Master System Design

TVUK System Team

Version v0.3, dd/mm/yyyy

Table of Contents

Design Document

Master System Design for the Common Architecture

COMMENTS and ISSUES

If you would like to raise comments or issues on this document, please do so by raising an Issue at the following URL https://github.com/EOEPCA/master-system-design/issues.

PDF

This document is available in PDF format here.

COPYRIGHT

1. Introduction

1.1. Purpose and Scope

This document presents the Master System Design for the Common Architecture.

1.2. Structure of the Document

TBD

1.3. Reference Documents

The following is a list of Reference Documents with a direct bearing on the content of this document.

Reference	Document Details	Version
[EOEPCA-UC]	EOEPCA - Use Case Analysis (https://eoepca.github.io/use-case-analysis), EOEPCA.TN.005	Issue TBD, dd/mm/yyy
[EP-FM]	Exploitation Platform - Functional Model, ESA-EOPSDP-TN-17-050	Issue 1.0, 30/11/2017
[TEP-OA]	Thematic Exploitation Platform Open Architecture, EMSS-EOPS-TN-17-002	Issue 1, 12/12/2017
[WPS-T]	Cauchy, A.: OpenGIS® WPS2.0 Transactional Extension Implementation Standard, Open Geospatial Consortium, OGC 13-071r2, http://www.opengeospatial.org/docs/discussion-papers (2013)	13-071r2
[CWL]	Common Workflow Language Specifications, https://www.commonwl.org/v1.0/	v1.0.2
[TB13-AP]	OGC Testbed-13, EP Application Package Engineering Report, OGC 17-023, http://docs.opengeospatial.org/per/17-023.html	17-023, 30/01/2018
[TB13-ADES]	OGC Testbed-13, Application Deployment and Execution Service Engineering Report, OGC 17-024, http://docs.opengeospatial.org/per/17-024.html	17-024, 11/01/2018
[TB14-AP]	OGC Testbed-14, Application Package Engineering Report, OGC 18-049r1, http://docs.opengeospatial.org/per/18-049r1.html	18-049r1, 07/02/2019

Reference	Document Details	Version
[TB14-ADES]	OGC Testbed-14, ADES & EMS Results and Best Practices Engineering Report, OGC 18-050r1, http://docs.opengeospatial.org/per/18-050r1.html	18-050r1, 08/02/2019

1.4. Terminology

The following terms are used in the Master System Design.

Term	Meaning
Admin	User with administrative capability on the EP
Algorithm	A self-contained set of operations to be performed, typically to achieve a desired data manipulation. The algorithm must be implemented (codified) for deployment and execution on the platform.
Analysis Result	The *Products* produced as output of an *Interactive Application* analysis session.
Analytics	A set of activities aimed to discover, interpret and communicate meaningful patters within the data. Analytics considered here are performed manually (or in a semi-automatic way) on-line with the aid of *Interactive Applications*.
Application Artefact	The 'software' component that provides the execution unit of the *Application Package*.
Application Deployment and Execution Service (ADES)	WPS-T (REST/JSON) service that incorporates the Docker execution engine, and is responsible for the execution of the processing service (as a WPS request) within the ‘target’ Exploitation Platform.
Application Descriptor	A file that provides the metadata part of the *Application Package*. Provides all the metadata required to accommodate the processor within the WPS service and make it available for execution.
Application Package	A platform independent and self-contained representation of a software item, providing executable, metadata and dependencies such that it can be deployed to and executed within an Exploitation Platform. Comprises the *Application Descriptor* and the *Application Artefact*.
Authentication Provider	An encapsulation of Authentication provision within the Exploitation Platform context. The Authentication Provider is an Authorization Server (OAuth) that is used purely for authentication.
Bulk Processing	Execution of a *Processing Service* on large amounts of data specified by AOI and TOI.
Code	The codification of an algorithm performed with a given programming language - compiled to Software or directly executed (interpretted) within the platform.
Consumer	User accessing existing services/products within the EP. Consumers may be scientific/research or commercial, and may or may not be experts of the domain

Term	Meaning
Development	The act of building new products/services/applications to be exposed within the platform and made available for users to conduct exploitation activities. Development may be performed inside or outside of the platform. If performed outside, an integration activity will be required to accommodate the developed service so that it is exposed within the platform.
Discovery	User finds products/services of interest to them based upon search criteria.
Execution	The act to start a *Processing Service* or an *Interactive Application*.
Execution Management Service (EMS)	The EMS is responsible for the orchestration of workflows, including the possibility of steps running on other (remote) platforms, and the on-demand deployment of processors to local/remote ADES as required.
Expert	User developing and integrating added-value to the EP (Scientific Researcher or Service Developer)
External Application	An application or script that is developed and executed outside of the Exploitation Platform, but is able to use the data/services of the EP via a programmatic interface (API).
Guest	An unregistered User or an unauthenticated Consumer with limited access to the EP’s services
Identity Provider (IdP)	The source for validating user identity in a federated identity system, (user authentication as a service).
Interactive Application	A stand-alone application provided within the exploitation platform for on-line hosted processing. Provides an interactive interface through which the user is able to conduct their analysis of the data, producing *Analysis Results* as output. Interactive Applications include at least the following types: console application, web application (rich browser interface), remote desktop to a hosted VM.
Interactive Console Application	A simple *Interactive Application* for analysis in which a console interface to a platform-hosted terminal is provided to the user. The console interface can be provided through the user’s browser session or through a remote SSH connection.
Interactive Remote Desktop	An Interactive Application for analysis provided as a remote desktop session to an OS-session (or directly to a 'native' application) on the exploitation platform. The user will have access to a number of applications within the hosted OS. The remote desktop session is provided through the user’s web browser.
Interactive Web Application	An Interactive Application for analysis provided as a rich user interface through the user’s web browser.
Kubernetes (K8s)	Container orchestration system for automating application deployment, scaling and management.
On-demand Processing Service	A *Processing Service* whose execution is initiated directly by the user on an ad-hoc basis.
Platform (EP)	An on-line collection of products, services and tools for exploitation of EO data

Term	Meaning
Processing	A set of pre-defined activities that interact to achieve a result. For the exploitation platform, comprises on-line processing to derive data products from input data, conducted by a hosted processing service execution.
Processing Result	The *Products* produced as output of a *Processing Service* execution.
Processing Service	A non-interactive data processing that has a well-defined set of input data types, input parameterisation, producing *Processing Results* with a well-defined output data type.
Products	EO data (commercial and non-commercial) and Value-added products and made available through the EP. *It is assumed that the Hosting Environment for the EP makes available an existing supply of EO Data*
Reusable Research Object	An encapsulation of some research/analysis that describes all aspects required to reproduce the analysis, including data used, processing performed etc.
Scientific Researcher	Expert user with the objective to perform scientific research. Having minimal IT knowledge with no desire to acquire it, they want the effort for the translation of their algorithm into a service/product to be minimised by the platform.
Service Developer	Expert user with the objective to provide a performing, stable and reliable service/product. Having deeper IT knowledge or a willingness to acquire it, they require deeper access to the platform IT functionalities for optimisation of their algorithm.
Software	The compilation of code into a binary program to be executed within the platform on-line computing environment.
Systematic Processing Service	A *Processing Service* whose execution is initiated automatically (on behalf of a user), either according to a schedule (routine) or triggered by an event (e.g. arrival of new data).
Terms & Conditions (T&Cs)	The obligations that the user agrees to abide by in regard of usage of products/services of the platform. T&Cs are set by the provider of each product/service.
Transactional Web Processing Service (WPS-T)	Transactional extension to WPS that allows adhoc deployment / undeployment of user-provided processors.
User	An individual using the EP, of any type (Admin/Consumer/Expert/Guest)
Value-added products	Products generated from processing services of the EP (or external processing) and made available through the EP. This includes products uploaded to the EP by users and published for collaborative consumption
Visualisation	To obtain a visual representation of any data/products held within the platform - presented to the user within their web browser session.
Web Coverage Service (WCS)	OGC standard that provides an open specification for sharing raster datasets on the web.
Web Feature Service (WFS)	OGC standard that makes geographic feature data (vector geospatial datasets) available on the web.

Term	Meaning
Web Map Service (WMS)	OGC standard that provides a simple HTTP interface for requesting geo-registered map images from one or more distributed geospatial databases.
Web Processing Services (WPS)	OGC standard that defines how a client can request the execution of a process, and how the output from the process is handled.
Workspace	A user-scoped 'container' in the EP, in which each user maintains their own links to resources (products and services) that have been collected by a user during their usage of the EP. The workspace acts as the hub for a user’s exploitation activities within the EP

1.5. Glossary

The following acronyms and abbreviations have been used in this report.

Term	Definition
AAI	Authentication & Authorisation Infrastructure
ABAC	Attribute Based Access Control
ADES	Application Deployment and Execution Service
AOI	Area of Interest
API	Application Programming Interface
CMS	Content Management System
CWL	Common Workflow Language
EMS	Execution Management Service
EO	Earth Observation
EP	Exploitation Platform
FUSE	Filesystem in Userspace
IAM	Identity and Access Management
IdP	Identity Provider
JSON	JavaScript Object Notation
K8s	Kubernetes
M2M	Machine-to-machine
OGC	Open Geospatial Consortium
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIP	Policy Information Point
RBAC	Role Based Access Control
REST	Representational State Transfer
SSH	Secure Shell
TOI	Time of Interest

Term	Definition
VNC	Virtual Network Computing
WCS	Web Coverage Service
WFS	Web Feature Service
WMS	Web Map Service
WPS	Web Processing Service
WPS-T	Transactional Web Processing Service
XACML	eXtensible Access Control Markup Language

2. Context

The Master System Design provides an EO Exploitation Platform architecture that meets the service needs of current and future systems, as defined by the use cases described in [EOEPCA-UC]. These use cases must be explored under 'real world' conditions by engagement with existing deployments, initiatives, user groups, stakeholders and sponsors within the user community and within overlapping communities, in order to gain a fully representative understanding of the functional requirements.

The system design takes into consideration existing precursor architectures (such as the Exploitation Platform Functional Model [EP-FM]] and Thematic Exploitation Platform Open Architecture [TEP-OA]), including consideration of state-of-the-art technologies and approaches used by current related projects. The master system design describes functional blocks linked together by agreed standardised interfaces.

The importance of the OGC in these activities is recognised as a reference for the appropriate standards and in providing mechanisms to develop and evolve standards as required in the development of the architecture. In order to meet the design challenges we must apply the applicable existing OGC standards to the full set of federated use cases in order to expose deficiencies and identify needed evolution of the standards. Standards are equally important in all areas of the Exploitation Platform, including topics such as Authentication & Authorisation Infrastructure (AAI), containerisation and provisioning of virtual cloud resources to ensure portability of compute between different providers of resource layer.

Data and metadata are fundamental considerations for the creation of an architecture in order to ensure full semantic interoperability between services. In this regard, data modelling and the consideration of data standards are critical activities.

The system design must go beyond the provision of a standalone EO Exploitation Platform, by intrinsically supporting federation of similar EO platforms at appropriate levels of the service stack. The Network of EO Resources seeks, ‘to unite the available - but scattered - European resources in a large federated and open environment’. In such a context, federation provides the potential to greatly enhance the utilization of data and services and provide as stimulus for research and commercial exploitation. From the end-user point of view, the federated system should present itself as a single consolidated environment in which all the federated resources are made available as an integrated system. Thus, the system design must specify federation-level interfaces that support this data and service-level interoperability in such a way that is seamless to the end users.

The goal is to create an Integrated Data Exploitation Environment. Users will apply their workflows close to the hosted data, supplemented by their own data. Processing outputs may be hosted as new products that can themselves contribute to the global catalogue. This paradigm can then be extended to encompass the federated set of Exploitation Platforms within the Network of EO Resources. The result is a Federated, Integrated Data Analysis Environment.

A Reference Implementation of the full architecture will be developed to prove the concepts and also to provide an off-the-shelf solution that can be instantiated by future projects to implement their EO Exploitation Platform, thus facilitating their ability to join the federated Network of EO Resources. **Thus, the Reference Implementation can be regarded as a set of re-usable platform**

services, in the context of a re-usable platform architecture.

2.1. Domain Areas

The system architecture is designed to meet the use cases as defined in [EOEPCA-UC] and [EP-FM]. [EOEPCA-UC] makes a high-level analysis of the use-cases to identify the main system functionalities organised into domain areas: 'User Management', 'Processing & Chaining' and 'Resource Management'. The high-level functionalities are often met by more than one domain area, and User Management (specifically Identification, Authentication and Authorisation) cuts across all use cases, and forms the basis of all access control restrictions that are applied to platform services and data.

User Management

All aspects of user identification, authentication and authorisation in a federated system-ofsystems environment.

Processing and Chaining

Hosting and maintaining an inventory of all processing tasks, analysis tools and interactive applications. Handles and abstracts the low-level complexities of the different underlying compute technologies, and ensures the compute layer is scaled in accordance with current demand. Provides an integrated development environment to facilitate development of new processing algorithms and applications. Facilitating the network of EO resources by providing a federated interface to other processing services within the wider EO network.

Resource Provisioning and Management

Storage and cataloguing of all persistent resources, including data and other supporting material such as documents. Handles and abstracts the low-level complexities of different underlying storage technologies and strategies. Facilitating the network of EO resources by providing a federated interface to other data services within the wider EO network.

3. Design Overview

The overall system design has been considered by taking the ‘Exploitation Platform – Functional Model’ [EP-FM] as a starting point and then evolving these ideas in the context of existing interface standards (with some emphasis on the OGC protocol suite) and the need for federated services.

Figure 1 depicts the domain areas as top level component blocks in a Platform ‘A’. The arrows may be read as “uses”, each implying one or more service interfaces.

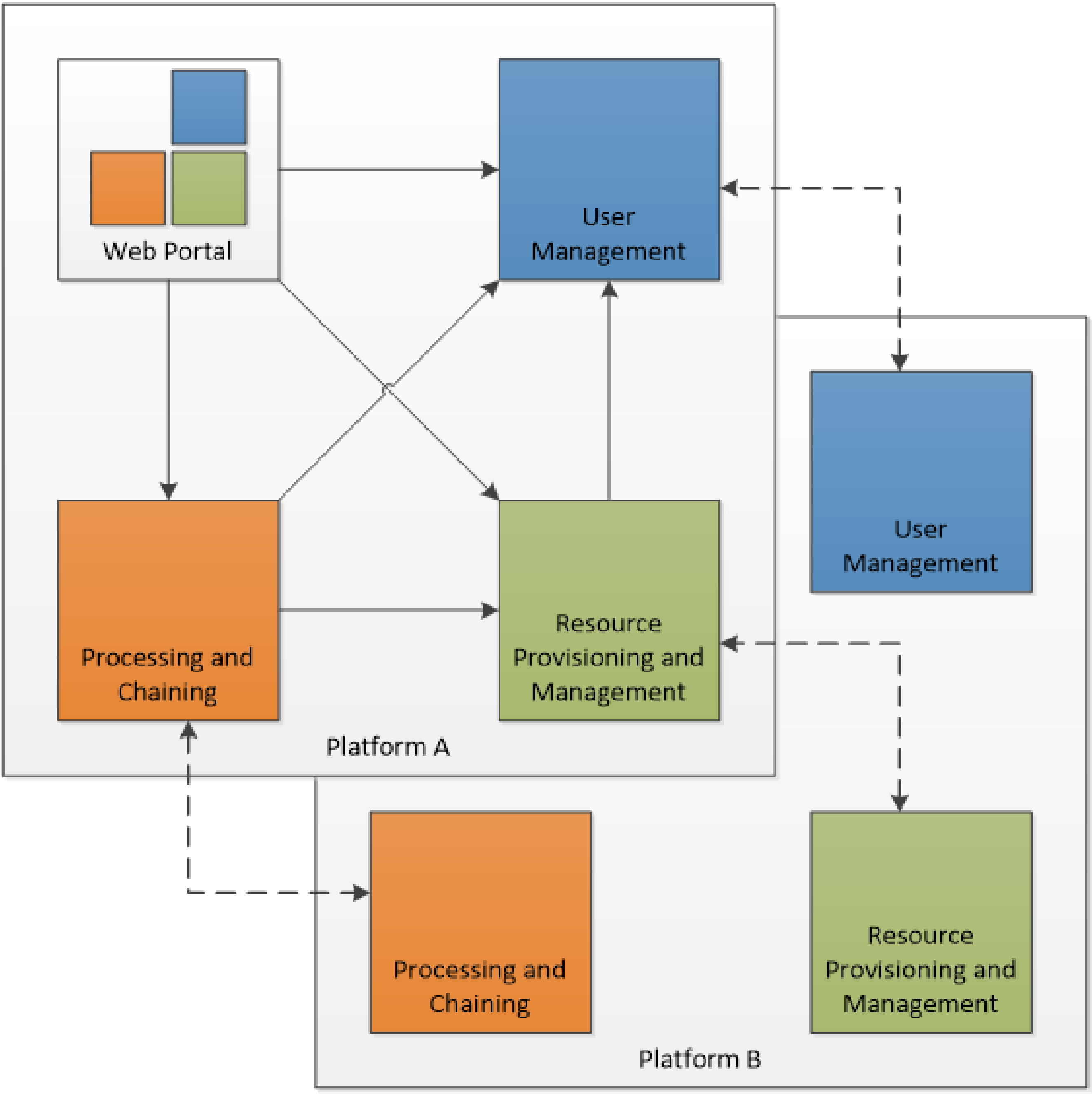

Figure 1. Top-level Architecture

A potential federation concept is represented by interactions between corresponding blocks in a collaborating Platform ‘B’. The architecture aims to minimise dependencies and is conducive to the principle of subcontracting the implementation to experts in the respective domains. The web portal integrates various client components to form a rich user-facing interface. **The Web Portal is depicted as it has interfaces with the other domain areas - but it is not a priority concern for**

the Common Architecture. Each exploitation platform would be expected to develop its own web interfaces according to its needs.

3.1. User Management

Responsible for all aspects related to the user's account, profile and identity.

It provides authentication, authorisation and accounting services that are required to access other elements of the architecture, including the processing-oriented services and resource-oriented services. Individual resources may have an associated access and/or charging policy, but these have to be checked against the identity of the user. Resource consumption may also be controlled e.g. by credits and/or quotas associated with the user account. In the Network of EO Resources, a user should not need to create an account on multiple platforms. Therefore some interactions will be required between the User Management functions, whether directly or in directly via trusted third party.

3.2. Processing and Chaining

Provides access to a variety of processing functions, tools and applications, as well as an executions environment in which to deploy them.

The user can have a private workspace in which to upload files, and develop their own processing chains, experiments and workflows. Integrated with this environment is the facility to use the services of the Resource Provisioning and Management subsystem to discover and access relevant published datasets and processing elements. Then, subject to appropriate controls and permissions, the user can publish their own processing services and results. Workflows can be executed within the context of the processing facility, with the possibility to execute steps 'remotely' in collaborating platforms, with the results being collected for the continuation of the workflow.

3.3. Resource Provisioning and Management

Responsible for storing and cataloguing a variety of resources.

First and foremost, this will contain multidimensional geo-spatial datasets. In addition it may include a variety of heterogeneous data and other resources, such as documentation, processing services, Docker images, etc.

The catalogue holds corresponding metadata for every published resource item in the local platform storage, as well as entries for resources that are located on remote collaborating platforms. Catalogue search and data access is provided through a range of standard interfaces, which are used by the local Web Portal and Processing & Chaining elements and may be exposed externally as web services. Access to services and resources is controlled according to an associated policy. For identity authentication, authorisation, and accounting, this will require the services of the User Management component. Similarly, the ingestion process is controlled, in order to ensure the quality of any published resource, including associated metadata, and to maintain the integrity of the catalogue. This component may interact with corresponding peer components on other platforms, for example to synchronise catalogue entries.

3.4. Web Portal

Presents the platform user interface for interacting with the local resources and processing facilities, as well as the wider network of EO resources.

It includes a web site, themed and branded according to the owning organisation. It provides the view onto the user’s private workspace and acts as their window on the world. It provides a rich, interactive web interface for discovering and working with all kinds of resources, including EO data, processing and documentation. It includes web service clients for smart search and data visualisations. It provides a workspace for developing and deploying processing algorithms, workflows, experiments and applications, and publishing results. It includes support and collaboration tools for the community.

Web Portal integrates together various web service clients that uses services provided by the specialist domains (Processing, Resource, User) on the local platform and collaborating platforms.

4. User Management

In the context of the Common Architecture, User Management covers the following main functional areas:

Identity and Access Management (IAM)

Identification/authentication of users and authorization of access to protected resources (data/services) within the EP.

Accounting and Billing

Maintaining an accounting record of all user accesses to data/services/applications, supported by appropriate systems of credits and billing.

User Profile

Maintenance of details associated to the user that may be needed in support of access management and billing.

These are explored in the following sub-sections.

4.1. Identity and Access Management (IAM)

The goal of IAM is to uniquely identify the user and limit access to protected resources to those having suitable access rights. We assume an Attribute Based Access Control (ABAC) approach in which authorisation decisions are made based upon attributes required by resources and possessed (as claims) by users. ABAC is is seen as a more flexible approach than Role Base Access Control (RBAC), affording the ability to express more sofisticated authorisations rules beyond the role(s) of the user - and noting the fact that a role-based ruleset could be implemented within an attribute based approach, (i.e. RBAC is a subset/specialisation of ABAC).

In achieving this there are three main concerns:

- Unique user identification
- Determine what attributes are required to access the protected resource
- Determine whether the user has the required attributes

For the Common Architecture, we establish separation of User Identification from Access Management. User identity is federated and handled external to the platform. Within the Network of EO Resources, resources held within an exploitation platform are made available to federated partner platforms, under the policy decision of the local platform:

- The identity is provided externally. The external IdP has no association to the exploitation platform, and hence is not the appropriate place to administer attributes that relate to EP resources.
- The protected resources are under the custodianship of the exploitation platform and hence the exploitation platform defines internally its rules for accessing its protected resources.
- The administrative domain for a set of attributes should not be tied to an exploitation platform, which facilitates the provision of federation and virtual organisations.

4.1.1. IAM Approach

Figure 2 presents the basic approach. At this stage it does not consider the case in which an exploitation platform accesses resources in another platform on behalf of a user, (for example a workflow step that is invoked on another platform). This is addressed in a later section. Users are authenticated by redirection to an external identity provider, (their ‘home’ IdP). This returns the authentication decision and some basic user information as required (such as name, email, etc.).

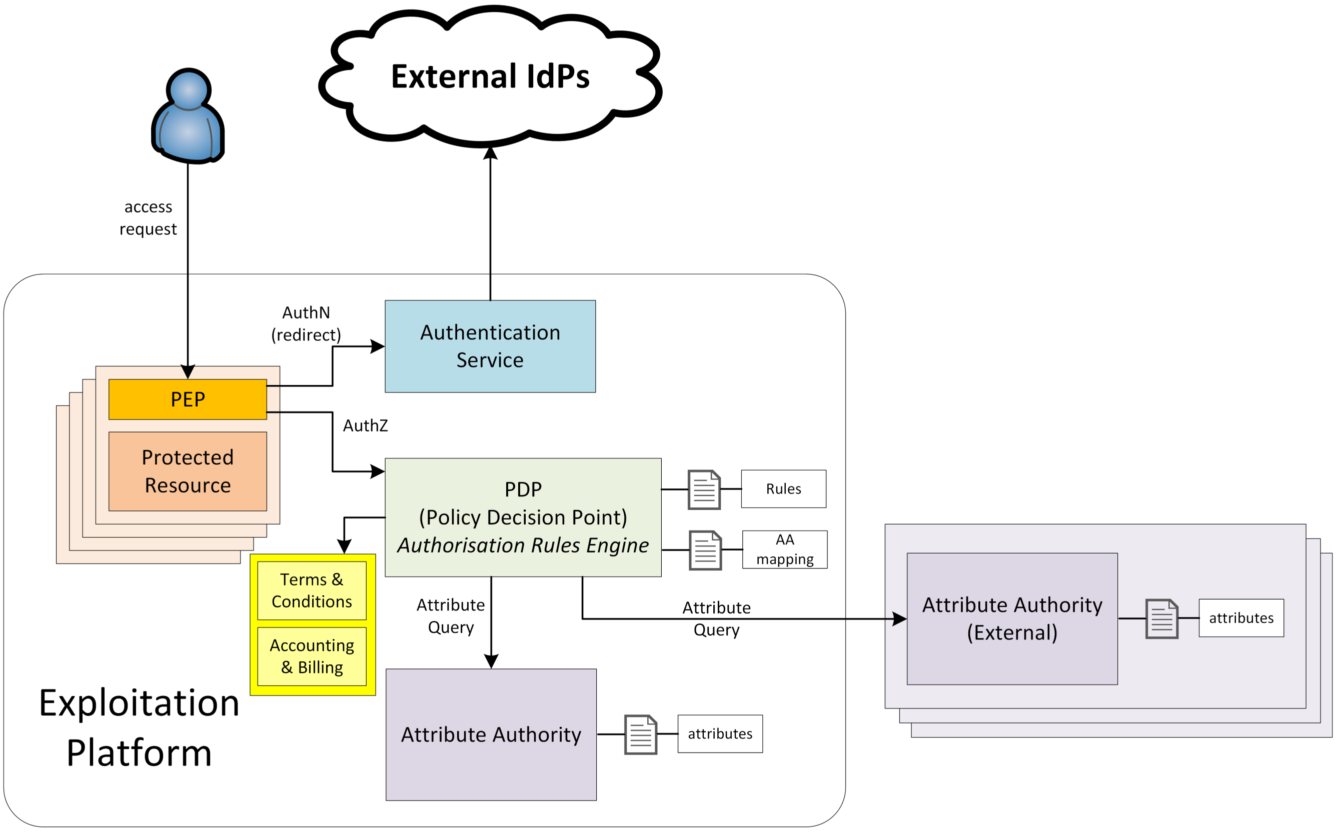

Figure 2. Identity and Access Management Overview

Each protected resource is fronted by its Policy Enforcement Point (PEP) that will only permit access if the appropriate conditions are met. This decision is made according to a set of rules that are under the control of and configured within the exploitation platform.

The Authentication Service is provided as a common component that is utilised by each PEP to perform the authentication flow with the external IdPs. In the case of an unauthenticated request, the PEP will initiate the Authentication Service by redirection of the User’s originating request. The successful flow ultimately redirects back to the PEP and so maintains the direct connection between the end-user agent and the resource server. This direct connection is of particular importance in the case of significant data volumes being returned to the User. An alternative approach would be the use of an API Gateway to perform the role of the PEP, acting as an intermediary between the end-user agent and the resource server. However, this would have the effect of proxying the connection and thus spoiling the direct communication between end-user agent and resource server, and so introduce an inefficiency in the data path.

The PEP interrogates the PDP for an authorisation decision. The PEP sends a request that indicates the pertinent details of the attempted access, including:

- Identity of end-user (XACML subject)

- The API (path/version etc.) being accessed (XACML resource)
- The operation (HTTP verb) being performed (XACML action)

The rules are expressed through attributes, and it is the job of the Policy Decision Point (PDP) to determine, for a given request, what attributes are required and what attributes the user possesses, in order to provide its access decision. In order to know whether a user possesses a given attribute it is necessary to interrogate the appropriate Attribute Authority for the attribute.

It should be additionally noted that the decision to allow the user access depends upon dynamic 'attributes', such as whether the user has enough credits to 'pay' for their usage, or whether they have accepted the necessary Terms & Conditions for a given dataset or service. Thus, the PDP must interrogate other EP-services such as 'Accounting & Billing' and 'User Profile' to answer such questions.

An Attribute Authority represents the administrative domain for a set of attributes - and acts as Policy Information Point (PIP) for the attributes under its governance. A given attribute is administered by a single Attribute Authority. Thus, when making its decision, the PDP must know for each attribute who is the responsible Attribute Authority (e.g. by lookup), and then interrogate that Attribute Authority to know whether the user possesses the given attribute. For any given attribute, the attribute authority can be one within the EP or administered externally.

Federated access and Virtual Organisations can be effected by use of ‘Federated Attributes’ (see below) that allow common attributes to be delegated under the administrative domain of a nominated Attribute Authority. Thus, for a given attribute there is a single authoritative endpoint for associated attribute queries. The exploitation platform supports the federation/VO by using the federated-attributes in its rules, and defers to the appropriate attribute authority when making policy decisions.

From the perspective of a given EP, two classes of attribute result from the above:

Local Attributes

Attributes that are used only internally by the EP, in which case the attributes are mapped to the local Attribute Authority.

Federated Attributes

Attributes that are used to facilitate federated access to resources and in the establishment of Virtual Organisations. In this case the EP maps the attributes to the appropriate Attribute Authority – noting the fact that it could be the local EP if it happens to be the administrative domain for the attribute.

Figure 3 provides an overview of the IAM Flow.

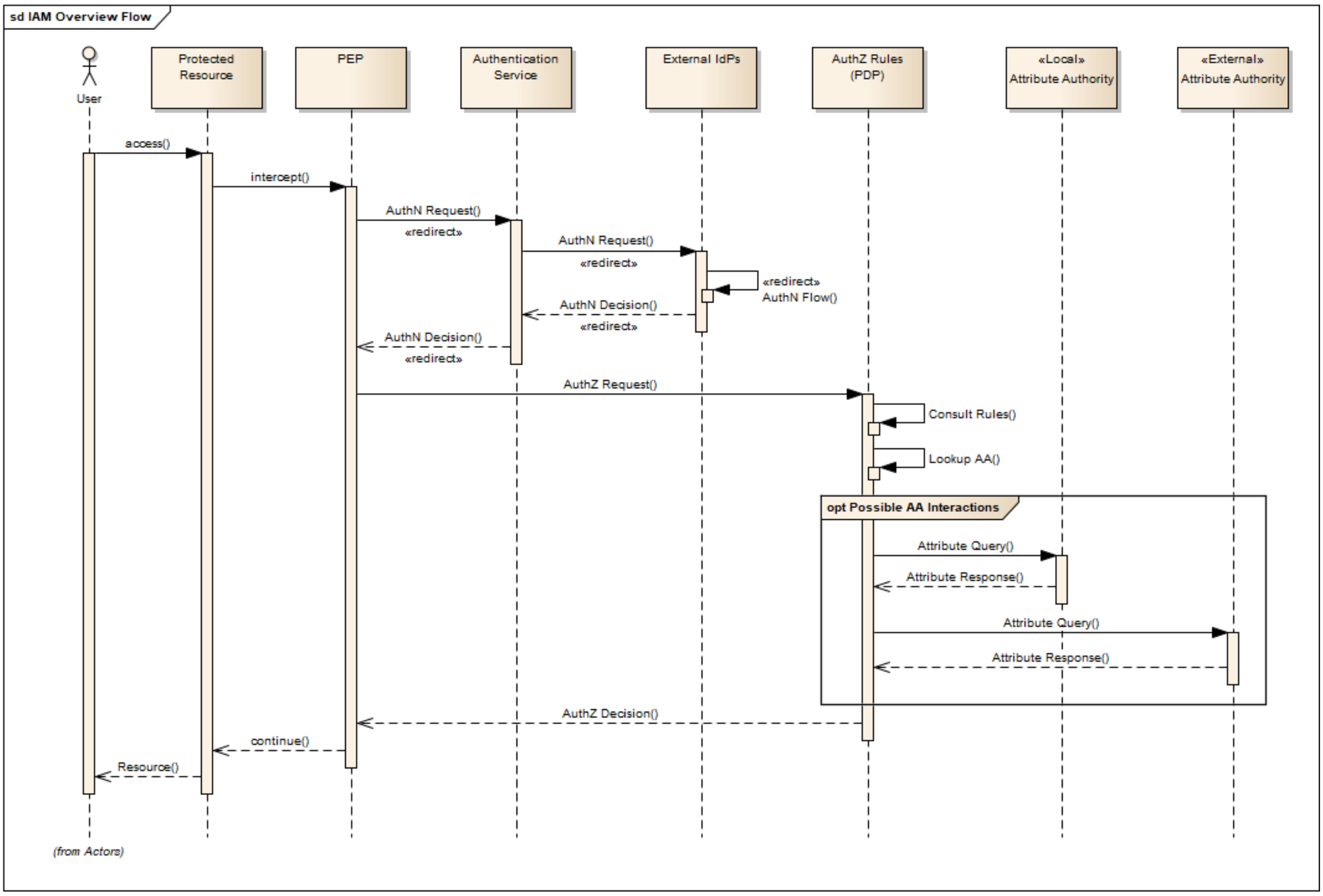

Figure 3. IAM Overview Flow

Note that the interface between the Authentication Service and the External IdPs is simplified in this view. It is expanded in later sections.

4.1.2. IAM Top-level Interfaces

Figure 4 illustrates the interfaces of the IAM architecture.

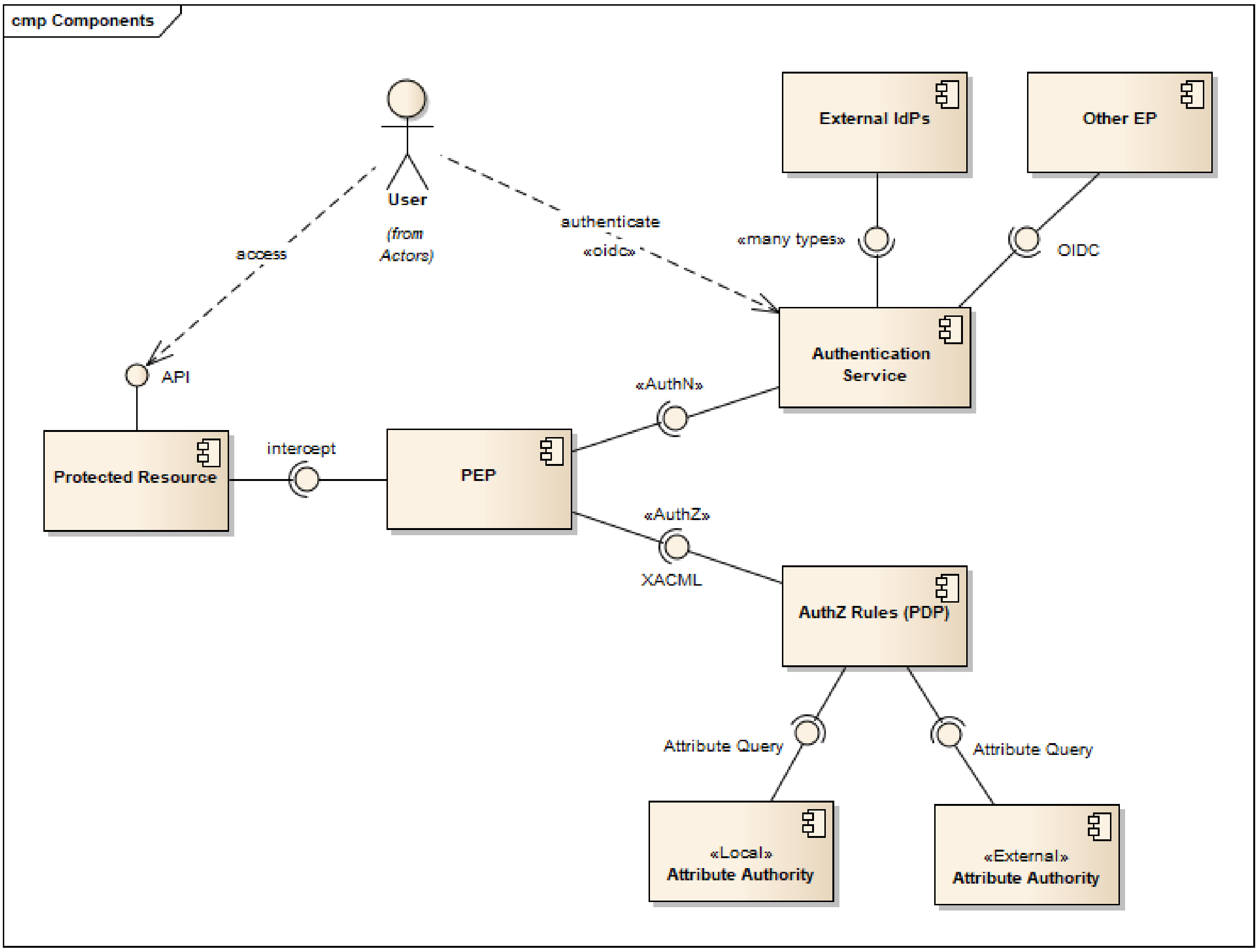

Figure 4. IAM Interfaces

User → Protected Resource

The Protected Resource exposes a public API for user consumption.

Protected Resource → PEP

The PEP is implemented either as an in-process component of the Protected Resource, or as an out-of-process shim. Either way, the PEP intercepts the incoming request in order to enforce the authorisation policy decision.

PEP → Authentication Service

The PEP uses a redirect to delegate the authentication flow to the Authentication Service.

Authentication Service → External IdP

In order to support multiple external identity suppliers, the Authentication Service must act as a client to multiple external IdPs, and so must establish individual trust relationships with each of these. Alternatively, the Authentication Service can instead interface to a single external IdP Proxy, that interfaces to the external IdPs on behalf of the EP. The IdP Proxy can provide this service to multiple EPs.

PEP → AuthZ Rules Engine (PDP)

Possible use of XACML requests for this interface.

PDP → Attribute Authority

Possible use of SAML attribute queries for this interface.

4.2. Authenticated Identity

The approach to user identity and authentication centres around the use of OpenID Connect. Each Exploitation Platform maintains their own OIDC Provider through which tokens can be issued to permit access to protected resources within the EP. The authentication itself is delegated to external Identity Providers at the preference of the end-user wishing to reuse their existing identity provision.

4.2.1. Overview

The Authentication Service is an OpenID Connect Provider that provides a ‘Login With’ service that allows the platform to support multiple external identity providers.

The Authentication Service presents an OIDC Provider interface to its clients, through which the OIDC clients can obtain Access Tokens to resources. The access tokens are presented by the clients in their requests to resource servers (intercepted by PEP). The PEP (acting on behalf of the resource server) relies upon the access token to establish the authenticated identity of the users making the requests. Once the user identity is established, then the PEP can continue with its policy decision (deferred to the PDP).

Thus, clients of the EP must act as OIDC Clients in order to authenticate their users to the platform, before invoking its services. Clients include the web applications that provide the UI of the exploitation platform, as well as other external applications/systems (including other exploitation platforms) wishing to use the services of the EP.

The Authentication Service must act as client to each of the External IdPs to be supported and offered as a ‘Login With’ option. The interface/flow with the External IdP is integrated into the OIDC flow implemented by the Authentication Service. This includes prompting the user to discover their ‘home’ Identity provider. The interactions with the external IdP represents the ‘user authentication step’ within the OIDC flows. Completion of a successful authentication with the external IdP allows the Authentication Service to issue the requested access tokens (depending on the flow used).

Figure 5 illustrates the basic user access flow, invoked through a web browser.

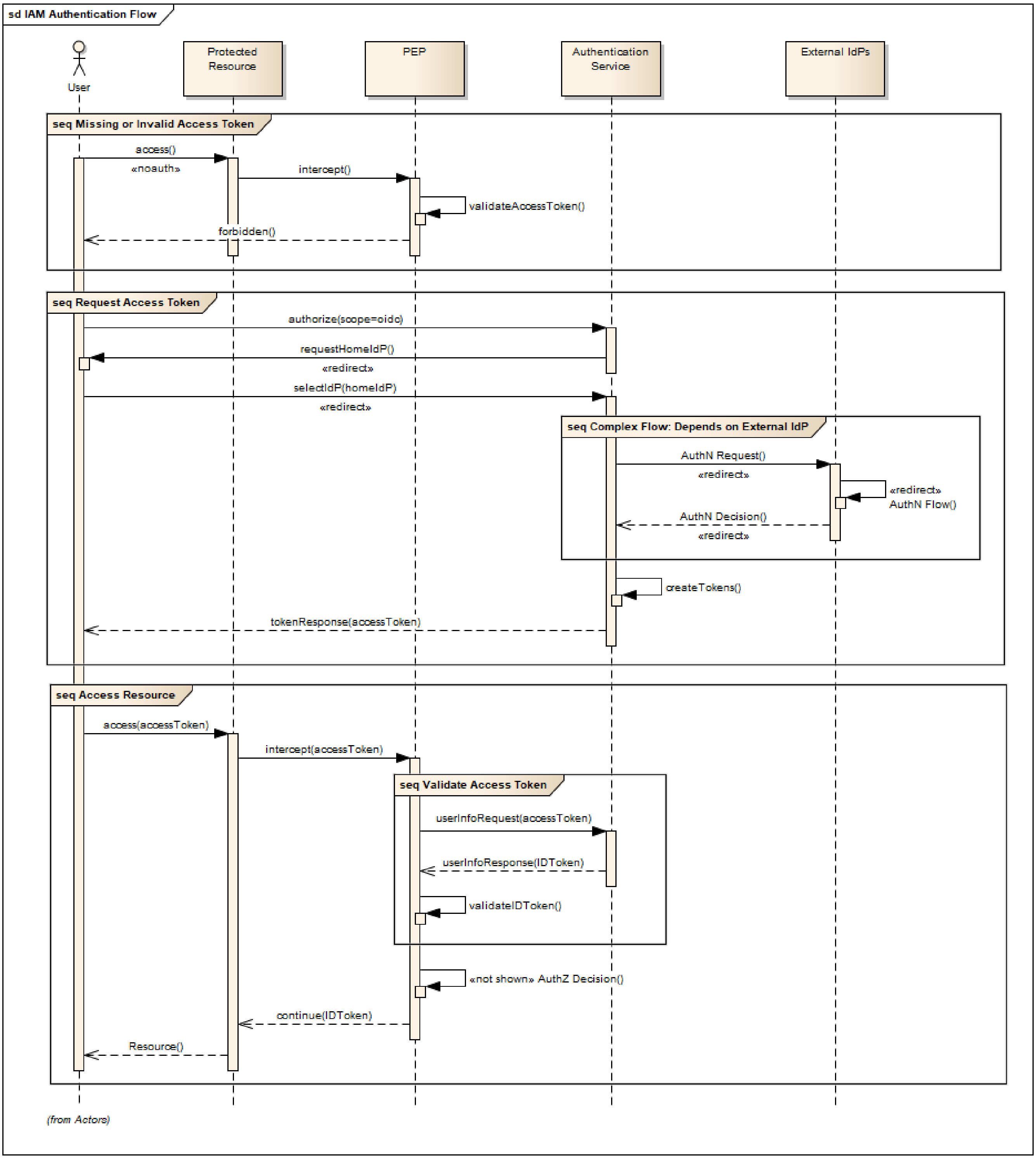

Figure 5. IAM Authentication Flow (Browser)

4.2.2. Authentication Service

The Authentication Provider is an OIDC Provider that provides a ‘Login With’ service that allows the end-user to select their Identity Provider for purposes of authentication.

The Authentication Provider is designed to support the onward forwarding of the authentication request through external identify services, which should be expected to include:

- EduGain
- GitHub

- Google
- Twitter
- Facebook
- LinkedIn
- Others TBD

The Authentication Provider must establish itself as a client of all supported external IdPs, with appropriate trust relationships and support for their authentication flows.

The primary endpoints required to support the OIDC flows are as follows (these endpoints are taken, by example, from OKTA OIDC discovery metadata, https://micah.okta.com/oauth2/aus2yrcz7aMrmDAKZ1t7/.well-known/openid-configuration):

authorization_endpoint (/authorize)

To initiate the authentication, and to return the access tokens / code grant (depending on flow).

token_endpoint (/token)

To exchange the code grant for the access tokens.

userinfo_endpoint (/userinfo)

To obtain the user information ID token in accordance with the scopes requested in the authorisation request.

jwks_uri (/keys)

To obtain signing keys for Token validation purposes.

end_session_endpoint (/logout)

To logout the user from the Authentication Provider, i.e. clear session cookies etc. Although, given that the actual IdP is externalised from the Authentication Provider, it would remain the case that any session cookies maintained by the external IdP would still be in place for a future authentication flow.

introspection_endpoint (/introspect)

Used by clients to verify access tokens.

revocation_endpoint (/revoke)

Used for (refresh) token revocation.

As described in section ‘Discovery’, the following endpoints relate to Discovery:

OIDC Discovery (/.well-known/openid-configuration)

Dynamic discovery of OIDC endpoints by clients.

As described in section ‘Client Registration’, the following endpoints relate to Dynamic Client Registration:

registration_endpoint (/clients)

Dynamic registration of clients (Authentication Agents).

As described in section ‘Federation’, the following endpoints relate to the establishment of a federation of collaborating Exploitation Platforms through a dynamic trust model:

/.well-known/openid-federation

OIDC Federation API endpoint through which Entity Statements are published about itself and other entities (such as other Exploitation Platforms). See section ‘Federation’.

4.2.3. OIDC ID Token

The ID Token is a JWT that is returned to from the /userinfo endpoint of the Authentication Service. The returned OIDC ID Token has been signed (JWS) by the Authentication Service and thus results in a token that asserts a user’s authenticated identity with integrity, and non-repudiation.

4.2.4. OIDC Clients

Clients act on behalf of users accessing the services of the Exploitation Platform. They will either pre-emptively obtain their access token for required resources, or will attempt resource access and be redirected by exception to the OIDC Provider authentication flow.

In the case of a web application (browser hosted), the Implicit Flow would be used. In other cases (TBD) the Authorisation Code Flow would be preferred.

The OIDC flows are initiated with the appropriate response_type (‘id_token token’ for Implicit Flow, ‘code’ for Authorisation Code Flow) and scope of ‘oidc profile’.

At the successful conclusion of the flow the client receives the Access Token and ID Token. The Access Token is then used by the client as a Bearer token in its subsequent calls to access the EP resources.

4.2.5. Resource Server (PEP)

The PEP (acting on behalf of the resource server) receives the client request to access the protected resource, with the expectation that the request includes a valid access token.

Thus, the PEP follows the logic:

- If the access token is not present then no user is logged in, so the request should be redirected to the /authorize endpoint (HTTP redirect)
- If the access token is present, then it should be validated with the Authentication Service (direct call), as described below
- If the access token validation completes successfully then the request can continue (pending authorisation), with the user identity provided by the ID Token received during token validation
- If the token is invalid, then the request should be redirected to the /authorize endpoint (HTTP redirect)

4.2.6. Access Token Validation

The PEP validates the access token by using it as a Bearer token in a request to the Authentication Service's /userinfo endpoint. A successful response has two outcomes:

- Confirms the validity of the access token from the point-of-view of the Authentication Service that issued it
- Provides an ID Token for the user that provides the information required to uniquely identify the user within the EP and utilise this identity within the subsequent policy decision made by the PDP

The ID Token is a JWT that has been signed by the Authentication Service. Using the jwks (see section 'OIDC Federation') endpoint of the Authentication Service, the PEP is able to obtain the necessary keys to validate the signature of the ID Token. This provides the full user context for the resource access.

4.2.7. Federated User Access

Based upon the above authentication model, an EP could access the resources of another EP by obtaining an access token through OIDC flows. However, considering that these EP→EP invocations will typically be Machine-to-machine (M2M), then we need to consider how the end-user (resource owner) is able to compete their consent. Two possibilities are explored in the subsequent sections:

1. The user pre-authorises the EP→EP access in advance of the operation
2. Use of OIDC JWKS for trusted federation of identity between platforms

User Pre-authorisation

Using the facilities of the Exploitation Platform, the user (perhaps via their User Profile management console) initiates the authorisation flow from one EP to another. The end result is that the originating EP obtains delegated access to another EP on behalf of the user - with the resulting access tokens being maintained within the user's profile on the EP.

At the point where the EP needs to access a resource on another EP, then the access tokens are obtained from the user's profile and used as Bearer token in the resource request to the other EP. Refresh tokens can be used to ensure that authorisation is long-lived.

Conversly, the user's profile at a given EP should also provide the ability to manage any inward authrosations they have granted to other EPs, i.e. ability to revoke a previous authorisation by invalidating the refresh token. This would invole interface with the Authentication Service.

Possible use of OIDC JWKS Federation

OIDC provides a distributed key-hierarchy that could be used to support federated user access between collaborating exploitation platforms. The concept is explored in this section.

Reference: https://openid.net/specs/openid-connect-federation-1_0.html

OIDC provides a framework in which RPs and OPs can dynamically establish verifiable trust chains, and so share keys to support signing and validation of JWTs.

Dedicated ‘federation’ endpoints are defined that allow an entity (such as RP or OP) to publish their own Entity Statements, and to obtain Statements for other entities that are issued by trusted third-parties within the federation. The metadata/signatures within the Entity Statements establish a chain of trust that can be followed to known (trusted) Trust Anchors, and so the Entity Statements and the included entity public keys can be trusted.

Thus, through this mechanism public keys can be shared to underpin the signing and validation of JWTs.

Within an EP, when a resource server is executing a user’s request, it may need to invoke a resource in another EP with which it is collaborating. The resource access to the other EP must be made on behalf of the originating user.

The nominal solution is for the originating EP to act as an OIDC Client to interface with the Authentication Service of the other EP, and so obtain the access token required to access the other resource. In this case, we should consider the fact that the resource access may be asynchronous to the end-user request and is not made within the context of the end-user’s user agent. Therefore, we should explore possibilities (flows) provided by OIDC/OAuth that support this type of access.

One possibility is to make use of the signed-JWT ID Token that can be carried through the calls into and across resource servers. Through the facilities provided by JSON Web Key Set (JWKS), ID Tokens can be verified and trusted by other platforms operating within the same JWKS key hierarchy.

Thus, using the trusted ID Token, it may be possible follow an OIDC/OAuth flow from one EP to another, in which the user is deemed to have a-priori authorised the third-party access. At this point it is only the user’s identity that has been established, with the authorisation decision subject to the rules of the PDP/PEP of the remote system. The identified user must have appropriate a-priori permissions (attributes) on the target resources to be granted access, (ref. ‘Federated Attributes’).

Thus, it is the ID of the user that has been passed machine-to-machine to facilitate the service federation. This effectively achieves cross-EP single sign-on, without relying upon the user agent of the end-user providing cookies to the other EP.

4.2.8. Additional OIDC Capabilities

OpenID Connect provides some additional functionalities that are of interest in the context of the Common Architecture.

OIDC Discovery

Reference: https://openid.net/specs/openid-connect-discovery-1_0.html

OpenID Connect makes provision for two types of discovery:

1. Discovery of the OpenID Provider Issuer based upon the user’s identifier
2. Discovery of the OpenID Provider Configuration Information

In the case of our usage within the Exploitation Platform, type 1) is not application since the user's ID comes from their 'Home' organisation and is not (necessarily) tied to an OpenID Connect Provider. Instead the Authentication Service must implement a discovery 'flow' in which the user is able to select the provider of their identity, as one that is supported by the Authentication Service deployment.

Regarding discovery type 2), the Authentication Service exposes an OIDC Provider interface, and this should support retrieval of OIDC Provider Configuration Information. Thus, OIDC Clients can utilise the discovery interface of the Authentication Service to exploit its services.

This is of most interest in the case of access to federated resources in other EPs, where a resource server in one EP may be acting as an OIDC client of the Authentication Service in another EP – in which case auto-discovery might be more attractive.

Client Registration

Reference: https://openid.net/specs/openid-connect-registration-1_0.html

The possibility exists for the OIDC Client (Relying Party) to perform auto-registration with the Authentication Service, using OIDC Client Registration. In doing so the OIDC client obtains its Client ID and Secret.

This may be of interest in a couple of cases:

- The case of access to federated resources in other EPs, where a resource server in one EP may be acting as an OIDC client of the Authentication Service in another EP – in which case auto-client-registration might be of interest.
- The case where a common Authentication Service is deployed outside of the context of a given Exploitation Platform, acting as an IdP Proxy. In this case, the local Authentication Service deployed in each EP would register as an OIDC Client of the IdP Proxy.

4.2.9. Authorization (Policy Decision)

TBD

Policy Decision Point (PDP)

TBD

Attribute Authority

TBD

4.3. Accounting and Billing

TBD

4.4. User Profile

The User Profile is a system resource that maintains a set of data for each user including:

- User details
- Terms and conditions accepted by the user
- License keys held by the user
- User API key management

The User Profile for a given user is tied to the unique identifier provided by their Home-IdP through the authentication process.

5. Processing and Chaining

The Processing & Chaining domain area must provide an extensible repository of processing functions, tools and applications (referred here generically as ‘processing services’) that can be discovered by search query, invoked individually, and utilised in workflows.

Processing services are published in an Application Catalogue that acts as a Marketplace and facilitates their discovery. Via the Marketplace users have a single point of access to all processing services that are published across the federated system. In order to invoke processing services and workflows, users must specify the data inputs and parameterisation.

Users must be able to define and execute workflows that chain processing steps, in which the input(s) of a step are provided by the output of preceding step(s). Users can publish workflows as new processing services, and so the possibility of workflow nesting.

A workflow comprises multiple steps (processing service invocations), each of which can be executed on the platform that is closest to the data. Thus, the workflow must be orchestrated to invoke the steps on the appropriate platform and stage in/out the data between platforms along the execution pipeline. Thus, processing services should be relocatable between federated EO platforms, such that they can be deployed and instantiated for execution ‘close to the data’. This implies that applications are packaged in a way that is self-contained, standardised and agnostic of the underlying hosting technology.

Users must be able to develop and integrate their own processing services into the platform. Once integrated the user can publish their processing service so that it is discoverable by search query and available in the federated marketplace – and hence available for exploitation by other users, including use in workflows. In support of this, an integrated development environment should be provided that allows users to develop, test and debug their applications before submission.

The interface between the Processing Framework and the compute resource should be abstract so that the solution is not tied to any particular provider (cloud, DIAS, etc.).

In meeting these requirements the following key challenges are identified:

- Processing and data interoperability must be established through clear and consistent metadata definitions, to ensure that type mismatches are avoided. This is particularly challenging across federated systems where it becomes more difficult to enforce use of common profiles and vocabularies
- Defining an application packaging approach whose paradigm is easy to work with, whilst providing a rich environment that allows expert users to exploit the full compute capability of the platform
- Federation of processing services across the network of EO resources, such that processing implementations can be made available ‘on-demand’ amongst federated platforms, to facilitate the movement of the “processing to the data”. Use of a common packaging format that is agnostic of underlying host characteristics is key to this challenge
- Enforcement of access controls to processing and data resources through multi-step federated workflows requires the user’s ‘request context’ to be carried through all layers of the request fulfilment. At each point of resource access, the user’s identify and access rights must be

asserted. The service interface standards, (such as WPS, CSW and WCS), must be evaluated and necessary enhancements identified to ensure that the user's access envelope is respected

5.1. Solution Overview

The Processing & Chaining solution is based upon the work performed in the OGC Testbeds, described by the following Engineering Reports:

- OGC 17-023 - OGC Testbed-13, EP Application Package ER [TB13-AP]
- OGC 17-024 - OGC Testbed-13, Application Deployment and Execution Service ER [TB13-ADES]
- OGC 18-049r1 – OGC Testbed-14, Application Package Engineering Report [TB14-AP]
- OGC 18-050r1 - ADES & EMS Results and Best Practices Engineering Report [TB14-ADES]

Additionally, the current OGC Testbed-15 Thread-2 Earth Observation Process and Application Discovery (EOPAD).

Processing-services are packaged as Docker images, which can then be deployed as self-contained applications within the Exploitation Platform's processing framework. OGC-WPS provides a standard interface to expose all processing services (and workflows) for invocation by WPS clients.

Each processing service is described by an Application Descriptor, which is a file that accompanies its deployment to the processing framework of the EP. The Application Descriptor provides all the metadata required to accommodate the processor within the WPS service and make it available for execution.

The architecture is defined by the following main components:

Execution Management Service (EMS)

WPS-T (REST/JSON) service that provides an umbrella orchestration service to deploy/invoke processing services within the ADES of the appropriate (close to data) Exploitation Platform. Thus, the EMS is responsible for the orchestration of workflows, including the possibility of steps running on other (remote) platforms, and the on-demand deployment of processors to local/remote ADES as required.

Application Deployment and Execution Service (ADES)

WPS-T (REST/JSON) service that incorporates the Docker execution engine, and is responsible for the execution of the processing service (as a WPS request) within the 'target' Exploitation Platform (i.e. one that is close to the data). The ADES relies upon the EMS to ensure that the processor is deployed as a WPS service before it is invoked.

Application Catalogue

An Application Catalogue provides an inventory of processing services that acts as a Marketplace for the discovery and browse for processing services. The Application Catalogue provides a service that can be searched by facet/keyword and provides supporting metadata and information.

Thus, each platform that supports processing should include an ADES, and each platform that supports workflow orchestration should additionally include an EMS.

Figure 6 illustrates the main architecture components and their interfaces.

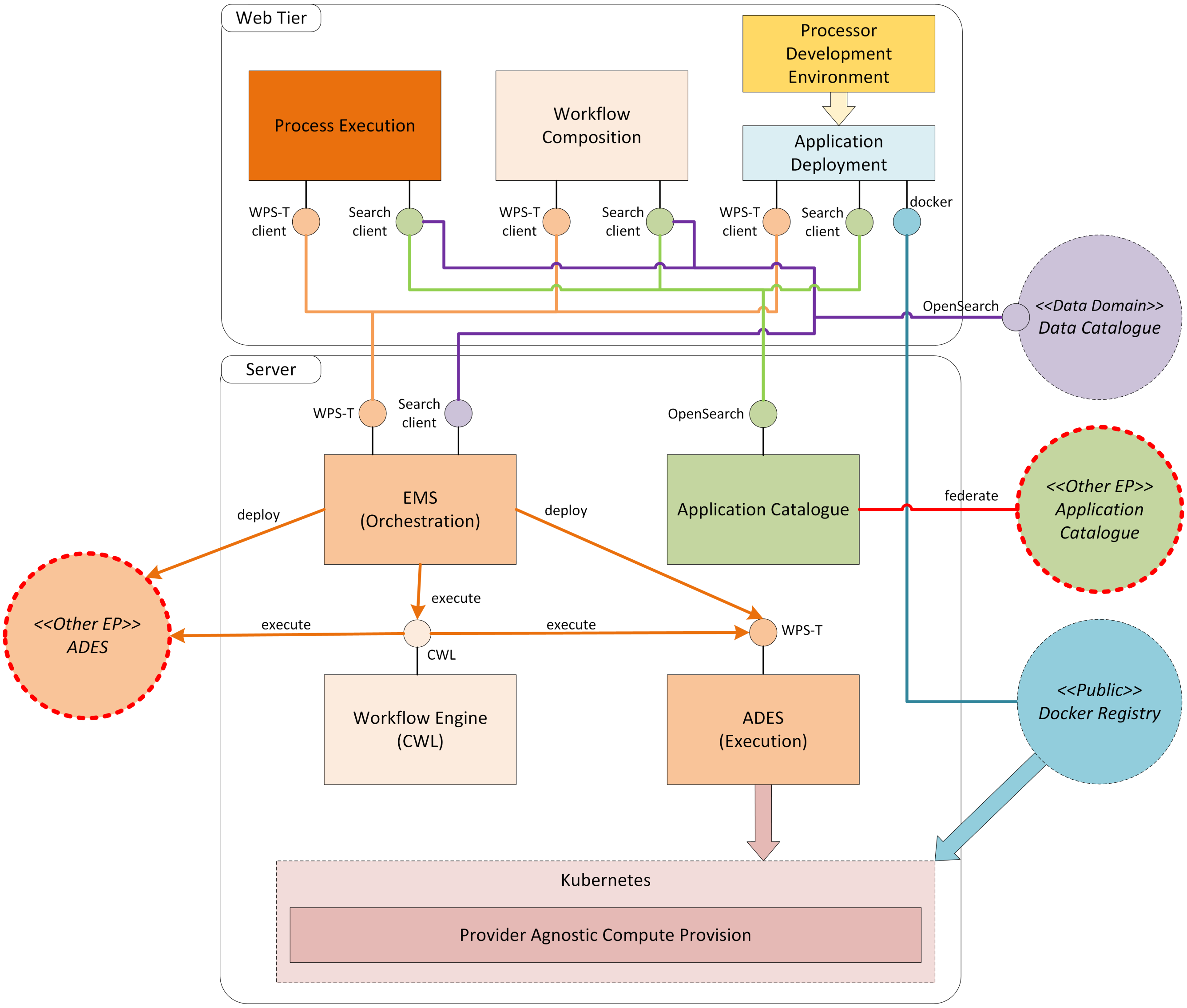

Figure 6. Processing & Chaining Overview

In order for processing services and their data input/outputs to be aligned a formalism is required to describe the data types for M2M consumption. This is required to ensure that a processor is invoked with compatible data inputs, its outputs are understood, and that coherent workflows can be constructed in which the outputs → inputs are aligned. For this purpose, Common Workflow Language (CWL) is used in the Application Package Description to describe the processor input/ouputs.

The Application Catalogue is the subject of the current Testbed-15 through which the Data Model and catalogue Service Interface are being explored.

5.2. Resource Layer (Infrastructure) Interface

The Processing & Chaining has significant points of interface with the hosting infrastructure for provision of scalable compute resource and access to data for input/output. The definition of this interface should be agnostic of the infrastructure provider onto which the Exploitation Platform is deployed.

Kubernetes provides an infrastructure abstraction layer that allows the EP to be architected in a way that is agnostic to the underlying hosting infrastructure – the only requirement being the existence of a K8s cluster in which to deploy and run the platform. This abstraction provides points of interface for:

- System deployment
- Access to back-end data
- Execution of processing services and applications

Thus, the Processing & Chaining solution is designed to utilise a Kubernetes Cluster, whose API provides the means to invoke the WPS processing services as docker containers, and also provides the means to support stage-in/out of data for the process execution.

This has particular impact on the ADES, as described in [ADES].

5.3. Application Packaging

The Application Package provides a platform independent and self-contained representation of a software item, providing executable, metadata and dependencies such that it can be deployed to and executed within an Exploitation Platform. Typically, in the context of the exploitation platform, the application is an EO data processing algorithm or a workflow.

The Application Package allows the application to be exchanged in an interoperable way on any platform within the EP ecosystem. Additionally, the developer of the package need only concern themselves with conformance to the package specification and need not concern themselves with the infrastructure details of any particular EP.

The Application Package comprises two main parts:

- Application Descriptor - metadata descriptor file
- Application Artefact – i.e. the ‘software’ component that represents to the execution unit

In accordance with the approach advocated in OGC Testbed-14 (ref. zzzER), the Application Descriptor is encoded in accordance with the WPS-T DeployProcess document defined by WPS-T JSON encodings (ref. zzz). In this way, the Application Descriptor broadly provides the following details:

- A link to the application execution unit
- A description of the application’s inputs and outputs
- Other auxiliary information

Currently supported are two types of application execution unit:

1. Docker container
2. Workflow, expressed in CWL

…but the design of the application package should be extensible to support future types.

The Application Descriptor must address the needs of at least two types of users:

Application Developers

Who may not be IT experts (such as scientists), requiring an encoding that is simple enough for them to create for themselves

Machine-To-Machine (M2M)

Requiring all the information to ensure that the application is fully portable and will behave the same on all supporting platforms

Figure 7 provides an illustration of the Application Descriptor structure.

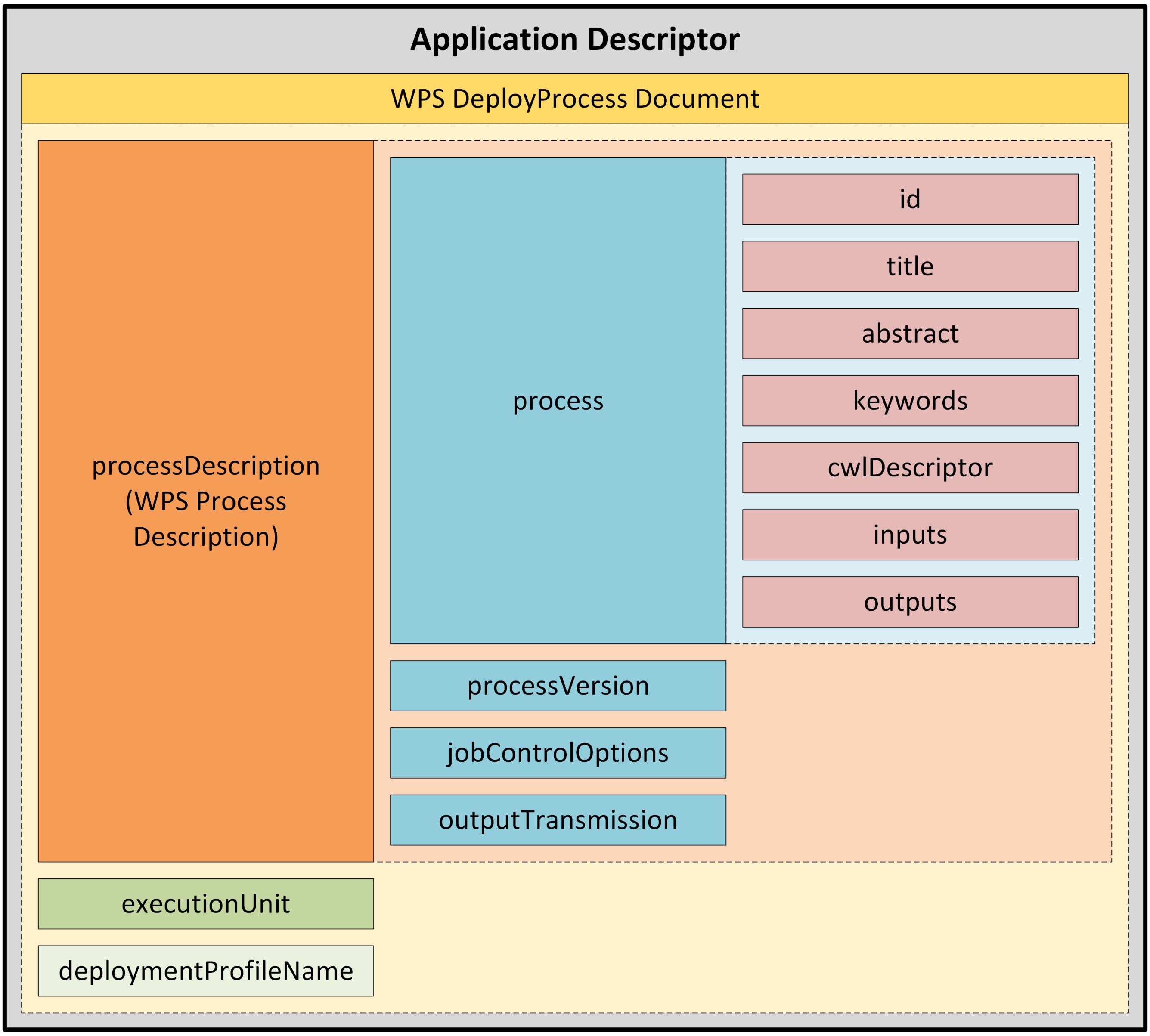

Figure 7. Structure of Application Descriptor data model

Thus, the WPS-T DeployDocument comprises the following parts:

processDescription (WPS Process Description (WPD))

Corresponds to a WPS Process Description document encoded in JSON, including details such as process ID, name, title, etc. as well as options to describe the job invocation and the output handling.

Additional points of note:

- cwlDescriptor
  The cwlDescriptor provides a CWL formatted (YAML) workflow definition of the application. This aids the stage-in/out of data by providing a CWL definition of the input/outputs of the application, and is given in addition to the inputs/outputs included in the body of the WPD. This entry is included as an extension to the WPD via an owsContext offering.
  Note that this is not required for the Execution Unit type of 'workflow' which already carries its CWL file in its executionUnit parameter.
- inputs/outputs
  Specifies the number and types of the data input/outputs. Provided as part of the WPD, and in addition to the contents of cwlDescriptor.
  The inputs can be provided as references to data, accessible through data access service endpoints, or can be specified as query parameters collection/AOI/TOI.

executionUnit

Specifies the 'software' item to be executed, with the context of the deploymentProfileName, as follows:

- dockerizedApplication
  executionUnit specifies the URL of the docker image to run.
- workflow
  executionUnit specfies the URL of the CWL file that defines the workflow.

deploymentProfileName

Enumerates the type of the executionUnit. Currently supported:

- Docker image (http://www.opengis.net/profiles/eoc/dockerizedApplication)
- CWL Workflow (http://www.opengis.net/profiles/eoc/workflow)

An example Application Descriptor follows…

```
{
    "processDescription": {
        "process": {
            "id": "EoepcaProcessor",
            "title": "EOEPCA Processor",
            "owsContext": {
                "offering": {
                    "code": "http://www.opengis.net/eoc/applicationContext/cwl",
                    "content": {
                        "href":
"https://eoepca.github.io/processor/cwl/EOEPCAProcessor.cwl"
                    }
                }
            },
            "abstract": "",
            "keywords": [],
            "inputs": [
                {
                    "id": "images",
                    "title": "Input Images",
                    "formats": [
                        {
                            "mimeType": "application/zip",
                            "default": true
                        }
                    ],
                    "minOccurs": 1,
                    "maxOccurs": "unbounded",
                    "additionalParameters": [
                        {
                            "role":
"http://www.opengis.net/eoc/applicationContext/inputMetadata",
                            "parameters": [
                                {
                                    "name": "EOImage",
                                    "values": [
                                        "true"
                                    ]
                                }
                            ]
                        }
                    ]
                }
            ],
            "outputs": [
                {
                    "id": "output",
                    "title": "Stacked Image",
                    "formats": [
                        {
                            "mimeType": "image/tiff",
                            "default": true
                        }
                    ]
                }
            ]
        },
        "processVersion": "1.0.0",
        "jobControlOptions": [
            "async-execute"
        ],
        "outputTransmission": [
            "reference"
        ]
    },
    "executionUnit": [
        {
            "href": "hub.docker.com/eoepca/processor:latest"
        }
    ],
    "deploymentProfileName":
"http://www.opengis.net/profiles/eoc/dockerizedApplication"
}
```

5.4. Execution Management Service (EMS)

The EMS provides a Transaction WPS 2.0 (WPS-T) interface, with REST/JSON encodings, as described in section WPS-T REST/JSON.

WPS-T extends standard WPS by adding DeployProcess and UndeployProcess operations. Once a process has been deployed to a WPS then the existing wps:Execute operation remains applicable for execution in the standard way.

The EMS provides a WPS-T (REST/JSON) interface that provides an umbrella orchestration service to deploy/invoke processing services within the ADES of the appropriate (close to data) Exploitation Platform. Thus, the EMS is responsible for the orchestration of workflows, including the possibility of steps running on other (remote) platforms, and the on-demand deployment of processors to local/remote ADES as required.

The description in this section refers to the WPS operations: GetCapabilities, DescribeProcess, Execute, GetStatus, GetResult, DeployProcess, UndeployProcess. See WPS-T REST/JSON for a mapping of these operations into the REST/JSON encoding.

The EMS provides the endpoint for the user's web client, through which applications and workflows are deployed to the EMS to make them available for execution.

Figure 8 illustrates the deployment of applications and workflows to the EMS.

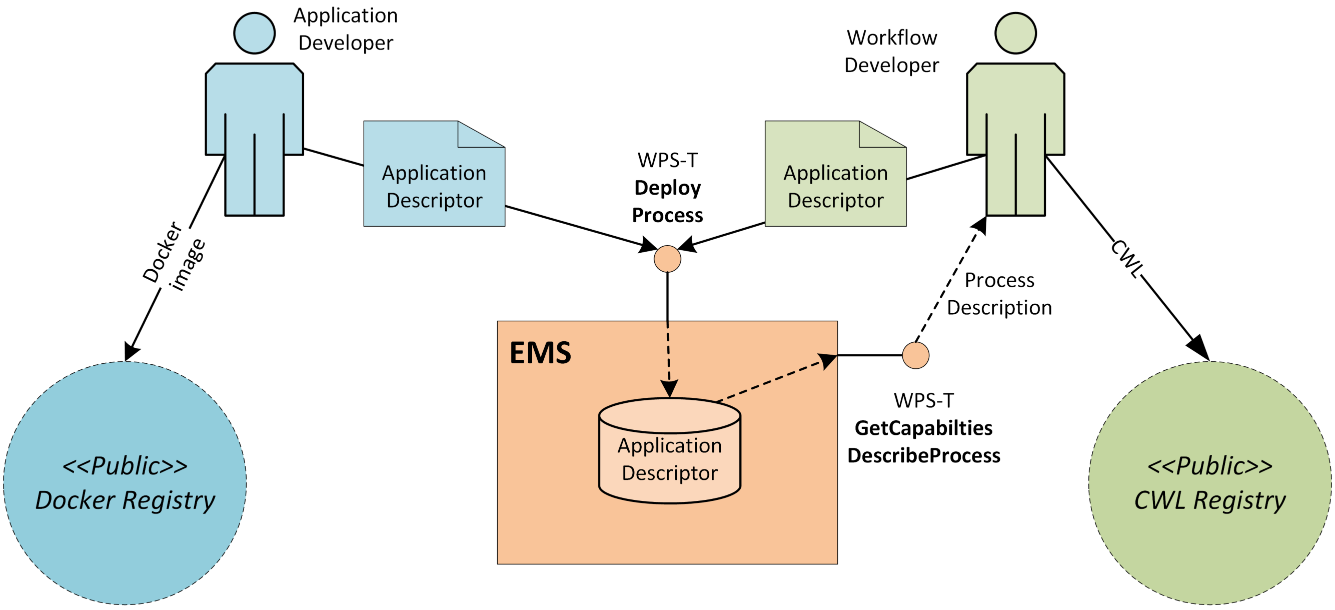

Figure 8. EMS Deployment

Applications are deployed to the EMS so that they are available for inclusion in workflows.

Workflows are deployed to the EMS where the steps of the workflow reference applications that are known to the EMS.

As illustrated in Figure 9, the EMS orchestrates the workflow execution by invoking the steps as subordinate invocations of wps:Execute at the ADES identified at time of task invocation. The EMS uses wps-t:DeployProcess on the target ADES to ensure that the process is registered before execution.

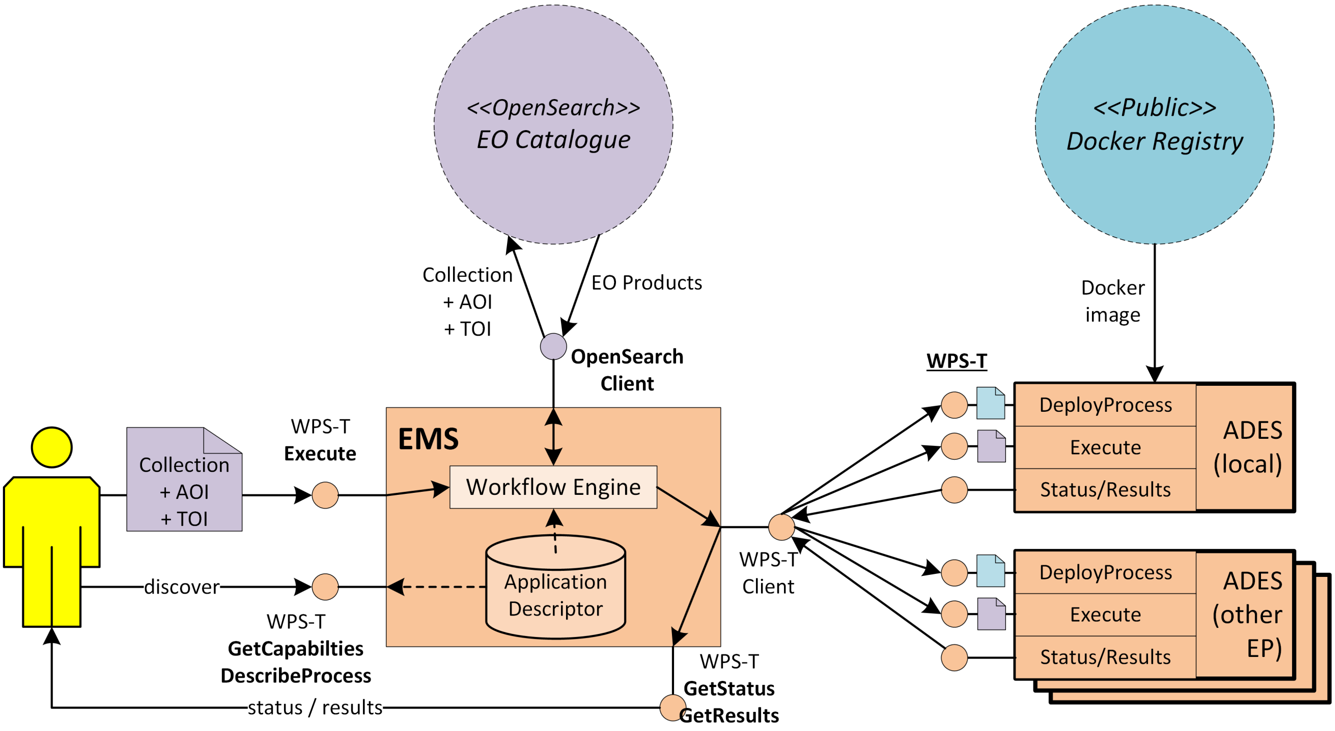

Figure 9. EMS Workflow Execution

At time of `wps:Execute` the input data must be specified by the invoking user. Two possibilities are currently identified, both of which should be supported by EMS:

1. Direct URL references to specific data products, accessible through data access service endpoints (such as WCS, WFS, etc.)
2. OpenSearch query parameters that identify the data characteristics as a combination of Collection/AOI/TOI

In case 1) the EMS can simply pass-through the input arguments to the ADES WPS-T.

In case 2) the EMS must resolve the input data by OpenSearch catalogue queries with the provided parameters. The OpenSearch catalogue end-point can either be defined by the application (in its Application Descriptor), or defined as a parameter of the `wps:Execute`.

In either case, the end result is that the EMS resolves the input specification to a set of data products URLs that can be passed on to the ADES for execution.

The EMS requires a means to determine the target platform (ADES) for the execution, i.e. typically the one closest to the data. In the case of the OGC Testbeds, this determination was made as a one-to-one mapping from the collection identified in the input data specification. If collections are identified to be globally unique, e.g. with a namespace prefix that identifies the hosting platform, then this assertion can be reliably made and the target ADES can be derived from the collection ID. Otherwise, the `wps:Execute` must be parameterised suitably to identify the target ADES.

Performing the orchestration between steps, the EMS must handle the stage-in and stage-out of data. In the simple case, the result URL returned from a step can be directly used as an input URL for the subsequent step. Use of CWL and the accompanying `cwl-runner` tool should facilitate this orchestration.

The end result of the successful execution is to present the output result to the invoking user. The EMS establishes the location of the results within the storage provision of the Exploitation Platform, and interfaces with the EP Workspace component ([User Workspace]) to register the result in the user's workspace. At this point the WPS execution is complete as reported by the `wps:GetStatus` and `wps:GetResult`.

5.5. Application Deployment and Execution Service (ADES)

The ADES provides a WPS-T (REST/JSON) service that incorporates the Docker execution engine, and is responsible for the execution of the processing service (as a WPS request) within the 'target' Exploitation Platform (i.e. one that is close to the data). The ADES relies upon the EMS to ensure that the processor is deployed as a WPS service before it is invoked.

The main responsibilities of the ADES are:

- Check the user is authorised to access the requested data
- Perform stage-in of data before execution
- Invoke the container from the Docker image in accordance with the ApplicationDescriptor and the `wps:Execute` request
- Monitor the status of the job and obtain the results
- Perform stage-out of results at execution conclusion

Resource Layer (Infrastructure) Interface introduces the use of Kubernetes (K8s) as the provider agnostic interface to the Resource Layer. The ADES has touch-points with the Resource Layer for access to data and compute resource. The following sub-section elaborate the approach.

5.5.1. Container Execution

The work carried out in the OGC Testbeds 13/14, performed the execution of the 'packaged' processing service by invoking the 'run' of a docker container in the machine that hosts the WPS-T service. The Common Architecture design builds upon this, by instead invoking the container as a K8s Job that is deployed for execution in the K8s cluster.

This approach is consistent with the current Application Package / ADES definition that specifies a docker image for the processing service. As illustrated in Figure 10, the ADES provides a K8s-aware Execution Engine that handles the complexities of constructing the jobs and interfacing with the K8s cluster.

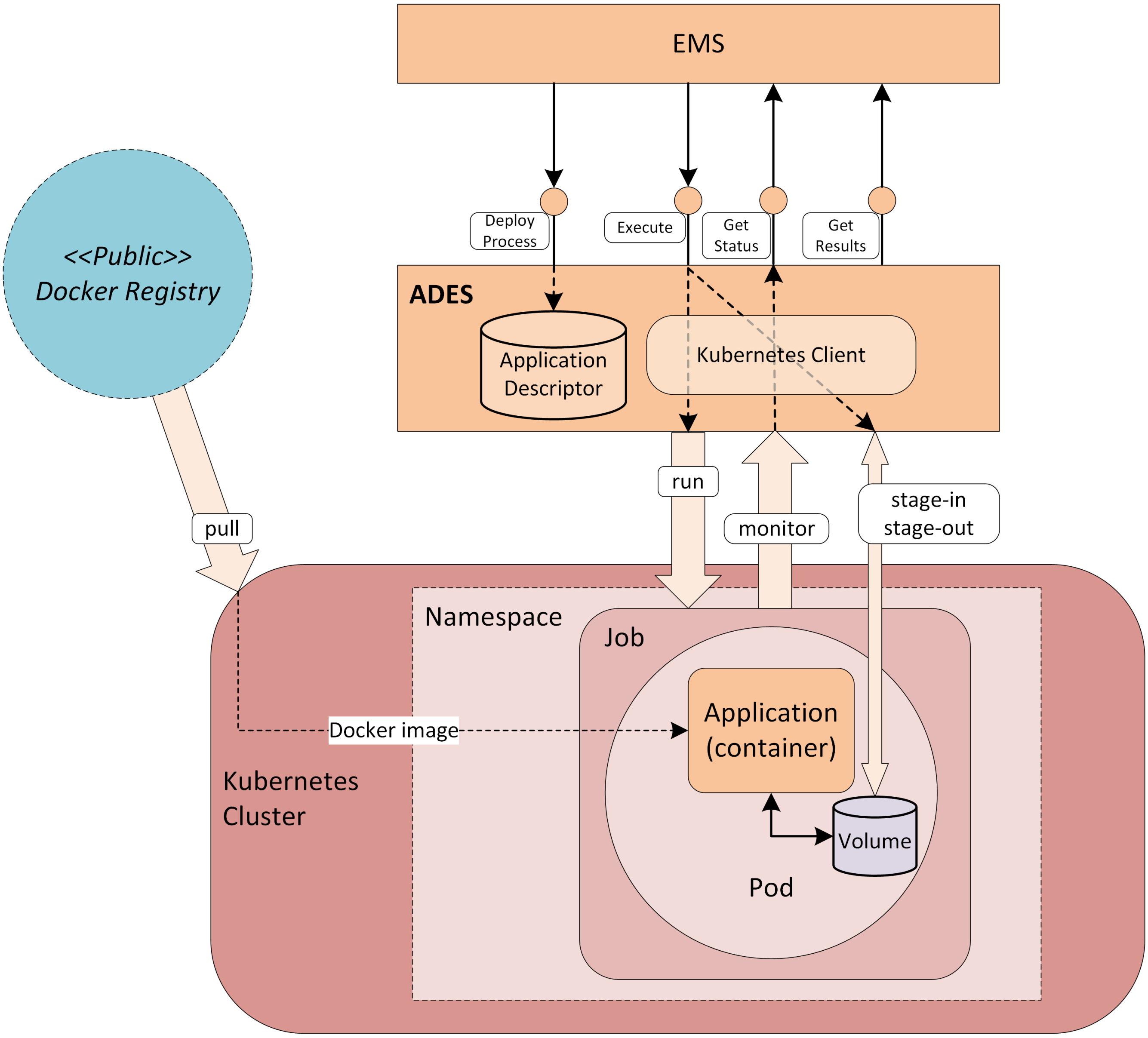

Figure 10. EMS Process Execution

A Kubernetes cluster comprises a set of Nodes. A Node is a worker machine in Kubernetes and may be either a virtual or a physical machine. Each Node is managed by the Master. A Node can have multiple pods, and the Kubernetes master automatically handles scheduling the pods across the Nodes in the cluster. The Master's automatic scheduling takes into account the available resources on each Node.

Pods are the atomic unit on the Kubernetes platform. A Pod is a Kubernetes abstraction that represents a group of one or more application containers. A Pod always runs on a Node. The containers in a Pod share an IP Address and port space, are always co-located and co-scheduled, and run in a shared context on the same Node.

Each WPS processing task will be constructed as a Pod and invoked as a dedicated K8s Job. A Job creates one or more Pods to perform a given task. The Job object takes the responsibility of Pod failures. It makes sure that the given task is completed successfully. Once the task is over, all the Pods are terminated automatically.

Kubernetes provides Namespaces, which are an abstraction that supports multiple virtual clusters on the same physical cluster. It may be interesting to explore the use of K8s Namespaces for the purpose of establishing a sandboxed execution environment for each task execution.

5.5.2. Data Stage-in/out

There is need for the Processing Framework (ADES) to prepare the inputs before job invocation, and collect the outputs at the job conclusion. The processing task is invoked as a Docker container. In doing so, the container execution environment must be provisioned with the input data for the task, and with the means to ‘export’ its outputs to the processing orchestration.

The work carried out in the OGC Testbeds 13/14 relied upon use of mounted volumes within the processing task docker container. These mounted volumes present the input data, and receive the output data, as ‘local’ file system access from the point of view of the running container and the processor running within. Use of mounted volumes is equivalently supported by K8s, and is an approach that is relied upon for many (existing) applications that are capable only of accessing data through POSIX file-system interfaces.

Thus, for Mounted Volumes, the approach is to use standard container volumes that present as well-identified directories within the container. The input data is provided in a read-only input directory that mounts into the container hosting infrastructure. Similarly, an empty writable directory is presented for the processing to write is outputs, to be collected by the Processing Framework.

Nevertheless, we might envisage that additional data access approaches need to be supported, including:

- AWS S3 Object Store
- Swift Object Store (OpenStack)
- OGC data access services: Web Map Service (WMS), Web Coverage Service (WCS), Web Feature Service (WFS)
- Mounted Volume (local storage)

With the exception of local storage, the above-mentioned data access technologies rely upon HTTP-based protocol access. Regarding these HTTP-based interfaces there are two possibilities that need to be handled by the Processing Framework:

1. The processing service / application natively supports the data access protocol, in which case there is no need to stage-in the data. Nevertheless, the Processing Framework must support the pass-through of input data as URL, and the reception of the output(s) as URL. Also, it must be ensured that any outputs, e.g. to object store, are appropriately directed.
2. The processing service / application only supports local storage (mounted volume), in which case the Processing Framework must perform the data retrieval to stage-in the data. At the conclusion of the processing the outputs must be marshalled into the appropriate platform storage for further consumption, e.g. object store.

Use of Filesystem in Userspace (FUSE)

Regarding case 2), another possibility exists that requires further investigation. Access to the back-end data storage (e.g. HTTP-based) is provided through a user-space driver that presents the remote data as if it were a local directory. For example, access to data in an S3 object store is provided through a FUSE mounted directory. Thus, from the perspective of the processing task, inputs (read) and outputs (write) are accessed through the local file-system interface – satisfying the constraints of the processor.

5.5.3. User Authorisation Context

The stage-in/out of data must operate within the context of the user's 'account'. Thus, the security context of the user must be passed through all aspects performed by the Processing Framework on behalf of the user. This is necessary to ensure that the user is only able to access data to which they are entitled and accounting & billing considerations are properly maintained.

5.6. WPS-T REST/JSON

This interface specification is used for both the Client <→ EMS, and the EMS <→ ADES interfaces.

WPS-T extends standard WPS by adding *DeployProcess* and *UndeployProcess* operations. Once a process has been deployed to a WPS then the existing `wps:Execute` operation remains applicable for execution in the standard way.

The following table is reproduced from [TB14-ADES].

Resource	**HTTP Method**	**Description**	**WPS operation**
/	GET	The landing page provides links to the API definition, the Conformance statements and the metadata about the processes offered by this API	
/processes	GET	Retrieve available processes	GetCapabilities
/processes	POST	Deploy a process	DeployProcess
/processes/{id}	GET	Retrieve a process description	DescribeProcess
/processes/{id}	DELETE	Undeploy a process	UndeployProcess
/processes/{id}/jobs	GET	Retrieve the list of jobs for a process	
/processes/{id}/jobs	POST	Execute a process	Execute
/processes/{id}/jobs/{jobID}	GET	Retrieve the status of a job	GetStatus
/processes/{id}/jobs/{jobID}	DELETE	Dismiss a job	
/processes/{id}/jobs/{jobID}/result	GET	Retrieve the result(s) of a job	GetResult

Resource	HTTP Method	Description	WPS operation
/processes/{id}/quotations	GET	Retrieve the list of quotation ids for a given process	
/processes/{id}/quotations	POST	Request a quotation for a given process	
/processes/{id}/quotations/{quotationID}	GET	Retrieve quotation information	
/processes/{id}/quotations/{quotationID}	POST	Execute a quoted process	
/processes/{id}/visibility	GET	Retrieve the visibility status for a process	
/processes/{id}/visibility	PUT	Change the visibility status for a process	
/quotations	GET	Retrieve the list of all quotation ids	
/quotations/{quotationID}	GET	Retrieve quotation information	
/quotations/{quotationID}	POST	Execute a quoted process	
/bills	GET	Retrieve the list of all bill identifiers	
/bills/{billID}	GET	Retrieve bill information	
/conformance	GET	list all requirements classes specified in the standard (WPS REST/JSON Binding Core) that the server conforms to	

5.7. Interactive (Graphical) Applications

The work carried out in the OGC Testbeds focused on non-graphical applications, i.e. non-interactive processing functions executing algorithms without intervention. It is also noted that WPS does not facilitate the invocation of GUI-based interactive applications which offer a synchronous experience to the end-user.

That said, the approach to application packaging undertaken in the testbeds does lend itself to the packaging of GUI-based applications, which can be packaged, deployed and executed as docker containers, including:

Native applications

A remote desktop (RDP) approach is used to present the interface to the user, typically rendered through a web page presented in the user’s browser

Web applications

The web application is delivered through the portal interface of the hosting exploitation platform.

In both cases, docker containers offer a good solution to package and deploy the application. At execution time it is necessary to ensure that the appropriate ports are exposed from the running container.

The Application Descriptor needs to be extended to:

- Introduce additional `deploymentProfileNames` and `executionUnit` types
- Provide parameterisation to support the delivery of the GUI to the end-user
- Ensure that data access is presented within the container in a way that is compatible with the GUI application (the mechanisms provided for non-interactive applications may be sufficient).

5.8. Parallel Processing

The OGC Testbeds 13/14 only consider serial processing jobs running in a single Docker container. Here we consider how this approach can be extended to accommodate job requiring parallelisation.

One possible approach, is to invoke the processing task as a docker container (as described above), but then within the implementation of this task it makes subordinate invocations that exploit some specific data processing clustering infrastructure available within the platform. For example, the invoked process executes some Python code that then invokes a dask or SLURM cluster to perform the processing work.

In this case, the processing task would have a dependency that the Exploitation Platform provides the required data processing technology. In order to resolve this capability dependency the following approach can be made:

- The processing service declares within its Application Deployment Package, that it ‘requires’ a particular service
- The Exploitation Platform declares within the capabilities document output from its WPS endpoint, that it ‘provides’ particular services
- The EMS must ensure that the target EP provides the required service of the processing task to be invoked
- The parameterisation for the ‘required’ service are passed to the processing task at invocation

A consistent vocabulary of services must be defined to unambiguously express the ‘required’ and ‘provides’ declarations.

6. Resource Management

The role of the Resource Management domain is the storage, discovery and access to resources in the Exploitation Platform. In this context, resources primarily refers to data and processing assets.

Storage is largely taken care of by the Resource Tier upon which the Exploitation Platform is hosted. The role of the Exploitation Platform is to ensure that the data can be accessed through common data access protocols based upon open standards.

This is important for:

- the end-user wishing to access data directly
- processing services accessing data for input/output
- other federated Exploitation Platforms accessing each other's data and services through well understood interfaces

To exploit the services of the Platform, users need to discover available data, obtain detailed collection/product information, including the ability to visualise the data in the platform. This applies to data held within the platform, data added by end-users and data produced as the result of processing operations within the platform.

Processing services and applications are also platform resources that must be discoverable by users, including the information required by users to exploit the service.

The inventory and presentation of resources to users must be organised in such a way as to facilitate the discovery and usage of resources in other federated Exploitation Platforms. For example, users must be able to discover data and services in other EPs in order to construct and execute workflows that span multiple federated EPs.

Access to resources must be controlled according to the privileges afforded to the logged in user, and appropriate hooks must be established into the EPs accounting and billing subsystems. Thus, the Resource Management services must be implemented according to the approach defined by User Management for authorisation, accounting and billing.

In addition to the resource holding of the underlying resource tier, the EP maintains a User Workspace in which each user is able to maintain specific data/services of interest to them, and also provides a place to hold results of processing operations. The User Workspace should be provided as a building block of the system that provides this personal inventory. Moreover, the concept can be extended to define Group Workspaces to create a place for sharing and collaboration.

A Data Ingestion component abstracts the interface to the underlying Resource Tier storage, ensures that incoming data is formatted in accordance with defined standards, is supported by appropriate metadata and directed towards the appropriate dataset collection.

Thus, the main components comprising the Resource Management domain are:

- Data Catalogue
- Application Catalogue

- Data Access Services
    - AWS S3 Object Store
    - Swift Object Store (OpenStack)
    - OGC Web Map Service (WMS)
    - OGC Web Coverage Service (WCS)
    - OGC Web Feature Service (WFS)
- Data Ingestion
- User Workspace

To some degree, the role of these components is to provide an integration of the Exploitation Platform to the Resource Tier, by providing public services that bridge to the underlying data supply.

7. Web Portal

The Web Portal represents the browser-based user interface through which the user interacts with the EO Exploitation Platform, and the public web-API that supports mobile and third-party (programmatic) access to services of the platform.

The Web Portal is not a domain area in its own right – it is contributed to by the collaborative developments of the defined domain areas. The user's view of the platform is consolidated through the browser-based user interface, and hence it is convenient to present all aspects of this view together. Thus, the Web Portal provides the user facing 'front' of the system and interfaces to the services provided by the domain areas identified in the system design.

The Web Portal must provide a consistent and cohesive user experience that aggregates data and processing services of the EO Exploitation Platform. In doing so it must provide the following main functionalities:

- User login
- Marketplace, that provides a federated system search for discovery of data and processing capabilities
- User workspace to support scientific analysis and collaboration
- Data discovery and download of data
- In-browser visualisation of data and processing/analysis results
- Discovery, execution and monitoring of processing jobs
- Definition of workflows from discovered data/processing resources
- Hosting of user defined applications with interactive user interfaces
- Web Application Programming Interface (API) providing user's external access to the platform's services outside of the browser interface, (expected to be a thin wrapper around the service end-points offered by other domain areas)
- Hosting of rich media content that is linked to catalogued resources. Such content ranges from documents & manuals to tutorials and i nstructional *media.
- Hosting of community and collaboration tools such as Wikis, FAQs and forums

These user-facing web components form part of other domain areas that together present a rich integrated user experience. Figure 11 presents these functional areas, organised within their respective domain areas.

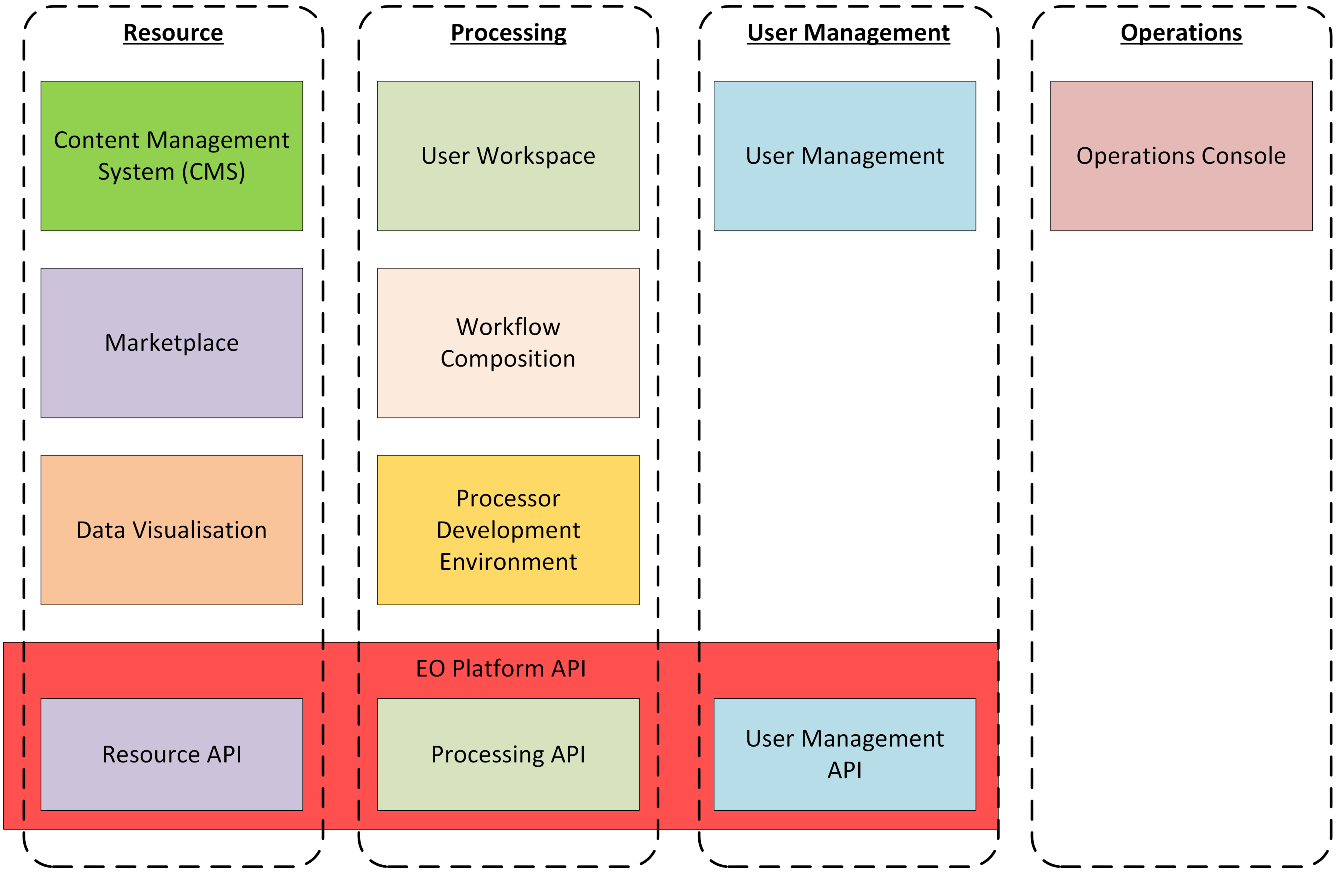

Figure 11. Web Portal: Overview

The platform should support provision of limited access to unauthenticated (guest) users, in which they can search the marketplace to discover the services and data available, and browse supporting materials. Access to the full capabilities of the platform requires registered users to identify and authenticate.

Optionally, a **Content Management System (CMS)** can provide a framework within which the platform's web presence is hosted. It facilitates the creation of user content that can be linked to data and processing resources in the Resource Catalogue. In addition, the CMS provides out-of-the-box facilities for Wikis, FAQs, forums etc.

The **Marketplace** builds a user experience on top of the Resource Catalogue that provides a consolidated inventory of all services and data published within the federated system. The user is presented with the ability to browse and to perform rich search queries to discover items of specific interest. The Marketplace content for a data item can include interactive Data Visualisation, such as providing a WMS viewer that exploits the WMS service provided with the platform's resource service. This **Data Visualisation** component is re-usable such that it can be used elsewhere in the user experience, for example from the user's workspace to visualise some processing results.

The *User Workspace *provides the environment where users are able to organise data and processing they are interested in, and to manage asynchronous 'tasks' they have submitted into the platform. Thus, they are able to monitor data retrieval and processing requests and obtain the outputs at completion. The facility is also provided for them to publish derived 'added-value' outcomes from their workspace into the Resource Catalogue, and so present them in the marketplace.

Experts use the **Workflow Composition** interface to chain and combine multiple processing functions and input data into reusable workflows. The interface allows them to select these resources discovered via the Marketplace, architect and execute their workflow, and ultimately publish it as a reusable processing function that is available to others in the Marketplace.

Experts are able to develop and submit to the EO Exploitation Platform their own custom processing algorithms, tools and applications. The **Processor Development Environment** provides a rich, interactive environment in which processing algorithms and services can be developed, tested, debugged and ultimately packaged so that they can be deployed to the platform and published via the marketplace.

User Management provides the functionality associated with user profiles. New users will have the ability to self-register and then manage all aspects of their profile interactively - noting that the intention in the Common Architecture is to delegate User Identity management to external IdPs.

Operators will have access to management interfaces for system monitoring and administration.