Technical Specification

ISO/IEC TS 7339

Information technology — Cloud computing — Overview of platform capabilities type and platform as a service

Technologies de l'information — Informatique en nuage — Vue d'ensemble des types de ressources et des services de plateformes à la demande

First edition
2024-12

Reference number
ISO/IEC TS 7339:2024(en)

Contents

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC had not received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents and https://patents.iec.ch. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 38, *Cloud computing and distributed platforms*.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Introduction

Many cloud services allow a cloud service customer (CSC) to develop, upload, and execute their own code, rather than uploading a complete virtual machine image, or being confined to software provided by the cloud service provider.

This ability to write and execute their own code allows CSCs and others to develop or customise their own applications without having to run their own private datacentres, and without having to install, patch and manage operating systems and other elements typically required in legacy IaaS services. This approach allows the CSC to concentrate on the code that directly meets their business need rather than having to create a lot of peripheral code just to make things work.

While some cloud services are specifically designed as Platform as a Service (PaaS) wherein the execution of CSC-provided code is the primary purpose of the cloud service, others include a greater or lesser amount of “platform capabilities type” as a feature or supplement to the main function of the cloud service. These platform capabilities can be as basic as telephone call routing scripts or database stored procedures or can be as extensive as large function libraries or microservices for use in other applications. The full range of possibilities is too extensive to list exhaustively and is constantly growing as new ideas emerge.

In addition, the world of cloud computing is seeing wholly new developments and technologies added all the time. Many of these include the execution of CSC-provided code, or its equivalent for a different paradigm. For example, Artificial Intelligence (AI) services (e.g. machine learning) can be deployed as cloud services, and in this case the “CSC-provided code” can include both training data and procedural code. As another example, the world is preparing for the availability of Quantum Computing (QC) technology which (like AI) will probably be exposed to CSCs as various forms of cloud services. Both AI and QC technologies appear as services that can be incorporated within more traditional application designs, thus contributing their specialised and unique capabilities.

It is therefore useful to describe both the purpose-built PaaS concept, and the more general “platform capabilities type” as it appears in other cloud services beyond PaaS. This document explains the differences between PaaS and other services, the types of CSC-provided code that such platforms can support, the general approaches to development of code for such services, common platform architectural approaches, and how cloud computing platforms can support new technology paradigms such as AI and QC in a consistent manner.

In particular, this document provides an introduction to the “cloud native computing” concept as a pattern of platform capabilities type, providing an architectural pattern that is focused on cloud-first development and that offers greater flexibility and modularity than many older software design patterns.

It is also important to define some general recommendations to promote good practice in the provision and use of digital technology platforms of these kinds especially with respect to transparency of platform service offerings to existing and potential CSCs.

Throughout this document, unless otherwise explicitly stated, the term “platform” is always used in the engineering sense, specifically referring to the “digital technology platform” in accordance with ISO/IEC TS 5928 as implemented in cloud services, edge services, mobile services, and other distributed platforms.

The common engineering usage of "digital technology platform" includes:

— operating systems,

— “platform as a service” cloud services in accordance with ISO/IEC 22123-1,

— other cloud services that exhibit “platform capabilities type” in accordance with ISO/IEC 22123-2.

As such, this usage of platform refers to cloud services that enable a CSC to create and maintain their own hosted application, rather than using digital technology for the creation of a multi-sided market as typically used by economists and competition regulators.

However, as described in ISO/IEC TS 5928, some types of cloud services (typically various forms of SaaS) that are implemented on top of the digital technology platform can also exhibit the characteristics of a digital economic platform by creating a multi-sided market. Such SaaS implementations by CSCs of the digital technology platform are generally outside the control and responsibility of the digital technology platform service operator.

The intended audience for this document is:

— businesses considering the use of technology platform capabilities for new cloud applications (both as CSDs and as purchasers of installable cloud software)

— for those seeking to understand or describe the various cloud application development options available

— for those seeking to clearly describe digital technology platform services that they offer to CSCs

— for those developing governmental or procurement policies covering CSC-provided cloud applications

— those developing other standards that need to reference cloud technology platform capabilities and approaches.

Information technology — Cloud computing — Overview of platform capabilities type and platform as a service

1 Scope

Within the context of digital technology platforms as defined in ISO/IEC TS 5928, this document provides:

— a description of the concepts of the platform capabilities type as it appears in various cloud service categories;

— a description of the specific cloud service category of platform as a service (PaaS);

— descriptions of common technology platform architectures, development approaches, and life cycles of elements of technology platform services, including a high-level description of the popular cloud native computing concept;

— recommendations for cloud services that include platform capabilities, including but not limited to PaaS.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 22123-1, *Information technology — Cloud computing — Part 1: Vocabulary*

ISO/IEC/TS 5928, *Taxonomy for digital platforms*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 22123-1, ISO/IEC/TS 5928 and the following apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.1
platform capabilities type
cloud capabilities type in which the cloud service customer can deploy, manage and run customer-created or customer-acquired applications using one or more programming languages and one or more execution environments supported by the cloud service provider

Note 1 to entry: In this context "applications" can include programs, software components (such as containers, code, function libraries, and microservices, etc.), unfinished or incomplete programs (such as test versions or prototypes), and other forms of source code or executable software code artefacts, with or without a user interface being included.

Note 2 to entry: This does not include any minimal scripting capability internal to a single application, such as simple macros within a spreadsheet.

[SOURCE: ISO/IEC 22123-1:2023, 3.5.4, modified — Notes 1 and 2 to entry have been added.]

3.2
CSC-provided code
software code artefacts created by CSDs acting on the CSC's behalf, that are designed to execute within a cloud service to meet the needs of the CSC

Note 1 to entry: Such code artefacts can be supplemented by data or metadata.

Note 2 to entry: The CSDs creating the code can be employees or contractors of the CSC, or of another organization such as a vendor, partner, or contractor to the CSC.

3.3
metadata
data that defines and describes other data

Note 1 to entry: Metadata can include many different kinds of attributes of the data it describes. Some metadata fields are almost universal, such as date of creation and last change, type of file or object, permissions, and ownership. Other metadata can be specific to the type of object, such as the camera settings of a digital image, the author's name for a document, the version of a code artefact, etc.

Note 2 to entry: Some metadata may be directly attached to (or embedded within) the data to which it relates and will always move with it. Other metadata may be stored separately and associated with the data by reference or other means.

[SOURCE: ISO/IEC 11179-3:2022, 3.2.30, modified — The notes to entry have been added.]

3.4
cloud native
cloud native computing
approach and practices for developing, deploying, and running applications and services in cloud computing systems with specific **platform capabilities type** which offer features such as functional decomposition, containerization, orchestration, microservices for applications, automation, monitoring and logging, and continuous operation

Note 1 to entry: Additional tools and technologies that support these capabilities include DevOps, CI-CD tools, automation of orchestration etc., which individually and collectively, enable autoscaling, resilience, superior performance, ease of portability and interoperability and ease of maintenance among other benefits.

Note 2 to entry: Orchestration refers to automatic or semi-automatic deployment, instantiation, interconnection, and management of containerised software.

Note 3 to entry: This approach contrasts with more traditional virtualisation-based approaches based largely on customer-managed Virtual Machines.

Note 4 to entry: See also Clause 10.

3.5
jurisdiction
geographical or corporate area over which a cloud computing policy extends

Note 1 to entry: In a government policy context this will generally be the geographical area over which the body enacting the policy has legal authority either as government or as authorised regulator. However, in an enterprise or government agency environment, the jurisdiction of a policy might cover a business function, department, agency, or other organisational area of responsibility not tied to geography.

[SOURCE: ISO/IEC TR 22678:2019, 3.2]

3.6
cloud service developer
sub-role of the cloud service partner role, with the responsibility for designing, developing, testing and maintaining the implementation of software that is either part of a cloud service, or is designed to run on a cloud service

Note 1 to entry: The cloud service developer sub-role can be performed by any party, including CSN, CSP, and CSC.

Note 2 to entry: Normally software that is part of a cloud service will only be developed by a CSD under the control of the CSP. Software that runs on top of a cloud service can be developed by a CSD under the control of a CSC (e.g. for IaaS or PaaS).

Note 3 to entry: This definition supersedes the definition in ISO/IEC 22123-1:2023, 3.3.13.

4 Abbreviated terms

AI	artificial intelligence
API	application programming interface
CaaS	communications as a service
CaaS	container as a service
CDM	cloud deployment model
CPU	central processing unit
CSA	cloud service agreement
CSC	cloud service customer
CSD	cloud service developer
CSN	cloud service partner
CSP	cloud service provider
CSU	cloud service user
DBaaS	database as a service
DSaaS	data storage as a service
FaaS	function as a service
FPGA	field programmable gate array
GPU	graphics processing unit
IaaS	infrastructure as a service
IAM	identity and access management
IDaaS	identity as a service
IoT	internet of things
ISV	independent software vendor
LLM	large language model
ML	machine learning
MLaaS	machine learning as a service
NaaS	network as a service
PaaS	platform as a service

PII	personally identifiable information
QC	quantum computing
QCaaS	quantum computing as a service
SaaS	software as a service
SBoM	software bill of materials
SQL	structured query language
UI	user interface
W3C	World Wide Web Consortium
WCAG	web content accessibility guidelines

5 Overview of platform capabilities type

5.1 General

The definition of "platform capabilities type" means supporting the execution of CSC-provided software code written in a programming language, whether the code is written by the CSC themselves or obtained from another CSD (see 5.6), or perhaps from an open-source project. This can also include pre-compiled code provided (created or purchased) by the CSC, such as bytecode, libraries, container images, or microservices as described in ISO/IEC TS 23167.

NOTE This document considers platform capabilities in a top-down sense and shows how they fit within the bigger picture of cloud services in general. By contrast ISO/IEC/TS 23167 took a more bottom-up approach, looking at specific technologies and techniques individually rather than showing how they relate to one another or to the bigger picture. As such, ISO/IEC/TS 23167 remains a useful reference for the details of some specific technologies, though some of it has been somewhat overtaken by advances in technology and concepts since it was published. The two documents do not conflict, but they are written from different viewpoints of the topics under discussion.

5.2 Examples of capabilities

Capabilities falling within the platform capabilities type include but are not limited to:

a) Capabilities to write, upload, compile, execute, monitor, and debug their own programming code within the service;

b) Execution of CSC-provided code that operates on cloud data storage, such as SQL stored procedures for a cloud database (DBaaS);

c) The provision of APIs that allow CSC-provided code to access functions and capabilities offered by the cloud service. For example:

   1) Connection routing APIs for Network as a Service (NaaS);

   2) Call routing APIs for customer-support via Communication as a Service (CaaS);

d) The provision of APIs that allow CSC-provided code to communicate with external software functions either within the same cloud service or elsewhere;

e) The provision of APIs that allow CSC-provided code to construct and offer one or more user interfaces for communication with human users;

f) The provision of APIs that allow CSC-provided code to construct and expose one or more CSC-defined APIs for communication with other software applications or cloud services;

g) The provision of APIs that allow CSC-provided code to be managed with respect to data security, confidentiality, privacy, integrity, and accessibility for disabled persons;

h) The provision of APIs that allow CSC-provided code to manage and extract CSC data within and from the cloud service, such as in preparation for moving to another cloud service.

5.3 Capabilities not included in the platform capabilities type

Capabilities falling outside the cloud computing platform capabilities type include but are not limited to:

a) Non-cloud capabilities:

   1) Anything in which the hosting environment does not meet the definition of cloud computing in accordance with ISO/IEC 22123-1;

b) Infrastructure capabilities:

   1) “Bare metal” or virtualised computer hardware, such as the hosting of virtual machine images;

   2) Scripting for the purpose of managing infrastructure resources such as virtual machines, network configurations, basic storage, etc., since such code does not directly provide a user interface to a CSU;

c) Application capabilities:

   1) Mathematical functions within spreadsheets;

   2) Email sorting rules;

   3) Macros within SaaS productivity applications (such as spreadsheets or presentation applications);

   4) Scripting that works only within the closed context of a specific user-facing SaaS application, such as animations within a presentation tool;

d) Others:

   1) The execution of CSC-provided code for the sole purpose of education or training, such as in a learning simulation;

   2) The execution of CSC-provided code for the sole purpose of entertainment of the CSU, such as within a puzzle game;

   3) Generally, any programming capability wherein a CSC’s code executes in an environment in which it is unable to interact with other system elements or connect beyond its own environment.

5.4 The legal status of CSC-provided code as customer data

It is important for CSCs and CSPs to recognise the legal status of all code and data provided by the CSC to the CSP, as these will almost always be categorised as customer data as defined in ISO/IEC 22123-1, and as such can be subject to applicable local jurisdictions and regulations.

CSC provided code or data that contains any personally identifiable information (PII) can be subject to privacy regulations.

5.5 Intellectual property considerations

In most cases, the CSC (or another body) will hold copyright on the code artefacts, data, and metadata they create and load to the cloud service. If the CSP wishes to claim any copyright on any such submissions, this should be made very clear to potential CSCs before they commit to the cloud service. See 15.1.

If the CSC-provided code uses AI functionality through a PaaS-type cloud service to create any derivative works of copyright material, it remains the CSC’s responsibility to ensure they have the appropriate rights for their application to function in this way.

It is also possible that CSC-provided code may implement concepts covered by patents held by the CSC or others. It is the CSC's responsibility to arrange for any licencing that is needed in such a case, since the CSP has no certain visibility of what the CSC-provided code may do, or what ideas it may embody.

5.6 Cloud service agreement elements for the platform capabilities type

Every cloud service is offered based on a cloud service agreement (CSA) between the CSP and the CSC.

While many elements of a CSA will be common for all cloud services, there will be some specific elements that are included when it comes to cloud services offering the platform capabilities type.

To govern how the cloud is used, CSPs will always include specific terms-of-use that govern how CSCs can use the platform and the types of applications they are allowed to create and run on it. Typical rules can include:

a) Adherence to applicable law (especially when the CSP is subject to a jurisdiction that does not directly apply to the CSC or CSD, such as when they reside in different countries)

b) Prevention of:

   1) Cybercrime

   2) Spamming

   3) Copyright violation

   4) Execution of denial-of-service attacks

   5) Malware delivery

   6) Child safety violations

   7) Privacy violations

   8) Hate speech and extremism

c) Conditions concerning

   1) Lawful intrusion (notably for "communications" applications)

   2) User interface and Accessibility requirements (such as a requirement to follow UI design guidelines, most common for mobile device and gaming device platforms)

   3) Exclusivity requirements

   4) Some platforms only allow CSC applications to be used within a specific service area, such as a single country or region.

   5) Some platforms require the use of specific payment mechanisms for purchases made through the CSC application (most common for mobile and gaming device platforms)

Some platform operators require some kind of exclusivity, that the application cannot also appear on a competing platform, or there can be a difference in pricing or other terms and conditions if this occurs. The CSA will also include elements describing the rights of the CSP to suspend or terminate the operation of CSC-provided applications that display incorrect or inappropriate behaviour, and the types of alerts and debugging that will be available to warn of such situations arising.

6 The cloud service developer sub-role (CSD)

6.1 The definition of cloud service developer

A cloud service developer is a sub role of the Cloud Service Partner role (CSN role) and is indicated as CSN:cloud service developer in accordance with ISO/IEC 22123-3.

However, this refers to the CSN role, not to the CSN as a separate party. There are three parties in cloud computing, that are Cloud Service Customer (CSC), Cloud Service Provider (CSP) and Cloud Service Partner (CSN). Each of these parties is by definition a natural person, a legal person or a group of either, whether or not incorporated, that can assume one or more roles.

CSC as a party can not only perform the CSC role but also perform the CSN role and can thus perform the activities associated with CSN:cloud service developer.

The important point here is to explain that the CSC as a party can explicitly include the sub-role called CSN:cloud service developer, thereby clarifying the responsibility boundary. Not every CSC is required to do this, but many will do so.

Thus an individual (or team) working as a CSD can be an employee of a CSC, of a CSP, or of a separate organisation acting as a CSN on the CSC's behalf. In every case, the CSD performs the activities defined in the CSN:cloud service developer role.

6.2 CSD activities

The activities of the CSD in creating code can thus be performed by various teams or individuals (or even by some applications) including but not limited to:

a) Employees of a CSC who write code for their own application to execute on a digital technology platform

b) Employees of a company which provides CSD support (e.g. consultants) to assist a CSC in developing their application

c) Employees of a company which acts as an ISV, developing code which will be sold to multiple CSCs for inclusion in their own applications

d) Contributors to open-source projects which can be (re-)used by CSCs, CSNs, CSPs, and ISVs in the development of their own applications

e) Employees of a CSP providing mechanisms that can be incorporated by any of the above when developing applications to run on the CSP's own platform, such as APIs or other means to access and use platform-specific features

NOTE CSDs are not only involved in writing and testing programming code. They will often be contributing data that the code needs to function and also be heavily involved in the entire life-cycle of the application. See 7.4.

6.3 CSDs and the CSC application code lifecycle

In each case, the code artefacts will go through phases of design, development, testing, deployment, operation, maintenance, and eventual retirement that will not be further discussed here. However, it is important to realise that that the participation of the CSD rarely ends with deployment. The CSD can also work with other stakeholders and other processes with various levels of commitment. This can include responsibilities in the areas of change management, risk management, and security amongst others.

7 Platform as a Service (PaaS) concepts

7.1 PaaS as a cloud service category

“Platform as a Service” is, like the other “XaaS” abbreviated terms, a broad category of possible cloud services that have some common attributes but can be implemented in very different ways and with little or no common core of functionality.

Some PaaS implementations are broadly useful to a wide range of CSCs, while others will be specialised to particular vertical industries or other specific needs.

Some PaaS implementations are entirely proprietary in nature, while others focus on open-source technologies and projects. Many will blend both of these development approaches.

Thus, Platform as a Service can only ever be a category of cloud services, not a rigid specification and certainly not limited to a single implementation. Different implementations of PaaS might choose to use different technologies, support different languages, choose to offer different special functionality, and offer very different network, storage, and computing functions.

EXAMPLE A specific PaaS implementation can choose to offer the customer:

a) A specific set of programming languages

b) The option to use GPUs for processing in addition to CPUs

c) Support AI functionality directly in the platform

d) Provide advanced scalability and load balancing across multiple datacentres and server instances

7.2 Benefits of choosing PaaS

7.2.1 General

Implementations of Platform as a Service can offer several advantages to CSDs compared to more basic categories of cloud services such as Infrastructure as a Service.

7.2.2 Reduced maintenance

In a PaaS environment, many of the software elements that are managed by the customer in an IaaS environment are handled by the CSP. Thus, there is no need for the customer to construct, manage, patch, or maintain virtual machine images, guest operating systems, or other incidental code elements. Instead, they can focus exclusively on writing and maintaining the code that is specific to their business solution.

7.2.3 Scalability

In a PaaS environment, creating multiple instances of the same application code can usually be done by means of a simple request to the cloud service. There is no need for the customer to manually create and manage additional virtual machines; that is done “under the covers” by the PaaS itself as required.

7.2.4 Access to advanced features

CSDs wanting to access advanced features at the API level are likely to choose some kind of PaaS that has these features included. Examples of this can include AI functions, blockchain, big data processing, and others. While these can also be implemented on an IaaS platform, that will often the require the customer to create specific VM images and install those functions as additional software components.

7.3 The “fuzzy” boundaries between IaaS, PaaS, and SaaS

7.3.1 General

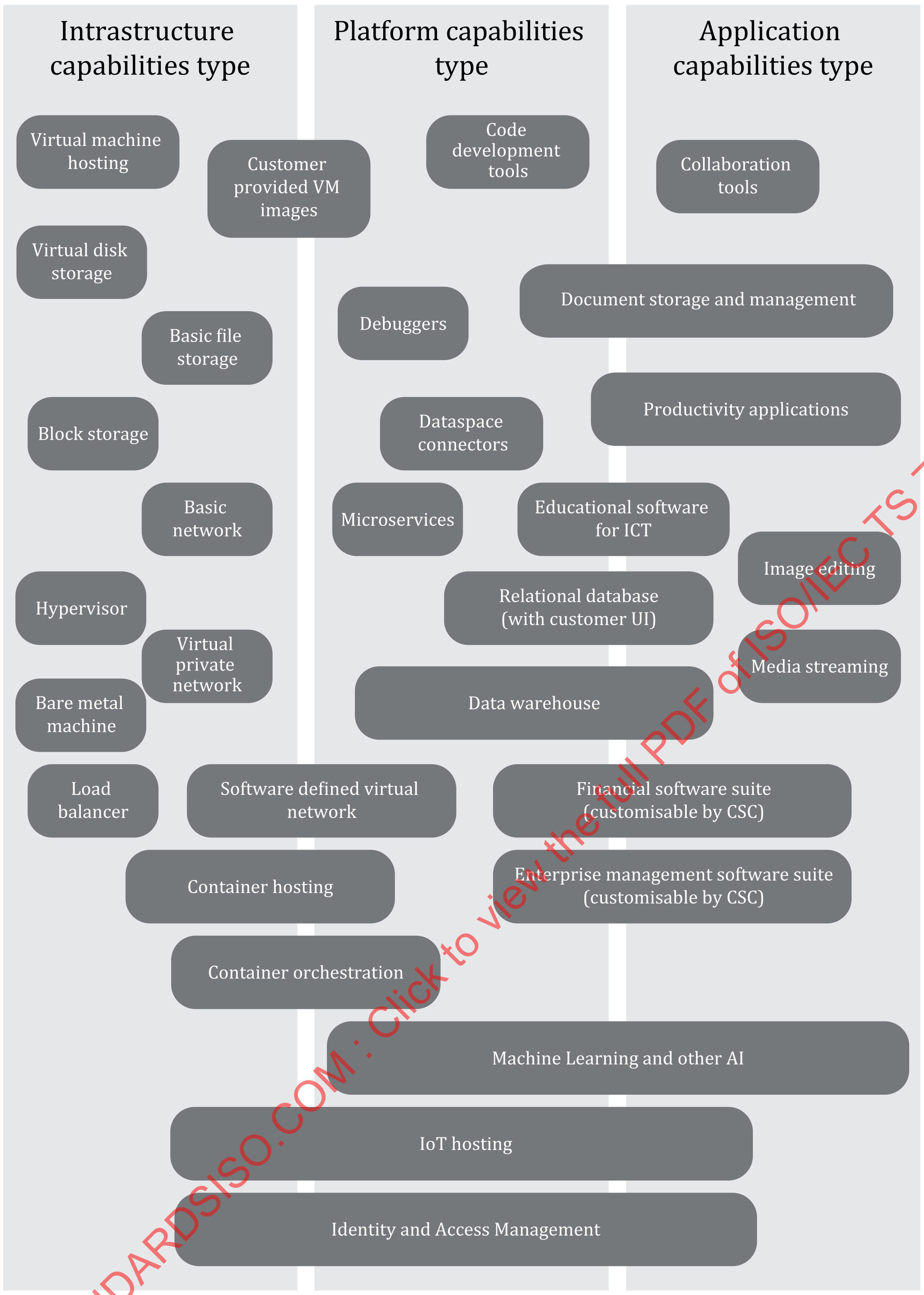
For the purposes of this document, “fuzzy” indicates something that is variable, vague, and indistinct. In this specific context, a fuzzy boundary is highly subjective and possibly changes over time as technology evolves, so a rigid definition of where the boundary falls is not possible.

The original NIST division of cloud services into IaaS, PaaS, and SaaS dates back to 2010/2011.[11] While this provided a useful vision in the very early days of cloud computing, it is no longer as clear-cut a distinction between cloud service types as was originally envisioned.

In 2014, ISO/IEC updated this concept and identified that these three basic concepts were better expressed as “capabilities types”, combinations of which could be exhibited by many different cloud service categories. See ISO/IEC 22123-1 and ISO/IEC 22123-2.

It was further observed that even this reorganisation did not fully encompass many types of cloud services, or the elements that went into them, hence the description of these divisions as being “fuzzy” rather that truly objective.

Figure 1 shows some (non-exhaustive) examples of the fuzzy boundaries that have emerged between the infrastructure, platform, and application capabilities types. Some capabilities clearly fall under a single capabilities type, while others exhibit characteristic of two or even all three capabilities types.


Intrastructure capabilities type
Platform capabilities type
Application capabilities type
Virtual machine hosting
Customer provided VM images
Code development tools
Collaboration tools
Virtual disk storage
Document storage and management
Basic file storage
Debuggers
Productivity applications
Block storage
Dataspace connectors
Basic network
Microservices
Educational software for ICT
Image editing
Hypervisor
Relational database (with customer UI)
Virtual private network
Media streaming
Bare metal machine
Data warehouse
Load balancer
Software defined virtual network
Financial software suite (customisable by CSC)
Container hosting
Enterprise management software suite (customisable by CSC)
Container orchestration
Machine Learning and other AI
IoT hosting
Identity and Access Management

NOTE The location of individual example items within this diagram has no significance other than whether they fall under infrastructure capabilities, platform capabilities, or application capabilities, or are spread across multiple capabilities types. This is not a “stack” diagram.

Figure 1 — Examples of fuzzy boundaries between IaaS, PaaS, and SaaS

One problem for customers and regulators is that companies running cloud services will choose whatever marketing label they think is in their best business interests, which does not necessarily align with what the standards (and potentially regulations) are designed to describe.

7.3.2 IaaS vs PaaS

The old description of an IaaS cloud service was heavily oriented towards a virtual-machine hosting environment wherein the CSP provided a hosting platform running a hypervisor environment. This would come with some basic storage mechanisms, such as virtual disks, and perhaps some kind of binary object storage, together with sufficient network capability to connect the virtual machines to one another and to things on the Internet. Pretty much every other function had to be implemented by the CSC within their virtual machine images, though the IaaS provider would often provide various templates to make this somewhat easier, such as pre-loaded with an operating system and perhaps some libraries. Still, it remained the CSC's responsibility to secure and maintain each running virtual machine, including any patches needed for the guest operating system.

In such an IaaS environment, CSCs can certainly write their own code, but it will execute within one or more of their virtual machines just as if they had rented a hosted hardware computer. If they need any complex functionality, such as load balancing, Identity management or a database, they may need to install it into a virtual machine and manage it themselves.

This approach worked well in the early days of cloud computing, when the vast majority of cloud workloads comprised virtual machines that were being moved from an on-premises virtual machine host to one in the cloud.

The original idea of a PaaS cloud service was designed from the ground up as an execution environment for the CSCs' application code. The CSP would provide an operating environment wherein a CSC can upload a set of code assets that dealt only with the business logic of their intended application, while the CSP dealt with all the other functions needed to support it, including any guest operating system or virtual machine layers. This freed the CSC from having to maintain and secure operating systems and associated code (such as load balancing) as the PaaS environment would provide and maintain all of this for them.

The problem we have today is that some of the functions originally regarded as PaaS functions are now often provided in cloud services titled as "IaaS". We also have "IaaS" offerings that actually host Containers rather than (or in addition to) complete VMs, which is quite a divergence from the original understanding of IaaS.

7.3.3 PaaS vs SaaS

The boundary between PaaS and SaaS is also blurred.

As noted above, the original intent was that PaaS cloud services would execute CSC-provided code and it would be the CSCs who developed all the applications that ran on it. History has shown that this model didn't really work. Even if the PaaS environment provides a lot of key functionality as described in this document, it has become normal practice to combine this code with licenced applications and use it primarily as "glue" between software elements running in the same cloud service or elsewhere.

SaaS was originally assumed to be the equivalent of a package bought to run on computers at the CSC premises, and to have a user interface with which humans could interact with it. By contrast, the old PaaS model assumed the CSC had to write their own user interface code for their application.

So, the old separation of "CSCs write code on PaaS but use a UI on SaaS" is now badly broken, making it very hard to draw a clear distinction between them. If a SaaS application with a complex UI also allows the CSC to upload and execute significant code artefacts within the application, it is impossible to force such an application into one category or the other.

7.3.4 DSaaS as an example

Data storage as a service (DSaaS) can crop up in any of the three basic models, and it isn't always clear whether any type is exclusive to a classical cloud service category.

The most basic form of DSaaS, commonly offered within or alongside IaaS, offers virtual disks and raw storage of binary objects ("blobs"). The CSP provides the storage mechanism, but usually has no idea of how the stored data is formatted or what it contains. This type of DSaaS exhibits only the infrastructure capabilities type.

A more advanced form of DSaaS is a cloud database which allows for the storage of data in a structured form, and wherein code and procedures such as SQL Stored Procedures can be uploaded and executed by the CSC. This type of DSaaS exhibits the platform capabilities type.

In some cases, such a database or structured document storage, the service can also offer a user interface with which a human CSU can directly control the database or storage, set up or delete data tables and other structures, enter or modify stored data, generate reports, or do various other things. When this is available, the DSaaS exhibits the application capabilities type.

7.3.5 Relevance of the boundaries

The decade-old IaaS, PaaS, and SaaS concept remains a useful shorthand for describing quickly what a cloud service offers, or when explaining to a non-technical audience, but are no longer useful for setting concrete boundaries between different categories of cloud services.

Any attempt to set different regulations or rules based on these categories is likely to fail since it can be highly subjective.

For example, while it can be tempting to think that IaaS is "simple so it needs less rules", that becomes an invitation for a CSP to describe a cloud service as IaaS when in truth it also exhibits platform or application capabilities types, such as if they offer some kind of SQL database cloud service within the "IaaS" cloud service.

It is even worse when attempts are made to apply different rules or codes of conduct to IaaS, PaaS, and SaaS cloud services. A single cloud service today can easily exhibit characteristics of two or even three of these categories, leaving the CSP or CSC unclear as to which set of rules or Code their cloud service is supposed to declare adherence, or potentially even facing contradictory requirements that might never be solved short of a court decision, if then.

Some cloud services such as AI machine learning are almost impossible to categorise as IaaS, PaaS, or SaaS. For example, does training data for an AI system constitute "CSC-provided code" or not? Does the user interface of an AI data-science tool constitute an "application" cloud service?

7.3.6 Selective use of the boundaries

It is probably impossible to remove all mention of IaaS, PaaS, and SaaS from standards and product documentation. The terms were very useful in the early days and have become embedded into the consciousness of the industry.

7.3.7 Implications

Therefore, it is wise to avoid trying to continue using the terms in environments wherein objectivity and precision of language is important such as in law, regulations, or industry codes of conduct. Such things should either be made to encompass all cloud service categories equally (the so called "XaaS approach") or confined to clearly defined feature-sets rather than broad-but-fuzzy categories like IaaS, PaaS and SaaS.

7.4 Common functions included within a PaaS

7.4.1 General

The following functions are typically (though not universally) implemented by the CSP in a PaaS cloud service and made available to CSCs for use with CSC-provided code and applications.

7.4.2 Documentation of CSP-provided interfaces

A CSP offering PaaS will need to provide sufficient documentation that allows all CSDs to understand the environment being offered, and the functions available to them.

This documentation will need to include those APIs that the CSD can invoke from their code, but also indicate preferred approaches and best practice, and in particular give adequate warning if any current API or function is likely to be deprecated or modified in the future.

Full documentation of available interfaces is also valuable for easing application portability.

7.4.3 CSC-provided code lifecycle management

As discussed in 7.5, the CSC-provided code used to build an application to run in a PaaS will go through various stages. The PaaS cloud service typically provides the CSC with the means to organise this through a user interface.

7.4.4 Identity and access management (IAM)

One of the most complex and vulnerable problems for any software development is the need to identify, authenticate, and authorise individual users, groups, and applications so that they can access and use the software functions they are entitled to, but nothing beyond that entitlement.

This is an essential element in cybersecurity, wherein a breach can open an entire organisation up to cybercriminals, so CSCs take it very seriously.

Of course, it’s possible for a CSC to write their own IAM functions from scratch and make use of them from their own applications. However, this is a complex and challenging task and introduces considerable security risks if done incorrectly, so many will choose to make use of an existing IAM on-premises application or private cloud service provided by their existing IT infrastructure, offered by one of their CSPs, or from a trusted third-party organisation. The latter two cases will often be categorised as IDaaS (Identity as a Service) cloud services.

There is also the advantage that most commercial software packages will work with existing IAM systems, but are not necessarily flexible enough to cope with situations where the CSC desires a uniquely customised identity data structure or credentials to reflect their business structure or other needs.

Modern IAMs support multi-factor authentication, biometrics, secure login methods and protocols (beyond passwords), and appropriate credential management and revocation mechanisms.

7.4.5 Message queuing

Large-scale applications are of necessity multi-threaded, such that the application code thread never has to sit waiting for a response. The most common approach to this is to use message queuing. When one element of the application software needs something done, requires some information, or needs to report something to another element, it constructs a message and “posts” this to a message queue that will be read by the destination element. When the result is determined, a reply message is placed on a message queue for retrieval by the original element. In this way, everything can occur asynchronously, and each element can continually process whatever messages it receives without waiting for a synchronous response.

This becomes especially important when an application has multiple instances of elements to spread the load across multiple servers or other computing platforms. If a specific message queue tends to “fill up” because the messages posted there are slower to process, it’s possible to assign additional instances of those elements that read and execute from that queue, thus reducing or removing the congestion.

7.4.6 Load balancing

For large scale applications, there is a need to balance the load across several, or perhaps even thousands, of servers. Message queues were mentioned in 7.4.5, but this is also true for things like HTTP requests to web pages, incoming emails, and many other kinds of interfaces.

With a good design, it doesn’t really matter which instance of a server receives a specific request, since there will be multiple servers each equally able to proceed with it.

So, most PaaS cloud services include load balancing functions that allow a CSC to both design and manage the loading of incoming requests across the right servers to give the fastest possible response.

In some cases, the PaaS allows the CSC's own code to interrogate the state of loading and react to adverse conditions, such as warning a human user, or using a lower resolution of images, etc.

Advanced load balancing can also deal with things like geographical differences in latency, such as knowing that a distant but lightly loaded server can be more suitable for handling a request than a closer but already heavily loaded one. It is not unknown to have a situation in which a response from the other side of a major ocean can be a better choice than one from a local datacentre.

7.5 Life-cycle of a CSC application on PaaS

7.5.1 General

All software proceeds through a life-cycle, starting with an idea which is developed and deployed, through to when it is no longer useful and is decommissioned.

This is also true for a CSC application running on a PaaS, and this needs to be planned. Some CSPs can provide tools to aid in the management of a CSC application through the stages of the lifecycle.

CSDs can be involved or responsible throughout the whole lifecycle of the application, from conceptualisation through to end-of-life.

7.5.2 Conceptualisation

The first task for the CSC is to decide the purpose and desired functions of the application they wish to develop or adopt.

7.5.3 Requirements analysis

Once the concepts are understood by the CSC, these need to be translated into specific requirements. In the PaaS context, this especially means identifying the functions and capabilities that are needed from the PaaS cloud service itself, as offered by the CSP running the PaaS.

Once these requirements are clearly defined, the available PaaS offerings can be compared based on these requirements, plus any other business requirements that apply, such as pricing, existing contracts, etc.

During this phase, it is important for the CSC to decide which if any unique features of a PaaS they will employ in their application. For an innovative application, it can be advantageous to choose a PaaS that has special capabilities that will make development of the innovation easier or more cost effective, but it is important to recognise that this can limit the ability of the CSC to switch to another PaaS later, since the alternative PaaS won't necessarily have these capabilities. This is a decision that only the CSC can make based on their specific business needs.

7.5.4 Code development

Development involves the writing and testing of software code by the CSD. This can often be done directly within the PaaS cloud service itself, or offline and then uploaded into the PaaS as one or more files.

ISO/IEC 22123-3 identifies the activities of the CSD which are performed to create usable code. This includes writing the code and authoring any required data or metadata that is needed for it, but also module testing of what is written at each stage. In modern "DevOps" processes, testing is typically performed by the same CSDs concurrent with code writing throughout the development process.

7.5.5 Application integration

In many cloud applications, the CSD will be relying on external functions provided either by the underlying PaaS cloud service itself, or by additional packages, libraries, other cloud services, or microservices already developed by the same CSC, or obtained from a third-party acting as a CSN. Integration involves configuring and testing each software element to work with the others that it depends upon. In some jurisdictions there can be a legal obligation to clearly document dependencies on which an application relies.

An important consideration in this stage is whether and how to plan for some future portability of the CSC application to a different PaaS environment, such as one offered by another CSP. See Clause 12.

7.5.6 Staging and Deployment of an application on PaaS

It is common practice to first "stage" all the elements of an application in a private testing environment that is identical to the final deployment, but which is not visible to external end-users. This staging environment can use fake data or real data as appropriate. For example, an application intended to process large amounts of incoming live data (such as from millions of IoT sensor devices) can be better tested with live data since fake data potentially does not have realistic distributions, burstiness, or other characteristics that the application will need to handle in real operation. However, if the application will be handling personally identifiable information (PII), it can be better to use fake data to avoid potential privacy concerns.

Deployment is the process of moving the application from the "staging" environment to the "operational" environment wherein it will perform its intended task with real data and real users.

It is common practice to "stage" any planned updates to the application and components so that the new configuration can be tested in the staging environment to reduce the risk of introducing errors in the operational system. It is also common to keep a copy of the previous operational system available so that the application can be "rolled back" to a "known good state" if any error is introduced by such an update.

If the application will interact with other systems, such as cloud services offered by another CSP, the staging process may need to account for this either by simulating that process, or by making appropriate arrangements with the concerned CSP(s).

Clearly there is a shared responsibility between the CSC and the CSP(s) parties to identify, isolate, and rectify any problems observed during staging, since each party is likely to have a different view of the operation of the system, and some problems may be obscured by lack of visibility of relevant information (such as network congestion). This is particularly true when CSC-provided code is executing in a PaaS, since minor coding errors can cause unpredictable symptoms. Many PaaS implementations will include tools to assist in this type of debugging. Good handling of the staging process is in the interests of all parties, so as to get the application fully operational (and thus profitable) as soon as possible with a good experience for the CSC.

7.5.7 Operation of an application on PaaS

An application built on PaaS will generally be either a cloud service that needs to be kept continuously running and available for end users every day and around the clock, or it will be an "active period" cloud service that has clear operations times with intervening "down" times.

In either situation, the PaaS will often provide APIs to the CSD that allows their code to understand the current state of the environment and receive a signal if it changes, so that the code can correctly suspend itself or shut itself down, and recover gracefully when needed. This could include flushing any buffers, storing or abandoning any uncompleted transactions, ensuring all states are correctly recorded to non-volatile storage, etc.

The cloud service agreement (CSA) will normally define the quality of service parameters such as anticipated downtime recovery, technical support commitments, etc. See ISO/IEC 19086.

7.5.8 Scaling of an application on PaaS

Designing an application for scalability has become a well-understood practice in cloud computing.

Typically, this involves dividing functionality up into smaller elements and implementing these as cloud services or microservices, linked together by some form of asynchronous communication such as high-speed message queues.

EXAMPLE Figure 2 shows a simple three-tier application which uses a group of web servers for the user interface, a group of business logic servers for much of the calculations, and a distributed database for storing all data. Using this approach, the application can be scaled up by adding new servers and configuring them to work with the message queues. For very large scaling, such as for a large social media network, far more sophisticated approaches are required.

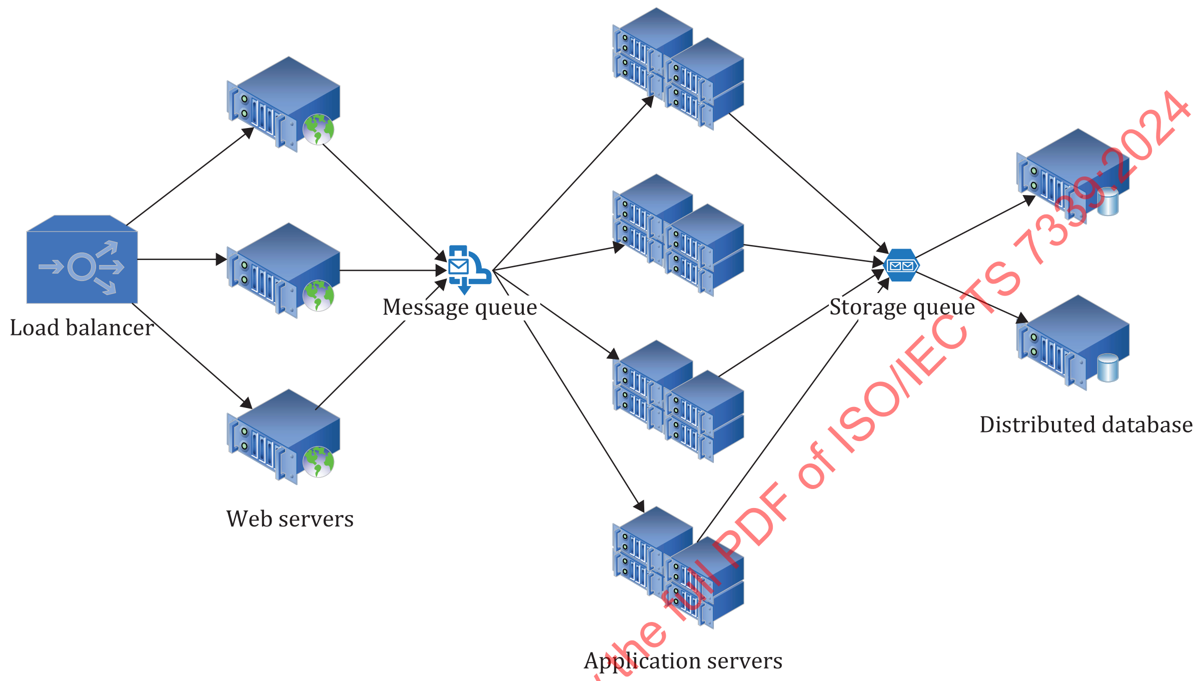


Figure 2 — Example scalable application

7.5.9 Maintenance of an application running on PaaS

All software requires periodic updates, whether to add features, to fix bugs, or to address newly identified security or privacy issues. Historically, poorly executed updates to a large cloud system have sometimes created major problems, so this process needs to be carefully planned and executed.

The "planned downtimes" approach is somewhat easier for operations, since any necessary patching or modifications can be staged and tested during the down times. For an urgent security fix, it can still be necessary to trigger an additional downtime, so CSDs are to take this possibility into account when writing their code so that this can happen gracefully without causing errors such as lost transactions.

For an always-available application, it may be necessary to adopt a worker/standby or other multi-part architecture wherein updates and changes can be made to part of the system while the rest remains running.

7.5.10 Disposal of an application running on PaaS

When it comes time to end the use of an application running on PaaS, it is important to ensure that all associated data is correctly handled, either by conversion for use by some successor application, or by the appropriate data disposal processes. The CSP is unlikely to know whether specific data or other resources need to be retained or not, so it is the responsibility of the CSC to plan and request the correct handling and disposal process.

Also, it is very important that any file, data, or interface access permissions that have been granted to the end-of-life application are correctly removed from all systems with which it was connected so that these do not remain unintentionally open as a potential security vulnerability.

7.5.11 Evolution of a CSC application over time

7.5.11.1 General

Much as with scalability, adopting a similar approach to that shown in Figure 2 can also allow for individual components or microservices to be upgraded with new functionality individually while the rest of the system remains in operation. In such a case, the message queues can be configured to omit the microservice that is due to be updated, then add it back in once the upgrade is fully in place.

7.5.11.2 Version control and forward compatibility

Version control is important for such architectures. Ideally every message should indicate the version of the instruction, or the source module, or the desired destination module. These fields can then be used in routing messages to destinations that know how to handle them.

Version control as part of a configuration management system has an important task in identifying the software iterations which can be delivered to a customer. This can also feed into the Software Bill of Materials (SBoM) which may be used for governance purposes.

It is also possible to design in forward compatibility, by including information in each message that tells a recipient what to do in the event that the message is of the wrong version or otherwise not understood.

EXAMPLE A customer application is running on PaaS. It uses message queues to communicate between numerous software modules, some of them arranged in processing chains. Each message includes a field of version information including some forward compatibility fields. A module receives a message that is of a later version than it knows how to process. It reads the forward compatibility field and sees that this message can be safely ignored and deleted. Another incompatible message arrives, and this one indicates that the message is to be passed to a specific message queue for processing elsewhere. A third incompatible messages arrives, and the forward compatibility field indicates that the module is to raise a major error condition and shut itself down.

7.6 Evolution of a PaaS cloud service implementation over time

In addition to the CSD updating their own application code, there will be times when the software implementing the PaaS cloud service needs to be updated. The CSP will usually do this by moving affected CSC workloads to other server instances while their original server is updated. In such a situation, it is normal that the PaaS notifies the affected instance CSC-provided code such that it can ride through the relocation gracefully without errors. See 7.5.7.

7.7 The use of PaaS to build SaaS offerings for others

When an organisation decides to build an interactive application for the use of their own customers, they will often choose to do this by building on a PaaS. The application thus built will appear as a SaaS application to the customers of the organisation. In this situation, the organisation using the PaaS is itself acting as a SaaS CSP to their own customers.

EXAMPLE A bank uses a PaaS to implement an eBanking application for its customers. The bank is therefore a CSC to the PaaS CSP, but a SaaS CSP to its own customers.

8 Architectures for PaaS

8.1 PaaS as a category of pure platform cloud service

8.1.1 General

PaaS is one of the main cloud service categories described in the general Cloud Computing Reference Architecture documented in ISO/IEC 22123-3.

However, PaaS remains a category of cloud service, not a specific type of cloud service. There are many different variations of the PaaS concept, and many different ways to implement each variation.

8.1.2 General layering

As with all cloud services, the PaaS architecture follows the general layering approach used for cloud computing, as shown in Figure 3.

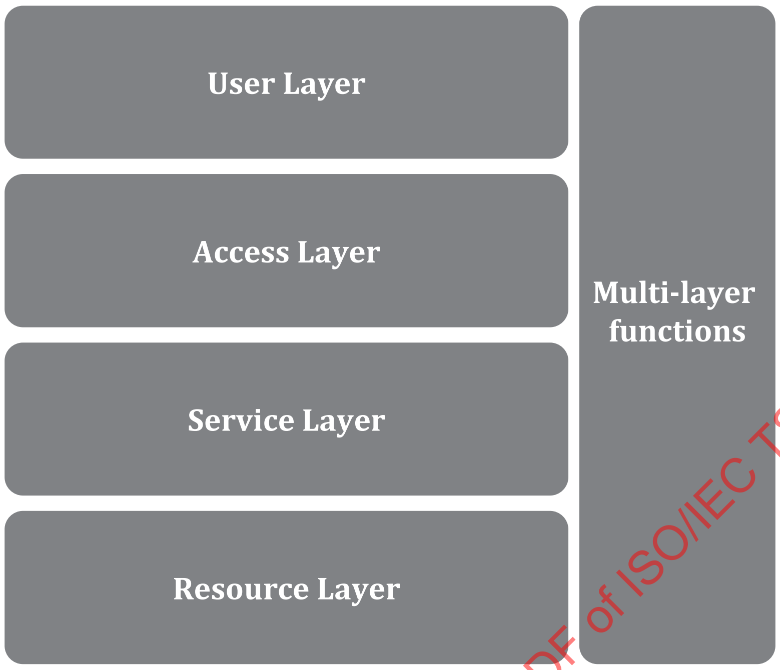
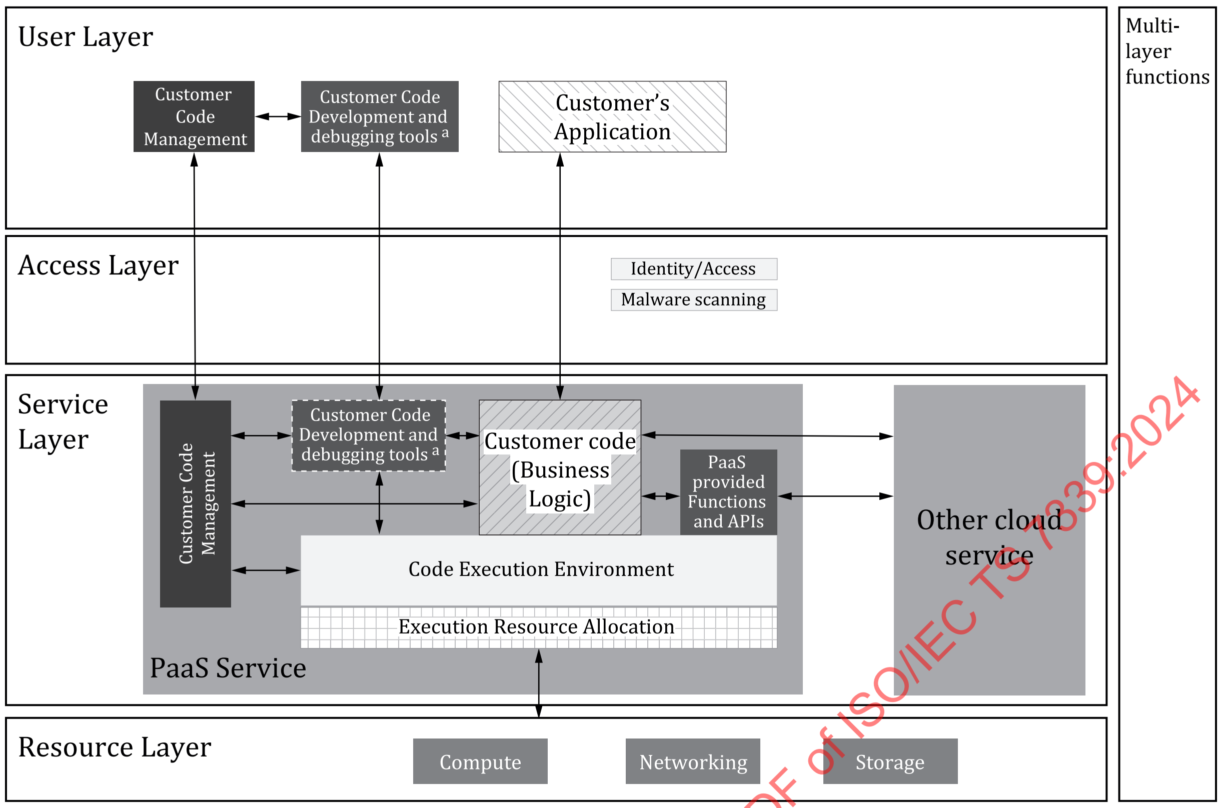


Figure 3 — General layering of cloud computing

For a full description of these layers and what they include, refer to ISO/IEC 22123-3.

8.1.3 Generic architecture for PaaS cloud services

Figure 4 shows a generic architecture that is applicable to most PaaS-like cloud services. Remember that PaaS remains a broad category of cloud services and is not a prescriptive specific implementation. There can be specific PaaS cloud services which do not follow this architecture, though most probably do.



[a] Optional.

NOTE This diagram is based on and is to be interpreted in the context of ISO/IEC 22123-3 where there is a single cloud service provider. For multi-cloud situations, please see Annex A.

Figure 4 — General PaaS architecture

This general architecture shows how CSC-provided code can be delivered to the PaaS cloud service, managed while there, and can be executed in the provided environment. The CSC-provided code does not have direct access to the Resources of the CSP, but it can request allocation of such resources within the service layer.

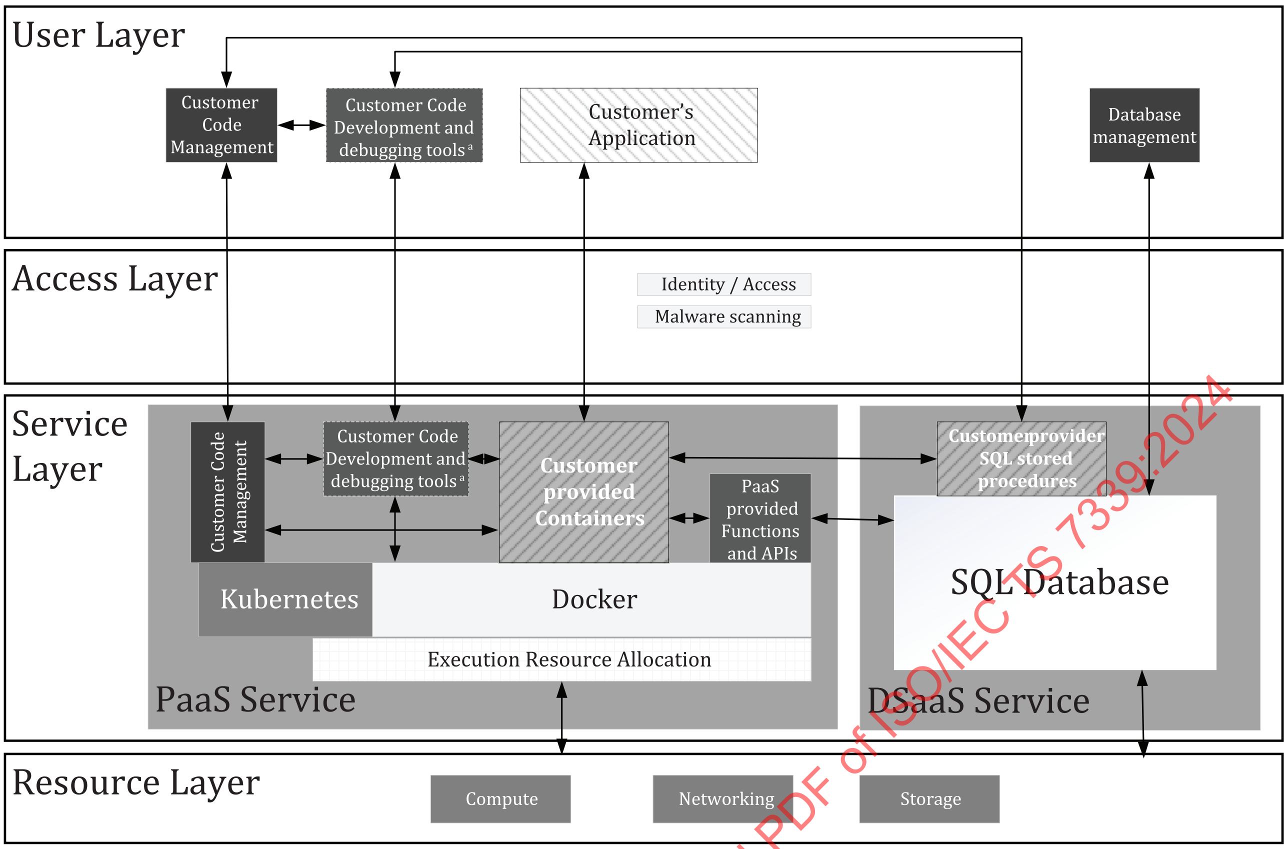
CSC-provided code can be developed and debugged either within the CSC's own environment (user layer), or within the cloud service, or a mix of the two. This CSC-provided code can make use of additional functions provided by the PaaS CSP within the PaaS cloud service, other functions from another cloud service of the same CSP, or indeed any functions made available to the CSC by a cloud service at another CSP.

Although it isn't shown directly here, it can also be possible for CSC-provided code to access cloud and non-cloud functions that are made available from anywhere, even functions that reside in their own on-premises systems. This can be very useful during a gradual migration of on-premises IT functions into cloud services, migrating from a private cloud to a public cloud, or when some functions are required to remain on-premises for security, compliance, or other policy or business reasons.

8.1.4 Example illustration of the generic PaaS architecture using Kubernetes

One popular approach to implementing a PaaS is to use the open-source Kubernetes[1)] framework to run Containers that hold the CSC-provided code. Also common is to use a cloud-based SQL database for structured data. We can illustrate how such a cloud service could be mapped to the generic model as shown in Figure 5.

1) Kubernetes is the trademark of a product supplied by the Linux foundation. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO or IEC of the product named. Equivalent products may be used if they can be shown to lead to the same results.



[a] Optional.

NOTE 1 This diagram is based on and is to be interpreted in the context of ISO/IEC 22123-3.

NOTE 2 For reasons of space and simplicity, the vertical block for multi-layer functions is not shown in this and subsequent figures, though they are still present in such a system.

Figure 5 — Example using Kubernetes and SQL

This example illustrates both the execution of CSC-provided code within the Kubernetes/Docker container execution environment, and those of a DSaaS which offers platform capabilities type, which can store and execute CSC-provided SQL stored procedures within the database cloud service. These are two distinct cloud services (in this example offered by the same CSP), but in practice they are sometimes offered as a combined cloud service product (thus illustrating the "fuzzy boundary" described in 7.3).

See also Annex B.

8.2 Microservice architectures

See also ISO/IEC/TS 23167.

Microservices-based architecture have become extremely popular – as opposed to traditional monolithic architectures, as they enable application codes to be written in smaller, modular pieces, which are then loosely coupled (through asynchronous messaging and queues) to provide agility, ease of debugging, autoscaling, increased resilience and support to developers in the entire development cycle.

This approach allows for highly modular code development, with an increased chance of reusing a given microservice in multiple applications. Containers are often implemented in microservices-based PaaS architectures which can then be ported, orchestrated and scaled without further development being required. Load balancing across multiple instances of a microservice in a single location can be used to increase throughput, but instances of containers can also easily be physically distributed into multiple locations, even in Edge nodes, to aid in reducing latency for common functions. This is especially valuable in large IoT deployments such as for millions of electricity meters or thousands of cellular signal masts for which local processing can greatly reduce backhaul bandwidth requirements from the edge to the core datacentre(s).

Containers, in turn, are orchestrated to put together and run applications, within a cloud or across multiple clouds as described in Annex A and Annex B.

8.3 Native PaaS

This is when a PaaS cloud service is provided separate from any other cloud service and is primarily devoted to the execution of CSC provided code. In such a cloud service, the CSC can have little or no visibility of any virtualisation environment beyond being able to manage the number of instances of their package to be deployed and executed. In a Native PaaS cloud service, the CSC has no access or control over any physical server, virtual machine, or operating system on which their code is hosted, since this is entirely managed by the CSP.

8.4 PaaS over an IaaS

This is when the cloud service offers a layer of PaaS functionality on top of an IaaS cloud service architecture.

This can be implemented by the CSP providing code or libraries that can be included in a VM or container running in an IaaS or CaaS cloud service, but which provide access to additional programmable functionality as for a native PaaS.

8.5 PaaS within a SaaS

Some cloud software applications can include significant programming capability allowing their CSCs to develop extensive customisation of the application, beyond mere simple scripting.

For example, a powerful financial package or database can allow for extensive business logic to be coded directly within the application itself.

Whether this really counts as a combination of PaaS within a SaaS or is better thought of as a completely new category that combines both application capabilities type with platform capabilities type would have to be decided for each such product, however this type of application is becoming more common especially for larger enterprise business applications. This is yet another example of the fuzzy boundary between PaaS and SaaS described above.

8.6 PaaS linked with a SaaS

In this situation, a CSC-provided application works closely with a separate SaaS application. This is slightly different to the case above, in that the PaaS environment allows for communication with the SaaS application but does not reside within it. So, a CSP offering a SaaS cloud service may include interfaces to it within a separate PaaS cloud service offering.

EXAMPLE A CSP that offers a standalone mapping and navigation application and allows CSC-provided applications built on their PaaS to natively invoke the mapping and navigation functions from customer's own code.

8.7 User interfaces for PaaS

Most PaaS implementations will include multiple user interfaces for the following purposes.

a) For the CSD. This will be used when the CSD is directly creating or editing code that already resides on the PaaS. This interface is also likely to include preparation, verification, debugging, and monitoring tools that allow the CSD to compile, test, and debug their code in the PaaS environment.

b) For operations managers. This will be used by whoever is administering the application, such as moving code from the staging environment to the live environment, managing the deployment of security fixes and other updates, and eventually retiring the application at the end of its life.

c) The PaaS may offer a programmable UI or UI components that can interact with the end-users of the hosted application. This can act like a web server to a browser, to a mobile device app or desktop application, or in some other way. For this approach, the PaaS can provide UI elements that the CSD's code can invoke for the CSU to interact with.

As with any user interface, CSDs should also give due consideration to the needs of those with disabilities of various kinds. This is true no matter whether the end users will be employees or members of the public. In many jurisdictions there are legal or procurement requirements that are to be followed. See 15.4.

9 Serverless architectures

9.1 General

Serverless architecture is a way of designing and running applications without having to manage servers. It is unique because it allows CSDs to focus on the business logic and functionality of their applications, rather than the infrastructure and scaling. Serverless architecture also offers benefits such as lower costs, faster deployment, and greater scalability. Some of the key features of serverless architecture are:

a) It uses ephemeral containers, such as Function-as-a-Service (FaaS), to execute custom code in response to events.

b) It only charges for the resources that are actually used by the application, rather than the provisioned capacity. This means that when the application is idle, there is no cost incurred.

c) It automatically scales up and down according to the demand, without requiring manual intervention or configuration.

In a serverless architecture, the CSD can write code or make use of code without having to deal with the management of servers, containers, or instances of the running code. These issues are all handled in the underlying execution environment and managed by the CSP.

See also ISO/IEC/TS 23167.

9.2 Function as a Service (FaaS)

Function as a Service is a serverless library of one or more common functions that can be invoked by events elsewhere in the environment or by calls from other parts of the application.

In many cases, a FaaS will be instantiated by the CSP within an ephemeral Container that only executes while the FaaS is being used.

A FaaS can be shared between multiple applications. A function of this kind normally produces or returns an immediate or near-immediate result to the calling application, though this can be delivered by asynchronous communications such as a message queue.

A FaaS can be used for many things, such as providing an interface to underlying or special features of the platform, carrying out database operations, or common mathematical or analytical tasks.

See also ISO/IEC/TS 23167.

9.3 Stateless functions

Stateless functions are those wherein the function or library retains no memory of the calling application or of any function calls made. The value of this is that subsequent calls to the stateless function can be made to a different instance (though this is not always visible to the CSD) since each call is complete in itself and nothing needs to be carried from one to the next (though this can still be done as a parameter to each call).

See also ISO/IEC/TS 23167.

10 Applications that leverage cloud native computing

10.1 Background

Cloud native computing is a way of developing, deploying, and running software applications that exploits the benefits of cloud computing. Applications developed using cloud native computing are composed of small, independent cloud services that can scale, update, and recover quickly and easily. These applications use tools and techniques such as containers, microservices, serverless functions, cloud native processors, immutable infrastructure, and declarative code. Such applications are designed to run on any cloud platform, whether public, private, or hybrid. Cloud native computing aims to increase efficiency, reduce cost, ensure availability, and foster innovation.

Cloud native computing is not to be confused with the older term "cloud native application", which remains valid and can be used to describe any application that is explicitly designed to run within and to take advantage of the capabilities and environment of cloud services. The term "cloud native application" is not limited to those developed using the "cloud native computing" approach described here.

As the term implies, cloud native computing is about applications that are primarily built for cloud computing environments. This is in stark contrast to the more traditional approach where workloads that used to run in traditional on-premises environments are lifted with minimal modification and brought to cloud computing environment to be executed in virtual machines (VMs). Cloud native computing typically does not expose virtualization to the developers; use of virtual machines happens under the hood and is managed automatically by the cloud-native environment, freeing the developers to focus on their application logic and functions, without having to explicitly address scalability requirements of their software, as well as underlying networking and security concerns.

10.2 Intended Advantages

Containerization of application modules is a modern development technology for all environments such as on-premises and traditional client and server architecture, IoT, edge and cloud computing. In cloud computing, however, containerization shines even more when it is coupled with comprehensive container orchestration, simplifying developers' challenge of building complex and scalable cloud applications.

Such cloud native capabilities liberate the developers from having to take into consideration the complexities of managing virtual machines, their numbers based on spontaneous usage demand, the complexities of their maintenance, updates, network connectivity and security. All of that is taken care of by the underlying cloud computing platform and infrastructure in cloud-native environments. This is similar to how traditional applications are freed from having to be aware of I/O ports of the computer, the details of the file system and the hardware storage components onboard. All of that is handled under the cover by the operating system.

10.3 Orchestration

Container orchestration manages the mounting of application code from containers and preparing them to execute in their declared execution environment of choice. The orchestrator loads and executes the application code from its containers, while managing the resource demands of the application modules dynamically throughout the life of the cloud application.

10.4 Architecture

Figure 6 illustrates the key differences between a traditional VM-based architecture and a cloud native architecture.

In an application architecture based on VM, the CSC has more responsibility and complexity to manage. Due to the nature of VMs, the CSC will have to design and manage all of the software included in each VM, and have responsibility for securing and patching all of it including their chosen guest operating system. This allows the CSP to hand off much of the responsibility for compliance and robustness to the CSC, thus allowing them to focus on hardware.

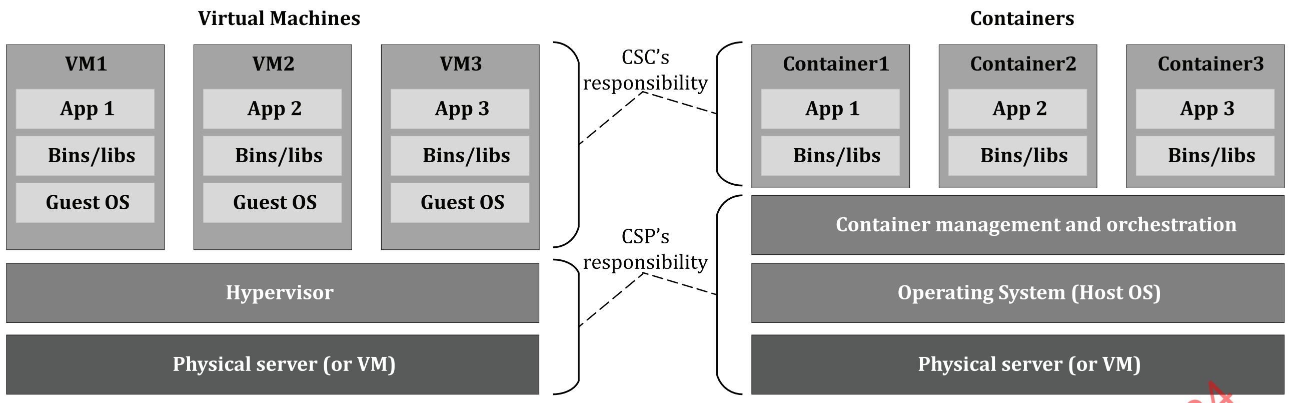


Figure 6 — Examples of contrasting VM-based and cloud-native application stacks

By contrast, the container-based cloud native architecture pattern allows the CSC to concentrate on the unique parts of their application that serve their business needs, especially since there is no guest OS to be configured, managed, and secured. The CSC can therefore eliminate significant costs associated with OS maintenance.

Thanks to the emergence of cloud native technologies, the crisp differentiation between IaaS and PaaS service categories are fading (as noted in 7.3.2). The more traditional IaaS and PaaS service categories were invented to facilitate migration from an on-premises VM environment to cloud computing. But now most new projects are designed to run in cloud native environments.

See Annex B for an example of one popular approach to implementation.

11 General considerations for platform capabilities and PaaS

11.1 General

The following considerations apply to all cloud services offering platform capabilities type. These largely though not entirely correspond to the "cloud computing cross cutting aspects" described in ISO/IEC 22123-2.

11.2 Auditability

See ISO/IEC 22123-2:2023, 7.2.

11.3 Availability

See ISO/IEC 22123-2:2023, 7.3.

11.4 Governance

See ISO/IEC 22123-2:2023, 7.4.

11.5 Interoperability

See ISO/IEC 22123-2:2023, 7.5.

Cloud services offering platform capabilities type will often need to enable communications between CSC-provided application code and other software applications located on the CSC's premises or in other cloud services, included those offered by other CSPs. This is commonly achieved by using industry standards such as protocols defined by the IETF or the W3C, though specialised protocols are also defined for some specific industry verticals.

See also ISO/IEC 19941.

11.6 Maintenance and Versioning

See ISO/IEC 22123-2:2023, 7.6.

See also 7.2.2, 7.5.7, 7.5.9, 7.5.10, 7.5.11 of this document.

11.7 Performance

See ISO/IEC 22123-2:2023, 7.7.

11.8 Portability

See ISO/IEC 22123-2:2023, 7.8. See also Clause 12 of this document.

11.9 Protection of PII (privacy)

See ISO/IEC 22123-2:2023, 7.9.

In many cases a CSP offering platforms capabilities type will be completely unaware of the existence or location of any PII within the data stored and processed by a CSC's application code. As such, the primary responsibility for the projection of PII rests with the CSC and the CSC's CSDs.

11.10 Regulatory

See ISO/IEC 22123-2:2023, 7.10.

In addition, CSPs, CSNs, and CSCs need to be aware of regulatory developments in their own and other relevant jurisdictions, such as new regulations covering switching and portability, data storage and data sharing, emergency services, accessibility, and others.

11.11 Resiliency

See ISO/IEC 22123-2:2023, 7.11.

This issue can be mitigated by adopting a multi-instance approach to application design, perhaps including instances running in different geographical locations so that a major outage in one region (such as an earthquake or the accidental cutting of a major fibre-optic link) does not bring down the entire application.

See also Annex A for improving resiliency through the use of multiple CSPs.

11.12 Reversibility

See ISO/IEC 22123-2:2023, 7.12.

Reversibility is the specific requirement that a customer be able to move an application and its data back from a cloud service into their on-premises environment. For those customers with this requirement, planning is needed to ensure that a compatible environment exists on-premises, and current code versions are always kept available for deployment on-premises, since any outage at the CSP may make it difficult or impossible to retrieve the versions running there.

11.13 Security

11.13.1 General

See ISO/IEC 22123-2:2023, 7.13.

In addition to the general cases, there are two specific issues for cloud services offering platform capabilities type.

Maintaining security is always a shared responsibility between the CSC and CSP. Each is also responsible for the behaviour and security of any CSN or other subcontractor using the system on their behalf.

11.13.2 Malware avoidance

There is always a danger that a CSC, or some third-party who has obtained access to CSC credentials, attempts to use the cloud service either to launch cyberattacks directly (such as installing code that creates a Denial of Service attack) or to inject malware (such as their code modifying files that will ultimately be sent to end users), or for some other such breach. The CSP therefore has a responsibility to watch for malicious behaviour by CSC-provided code, and as far as possible prevent malicious code from running in their cloud service.

11.13.3 Code access to resources

It is also essential that CSC-provided code is denied access to resources (such as stored data, file systems, other code, network connections) unless it has a genuine need to access them and does so correctly. As such, all CSC-provided code needs to be treated as untrustworthy. The established cybersecurity principle of “least necessary access” always applies.

11.14 Service levels and service level agreements

See ISO/IEC 22123-2:2023, 7.14.

11.15 Sustainability

CSC-provided code running in a cloud service is often not written with energy conservation in mind, or the CSDs creating the code might not have the skills to write the most energy efficient algorithms. As such, the CSP should plan for CSC-provided applications to be rather more energy hungry than those written by fully skilled cloud developers.

It is therefore advantageous to all parties if the CSP is able to detect and identify application elements that “run hot” so that the CSC can be alerted to this, and perhaps their CSDs educated on how to write more efficient code algorithms that will achieve the same result with less power.

12 Portability between PaaS Cloud services

12.1 General

ISO/IEC 19941:2017, Clause 9 describes the types of cloud application portability based on cloud capabilities types for each facet of application portability. Each cell at the intersection of the rows and columns of the table is a cloud application portability type and it is described in the indicated subclause in that document. Each clause describes what needs to be considered for the porting of the application. Notice that in some cases, there is a need to differentiate between application portability from non-cloud deployments to cloud services as opposed to cloud service to cloud service scenarios.

One specific concern arises when third-party software is licensed for a specific purpose which may not permit porting the application code to or between cloud service providers. This is a legal matter beyond the scope of this document, however the issue is described in ISO/IEC 19941:2017, 9.3.1.6.

12.2 Endpoint identification

No software is an island. Any code is going to connect to other software applications or devices, so it is important to identify all of those that are relevant.

It is also necessary to understand the security mechanisms used on the existing connection, such as any application credentials, authorisations, certificates, etc.

For each endpoint concerned, it is important to plan how the current connections will change, what new authorisations need to be obtained, and the credentials to be used. It is not enough to simply substitute in a new set of addresses. This includes third-party endpoints that are outside both the origin and destination cloud services.

It is also important to plan for the closure of all the original endpoints, and the removal of any credentials associated with them, so as not to leave a potential security vulnerability.

12.3 Portability-ready development

Porting of CSC application code from one PaaS environment to another can be made somewhat easier if the initial development of the code adopts a “portability-ready” approach.

This can work by “wrapping” all calls to platform-specific functionality within specific modules or libraries. So, a CSC might build a library of Identity and Access Management functions that can be used by any of their code, such that only this library contains any calls to the IAM of the current PaaS. In this way, if a decision is made to move to another PaaS, this library is the only part that needs to be changed to use a different IAM system. Of course, that doesn’t address how the actual identity data is moved between the two IAM systems, which is out of scope for this document. Such a “wrapper” library is often best constrained to the minimum functions needed, as any PaaS-specific functions may be difficult or impossible to replicate in another PaaS.

Wrapping PaaS-specific functions into a small set of libraries or other elements means that the majority of the code doesn’t need to be reworked for a new environment, which can reduce the cost and time needed to port the application. However, that can also mean that the application is unable to take advantage of any special or high-value functions that the PaaS provides. Thus, taking this approach is quite a strategic decision that needs careful thought. If the development approach is not portability-ready, CSDs have greater freedom and access to richer functionality, but they are at risk of locking themselves in to the original PaaS in the process.

Maintaining such a “wrapper” for an evolving application on an evolving PaaS can itself become a considerable development, maintenance, and cost burden for the CSC even without any actual attempt to port the application to a new environment, so the cost and effort for this needs to be taken into account when planning such a strategy.

12.4 Addressing “Exit strategy” requirements

See also 11.12 and ISO/IEC 22123-2:2023, 7.12.

“Exit strategy” requirements appear in regulations covering certain vertical industries, notably in financial services, wherein a bank or financial institution may be obliged to have an exit strategy from each of their suppliers (including their CSPs) to ensure continuation of the financial services in the event that such a supplier fails.

However, an “exit strategy” requirement does not generally require the ability to migrate an application to another CSP, only the ability to bring the essential functionality out of the current CSP, which typically means returning to an on-premises solution. This may be simpler than moving to an unknown new environment, though it does require the CSC to maintain their own datacentre capacity sufficient to the task. So, a CSD working in a business environment wherein this is a requirement needs to be careful with the features and functions they choose to employ from the PaaS, since the wrong choice may make it difficult or impossible to prove that they can meet the Exit strategy obligations.

This means they need to rely on the “Reversibility” of their chosen cloud services per 11.12.

12.5 Portability through container orchestration

Possibly the simplest and most practical way to retain the option of portability for a new PaaS based application is to employ a “cloud native” approach using orchestration technology to control a multi-cloud deployment model of software containers across multiple CSPs. This allows elements of a workload to be moved between CSPs with minimal effort. This approach is described in Clause 10, Annex A, and Annex B.

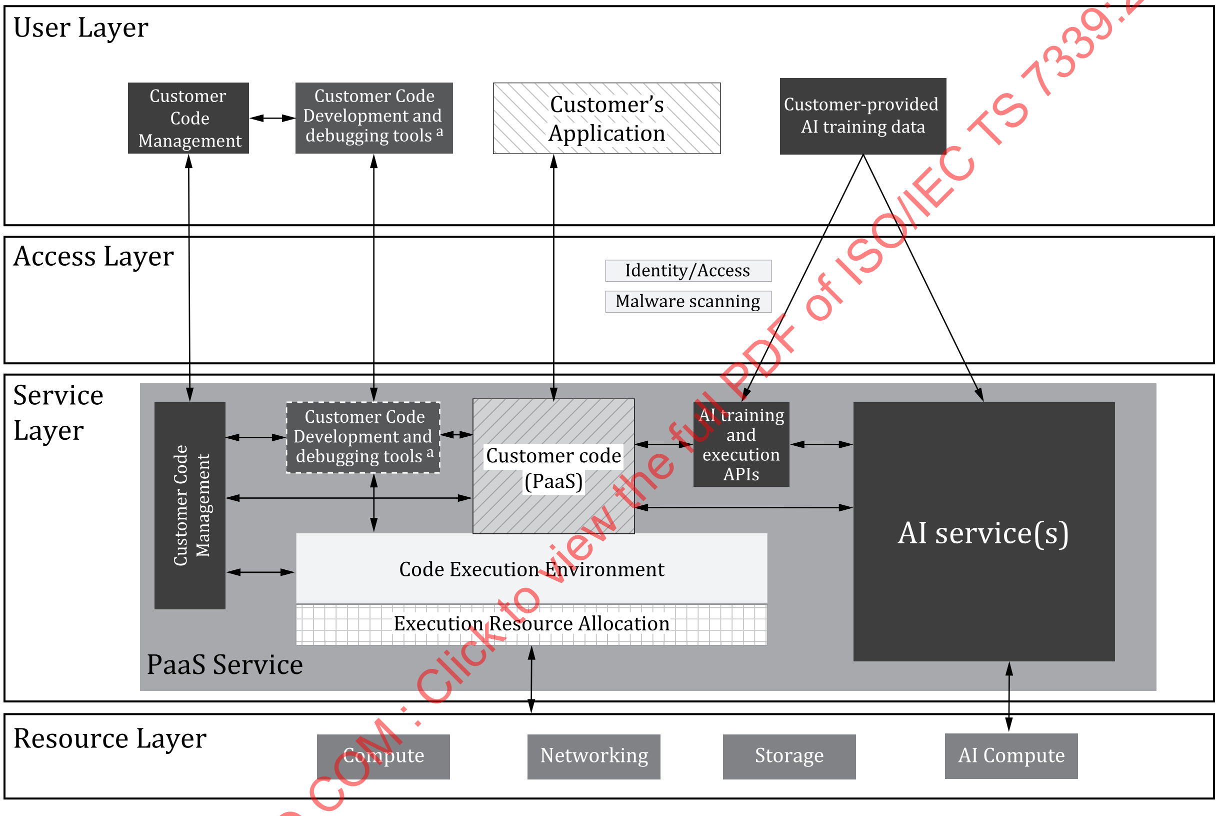
13 Related technologies and platform capabilities type

13.1 General

New cloud services and computing capabilities have appeared and will continue to do so. Often, these provide additional functions that can be incorporated into existing and new cloud applications. In this sense, PaaS or other services offering platform capabilities type can be the central platform from which these new technologies and capabilities can be introduced into customer applications.

13.2 Artificial Intelligence (AI)

The generic PaaS model shown in Figure 4 can be extended to include AI/Machine Learning functions (MLaaS) as follows.



[a] Optional.

NOTE 1 This diagram is based on and is to be interpreted in the context of ISO/IEC 22123-3.

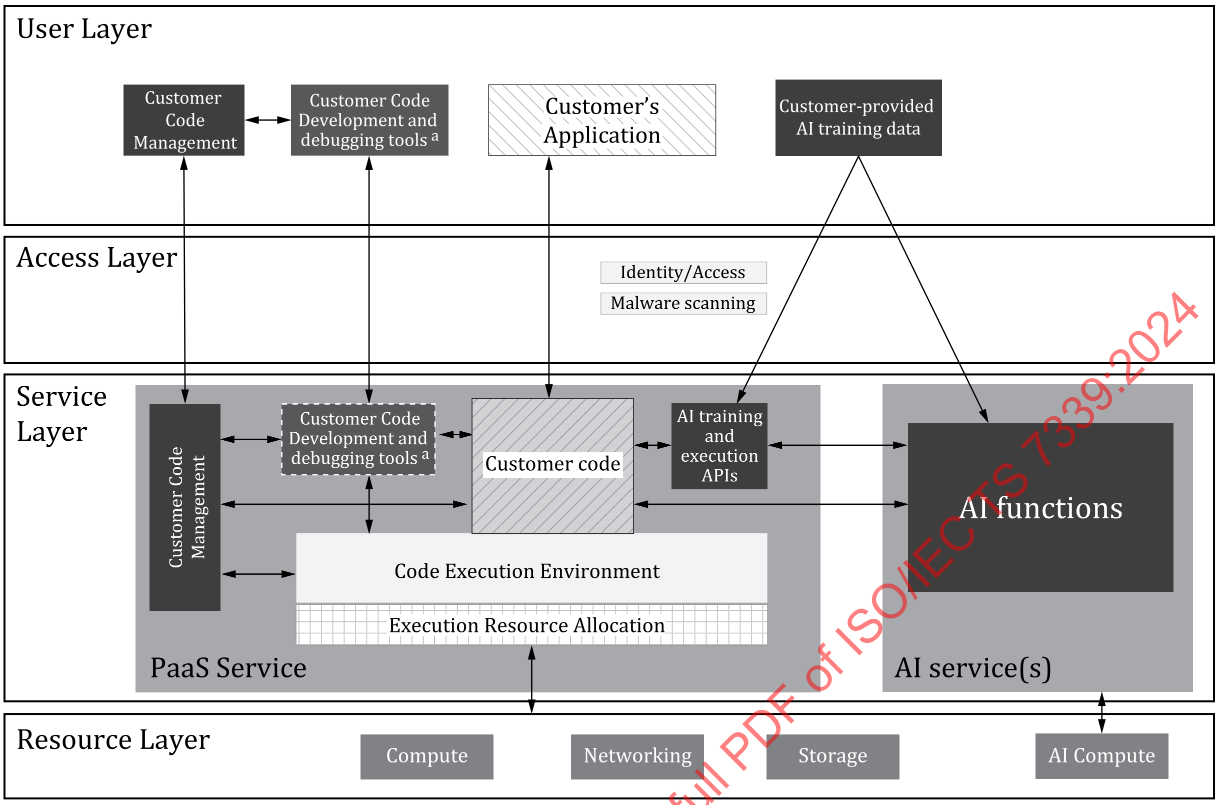
NOTE 2 "AI Compute" indicates specialised hardware to support AI functions, such as the use of FPGAs, GPUs or GPU-like processors.

Figure 7 — Use of AI functions within a PaaS service

Figure 7 shows an example of how AI services can be implemented within a PaaS cloud service and made available directly to customer code running on the platform provided by the PaaS cloud service.

However, as shown in Figure 8, AI functions are also often implemented in a separate cloud service from the PaaS execution environment, not least because they often have different hardware resource requirements that go beyond general purpose CPUs. For example, many AI functions are implemented using GPU hardware to allow for far greater parallel processing across large datasets. These are represented in the figures as "AI Compute". As with all reference architecture elements, these are an optional implementation choice.

In some cases, a CSD using a PaaS may wish to invoke AI functions provided by a different CSP, and thus from a different cloud service.



[a] Optional.

NOTE This diagram is based on and is to be interpreted in the context of ISO/IEC 22123-3.

Figure 8 — Use of external AI functions from within a PaaS

Figure 8 shows an example of how an external AI function, perhaps offered by a different cloud service provider (or as a separate cloud service from the same CSP), could be employed via a PaaS implementation. This does not preclude the AI function also having its own user interface or other means of being operated or managed, which are out of scope for this document.

See also ISO/IEC 22989.

AI/machine learning systems may require the CSC to also provide training data in addition to logical code. For some AI applications, the AI components will already have been trained by the CSP. In other cases, they can be blank and require the CSC to do all the training. Mixed cases are also possible, in which both CSP and CSC contribute to training the AI. Some AI applications will also learn based on data they are given or observe while performing their task. It is therefore useful to represent the presence of such data in the extended architecture.

13.3 PaaS and generative AI

A good example of an external AI function is the capability of Generative AI. A CSC writing code to run on one CSP may choose to invoke a more advanced Generative AI capability from a specialist AI CSP to take advantage of their large language model (LLM) built on a very specialised GPU-based datacentre architecture.

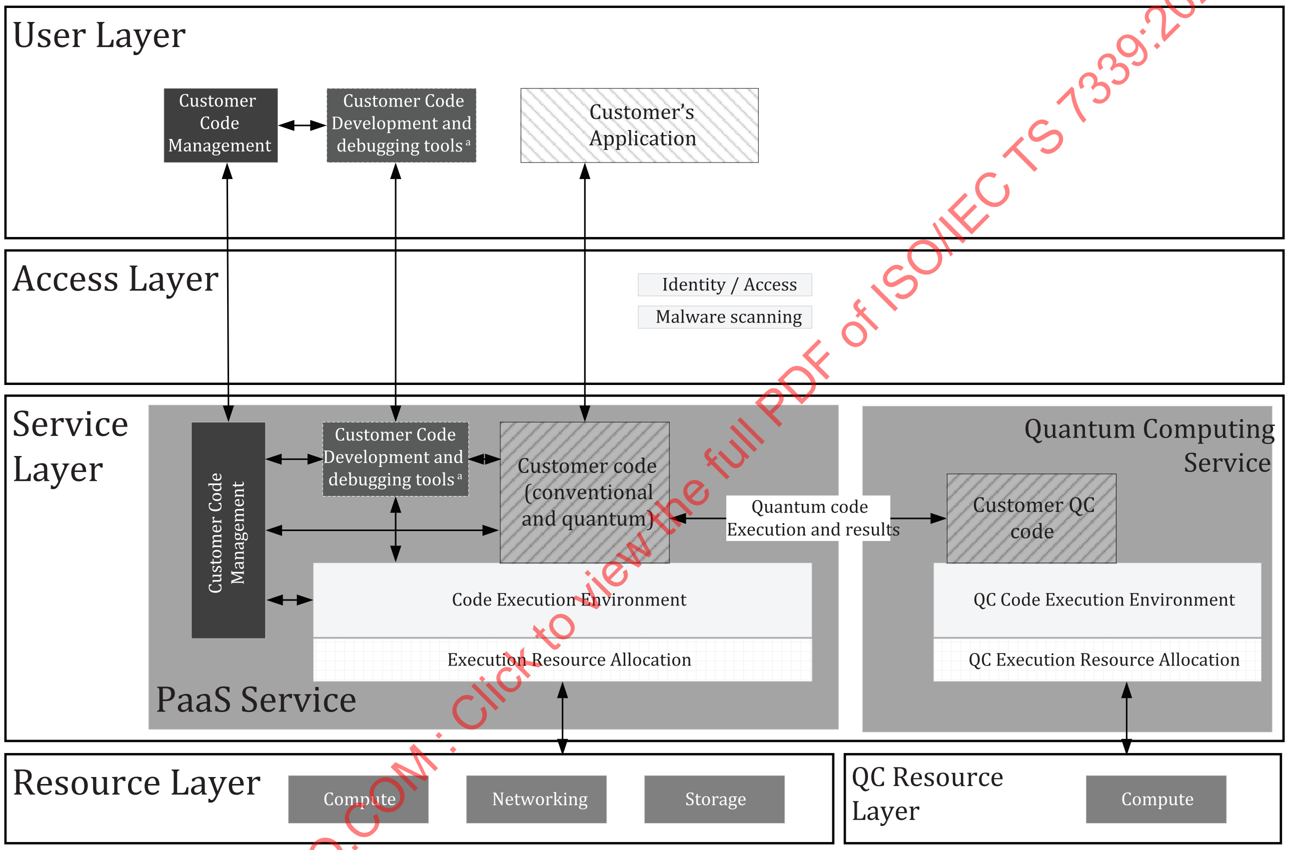
Such a CSC application might take user input, then provide additional input to construct a richer prompt that can be sent to the LLM for processing. Depending on the application, both input to the LLM and output from the LLM might need to be further processed by CSC code for security, privacy, and other compliance, and other purposes.

The reference architecture for such an application would remain as shown in Figure 8.

13.4 Quantum Computing (QC)

Similar to what is shown in 13.2, we take a similar approach to modelling the use of a Quantum Computing service (QCaaS) from a PaaS.

True QC requires very specialised hardware. While early systems are likely to be limited to Compute functions, we can also anticipate quantum-based storage and network hardware also being developed at some point in the future, thus providing a parallel set of resources to those of conventional computing.


User Layer
Customer Code Management
Customer Code Development and debugging tools [a]
Customer's Application
Access Layer
Identity / Access
Malware scanning
Service Layer
Customer Code Management
Customer Code Development and debugging tools [a]
Customer code (conventional and quantum)
Quantum code Execution and results
Quantum Computing Service
Customer QC code
Code Execution Environment
Execution Resource Allocation
QC Code Execution Environment
QC Execution Resource Allocation
PaaS Service
Resource Layer
Compute
Networking
Storage
QC Resource Layer
Compute

a Optional.

NOTE This diagram is based on and is to be interpreted in the context of ISO/IEC 22123-3.

Figure 9 — Use of QC from within a PaaS

Figure 9 shows an example of how a QCaaS function could be employed via a PaaS implementation. This does not preclude the QCaaS function also having its own user interface or other means of being operated or managed, which are out of scope for this document.