OPUS-HSO

Repositorium der Hochschule Offenburg
https://opus.hs-offenburg.de

Gaia-X4KI - TP2.1 Scheduler Report

Dominik Straßel, Ralph Isenmann, Janis Keuper

Zitiervorschlag im APA Stil:

Straßel, D., Isenmann, R., & Keuper, J. (2024). *Gaia-X4KI - TP2.1 Scheduler Report* (Release V2).

Abstract

Tech Report

Nutzungsbedingungen

Dieses Dokument wird unter diesen Bedinungen zur Verfügung gestellt:
Urheberrechtlich geschützt
Für weitere Informationen siehe:
https://rightsstatements.org/page/InC/1.0/

Kontakt

Hochschule Offenburg | Bibliothek
Badstraße 24
77652 Offenburg
Telefon: (0781) 205-240
E-Mail: bibliothek@hs-offenburg.de
www.hs-offenburg.de/bibliothek

GAIA-X 4 KI

Ein Projekt der GAIA-X 4 Future Mobility Projektfamilie

TP2.1 Scheduler Report

D. Straßel, R. Isenmann, and J. Keuper

May 14, 2024

Contents

1 Introduction **1**
1.1 Versioning of this document . . . 1
1.2 Relevant Literature . . . 2
1.3 Overview Scheduler . . . 3

2 Orchestrator Independent Resource Description **4**
2.1 What is TOSCA? . . . 5
2.2 How TOSCA works . . . 5
2.3 Usability . . . 6

3 Deliverable E2.1.1: Schedulers Overview **7**
3.1 Summary E2.1.1 . . . 16

Document History

- December 1, 2021: Release V1
- May 14, 2024: Release V2 (ongoing)

1 Introduction

1.1 Versioning of this document

It is intended to process this document in several iterations. Each version should narrow down technical solutions for the scheduling in Gaia-X4KI.

- V1 [current state]: Broad technical overview of orchestration tools
  - **Deliverable E2.1.1**
  - Gathering information from the current Gaia-X **"Architecture of Standards"**
  - Gathering technical details and capabilities of available schedulers/orchestrators
  - Recommend 3-5 candidates for further investigation
- V2 [Q1 22]: Use case requirements
  - present V1 to use cases
  - sync with TP2.2 (Containers)
  - gather feedback on technical requirements
  - match requirements with candidate list
- V3 [Q2 22]: Test setup on HSO cluster
  - test and evaluate candidate
  - create feature list for middle layer -> which parts need to be implemented?
- V4: Check against Gaia-X "Architecture of Standards" updates

1.2 Relevant Literature

- Gaia-X Architecture Document (10/2021) [1]
- Gaia-X Architecture of Standards Document (11/2021) [2]
- Software Requirements Specification for Gaia-X Federation Services – Integration & Portal Orchestration (2021) [3]
- Topology and Orchestration Specification for Cloud Applications Version 1.0 (11/2013) [4]
- TOSCA Simple Profile in YAML Version 1.3 (02/2020) [5]
- Security Aspects:
    - German Federal Office for Information Security APP.4.4 Kubernetes (Edition 2022) [6]

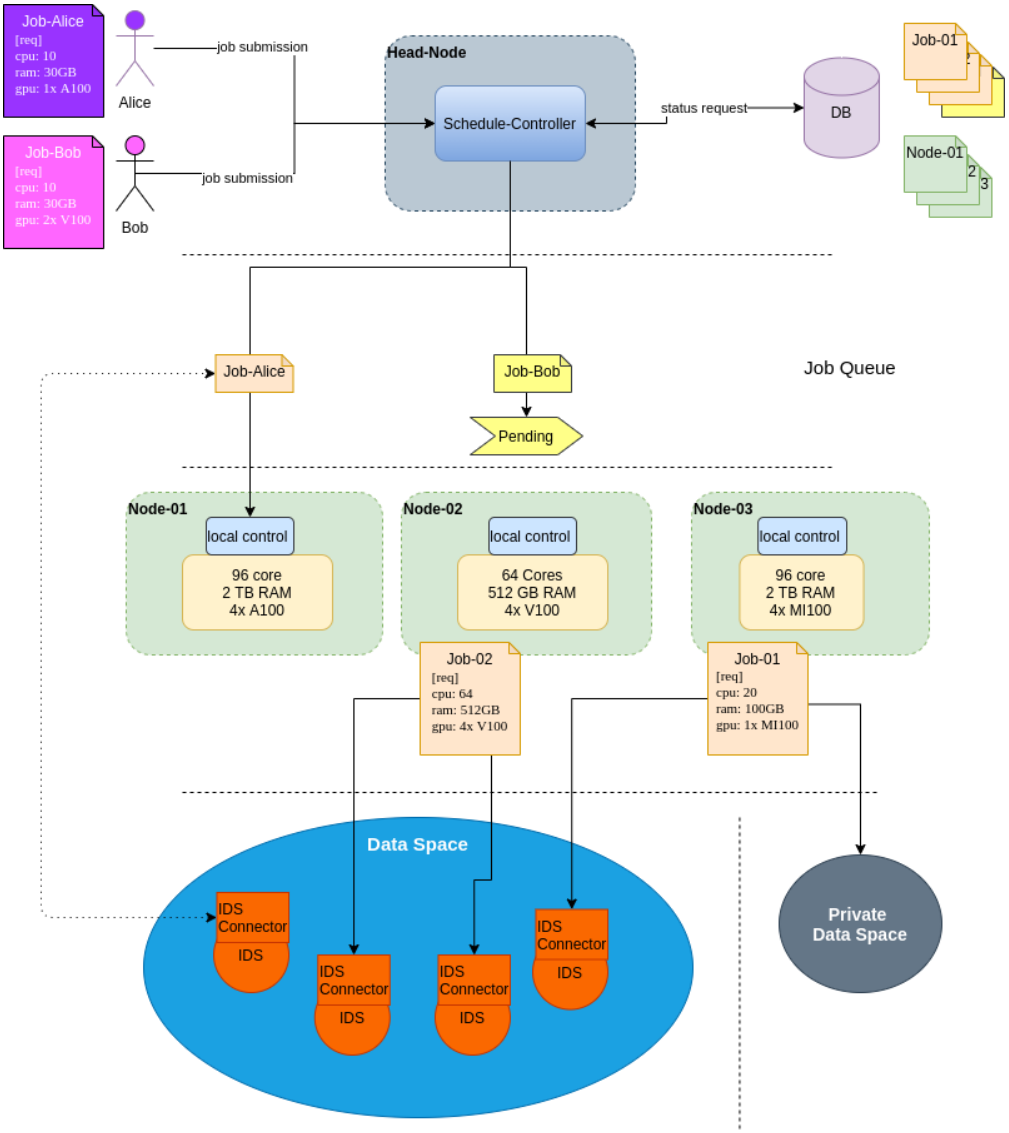

FIG. 1: *Schematic overview of Scheduling-/Orchestrator-Software*

1.3 Overview Scheduler

No matter if we talk about compute resources that are located on a high performance computing cluster, a bunch of nodes in a small or medium sized company or big hyperscalers it is necessary to "manage" these resources. This management is done by so-called *"schedulers"* or *"orchestrators"* (which do more than simply scheduling).

Managing these resources means to take care that a task/batch/job/service is started on the needed resources, it runs for a specified time and can use its resources exclusively. Therefore the scheduler/orchestrator has to be aware of all compute resources in the system – free ones and those in use. To know which resources are in use means that every "job" that is handed over to the scheduler as to specify a certain number of parameters, e.g.

- #(CPUs)
- CPU types *(optional)*
- #(accelerators) – e.g. GPUs
- accelerator types *(optional)*
- #(RAM)
- #(nodes)
- node type or a queue to use
- available storage on the node(s) *(optional)*
- run-time
- network requirements (e.g. Ethernet or IB) *(optional)*
- unique name *(optional)*
- special features to fulfill - e.g. GPI support *(optional)*

With the help of these values the scheduler can put the job at a specific position in a queue. The higher the position within the queue the shorter the time till the job starts. If the needed resources are available the moment the job is scheduled it will start immediately, if not the job will get a priority and has to wait until the resources are free. At this point it is important to mention that after a defined time (to be configured) the scheduler will start to block resources so that the queued job will start in a reason time.

No matter which scheduler (on premise or in the cloud) one has to specify the resources that are needed for a job. If this specification is wrong or does not mach the real resources a job needs there will be undesired side effects, e.g. the job will die or run with a bad performance. This means that on the one side a user has to define the needs of his job as appropriate as possible, but on the other side the scheduler has to provide enough freedom to define the resources in such a way that it matches the needs of the users.

2 Orchestrator Independent Resource Description

Every orchestrator/scheduler that is available at the moment relies on its own and specific way to declare the resources – e.g. number of nodes, accelerator types – that a service needs to work.

As more and more companies and institutions use cloud based infrastructures and therefore have to define on such a platform how their service, app, CI/CD, etc. should use these resources. The "problem" is that such resource specifications are not general, but rather specific for each cloud platform available. As a consequence, users tend to stick to one cloud platform because utilizing another provider – e.g. to shortly increase your compute power – means rewriting the scheduling process. Such a rewrite can be time-consuming as it is likely that one has to modify not only the resource specifications but the source code of your application as well.

From a user point of view, this is really bad and creates a huge barrier to make use of various cloud platform at the same time. From the view of a cloud provider, this is perfect as one ties users to your product, and you increase your benefit.

In order to become more independent of the specific orchestrator/scheduler that is used on a platform it would be great to use a resource description that is universal and improves portability. One possibility is to use easy to parse formats like *JSON* or *XML* or even better you use an accepted and trusted "*standard*".

Talking about a "*standard*" there is at the moment one potential candidate to be in question which is *TOSCA* [4]. It is also mentioned in the "Gaia-X Architecture Document" [1] and in the "Gaia-X Architecture of Standards Document" [2].

2.1 What is TOSCA?

TOSCA – Topology and Orchestration Specification for Cloud Applications – is an open "standard" that defines the interoperable description of services and applications hosted on the cloud and elsewhere; including their components, relationships, dependencies, requirements, and capabilities, thereby enabling portability and automated management across cloud providers regardless of underlying platform or infrastructure; thus expanding customer choice, improving reliability, and reducing cost and time-to-value. These characteristics also facilitate the portable, continuous delivery of applications (DevOps) across their entire lifecycle. In short, they empower a much higher level of agility and accuracy for business in the cloud.[7]

It is worth to mention that the specification of the standard is driven by the global nonprofit consortium called *OASIS – Organization for the Advancement of Structured Information.* Standards. Among the members of OASIS are companies like Dell, IBM, Microsoft, Novell, Oracle, Red Hat but also universities, government agencies and individuals.

Even though there are big IT players associated to the standardization process and the actual implementation of the standard, the number of systems that make it possible to use TOSCA is – until today – rather small.[8] With respect to *Gaia-X 4KI* we can say that the only project relevant tools that provide a more or less direct TOSCA implementations are

- Cloudify (using a TOSCA-inspired dialect)
- OpenStack

2.2 How TOSCA works

A TOSCA profile consists of different templates. A Template is in that sense a set of parameterized constructors from which services and service components are created and includes (or reference) all necessary configuration and infrastructure requirements.[9]

TOSCA uses a "metamodel" to describe the base structure of a cloud oriented application. This "metamodel" aims to be standardized and machine-readable. The top level of this model is the so-called *service template* (FIG. 2). This *service template* consists of different sub-templates and definitions

- topology template
  - node template
  - relationship template
- node types
- relationship types
- policies

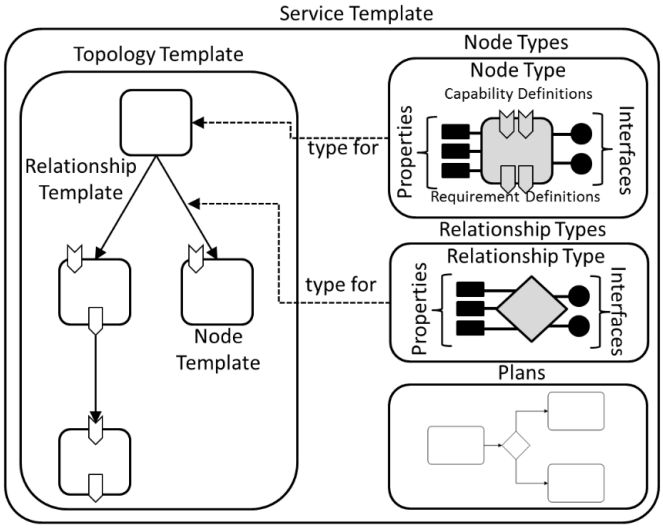

FIG. 2: *Schematic diagram of a TOSCA service template.[2]*

- management plans
    - set of workflows, tasks

The *topology template* is a directed graph mirroring the actual structure of the applications. This structure is defined by the *node template* and *relationship template* that consist of the needed nodes and the relation between these nodes. This set of dependencies and interactions are described in the *node types* and *relationship types* defining the needed resources e.g. CPUs, RAM, disks, network and the "interface" (entry points) of the applications. *Policies* can be used to define conditions or a specific set of actions that have to be done on the nodes – before, during or after the application runs. The *management plans* define how for example the application is started or stopped (including the respective order) or the different steps that have to be executed.

Each of these templates and definitions can be expressed in a XML or YAML inspired file. In the case of YAML, there is a specific implementation "defined" by OASIS called *TOSCA Simple Profile in YAML* [5]. This language implementation not only defines the grammar of the language, but also a set of common base types. This can be seen as a kind "library" that supports people when writing a TOSCA profile file.

2.3 Usability

The usability of TOSCA highly depends on the distribution of the standard. The more could platforms make use of this application description, the more schedulers and orchestrators will provide APIs to hand over TOSCA profiles. Right now, the majority of the schedulers – cloud or HPC – do not provide capabilities to start applications with a TOSCA profile.

But with the TOSCA YAML implementation and the focus on being machine-readable, it should be possible to write a parser that converts the TOSCA profile to something that state-of-the-art schedulers can work with. On the other side it is a point that has to be discussed, how reliable it is to depend on a standard that is not well established and not used by any of the big hyper-schedulers.

3 Deliverable E2.1.1: Schedulers Overview

TAB. 1: Batch-Job Schedulers. *Schedulers mostly used on "classical" - on premise - HPC clusters. Job - respec-tively batch job - centred workflow. Jobs run for a previously defined "walltime". The moment this "walltime" is over the job ends automatically. Failed compute resources are not replaced automatically.*

Name	**Website**	**Licence**	**Type**	**Packages**	**Containers**	**Accounting**	**special Features**	**Comment**
SLURM	docs, github	GPLv2	Scheduler, Orchestrator	deb, rpm, source	various see here	Yes	Different plugins available (e.g. containers, GPU support, MPI integration, etc.). Allows to write/add plugins on your own.	Business support. "Rootless containers"" can be scheduled via batch jobs (ref.). Widely used in the "classical" HPC world.
Altair PBS Professional	website	proprietary	Scheduler, Orchestrator	RedHat, SUSE, CentOS & Ubuntu	Docker (+ rootless containers via batch jobs)	Yes	EAL3+ Certified, SELinux integration	–

TAB. 1: Batch-Job Schedulers. *(continued from previous page)*

Name	**Website**	**Licence**	**Type**	**Packages**	**Containers**	**Accounting**	**special Features**	**Comment**
IBM Spectrum LSF	website	Proprietary	Scheduler, Orchestrator	tar.Z, msi	yes: Docker, Singularity, Shifter, Podman, Enroot	Yes	integrated UI, cloud–HPC scheduling, GPU support, business support	heavy features, good development, high costs
OpenPBS	website, github	GNU AFFERO GENERAL PUBLIC LICENSE	Scheduler, Orchestrator	rpm, deb, source	*partially* (no clear statements)	Yes	–	Scheduling rootless containers via batch jobs is possible but direct integration or plugins unclear.
TORQUE	website, github	TORQUE v2.5+ Software License (proprietary)	Scheduler, Orchestrator	rpm, source	*partially* (no clear statements)	yes	–	Scheduling rootless containers via batch jobs is possible but direct integration or plugins unclear.

TAB. 1: Batch-Job Schedulers. *(continued from previous page)*

Name	**Website**	**Licence**	**Type**	**Packages**	**Containers**	**Accounting**	**special Features**	**Comment**
HTCondor	website, docs	Apache License, Version 2.0	Scheduler	tar.gz, docker, k8s	docker (non-root)	No	hibernate worker, controlled job termination on shared worker	–

TAB. 2: Meta Schedulers. *These are schedulers that are not stand alone but need in addition some kind of additional scheduling/orchestrating tool in the background.*

Name	**Website**	**Licence**	**Type**	**Packages**	**Containers**	**Accounting**	**special Features**	**Comment**
krake	docs, gitlab	Apache License	meta scheduler	git clone	backend depended	No	Backends extendable via ApiDef, Scheduling by Metrics and Labels to Backends	k8s and magnum (OpenStack) as backend
Apache Mesos	website	Apache License 2.0	scheduler	rpm, source	Docker, AppC (via Marathon)	yes (ony role based model)	built-in Web UI	PaaS/IaaS targeted to run other schedulers. Interoperation with Spark & Kafka
Marathon	github	Apache License 2.0	meta scheduler	tar.gz file	Docker, AppC	not implemented (accounting depends on Mesos)	–	Plugin for Apache Mesos and DC/OS
Apache Hadoop YARN	website	Apache License 2.0	scheduler	tar.gz	Docker	No	–	Hadoop Ecosystem

TAB. 3: Cloud-Workflow Oriented Schedulers. *The majority of the schedulers here focus on workflows that are mostly used in cloud setups, e.g. CI/CD.*

Name	Website	Licence	Type	Packages	Containers	Accounting	special Features	Comment
Rancher	website, github	Apache License 2.0	container orchestration / meta scheduler	installation script	Docker, Podman	via plugin (kubecost)	Hybrid Cloud Setup, Marketplace with deployable features	Rancher 2 exclusively deploys and manages Kubernetes clusters running anywhere
Nomad	website	Mozilla Public License 2.0	container orchestration	Package Manager, Binary	Docker, Podman, Containerd, Windows IIS, Jailtask, Pot, Singularity, Firecracker, LXC	yes but only in enterprise version	different scheduling types integrated, heavy featured ecosystem (e.g. Gitlab + Jenkins integration, NVIDIA support,...)	Nomad vs. Kubernetes | "workload orchestrator for containers and non-containerized applications across on-prem and clouds" | Free vs. Enterprise Version

TAB. 3: Cloud-Workflow Oriented Schedulers. *(continued from previous page)*

Name	**Website**	**Licence**	**Type**	**Packages**	**Containers**	**Accounting**	**special Features**	**Comment**
Cloudify	website, github	Apache License 2.0	"multi-cloud orchestration platform": provisioning, scheduler etc.	rpm, Docker, deb (partially)	Docker (via Kubernetes)	not directly (possible via 3rd party tools/plugins)	Using one language to schedule your CI/CD on local clusters, hybrid setups and clouds	Similar to Rancher. Open source with business model. Set up not so easy.
Kubernetes	website, github	Creative Commons Attribution 4.0 International Public License	container orchestration engine	direct download, dep, rpm	Docker	No	All the resources in Kubernetes can be expressed as YAML files (easy git versioning). Very extensible - modify the resource definitions according to your needs.	Rather easy to scale up your application/CI/CD.
OpenShift	website	Apache-Lizenz 2.0	container orchestration	rpm	Docker, Podman	No	–	Kubernetes fork

TAB. 3: Cloud-Workflow Oriented Schedulers. *(continued from previous page)*

Name	**Website**	**Licence**	**Type**	**Packages**	**Containers**	**Accounting**	**special Features**	**Comment**
Cloud Foundry's Diego	docs	Apache License 2.0	Scheduler	git clone	gardener(-runC), (Docker)	yes	distinguish between jobs (Task and long-running processes)	diego execute Jobs in VMs, missing GPU Support(?)
CoreOS Fleet	github	–	–	–	–	–	–	deprecated in favor of k8s

TAB. 4: "Closed" Cloud Schedulers. *These schedulers are listed for completeness, but they work only in their respective closed sourced cloud ecosystems – Google Cloud, AWS, Azure.*

Name	**Website**	**Licence**	**Type**	**Packages**	**Containers**	**Accounting**	**special Features**	**Comment**
Google Kubernetes Engine	website	Apache-2.0 License	container orchestration engine	on demand in cloud service integrated	Docker	done via Google Cloud pricing model	list of all features	Rollout, orchestration and scaling of orchestrated applications in the google framework
Amazon ECS	website	Apache-2.0 License	container orchestration engine	on demand in cloud service integrated	Docker	done via AWS pricing model	–	specialized Environment for running Container in Amazon AWS, deploy on-premise with Amazon ECS Anywhere

TAB. 4: "Closed" Cloud Schedulers. *(continued from previous page)*

Name	**Website**	**Licence**	**Type**	**Packages**	**Containers**	**Accounting**	**special Features**	**Comment**
Azure Container Services	website	MIT License	Scheduler	on demand in cloud service integrated	Docker (Azure Kubernetes Service), Docker + Podman (Azure Red Hat OpenShift)	done via Azure Cloud pricing model	–	specialized Environment for running Container in Azure, on-premise with Azure Stack
Azure Kubernetes Service	website	unknown	Orchestrator, Scheduler	on demand in cloud service integrated	Docker	done via Azure Cloud pricing model	–	running K8s in prepared Linux VMs in Microsoft Azure, on-premise with Azure Stack
Azure Red Hat OpenShift	website	unknown	container orchestration	on demand in cloud service integrated	Docker, Podman	done via Azure Cloud pricing model	–	–

3.1 Summary E2.1.1

Based on the broad evaluation of existing scheduling and orchestration tools in V1, we suggest the following most promising candidates (order as listed before) for V2.

- The combination Krake with SLURM, K8 and OpenStack
- Rancher with SLURM, K8 and OpenStack
- Nomad with SLURM, K8 and OpenStack

The second most promising schedulers - second because of either being proprietary software or having minor issues - are

- IBM Spectrum LSF
- Apache Mesos + Marathon
- Cloudify

References

[1] Gaia-X Technical Committee, Gaia-X Work Packages, Gaia-X Working Group Architecture, Gaia-X Working Group Federation Services / Open Source Software, Gaia-X Working Group Portfolio, Gaia-X Working Group Provider, Gaia-X Working Group User Gaia-X Working Group X-Assocation. *Gaia-X Architecture Document*. Gaia-X European Association for Data and Cloud AISBL. 10/2021. URL: https://www.gaia-x.eu/sites/default/files/2021-10/Gaia-X_Architecture_Document_2109.pdf.

[2] Gaia-X European Association for Data and Cloud AISBL. *Gaia-X Architecture of Standards*. (internal document). 11/2021.

[3] Gaia-X European Association for Data and Cloud AISBL. *Software Requirements Specification for Gaia-X Federation Services – Integration & Portal Orchestration*. eco – Association of the Internet Industry. 2021. URL: https://www.gxfs.de/wp-content/uploads/2021/06/SRS_GXFS_IP_ORC.pdf.

[4] OASIS Standard. *Topology and Orchestration Specification for Cloud Applications Version 1.0*. 11/2013. URL: https://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html.

[5] Matt Rutkowski, Chris Lauwers, Claude Noshpitz, and Calin Curescu. *TOSCA Simple Profile in YAML Version 1.3*. OASIS Standard. 26/02/2020. URL: https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.3/TOSCA-Simple-Profile-YAML-v1.3.html.

[6] Bundesamt für Sicherheit in der Informationstechnik. *APP.4.4 Kubernetes*. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/Drafts/Final_Draft/APP_4_4_Kubernetes_Edition_2022.pdf?__blob=publicationFile&v=1. (accessed: 08/12/2021).

[7] TOSCA TC. *FAQ*. URL: https://www.oasis-open.org/committees/tosca/faq.php. (accessed: 03/12/2021).

[8] TOSCA Wiki. *Known TOSCA Implementations*. URL: https://wiki.oasis-open.org/tosca/TOSCA-implementations. (accessed: 03/12/2021).

[9] UBICITY. *TOSCA Modeling Overview*. 02/2018. URL: https://www.oasis-open.org/committees/download.php/62645/%20Overview%202018%2002%2026.pdf.