Melodic
Big data cloud

Title:

Resource Management Framework Prototype

Multi-cloud Execution-ware for Large-scale Optimised Data-Intensive Computing

H2020-ICT-2016-2017
Leadership in Enabling and Industrial Technologies; Information and Communication Technologies

Grant Agreement No.:
731664

Duration:
1 December 2016 -
30 November 2019

www.melodic.cloud

Deliverable reference:
D4.3

Date:
15 December 2018

Responsible partner:
UULM

Editor(s):
Daniel Baur

Author(s):
Daniel Baur, Daniel Seybold

Approved by:
Ernst Gunnar Gran

ISBN number:
N/A

Document URL:
http://www.melodic.cloud/deliverables/D4.3 Resource Management Framework Prototype.pdf

Executive summary:

This deliverable presents the prototype of the resource management layer of the Executionware. The resource management layer has the task to i) propose possible deployment options to the Melodic Upperware, ii) allocate, manage and release the selected resources, as well as iii) monitor them.

This document gives an overview of the implementation (code) of the prototype, describes its functionality, documentation and integration.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Document	
Period Covered	M8-20
Deliverable No.	D4.3
Deliverable Title	Resource management framework prototype
Editor(s)	Daniel Baur
Author(s)	Daniel Baur, Daniel Seybold
Reviewer(s)	Sebastian Schork, Feroz Zahid
Work Package No.	4
Work Package Title	Executionware
Lead Beneficiary	Ulm University
Distribution	PU
Version	1.0
Draft/Final	Final
Total No. of Pages	12

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Table of Contents

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

1 Introduction

The purpose of this deliverable is to describe the resource management layer prototype of the Executionware. The main purpose of the resource management layer is to i) provide to the Melodic Upperware a set of node candidates from which the Reasoner of the Upperware can select a fitting set, ii) allocate the resources selected by the Upperware from different cloud providers using a provider-agnostic interface (see Deliverable 4.1 [1]) and iii) monitor the usage of these resources.

To achieve this functionality, the Executionware relies on the Cloudiator framework[1], meaning that all functionality of the resource management layer was implemented within the Cloudiator framework. As the following sections of this document are implementation-driven, they will therefore discuss the implementation effort within Cloudiator.

Figure 1 gives an overview of the complete architecture of Cloudiator. The resource management layer discussed in this deliverable concerns the resource management entity that is responsible for generating the node advertisements passed to Melodic's Upperware (Resource Broker and Discovery Agent). In addition, it contains the Node Agent responsible for allocating nodes by either calling the VM Agent for IaaS resources or the PaaS Agent for PaaS resources. Finally, the monitoring entity is responsible for monitoring the usage of all resources acquired by Cloudiator. The functionality of the different entities and agents is discussed in detail in Deliverable 4.1 [1].

[1] http://cloudiator.org

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

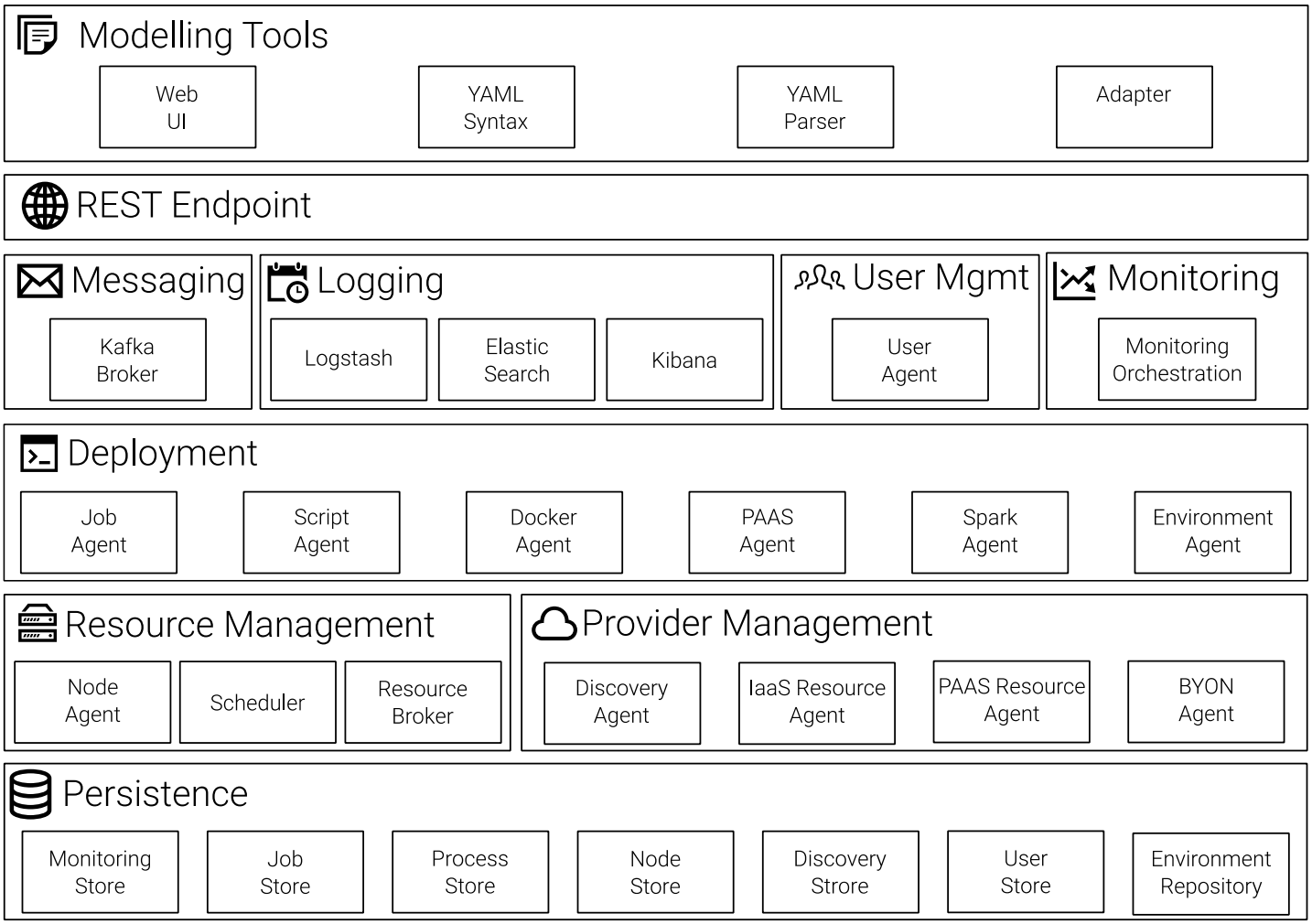

Figure 1: Cloudiator Architecture Overview

1.1 Scope of the document

This document is intended for the general audience that is interested in the development process of the resource management layer prototype and which capabilities it provides. The work of this deliverable mainly depends on the "Provider agnostic interface definition & mapping cycle" described in deliverable D4.1 [1].

1.2 Structure of the document

The following sections are structured as followed: First, Section 2 describes the functionality covered by the prototype. Afterwards, Section 3 gives an overview of the code that was implemented to achieve this functionality. Section 4 provides documentation sources helping to use the prototype, whereas Section 5 will present the integration of Cloudiator. Finally, Section 6 summarizes the content of this deliverable.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

2 Functionality

This section discusses the functionality that is contained in the prototype.

2.1 Provider-agnostic interface definition and mapping

As described in Deliverable D4.1 [1], Cloudiator provides a provider-agnostic interface definition and maps it to the different cloud providers. The functionality of the prototype includes all functionality described in this deliverable.

2.2 Resource Management Prototype

The resource management layer prototype provides the following functionality: i) it automatically discovers available offers of IaaS providers, ii) it includes the node advertisement logic that generates possible node candidates from the discovered offers and passes them to the Upperware, and iii) it is capable of allocating the selected resources across the cloud providers supported by the provider-agnostic interface. Again, a more detailed description of this functionality can be found in deliverable D4.1 [1].

Current limitations of the prototype are that the resources can currently be allocated, but not released. While the API for this purpose is already designed, the logic is currently missing. Additionally, the monitoring orchestration is currently only capable to install the sensors statically when the resource is allocated. Changing the monitoring demand with respect to the application and the acquired resources during runtime is currently not supported, but will be implemented in a future release.

3 Implementation

This section discusses the code implementing the aforementioned functionality. It gives an overview of the license used, the main third-party dependencies Cloudiator has and the structure of the code repositories where Cloudiator's code resides.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

3.1 License

All developed code is released in the public domain under the Apache License 2.0[2].

3.2 Main dependencies

Cloudiator's resource management layer relies on multiple open source projects. An overview of all the main dependencies can be found in Table 1 below.

Name	Description	Link
Apache jclouds	Multi-cloud library that provides a common Java-based API to multiple cloud providers (e.g. EC2, Google Compute Cloud, Openstack).	https://jclouds.apache.org/
Apache Kafka	Apache Kafka is a distributed streaming platform; in our case, it is used as a messaging system for the communication between the individual micro-services of the Executionware.	https://kafka.apache.org/
Docker	Docker is a container engine, making it easier to build, run and ship applications. The Executionware uses it for two main purposes: a) the Executionware itself is distributed using Docker and b) the Executionware optionally uses it to deploy the users' applications.	https://www.docker.com/
Apache Maven	Apache Maven is a project dependency management software for Java. It is used for	https://maven.apache.org/

[2] https://www.apache.org/licenses/LICENSE-2.0.html

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

	resolving dependencies and building Java artifacts.	
MariaDB	MariaDB is a relational database that is used for persisting the state of multiple components.	https://mariadb.org/
Hibernate	Hibernate is an object relational mapper for Java. It is used to abstract the database access logic from an individual database.	http://hibernate.org/
Protocol Buffers	Protocol Buffers is a minimal message format that is used for communication over the Kafka message bus.	https://developers.google.com/protocol-buffers/
Etcd	Etcd is a key-value store used for synchronizing the execution state of multiple deployed applications.	https://github.com/coreos/etcd
Swagger	Swagger is a toolset for describing REST-APIs using the OpenAPI[3] standard. This toolset includes an editor as well as code generators.	https://swagger.io/
Spring (Boot)	The REST endpoint is implemented using Spring Boot. Basic model objects and interfaces are automatically generated using Swagger.	https://spring.io/projects/spring-boot
Travis CI	Travis CI provides continuous integration as a service.	https://travis-ci.org/
Eclipse Modelling Framework	The Eclipse Modelling Framework (EMF) is used to provide the Object Constraint Language	https://www.eclipse.org/modeling/emf/

[3] https://www.openapis.org/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

	implementation required in the matchmaking agent.	
ELK Stack	The ELK (Elasticsearch, Logstash and Kibana) stack is used for distributed logging across all services of Cloudiator.	https://www.elastic.co/elk-stack
Google Guice	A dependency injection library. Used in all services of Cloudiator for dependency management.	https://github.com/google/guice

Table 1: Main dependencies of Cloudiator

3.3 Source Code Repositories

All source code of Cloudiator is hosted on Github under the organization Cloudiator[4]. Table 2 gives an overview of the individual repositories and their content.

Repository	Description
rest-server	Hosts the REST endpoints of Cloudiator. A spring boot application that is partially automatically generated from the Swagger API specification.
rest-swagger	Hosts the Swagger documentation of Cloudiator's REST API. Can be accessed in a more readable way under cloudiator.org/rest-swagger
common	Common utility functionality required by all Cloudiator components, e.g., the messaging logic or basic persistence entities.
java-rest-client	Automatically generated client for the REST endpoint of Cloudiator. It is for instance used by the Upperware to access Cloudiator's logic.

[4] http://github.com/cloudiator

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

visor-java-rest-client	Java client for the REST endpoint of the Visor monitoring agent. Automatically generated. Used internally to communicate with the monitoring agent.
matchmaking	Hosts the matchmaking logic of the resource manager. Responsible for offering possible node candidates to Melodic's Upperware. Relies on the OCL implementation of EMF.
docker	Hosts the docker-compose file to install all components of Cloudiator with one command.
orchestration	Hosts all orchestration components of Cloudiator, which are responsible for allocating resources from cloud providers.
pom	The Maven Project Object Model (POM) for all Cloudiator components. Ensures that the same version is used for common dependencies.
sword	The multi-cloud abstraction layer of Cloudiator. A java library that provides a common interface to all supported cloud providers.
lance	The deployment agent of Cloudiator. Lance is installed on all resources managed by Cloudiator and is required for managing the lifecycle (e.g. install, start, stop) of the user's application.
monitoring	The monitoring orchestration engine of Cloudiator. Responsible for installing the monitoring agent on all managed resources and to launch required sensors based on the monitoring demand of the user.
management	Contains management services of Cloudiator, responsible for handling users and their authentication/authorization. Additionally, provides an encryption service for, e.g., encrypting cloud credentials and a secure store for variables in deployment scripts.
cloudiator.github.io	The source code for the Cloudiator website.
deployment	Contains the deployment services of Cloudiator.
visor	Visor is the Cloudiator's monitoring agent.

Table 2: Source code repositories

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

4 Documentation

All documentation can be found on the Cloudiator website (http://cloudiator.org).

4.1 Installation and Packaging

To ease the installation process, Cloudiator uses Docker for each of its components. In combination with docker-compose, this allows to deploy all components using a single command. The docker-compose description and usage instructions can be found on GitHub at: https://github.com/cloudiator/docker.

Currently, the prototype is not optimized with respect to resource usage, meaning that a machine with at least 16GB of RAM is required to run all agents of the Cloudiator framework. However, the Cloudiator roadmap includes performance optimization as an import target for a future release.

4.2 Usage

The Cloudiator website (http://cloudiator.org) provides an introduction into Cloudiator's usage and a step-by-step tutorial using the popular Mediawiki[5] as an example.

5 Integration

Cloudiator uses a fully-featured integration process as depicted in Figure 2. Each developer of Cloudiator has its own development environment. Using git[6] as a version control system, every developer can push his/her changes to the code repository hosted on GitHub. These pushes will automatically trigger a new build on the Travis CI continuous integration platform. On successful builds, Travis CI will automatically publish build artefacts to i) Docker Hub which we use for distribution of binaries (run as Docker containers), ii) the Central Maven repository which we use for dependency management for Java projects, and iii) sonarcloud[7] which we use for automated bug and vulnerability detection.

[5] https://www.mediawiki.org/wiki/MediaWiki
[6] https://git-scm.com/
[7] https://sonarcloud.io

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

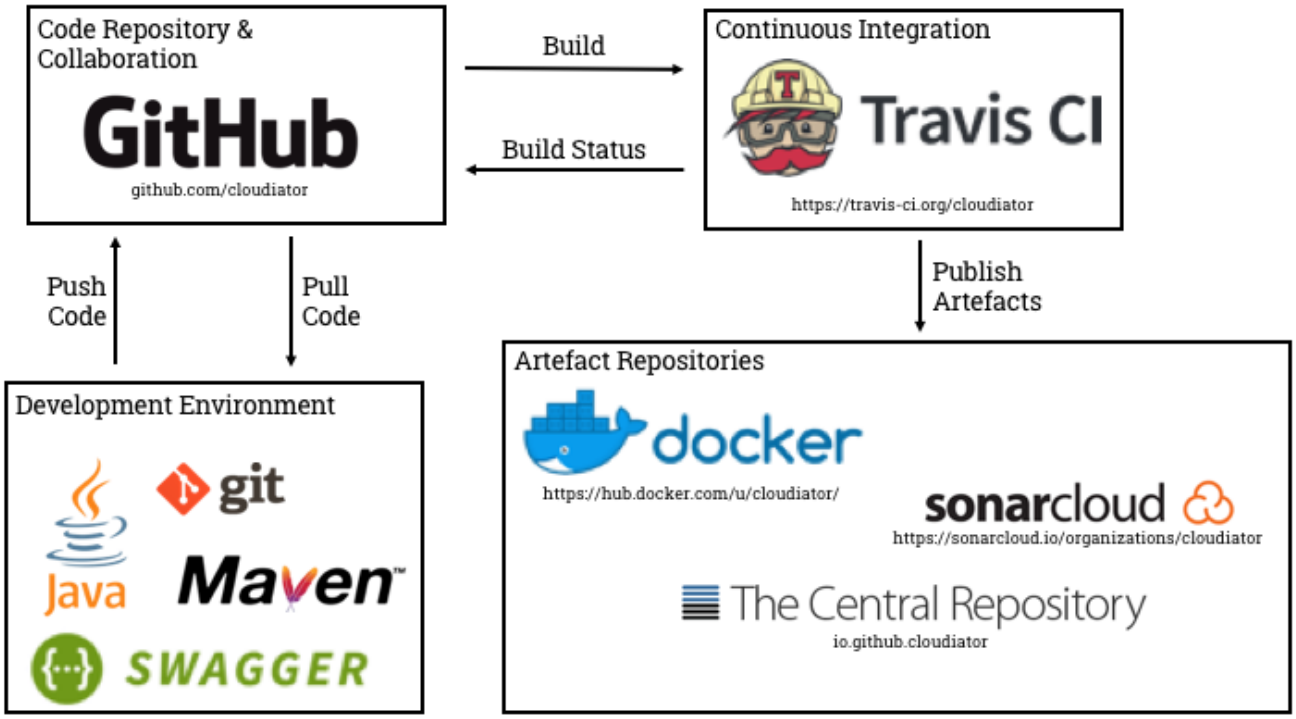

Figure 2: Integration tools and process

6 Summary

In this deliverable, we have described the functionality of the resource management layer prototype, which within the Melodic project is responsible for allocating and managing the resources from a multi-cloud environment. The upcoming deliverable D4.4 Data Processing Layer Prototype will rely on this work and describe the deployment and orchestration logic that is executed on top of this prototype. With the upcoming release of Melodic (2.0), both the currently missing functionality (see Section 2) will be implemented, as well as the data processing layer prototype will be available.

7 References

[1] D. Baur and D. Seybold, "D4.1 Provider agnostic interface definition & mapping cycle," Melodic Project Deliverable.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664