|  |  |  |
| --- | --- | --- |
| Domino-X Cloud Guidelines | | |
| Domino-X | | |
|  | Prepared by: | C. Brandon (TAS), M. Vansteene (CG), S. Marsolle (CG), O. Lagarde (ADS), C. Bouisson (CS), R. Leveque (CG), V. Baccini (ADS) |
|  | Approved by: | WP 3.1 |
|  | Authorized by: | N. Mayer (WP 3.1 leader) |
|  | | |
|  | Code: | DX-GDL-000000001 |
|  | Version: | 1.0 |
|  | Date: | October 19th, 2022 |

Document Status Sheet

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Pages | Changes |
| 1.0-Draft | July 4th, 2022 | All | Initial version based on consortium feedback on cloud. |
| 1.0 | October 19th, 2022 |  | RIDS implementation   * Summary: moved from annexe * §1.1: Typos * §1.3 : add mention of Gaïa-X * §4.1.1 & §4.1.2: merge of the chapters |
| 02.00 |  |  |  |

Table 1: Status sheet

Table of Contents

[1. Introduction 6](#_Toc107907683)

[1.1. How this document was constructed 6](#_Toc107907684)

[1.2. How this document is structured 7](#_Toc107907685)

[2. Glossary 8](#_Toc107907686)

[3. Physical aspect 10](#_Toc107907687)

[3.1. Definitions of cloud 10](#_Toc107907688)

[3.2. Geographic repartition 11](#_Toc107907689)

[3.3. Trade off on “On premises” vs “Cloud provider” 11](#_Toc107907690)

[3.4. Cloud hardware 11](#_Toc107907691)

[3.5. Development and production platforms 12](#_Toc107907692)

[3.6. Storage 13](#_Toc107907693)

[3.6.1. Classes of storage 13](#_Toc107907694)

[3.6.2. Life Cycle between classes of Storage 14](#_Toc107907695)

[3.6.3. Define the TTL of your data to automate the cleaning 14](#_Toc107907696)

[3.6.4. Analyze the SLA of storage services of your provider 14](#_Toc107907697)

[3.6.5. Analyse the cost structure of your storage 14](#_Toc107907698)

[3.7. Cost aspect 15](#_Toc107907699)

[4. Software aspect 16](#_Toc107907700)

[4.1. Third party (part of) software 16](#_Toc107907701)

[4.1.1. Open-Source Software usage 16](#_Toc107907702)

[4.1.1.1. OSS presentation 16](#_Toc107907703)

[4.1.1.2. OSS Selection 17](#_Toc107907704)

[4.1.2. COTS & FOSS 19](#_Toc107907705)

[4.1.2.1. Selection 19](#_Toc107907706)

[4.1.2.2. Integration 20](#_Toc107907707)

[4.1.2.3. Stay "in control" of the integrated elements 20](#_Toc107907708)

[4.1.3. Software Provided over the Cloud 20](#_Toc107907709)

[4.1.3.1. Managed service usage 20](#_Toc107907710)

[4.1.3.2. Software as a Service (SaaS) 22](#_Toc107907711)

[4.2. Containers and orchestrations 22](#_Toc107907712)

[4.2.1. Containers 22](#_Toc107907713)

[4.2.2. Network 23](#_Toc107907714)

[4.2.3. Storage 23](#_Toc107907715)

[4.3.1. Cloud observability 24](#_Toc107907716)

[4.3.2. Kubernetes observability 25](#_Toc107907717)

[4.3.3. Application obervability 25](#_Toc107907718)

[4.3.3.1. Flow logs and traces exploitation 25](#_Toc107907719)

[4.3.4. Production Observability 27](#_Toc107907720)

[5. Architecture aspects 28](#_Toc107907721)

[5.1. Microservices 28](#_Toc107907722)

[5.2. Containers 28](#_Toc107907723)

[5.3. Stateless components 31](#_Toc107907724)

[5.4. Communication 31](#_Toc107907725)

[5.4.1. Web service: REST API interfaces 31](#_Toc107907726)

[5.4.2. Message broker 32](#_Toc107907727)

[6. Security aspects 33](#_Toc107907728)

[6.1. Identify sensitive elements 33](#_Toc107907729)

[6.2. Putting in place protections 33](#_Toc107907730)

[6.2.1. Virtual Private Cloud & security group 33](#_Toc107907731)

[6.2.2. Encryption in transit 34](#_Toc107907732)

[6.2.3. Encryption at rest 34](#_Toc107907733)

[6.2.4. VPC entry point 34](#_Toc107907734)

[6.2.5. Hybrid architecture 35](#_Toc107907735)

[6.2.6. Deny of Service 35](#_Toc107907736)

[6.3. Source code 35](#_Toc107907737)

[6.3.1. OSS & produced code 36](#_Toc107907738)

[6.3.2. COTS and FOSS 36](#_Toc107907739)

[7. Human and organisation aspect 38](#_Toc107907740)

[7.1. What is the cloud mindset? 38](#_Toc107907741)

[7.2. Organization culture needs to be ready (from management to technical) 38](#_Toc107907742)

[7.3. Technical aspect 39](#_Toc107907743)

[7.4. Be ready for more innovation 40](#_Toc107907744)

[7.5. Technological watch 40](#_Toc107907745)

[7.6. Automation 40](#_Toc107907746)

[7.6.1. Automation process 41](#_Toc107907747)

[7.6.2. Continuous Integration 43](#_Toc107907748)

[7.6.3. Continuous deployment 43](#_Toc107907749)

[7.6.4. Deployment strategies 44](#_Toc107907750)

[7.7. Human Agility 45](#_Toc107907751)

[7.7.1. DevOps not a tool 45](#_Toc107907752)

[7.7.2. Define improvement areas together 46](#_Toc107907753)

[7.7.3. Define and carry out sprints 46](#_Toc107907754)

[7.7.4. Jointly define baseline 46](#_Toc107907755)

[7.7.5. Documentation 46](#_Toc107907756)

[7.8. Quality, Security & Compliance auditing 47](#_Toc107907757)

[7.8.1. Development phase 48](#_Toc107907758)

[7.8.2. Architecture phase 50](#_Toc107907759)

[7.8.3. Maintenance phase 51](#_Toc107907760)

[Annex-1: Main’s recommendations 52](#_Toc107907761)

List of figures

[Figure 1: Clouds definition 10](#_Toc107907762)

[Figure 2: Dev, IVV and production platforms 13](#_Toc107907763)

[Figure 3: OSS ecosystem 16](#_Toc107907764)

[Figure 4: OSS Landscape 17](#_Toc107907765)

[Figure 5: CNCF software solutions 17](#_Toc107907766)

[Figure 6: OSS selection points 18](#_Toc107907767)

[Figure 7: Observability 24](#_Toc107907768)

[Figure 8: VM vs Containers 29](#_Toc107907769)

[Figure 9: Life cycle management 30](#_Toc107907770)

[Figure 10: Different ways to use containers 30](#_Toc107907771)

[Figure 11: Cloud Information System Stack 48](#_Toc107907772)

[Figure 12: Application Security Testing 49](#_Toc107907773)

[Figure 13: Model-driven deployment 50](#_Toc107907774)

[Figure 14: Security assessment service 51](#_Toc107907775)

List of TableS

[Table 1: Status sheet 2](#_Toc107907776)

[Table 2: Domain of interests 7](#_Toc107907777)

[Table 3: Glossary 9](#_Toc107907778)

[Table 4: Cloud recommendations 54](#_Toc107907779)

# Introduction

This document is a guideline for the development of a virtualized ground segment. It deals with three subjects:

* Cloud readiness,
* Cloud agnosticism,
* DevSecOps.

It is written in the scope of Domino-X project which aims, among other things, to **virtualize a ground segment**.

The cloud is, therefore, the deployment target of the project. The “***cloud readiness****”* advice in this document is intended to help achieve this objective. Then, the customers targeted by this project are export customers who will have specific deployment requirements (on their national clouds, on machines hosted locally...). It is therefore necessary to develop a product that is as “***agnostic****”* as possible, that is to say independent from the technical solutions implemented by the cloud providers. The other reason for agnosticism is the need to be able to use this study over a long period of time and therefore to be able to move towards future technologies not yet available.

Finally, DevOps (and “**DevSecOps**”) techniques are the modern way of developing software, particularly adapted to clouds, and allow multiple contributors to collaborate on a common project. However, the idea of Domino-X is to split the software bricks to several industries.

We assume that the manufacturers selected for the dominoes’ implementation already have cloud and DevOps experience and even formal implementation processes. This document is therefore not intended to be a collection of requirements but advises and/or common practices necessary for the realization and assembly of Dominoes.

## How this document was constructed

This first version is produced based on feedbacks from each company in the Domino-X consortium.

During a workshop, held on January 11, 2022, most of the companies in the consortium presented achievements carried out on the cloud and the lessons they have learned in terms of good practices.

Speakers of this workshop where:

* Airbus Defence and Space,
* Thales Alenia Space,
* Safran,
* CS group,
* Orange,
* Capgemini,
* Gisaïa,
* stack labs,
* CNES.

These topics were gathered and discussed by cloud experts from Thales Alenia Space, Airbus Defence and Space, CS group and Capgemini.

This document is the result of these exchanges.

## How this document is structured

The subjects of the three themes covered by the document (cloud readiness, cloud agnosticism and DevSecOps) continually overlap. It is therefore not possible to tackle the themes one after the other. We have therefore decided to deal the subjects one by one by indicating the theme or themes to which they are attached.

Some subjects are also labelled with a yellow star emphasizing the subject importance.

|  |  |  |
| --- | --- | --- |
| Cloud readiness |  |  |
| Cloud agnosticism |  |  |
| DevSecOps |  |  |

Table 2: Domain of interests

## Additional informations

During the proofreading phase of the document, the Gaia-X[[1]](#footnote-2) project was mentioned. This one aim to define a cloud on a European scale. The sources of this project were not used in the drafting of this document, but a curious reader will be able to complete his reading with the help of the productions of this project, in particular the Architecture document

# Glossary

|  |  |  |
| --- | --- | --- |
| Acronym | Meaning | Short explanation (optional) |
| ANSSI | Agence nationale de la sécurité des systèmes d'information | French service created on 7 July 2009 with responsibility for computer security |
| API | Application Programming Interface | Service or Software interface in order to communicate with it and exchange data or orders. |
| ASOC | Application Security Orchestration and Correlation |  |
| AST | Application Security Testing |  |
| AWS | Amazon Web Services |  |
| AZ | Availability Zone | Each Region has multiple, isolated locations known as Availability Zones. |
| CADU | Channel Access Data Unit | CCSDS transfer frames |
| CAPEX | Capital Expenditure | Capital used to acquire, upgrade, and maintain physical asset |
| CCA | Continuous Compliance Automation |  |
| CD | Continuous Deployment,  Continuous Delivery | See §7.6.3 “Continuous deployment” for difference |
| CI | Continuous Integration |  |
| CIS | Center for Internet Security |  |
| CLR | Common Language Runtime |  |
| CNCF | Cloud Native Computing Foundation |  |
| CNI | Container Network Interface | Standard for Kubernetes |
| COTS | Commercial Off-The-Shelf |  |
| CRI | Container Runtime Interface | Standard for Kubernetes |
| CSI | Container Storage Interface | Standard for Kubernetes |
| CSP | Cloud Service Provider |  |
| CVE | Common Vulnerabilities and Exposures |  |
| DAST | Dynamic Application Security Testing |  |
| DMZ | Demilitarized Zone |  |
| (D) DOS | (Distributed) Deny Of Service |  |
| EAR | Export Administration Regulations |  |
| EC | Export Control |  |
| ECCN | Export Control Classification Number |  |
| ECS | (Amazon) Elastic Container Services | Amazon fully managed container orchestration service |
| ECSS | European Cooperation for Space Standardization |  |
| ELK | Elasticsearch, Logstash and Kibana |  |
| FOSS | Free and Open-Source Software |  |
| FPGA | Field-Programmable Gate Array |  |
| GDPR | General Data Protection Regulation |  |
| HSM | Hardware Security Module |  |
| http | Hyper Text Transfer Protocol |  |
| IaaS | Infrastructure as a Service |  |
| IaC | Infrastructure as Code | Manage and provision data centres’ infrastructure through machine-readable definition files. |
| IAM | Identity and Access Management |  |
| IAST | Interactive Application Security Testing |  |
| ID | Identifier |  |
| IP | Internet Protocol |  |
| IPR | Intellectual Property Rights |  |
| IS | Information System |  |
| IT | Information Technology |  |
| IVV | Integration, Verification and Validation |  |
| JVM | Java Virtual Machine |  |
| K8s | Kubernetes |  |
| KPI | Key Performance Indicator |  |
| MoD | Ministry of Defence |  |
| MOM | Message oriented middleware |  |
| NFS | Network File System |  |
| NIST | National Institute of Standards and Technology |  |
| OEM | Original Equipment Manufacturer |  |
| OPEX | Operating Expense | Ongoing cost for running a system |
| OS | Operating System |  |
| OSS | Open-Source Software |  |
| PaaS | Platform as a Service |  |
| RAM | Random Access Memory |  |
| REST | Representational State Transfer | Communication principle based on web services. |
| RETEX | Retour d’Expérience (Feedback) |  |
| RFC | Request For Comments |  |
| RPO | Recovery Point Objective |  |
| RTO | Recovery Time Objective |  |
| S3 | Simple Storage Service | Amazon’s Object Storage |
| SaaS | Software as a Service | Software accessible via internet |
| SAST | Static Application Security Testing |  |
| SCA | Software Composition Analysis |  |
| SDC | Software-Defined Compute |  |
| SLA | Service Level Agreement |  |
| SSD | Solid State Drive |  |
| TTL | Time To Live |  |
| VCPU | Virtual Central Processing Unit |  |
| VLAN | Virtual Local Area Network |  |
| VPC | Virtual Private Cloud |  |

Table 3: Glossary

Summary: Main recommendations

This summary presents the main recommendations od this guideline. Details can be found in the dedicated chapters.

|  |  |  |
| --- | --- | --- |
| § | Theme | Recommendation |
| 3.2-a |  | Mainly deploy your dominoes in a single region in order to reduce communication latency |
| 3.2-b |  | Be redundant in multiple Availability Zones in order to be resilient to external outages |
| 3.3-a |  | Build dominoes that can be deployed on the cloud but also on an on-premises infrastructure |
| 3.3-b |  | Deployment strategy (on premises vs public) shall be studied at domino definition but also at bidding time |
| 3.4 |  | Be hardware agnostic. If that’s not possible document the case for future re-assessment |
| 3.5-a |  | Similar environments are requested on development, validation, and production platforms |
| 3.5-b |  | A pre-production platform is used on Prime contractor side (or by a dedicated consortium member) in order to perform Continuous Delivery without impact on production |
| 3.5-c |  | Deployment is performed by production maintenance team when they are ready to do so |
| 3.6.1-a |  | Select storage classes compatible with your needs in terms of access latency and performance |
| 3.6.1-b |  | Evaluate the need in term of redundancy of each kind of data |
| 3.6.2 |  | Data can be moved from one storage class to another according to the need in term of access frequency |
| 3.6.3 |  | Avoid an endless increase in stored data by defining retention periods for this data |
| 3.6.4 |  | Develop a data backup strategy depending of the managed services of the CSP |
| 3.6.5 |  | In the data class selection, price can also be a discrimination factor in function of cloud provider pricing policy |
| 3.6.5 |  | In the data class selection, price can also be a discrimination factor in function of cloud provider pricing policy |
| 4.1.1-a |  | Choice between supported open source or not (i.e. supported by a community without SLA) depends on global SLA project |
| 4.1.1-b |  | Select open source software referenced in the CNCF landscape |
| 4.1.1-c |  | Assess legal aspects of the open source (export control, usage authorisation, business domain…) |
| 4.1.2-a |  | Select commercial software from reliable manufacturer |
| 4.1.2-b |  | Reduce dependency to COTS/FOSS by architecture |
| 4.1.2-c |  | Maintain control of the COTS and FOSS in the application by implementing a validation procedure for the elements to be used before making them available |
| 4.1.3-a |  | Do not systematically reject managed services based on standardized product but perform cost/constraint studies to choose the best solution |
| 4.1.3-b |  | Do not bind hand and foot with a managed service but plan for a withdrawal from the design stage |
| 4.1.3-c |  | Understand what is in a managed service in terms of security dependency |
| 4.1.3-d |  | Do not systematically reject SaaS but perform cost/constraint studies to choose the best solution (preferred SaaS based on standards) |
| 4.2.2 |  | Improve Kubernetes network capabilities thanks to CNI standard in order to add features tailored to your need |
| 4.2.3-a |  | Improve Kubernetes storage capabilities thanks to CSI standard in order to add features tailored to your need |
| 4.2.3-b |  | CSI solution can be changed one by another if the needs evolve or if the performance are not the expected one |
| 4.3.1 |  | During all project phases, observability strategies need to be setup in order to check the functioning of the system against what is planned |
| 4.3.2 |  | Specific monitoring needs to be setup for Kubernetes observation. This monitoring survey the K8s infrastructure (Node, Pod, CPU…) but also the operation (deployments, failure, restart…) |
| 4.3.3-a |  | Centralise logs in common location in order to ease navigation between entries |
| 4.3.3-b |  | Add a trace ID to each entry in order to rebuild the causes and effects pathway in the processing of an action |
| 4.3.3-c |  | Identify the severity level of each log with a common gravity scale |
| 5.1-a |  | Microservices must have a limited functional perimeter to be easy to understand, develop, integrate, replace |
| 5.1-b |  | Microservices shall be scalable |
| 5.1-c |  | Each service should be designed to handle cases where other needed services are unreachable |
| 5.2-a |  | Apply OCI Format Specification to build a container image, especially the layers approach |
| 5.2-b |  | Select Container Engine following OCI Runtime Specifications |
| 5.2-c |  | Define a strategy of update for the Container Runtime including the underneath Operating System |
| 5.3 |  | Only use stateless components and justify when a component must be stateful |
| 5.4.1-a |  | Expose your APIs to the outside world in the form of REST APIs (with http ecosystem: security, load balancing…) |
| 5.4.1-b |  | Consider using REST protocol for synchronous communication between microservices |
| 5.4.2-a |  | Consider using message broker protocol for asynchronous communication between microservices |
| 5.4.2-b |  | Event messages help manage the load inside the system: "accept the task now and process it later" |
| 6.1 |  | Analyse your product in order to list all sensitive elements and define the appropriate means of protection |
| 6.2.2 |  | Encrypt all outgoing VPC communications (in transit) |
| 6.2.3 |  | Encrypt all data stored on storage media (at REST) |
| 6.2.4-a |  | Protect VPC entry points against unauthorized access |
| 6.2.4-b |  | Create a DMZ to protect inbound and outbound traffic |
| 6.2.5 |  | Considers hosting sensitive code or data on an on-premises cloud or on a sovereign cloud in order to increase those data access control |
| 6.2.6 |  | For all internet access of a Domino, DoS (and intrusion) preventive countermeasures must be put in place |
| 6.3.1 |  | Uses CI pipeline to run code check tools |
| 6.3.2-a |  | Perform regular analysis of COTS and FOSS for newly discovered vulnerability |
| 6.3.2-b |  | Use the more up-to-date COTS and integrate new version as soon as possible |
| 7.1-a |  | Use a cloud mind-set focussed on evolving environment (scaling, elasticity, evolving functional perimeter) |
| 7.1-b |  | Use continuous integration and deployment (and DevOps) techniques to accelerate the development phase |
| 7.2 |  | Do not consider cloud just as “technological means”. Good use of the cloud implies changes in project governance |
| 7.3 |  | The technological tools to be mastered to work on the cloud are numerous and complex, implying the need of significant know-how. Organizations have an important role to play in enabling the acquisition and dissemination of this knowledge |
| 7.4 |  | By offering access to almost unlimited resources, the cloud gives the opportunity to explore new technological support for future innovations |
| 7.5 |  | Perform regular technological watch on the cloud ecosystem to understand technical changes early |
| 7.6 |  | Automate processes so that they can be replayed frequently and in the same way |
| 7.6.2 |  | Perform Continuous integration in order to increase fast detection of anomalies |
| 7.6.3 |  | Perform Continuous Delivery/Deployment in order to regularly and easily release new version of the software |
| 7.6.4 |  | Apply a deployment strategy to avoid service interruption |
| 7.7.3 |  | Consider using Agile and DevSecOps methodologies |
| 7.7.5 |  | Take great care to keep documentation up to date |
| 7.8-a |  | Project needs to be compliant with security standards selected at the beginning of this one regarding National Space regulations |
| 7.8-b |  | Each phase of the project may need to implement specific aspects of these standards to ensure the safety of the overall project |
| 7.8.1 |  | Setup a process of content trust at each stage of development |
| 7.8.2 |  | Use MBSE approach and a modelling method adapted to the Cloud Environment (OASIS TOSCA). It participates to the portability of Cloud Services |
| 7.8.3 |  | Maintain the right level of security of the Cloud Environment with a security situation awareness real-time status using dedicated tools |

Table 4: Cloud recommendations

# Physical aspect

The goal of virtualization is to create applications by disregarding the material aspect to concentrate on the application aspect. The hardware being left to the responsibility of the cloud provider.

Nevertheless, we need to deal with a certain number of responsibilities related to the material aspects such as the geographical location of data and applications, the kind of platform involved in production chain, the storage policy…

## Definitions of cloud

**Cloud computing**: Cloud computing is the delivery of computing services – including servers, storage, databases, networking, software, analytics and intelligence – over the Internet.

Various cloud types exist:

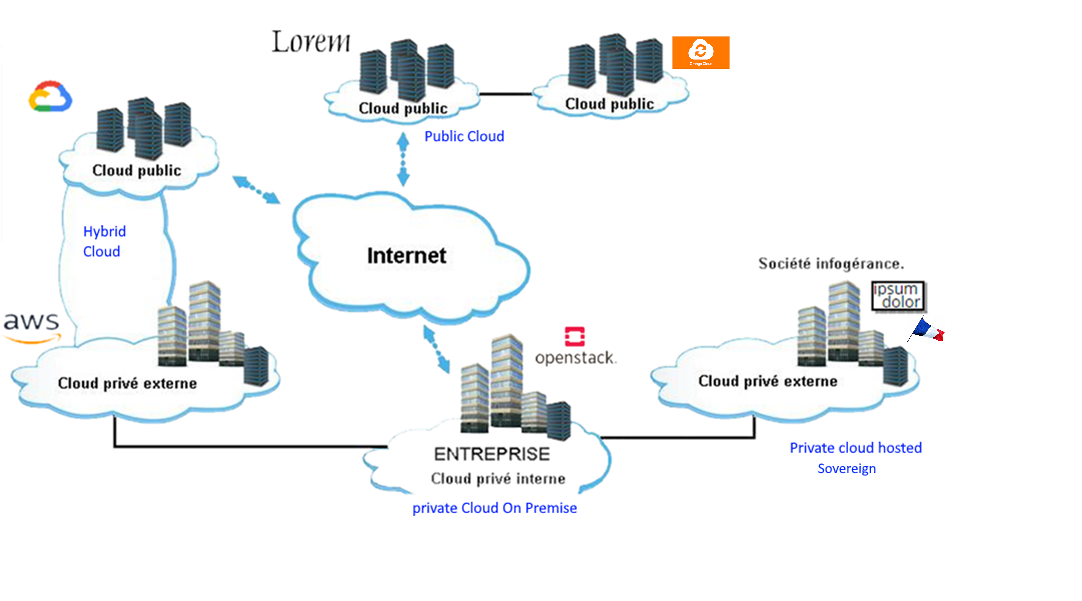


Figure 1: Clouds definition

**Public cloud**: Public clouds are owned and operated by a third-party cloud service provider, which delivers their computing resources, like servers and storage, over the Internet. Resources are shared between customers.

**Private cloud**: A private cloud refers to cloud computing resources used exclusively by a single business or organization. A private cloud can be physically located on the company’s on-site datacentre – **on premises** – or hosted by third-party – **dedicated hosted**.

**Hybrid cloud:** Hybrid clouds combine public and private clouds, bound together by technology that allows data and applications to be shared between them.

We can also define the **sovereign clouds**: A sovereign cloud essentially aims to maintain the sovereignty of data in all possible ways for any entity (country, region, enterprise, government, etc.). Thus, it requests that all data reside locally, the cloud is managed and governed locally, all data processing – including API calls –happen within the country/geography, the data are accessible only to residents of the same country, and the data are not accessible under foreign laws or from any outside geography. Such a cloud is certified by a governmental authority.

## Geographic repartition

The geographical location of clouds is generally divided into regions. A region is a geographical unit where datacentres (availability zones) are connected to each other by low latency networks. Usually, regions are independent of each other and do not share resources. Data can be exchanged between regions via Internet network but those exchanges induce cost so that it is advisable to limit them to the strict necessary.

In Domino-X context, several regions must be used:

* A region, close to the operator, where the ground segment will be deployed.
* And regions near the antennas where the virtual modems will be located.

Within a region, there are several datacentres that are independent of each other (i.e. a physical problem impacting one of them has no impact on the others). These centres are called Availability Zone (AZ) and communicate with each other through low latency network infrastructures.

It is possible to build a VLAN located on several AZ. For safety purpose, Domino-X system must be able to redeploy entirely on a subset of availability zones in case of failure of one of them (and conversely, return to normal when reactivating the defective centre). In the same way, the data must be duplicated on several AZ to be protected against disasters. These precautions make the system compliant to the “high availability” requirement.

|  |  |  |
| --- | --- | --- |
| 3.2-a |  | Mainly deploy your dominoes in a single region in order to reduce communication latency |
| 3.2-b |  | Be redundant in multiple Availability Zones in order to be resilient to external outages |

## Trade off on “On premises” vs “Cloud provider”

The aim of Domino-X is to virtualize a ground segment. A cloud deployment is a logical target but an on-premises deployment, is also possible. Several elements must be studied for deployment target selection:

* CAPEX/OPEX: On-premises deployment includes an initial expense to buy the servers and operating and maintenance costs. On the contrary, for a cloud deployment, those costs are smoothed on the duration of exploitation and proportional to the usage. There is a point of crossing of the 2 curves of total costs that can help in the choice of the solution.
* Elasticity: The advantage of the cloud is its ability, almost infinite, to add or remove resources. It allows scalable applications and faster processing. In the case of non-scalable applications (legacy or constant load) the need for hardware can be precisely sized and a cloud deployment is not necessarily the most relevant.
* Legal aspect: local legislation can introduce some restrictions to the cloud usage. They must be taken into account in order to remain in good standing. Moreover, the strategic interests of Domino-X usage can also be a brake on a deployment on the cloud, even if it is a private and sovereign one.

|  |  |  |
| --- | --- | --- |
| 3.3-a |  | Build dominoes that can be deployed on the cloud but also on an on-premises infrastructure |
| 3.3-b |  | Deployment strategy (on premises vs public) shall be studied at domino definition but also at bidding time |

## Cloud hardware

Domino-X is an export program to be deployed on a cloud selected by the customer. The characteristics of this cloud being unknown, it is advisable that the whole software be independent of the hardware layers on which they run.

Nevertheless, some processing may require specific hardware infrastructure (FPGA, graphical processors...). For example, current virtual modems are based on FPGA. In this specific case, where there is no alternative, the hardware constraints must be carefully studied by the system architect prior to the developments.

This study must include:

* the physical constraints that led to this choice (with enough details to allow re-evaluation in the future according to the evolution of the technology),
* a deployment plan indicating the target hardware and its inclusion with the virtual architecture in the cloud.

|  |  |  |
| --- | --- | --- |
| 3.4 |  | Be hardware agnostic. If that’s not possible document the case for future re-assessment. |

## Development and production platforms

In order to use DevOps process, at least 3 platforms are necessary:

* The **Dev platform** used by the developers to produce the code.
  + Usually, developers have advanced admin rights on their components in this platform, in order to tune the parameters for development our debugging purpose.
* The **IVV platform** used to integrate and validate newly produced code.
  + As the developers do not have direct access, the goal of this platform is to ensure that all components are deployed using the official CI/CD pipelines (ex: Infra-as-Code validation).
* The **Production platform** used to operate the system.
  + In spatial world, Continuous Deployment, on production platform, is not an acceptable method because of the risk of regression. An additional platform is inserted between IVV and production. Called **Pre-production platform**, it is used to validate and package a version of Domino-X before deployment in production.

All those platforms shall be similar in terms of usage (same IAM tools, log service, cluster…) in order to avoid divergent reactions of this environment on identical stimuli. An additional advantage is the possibility to reproduce an observed malfunction on a “out of production” environment in order to investigate.

Nevertheless, sizing differs with low capabilities on DEV/IVV platforms which can be smaller than target one:

* Physical: less VCPU, RAM, disk size…
* Deployment: no redundancy of the services.

It is also possible to have differences in COTS or FOSS version in order to avoid continual changes in dev or IVV platforms.

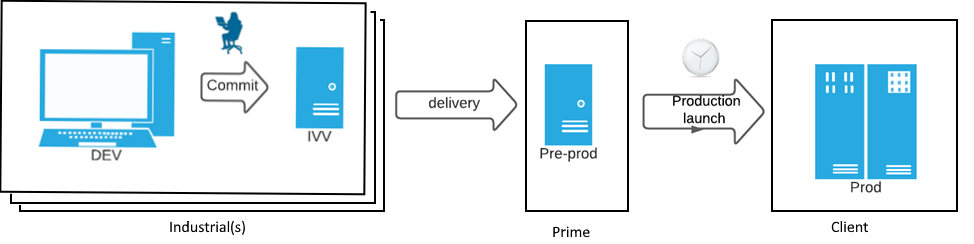


Figure 2: Dev, IVV and production platforms

The transition process is quite simple:

1. The company in charge of a Domino realisation owns Dev and IVV platform and handles them according to its own process.
2. The company domino is released to the pre-prod server, owned by the prime manufacturer of the ground segment, which can integrate all dominoes, perform tests and various product scans (quality, weakness…) thanks to its CI chains.
3. When a release is ready, the production platform operator deploys it on the production system.

|  |  |  |
| --- | --- | --- |
| 3.5-a |  | Similar environments are requested on development, validation, and production platforms |
| 3.5-b |  | A pre-production platform is used on Prime contractor side (or by a dedicated consortium member) in order to perform Continuous Delivery without impact on production |
| 3.5-c |  | Deployment is performed by production maintenance team when they are ready to do so |

## Storage

### Classes of storage

Several classes of storage are available in the different cloud providers (public or private). In your design you should chose the class corresponding to your usage, depending on the access speed you need.

From the hottest (low latency) to the coldest classes, you have:

* Physical or virtual **disks**: they are directly attached to your machines and have a very low latency.
  + Be aware that different types of disks are available, the SSD is usually the best in low latency access.
* Magnetic disks offer a good throughput in writing.
* Shared **network disks** accessible from several machines on protocol like NFS. It is typically used to share data in multi-node cluster of machines. Some cloud providers include automated space extension, but they usually have storage size limits.
* **S3** store for objects, equivalent to a “Web file system”. Each file is addressable with a standard REST protocol. This class of storage has a good ratio cost vs access time, and it should be used to store any data prior to disk classes when it is compatible with the processing constraints. Quite unlimited storage on public cloud.
* **Archive** storage for cold data, like S3 Glacier. This class of storage has a low cost to store data that is not processed but it requires a high latency, usually between seconds to hours, to get access to the data.

These classes of storage also define the redundancy of the data. Depending on the cloud provider, you may have different levels of redundancy per class of storage.

For instance, S3 on AWS is stored in 5 different disks in the same datacentre.

|  |  |  |
| --- | --- | --- |
| 3.6.1-a |  | Select storage classes compatible with your needs in terms of access latency and performance |
| 3.6.1-b |  | Evaluate the need in term of redundancy of each kind of data |

### Life Cycle between classes of Storage

To leverage the cost of the storage, you should design the life cycle of your data between the different classes of storage.

Example of life cycle:

S3 as input data for processing => copy to DISK for Processing by a dedicated virtual machine, new files created on DISK, merged in a big, compressed file then copy to S3 => S3 for download on demand => move to Archive storage with infrequent access (more than 2 months).

*Remark*: if using a single compressed file then you will have to read the full archive to extract any data. Please design the granularity of your files according to the data granularity needed to execute the processing.

|  |  |  |
| --- | --- | --- |
| 3.6.2 |  | Data can be moved from one storage class to another according to the need in term of access frequency |

### Define the TTL of your data to automate the cleaning

As some class of storage like the S3 are virtually infinite on a public cloud you need to define the “Time To Live” of your data.

The S3 storage services usually manage TTL rules you can set on your storage: for instance, on some logs of processing you can define 6 months of TTL, older data is going to be automatically deleted and you will not pay any more for its storage.

|  |  |  |
| --- | --- | --- |
| 3.6.3 |  | Avoid an endless increase in stored data by defining retention periods for this data |

### Analyze the SLA of storage services of your provider

Depending on the cloud provider, you may have different levels of SLA, in particular in the resiliency, i.e. the risk to lose the data.

Depending on this SLA and the type of data stored, the data backup/duplication strategy can be refined (for example: “Hot” data for processing, duplicated with a "warm" storage in another Availability Zone).

|  |  |  |
| --- | --- | --- |
| 3.6.4 |  | Develop a data backup strategy depending of the managed services of the CSP |

### Analyse the cost structure of your storage

Depending on the cloud provider you may pay for various aspects of the storage:

* Write queries, you pay per number of queries and data volume.
* Read queries (some of these are free depending on the zone where data are read from).
* Storage of data, you pay per data size and time of storage.

|  |  |  |
| --- | --- | --- |
| 3.6.5 |  | In the data class selection, price can also be a discrimination factor in function of cloud provider pricing policy |

## Cost aspect

Cost is an important aspect of Cloud Readiness because it is a transformation of CAPEX to OPEX (pay what you really use). Nevertheless, this subject will not be handled in this document as there is a dedicated work package for this subject in Domino-X scope (WP 3.2 - FINOPS).

However, we recommend regular monitoring of costs because a bad configuration on services billed by the unit of use can generate additions of small sums which, in the end, make significant amounts.

|  |  |  |
| --- | --- | --- |
| 3.7 |  | Regularly monitor your usage expenses |

# Software aspect

## Third party (part of) software

In current days, there is no more software development completely performed from scratch. Software relies on COTS or FOSS which are portions of code developed by third parties and integrated, as is, in the solution. That could be libraries to be integrated in the code or products to be executed separately in order to provide part of the services.

### COTS and OSS

#### OSS, FOSS, COTS presentation

In the world of IT and Cloud, both software sourcing approaches are currently used to provide IT services, needed to run the Cloud Infrastructure and also to provide Business applications with interfaces from the IaaS layer to the user interface. One is the use of proprietary software, we could call ‘Vendor locked’ software and the other is call ‘Open-Source’ or ‘OSS ‘software.

During the last years, rather driven by Cloud adoption and digitalisation transformation, a massive use of Open-Source software has been adopted especially by developers.

“Open source” is a model for the development and distribution of software providing access to source code and promotes community relationship and support of the technology.

Figure 3: OSS ecosystem

OSS is defined by a specific license scheme, enabled by broad collaboration among developers and users especially through Internet, and empowered by large and diverse communities that leverage open innovation principles. In 2022. Most of the time, an OSS is linked to a brand or a company.

Regarding the last figures provided by organization like Gartner or IDC, Open-source software is used within mission-critical IT workloads by more than 95% of IT organizations. Concerning the Space industries, OSS are systematically adopted to create, recode or transform Space applications, for Ground Information systems or for on-board software.

One of the promises of OSS is to trigger cost savings, flexibility and innovation benefits over third-party commercial alternatives.

On each layer of a Cloud Information System delivered on premises or provided by a Cloud Service Provider, the Cloud architect has the possibility to choose a proprietary or OSS software.

Most of ‘Cloud Ready’ solutions are referenced by the Cloud Native Foundation organization. This landscape is probably one of the right ways to identify solutions to assess.



Figure 4: OSS Landscape

However, it is important to underline that OSS and proprietary software are listed within the database of Cloud Native Computing Foundation. That is a proof Open Source usage is not a dogma in a Cloud Information System and may depend of the context. Open Source is not always the solution, especially regarding some licence constraints.

In 2022, within 1498 Cloud Native referenced software, only 508 are Open Source.

Figure 5: CNCF software solutions

Note: the acronym FOSS designate a **Free** Open Source Software. This financial quality does not imply any change compared to what is described in the present chapters

A Commercial Of-The Shelf designates a component manufactured in large series and not for a particular project. Basically it's the same as an OSS but the source code is the property of the editor and is not published on the internet.

#### OSS Selection

Obviously, not all software solutions are referenced in this inventory, that is why, in any cases, it is necessary to evaluate the suitability of open-source integration technology by assessing it against several criteria.

Some rules to make the OSS the right choice:

1. Cloud Native Landscape is a good start point.[[2]](#footnote-3)
2. Ensure that a vital community is supporting the product – check this every year or ensure you will rely on a software company. It is mandatory to check some metrics about the community supporting the OSS:
   1. Volume of contributors: based on information from repositories like ‘GitHub’.
   2. Spread of contributors: One or several organisations? Not a single company!
   3. Update Frequency: dynamicity of the interaction with community, patches ‘Beat rate’.
   4. Activity of the support: timeliness of answers and proposed solutions.
   5. Origin of the contributors / Support of Software Companies: Canonical / SUSE / Red Hat / Uber / Facebook…
   6. Promotion of the software.
3. Check the Intellectual Property Rights (IPR) and licenses especially if you sell or build Internet-facing products or if you apply for patent. Involve your procurement and your legal department or contract an external consultant if your company may not have the expertise.
4. Check the impact to operation: do you observe any issue e.g. out-of-memory with an OSS product?
5. Anticipate the training, especially for your operational team or your final customer.
6. Check the security flaws of an OSS regularly and address them. How secure is the OSS and the frequency of the CVE.
7. Discuss with all your developers your recommendations regularly – communicate about the existing and assessed solutions.
8. Check the usage of Open Source and address technical debt frequently – your security officer may ask you for fast action (e.g. Heartbleed)
9. Try to contribute to the Open-Source community injecting your competence or your feedback.

Figure 6: OSS selection points

‘Open-Source is not free’, this sentence could generate some misunderstanding. Indeed, most of the time, OSS is free to use but, in any cases, not free to support. On the one hand, we can rely on the support of OSS Software community to deliver the patch or the solution but without commitment. We talk about ‘best effort’ from the community. On the other hand, software companies involved on the OSS development commit on the support regarding a yearly subscription. We recommend to setup a dedicated contract (OEM Contract) to integrate OSS on your final product or to understand the SLA proposed by Software players.

Finally, the choice of the way to provide the OSS support is driven by the final customer requirements (901 government directive for instance) or by your maintenance contract. The only question you have to ask yourself: Am I ready to support myself the OSS?

‘Ready’ means:

* Ability to patch the source code.
* Provide the modification to the community.
* Commit on the SLA to provide and maintain the modification.
* Maintain the right skills.
* …

In addition, all those OSS are protected by a brand, under a trademark potentially with restrictions regarding some use case or business cases (defence end-use, peace-full, etc.). For this reason, the legal department has to be involved each time, having in mind that the legal and management risks associated with OSS are increasing in quantity, severity and complexity, implying difficulties to find the right skill to analyse legal aspect.

Most of the time ‘Open Source Software’ come from US, that is why they are controlled in term of Export Control[[3]](#footnote-4). Our recommendation is to involve the License and Compliance Department to check the EC compliance of OSS, focus on ECCN number (Export Control Classification Number, EAR99 or 5D992 are the preferred classification, the others could be more restrictive). Some information are available directly on the web site of the software providers:

* Apache[[4]](#footnote-5),
* Red Hat[[5]](#footnote-6),
* SUSE[[6]](#footnote-7).

|  |  |  |
| --- | --- | --- |
| 4.1.1-a |  | Choice between supported open source or not (i.e. supported by a community without SLA) depends on global SLA project |
| 4.1.1-b |  | Select an open source software referenced in the CNCF landscape |
| 4.1.1-c |  | Assess legal aspect of the open source (export control, usage authorisation, business domain…). |

#### COTS Selection

Third party software selection is always a strategic part of the development as development teams do not have the mastery of code nor of product roadmap. Selection criteria are basically the same than for open source (indeed lots of them are open source).

For commercial ones, you cannot rely on community for bug fixing or assistance in usage. So that the use of the "services" of the manufacturer is necessary. It is therefore necessary to check the SLA that the manufacturer imposes on itself as well as its robustness to ensure its durability over the entire life cycle of the domino.

|  |  |  |
| --- | --- | --- |
| 4.1.2-a |  | Select commercial software from reliable manufacturer |

#### COTS & OSS Integration

The decision to integrate a third party software (a COTS or a OSS) in the solution induces, de facto, a dependence on it. It is desirable to reduce this dependency as much as possible.

A good way to reduce dependency to a software is to select one compliant with standards, RFC-Standards[[7]](#footnote-8) or reference due to their market success.

It is also preferable to use these products through APIs that fully implement the standards presented above. The implementation tricks allowing to abstract from a standard are probably not portable to another similar product in case of need for replacement.

|  |  |  |
| --- | --- | --- |
| 4.1.2-b |  | Reduce dependency to COTS/FOSS by architecture |

#### Stay "in control" of the integrated elements

The control of COTS and FOSS introduced in a software is essential to its security and to its mastery.

For this, the project security officer shall audit it when it is introduced into the project, control the availability of updates, and regularly control the discovery of CVE (Common Vulnerabilities and Exposures). During this audit, the operator must also be aware that some third party software rely on other third party software (nested COTS or FOSS) which must also be controlled and mastered.

It is recommended to host COTS and FOSS artefacts in a common trusted repository hosted by the company. It is fully mastered by a company (or project) responsible, in charge of regularly updating software, performing regular scans to detect vulnerabilities (which can be done automatically by the CI/CD chain) and deciding whether to add (or quarantine) a COTS.

|  |  |  |
| --- | --- | --- |
| 4.1.2-c |  | Maintain control of the COTS and FOSS in the application by implementing a validation procedure for the elements to be used before making them available |

### Software Provided over the Cloud

Software as a Service (SaaS) and Managed Services are different as the first provides software that can be used over the cloud while the second one goes a step further by managing the software on the cloud.

#### Managed service usage

In the cloud context, managed services are tools made available to the application and on which it can rely to perform some of these functions.

Managed services can be found at every level of the solution. We can find:

* Support service as Virtual Private Cloud, firewall, IAM…
* IT services as S3 Storage, Kubernetes…
* Service intrinsically linked to the application such as database, communication broker…
* And so and so.

Generally, the cloud provider offers the major release versions of a product, performs security monitoring and applies security patches to these versions. Some specific operations (backup, database defragmentation…) can also be included in the service and handled as a managed service.

The advantage of managed services lies in the fact that they are directly usable and do not require maintenance, which significantly reduces the cost of development. In return those service can be customised or fully owned by the cloud provider, in which case it introduces a dependence link towards it. This dependence is at two levels:

1. As the service is provided by the cloud owner, it is not possible to ensure that it will be available unchanged until the end of the life of the system. Version upgrades or migrations can be imposed.
2. As the service is provided by the cloud owner, there is no guarantee that the same service will be rendered in the same way in another cloud. Migration may require user adaptations.

This does not mean that you should avoid managed services because their technical and financial advantages are real. We have selected the following criteria to guide the choice of a managed service:

* Trust in cloud provider: does it correctly deploy the managed services, manage them correctly and ensure a significant availability period?
* Maturity: as for COTS and FOSS selection, is the managed service has enough maturity?
* Sovereignty: a managed service can rely on several COTS developed by various entities. All those entities shall be known in order to verify the compatibilities with project rules (confidentiality, export control…).
* Feedback: college and Web feedbacks on the services is a way to discover advantages and disadvantages of the service.
* Cost:
  + The usage of a managed service replaces the development cost by a payment for the use.
  + Migration and withdrawal costs shall also be anticipated. Migration consists in evaluating the changes and adaptations required to switch to an equivalent managed service from another provider. Withdrawal is a complete change of technical solution.
  + Version upgrade: Even if the service provider guarantees a long availability of the service this will not necessarily cover the lifetime of the ground segment and will require upgrading the solution in the future.
* On the contrary, check that there is not a cloud agnostic COTS that fulfils the same service. For example, Terraform allows deploying on multiple clouds with the same configuration thanks to their notion of building block which 'agnosticises' the deployment configuration.

In our context of marketing to multiple clients, it is important to ensure that there is no strong link with any particular instance or managed service. This is in order to be able to adapt to the various deployment constraints. To do this, we will ensure:

* Not to use managed service based on cloud provider own product (such as a database) but rather on open-source technology that can be redeployed in case of withdrawal (with a development/packaging effort around this COTS);
* To use wrapper if your code needs to interact with the managed service.

|  |  |  |
| --- | --- | --- |
| 4.1.3-a |  | Do not systematically reject managed services based on standardized product but perform cost/constraint studies to choose the best solution |
| 4.1.3-b |  | Do not bind hand and foot with a managed service but plan for a withdrawal from the design stage |
| 4.1.3-c |  | Understand what is in a managed service in terms of security dependency |

#### Software as a Service (SaaS)

SaaS is a software distribution mean whereby a manufacturer can make a service available on its infrastructure (owned or in the cloud).

Advantages and usage condition are approximatively the same than for Managed service except that there is a remote infrastructure to be used which requires security studies as well as the consideration of more distant (and therefore potentially longer) network flows.

The additional advantage is to be directly on the manufacturer's site which can be accessed regardless of the cloud on which a domino is deployed.

|  |  |  |
| --- | --- | --- |
| 4.1.3-d |  | Do not systematically reject SaaS but perform cost/constraint studies to choose the best solution (preferred SaaS based on standards) |

## Containers and orchestrations

One aspect of virtualisation is based on containerisation and orchestration technologies (as opposed to the use of virtual machines).

* A **container**[[8]](#footnote-9) is a lightweight, standalone package that encapsulates a complete runtime environment including an application and its dependencies (libraries, binaries, and any additional configuration files), increasing an application’s portability, scalability, security, and agility.
* Container **orchestration**7 is the automation of much of the operational effort required to run containerized workloads and services. This includes a wide range of things software teams need to manage a container’s lifecycle, including provisioning, deployment, scaling (up and down), networking, load balancing and more.

At the time of writing, Kubernetes dominates the world of orchestration to the point of being considered as a standard. As a result, we will consider Kubernetes as a basis for the elements that follow. Keep in mind that there are alternatives (such as nomad by HashiCorp) and different distributions (Rancher, Red Hat OpenShift, VMware Tanzu...) each with their own advantages. The choice of technology is left to each company in charge of implementing a domino.

We cannot determine what will be the dominant technology when making dominoes. However, we recommend using tools certified by the Cloud Native Computing Foundation[[9]](#footnote-10).

### Containers

Containers rely on runtimes to execute their content. It is therefore appropriate to select a containerisation technology based on a standardised runtime and accepted market standard orchestrators.

The runtime ecosystem is vast, so it is not possible to list them all. In order to be able to operate a runtime, Kubernetes has defined a standard: **C**ontainer **R**untime **I**nterface (CRI), allowing communication between Kubelet and runtime.

The selection of a containerisation technology must take into account the runtime to be used and the existence (and the performance) of a CRI allowing the connection to Kubernetes.

For example:

* Containerd is the docker runtime and CRI-Containerd is the interface.
* Docker is also a runtime (high level runtime base on Containerd) and Docker shim the interface:
  + **Note** this interface is now deprecated.
* Open Container Initiative is an open standard and relies on CRI-O for the interface.
* Kata is another runtime which supports both CRI-O and CRI-Containerd.
* …

|  |  |  |
| --- | --- | --- |
| 4.2.1 |  | Select a container runtine based on Open Container Initiative Runtime Specification |

### Network

In a virtual environment a virtual network is necessary in order to allow containers to communicate with each other or with external applications. Kubernetes and the Cloud Native Computing Foundation have defined the **C**ontainer **N**etwork **I**nterface (CNI) which specifies the configuration of the container network interfaces. This specification works have allowed the creation of multiple network plugins (Calico, Cilium, Flannel, Cintiv…) to address different situations. It is up to the creator of a domino to select the most suitable plugins to fulfil these functions.

|  |  |  |
| --- | --- | --- |
| 4.2.2 |  | Improve Kubernetes network capabilities thanks to CNI standard in order to add features tailored to your need |

### Storage

In a virtual world, it is necessary to access physical resources for storage and data persistence. Here again there are multiple technology solutions to interface with containers (block storage, file storage...) and several ways to do it. Here again, Kubernetes allows choosing your solution using the Container Storage Interface (CSI) standard. The advantage of this standardization is to allow an easy change of solution if the need evolves. This change will be made by replacing the CSI component without impacting the product.

Many software solutions have been developed using this standard, which offers many choices to the developers of dominoes to select a solution.

|  |  |  |
| --- | --- | --- |
| 4.2.3-a |  | Improve Kubernetes storage capabilities thanks to CSI standard in order to add features tailored to your need |
| 4.2.3-b |  | CSI solution can be changed one by another if the needs evolve or if the performance are not the expected one |

* 1. Observability

If we refer to Wikipedia[[10]](#footnote-11), “in software systems, observability is the ability to collect data about program execution, internal states of modules, and communication between components”.

### Cloud observability

Observability is the new way of monitoring, especially in the ‘Cloud Native’ ecosystem. It is one of the features offered by an application or a system giving the possibility to be seen by other components.

In a context of multi-Cloud environments (including Hybrid-Cloud), such approach is mandatory due the underneath complexity of interactions between multiple systems.

To support the different phases of the Cloud information system lifecycle, the system has to implement a strategy, processes and tooling to capture status information from the different assets deployed within the system.

Even if the current way to drive a Cloud project is more focused on Agile approach particularly defined by several sprints, it looks like necessary to introduce ‘Observability’ very soon.

That information will be exploited at different stages:

* Engineering phase: to improve the architecture of the IS (Information System),
* Development phase: to identify issues during the development of a component,
* Integration phase: to setup correctly the different applications or to trace and feedback the potential issues,
* Exploitation phase: to setup the right operational dashboards needed to follow the good running of the IS,
* Security Monitoring phase: to identify non conformant behaviours of the IS,
* Obsolescence phase: to accurate the right view of the running architecture and elaborate the migration or transformation plan.

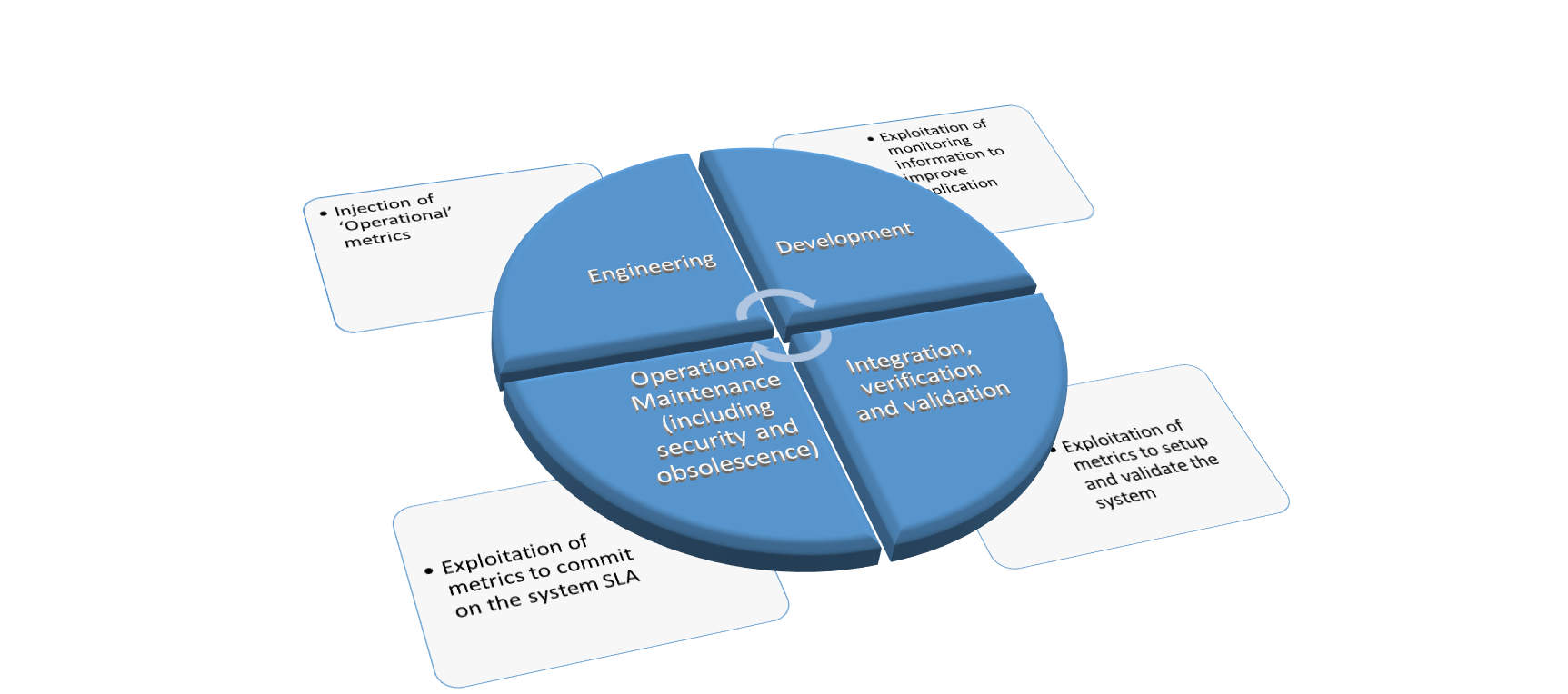


Figure 7: Observability

Ideally, to be ‘Observable’ an application or an asset has to propose several methods:

* Information ‘push’, sending information to a consumer,
* Information ‘pull’, allowing ‘questions’ about its behaviour,
* Methods have to be secured.

|  |  |  |
| --- | --- | --- |
| 4.3.1 |  | During all project phases, observability strategies need to be setup in order to check the functioning of the system against what is planned |

### Kubernetes observability

In the containerisation context, the container orchestrator have to produce KPI to be monitored in exploitation in order to ensure its good behaviour. Without going into application details, here are the main observables to control:

* Nodes: measure how many nodes are available and so determine if the cluster is well dimensioned.
* Pods: measure how many pods are running and compare this number to the expected one which allows to know if there are enough nodes to operate the system at the defined scale.
* Resources used: measure the computing resources used (vCPU, memory, bandwidth, disk utilisation). In order to help in decision to increase or decrease allocated resources.
* Application metrics are relative to business logic. Some of them could provide information on system needs. For example, a high number of CADU data indicates the needs of processing resources in order to handle them (and so the need to scale those containers).

In Kubernetes case, numerous probes are natively present in the solution allowing fine observation of the tool behaviour. Various tools also exist in order to exploit, synthesize, and display this information (Kubernetes Dashboard, Prometheus, Grafana, ELK…).

|  |  |  |
| --- | --- | --- |
| 4.3.2 |  | Specific monitoring needs to be setup for Kubernetes observation. This monitoring survey the K8s infrastructure (Node, Pod, CPU…) but also the operation (deployments, failure, restart…) |

### Application obervability

The application observability, also called monitoring, consists in writing, on a persistent medium, information on the functioning of the software. This information is written on an event mode (the software says it has to do / has done this).

Their purposes are double:

* To record the progress of the functions implemented in order to be able to analyse them in the future.
* To provide information for real time analysis of the software health and load thanks to automatic analysis of this logs for KPI computation. Those KPI are presented to operators via monitoring system.

Beyond the production of KPIs, application monitoring aims at understanding a system malfunction by locating the component and then the functionality at its source. For this, it is necessary that the system offers logs and a mean of navigating through them. This requires several techniques to be put in place.

#### Flow logs and traces exploitation

To find out what happened on the system, two main sources of information can be consulted:

* **Log-flows** record: it is a capture of all IP traffic going from/to the network interfaces of the VPC. Their study makes it possible to define the origin of the traffic and to work on the security rules (too or not restrictive enough).
* **Traces** record: they provide knowledge of what has happened inside a system.

Due to their modular nature, dominoes can be deployed on different machines, or even on different clouds. In order to facilitate the analysis of the logs, it is advisable to gather all the logs in a single and centralized log location.

This objective is not always achievable due to technical and/or financial problems. In this case it is necessary to define two levels of log location:

* A first one at global system level, in which each Domino will send a subset of logs allowing operators to have an overall view of the operation (alerts but also the logs necessary for KPIs and dashboards).
* A second one at domino level, in which each component of the domino will deposit the operating logs allowing fine investigations for this specific domino.

Whatever the location of the logs, it is necessary to have a query mechanism for exploring them by query language in order to be able to locate the information relating to the analysis operation in progress (bug investigation, KPI computation…).

|  |  |  |
| --- | --- | --- |
| 4.3.3-a |  | Centralise logs in common location in order to ease navigation between entries |

* + - 1. Activity tracking

In a microservices architecture, the processing of data may involve the implementation of several processes, distributed over several machines. In order to be able to monitor the functioning of this processing in its entirety, it is necessary to be able to track data throughout the whole processing chain.

For this, a unique ID is added to the data when it is created and travels with it from one micro-service to another. This ID is added to each log trace, which makes it possible to trace the overall processing of the data on the entire system.

A convention must be adopted when two data are combined to produce a single result or when, on the contrary, a data is split towards 2 different processing steps.

For example an AAAAA data (id x001) and a BB data (id y578) are combined into a CCCC data. This CCCC data can take a new id and a log line indicates that it is created from x001 and y578.

|  |  |  |
| --- | --- | --- |
| 4.3.3-b |  | Add a trace ID to each entry in order to rebuild the causes and effects pathway in the processing of an action. |

* + - 1. Hierarchy of logs

This point is fairly standard in software production.

Not all logs have the same severity or the same functional purpose. In order to filter quickly, it is necessary to prioritize them, from the creation using the following levels:

* **DEBUG**: log set up by the developer for his use, this level is generally not activated in production.
* **INFORMATION** level: these logs provide information on the overall operation of the system (configuration change, operator log…). They are not intended to indicate an anomaly but rather to give a view of the operating context in the event of an occurrence of a bug.
* **WARNING** level: detection of an operating anomaly that is not serious for the system because it is capable of restoring itself (cancellation of a command, operation outside the usual characteristics, etc.).
* **ERROR** level: operating anomaly requiring the creation of an investigation ticket in order to modify the operation of the software to remove the cause of the error.

**Remark:** Some coding conventions add error levels (notice, critical, etc.). This is completely acceptable provided that the definitions of these levels are explained in the coding rules and uniform for all dominoes.

**Note**: a warning on a software feature can lead to an error on another feature in the rest of the processing. It is advisable to keep the warning / error convention and not to presuppose that the warning will have a fatal consequence later. It is the investigation of the error which will allow to analyse the cause.

|  |  |  |
| --- | --- | --- |
| 4.3.3-c |  | Identify the severity level of each log with a common gravity scale |

* + - 1. Log format

As a conclusion, we can recall that a log is constituted of a textual message with addition of several metadata such as:

* Log level,
* Data identification (trace id),
* Original domino with indication on its instance,
* Time stamp.

**Caution**: Logs can be an interesting source of data leakage for an external organisation (putting a password in a log would not normally occur to a developer but other sensitive data should be considered e.g. the AOI of the acquisition reflects an interest in that area). It is therefore advisable to consider logs as sensitive data and to think about what to log or even to set up protections such as a double log circuit (one for normal data and one, protected, for sensitive data).

Example of logbook:

*INFO x001 17/06/22-08:11:23 Dx-1 Creation of image AAAA*

*…*

*DEBUG x001 17/06/22-08:11:25 Dx-1 check of image AAAA OK*

*…*

*DEBUG y578 17/06/22-08:11:23 Dx-2 acquisition of order request*

*DEBUG x001 17/06/22-09:00:23 Dx-2 Reception of image AAAA*

*…*

*INFO z666 17/06/22/09:25:24 Dx-2 Analyse of image AAAA (x001) according to instruction of order request (y578)*

*INFO z666 17/06/22/09:25:24 Dx-3 Storage of enhanced product*

With those kinds of logs, we can trace lifecycle of enhanced product: “When”, ”Where” and “With what” it had been generated. We also understand the necessity of a centralized log location because 3 dominoes (Dx-1, Dx-2 and Dx-3) are involved.

### Production Observability

In order to finish with this subject, a quick note issued from feedback.

An interesting monitoring is that of the production chain (CI-CD). Indeed, it can happen that a task of such a chain is deactivated for one reason or another (for example a code quality gate failed and is deactivated for the time of the correction, so as not to block production).

It is necessary to have a view on this temporary state to keep it in memory and not to forget it.

# Architecture aspects

Cloud readiness mainly benefits from appropriate modern design relying on:

* Microservices architecture,
* Use of containerisation,
* As much as possible stateless components,
* Standard Web API.

These design elements are described in this paragraph.

## Microservices

Microservices architecture relies on loosely coupled and collaborating services. It can be seen as an extension of UNIX motto: “Make each program do one thing well”.

It has several advantages, relevant even when not related to cloud deployment:

* Allowing developing and testing each microservice independently by a different team,
* Allowing deploying each microservice independently,
* Language agnostic, so that each team can develop microservices in their preferred language (even though it is a good practice to have only a reasonable number of different languages),
* Replaceable, if the service is not reliable enough, or does not reach expected performances.

In addition, this architecture also provides more “cloud-related” benefits:

* It can allow scaling in or out, when using stateless services in charge of specific tasks (cf. below),
* It can allow using some managed services provided by cloud providers, when relying on standard interfaces for those services,
* It can even allow discoverability of services.

Nevertheless, microservices architecture also comes with some drawbacks, to have in mind when choosing this kind of architecture:

* It is distributed, which can have some main impacts:
  + some data are exchanged between the services, which can have a cost depending on the cloud provider, the amount of data exchanged and the frequency of the exchanges,
  + distributed systems are more complex to maintain due to synchronisation of messages and remote calls,
  + the previous point can also lead to a reduction of the global performance of the system,
* As a result, consistency is harder to achieve and maintain,
* Each service should not fail even when a service it needs is not reachable.

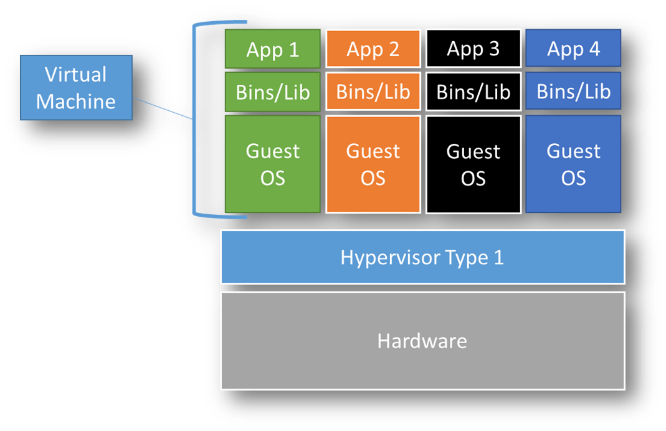
|  |  |  |
| --- | --- | --- |
| 5.1-a |  | Microservices must have a limited functional perimeter to be easy to understand, develop, integrate, replace |
| 5.1-b |  | Microservices shall be scalable |
| 5.1-c |  | Each service should be designed to handle cases where other needed services are unreachable |

## Containers

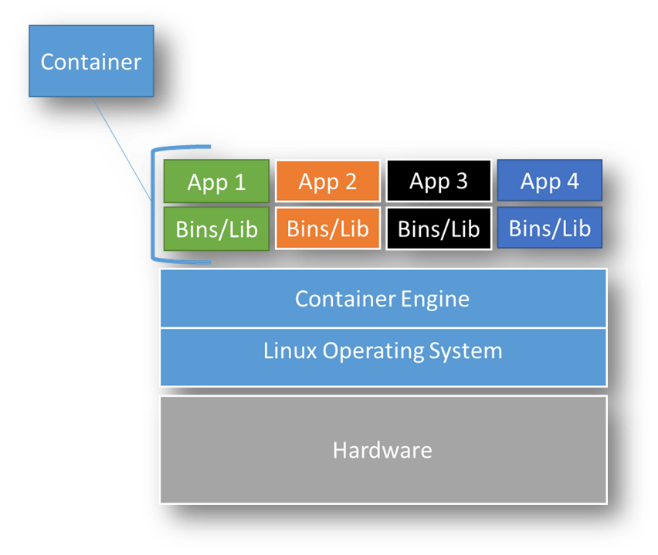
Docker has popularized the use of containers to propose a new way of developing and deploying an application. Regarding Docker’s definition, a container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.

Containers and Virtual Machines share the same objectives which are to isolate an application and the underneath dependencies into a self-contained envelop that can be ran more or less anywhere.

Containers technologies vs Virtual Machines technologies:



Applications running on Virtual machine need an Operating System implying the additional life-cycle management for the entire scope of the OS.



Applications running on containers rely on their own libraries when they are not available on the underneath Operating Systems.

Figure 8: VM vs Containers

Container technology proposes a new way to increase the agnosticism to the underneath infrastructure:

1. Operating System level even if the border between Linux and Microsoft Windows is not very clear.
2. Hardware level, due to the large scope of hardware compatibility with Linux Operating Systems.

But, to support this agnosticism, it is mandatory to fix a strategy in terms of maintenance:

* + 1. Hardware / Operating Systems choices:
       - If the project needs a low commitment in terms of support, a full OSS approach could be a good choice but the support will be provided with a ‘best effort’ way.
       - If the project requires a SLA in terms of maintenance, the trio ‘Hardware’, ‘Operating Systems’, ‘Container Engine’ compatibility has to be respected and maintained during the lifecycle of the project.
    2. Container Engine has to stick to **OCI Runtime Specification**.
    3. Container Image Format and build have to follow **OCI Format specifications**.
    4. To support and ease the obsolescence phase of the hardware, it is recommended to adopt a strategy of software update, especially for the Operating System and by the way of the Container Engine.

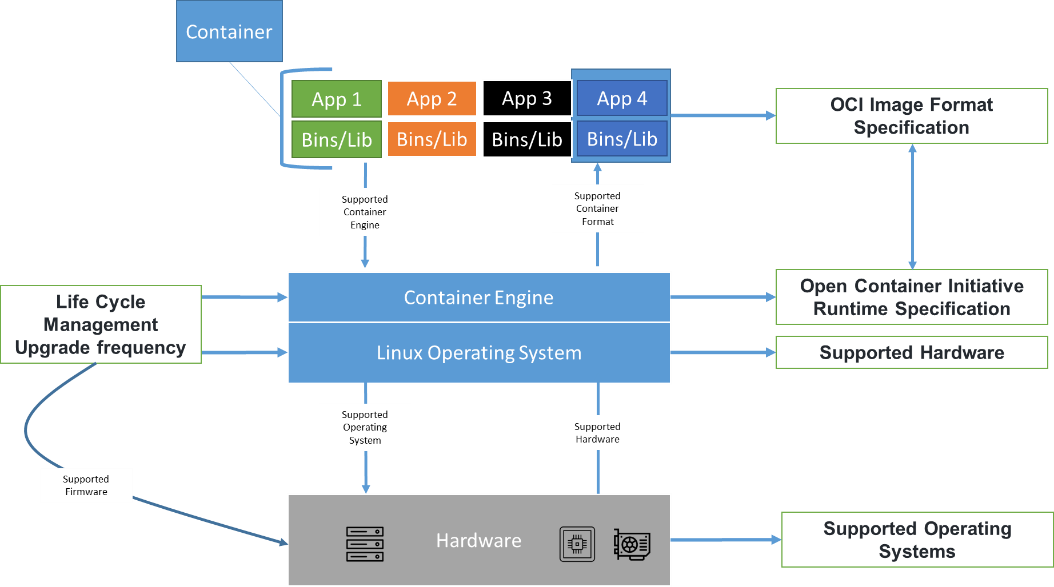


Figure 9: Life cycle management

There are different ways to use containers:

* + 1. On a single node, the Linux Operating Systems offers the Container Engine to support the execution of multiple containers.
    2. On a Container orchestration engine across a cluster of nodes, enabling the high availability of the workloads.

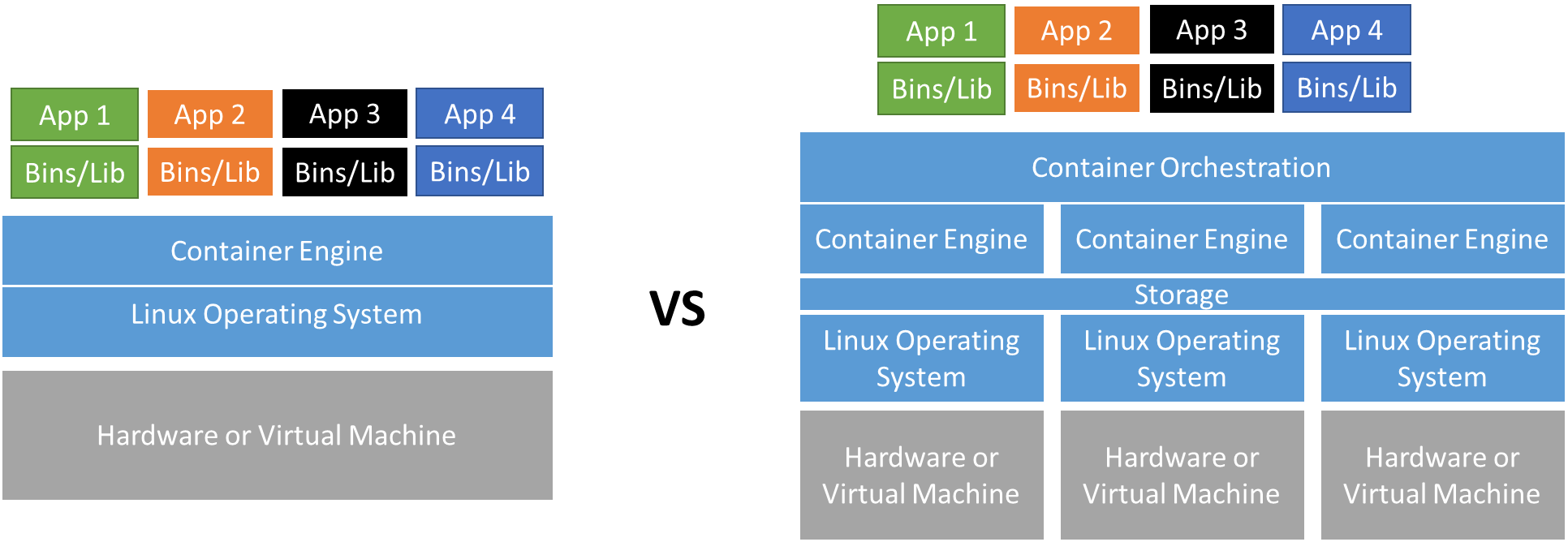


Figure 10: Different ways to use containers

You can use independently the different ways to run containers, depending of the use-cases.

|  |  |  |
| --- | --- | --- |
| 5.2-a |  | Apply OCI Format Specification to build a container image, especially the layers approach |
| 5.2-b |  | Select Container Engine following OCI Runtime Specifications |
| 5.2-c |  | Define a strategy of update for the Container Runtime including the underneath Operating System |

## Stateless components

Stateless components or services do not retain information from one request to another. It allows implementing one of the main DevOps motto: “cattle not pets”, meaning that the loss of a service or a host has no impact on the system: the service is relaunched on another host, and as it has no need of information to achieve its task, the system is fully operational.

The stateless feature of a component is of paramount importance for cloud systems, designed to be scalable. Indeed, if by design a component is made stateless and invoked with a given short task to accomplish, then if the number of tasks increases in the future, it can be easily handled just by increasing the number of instances.

This is particularly important in microservices architecture, where there are, by essence, a lot of services. Of course, at some point some stateful services will be needed, such as database (at least IAM), but, as a rule of thumb, each service should be stateless, and each service needing to be stateful shall duly justify why.

|  |  |  |
| --- | --- | --- |
| 5.3 |  | Only use stateless components and justify when a component must be stateful |

## Communication

For Cloud Readiness, another very important aspect is communication between components. As mentioned above, microservices architecture are well adapted to cloud. But they are composed of dozens of services that need to communicate at some point. These services being decentralised, communication must be done via Web channels.

Two major means are generally used on the cloud:

* Web services,
* Message brokers.

### Web service: REST API interfaces

The use of a Web protocol has many advantages. For instance, it allows to address a service using its domain, without worrying about its actual IP address (another service will handle this, a Load Balancer or an Ingress Controller).

REST API usage is particularly recommended for:

* Synchronous requests/responses based on http(s) req/rep principle.
* Point to point message: the communication is an exchanged between two IP addresses.
* Time limitation between request and response due to the blocking effect of the protocol (that could also be considered to a limitation).
* Standardisation such as open API allows to document all messages.
* A vast and robust securing ecosystem base on well-known http(s) protocol which is the main Web technology (firewall, authentication, load balancer…).

|  |  |  |
| --- | --- | --- |
| 5.4.1-a |  | Expose your APIs to the outside world in the form of REST APIs (with http ecosystem: security, load balancing…) |
| 5.4.1-b |  | Consider using REST protocol for synchronous communication between microservices |

### Message broker

The principle of a message broker is to rely on a communication agent, located between the components, in charge of handling the messages. The producers send messages to the broker who keeps them and makes them available to consumers. These latter come explicitly to recover it.

The advantages of message broker are:

* Asynchronous communication: each component takes care of its task without worrying about what happens before or after. The topics make it possible to distribute or smooth the load. And notifications make it possible to inform anyone who wants to know about the occurrence of an event.
* Loose coupling: the components do not know each other and can be tested independently.
* Diffusion: if multiple clients are interested in a message type, they can all get a copy of it after the broker.
* Easier scalability: each instance of a microservice, pops a message in a common queue. Next microservice will take the next one.
* Recovery support:
  + If the broker fails, it can be restarted and re-populated in messages by replaying the event log.
  + If a client fails or is not available, the message remains in the broker waiting for the customer to be available.
* Better resilience and error management by relying on MOM features (such as dead letter queues, poison queues, persistence, etc.).

|  |  |  |
| --- | --- | --- |
| 5.4.2-a |  | Consider using message broker protocol for asynchronous communication between microservices |
| 5.4.2-b |  | Event messages help manage the load inside the system: "accept the task now and process it later" |

# Security aspects

The security aspect is an important element of cloud application deployment. It is in this perspective that we talk about Dev**Sec**Ops which is a customisation of the DevOps cycle with introduction of security element at each step of the DevOps infinite loop.



## Identify sensitive elements

The first step to securing your application in the cloud is to **analyse** all elements (data, algorithm, rendered services, etc.) in order to determine the possibility, of each of them, to be deployed on the cloud and if protection measures should be considered. Multiple analytical perspectives must be taken into consideration:

* Legal aspects: depending on the legislation of the countries concerned by the project, provisions must be made for installation in the cloud.
* Industrial aspects: the industrial secret and the specific know-how must be protected:
  + Algorithms and data are of course part of these secrets but also technical or procedural know-how of the company.
  + Network infrastructure or credential.
  + Cost aspect of the solution can also be considered as sensitive data.
* Operating aspects:
  + The domino architecture, as planned to date, guesses external interfaces (inter-domino communication) and internal interfaces (intra-domino communication). These interfaces are more or less attackable depending on their exposure on open networks and require appropriate protectives measures.
  + Of course, like for all kinds of deployment, unauthorized persons should be prevented from accessing to software and stored data.

Even if not dedicated to cloud, **export control** aspect shall be considered. This aspect can induce requirement on cloud provider selection (location of the datacentres, nationality of the company providing the cloud services, nationality of employees who can access physical machines).

|  |  |  |
| --- | --- | --- |
| 6.1 |  | Analyse your product in order to list all sensitive elements and define the appropriate means of protection |

## Putting in place protections

Once the sensitive elements listed, it is possible to define protection strategy in order to counter the identified risks.

The strategies most often implemented are listed in the chapters here after.

### Virtual Private Cloud & security group

A **VPC** can be seen as a subnet structure isolated from its environment by a **security group**.

Inside the VPC, all communications are safe and cannot be intercepted or altered. Incoming or outgoing communications circulate on public channels likely to be accessible by unauthorized actors.

All those communications transit via the security group which acts as a firewall.

In this document, we assume that a domino can be fully deployed in the VPC, so that you only have to worry about inter dominoes communication. If it is not the case, protection shall also apply to the dominoes’ internal communications.

### Encryption in transit

A good practice is to **encrypt all communication on “external network”** (i.e. communication outside the VPC). It can be achieved by several means:

* Installation of VPN between each VPC,
* Encryption of all messages before leaving the VPC and deciphering them when reaching the target VPC,
* Over-encryption of the data, when this one is sensitive. The data is then encrypted by its producer and remains encrypted until reception by the consumer (this does not remove the message encryption seen above. As a result, there is a double encryption in transit).

|  |  |  |
| --- | --- | --- |
| 6.2.2 |  | Encrypt all outgoing VPC communications (in transit) |

### Encryption at rest

**Encryption at rest** consists in encrypting data when they are stored on the hard disk in order to protect them against read or modification by unauthorized actors.

There are several levels of privacy of the encryption key depending on the privacy of the data and its criticality:

* Key managed by the storage service of the cloud provider.
* Key managed by a key manager from the cloud provider.
* Key managed by dedicated hardware, hardware security module (HSM). To ensure the segregation of role in key management they must not be provided by the cloud provider: this type of key ensures a maximum level of data privacy.

Sometime encryption of data in memory is recommended but that depends on the hardware and services provided by the cloud provider. However, for the purpose of genericity, this operation must be deployed independently from the domino (i.e. during the configuration of the infrastructure) and must not impact performance.

|  |  |  |
| --- | --- | --- |
| 6.2.3 |  | Encrypt all data stored on storage media (at REST) |

### VPC entry point

It is easy to guess that the entry point of the VPC is a critical point in terms of security. It is necessary to deploy a certain number of measures to secure it:

* Firewall: a basic functionality,
* Gateway: a basic functionality,
* Reverse-proxy: a basic functionality,
* **I**dentity and **A**ccess **M**anagement (IAM):
  + set up authentication for all humans having to interact with the system to limit access to this detailed list of people,
  + set up a “trustability” strategy between the dominoes so that each domino is able to identify the source of a message before accepting it (or rejecting it).

During the deployment of the infrastructure, it is advisable to study the services of the cloud provider. Indeed, some of them, which do not impact the functionality of the applications to be deployed, offer services that increase security, such as:

* Analysis of unusual access: capability of detecting that a user starts making requests that he has never made. Or that a machine replaces a human user to request the service...
* Various strategies against Deny of Service attacks, in order to detect, counter and notify operator while trying to keep the service accessible.

|  |  |  |
| --- | --- | --- |
| 6.2.4-a |  | Protect VPC entry points against unauthorized access |
| 6.2.4-b |  | Create a DMZ to protect inbound and outbound traffic |

### Hybrid architecture

Finally, the choice of a hybrid architecture should be considered for data that is too sensitive. With an **on-premises deployment** for very sensitive data and a private or even sovereign cloud deployment for sensitive data.

But this strategy is technically not always possible and some operations require the use of a given cloud (for example the antenna as a service requires the deployment of virtualized modem at the foot of the antenna and therefore in the cloud of the service). In this case, the recommendation is to limit, to what is strictly necessary, processing in this cloud and to repatriate it as quickly as possible to a "safer" location.

|  |  |  |
| --- | --- | --- |
| 6.2.5 |  | Considers hosting sensitive code or data on an on-premises cloud or on a sovereign cloud in order to increase those data access control |

### Deny of Service

Operating a system on the cloud leads to a greater risk of a denial-of-service attack (DOS). These attacks aim at reducing the availability of a service by subjecting it to deliberately abnormal input data (large volume of data, slow sending of requests, etc.).

Some protections have already been implemented through the gateway, reverse proxy and IAM. Others can be defined by architecture of the domino interface:

* Usage of pull mechanism: the domino is at the origin of the data recovery that he manages at his own pace. This avoids a surge which ends up saturating the operation.
* Scalable infrastructure to provide more resources capable of absorbing increased traffic. That includes the network throughput.

At last, managed service of the cloud provider shall be studied. Cloud providers usually have some tools in order to detect DOS attack and trash aggressive messages. The important aspect of these tools is that they do not adhere to the dominoes so that they can be selected by provider without changing our solution.

|  |  |  |
| --- | --- | --- |
| 6.2.6 |  | For all Internet access of a Domino, DoS (and intrusion) preventive countermeasures must be put in place |

## Source code

In previous chapters we saw active countermeasures to be taken for security. We must be aware that measures taken, from the design phase, also make it possible to reduce the exposition surface of our software and participate in the security of the ground segment.

### OSS & produced code

As far as the code is concerned, the misuse of certain commands can allow malicious users to recover confidential information (password and others), to take control of the process, or even to inject their own code into the system. It is therefore necessary to detect, as soon as possible, the weaknesses of the **code** by **scanning** it. In order to systematize this control and to perform it as soon as possible, good practices recommend integrating into the **continuous integration** chain the execution of analysis tools configured with security coding standards. This CI check has the advantage of including the developers as soon as possible in the activity of securing the code rather than postponing this until the end of the project.

It is also advisable to set up coding rules to harden, simplify and homogenise the code. The continuous integration pipeline can also control the respect of these rules.

|  |  |  |
| --- | --- | --- |
| 6.3.1 |  | Uses CI pipeline to run code check tools |

### COTS and FOSS

COTS and FOSS integration has been presented in §4.1.2, in this section, we will focus on the security aspect of those components.

For COTS/FOSS, developers do not have the code mastery and are therefore dependent on developed third parties, which is not safe. Worse, this code being generally spread, numerous actors can discover weakness and communicate on them.

When adding a COTS or FOSS to the application baseline, a proper analysis must be done for each against dependencies vulnerabilities:

* this analysis must be planned and estimated when selecting the package from the shelf.
* integrate – if possible – bots or scripts executed in the code repository to verify all dependencies.
* if a package is not valid, update its version to a valid version, otherwise the package must be replaced.

We can recall some rules or good practices regarding development and software packaging:

* consider using the least COTS and FOSS components.
* to reduce the attack angle, avoid using too many dependencies nested together.
* have the smallest Docker image with the least components to fulfil the required services, do not keep unused dependencies.
* software editor of each external component must be known and validated internally.
* use pre-defined images if they are security compliant.
* avoid dead code.
* source code must be scanned frequently for vulnerabilities.
* adopt proven development pattern such as gateway pattern for client communication with microservices.

DevOps engineer must be able to quickly assess the validity and security of the packages and components used in the code and provide measures to avoid vulnerabilities during the development phase as well as production phase. That is why dependency upgrade must be applied on a fix time basis (such as once a week) and then CI/CD chain must be executed after such upgrade (including based OS image) for non-regression tests.

In order to simplify this activity, it is recommended to control used COTS thanks to local repository, managed by security officer. Developers can only use COTS from this repository and need to ask for the addition of new ones. This allows, for security, to make preliminary analysis for the introduction of a new COTS and to issue any recommendations for use.

Then, during all life of the product, it is necessary to conduct a **watch activity** on the communication[[11]](#footnote-12) about **detected flaws** and to react if one of them concerns one of the COTS used. A cybersecurity scan for vulnerability detection of an application environment must be scheduled on weekly basis or after each build and can occurs on a Docker image including based OS image (instead of executing it on real platform such as PROD).

|  |  |  |
| --- | --- | --- |
| 6.3.2-a |  | Perform regular analysis of COTS and FOSS for newly discovered vulnerability |
| 6.3.2-b |  | Use the more up-to-date COTS and integrate new version as soon as possible |

# Human and organisation aspect

Moving to cloud has an important impact on the human aspect and in the project organisation. This is mainly due to the change of paradigm induced by the much simpler access to many resources and technologies. It increases the possibilities but also increases the technical complexity.

## What is the cloud mindset?

From a financial perspective, a "Cloud Mind-set"[[12]](#footnote-13) enables the organization to shift from CAPEX to OPEX. For IT, a “Cloud Mind-set” means being able to move from the expensive maintenance of hardware, software, and services, to a hybrid cost-efficient version of IaaS, PaaS, and SaaS.

One of the common characteristics of the cloud methodology is that you must be willing to experience continuous innovation and regularly provision new features while effortlessly scaling in the future. Still, the development of cloud-native solutions comes along with a change in the way we work.

Here are a few high-level pointers to keep the Cloud Mentality in the foreground and ensure success during your migration:

* Design purpose-built systems with little or no overlap with other systems.
* Plan each tier for scaling and elasticity.
* Rigid, intertwined designs do not work well in the cloud.
* Avoid technology partners and vendors that sell products and services in the cloud based on the “engineer mentality”, that is by just trying to find a way to add new features and applications by fitting it into existing infrastructure.
* Avoid using infrastructure that does not offer scaling and/or elasticity.

Cloud scaling that is present in many services – such as compute, database, storage, containers – is an import criterion when moving to cloud as it also reduces the effort usually spent on trying to solve performance issues.

* But by enabling cloud scaling, it may also have import cost overhead so it must be properly adjusted to not consume useless resources.

Scaling is not the only answer, one way to reduce cost and / or to improve performance is to make sure that the applications code that is implemented in the cloud (packaged installed on compute system, serverless functions, container tasks) is optimized in a way to guarantee best performance and to consume less cloud services.

* By ending with such mastery of the code this will result of mastering the OPEX. Mastering the application code requires *ad-hoc* algorithm, monitoring of resources and FinOps.

|  |  |  |
| --- | --- | --- |
| 7.1-a |  | Use a cloud mind-set focussed on evolving environment (scaling, elasticity, evolving functional perimeter) |
| 7.1-b |  | Use continuous integration and deployment (and DevOps) techniques to accelerate the development phase |

## Organization culture needs to be ready (from management to technical)

The people, culture and change leadership domain is critical to establishing your organization’s cloud readiness and implementing a migration at scale. The impact of the cloud will be felt across your entire organization and will significantly affect, and be affected by, your organizational culture. Understanding these cultural implications, your company's receptivity to change, prior change successes and failures, organizational communication patterns to date, organizational structure, and level of executive sponsorship, commitment, and alignment, are all important elements of building a successful approach to cloud adoption. Your organization must have a critical mass of people with production experience in the cloud providers you will work with, established operational processes, and a leadership team dedicated to mobilizing the appropriate resources and leading teams through the many organizational and transformational challenges presented over the course of a large-scale migration effort.

To prepare for an enterprise migration, your organization must have a clear understanding of the potential issues a cloud project has raised in the past or by checking with other services or companies for such feedback. This will make sure to estimate in advance what is needed to be successful.

The cloud providers provide lots of services to manage the cloud complexity: for example, AWS proposes Control Tower to manage the creation of the Landing Zone, or AWS Organization to manage the different cloud environments. In the area of Infrastructure as Code (IaC), cloud providers have dedicated services and in this area one can also leverage the third-party tool like Ansible or Terraform in order to be able to reduce complexity when moving to several cloud platforms.

Note: AWS propose a complete guideline for above concerns: Accelerating cloud adoption through culture, change, and leadership[[13]](#footnote-14).

|  |  |  |
| --- | --- | --- |
| 7.2 |  | Do not consider cloud just as “technological means”. Good use of the cloud implies changes in project governance |

## Technical aspect

Being a cloud expert requires new skillset and lots of time to achieve this.

Organization needs to ramp-up with many new services and technologies when moving to cloud. It must necessarily have a plan for constituting dedicated team who will be involved probably full-time using cloud ecosystems.

Important skill set must be built for each cloud contributor with import set of new tools like Kubernetes for containerization or Terraform for IaC and not only the architecture must be understood, but also the APIs. Other areas such as DevOps or FinOps have also their own set of tools for which the cloud architects need to be properly trained.

The technical people involved in the cloud need to shift into different skills as they will not work on many infrastructure tasks as they would do on premises, but instead they will need broader knowledge to integrate all the cloud services together (API, Security...). In order to help, organization shall provide technical support from experts to address the project set up and the very specialized questions/problems raised by developments. It shall provide basic training to level up knowledge on the required technologies.

|  |  |  |
| --- | --- | --- |
| 7.3 |  | The technological tools to be mastered to work on the cloud are numerous and complex, implying the need of significant know-how. Organizations have an important role to play in enabling the acquisition and dissemination of this knowledge |

## Be ready for more innovation

The Cloud Accelerates Innovation. This Forbes article[[14]](#footnote-15) presents the value of the cloud as an innovation catalyst. It presents the five ways the cloud accelerates innovation, and it also presents seven mantras for realizing true cloud value for innovation.

Here is a concrete example of how companies can innovate by using proper services in the cloud: project team can organize “Innovation Sprint” as a time during project implementation phase dedicated to innovation (like the sprint 5 of Safe), such as to take full advantage of the most innovated solution cloud services can provide to solve project tasks.

A business can ensure its cloud resources run efficiently without human activities overheads increase by using “managed” cloud services, which are partial or complete management and control of a client's cloud services, including migration, maintenance, and optimization.

|  |  |  |
| --- | --- | --- |
| 7.4 |  | By offering access to almost unlimited resources, the cloud gives the opportunity to explore new technological support for future innovations |

## Technological watch

When operating a platform over a cloud provider, this cloud provider delivers regularly new services or update existing services. A regular technology watch (every 6 months at least) is necessary to stay up to date with the evolution of the provider's ecosystem.

The person in charge of this watching activity must identify:

* New managed services that are eligible to your platform (ex: S3 API for storage, Kubernetes managed service).
* Update of features in existing services you already use.
* Improvement of reliability of existing services so you can re-evaluate to use it or not.

|  |  |  |
| --- | --- | --- |
| 7.5 |  | Perform regular technological watch on the cloud ecosystem to understand technical changes early |

## Automation

DevOps involves more frequent changes to code and more dynamic infrastructure use when compared to traditional, manual management strategies.

* DevOps teams are likely familiar with using automation to speed the development process. It reduces feedback loops and eliminates repetitive processes. Security teams also typically recognize that automation increases their efficiency by improving incident response and lessening the need for manual work like policy setting.
* By shifting left to DevSecOps, organizations ensure that automation improves both development and security, from the use of auto-completed code all the way through to identification of high-risk threats.
* There are certainly other considerations toward shifting toward DevSecOps[[15]](#footnote-16). For example, there are specific tools and technologies that inherently improve the security aspect of development — both from infrastructure and application standpoints. Read more about these technology factors, including ways to reduce DevSecOps complexity, in this article[[16]](#footnote-17).
* That said, collaboration and automation underpin DevSecOps and become increasingly important as organizations move into multi-cloud environments where complexity heightens. The more teams can talk to each other and the more they can lean on trusted processes, the less risk for errors and incidents.

How to Move from DevOps to DevSecOps:

* Many aspects of DevSecOps can seem overwhelming, especially if your organization faces shortages of skilled developers, security experts, or cloud operations personnel.
* Take an incremental approach to addressing DevSecOps.
* That starts by identifying baseline familiarity with collaboration technologies, automation, infrastructure as code, containers and more.
* Then map a journey to help the organization smoothly transition into DevSecOps processes.
* Along the way, get support & coaching, develop one-click deployment pipelines, and offer templates that ease enterprise adoption.

|  |  |  |
| --- | --- | --- |
| 7.6 |  | Automate processes so that they can be replayed frequently and in the same way |

### Automation process

Automation is the use of technology to perform tasks with reduced human assistance. DevOps automation speeds up how an idea goes from development to deployment:

* DevOps relies on automating routine operational tasks and standardizing environments across an app’s lifecycle. Containers can offer standardized environments, but you need a platform to manage them that also offers built-in automations and support for any infrastructure.
* Automate all phases (build, test, deploy, run) and integrate trust verification at each step using cryptography.
* Integrate Security, FinOps (Finance), Green in your DevOps strategy:
  + Security: embed security controls into your CI/CD process. Make it an integral part and consider it as part of the security feedback loop. Integrate your CI/CD toolkit with security solutions like for example:
    - Azure Sentinel, native to the cloud.
    - Snyk for dependencies vulnerabilities checking in your code.
    - Gitlab Ultimate for advanced security capabilities.
  + FinOps, or cloud financial operations, allow businesses to track the cost of resources, down to a single action, and in a perfect scenario, align it with engineering and development.
  + Green DevOps is the term used for doing DevOps the right way: adopt the 6 C’s of green DevOps (see this [article](https://jaxenter.com/how-green-is-your-devops-131461.html)[[17]](#footnote-18) for details):
    - Continuous business planning.
    - Collaborative development.
    - Continuous testing.
    - Continuous release and deployment.
    - Continuous monitoring.
    - Customer feedback and optimization.
* Define the tool to be used by all teams.
* Define how environments are shared and used by all the teams.
* Define at least one test of “good functioning” that can be run “as is” on all platforms.
* Consider the possibility of using a specific tool to launch tests before committing for example, using sonar-light which allows you to have a report on the quality of the code.

The code commit using git triggering function can handle various automated process:

* Automatic CI/CD chain trigger on development environment.
* Early integration regression detection.

Usage of state-of-the-art tools is recommended for pipelines, source management, deployment process, for the targeted environments:

* Tools and methods are at the free choice of manufacturers, however the KPIs must be common to all.
* Tools are not imposed but it must be used to justify the KPIs.
* The development teams use their own tools independently of the other teams, but must also respect the KPIs (comment rate, number of cycles, 0 critical CVE in a Docker image, etc.) and must provide proof of compliance with the requirements.
* The microservice approach can impose constraints on environments: size of Docker images, start-up time.

An efficient and optimal model must define the CI/CD workflow.

For example, using Github flow[[18]](#footnote-19) process from *Github*:

* Create a branch: short, descriptive branch name enables your collaborators to see ongoing work at a glance.
* Make your changes, commit and push to your branch, give each commit a descriptive message: you back up your work to remote storage. This means that you can access your work from any device.
* Create a pull request to ask collaborators for feedback on your changes. Pull request review is so valuable that some repositories require an approving review before pull requests can be merged. Include a summary of the changes and what problem they solve.
* Address review comments: reviewers should leave questions, comments, and suggestions. Reviewers can comment on the whole pull request or add comments to specific lines.
* Merge your pull request: once your pull request is approved, merge your pull request. This will automatically merge your branch so that your changes appear on the default branch. GitHub will tell you if your pull request has conflicts that must be resolved before merging. Branch protection settings may block merging if your pull request does not meet certain requirements.
* Delete your branch: after you merge your pull request, delete your branch. This indicates that the work on the branch is complete and prevents you or others from accidentally using old branches.

Or using the simple git flow process[[19]](#footnote-20) from *Atlassian* for continuous delivery. The prerequisite is that you and your team are at least a little bit acquainted with git and have good knowledge of the rebase command (replay commits one by one on top of a branch to preserve the order of change-sets) in the two forms (interactive and not). This simple workflow has two guiding principles:

* master is always production-like and deployable.
* rebase during feature development, explicit (non-fast-forward) merge when done.

Pulling change-sets using rebase rewrites the history of the branch you are working on and keeps your changes on top.

Armed with these guiding principles let’s breakdown the seven steps of the simple git flow process[[20]](#footnote-21):

* Start by pulling down the latest changes from master.
* Branch off to isolate the feature or bug-fix work in a branch.
* Now you can work on the feature.
* To keep your feature branch fresh and up to date with the latest changes in master, use rebase.
* When ready for feedback push your branch remotely and create a pull request.
* Perform a final rebase clean-up after the pull request has been approved.
* When development is complete record an explicit merge.

Cloud providers propose reference architecture regarding automation and CI / CD:

* AWS “Building end-to-end AWS DevSecOps CI/CD pipeline with open source SCA, SAST and DAST tools”[[21]](#footnote-22).

### Continuous Integration

The “CI” for Continuous Integration process consists in the following steps:

* Developers push the code to a code repository as often as they want.
  + GitLab / GitHub / AWS CodeCommit / Bitbucket / etc.
* A testing / build server checks the code as soon as it is pushed, it gets the code, performs a build and passes integration tests.
  + GitLab pipelines / AWS CodeBuild / Jenkins CI / etc.
* The developer gets feedback about the tests and checks that have passed / failed.

CI allows to find bugs early, and then fix bugs. It delivers faster as the code is tested, and the deployment is performed more often. With this process, developers should not be blocked during development phase.

* Automatize as much as possible component and IVV tests by the use of CI chain: Gitlab CI / Jenkins…

In the case of a project with multiple development teams, it is recommended that all teams deliver a code compatible with client (or prime contractor) pipeline tools and process. This one must be able to re-run certain stages of the pipelines or run its own pipelines to do specific actions on the delivered code.

|  |  |  |
| --- | --- | --- |
| 7.6.2 |  | Perform Continuous integration in order to increase fast detection of anomalies |

### Continuous deployment

The “CD” for Continuous Delivery or Continuous Deployment process ensures that the software can be released reliably whenever needed. Also deployments happen often and are quick.

* That usually means automated deployment.
  + Gitlab CD / AWS CodeDeploy / Jenkins CD / Spinnaker/ etc…

Some distinctions can be made between Continuous Delivery or Continuous Deployment process:

* Continuous Delivery:
  + Ability to deploy often using automation.
  + May involve a manual step to “approve” a deployment.
  + The deployment itself is still automated and repeated!
* Continuous Deployment:
  + Full automation, every code change is deployed all the way to production.
  + No manual intervention of approvals.
  + Possibility to deploy a single micro-service and let with whole remaining unchanged.

In space IT industry, it is up to the operator to trigger it for production deployment.

Following guidance can be made for CD process definition:

* Automatize as much as possible deployment from scratch or upgrade (automatic trigger or manual trigger) using CD chain.
* Idempotent deployments (same result if triggered several times on different environment) and easy rollbacks.
* Applications automated deployment (based production line platform).
* Use DevOps tools for deployment such as Gitops, Jenkins, etc.
* Infrastructure as Code is a must have to automate the deployment process:
  + Use Terraform for infrastructure deployment (or similar tools).
  + Drive your IaC with a CI/CD (Gitlab CI, Argo CD…).
* Kubernetes deployment tool is a must have to automate container services deployment:
  + Use Ansible for Kubernetes services deployment (container, Helm).

The code deployment strategy (full or partial) depends on the modified code, the application of the strategy to be used is the responsibility of the human (development, production, validation strategy).

|  |  |  |
| --- | --- | --- |
| 7.6.3 |  | Perform Continuous Delivery/Deployment in order to regularly and easily release new version of the software |

### Deployment strategies

In addition to selecting the right tools to update your application code and supporting infrastructure, implementing the right deployment processes is a critical part of a complete, well-functioning deployment solution. The deployment processes that you choose to update your application can depend on your desired balance of control, speed, cost, risk tolerance, and other factors. The deployment process must be defined at the beginning of the project because there is multiple impact on IT, deployment process and architecture:

* It is necessary to know the target environment to perform the configuration: This is somewhat at odds with the agnostic objective of Domino-X but there is a point in the deployment process where the target infrastructure must be considered. The use of IaC techniques can reduce the workload but one must be aware of the limitations of these tools:
  + they are not automatically adaptable from one cloud to another (adaptations are needed in some areas. For example, if the configuration of the number of vCPUs is fairly standardised this is not the case for storage solutions which require adjustment from one cloud to another).
* **Blue-green** Deployments: strategy in which you create two separate, but identical environments. One environment (blue) is running the current application version and one environment (green) is running the new application version. Once testing has been completed on the green environment, live application traffic is directed to the green environment and the blue environment is deprecated:
  + Zero downtime and release facility.
  + Create a new “stage” environment and deploy v2 there.
  + The new environment (green) can be validated independently and rolled back if issues exist increase application availability and reduce deployment risk by simplifying the rollback process if a deployment fails.
  + Example: some AWS deployment services support blue/green deployment strategies including Elastic Beanstalk, OpsWorks, CloudFormation, CodeDeploy, and Amazon ECS.
* **Rolling** Deployments: strategy that slowly replaces previous versions of an application with new versions of an application by completely replacing the infrastructure on which the application is running:
  + For example, in a rolling deployment in Amazon ECS, containers running previous versions of the application will be replaced one-by-one with containers running new versions of the application.
  + Faster than a blue/green deployment; however, unlike blue/green there is no environment isolation between the old and new application versions. This allows rolling deployments to complete more quickly, but also increases risks and complicates the process of rollback if a deployment fails.
  + Example: some AWS deployment services support rolling deployment strategies including Elastic Beanstalk, OpsWorks, CloudFormation and ECS.
* The **Canary** Deployment pattern is similar to a rolling deployment in that the IT team makes the new release available to some users before others. However, the canary technique targets certain users to receive access to the new application version, rather than certain servers.
* **In-Place** Deployments: strategy that updates the application version without replacing any infrastructure components: the previous version of the application on each compute resource is stopped, the latest application is installed, and the new version of the application is started and validated. This allows application deployments to proceed with minimal disturbance to underlying infrastructure.
  + deploy application without creating new infrastructure. However, the availability of your application can be affected during these deployments. This approach minimizes infrastructure costs and management overhead associated with creating new resources.
  + Example: some AWS deployment services support rolling deployment strategies including Elastic Beanstalk, OpsWorks, CloudDeploy.

|  |  |  |
| --- | --- | --- |
| 7.6.4 |  | Apply a deployment strategy to avoid service interruption |

## Human Agility

Combining security and agility in project organization can be tedious. Through the Covid pandemic, we have seen more virtual collaboration within IT industry, and security has become integral to protecting businesses and their customers.

The human organization of the team is, of course, the responsibility of the structure in charge of development, possibly framed by standards such as the ECSS.

This chapter aims to highlight what has been reported as good practice by the members of the consortium.

### DevOps not a tool

DevOps should be considered a culture. It is more about people and how they work together than about technology or using a specific set of tools:

* Agile transformation can help change from management to technology.
* Working with native agility tools is an important choice where integration between different tools is available (Gitlab, Jira, etc.).
* Integration of the components on different environments and the unit tests shall be automatized.

### Define improvement areas together

You can define and group work items by team, product, or feature area. Then you need to group work into sprints, milestones, or other event-specific or time-related period.

### Define and carry out sprints

Agile DevOps practice can be driven with agreed methodology such as:

* Define the operating rules for the conduct of agile meetings.
* Define some dashboards within the CI/CD pipeline to measure KPIs for each delivery.
* Create and view customised Agile dashboards with Kanban metrics and/or Scrum metrics.

CI / CD toolchain should be configured to have a more driven integration flow:

* A commit is triggered, and the pipeline is executed such as build, deploy on a testing environment, automatic tests, and results that are sent to each project member.
* Nightly build can help optimize the availability of the environment.
* Source code audit involving analysing the code in depth to identify vulnerabilities or security improvements.
* CI / CD process (called “IVV”) is triggered for each new Domino-X module delivered on development environment. Several pipelines are available:
  + A Pipeline task performs generic control of the delivery (naming, rules, etc.).
  + A build pipeline executed at night on released version of the module to deploy the code and to perform additional vulnerability tests.

All developers are involved, and they directly see the results.

|  |  |  |
| --- | --- | --- |
| 7.7.3 |  | Considers to uses Agile and DevSecOps methodologies |

### Jointly define baseline

Reports and documentation must be shared with all parties concerned. To expect a reaction as soon as possible if a problem persists in the resolution of a problem, a request for validation or informal feedback must be made for each documentation.

Regarding the management of the “technical debts”, which are the items related to poor conception or design, code refactoring, COTS, etc.:

* They are often pushed back between sprints.
* The risk is to have a serious blocking point.

Do not continually push back, the technical debt should be considered and included in roadmap as soon as possible.

### Documentation

Documentation shall be kept up to date at each cycle. At each sprint it is important to have up-to-date documentation consistent with the existing one.

|  |  |  |
| --- | --- | --- |
| 7.7.5 |  | Take great care to keep documentation up to date |

## Quality, Security & Compliance auditing

When adopting Agile & DevOps methods and tools, one of the major challenges is to fit with security and compliances requirements while supporting continuous integration and delivery workflows in a Cloud Native environment.

Secured, robust and trusted software is key to a modern Cloud information system, and is increasingly required by boards, regulators and developers.

Securing an application is not only a process to identify vulnerabilities, it is also a question of code quality during the continuous integration / continuous delivery phase. However, regarding the number of compliances & rules, in the future, we will have to deal with a huge of verifications.

Regarding some external feedbacks, some key challenges have to be taken into account:

* Traditional compliance practices are incompatible with continuous software delivery processes, leading to slower delivery and unexpected remediation work.
* DevOps approach and especially use of services ‘Cloud Native Ready’ are most of the time uncontrolled (‘Docker Hub’-like savage download).
* Implement DevOps methods without technical governance and without accounting security and compliance requirements & tooling often causes increase of costs for a project especially if the security & compliance alignment has to be done at the end of the project.

That’s the reason to include some kinds of audit phase during the development and integration phase. Depending of the domain, some compliances have to take into account:

* Sarbanes-Oxley Act (SOX);
* General Data Protection Regulation (GDPR);
* Service Organization Controls (SOC 2);
* National Institute of Standards and Technology (NIST) standards;
* Health Insurance Portability and Accountability Act (HIPAA);
* Federal Risk and Authorization Management Program (FedRAMP);
* The Payment Card Industry Data Security Standard (PCI DSS);
* Federal Information Security Management Act (FISMA);
* Family Educational Rights and Privacy Act (FERPA);
* Basel or Gramm-Leach-Bliley Act (GLBA);
* California Consumer Privacy Act (CCPA);
* Authority to Operate (ATO);
* Center for Internet Security (CIS);
* Security Technical Guides from DISA;
* ANSSI Recommendations or Cloud Labels as SecNumCloud;
* French MoD Security Requirements;
* Export Control Regulations;
* Sustainability Regulations;
* Environmental Regulations.

Those additional verifications will participate to a ‘**DevSecOps**’ approach instead of traditional ‘**DevOps**’ method.

In addition, with the perspective of adopting a ‘Zero Trust’ architecture involving the removal of implicit trust of the Information Systems functions or services, developed or provided on the shelf, it appears the need to add audit security & compliance at each stage of services building. Zero trust replaces implicit trust with continuously assessed, explicitly calculated adaptive trust.

More generally, systematization of compliance audit is called ‘Continuous Compliance Automation’: CCA enables ‘Cloud Native projects’ to integrate compliance into all phases of the delivery pipeline and consistently enforce compliance policies without sacrificing operational agility.

Those audits have to improve the security level, and the trust level of the different Cloud Native applications developed and used within a Cloud Information system.

To cover all the stacks of the Cloud Information System, each layer has to implement auditing processes, necessary to trust all components used on a Cloud Infrastructure:

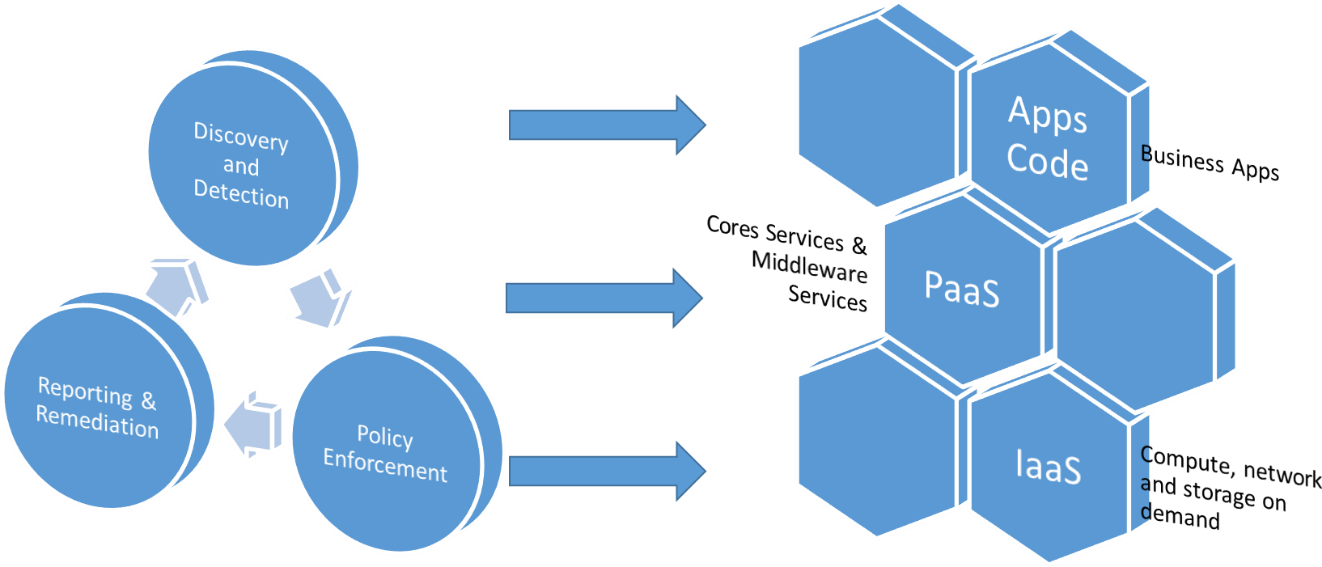


Figure 11: Cloud Information System Stack

|  |  |  |
| --- | --- | --- |
| 7.8-a |  | Project needs to be compliant with security standards selected at the beginning of this one regarding National Space regulations |
| 7.8-b |  | Each phase of the project may need to implement specific aspects of these standards to ensure the safety of the overall project |

### Development phase

Compliance’s commitment, like security and quality, have to be injected from the upstream project, during the development phase.

Some tools are available to support such compliance: security and quality audit tools could be used ‘on premises’ (Dark Site mode) or ‘SaaS’ but in the Space Industry domain, and more generally ‘Industrial’ domains, an on-premises deployment is mandatory, to protect the Industrial heritage in France (must be the rule for any country).

Our recommendation is to adopt an ‘Application Security Testing (AST)’ approach within the development process and environment.

AST tools offer different types of testing:

Figure 12: Application Security Testing

* Static AST (SAST) analyses an application’s source, bytecode or binary code for security vulnerabilities, typically during the programming and/or testing phases of the Software Development Life Cycle (SDLC).
* Dynamic AST (DAST) analyses applications in their running (i.e. dynamic) state during testing or operational phases. DAST simulates attacks against an application (typically Web-enabled applications, but increasingly, APIs as well), analyses the application’s reactions and, thus, determines whether it is vulnerable.
* Interactive AST (IAST) instruments a running application (e.g. via the Java Virtual Machine [JVM] or the .NET Common Language Runtime [CLR]) and examines its operation to identify vulnerabilities. Most implementations are considered passive, in that they rely on other application testing to create activity.
* Software Composition Analysis (SCA) is used to identify open-source and, less frequently, commercial components in use in an application. From this, known security vulnerabilities, potential licensing concerns and operational risks can be identified.

Traditional AST toolsets may not fully test the developed APIs, leading to the requirement for specialized tools and capabilities. That means it could be necessary to use additional tools having the capability to discover APIs in both development and production environments and test API source code, as well as the ability to ingest recorded traffic or API definitions to support the testing of a running API.

**Application Security Orchestration and Correlation (ASOC):** ASOC tools ease software vulnerability testing and remediation by automating workflows and processing findings. They automate security testing within and across development life cycles and projects, while ingesting data from multiple sources. ASOC tools correlate and analyse findings to centralize efforts for easier interpretation, sorting and remediation. They act as a management and orchestration layer between application development and security testing.

**Container security:** Container security scanning examines container images, or a fully instantiated container before deployment, for security issues. Container security tools focus on a variety of tasks, including configuration hardening and vulnerability assessment tasks. Tools also scan for the presence of secrets, such as hard-coded credentials or authentication keys. Container security scanning tools may operate as part of the application deployment process, or be integrated with container repositories, so security assessments can be performed as images are stored for future use.

**Developer enablement:** Developer enablement tools and features support developers and members of the engineering team in their efforts to create secure code. These tools focus primarily on security training and vulnerability remediation guidance — in stand-alone form or integrated into the development environment.

**Fuzzing:** Fuzz testing relies on providing random, malformed or unexpected input to a program to identify potential security vulnerabilities — e.g., application crashes or abnormal behaviour, memory leaks or buffer overflows, or other results leaving the program in an indeterminate state. Fuzzing, sometimes called nondeterministic testing, can be used with most types of programs, although it is particularly useful with systems that rely on a significant amount of input processing (e.g. Web applications and services, APIs).

**Infrastructure as Code (IaC) testing:** IaC is the process of creation, provisioning and configuration of Software-Defined Compute (SDC), network and storage infrastructure as source code. IaC security testing tools help ensure conformance with common configuration hardening standards, identify security issues associated with specific operational environments, locate embedded secrets, and perform other tests supporting organization-specific standards and compliance requirements.

(Gartner sources)

|  |  |  |
| --- | --- | --- |
| 7.8.1 |  | Setup a process of content trust at each stage of development |

### Architecture phase

During the architecture phase, modelling of the Cloud Information System helps to manage the level of compliance specially to stick with different standards of quality.

In addition, it supports the abstraction approach, eases the definition and the deployment of resources and services across distributed and heterogeneous infrastructure. Model-driven deployment is an important step for achieving platforms and for preventing high maintenance costs in case of changes.

This approach increases the level of quality.

For instance, the use of OASIS – TOSCA supports the description of the Cloud services topology. The outcome of this modelling phase helps to verify the conformity of the deployed services.

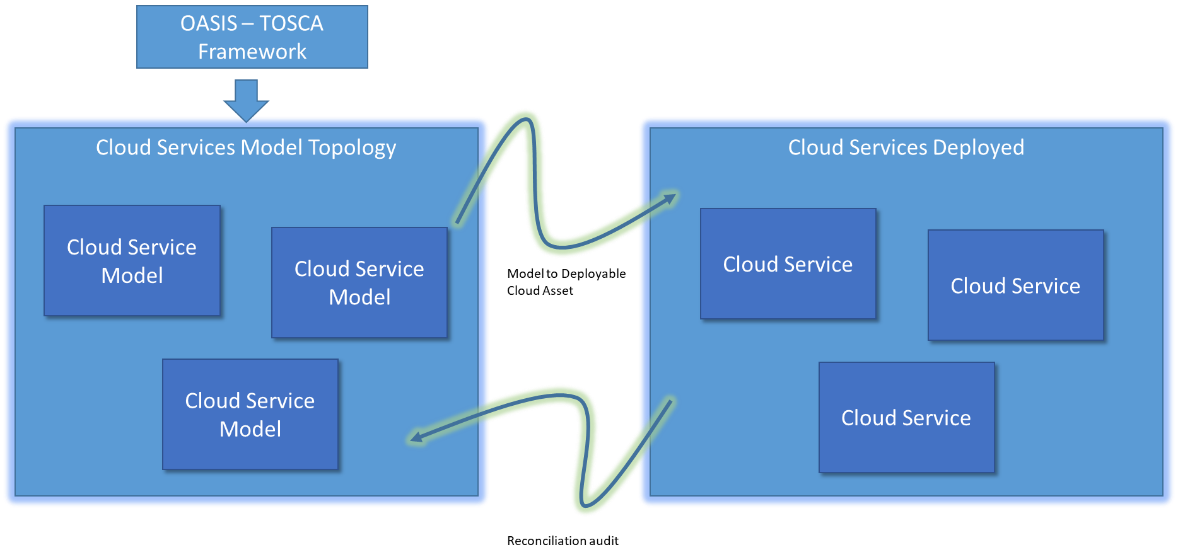


Figure 13: Model-driven deployment

|  |  |  |
| --- | --- | --- |
| 7.8.2 |  | Use MBSE approach and a modelling method adapted to the Cloud Environment (OASIS TOSCA). It participates to the portability of Cloud Services |

### Maintenance phase

One way to support the quality level of the Cloud Infrastructure (Public or Private) is to have the permanent right vision of the services deployed.

We recommend to implement Inventory function to provide the list and version of the asset, starting from the Virtual Machine flavour to the version of Database used by a business application.

This kind of information will be useful to support the security maintenance aspect.

In addition, security auditing tool able to detect vulnerabilities has to be implemented on the production and linked to a Security Operation Centre (external or internal), combining active network scanning with passive, agent and API-based scanning to increase the existing capabilities, and to have real-time visibility with improved asset coverage.

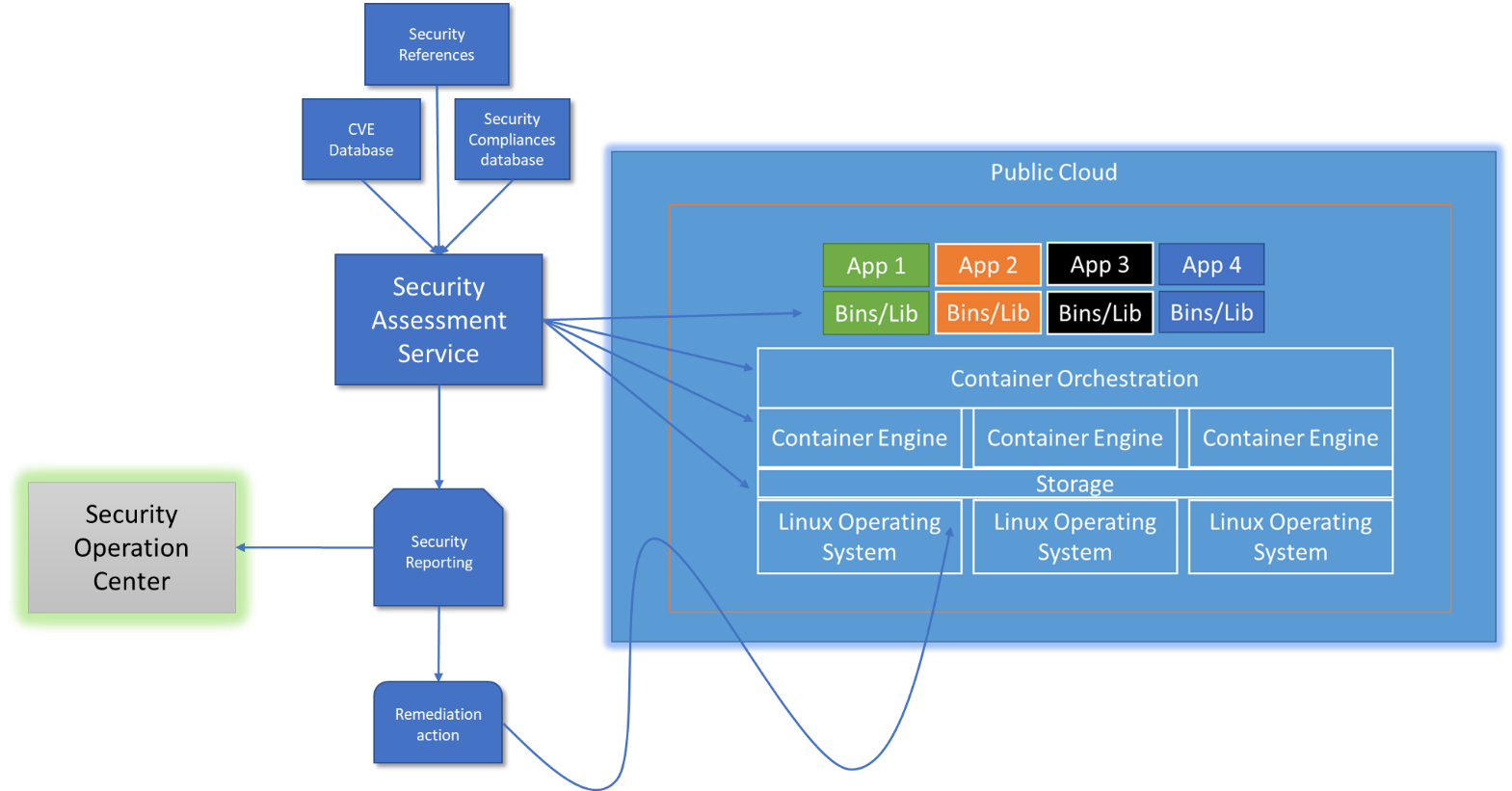


Figure 14: Security assessment service

The security assessment service can be external to the Cloud Provider or an external and trusted service (SaaS).

More globally, to keep those audits operational, the Cloud Architecture should foresee a process of reference databases updates.

An ‘Air gap’ process have to be setup to synchronize reference databases (from the editor) to the ‘Dark Site’ repository. The use of a kind of ‘Sheep Dip’ Station is fully recommended to check the data coming from Internet before move them on the final Development environment (Malware, Virus checking).

|  |  |  |
| --- | --- | --- |
| 7.8.3 |  | Maintain the right level of security of the Cloud Environment with a security situation awareness real-time status using dedicated tools |



1. https://gaia-x.eu [↑](#footnote-ref-2)
2. <https://landscape.cncf.io/> [↑](#footnote-ref-3)
3. <https://www.bis.doc.gov/index.php/licensing/commerce-control-list-classification/export-control-classification-number-eccn> [↑](#footnote-ref-4)
4. <https://apache.org/licenses/exports/> [↑](#footnote-ref-5)
5. <https://access.redhat.com/sites/default/files/attachments/producttechnologymatrix_38.pdf> [↑](#footnote-ref-6)
6. <https://www.suse.com/company/legal/eccn/> [↑](#footnote-ref-7)
7. List of RFC: [*https://en.m.wikipedia.org/wiki/List\_of\_RFCs*](https://en.m.wikipedia.org/wiki/List_of_RFCs) [↑](#footnote-ref-8)
8. Definitions from VMware glossary <https://www.vmware.com/topics/glossary.html> [↑](#footnote-ref-9)
9. See: <https://www.cncf.io/projects/> [↑](#footnote-ref-10)
10. <https://en.wikipedia.org/wiki/Observability#Observability_in_software_systems> [↑](#footnote-ref-11)
11. Vulnerabilities are tracked by organisations like “Common Vulnerabilities and Exposures” (<https://www.cve.org/>) or “National Vulnerability Database” (<https://nvd.nist.gov/>). [↑](#footnote-ref-12)
12. See <https://www.imperva.com/blog/getting-into-a-cloud-mentality/> [↑](#footnote-ref-13)
13. <https://docs.aws.amazon.com/prescriptive-guidance/latest/migration-ocm/migration-ocm.pdf> [↑](#footnote-ref-14)
14. <https://www.forbes.com/sites/forbestechcouncil/2021/05/24/realizing-the-true-value-of-the-cloud-as-an-innovation-catalyst> [↑](#footnote-ref-15)
15. See <https://blog.simpletechnology.io/devops-vs-devsecops> [↑](#footnote-ref-16)
16. See <https://blog.simpletechnology.io/overcoming-federal-devsecops-challenges> [↑](#footnote-ref-17)
17. See <https://jaxenter.com/how-green-is-your-devops-131461.html> [↑](#footnote-ref-18)
18. <https://www.atlassian.com/blog/git/simple-git-workflow-is-simple> [↑](#footnote-ref-19)
19. <https://www.atlassian.com/blog/git/simple-git-workflow-is-simple> [↑](#footnote-ref-20)
20. <https://www.atlassian.com/blog/git/simple-git-workflow-is-simple> [↑](#footnote-ref-21)
21. <https://www.bis.doc.gov/index.php/licensing/commerce-control-list-classification/export-control-classification-number-eccn> [↑](#footnote-ref-22)