International Telecommunication Union

# ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

# Y.3535

(02/2022)

SERIES Y: GLOBAL INFORMATION INFRASTRUCTURE, INTERNET PROTOCOL ASPECTS, NEXT-GENERATION NETWORKS, INTERNET OF THINGS AND SMART CITIES

Cloud Computing

## Cloud computing – Functional requirements for a container

Recommendation ITU-T Y.3535

ITU-T Y-SERIES RECOMMENDATIONS

**GLOBAL INFORMATION INFRASTRUCTURE, INTERNET PROTOCOL ASPECTS, NEXT-GENERATION NETWORKS, INTERNET OF THINGS AND SMART CITIES**

| | |
|---|---|
| GLOBAL INFORMATION INFRASTRUCTURE | |
| General | Y.100–Y.199 |
| Services, applications and middleware | Y.200–Y.299 |
| Network aspects | Y.300–Y.399 |
| Interfaces and protocols | Y.400–Y.499 |
| Numbering, addressing and naming | Y.500–Y.599 |
| Operation, administration and maintenance | Y.600–Y.699 |
| Security | Y.700–Y.799 |
| Performances | Y.800–Y.899 |
| INTERNET PROTOCOL ASPECTS | |
| General | Y.1000–Y.1099 |
| Services and applications | Y.1100–Y.1199 |
| Architecture, access, network capabilities and resource management | Y.1200–Y.1299 |
| Transport | Y.1300–Y.1399 |
| Interworking | Y.1400–Y.1499 |
| Quality of service and network performance | Y.1500–Y.1599 |
| Signalling | Y.1600–Y.1699 |
| Operation, administration and maintenance | Y.1700–Y.1799 |
| Charging | Y.1800–Y.1899 |
| IPTV over NGN | Y.1900–Y.1999 |
| NEXT GENERATION NETWORKS | |
| Frameworks and functional architecture models | Y.2000–Y.2099 |
| Quality of Service and performance | Y.2100–Y.2199 |
| Service aspects: Service capabilities and service architecture | Y.2200–Y.2249 |
| Service aspects: Interoperability of services and networks in NGN | Y.2250–Y.2299 |
| Enhancements to NGN | Y.2300–Y.2399 |
| Network management | Y.2400–Y.2499 |
| Computing power networks | Y.2500–Y.2599 |
| Packet-based Networks | Y.2600–Y.2699 |
| Security | Y.2700–Y.2799 |
| Generalized mobility | Y.2800–Y.2899 |
| Carrier grade open environment | Y.2900–Y.2999 |
| FUTURE NETWORKS | Y.3000–Y.3499 |
| **CLOUD COMPUTING** | **Y.3500–Y.3599** |
| BIG DATA | Y.3600–Y.3799 |
| QUANTUM KEY DISTRIBUTION NETWORKS | Y.3800–Y.3999 |
| INTERNET OF THINGS AND SMART CITIES AND COMMUNITIES | |
| General | Y.4000–Y.4049 |
| Definitions and terminologies | Y.4050–Y.4099 |
| Requirements and use cases | Y.4100–Y.4249 |
| Infrastructure, connectivity and networks | Y.4250–Y.4399 |
| Frameworks, architectures and protocols | Y.4400–Y.4549 |
| Services, applications, computation and data processing | Y.4550–Y.4699 |
| Management, control and performance | Y.4700–Y.4799 |
| Identification and security | Y.4800–Y.4899 |
| Evaluation and assessment | Y.4900–Y.4999 |

*For further details, please refer to the list of ITU-T Recommendations.*

# Recommendation ITU-T Y.3535

## Cloud computing – Functional requirements for a container


### Summary

Recommendation ITU-T Y.3535 provides an overview and functional requirements for a container in cloud computing. It describes the technical aspects of a container and provides the relationship between containers and cloud computing. It also provides functional requirements for a container in terms of its engine, management system and cloud computing support.



### History

| Edition | Recommendation | Approval | Study Group | Unique ID* |
|---|---|---|---|---|
| 1.0 | ITU-T Y.3535 | 2022-02-13 | 13 | 11.1002/1000/14860 |

### Keywords

Cloud computing, container, container engine, container image, container management system.


* To access the Recommendation, type the URL http://handle.itu.int/ in the address field of your web browser, followed by the Recommendation's unique ID. For example, http://handle.itu.int/11.1002/1000/11830-en.

## FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

## NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure, e.g., interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

## INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had received notice of intellectual property, protected by patents/software copyrights, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the appropriate ITU-T databases available via the ITU-T website at http://www.itu.int/ITU-T/ipr/.

## Table of Contents

# Recommendation ITU-T Y.3535

## Cloud computing – Functional requirements for a container

### 1 Scope

This Recommendation provides an overview and functional requirements for a container in cloud computing, including:

- an overview of a container including concept and technical aspects;
- the container in cloud computing;
- functional requirements for a container;
- use cases of a container.

### 2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

[ITU-T Y.3502] Recommendation ITU-T Y.3502 (2014), *Information technology – Cloud computing – reference architecture.*

### 3 Definitions

#### 3.1 Terms defined elsewhere

This Recommendation uses the following terms defined elsewhere:

**3.1.1 application** [b-ITU-T H.764]: A functional implementation realized as software running in one or spread over several interplaying hardware entities.

**3.1.2 cloud service** [b-ITU-T Y.3500]: One or more capabilities offered via cloud computing invoked using a defined interface.

**3.1.3 cloud service customer** [b-ITU-T Y.3500]: Party which is in a business relationship for the purpose of using cloud services.

NOTE – A business relationship does not necessarily imply financial agreements.

**3.1.4 cloud service provider** [b-ITU-T Y.3500]: Party which makes cloud services available.

**3.1.5 hypervisor** [b-ITU-T Y.3510]: A type of system software that allows multiple operating systems to share a single hardware host.

**3.1.6 virtual machine** [b-ITU-T Y.3504]: The complete environment that supports the execution of guest software.

#### 3.2 Terms defined in this Recommendation

This Recommendation defines the following terms:

**3.2.1 container**: A set of software to provide isolation, resource control and portability for virtualization processing of an application.

NOTE 1 – A container runs on the kernel in a bare-metal machine or virtual machine.

NOTE 2 – "Application" implies business logic including a required library or binary to run in a container.

**3.2.2 container image**: A software package configured to execute all or part of an application for a container.

NOTE – A software developer packages up all or the parts of applications into a container image.

## 4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

API Application Programming Interface

CD Continuous Deployment

CI Continuous Integration

CLI Command Line Interface

CMS Container Management System

CPU Central Processing Unit

CSC Cloud Service Customer

CSN Cloud Service partner

CSP Cloud Service Provider

GUI Graphical User Interface

IaaS Infrastructure as a Service

ID Identifier

I/O Input/Output

NaaS Network as a Service

NVMe Non-Volatile Memory express

OS Operating System

PaaS Platform as a Service

RAM Random Access Memory

SaaS Software as a Service

SLA Service Level Agreement

SSD Solid-State Drive

VM Virtual Machine

## 5 Conventions

In this Recommendation:

The phrase "**is required to**" indicates a requirement that must be strictly followed and from which no deviation is permitted if conformance to this Recommendation is to be claimed.

The phrase "**is recommended**" indicates a requirement that is preferred but which is not absolutely required. Thus, this requirement need not be present to claim conformance.

# 6 Overview of container

## 6.1 Concept of container

A container enables software developers to rapidly develop and deploy applications using container images. A container executes an application on top of a host operating system (OS) and it requires fewer resources than a virtual machine (VM) because it uses the resources of the host as necessary to run the application. A container also provides isolation, resource control and portability for virtualization processing of applications as follows:

– **Isolation**: A container assigns a separate kernel space and individual processor to each application used for its isolation in the container from other applications on a host.

– **Resource control**: To control resources, a container sets namespaces that allocate resources.

NOTE 1 – A namespace includes network namespace, processor namespace, mount namespace and user namespace. A namespace isolates system resources when a process is running. A namespace also provides an independent space for each container and prevents collisions with others in the kernel system.

– **Portability**: A container image developed by a software developer is built by all or part of an application. To use a container image, it is pushed to a container registry to share with other hosts to support portability. The container image uploaded to the container registry is pulled to execute.

Figure 6-1 shows the usage of an application, container image and container registry for a container. An OS either on a bare metal server or as a guest on a VM is included on the host.

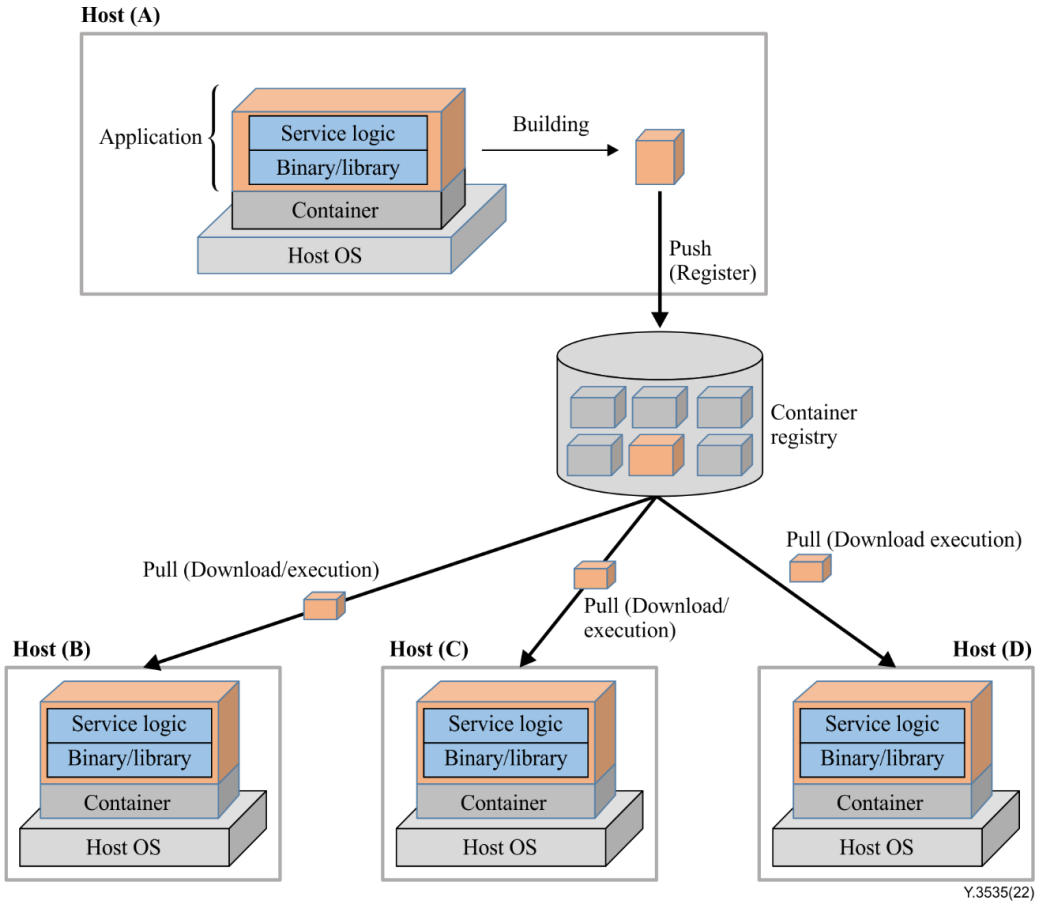



**Figure 6-1 – Usage of application, container image and container registry for a container**

The container registry in Figure 6-1 is a repository for container images in order for the container engine to store and access them. "Push" refers to the upload of a container image to hosts in the container registry and "pull" refers to the download of the container image from a container registry.

NOTE 2 – The container registry includes local registry, private registry and public registry.

NOTE 3 – The container registry works in the form of master-slave architecture. The master registry has authentication and authorization to access the slave registry. The master registry also supports configuring and updating image synchronization between two registries.

## 6.2 Technical aspects of a container

### 6.2.1 Container engine

The container engine is a group of kernel processes to administer the execution from the container image on a isolated kernel space. The container engine provides the following key features:

– execution of an application from the container image;

– configuration of the isolated kernel space, as well as the securing process.

The container engine configures an isolated kernel space for the independent environment per application. The isolated kernel space is logically separated by namespaces. The isolated kernel space provides resource allocation, file system mounting or unmounting, allocation of independent processes and networks.

To execute an application in a container, the container works as follows:

– pulling the container images from a container registry;

– unpacking container images into a file system;

– setting file system ownership;

– running applications using a kernel process;

– exposing the external links of the application for user access.

The configuration of the isolated kernel space includes:

– setting a directory of a container file system for container image;

– setting the limitation of memory, central processing unit (CPU) and device access;

– creating the application's own namespace for resources;

– configuring to prevent a system call that is not allowed;

– setting the processor's secure access.

The isolated kernel space supports independent communication path allocation between processes and assigns the independent hostname and users.

### 6.2.2 Container image

A container image is a software package configured to execute an application for a container. The container image includes the following:

– **image object**: a group of files to execute an application including library, file system and binary at layers in the container file system;

NOTE 1 – An image object is provided with the archived and compressed format.

– **image manifest**: the information about a hierarchy of container images according to the file system and OS;

NOTE 2 – An image manifest provides the schema version of manifest, media type, image size and image identification.

– **container file**: the information about a container image for use with a container engine such as the execution commands, resource information, service port, user, the architecture of host machine and its relationship to file system changes in a container image.

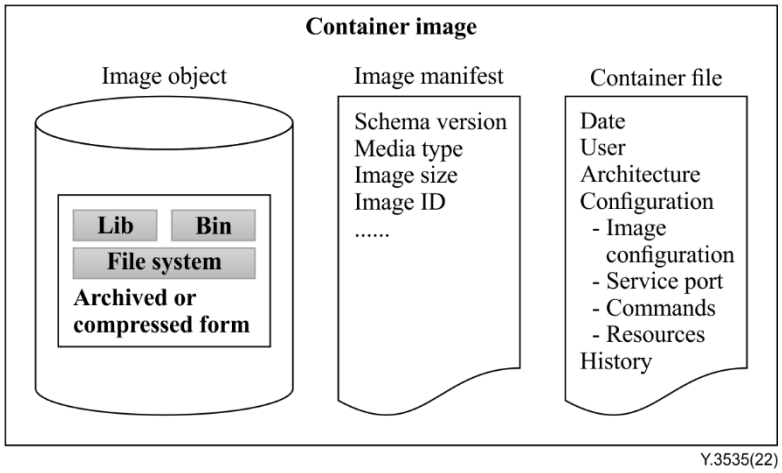



**Figure 6-2 – The example of container image**

### 6.2.3 Container file system

The container file system is composed of directories and files per container. The container file system also is managed by the container engine to use container images.

NOTE – The container file system is provided in the host's storage device such as main memory, solid-state drive (SSD) and hard-disc drive.

In Figure 6-3, the container file system provides layers that are composed of directories. A container image is located in a directory on the container layer and image layer. The application uses container images from the merged layer, which is mounted into a single directory from the container layer and image layer.

– **Image layer**: provides images from container repository. The images in this layer are shared with the merged layer to save disc space. The application accesses the container image in the image layer with read-only access rights.

– **Container layer**: provides each container's directory. The application accesses the container image in the container layer with read and write access rights.

– **Merged layer**: provides links of all files in the container layer and the image layer to application. This layer allows access to all files by application.

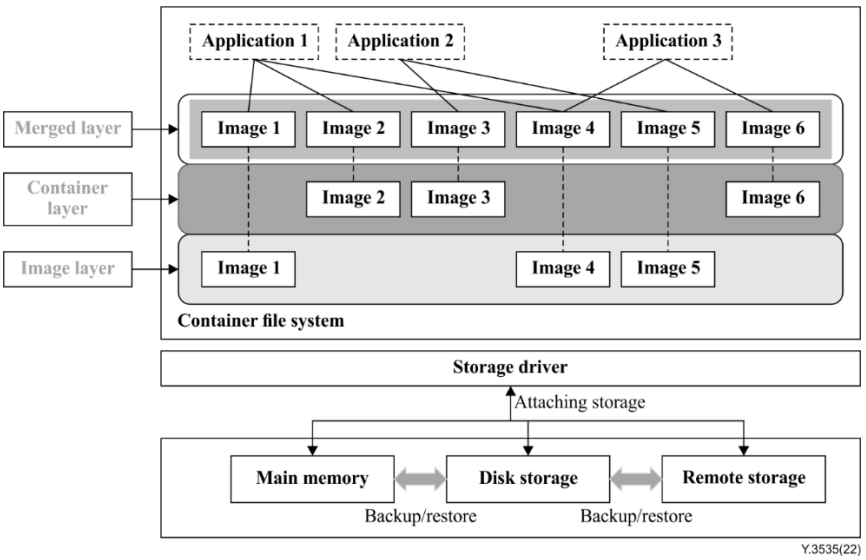



**Figure 6-3 – An example of the file system for a container**

To unify a container image in a container file system, sub-directories in the container layer and image layer are mounted for merger into the parent file system directories so that the container image can be accessed logically by applications. The container file system also provides searching of container images for the entire file system shared on the system locally.

The container image in the image layer is stored by a container registry when the container is executed. For storage, container images from the container registry are pulled by the container engine locally when the application uses the container. Also, the container image is stored in advance of execution for better performance. Once a container image is pulled, it is reused by applications and not pulled again into the image layer.

When the capacity of a file system is exceeded by container images in the image layer, the container images are backed up by storage on other discs or remotely.

### 6.2.4 Container management system

A container management system (CMS) manages a single as well as multiple containers as shown in Figure 6-4.

– **CMS for a single container**: The CMS is in charge of managing operations of the container such as execution, versioning, configuration and networking of the container through a container application programming interface (API).

– **CMS for multiple containers**: The CMS is responsible for managing containers on multiple host OSs if the application is deployed in containers on multiple hosts. The CMS supports extending containers to the distributed system and cluster if multiple containers are used simultaneously.

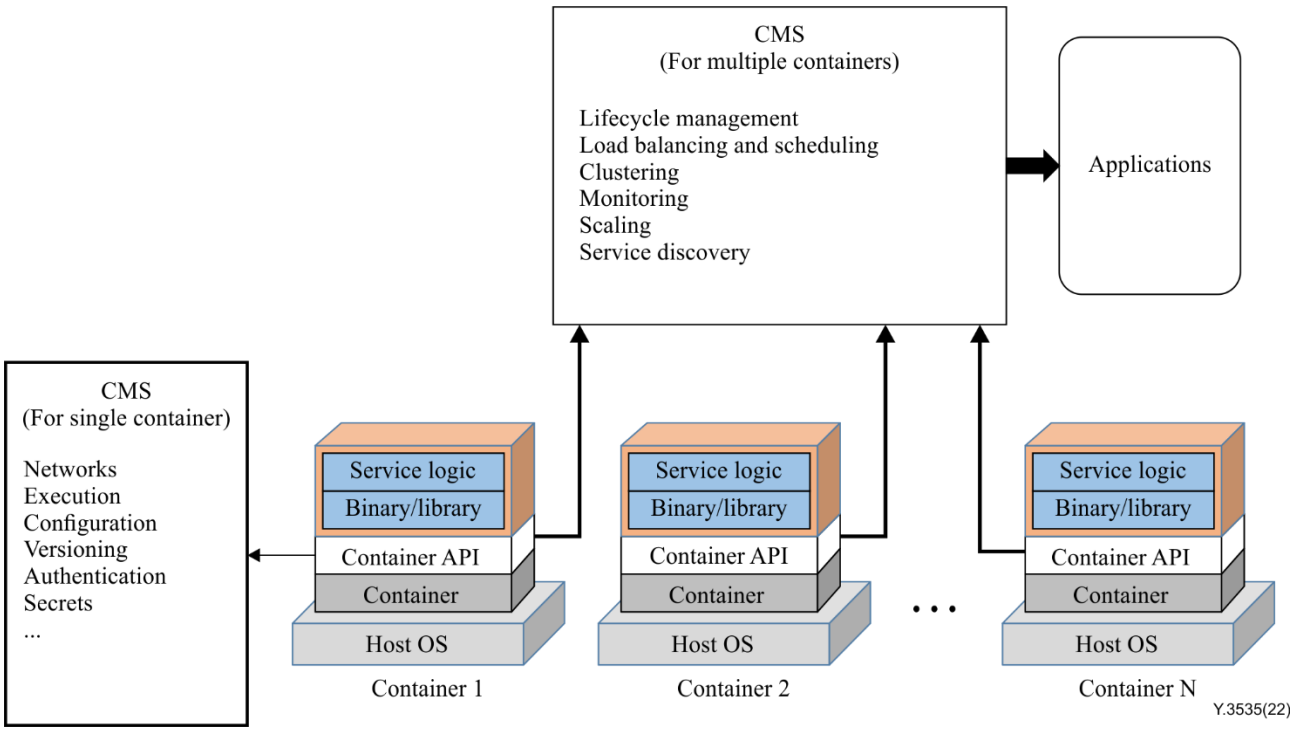



**Figure 6-4 – Concept of the container management system**

The CMS is responsible for container life cycle management, load balancing and scheduling, container clustering, monitoring, scaling and application discovery for the container as follows.

– **Lifecycle management**: a CMS manages the entire lifecycle of a container such as creation, pausing, resuming, restarting and setting configurations.

– **Load balancing and scheduling**: a CMS distributes network traffic for the application so that the deployment is stable. It works to maximize the scalability and availability of containers.

– **Container clustering**: a CMS creates a cluster by grouping multiple containers. A cluster enables the stable provision of application services by logically connecting multiple containers. The CMS replaces or restarts containers that are not working within the cluster.

– **Monitoring**: a CMS monitors the status of containers and the cluster. It monitors the availability of container and resource consumption (CPU, memory, storage, etc.) of the container.

– **Scaling**: a CMS supports scaling of a container to the cluster according to service workload.

– **Application discovery**: a CMS finds an application running in the container and the location of the container.

## 7 Container in cloud computing

A container is a kind of OS-level virtualization using virtualized resources that are supported by cloud computing. Furthermore, a number of virtual hosts are provided in cloud computing and multiple containers are deployed and operated on each virtual host.

There are two ways of container provision in cloud computing. The first is a container on a guest OS of a VM as shown in Figure 7-1-a, and the other is a container on the host OS as shown in Figure 7-1-b.

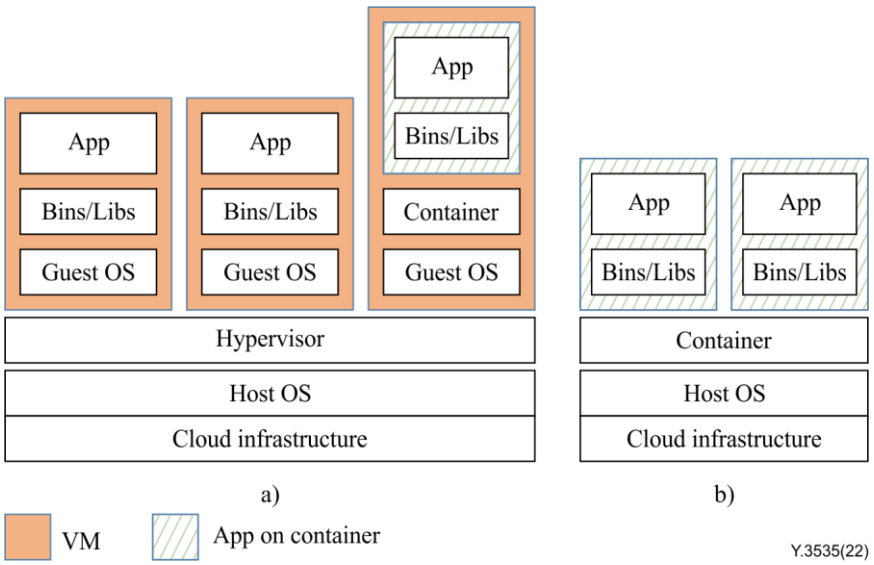
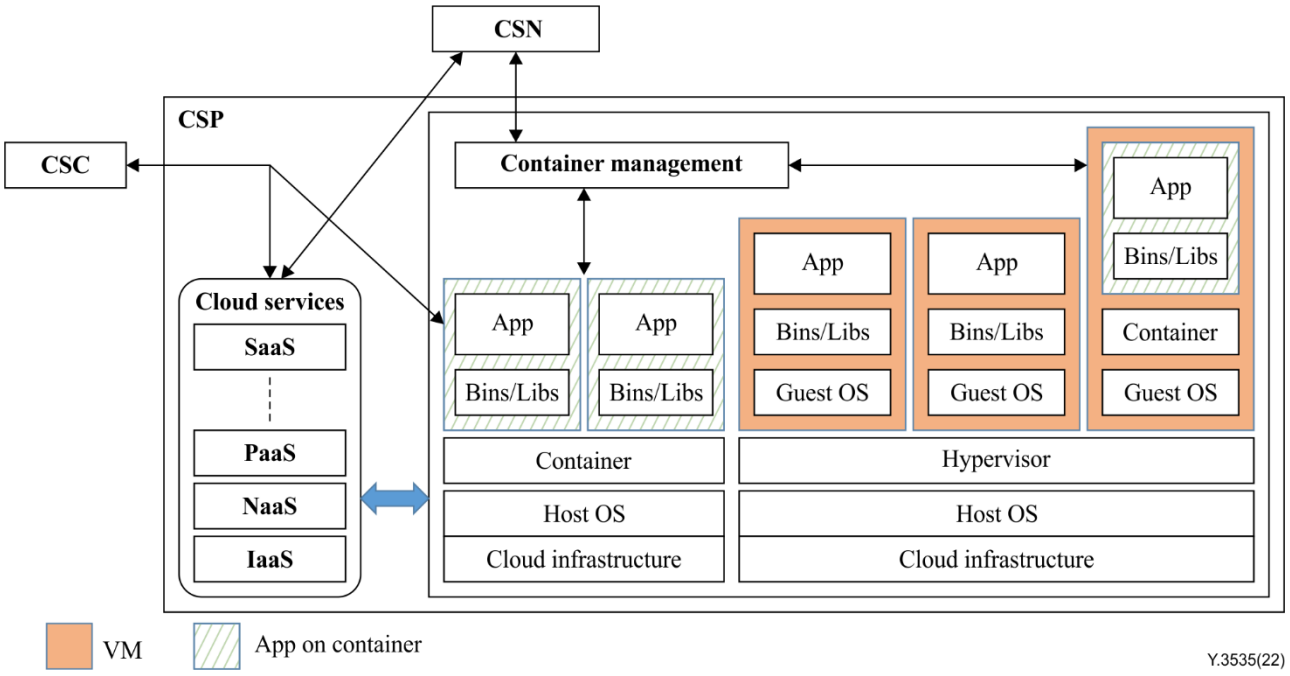



**Figure 7-1 – Container running features in cloud computing**

Figure 7-2 shows the relationship between a container and cloud computing. Cloud computing provides an infrastructure for the container. A container is operated on a host OS with or without a VM in cloud computing.




IaaS: infrastructure as a service; NaaS: network as a service; PaaS: platform as a service; SaaS; software as a service

**Figure 7-2 – The relationship between cloud computing and container**

In terms of cloud computing reference architecture [ITU-T Y.3502]:

– A cloud service provider (CSP) delivers, for example, IaaS, PaaS and SaaS with the container. Also, a container uses cloud services to support an application. An application in the container is designed while taking compatibility with the cloud service of the VM into account.

– A cloud service partner (CSN) develops the application of a container using container management and cloud service, and the CSP provides an interface to use cloud service for the convenience of development of a CSN.

– A cloud service customer **(**CSC**)** uses a cloud service with a container, as well as an application on the container.

# 8 Functional requirements for supporting a container in cloud computing

## 8.1 Functional requirements of a container

### 8.1.1 Functional requirements of a container engine

1) It is required that a container engine execute an application in the kernel.

NOTE 1 – The container engine uses a kernel processor to run the application with the execution commands in a container file.

2) It is required that a container engine provide a network interface that is isolated from the host network.

NOTE 2 – A container engine sets the network namespace according to the application.

NOTE 3 – A network namespace refers to the separation of the network used by a container and one used by a host using a virtual network interface and its routing to communicate among applications and users.

NOTE 4 – A network namespace allocates a port mapped with the host network or the new Internet protocol address to the application.

NOTE 5 – A network interface includes an interface for multiple containers in multiple nodes and multiple clusters.

3) It is required that a container engine provide a container file system for an application.

NOTE 6 – A container file system runs on high-performance storage devices for fast input/output (I/O).

NOTE 7 – High-performance storage devices include a main memory, random access memory (RAM), non-volatile memory express (NVMe) and their combined storage.

NOTE 8 – The combined storage includes a RAM disc combined with a block storage device, NVMe or SSD connected to a network interface, the storage federated with other storages, including cloud storage.

4) It is recommended that a container engine provide high availability of a container file system.

NOTE 9 – For high availability, container images in the image layer are shared with other hosts through shared storage.

NOTE 10 – Shared storage backs up and synchronizes container images in all image layers for high availability.

5) It is recommended that a container engine provide location information of a container image in a container file system for application to use.

NOTE 11 – Location information exists in the merged layer and provides mount information of directories in other layers.

6) It is required that a container engine allocates storage volume.

NOTE 12 – Storage volume is provided in a binding host file system, mounting host storage devices, host directory and remotely connected storages.

NOTE 13 – A container engine allocates storage volumes in a timely fashion or persistently to applications.

7) It is required that a container engine provide a monitor of usage of storage volume.

8) It is required that a container engine provide a standard I/O interface for a storage volume to receive read/write commands.

9) It is required that a container engine provide an isolated kernel between applications.

NOTE 14 – An application is isolated by the namespace of the user, file system, process identifiers (IDs), memory and CPU limitation, which is set by the container engine. Allocated namespaces provide an independent space for each application to isolate resources in a kernel.

10) It is required that a container engine builds a container image to push to the container registry.

NOTE 15 – A container engine rebuilds the running application based on the container image and pulls it to the container registry for reuse.

11) It is required that a container engine pull container images to execute an application from the container registry.

12) It is required that a container engine provide a packaged container image for easy transport.

13) It is required that a container engine provide an image object in the container image.

NOTE 16 – An image object includes a library, file system and binary in the container file system.

14) It is required that a container engine provide an image manifest in the container image.

15) It is required that a container engine provide an application execution procedure from the container file.

16) It is recommended that a container engine reuse the container file to create a new container image.

NOTE 17 – A container engine reuses the already created container file in a container image and recreates a new container file from a partial container file.

NOTE 18 – A partial container file is part of a container file including the container information to reuse, application name, OS and version.

NOTE 19 – A container engine verifies the version of a partial container file and container images, creates a container image and pushes the recreated container file to the registry to reuse other applications.

17) It is required that a container engine search the location of a container image in a container registry.

NOTE 20 – The image ID allocated to a container image is a hashed number for its searching.

18) It is required that a container engine eliminate duplicated container images in a container file system to save storage capacity.

19) It is recommended that a container engine provide a local registry for a container image.

NOTE 21 – A local registry includes the image layer of a container file system.

### 8.1.2 Functional requirement of a container management system

1) It is required that a CMS provide management of a single container.

NOTE 1 – Management of a single container includes the management of container lifecycle, network, execution application, the configuration of the container engine, versioning, security, logs and hosts.

2) It is required that a CMS authenticate access to a container engine for a single container.

3) It is required that a CMS provide remote access to a container engine for a single container.

4) It is required that a CMS provide management for multiple containers.

NOTE 2 – The management of multiple containers includes managing containers on multiple host OS, extending containers to the distributed system and clustering containers.

5) It is recommended that a CMS provide a shared resource for multiple containers.

NOTE 3 – The shared resources include network and storage.

6) It is recommended that a CMS monitor the status of an application for availability.

NOTE 4 – The status of an application includes whether it is operating.

7) It is required that a CMS monitor resource utilization of applications so that the allocated resources limit is not exceeded.

NOTE 5 – Application resources include CPU, memory and storage.

8) It is required that a CMS balance load to ensure availability of containers.

NOTE 6 – Targets for load balancing include the CPU, memory, storage, network and accelerator, such as a graphics processing unit or tensor processing unit.

9) It is recommended that a CMS discover applications to find their location when running.

10) It is recommended that a CMS scale a container according to a scaling policy.

NOTE 7 – A scaling policy includes scaling up or down of the number of containers based on user policy, such as load distribution of resources.

11) It is recommended that a CMS reallocate resources for a container according to user requests.

12) It is recommended that a CMS optimize resource utilization based on workload.

13) It is recommended that a CMS cluster a container across multiple hosts.

14) It is recommended that a CMS synchronize container images between container registries.

### 8.2 Functional requirements of cloud computing to support a container

1) It is required that a CSP provide container engines according to the kernel of the host.

NOTE 1 – Container engines depend on the kernel of the host OS.

2) It is recommended that a CSP provide a CMS to a CSN to develop cloud service.

3) It is recommended that a CSP provide cloud service that is implemented by a container.

4) It is recommended that a CSP provide containers on a guest OS of a VM, as well as a bare-metal machine.

NOTE 2 – A container on a guest OS of a VM is used without changes to the existing cloud infrastructure.

5) It is recommended that a CSP evaluate performance of an application that runs on a VM and container for a CSN.

6) It is recommended that a CSP manage a network to use multiple CMS.

7) It is recommended that a CSP provide compatibility between applications on containers and VMs.

8) It is recommended that a CSP provide information about the host on which the container is running.

NOTE 3 – Host information includes the OS, bare-metal machine and VM.

9) It is required that a CSP provide a multiple container registry to upload container images.

10) It is recommended that a CSP provide information about a container registry to a CMS and CSN.

NOTE 4 – Information about a container registry includes its location, access mechanism and permission rules.

11) It is recommended that a CSP provide secure access for a container.

NOTE 5 – Secure access includes gateway information and firewall configuration.

## 9 Security considerations

Security aspects for consideration within the cloud computing environment are addressed by security challenges for CSPs, as described in [b-ITU-T X.1601]. In particular, [b-ITU-T X.1601] analyses security threats and challenges, and describes security capabilities that could mitigate these threats and meet the security challenges.

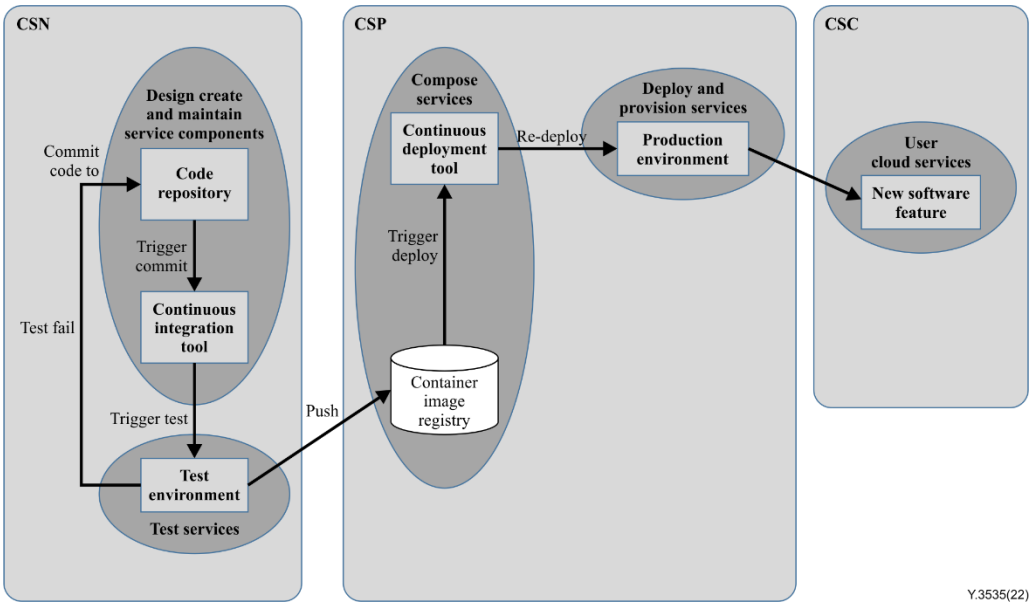
[b-ITU-T X.1631] provides guidelines to support the implementation of information security controls for CSCs and CSPs. Many guidelines lead CSPs to assist CSCs in implementing controls, and lead CSCs to implement such controls. Selection of appropriate information security controls, and the application of the implementation guidance provided, depends on risk assessment, as well as any legal, contractual, regulatory or other cloud-sector specific information security requirements.

# Appendix I

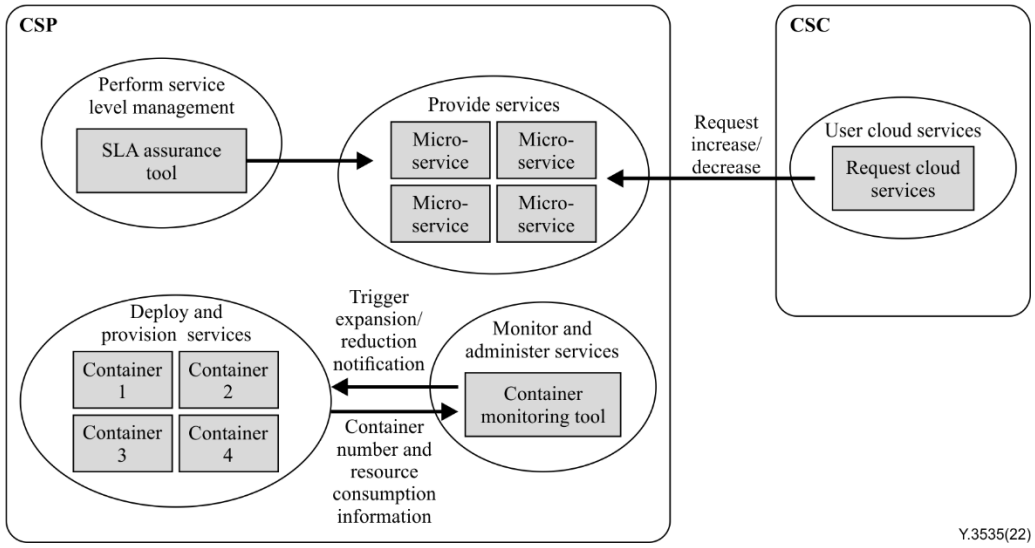
## Use cases of containers

(This appendix does not form an integral part of this Recommendation.)

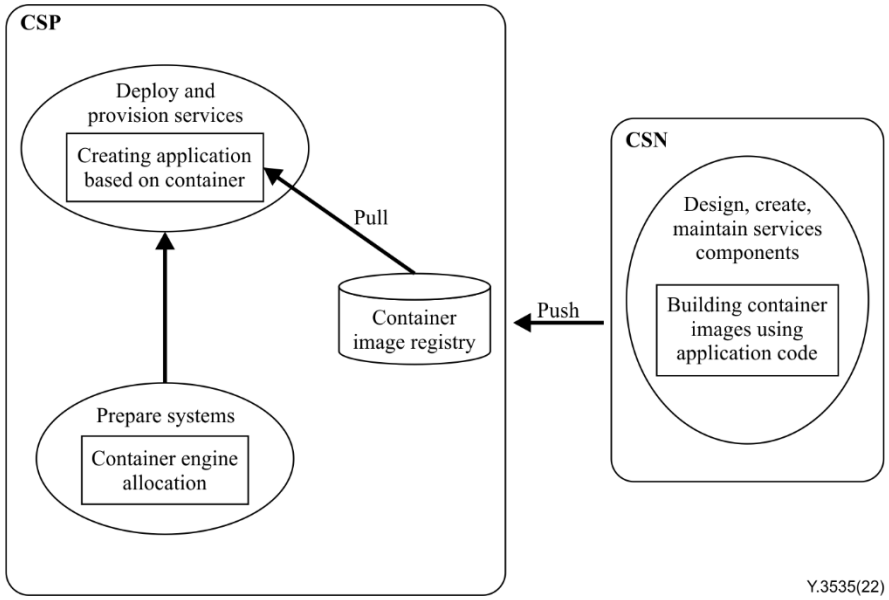
### I.1 Fast continuous integration and continuous deployment

| Title | Fast continuous integration and continuous deployment |
|---|---|
| Identifier | Fast-CI&CD |
| Description | This use case is common in the software development scenario. By using containers, the whole continuous integration and continuous deployment (CI/CD) process is accelerated and more convenient. |
| Roles | CSN, CSP, CSC |
| Figure (optional) |  <br>There are several steps in this use case:<br>(1) new software features are added by the CSN; (2) the updated codes are committed into the code registry; (3) which triggers the CI tool to download from the code registry and perform a unit and integration test in a test environment; (4) once the new feature passes the test, the new container images are built and pushed into the container registry; (5) which triggers the CD tool to pull the container image from the container registry and redeploy into the production environment; (6)either the test or build failure is notified to CSN; (7) the CSC can access the new feature of the software. |
| Pre-conditions (optional) | New software features are developed in source code by the CSN |
| Post-conditions (optional) | New software features are deployed into production environment for CSC use |
| Derived requirements | – Clause 8.2 requirement 6) |

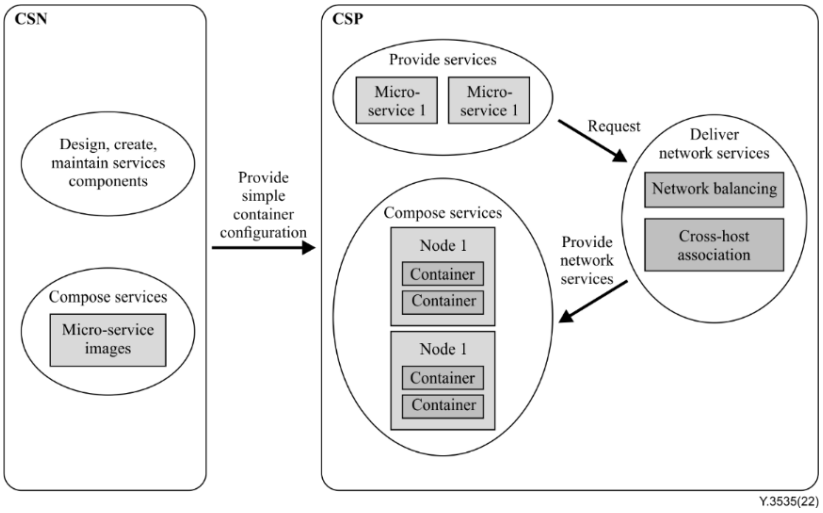
## I.2 Scaling of container clusters according to application workload

| Title | Scaling of container clusters according to application workload |
|---|---|
| Identifier | Scaling of container |
| Description | Instead of remaining static, the workload of most applications fluctuates due to many factors, such as work/off-work hours, holidays, online marketing activities, and even momentary increases or decreases of user requests.<br>In this use case, the CSP supplies microservices (e.g., a web portal) for a CSC to request. The CSP configures and provisions a container cluster for the microservices through a command line interface (CLI) or graphical user interface (GUI) in physical or virtual machines. The CSP has the capabilities to monitor resource consumption of the container cluster (e.g., the container number in the container cluster and CPU/memory/disc usage of each container). Whenever the resource consumption exceeds a predefined threshold for expansion (e.g., the average CPU usage of the container cluster exceeds 80% for 10 min successively), the CSP notifies that it will expand the scale of the container cluster following a specified elastic scaling policy (e.g., add a group of two containers to the cluster). Consequently, the service provided by the CSP is able to handle more requests from a CSC, and the service level agreement (SLA) can be fulfilled (e.g., the CSP performs SLA assurance activity). In contrast, when the resource consumption of a container cluster falls below a predefined threshold for reduction (e.g., the average CPU usage of the container cluster falls below 30% for 10 min successively), the CSP notifies that it will reduce the scale of the container cluster to save energy. |
| Roles | CSP, CSC |
| Figure (optional) |  |
| Pre-conditions (optional) | |
| Post-conditions (optional) | |
| Derived requirements | – Clause 8.1.1 requirement 1)<br>– Clause 8.1.2 requirement 6)<br>– Clause 8.1.2 requirement 9)<br>– Clause 8.1.2 requirement 10)<br>– Clause 8.1.2 requirement 11)<br>– Clause 8.2 requirement 3)<br>– Clause 8.2 requirement 4) |

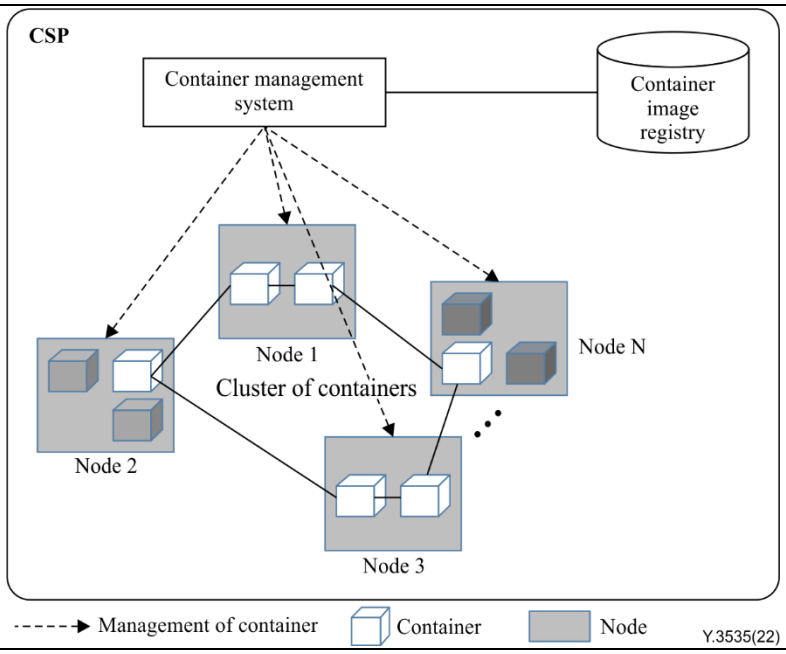
## I.3 Container allocation for launching microservice

| Title | Container allocation for launching microservice |
|---|---|
| Identifier | Container allocation |
| Description | This use case describes how to deploy applications based on a container in a cloud environment. In this scenario, the CSN is responsible for developing a program that describes what it needs to create an application. The CSP creates a container image using the program. When the CSP builds the container image into a container, an application can be operated. In this case, the CSP is responsible for preparing container engines and deploying applications based on the container. Detailed procedures are as follows:<br>– the CSN develops programs for the application;<br>– the CSN provides the CSP with configuration information to create microservices;<br>– the CSP prepares container engines and allocates containers;<br>– the CSP deploys applications based on the container. |
| Roles | CSP, CSN |
| Figure (optional) |  |
| Pre-conditions (optional) | The container engine has various OSs |
| Post-conditions (optional) | |
| Derived requirements | – Clause 8.1.1 requirement 17)<br>– Clause 8.1.1 requirement 19)<br>– Clause 8.1.2 requirement 8)<br>– Clause 8.2 requirement 1)<br>– Clause 8.2 requirement 5)<br>– Clause 8.2 requirement 8) |

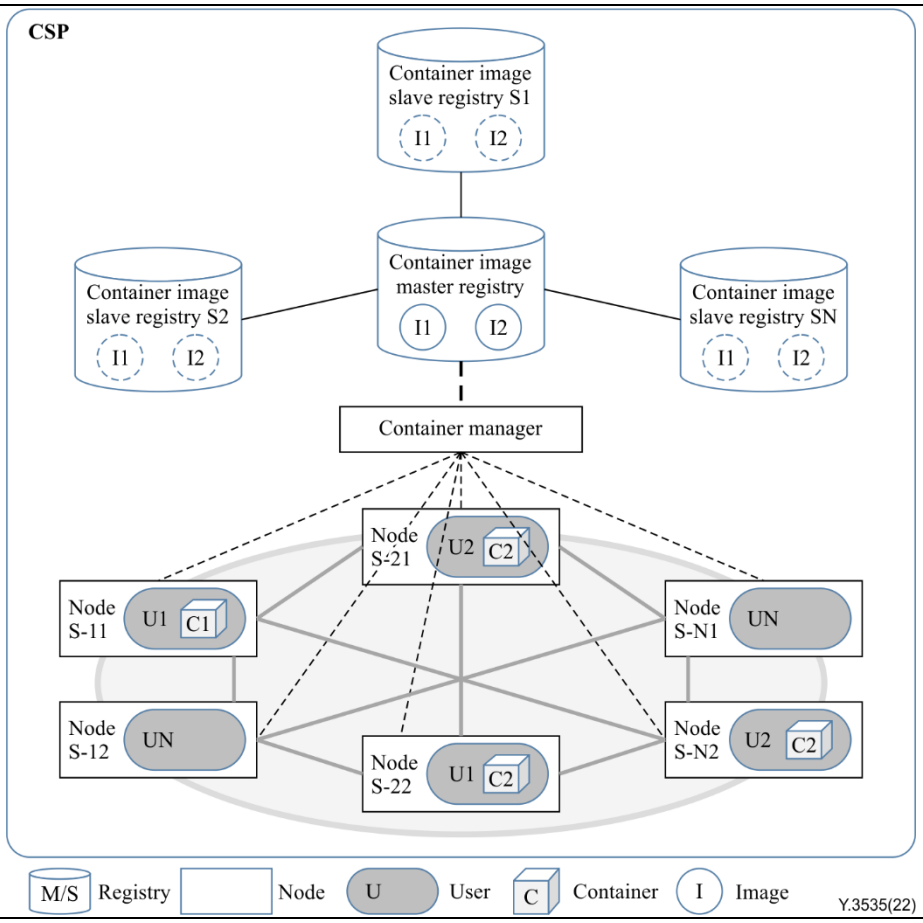
## I.4 Load-balancing containers in cloud application deployment

| Title | Load-balancing containers in cloud application deployment |
|---|---|
| Description | An application such as a microservice is difficult to operate and manage after being deployed in a cloud environment because of its numerous independent processes. Containers provide a way to run applications in a secure, isolated environment, isolating applications from the infrastructure layer, and also managing infrastructure as a program, significantly improving operational management efficiency. The application container provides a running environment that addresses the challenges of deploying microservice programs in a cloud environment. Since the containers are deployed on different nodes in a cloud environment, appropriate network services should also be used to accomplish communication between containers.<br>Based on that background, this use case describes how containers communicate and coordinate with each other to accomplish the efficient deployment of microservice cloud software.<br>In this scenario, a CSN is responsible for creating a microservice and providing a simple container configuration. The CSP is also responsible for the scheduling of computing resources flexibility, packaging code, test and deploying software efficiently. In addition, the CSP supports network load balancing and cross-node correlation to meet communication and coordination requirements between components under the microservice architecture. Whenever the microservice changes, a new container image is built for later deployment. |
| Roles | CSN, CSP |
| Figure (optional) |  <br>– The CSN creates a microservice and provides a simple container configuration for the CSP.<br>– The CSP schedules the computing resources, packages code, tests and deploys software and provides log monitoring and management services. The CSP also supports network load balancing and cross-node correlation to meet communication and coordination requirements between components under the microservice architecture. |
| Pre-conditions (optional) | The containers can be deployed on different nodes in a cloud environment. |
| Post-conditions (optional) | |
| Derived requirement | – Clause 8.1.1 requirement 2)<br>– Clause 8.1.2 requirement 8)<br>– Clause 8.1.2 requirement 12) |

## I.5 Container clustering across multiple node

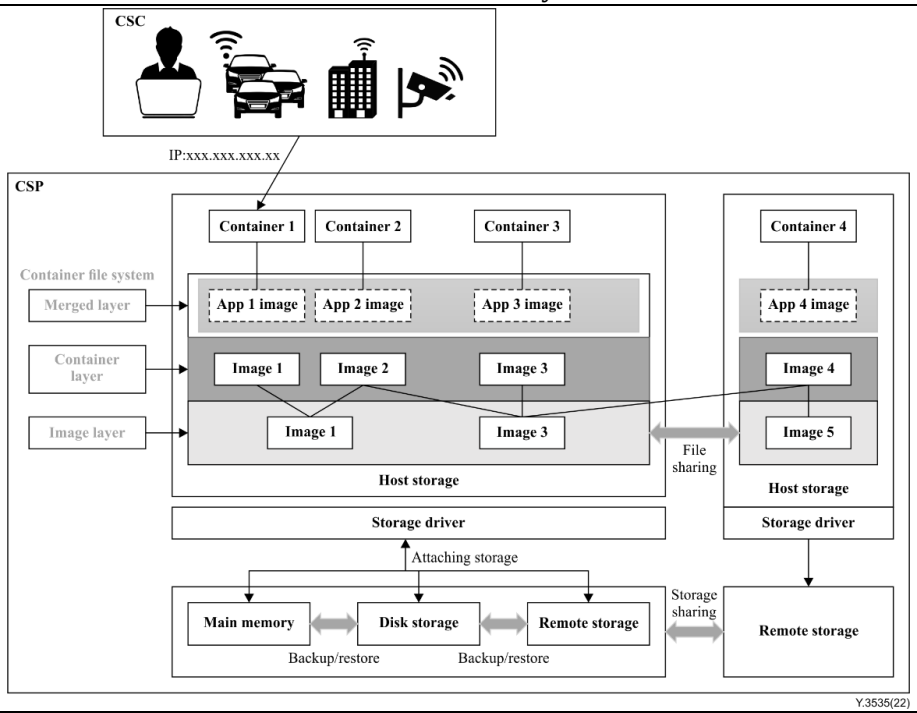
| Title | Container clustering across multiple node |
|---|---|
| Identifier | Container clustering |
| Description | This use case describes how to set up and manage the clustering of the node for containers in a cloud environment. The node clustering with multiple containers is to distribute the containers among multiple nodes and cluster them. Clustering of containers can easily scale out and provide smooth service when it is in a large-scale service request.<br>In this scenario, the container management function in CSP is responsible for determining the node on which to install the application and installs the application using the container image stored in the container image registry.<br>In this scenario, the container management function is responsible for monitoring the service delivery status (response time, etc.) and resource usage of each node installed in the service. If there is a problem with the service delivery on a particular node, the service is replaced by the one on which the service in the clustering is installed. The container management function also looks at the status of the service, and if it recognizes that the service is concentrated in a specific node and is overloaded, the service manager distributes the service to a more relaxed node. In other words, the container management function performs a load-balancing operation. In addition, the container management function performs the task of adding a new node to the cluster when it is necessary to expand the cluster to expand the service. |
| Roles | CSP |
| Figure (optional) |  |
| Pre-conditions (optional) | The CSP provides node and for container. |
| Post-conditions (optional) | |
| Derived requirements | – Clause 8.1.1 requirement 2)<br>– Clause 8.1.2 requirement 1)<br>– Clause 8.1.2 requirement 2)<br>– Clause 8.1.2 requirement 3)<br>– Clause 8.1.2 requirement 4)<br>– Clause 8.1.2 requirement 5)<br>– Clause 8.1.2 requirement 6)<br>– Clause 8.1.2 requirement 13)<br>– Clause 8.2 requirement 7) |

## I.6 Container image distribution

| Title | Container image distribution |
|---|---|
| Description | In this scenario, the container image registry has to distribute container images to hundreds of cluster nodes. A single registry instance can no longer support a huge number of requests. So multiple registry instances (e.g., S1, S2, …, SN) are configured for load balance. In addition, image I1 is only for user U1, and the other images are open to all users. In order to achieve this:<br>first, images (e.g., I1, I2) should be replicated (synchronized) from master-registry instance M1 to slave-registry instance (e.g., S1, S2, …, SN);<br>second, user U1 pulls image I1 from slave-registry instance S1 based on its authority. |
| Roles | CSP |
| Figure (optional) |  |
| Pre-conditions (optional) | Container images are stored as master-registry instances.<br>Registry instances are configured with synchronization policies. |
| Post-conditions (optional) | |
| Derived requirements | – Clause 8.1.2 requirement 14)<br>– Clause 8.2 requirement 9)<br>– Clause 8.2 requirement 10) |

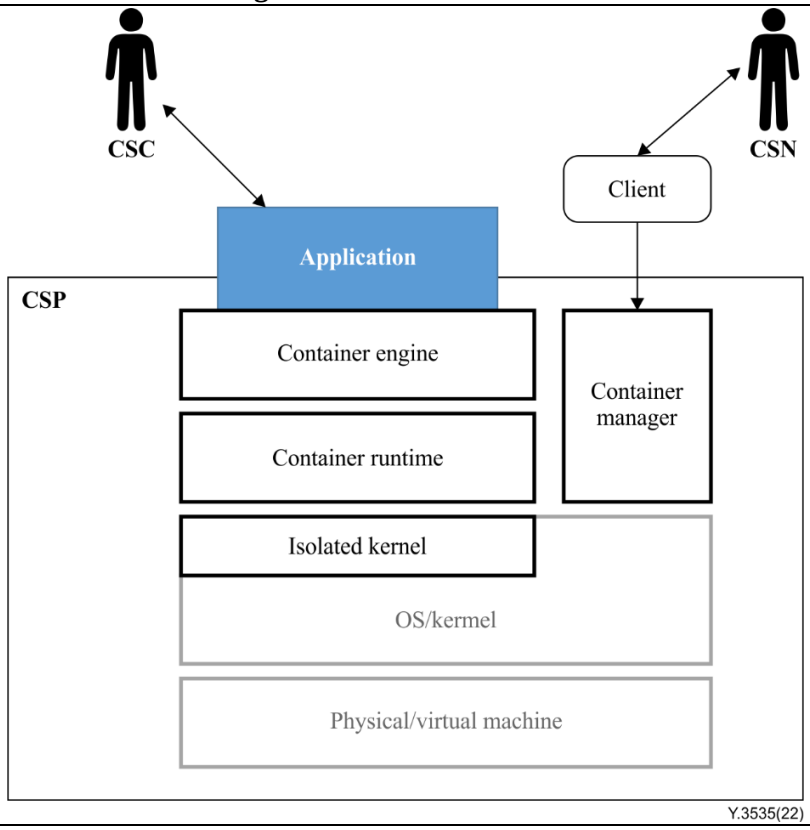
## I.7 The container file system

| Title | The container file system |
|---|---|
| Description | This use case is about a container with a unifying file system and sharing images.<br>A container file system makes a specific directory or files appear as the root file system which is independently used by one container. A container file system also needs to manage the images efficiently.<br>A container file system runs on a host storage device (main memory, SSD, hard disc drive, etc.). In the case of containers, the files required by the user are provided individually by using the unifying file system included in the existing kernel.<br>A unifying file system mounts multiple file systems on a single mount point, and instead of creating a new file system type, all directory entries are unified in the virtual file system layer. With file system consolidation, directory entries from the child file system are merged with those from the parent file system to create a logical combination of all mounted file systems. Therefore, it is possible to manage and find files for the entire file system shared on the system locally and file management for the entire share becomes easy.<br>As previously described, the container file system is composed of an integrated file system and is composed of layers. It consists of a merged access area, a container layer and an image layer. Each layer operates by creating and mounting a specific directory on the host storage.<br>The container layer is a writable layer and is created on the top layer for each container, allowing each container to have its own state. After a container is created, all changes are made in this layer.<br>The image layer is a read-only layer that can be shared with other containers. In addition, multiple images shared with other layers can be operated in the container layer.<br>The merged layer also includes link information about the layer accessible to all file systems of the container layer and the image layer and is shared with other containers. This allows access to the file.<br>The image layer can be shared with many different systems to increase its efficiency. As shown in the figure, the container image of the image layer should be pulled from a public registry (e.g., Github) when the container is deployed. In this case, to ensure performance, it is efficient to store the image used in the container system locally or to bring it in advance. In this system, the images that have already been pooled in shared storage can be reused with shared storage. As previously mentioned, many images in the image layer exist in container storage, and the container images of the entire system are backed up and stored on disc or remotely. Adding images to the image layer by sharing storage could also make them available to the container layer, and images are continuously provided in the merged layer. |
| Roles | CSC, CSP |

| Title | The container file system |
|---|---|
| Figure (optional) |  |
| Pre-conditions (optional) | |
| Post-conditions (optional) | |
| Derived requirements | – Clause 8.1.1 requirement 2)<br>– Clause 8.1.1 requirement 3)<br>– Clause 8.1.1 requirement 4)<br>– Clause 8.1.1 requirement 5)<br>– Clause 8.1.1 requirement 6)<br>– Clause 8.1.1 requirement 7)<br>– Clause 8.1.1 requirement 8)<br>– Clause 8.1.1 requirement 18) |

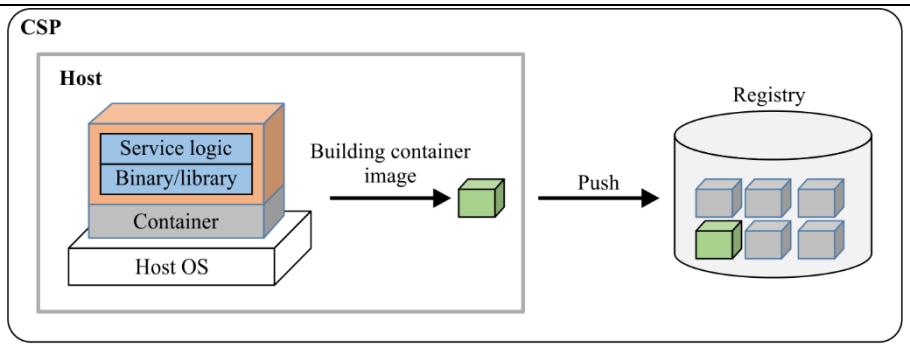
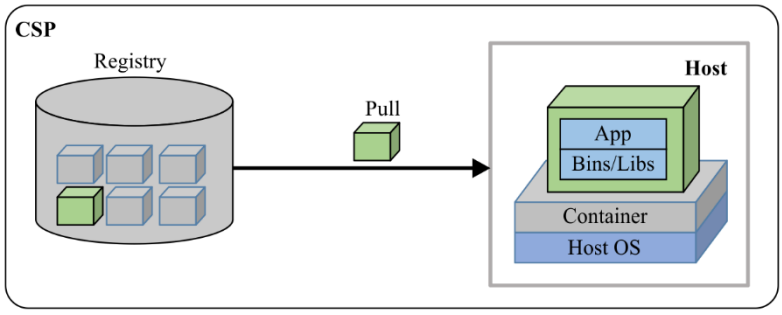
## I.8 The general use case of container

| Title | The general use case of container |
|---|---|
| Description | The container generally consists of four components: container runtime; container engine; container manager; and isolated kernel.<br>The container manager manages the deployment of the containerized application, containerized application life-cycle management (e.g., create, start, stop and delete applications). A container manager also supports user interface (such as CLI and API) management for the container engine.<br>The container engine is a program that runs on the host OS. The container engine creates and executes the container with a API.<br>Container runtime provides a daemon or software exposing API based on the container engine. Container runtime provides low level capabilities such as execution of the application and interfaces for capabilities of the container engine.<br>The isolated kernel provides a secure kernel to operate containerized applications in an independent kernel space. The isolated kernel logically separates the secured namespace and resource allocation in the host OS/kernel. |
| Roles | CSC, CSN, CSP |

| Title | The general use case of container |
|---|---|
| Figure (optional) |  |
| Pre-conditions (optional) | |
| Post-conditions (optional) | |
| Derived requirements | – Clause 8.1.1 requirement 2)<br>– Clause 8.1.1 requirement 3)<br>– Clause 8.1.1 requirement 6)<br>– Clause 8.1.1 requirement 9)<br>– Clause 8.1.1 requirement 11)<br>– Clause 8.1.1 requirement 12)<br>– Clause 8.1.2 requirement 6)<br>– Clause 8.2 requirement 5)<br>– Clause 8.2 requirement 11) |

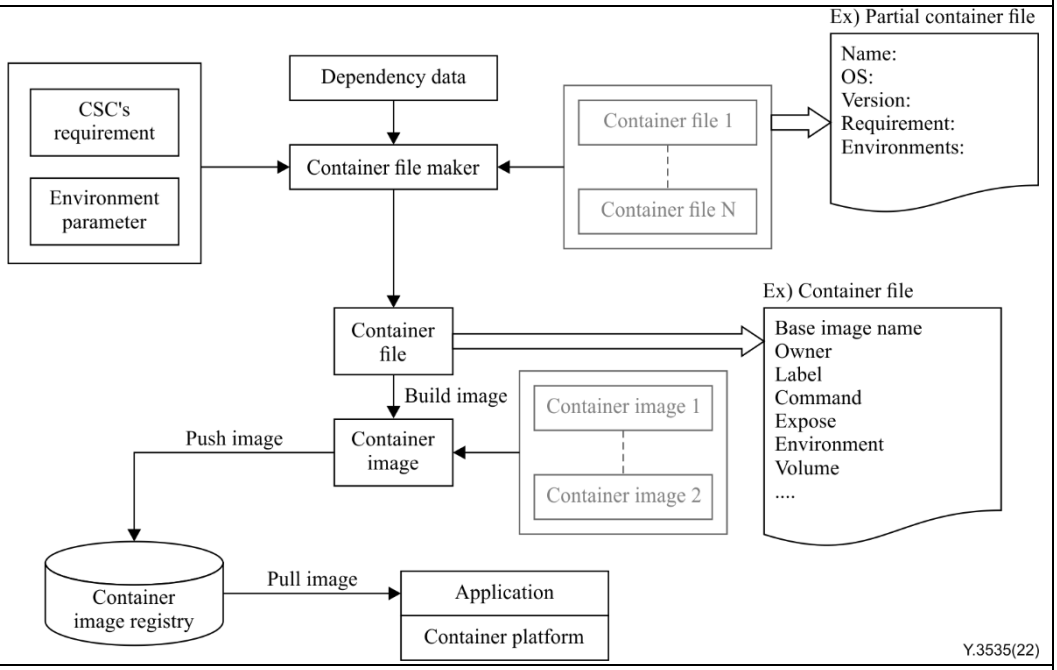
## I.9 Building and registering container images

| Title | Building and registering container images |
|---|---|
| Description | This use case shows how to build a container image and enter it into the registry. Figure 1) shows the building and pushing container images. A container engine in each host builds a container image file as it is in the current state of the installed and operated application, and these container images can be registered using push commends.<br>This use case also shows how to use the container image. In Figure 2), a host which wants to use container images registered in the registry uses the pulling method to retrieve container images. After finishing downloading the container image, the host creates a container using that image. |
| Roles | CSP |

| Title | Building and registering container images |
|---|---|
| Figure (optional) |  <br>1) Building container image and registering container image<br>  <br>2) Pulling container image and creating container<br>Y.3535(22) |
| Pre-conditions (optional) | |
| Post-conditions (optional) | |
| Derived requirements | – Clause 8.1.1 requirement 10)<br>– Clause 8.1.1 requirement 11)<br>– Clause 8.1.1 requirement 13)<br>– Clause 8.1.1 requirement 14)<br>– Clause 8.1.1 requirement 15)<br>– Clause 8.1.2 requirement 14) |

## I.10 Container image creation

| Title | The use case for container image creation |
|---|---|
| Description | In a container platform, it is essential to create and register images corresponding to various execution environments. It can be deployed and managed in various environments through the push and pull commands registered using the container registry instead of copying files directly to distribute the image built in this way to the server.<br>Typically, to create a container, you create the final image through a series of commands. An interface is required for the user to describe a series of commands and to reflect them on the system.<br>To create a container, a user's container image is created with a container file in which a series of commands is described.<br>As shown in the figure, containers are used in combination with various images and the container file is manually written by the developer.<br>When applied to a system using a container file, each command in the file is executed in order on the system, each image downloads the required image from the local registry or remote registry to a fixed location in the container file system and writes it to the script. Images are created according to the content. |

| Title | The use case for container image creation |
|---|---|
| | The container image is used by sharing the image of the part using the layered file system of the actual container file system. It is structured to reuse images common to each layer, and this follows the characteristics of the integrated file system.<br>As such, in the case of a user-made container file, the desired container file can be created by combining necessary parts among the container files previously created through the partial container file.<br>In the partial container file in the figure, CSC requirements are to run the application service, the environment parameter corresponding to the execution environment variable and a partial container that plays the same role as the base image used to create the existing container file.<br>Additional dependency data has dependency information between requirements during the installation.<br>The container file maker refers to the dependency data and finally creates the container file through the combination of requirement, environment parameter, and partial container file. |
| Roles | CSC, CSN, CSP |
| Figure (optional) |  |
| Pre-conditions (optional) | |
| Post-conditions (optional) | |
| Derived requirements | – Clause 8.1.1 requirement 1)<br>– Clause 8.1.1 requirement 11)<br>– Clause 8.1.1 requirement 16)<br>– Clause 8.1.1 requirement 18) |

# Appendix II

## An example for illustration of a comparison between a container and a virtual machine

(This appendix does not form an integral part of this Recommendation.)

Table II.1 is a comparison between containers and VMs. The values of quantifiable entries in Table II.1 can be changed rapidly.

Containers have similar resource isolation and allocation benefits as VMs but a different architectural approach allows them to be much more portable and efficient. Each VM includes the application, the necessary binaries and libraries, and an entire guest OS – all of which may be 10s of gigabytes in size.

Containers include the application and all of its dependencies, but share the kernel with other containers. They run as an isolated process in user space on the host OS. They are also not tied to any specific infrastructure – containers can run on most types of servers and OSs, and most cloud infrastructures.

**Table II.1 – Comparison between virtual machines and containers**

| | **Virtual machines** | **Containers** |
|---|---|---|
| Software stack | Application + Binary/library + OS | Application + Binary/library |
| Image size | 10s of gigabytes | 10s to 100s of megabytes |
| Instances per host | 10s | 100s to 1 000s |
| Deployment time | Several minutes | Several seconds |

# Bibliography


[b-ITU-T H.764] Recommendation ITU-T H.764 (2019), *IPTV services enhanced script language.*

[b-ITU-T X.1601] Recommendation ITU-T X.1601 (2015), *Security framework for cloud computing*.

[b-ITU-T X.1631] Recommendation ITU-T X.1631 (2015), *Information technology – Security techniques – Code of practice for information security controls based on ISO/IEC 27002 for cloud services*.

[b-ITU-T Y.3500] Recommendation ITU-T Y.3500 (2014), *Information technology – Cloud computing – Overview and Vocabulary.*

[b-ITU-T Y.3504] Recommendation ITU-T Y.3504 (2016), *Functional architecture for desktop as a service.*

[b-ITU-T Y.3510] Recommendation ITU-T Y.3510 (2016), *Cloud computing infrastructure requirements.*

## SERIES OF ITU-T RECOMMENDATIONS

| | |
|---|---|
| Series A | Organization of the work of ITU-T |
| Series D | Tariff and accounting principles and international telecommunication/ICT economic and policy issues |
| Series E | Overall network operation, telephone service, service operation and human factors |
| Series F | Non-telephone telecommunication services |
| Series G | Transmission systems and media, digital systems and networks |
| Series H | Audiovisual and multimedia systems |
| Series I | Integrated services digital network |
| Series J | Cable networks and transmission of television, sound programme and other multimedia signals |
| Series K | Protection against interference |
| Series L | Environment and ICTs, climate change, e-waste, energy efficiency; construction, installation and protection of cables and other elements of outside plant |
| Series M | Telecommunication management, including TMN and network maintenance |
| Series N | Maintenance: international sound programme and television transmission circuits |
| Series O | Specifications of measuring equipment |
| Series P | Telephone transmission quality, telephone installations, local line networks |
| Series Q | Switching and signalling, and associated measurements and tests |
| Series R | Telegraph transmission |
| Series S | Telegraph services terminal equipment |
| Series T | Terminals for telematic services |
| Series U | Telegraph switching |
| Series V | Data communication over the telephone network |
| Series X | Data networks, open system communications and security |
| **Series Y** | **Global information infrastructure, Internet protocol aspects, next-generation networks, Internet of Things and smart cities** |
| Series Z | Languages and general software aspects for telecommunication systems |

Printed in Switzerland
Geneva, 2022