Cookbook for Cloud Enabled Applications

APPROVAL

Title Cookbook for cloud enabled applications	
Doc.Ref. EOP-SD-MA-0018	
Issue Number 1	**Revision Number** 0
Status Issued	
Author Platform Support Team	**Date** 14/02/2019
Approved By	**Date of Approval**
H/EOP-SDD	**14/02/2019**

DISTRIBUTION

public

1. INTRODUCTION

1.1. Purpose

The purpose of this document is to provide a set of best practices to support the development of a new generation of cloud-ready applications in the context of EO Satellite data processing and algorithms elaboration.

Each step of this guideline represents a single, essential rule for the developers to move forward to the design of Services based on their idea of Applications.

Within this document, the term "Application" is used to describe the whole set of modules and dependencies of the designed software, in opposition to what the users can access through it (the Services).

All the concepts and recommendations in this document can be extended to the design of the Exploitation Platforms, where the services are to be intended as the different functionalities offered by the platform itself.

1.2. Document structure

The first section of this document contains a brief introduction, the definition of terminology and a list of web references and acronyms.

The second section is a collection of paragraphs, each one containing a rule for the development of a cloud enabled application.

1.3. URL list within this document

[URL-1]	The Open Geospatial Consortium (OGC) https://www.opengeospatial.org/
[URL-2]	The Web Processing Service (WPS) https://www.opengeospatial.org/standards/wps
[URL-3]	The Web Map Service (WMS) https://www.opengeospatial.org/standards/wms
[URL-4]	The Web Coverage Service (WCS) https://www.opengeospatial.org/standards/wcs
[URL-5]	The Web Feature Service (WFS) https://www.opengeospatial.org/standards/wfs
[URL-6]	Stateless applications https://whatis.techtarget.com/definition/stateless-app
[URL-7]	Cloud Controls Matrix (CCM) https://cloudsecurityalliance.org/group/cloud-controls-matrix/
[URL-8]	The eduGAIN interfederation service https://edugain.org/
[URL-9]	Security Assertion Markup Language (SAML) https://wiki.oasis-open.org/security/FrontPage
[URL-10]	OAuth 2.0 standard (RFC 6749) https://oauth.net/2/
[URL-11]	Role based access control (RBAC) https://csrc.nist.gov/CSRC/media/Publications/conference-paper/1992/10/13/role-based-access-controls/documents/ferraiolo-kuhn-92.pdf

1.4. Acronyms and Abbreviations

Acronym	Description
SaaS	Software as a Service
GUI	Graphical User Interface
AWS	Amazon Web Service
COG	Cloud Optimized GeoTIFF
OGC	Open Geospatial Consortium
REST	Representational State Transfer
WMS	Web Map Service
WPS	Web Processing Service
WCS	Web Coverage Service
WFS	Web Feature Service
API	Application Programming Interface
IAM	Identity and Access Management
FIM	Federated Identity Management
SAML	Security Assertion Markup Language
RBAC	Role Based Access Control

1.5. Terminology

Term	Meaning
Cloud Computing	Cloud computing (also called simply, Cloud) describes the act of storing, managing and processing data online — as opposed to on local, physical computer or network.
Exploitation Platform	A collection of on-line services, products and tools for development, analysis, support and collaboration in the exploitation of the EO Data in a collocated way, bringing the users to the Data.
Cloud service providers	A company that offers some component of cloud computing -- typically infrastructure as a service (IaaS), software as a service (SaaS) or platform as a service (PaaS) -- to other businesses or individuals.
The Agile Method	It is an incremental approach to project management which assists teams in responding to the unpredictability of constructing software.
DevOps	It is an operational philosophy that promotes better communication between development and operations as more elements of operations become programmable.
Horizontal scalability	The capability of distributing the load over a cloud-based network of "workers" nodes and storage devices. The right framework can achieve it dynamically if properly configured.
Vertical scalability	Increasing the physical (or virtual) resources (typically CPU, memory and storage) of the single computing machine. This typically requires a down time of the system.
Tokenization	An authentication method which allows users to enter their username and password in order to obtain a token which allows them to fetch a specific resource, for a time.

2. CORE PRINCIPLES

2.1. Provisioning of the application as a SaaS

The new generation of cloud-based applications should be provided as a Software as a Service (SaaS). The services of the application should be exposed through a well-defined list of APIs, allowing other application builders to build on top of these APIs.

The application's overall service needs to be exposed to the outside world through individual functions accessible via Web service APIs, i.e. not requiring knowledge of the application's inner workings, not resulting in side effects, and using an interface that will change very little over time. The same approach needs to be kept within the applications when exposing (sub) components to each other.

Decomposing the application service into single-purpose functions enables reuse, facilitates elastic scalability, and supports the Agile development method and the DevOps organization of software lifecycle.

On top of these APIs, the application can be accessible from various client devices through either a thin client interface, such as a web browser or a custom Graphical User Interface (GUI). The consumer does not manage or control the underlying cloud infrastructure, including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.

2.2. Accessing data on the cloud

The desktop applications which are not designed to be deployed on the cloud, make use of traditional, local storage systems. As the data grows, these local storage systems become a bottleneck for most applications.

In particular for the storage, there are many different options available to store the data in a distributed way (e.g. Object storage, No-SQL database, RDB, Cache storage, Data warehouse, Data Archive storage, Elastic block storage), and they should be leveraged by the new generation of applications.

To ensure maximum compatibility and versatility, the applications should be able to read/write data on the most prominent storage services like:

- Swift Object Store (from OpenStack - largely used among the most important EO data providers like the DIAS)
- AWS S3
- OGC interfaces abstracting from file storage like WCS or WMS (see below), as available for the used data and suitable for the application.

Also, the support to data formats allowing a more efficient web access, can enable clients to issue HTTP GET range requests to ask for just the parts of a file they need.

For example, when available on the Cloud provider, Cloud Optimized GeoTIFF (COG) format, can make the software stream just the portion of image product that it needs, improving processing times and creating real-time workflows previously not possible.

2.3. Adopting standard interfaces

In order to improve the extensibility of the application, a set of standard interfaces should be used to interact either with external services providing discovery, data or processing services or with customers of the application.

These interfaces should be designed in such a way that:

- they will change very little over time
- they are backwards compatible whenever possible

Opting for an HTTP-based infrastructure based on paradigms like REST, will make it easier to port the system to a new environment and gives to the application a reliable machine-to-machine communication capability.

All data provided to or by the subcomponents of the applications, need to be transferred through standardized interfaces, both in structure and technology. The Open Geospatial Consortium (OGC) [URL-1] offers quality open standards for the global geospatial community. These standards are made through a consensus process and are freely available for anyone to use, and they improve sharing of the world's geospatial data.

Mainly crucial OGC Web Services for the EO class of applications include:

- Web Processing Service (WPS): the standard which defines how a client can request the execution of a process, and how the output from the process is handled. [URL-2]
- Web Map Service (WMS): the standard which provides a simple HTTP interface for requesting geo-registered map images from one or more distributed geospatial databases. [URL-3]
- Web Coverage Service (WCS): the standard which provides an open specification for sharing raster datasets on the web. [URL-4]
- Web Feature Service (WFS): the standard which makes geographic feature data (vector geospatial datasets) available through a highly configurable interface. [URL-5]

2.4. Making the application ready for scaling

Using cloud resources is only efficient if payment for idle resources is minimized. This implies that resources, needed to serve on-demand requests by the users, are allocated dynamically in correspondence to increasing user load and are deallocated when becoming idle.

Typical increasing load scenarios include:

- more users logging on at the same time
- the request to process a greater amount of data in a given time constraint
- increasing the number of services

In order to achieve a better level of flexibility and reduce the footprint of idle resources, the application should be able to replicate the processing chain over different instances of a worker template, distributed over different nodes of a cloud infrastructure, following a floating load of processing demand (horizontal scalability). This will reduce the need to increase the hardware of the single machine the application is based on (vertical scalability). A good assessment of the request of new resources, should be tailored on the number of users having access to the application, so that also the budget allocation for the application exploitation will significantly be improved and made clear.

It is important to figure out how to better scale the application by automatically spinning up resource instances that are needed. In some cases, cloud service providers offer auto-scaling capabilities, where provisioning occurs automatically. Applications should be packaged, deployed and executed without manual steps or complex dependencies. All provisioning, instantiation and configuration for an application should be scripted and preferably encapsulated in an automatically unpacked and deployable template. Automating deployment also means encapsulated application dependencies for deployment as a single package.

The replication mechanism over the cloud infrastructure, strictly implies to have a stateless processing chain (or sub-module) [URL-6]. In this way, since a service does not store any per-client state, the service remains lightweight in terms of resource consumption, ensuring more efficient use of resources and allowing each individual server to support a higher request load.

The most efficient path, however, lies in understanding the application's workload profile and defining the path to scaling the application, as well as putting mechanisms in place to ensure that it will, indeed, scale.

Finally, it is important to monitor overall application performance using application-aware performance monitoring tools, and create interfaces within the application to better enable performance monitoring. How the application provisions and de-provisions resources should be innate to the application as well.

2.5. Infrastructure Independence

Maintaining infrastructure independence is consistently a recommended software engineering practice, but in a cloud environment, using the cloud provider's proprietary interfaces, results in cloud vendor lock-in. Vendor lock-in reduces portability and makes it difficult or impossible to run the application on other cloud platforms.

With abstraction, it is possible to replace the implementation of an underlying service with another, without rewriting application code as long as the interface "contract" between the application and the service is maintained.

This is why applications shouldn't make any assumptions about the underlying infrastructure, using abstractions in relation to the operating system, file system, and so on. As a best practice, the application should use cloud API within the business logic of the application but to collect all the cloud specific API calls in drivers which can be changed at need. In this respect, it is recommended the use of already available libraries as LibCloud, jCloud, Fog, Libretto, pkgcloud.

2.6. Designing for security

Cloud-based application architecture should make security systemic to the application.

The designer needs to pick a security approach and technology prior to building its application that will be effective for the type of application running and that will address any compliance or other data-level security issues. Probably the application will need to handle sensitive data in specified ways, with required levels of security, such as encryption.

For example, cloud-based applications should leverage identity and access management (IAM). Applications developed with mature IAM capabilities can reduce their security costs and, more importantly, become significantly more agile at configuring security for cloud-based applications.

As a reference, the Cloud Controls Matrix (CCM) [URL-7] by the Cloud Security Alliance, defines fundamental security principles to guide cloud vendors and to assist prospective cloud customers in assessing the overall security risk of a cloud provider.

2.7. Federated Identity and Tokenization

Authentication should be delegated to an external identity provider. This pattern can simplify development, minimize the requirement for user administration, and improve the user experience of the application.

Federated identity management (FIM) refers to a way to connect identity management systems together. With FIM, a user's credentials are always stored with a "home" organization (the "identity provider"). When the user logs into a SaaS application, instead of providing credentials to the service provider, the service provider trusts the identity provider to validate the credentials.

So, the user never provides credentials directly to anyone but the identity provider. The service provider is federating itself with its FIM (identity provider). It's a many to one mapping, many SaaS applications to one identity provider.

In the context of the research, an important interfederation service is the eduGAIN [URL-8], which is based on Security Assertion Markup Language (SAML) [URL-9] and provides an efficient way for national federations to interconnect. Each federation sends its trust registry (e.g. metadata) to eduGAIN, where it is combined with all the national registries and republished.

About tokenization, the OAuth 2.0 standard (RFC 6749) [URL-10] introduces the concept of token-based access control. In OAuth, an authorization server issues tokens to clients that grant specific access rights to protected resources.

Any request and response are no different from conventional web service standards apart from the passing of the access token in the request. An access token has a limited lifetime for security reasons. When it expires, the client uses a special refresh token to request a new access token.

2.8. Role-Based Access Controls (RBAC)

A cloud-based environment often requires the sharing of the resources in a multitenancy model where a single instance of the software runs on a server, serving multiple client-organizations (tenants). Multitenancy contrasts with multi-instance architectures where separate software instances (or hardware systems) operate on behalf of different client organizations.

Role based access control (RBAC) (also called "role-based security"), as formalized in 1992 by David Ferraiolo and Rick Kuhn [URL-11], has become the predominant model for advanced access control because it reduces its cost.

For example, some clients might be granted read-only access to the service, and others might have both read and write access. Authorization can also be used to enforce architectural constraints by limiting which clients are able to access an API. Another example is a database API that might be protected with access permissions that ensure that clients use a preferred data access service. In this case, the database API would grant read/write access only to the data access service.

2.9. Design for Manageability (Instrumentation)

Sufficient telemetry data should be available from applications to determine health and other operational and performance characteristics.
Metered consumption changes service invocation patterns and design efficiency, depending on the relative cost of resources. For example, according to the retrieved metrics, it could be crucial to adapt the application to one of the following business models: charged per API call, per megabit, or per CPU/hour.

2.10. Resilience to failure

In a large-scale cloud environment, applications run on an infrastructure prone to certain types of failures that can disrupt operations. Potential vulnerabilities include hardware failure of a server, network, or storage physical appliance, a software failure in an application component or cloud service, or a network failure due to an unresponsive component or a transient network connectivity problem.

To be resilient to these types of failures, applications must be architected to handle them seamlessly and continue to operate without intervention, either at a degraded performance level or with gracefully degraded functionality.

Resilience is achieved by replication, containment, isolation and delegation. Failures are contained within each component, isolating components from each other and thereby ensuring that parts of the system can fail and recover without compromising the system as a whole.
Recovery of each component is delegated to another (external) component and high-availability is ensured by replication where necessary. The client of a component is not burdened with handling its failures.
The failure of one feature should not cause the failure of other features. When one feature — like an address book — is temporarily unavailable, the rest of the application still runs.
After the failure has been contained, an instruction set activates, restarting the failing component. These steps need to be automatic, immediate, and reliable. When the component's functionality has been restored, normal collaboration with other components can resume.

Providing stateless services is, again, crucial also for a better resilience to accidents.

2.11. Resilience to latency

Applications should remain operational despite variable and extended latency for services accessed across public networks and between cloud provider data centres.

Latency varies significantly in a cloud environment.
When architecting applications for the cloud, developers should limit the level of service nesting and ensure that time-dependent tasks are not subject to response times that result from traversing a deep stack of services.

Reducing the number of network requests can help manage latency. This can be accomplished by designing less "chatty" protocols, where a service can request only the data it needs to function (rather than make multiple requests over time to accumulate the necessary data).