficiency reasons.

Both mentioned limitations of DockerSpaces can be partially mitigated. The applications requiring persistence can be included as separate processes[2] in the DockerSpaces container

[2] optionally, persistent processes can be scaled with xinetd deamon

itself (see Figure 3), which by default is run in the persistent mode with the "--restart=always" option as illustrated in Section 2.1. Meanwhile authentication, accounting, and web server functionality can be added through pairing DockerSpaces with a state-full server (run as a separate process inside the persistent DockerSpaces container) such as PiniTree server (described in D2.2) illustrated in Figure 3.

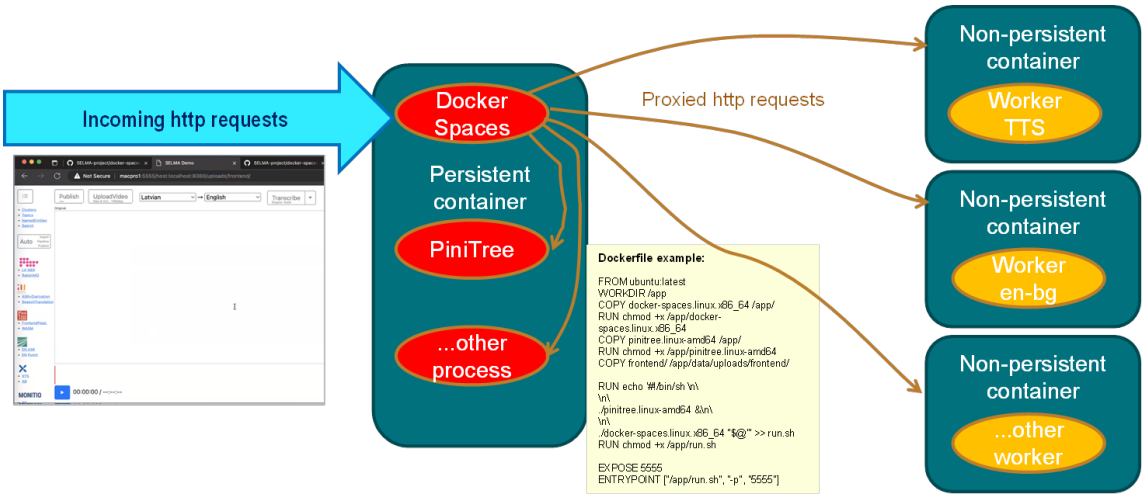

Figure 3 *Example of a persistent PiniTree server run inside the DockerSpaces main container*

Why DockerSpaces was needed? Because in SELMA project we deal with hundreds of Docker containers each the size of 2-10GB RAM footprint. Legacy solutions would keep them all active in RAM, thus consuming vast resources. DockerSpaces allows to dynamically start/stop these containers upon demand; moreover, DockerSpaces can start several copies of the same container if the demand is high, thus scaling seamlessly both downwards and upwards. In this way even a small computer can serve hundreds of Docker images, if they are not used simultaneously (thanks to the TokenQueue queueing implemented inside DockerSpaces), or a large compute cluster can dynamically scale number and type of workers according to the demand. DockerSpaces precisely fulfil the SELMA platform requirements as described in D1.1 and D1.3.

2.1 Starting and using DockerSpaces

Setting up DockerSpaces on any Linux server, where Docker is already installed, is as easy as executing a single command (no installation required):

docker run -p 9100:8888 -d -v /var/run/docker.sock:/var/run/docker.sock

--restart=always selmaproject/spaces:latest

With DockerSpaces running, one can launch and use any public Docker container from the DockerHub (e.g.: https://hub.docker.com/repository/docker/selmaproject/tts) by merely visiting the following URL from the browser (or constructing an appropriate REST API call):

http://ents.pinitree.com:9100/x:selmaproject:tts:777:5002/

This would launch a SELMA Brazilian Portuguese Text to Speech (TTS) container from the DockerHub as shown in Figure 3.

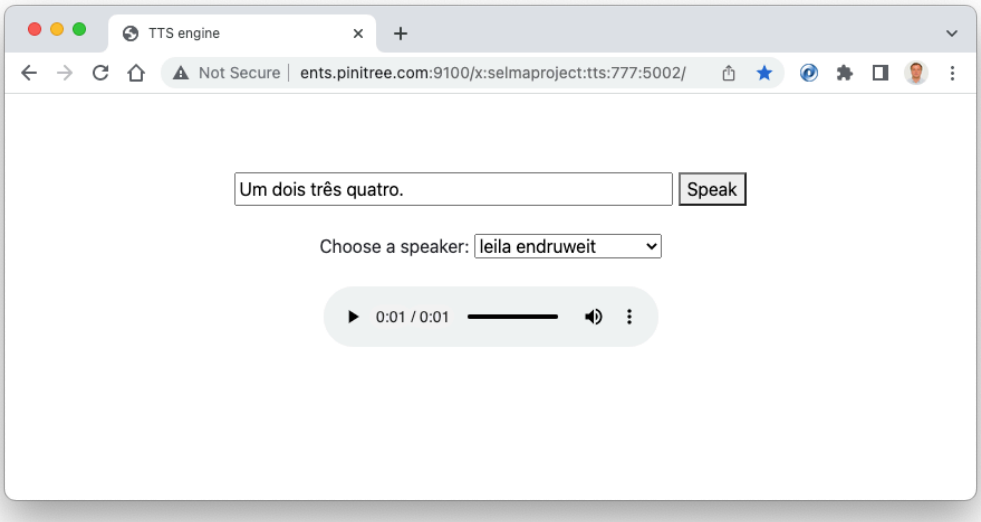

Figure 4 *Example of Text to Speech service launched via SELMA DockerSpaces*

During the first call to DockerSpaces one has to be patient – the specified container is pulled on-the-fly from DockerHub and cached locally for the future use; download may take about 5 minutes for a large 6GByte TTS neural-network container illustrated here, before the response will appear in the web browser.

The field *"/x:selmaproject:tts:777:5002/"* in the URL path passed to DockerSpaces is stripped before handing the URL path to the actual Docker container; this filed is only used by the DockerSpaces to acquire a temporary state-configuration for this HTTP request (*x*=single thread container[3] : *selmaproject* = DockerHub organisation : *tts* = repository : *777* = tag : *5002* = active port of the container). DockerSpaces ability to scale single thread containers via "x" mode effectively provides the "massive" aspect to the SELMA platform.

2.2 Internal architecture of DockerSpaces

DockerSpaces is an open-source software and full details can be obtained by visiting the DockerSpaces GitHub repository https://github.com/SELMA-project/docker-spaces and documentation there.

Here we outline only the core design ideas behind the DockerSpaces, which are implemented in the Go language and released as a single universal binary file (no installation scripts required).

From the user point of view DockerSpaces is a primitive reverse-proxy similar to the popular NGINX reverse-proxy – it accepts TCP connections on the specified IP address and port and interprets them via HTTP or HTTPS protocol to further connect to the Docker containers launched on-demand depending on the configuration parameters extracted from the GET or PUT request path as was illustrated in the Figure 3. The tricky part starts when there are more and varied incoming HTTP requests than there are CPU and RAM resources in the underlying server – in this case requests have to be queued and load-balancing has to be enacted to launch the optimal number of Docker containers for each job-type and to orderly route queued HTTP requests to these containers evenly.

[3] Most WP2 and WP3 developed neural NLP containers are SingleThreadContainers – with "x" mode DockerSpaces allow to forego smart batching or internal queueing in NLP containers; DockerSpaces "y" mode supports also multithread containers

To handle the queues and load-balancing, at the heart of DockerSpaces is the Broker module, which implements the critical TokenQueue algorithm initially described in the SELMA deliverable D4.1. The DockerSpaces Broker module in action is shown in Figure 5.

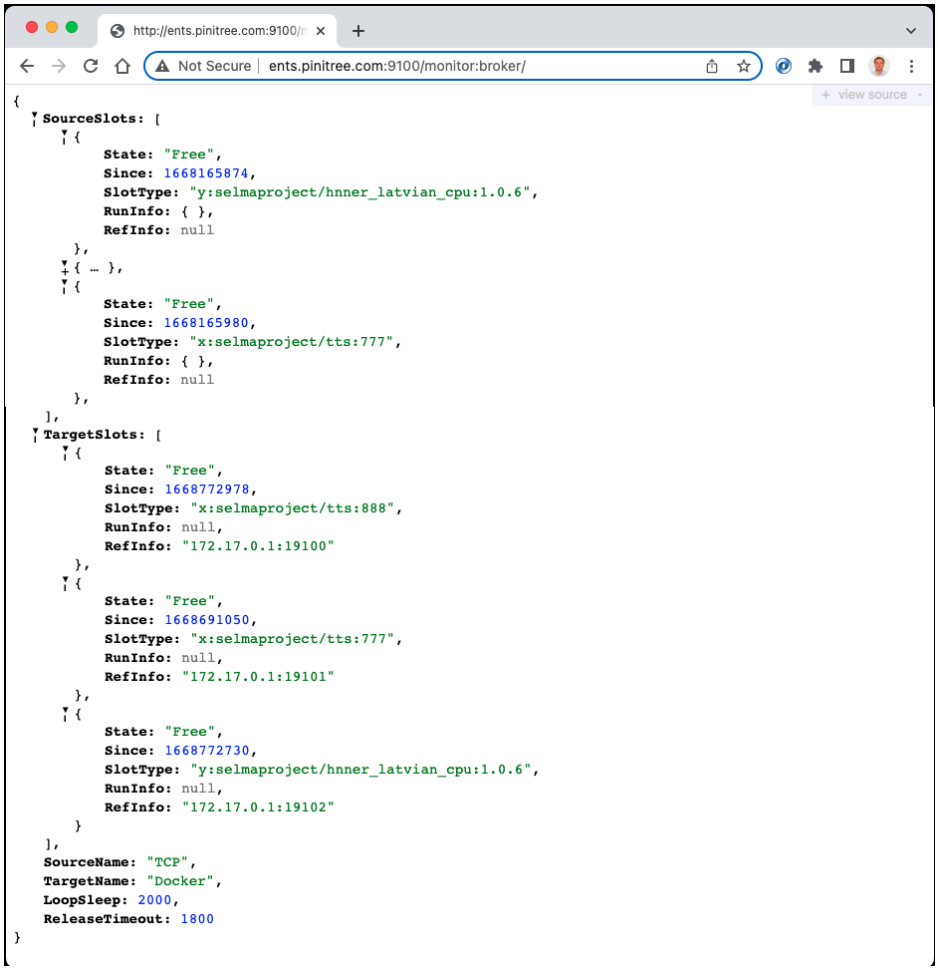

Figure 5 *DockerSpaces Broker module implements the TokenQueue algorithm*

The Broker module consists of the SourceSlots (incoming TCP/HTTP sessions) and TargetSlots (currently running Docker containers), see Figure 6. Both SourceSlots and TargetSlots have a field named “SlotType”, which needs to match for the TargetSlot to be able to serve the SorceSlot.

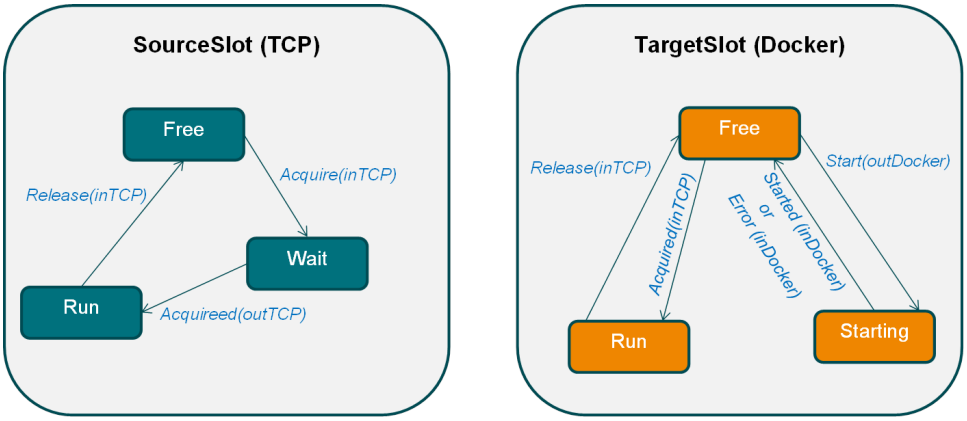

Figure 6 *DockerSpaces Broker module state diagram*

SourceSlots are activated by the standard TokenQueue protocol message-passing sequence "Acquire", "Acquired", "Release" causing the assigned SourceSlot to switch between the states "Free", "Wait", "Run", "Free".

TargetSlots have two switching sequences.

1. "Free", "Run", "Free" sequence is triggered by the SourceSlot in the "Wait" state, if it finds a "Free" TargetSlot with the matching "SlotType"; this is the most common sequence where the incoming HTTP request is served by an already started Docker container. Once the HTTP request is served, "Release" message returns both SourceSlot and TargetSlot into the "Free" state.
2. "Free", "Starting", "Free" sequence is triggered by the Broker internally to handle load-balancing through changing the "SlotType" of the TargetSlot; this is achieved by killing the previous Docker container and starting a new Docker container on the host/port specified in "RefInfo". The DockerSpaces Docker-protocol driver communicates with the TargetSlot via message sequence "Start", "Started" or "Start", "Error" depending on drivers' ability to pull from DockerHub a docker container specified in the "SlotType" field – "Error" message is typically caused by the misspelled name or non-existent Docker container in the the DockerHub, or container crashing during the startup.

To assure non-blocking operation of the Broker module, it communicates with other DockerSpaces modules exclusively via standard message-passing mechanism of the Go language.

```
ents@ubuent:~/build21$ ./docker-spaces.linux.x86_64 --help
Usage of ./docker-spaces.linux.x86_64:
  -cert string
        certificate path (default "./certs/localhost.pem")
  -cluster value
        cluster configuration in following format: <host-start-port>:<target-slot(docker-runner)-count>:<docker-host-url>
  -cors
        enable CORS
  -docker string
        set path to docker engine, e.g., unix:///var/run/docker.sock, ssh://user@remote-host or http://remote-host (default "unix:///var/run/docker.sock")
  -email string
        docker registry email
  -gpu
        enable GPU
  -key string
        private key path (default "./certs/localhost-key.pem")
  -loop-sleep int
        broker loop sleep in milliseconds (default 2000)
  -p int
        port number (default 8888)
  -password string
        docker registry password
  -port int
        port number (default 8888)
  -registry string
        docker registry address (defaults to hub.docker.com registry)
  -release int
        container release timeout in seconds (default 1800)
  -s    use TLS
  -source int
        number of source (TCP) slots (default 3)
  -start-port int
        start port of docker containers port range (default 9100)
  -stop int
        container stop timeout before kill in seconds
  -target int
        number of target (docker) slots (default 5)
  -tls
        use TLS
  -user string
        docker registry username
ents@ubuent:~/build21$
```

Figure 7 *DockerSpaces command-line parameters*

DockerSpaces has a number of command-line parameters (see Figure 7), which enable sizing and scaling DockerSpaces across multiple computers in the cluster, providing access to the CUDA GPU resources, using private container repository, and more. Particularly, the command-line parameter "-release" (default 1800 seconds) can be set to a higher value if DockerSpaces is used to schedule batch-jobs such as neural network training jobs or scaling a worker pool servicing a Rabbit MQ – both of these uses are very relevant to SELMA, as they enable continuous massive stream learning via continuous retraining of neural models, as well as tight integration with Maestro Orchestrator employing RabbitMQ. These command-line parameters can be passed to both the standalone DockerSpaces executable, or optionally also to DockerSpaces packed as a Docker container (as was illustrated in the Section 2.1).

2.3 Integration of Docker Spaces in Monitio and Plain X

The UC1 and UC2 prototypes, Monitio and plain X, take advantage of the worker management and scalability of DockerSpaces through the SELMA Maestro orchestrator (introduced in D4.1). In this setup (Figure 8), Maestro handles the orchestration of dependency graphs of NLP jobs so that they occur in the correct order, and DockerSpaces handles worker allocation, placement and replication. This architecture allows plain X and Monitio to scale by integrating more compute servers for NLP workers to scale into.

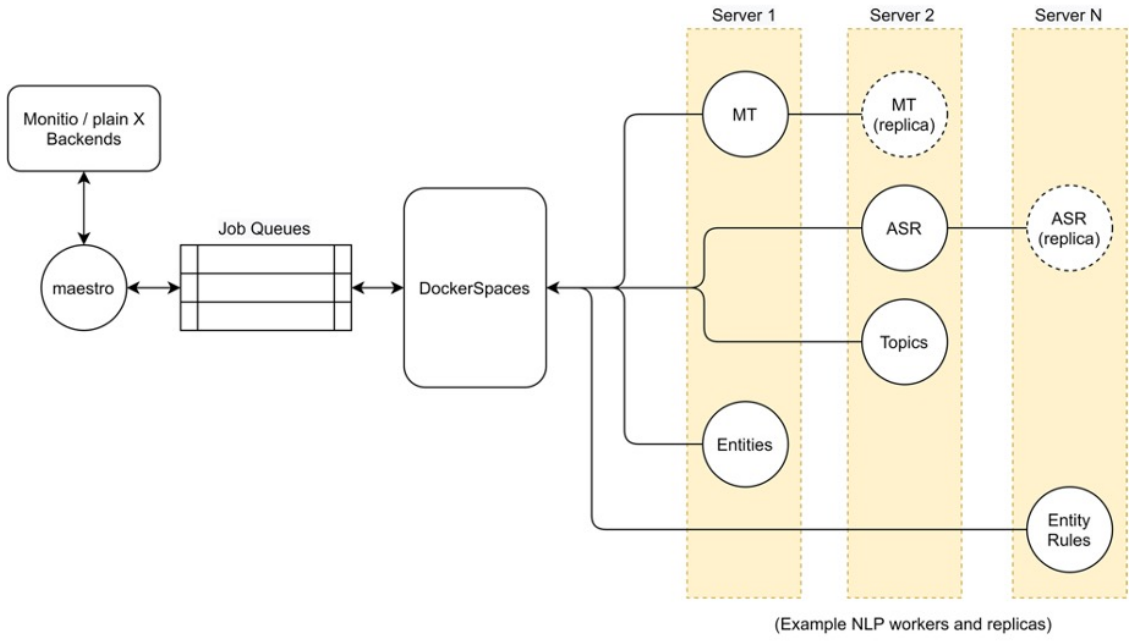

Figure 8 *How DockerSpaces integrates with plain X and Monitio*

DockerSpaces is being integrated in the UC1 and UC2 prototypes, Monitio and plain X, respectively. At the time of the writing of this deliverable, plain X is already calling translation models through DockerSpaces, whereas the work in Monitio is in progress.

3. Continuous software support in SELMA

In SELMA project we developed the new open-source DockerSpaces technology to streamline and scale AI applications in the NLP domain, particularly those developed within the SELMA WP2 and WP3. The issue at stake is that, even open-source libraries change, and then a code written in 2022, may not work in 2026 (or later), as our consortium has already witnessed from the SUMMA project (2016-2019), a direct predecessor of the current SELMA project (2021-2023).

Continuous maintaining of systems that depend on external libraries and dealing with legacy systems is a never-ending challenge from software engineering point of view. For many reasons, including security ones, software needs to be updated often. However, such updates may break the current stack. Also, for some languages and frameworks, e.g., R or Python, packages need to have the commitment of a developer to solve its bugs. Otherwise, they are removed from the current packages and left as legacy. Thus, a process is needed if we want to support, for a few years, the current software stack.

With DockerSpaces we attempted to solve the software longevity / reliability issue by "freezing" the versions of the software and packages used and keep them as a Docker container image with explicit version control and local backup in the form of large ZIP files, to gain independence from the DockerHub cloud infrastructure. DockerSpaces allow to largely forego external cloud infrastructures such as MS Azure, Google Cloud, Digital Ocean, AWS, IBM Watson, HuggingFace.io, etc. as they are troublesome over time – the cloud service providers change or retire APIs, change pricing schedules, privacy and data security requirements may change.

The "freezing" requirement applies also to the underlying operating system. Ubuntu has a set of releases, the Long-Term Support (LTE) ones, that keep the stable versions. In SELMA we have created an "air gapped" Ubuntu Linux 20.04 LTS installation package with Docker and GPU CUDA drivers to protect against background software updates over the Internet, as well as expired certificates and similar causes of long-term service failures, see https://github.com/SELMA-project/BootFlash.

DockerSpaces approach has got initial traction also outside SELMA project – DockerSpaces were evaluated by the Deutsche Welle computing operations staff, integrated with the IMCS startup PiniTree.com ontology editor software suite, and included as a software and data archiving technology in the multilateral proposal submitted to HORIZON-CL4-2022-HUMAN-02 call. Our intention is to promote DockerSpaces widely in the open-source community along with other recent "Spaces" initiatives like GitHub Spaces, HuggingFace Spaces, Google Spaces, Data Spaces, etc.

4. Conclusion

This document presents the intermediate SELMA platform release (originally described in deliverables D4.1 and D4.2) with the focus on the novel DockerSpaces technology as the enabling factor for the continuous massive stream learning developed in WP2, WP3 and described in separate deliverables D2.1, D2.2, D2.4, and D2.5.