Draft new Recommendation ITU-T Y.3535 (formerly Y.cccm-reqts)

Cloud Computing – Functional requirements for container

Summary

This Recommendation provides the overview and functional requirements of container in cloud computing. It describes the technical aspects of container and provides the relationship between containers and cloud computing. It also provides functional requirements for container in term of container engine, container management system and cloud computing to support container.

Keywords
container, cloud computing, container engine, container image, container management system

Table of Contents

Draft new Recommendation ITU-T Y.3535 (formerly Y.cccm-reqts)

Cloud Computing – Functional requirements for container

1 Scope

This Recommendation provides the overview and functional requirements of container in cloud computing. The scope of this Recommendation includes:

- Overview of container including concept and technical aspects;
- Container in cloud computing;
- Functional requirements of container;
- Use cases of 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.3500] Recommendation ITU-T Y.3500 (2014), *Cloud computing - Overview and Vocabulary.*

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

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

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

3.1.3 cloud service customer [ITU-T Y.3500]: A person or organization that consumes delivered cloud services within a contract with a cloud service provider.

3.1.4 cloud service provider [ITU-T Y.3500]: An organization that provides and maintains delivered cloud services.

3.1.5 hypervisor [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 the application.

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

NOTE 2 – The 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 container image.

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

API	Application Programming Interface
CLI	Command Line Interface
CSC	Cloud Service Customer
CSN	Cloud Service partNer
CSP	Cloud Service Provider
GUI	Graphical User Interface
IaaS	Infrastructure as a Service
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 keywords "**is required to**" indicate a requirement which must be strictly followed and from which no deviation is permitted if conformance to this document is to be claimed.

The keywords "**is recommended**" indicate a requirement which is recommended 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

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 and it requires less resources than a virtual machine because the container uses the resource of the host as necessary to run the application. Container also provides isolation, resource control, and portability for virtualization processing of applications as follows:

- **Isolation**: The container sets a separated kernel space and individual processor to each application for isolation of an application used in the container from other applications on a host.
- **Resource control**: To control the resource, container sets namespaces that allocate resources.

  NOTE 1 – Namespace includes network namespace, processor namespace, mount namespace, user namespace, and etc. Namespace has the role of the isolation of system resources when a process is running. Namespace also provides the independent space for each container and prevent collisions with each other in the kernel system.
- **Portability**: Container image developed by software developer is built all or part of an application. To use of container image, it is pushed to a container registry to share with other hosts to support the portability. The uploaded container image in the container registry is pulled to execute.

Figure 6-1shows the usage of an application, container image, container registry for a container. A host OS includes the OS on the bare metal server or the guest OS on the virtual machine.

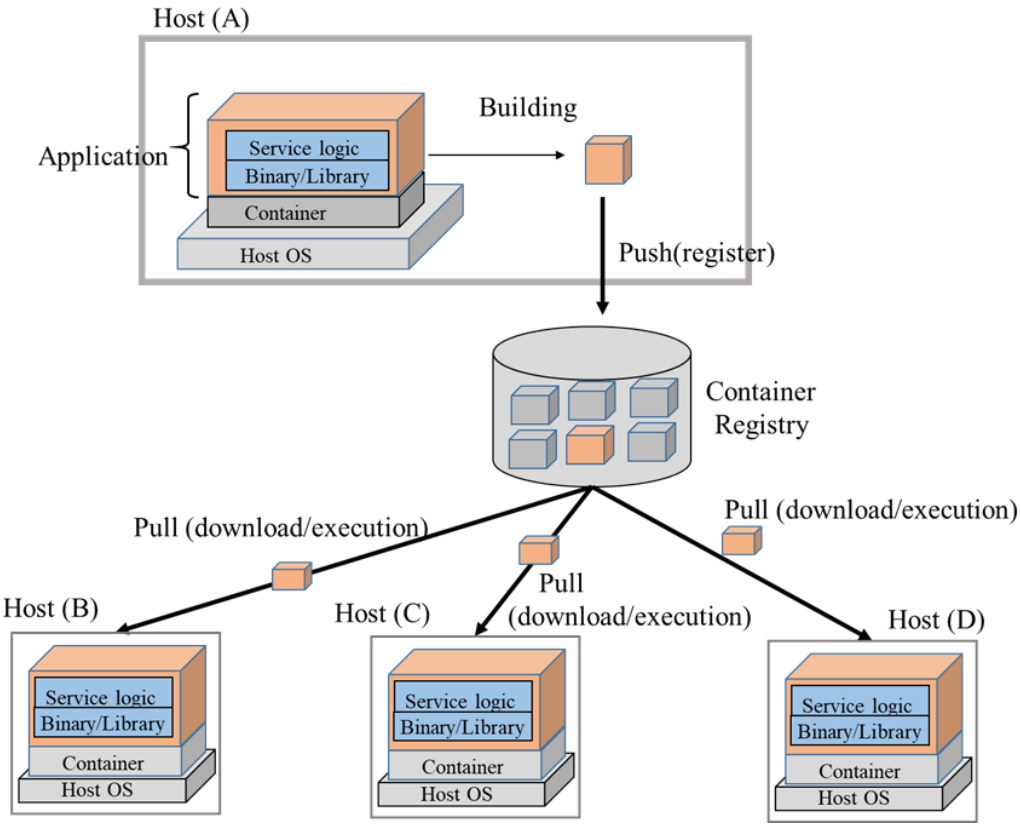


Figure 6-1 – Usage of application, container image, container registry for container

The container registry in Figure 6-1 is the repository for the container images in order for the container engine to store and access. Push refers to upload of the container image on hosts to the container registry, and pull refers to 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 forms of master-slave architecture. The master registry has authentication and authorization to access to slave registry. Master registry also supports configuring and updating image synchronization between two registries.

6.2 Technical aspects of container

6.2.1 Container engine

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

- The execution of an application with container image;
- The configuration of the isolated kernel space as well as 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, mounting or unmounting file system, allocating independent processes and networks.

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

- Pulling the container images from container registry;

- Unpacking container images into a file system;
- Setting file system ownership;
- Running applications using kernel process;
- Exposing the external links of the application for user access.

The configuration of the isolated kernel space includes as follows:

- Setting a directory of a container file system for container image;
- Setting the limitation of memory, CPU and device access;
- Creating the application's own namespace for resources;
- Configuring to prevent the system call which is not allowed;
- Setting the processor's secure access.

The isolated kernel space provides supporting the independent communication path allocation between processes and assigning 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 as following:

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

  NOTE 1 – 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 operating system;

  NOTE 2 – 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, its relationship to file system changes in a container image, and etc.

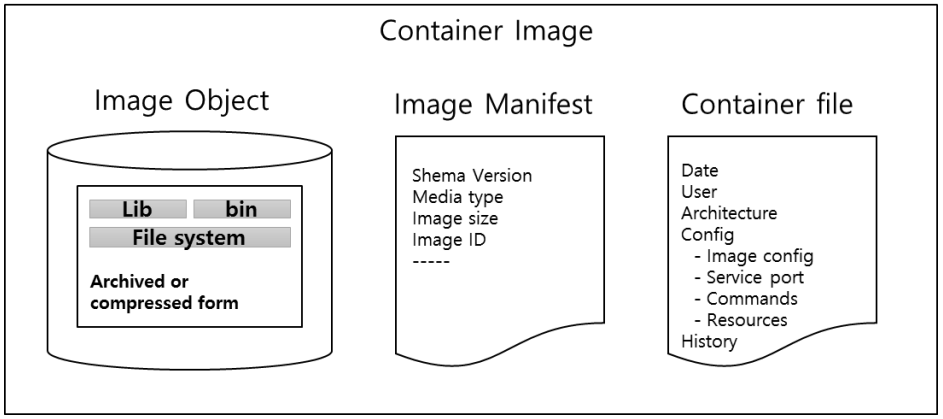


Figure 6-2 – The example of container image

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, SSD, HDD, etc.

For the container file system as shown in Figure 6-3, the 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 the container images from the merged layer, which is mounted to 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 the disk space. The application accesses the container image in the image layer with read-only access right.
- **Container layer**: provides each container's directory. The application accesses the container image in the container layer with read and write access right.
- **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.

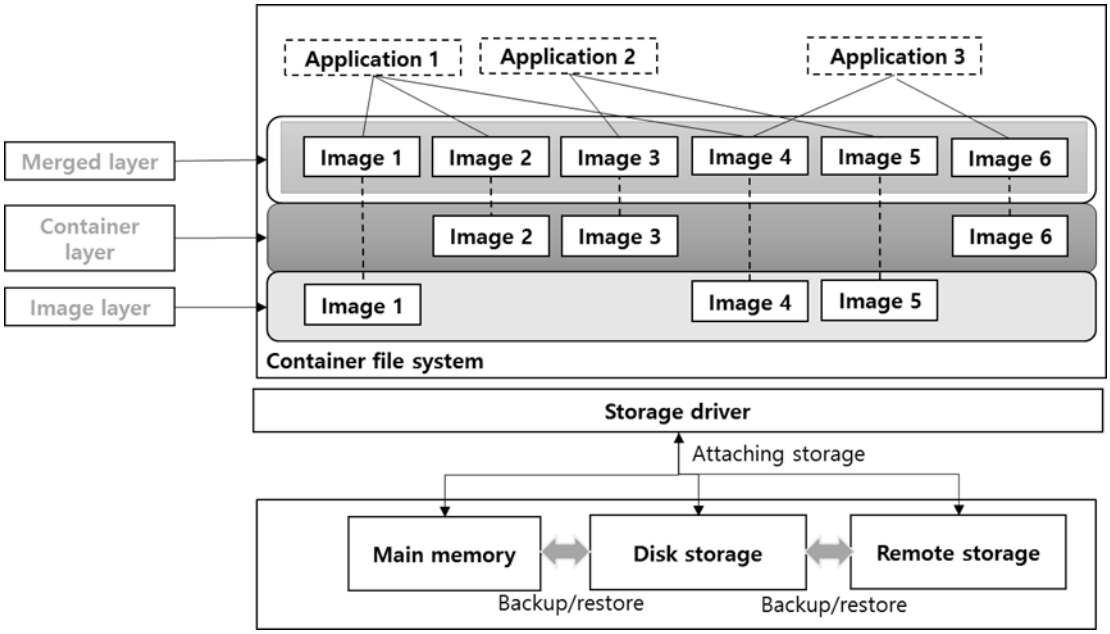


Figure 6-3 – A example of the file system for container

To unify the container image in the container file system, sub-directories in the container layer and image layer are mounted to be merged to the parent file system's directories to access logically the container image by applications. The container file system also provides searching container images for the entire file system shared on the system locally.

The container image in the image layer is stored from a container registry when the container is executed. For storing container images from the container registry, the container image is pulled by the container engine locally when the application uses the container. Also, the container image is stored in advance before the execution for better performance. Once the container image is pulled, it is reused by applications not to pull again in the image layer.

When the capacity of the file system is exceeded with container images in the image layer, the container images are backed up in other disk storage or remote storage.

6.2.4 Container management system

Container management system (CMS) provides container management for single container as well as multiple containers as shown in Figure 6-4:

- **CMS for the single container**: The CMS is in charge of managing operations of the container such as executing, versioning, configuring, and networking of the container through container 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 in case of usage of multiple containers at the same time.

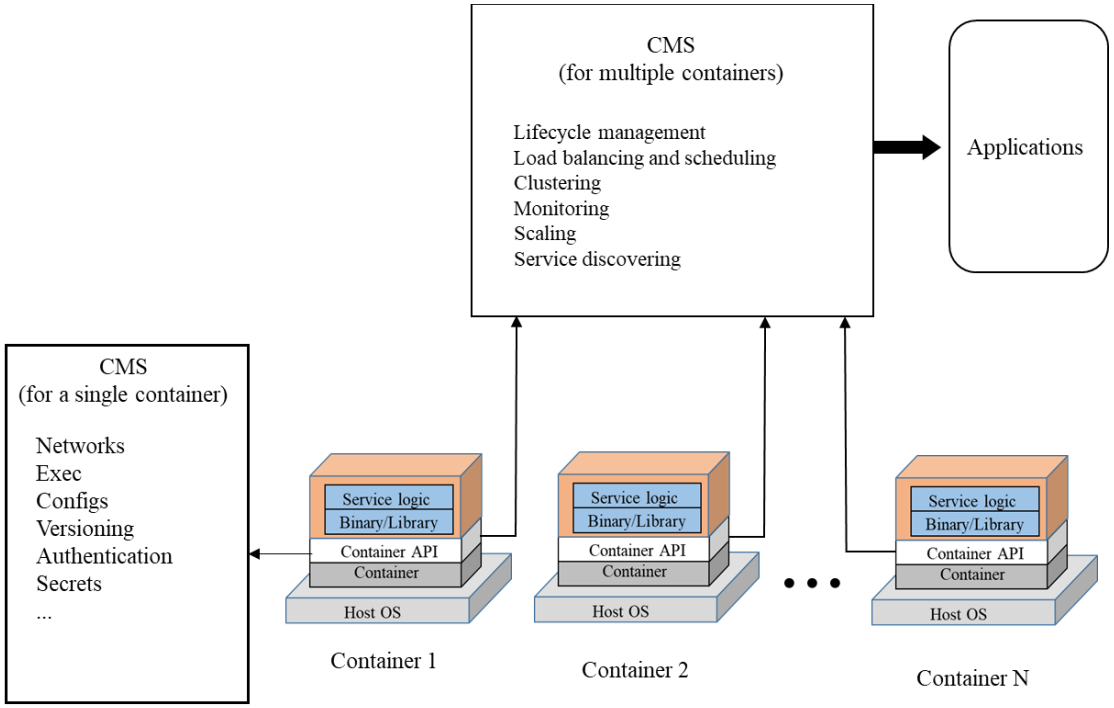


Figure 6-4 – Concept of the container management system

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 creating, pausing, resuming, restarting, setting configures, and etc.;
- **Load balancing and scheduling**: a CMS performs the task of distributing 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 restart 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 support scaling of container to the cluster according to service workload;
- **Application discovering**: a CMS performs the task of finding the application running in the container and the location of the container.

7 Container in cloud computing

In cloud computing, a container is a kind of OS-level virtualization using virtualized resources that are supported by cloud computing. And 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 containers provision in the cloud computing. The first is the container on a guest OS of a virtual machine as shown in Figure 7-1 (a), and the other is the container on the host operating system as shown in Figure 7-1 (b).

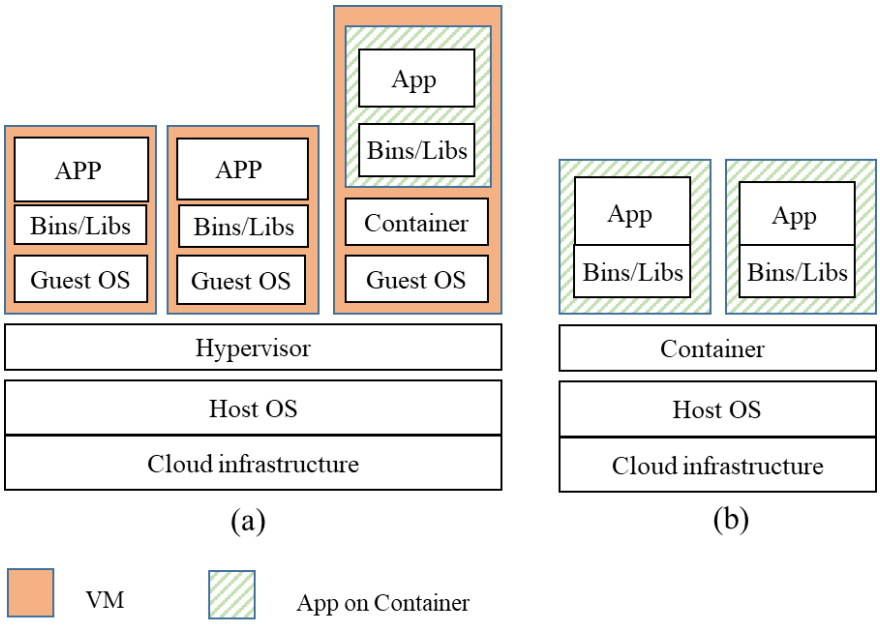


Figure 7-1 – Container running features in cloud computing

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

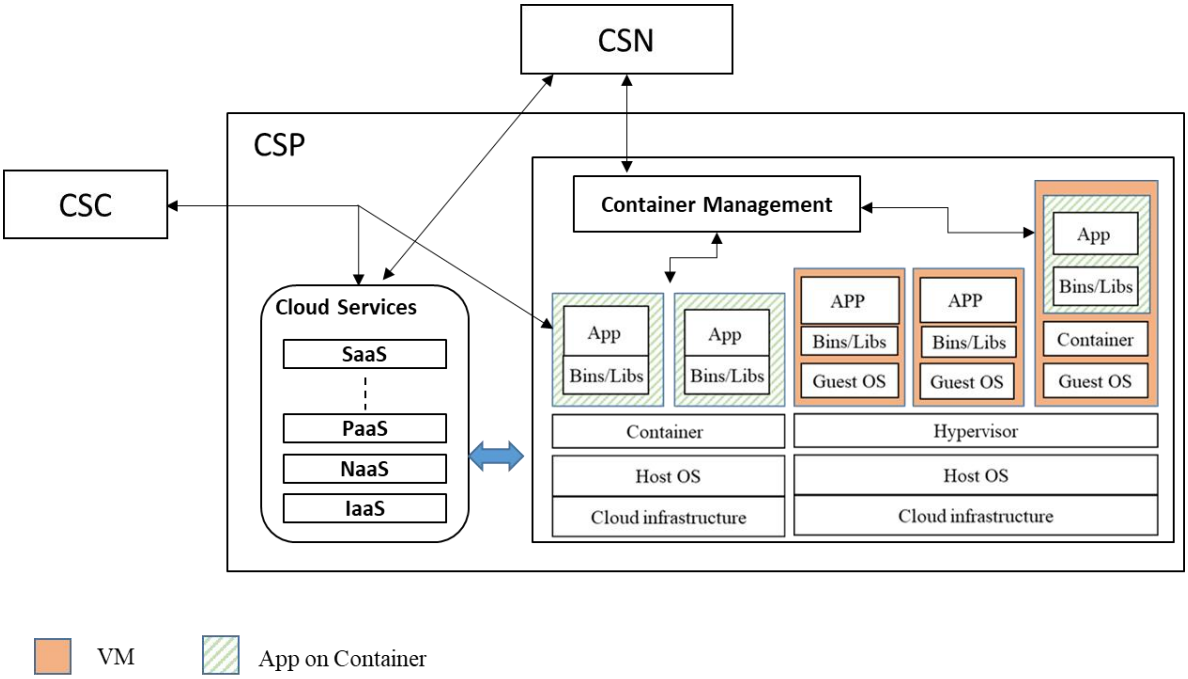


Figure 7-2 – The relationship between cloud computing and container

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

- **CSP** provides cloud services such as IaaS, PaaS and SaaS with the container. Also, container uses cloud services to support an application. An application in the container is designed in consideration with compatibility with the cloud service of the virtual machine;
- **CSN** develops the application of container using container management and cloud service, and CSP provides an interface to use cloud service for the convenience development of CSN;
- **CSC** uses a cloud service with the container as well as application on the container.

8 Functional requirements for supporting container in cloud computing

8.1 Functional requirements of container

8.1.1 Functional requirements of container engine

1) It is required that a container engine provides the execution of application in the kernel.

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

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

   NOTE 2 – Container engine sets the network namespace per application.

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

NOTE 4 – Network namespace allocates the port mapped with the host network or the new IP address to the application.

NOTE 5 – The network interface includes the interface for the multiple containers in multiple nodes and multiple clusters.

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

NOTE 6 – Container file system runs on high-performance storage devices for fast I/O.

NOTE 7 – The high-performance storage devices include a main memory, RAM, NVMe, their combined storage and etc.

NOTE 8 – The combined storage includes ram-disk combined to block storage device, NVMe or SSD connected with a network interface, the storage federated with other storages including cloud storages, and etc.

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

NOTE 9 – For high availability, container images in the image layer are shared with other hosts through the 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 provides the location information of the container image in container file system for application to use.

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

6) It is required that a container engine provides the allocation of storage volume.

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

NOTE 13 – Container engine allocates the storage volumes timely or persistently to applications.

7) It is required that a container engine provides the monitor of the usage for storage volume.

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

9) It is required that a container engine provides the isolated kernel between applications.

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

10) It is required that a container engine provides the building of a container image to push to the container registry.

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

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

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

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

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

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

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

16) It is recommended that a container engine provides the reuse of the container file for creating a new container image.

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

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

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

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

NOTE 20 – The image ID allocated per container image is a hashed number for the searching container image.

18) It is required that a container engine provides elimination of the duplicated container image in a container file system for saving storage capacity.

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

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

8.1.2 Functional requirement of container management system

1) It is required that CMS provides the management of the single container.

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

2) It is required that CMS provides the authentication to access to container engine for a single container.

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

4) It is required that CMS provides the management for multiple containers.

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

5) It is recommended that CMS provides the shared resource for multiple containers.

   NOTE 3 – The shared resources include network, storage, and etc.

6) It is recommended that CMS provides monitoring of the status of the application for availability.

   NOTE 4 – The status of application includes whether the application is operating or not.

7) It is required that CMS provides monitoring the resource utilization of applications not to exceed the limitation of the allocated resources.

   NOTE 5 – The resource of the application includes CPU, memory, storage etc.

8) It is required that CMS provides load balancing to ensure the availability of containers.

   NOTE 6 – The target for load balancing includes CPU, memory, storage, network and accelerator such as GPU and TPU.

9) It is recommended that CMS provides application discovery to find the location of the running application.

10) It is recommended that CMS provides scaling of the container according to the scaling policy.

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

11) It is recommended that CMS provides the reallocation of resources for the container according to the user's requests.

12) It is recommended that CMS provides the optimization of resources utilization based on workload.

13) It is recommended that CMS provides container clustering across multiple hosts.

14) It is recommended that CMS provides synchronization of container images between container registries.

8.2 Functional requirement of cloud computing to support container

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

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

2) It is recommended that CSP provides CMS to CSN to develop the cloud service.

3) It is recommended that CSP provides cloud service which is implemented by container.

4) It is recommended that CSP provides containers on a guest OS of a virtual machine as well bear-metal machine.

   NOTE 2 – The container on a guest OS of a virtual machine is used without changes to the existing cloud infrastructure.

5) It is recommended that CSP provides the performance evaluation of the application, which runs on a virtual machine and container for CSN.

6) It is recommended that CSP provides the network management to use multiple CMS.

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

8) It is recommended CSP provides the host information on which the container is running.

   NOTE 3 – The host information includes OS, bare-metal machine, virtual machine, and etc.

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

10) It is recommended CSP provide the information of container registry to CMS and CSN.

    NOTE 4 – The information of the container registry includes the location of the registry, access mechanism, permissions rules, and etc.

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

    NOTE 5 – Secure access includes the information of gateway, the configuration for firewall, and etc.

9 Security considerations

Security aspects for consideration within the cloud computing environment are addressed by security challenges for the 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.

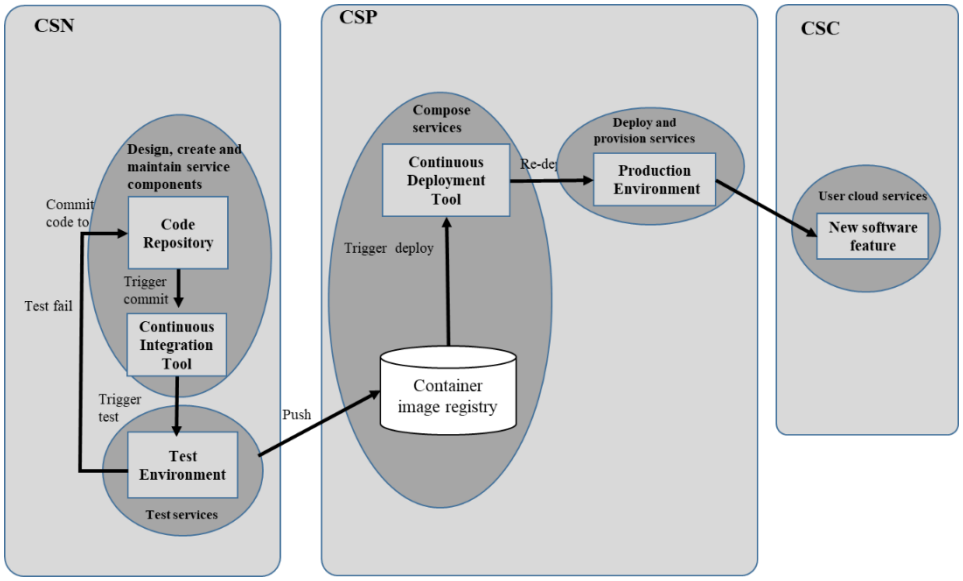
[b-ITU-T X.1631] provides guidelines supporting the implementation of information security controls for cloud service customers and cloud service providers. Many of the guidelines guide the cloud service providers to assist the cloud service customers in implementing the controls, and guide the cloud service customers to implement such controls. Selection of appropriate information security controls, and the application of the implementation guidance provided, will depend on a risk assessment as well as any legal, contractual, regulatory or other cloud-sector specific information security requirements.

Appendix I

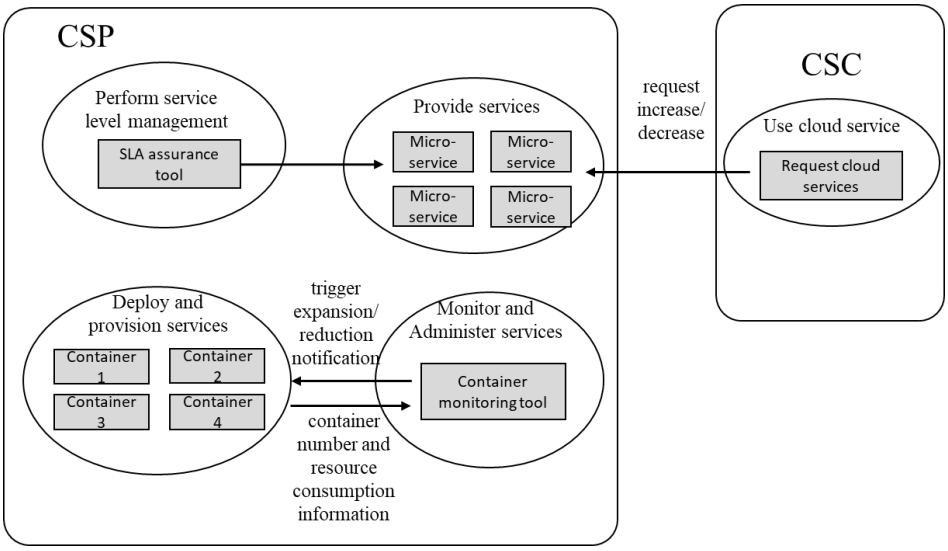
Use cases of containers

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

I.1 Fast Continuous Integration & Continuous Deployment

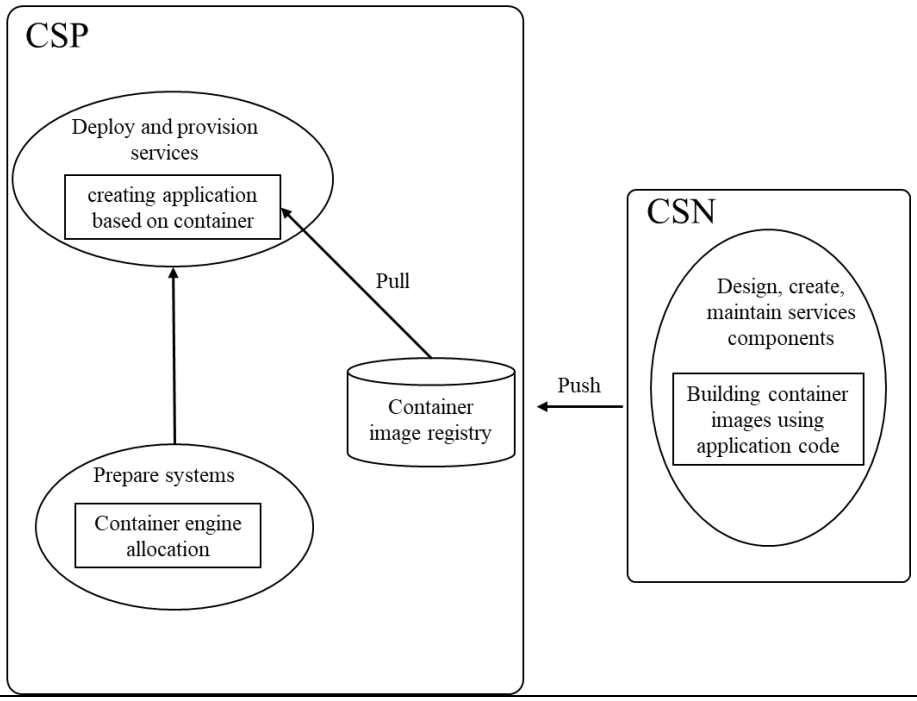
Title	Fast Continuous Integration & Continuous Deployment
Identifier	Fast-CI&CD
Description	This use case is common in the software development scenario. By using containers, the whole CI/CD process is accelerated and more convenient.
Roles	CSN, CSP, CSC
Figure (optional)	 There are several steps in this use case: (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 code registry and perform unit and integration test in a test environment; (4) once the new feature passed the test, the new container images will be built and pushed into the container registry; (5) which triggers the CD tool to pull the container image from container registry and re-deploy into the production environment; (6)either the test or build failure will be 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. In this use case, the CSP provide the micro-services (e.g. a web portal) for CSC to request. The CSP configures and provisions the container cluster for the micro-services through CLI (command line interface) or GUI (graphical user interface) in physical or virtual machines. The CSP has the capabilities to monitor the resource consumption of the container cluster (e.g. the container number in the container cluster and CPU/memory/disk usage of each container). Whenever the resource consumption exceeds a pre-defined threshold for expansion (e.g., the average CPU usage of the container cluster exceeds 80% for successive 10 minutes), the CSP will notify CSP to expand the scale of the container cluster following certain elastic scaling policy (e.g., add a group of two containers to the cluster). Consequently, the service provided by CSP is able to handle more requests from CSC, and the service level agreement could be fulfilled (e.g. CSP perform SLA assurance activity). Similarly, on the contrary, when the resource consumption of container cluster falls below a pre-defined threshold for reduction (e.g., the average CPU usage of the container cluster falls lower than 30% for successive 10 minutes), the CSP will notify CSP to reduce the scale of container cluster to save energy.
Roles	CSP, CSC
Figure (optional)	
Pre-conditions (optional)	

Post-conditions (optional)	
Derived requirements	− Clause 8.1.1 requirement (1) − Clause 8.1.2 requirement (6) − Clause 8.1.2 requirement (9) − Clause 8.1.2 requirement (10) − Clause 8.1.2 requirement (11) − Clause 8.2 requirement (3) − Clause 8.2 requirement (4)

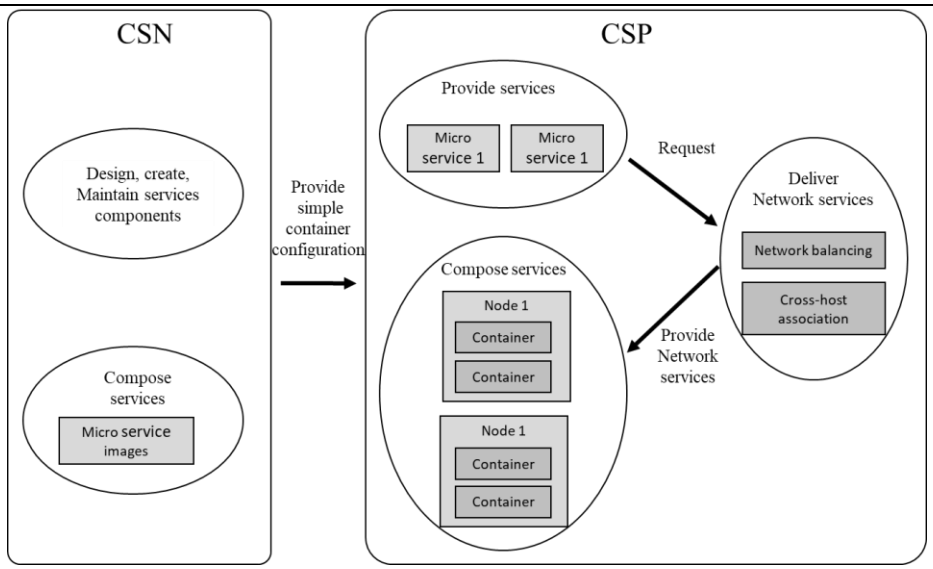
I.3 Container allocation for launching micro-service

Title	Container allocation for launching micro-service
Identifier	Container allocation
Description	This use case describes how to deploy applications based on the container in the cloud environment. In this scenario, the CSN is responsible for developing a program that describes what it needs to create an application. The CSP create 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. Detail procedures are as follows; CSN develops programs for the application CSN provides the CSP with configuration information to create microservices; CSP prepares container engines and allocates containers; CSP deploys applications based on the container.
Roles	CSP, CSN
Figure (optional)	

Pre-conditions (optional)	The container engine has various operating systems(OS)
Post-conditions (optional)	
Derived requirements	− Clause 8.1.1 requirement (17) − Clause 8.1.1 requirement (19) − Clause 8.1.2 requirement (8) − Clause 8.2 requirement (1) − Clause 8.2 requirement (5) − Clause 8.2 requirement (8)

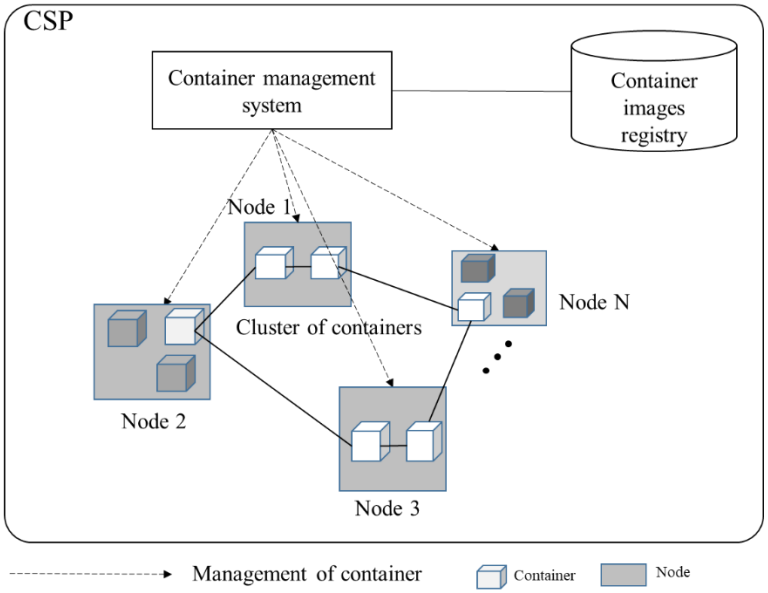
I.4 Load balancing containers in cloud application deployment

Title	Load balancing containers in cloud application deployment
Description	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. Based on that background, this use case describes how the containers communicate and coordinate with each other to accomplish the efficient deployment of microservice cloud software. In this scenario, CSN is responsible for creating micro-service and providing simple container configuration. And the CSP is responsible for the scheduling of computing resources flexibility, packaging code, test and deploying software efficiently. In addition, CSP also supports network load balancing and cross-node correlation to meet the communication and coordination requirements between components under the micro-service architecture. Whenever the micro-service changes, a new container image is built for later deployment.
Roles	CSN, CSP

Figure (optional)	 - CSN creates micro-service and provides simple container configuration for CSP; - CSP schedules the computing resources, packages code, tests and deploys software and provides log monitoring and management services. And CSP supports network load balancing and cross-node correlation to meet the communication and coordination requirements between components under the microservice architecture.
Pre-conditions (optional)	The containers can be deployed on different nodes in cloud environment.
Post-conditions (optional)	
Derived requirement	– Clause 8.1.1 requirement (2) – Clause 8.1.2 requirement (8) – Clause 8.1.2 requirement (12)

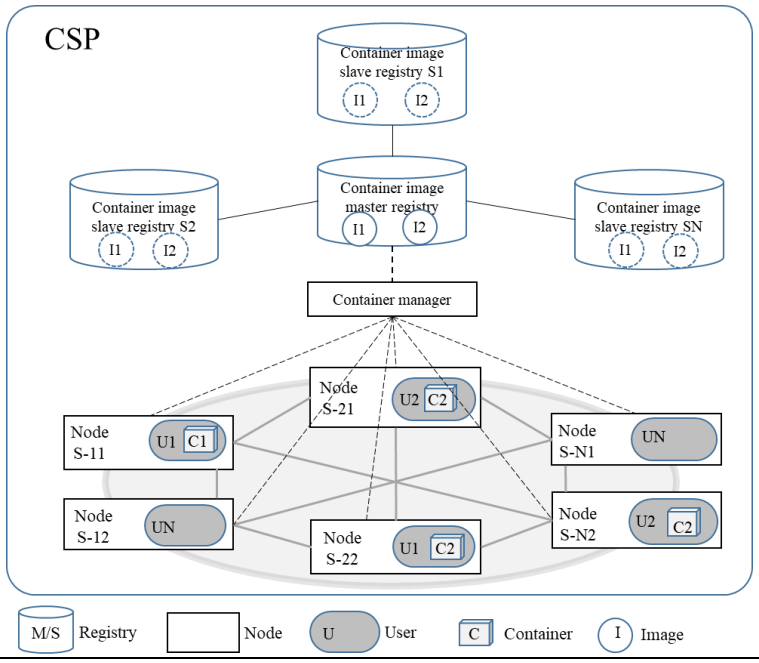
I.5 Container clustering across multiple node

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. 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. 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. And the container management function 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 the load balancing operation. And 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) – Clause 8.1.2 requirement (1) – Clause 8.1.2 requirement (2) – Clause 8.1.2 requirement (3) – Clause 8.1.2 requirement (4) – Clause 8.1.2 requirement (5) – Clause 8.1.2 requirement (6) – Clause 8.1.2 requirement (13) – Clause 8.2 requirement (7)

I.6 Container image distribution

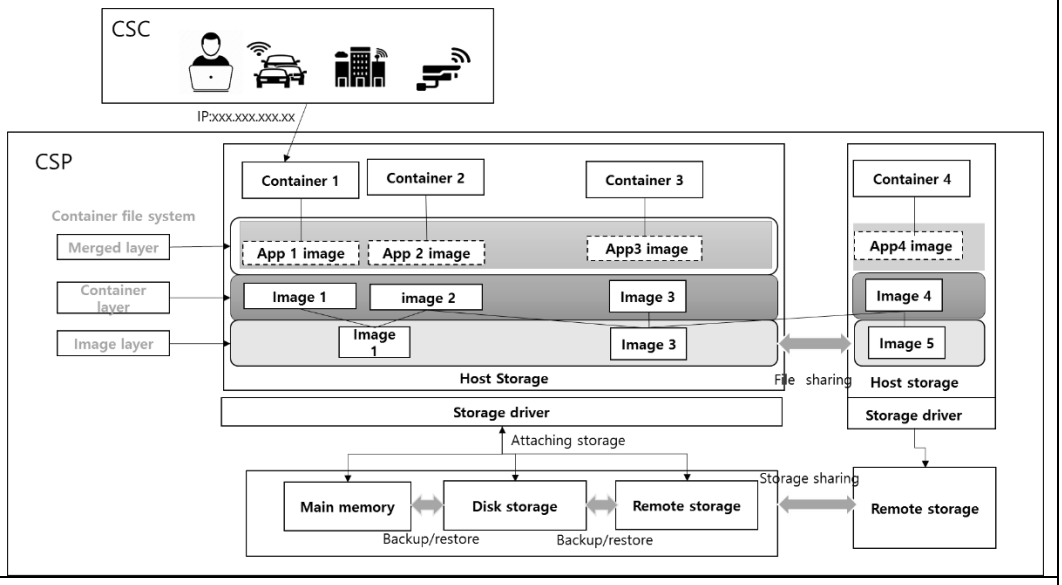
Title	Container image distribution

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, First, images (e.g., I1, I2) should be replicated (synchronized) from master-registry instance M1 to slave-registry instance (e.g., S1, S2, …, SN). 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 at master-registry instance. Registry instances are configured with synchronization policies.
Post-conditions (optional)	
Derived requirements	– Clause 8.1.2 requirement (14) – Clause 8.2 requirement (9) – Clause 8.2 requirement (10)

I.7 The container file system

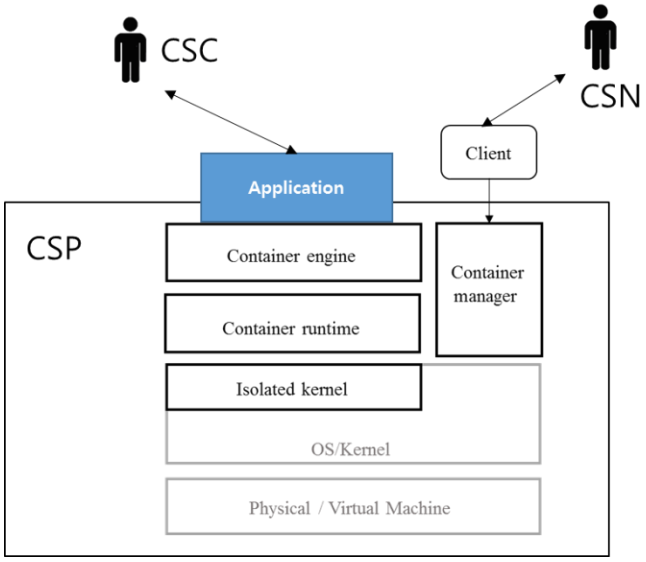
Title	The container file system
Description	This use case is about a container with a unifying file system and sharing images. A container file system makes a specific directory or files appear as the root file system which is independently used by one container. And container file system also needs to manage the images efficiently. For containers, the container file system runs on the host storage device (main memory, SSD, HD, 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. A unifying file system is a concept of mounting multiple file systems on a single mount point, and instead of creating a new file system type, unifying all directory entries is performed in the virtual file system (VFS) layer. With file system consolidation, directory entries from the child file system are merged with directory entries 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. As described above, 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. 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 the container is created, all changes are made in this layer. 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. And the merged layer includes link information of 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. 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 mentioned above, many images of the image layer exist on container storage, and the container images of the entire system are backed up and stored in disk storage or remote storage. Adding images to the image layer with sharing storage and could be available to the container layer as well, and images are continuously provided in the merged layer.
Roles	CSC, CSP

Figure (optional)	
Pre-conditions (optional)	
Post-conditions (optional)	
Derived requirements	– Clause 8.1.1 requirement (2) – Clause 8.1.1 requirement (3) – Clause 8.1.1 requirement (4) – Clause 8.1.1 requirement (5) – Clause 8.1.1 requirement (6) – Clause 8.1.1 requirement (7) – Clause 8.1.1 requirement (8) – Clause 8.1.1 requirement (18)

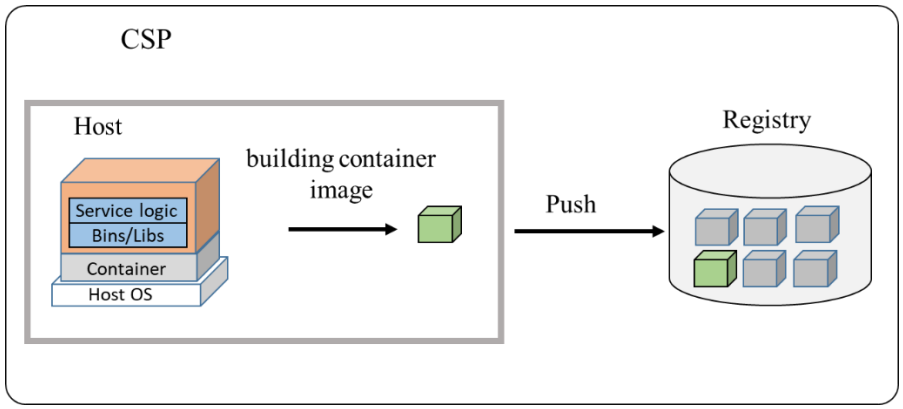
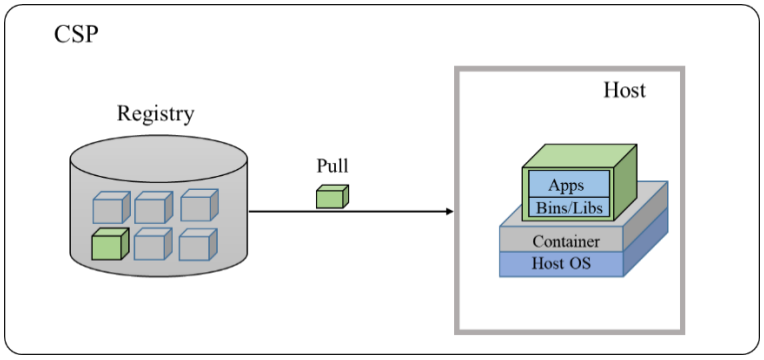
I.8 The general use case of container

Title	The general use case of container
Description	The container is generally consisted of four components: a container runtime, container engine, a container manager and isolated kernel. The container manager manages the deployment of the containerized application, containerized application life-cycle management (e.g., create, start, stop and delete applications). And container manager supports user interface (such as CLI and API) management for container engine. The container engine is a program that runs on the host operating system. The container engine creates and executes container with API. Container runtime provides daemon or software exposing API based on container engine. Container runtime provides low level capabilities such as execution of application and interfaces for capabilities of container engine

	Isolated kernel provides secure kernel to operate the containerized applications in independent kernel space. The isolated kernel logically separates the secured namespace and resource allocation in the host OS/kernel.
Roles	CSC, CSN, CSP
Figure (optional)	
Pre-conditions (optional)	
Post-conditions (optional)	
Derived requirements	– Clause 8.1.1 requirement (2) – Clause 8.1.1 requirement (3) – Clause 8.1.1 requirement (6) – Clause 8.1.1 requirement (9) – Clause 8.1.1 requirement (11) – Clause 8.1.1 requirement (12) – Clause 8.1.2 requirement (6) – Clause 8.2 requirement (5) – Clause 8.2 requirement (11)

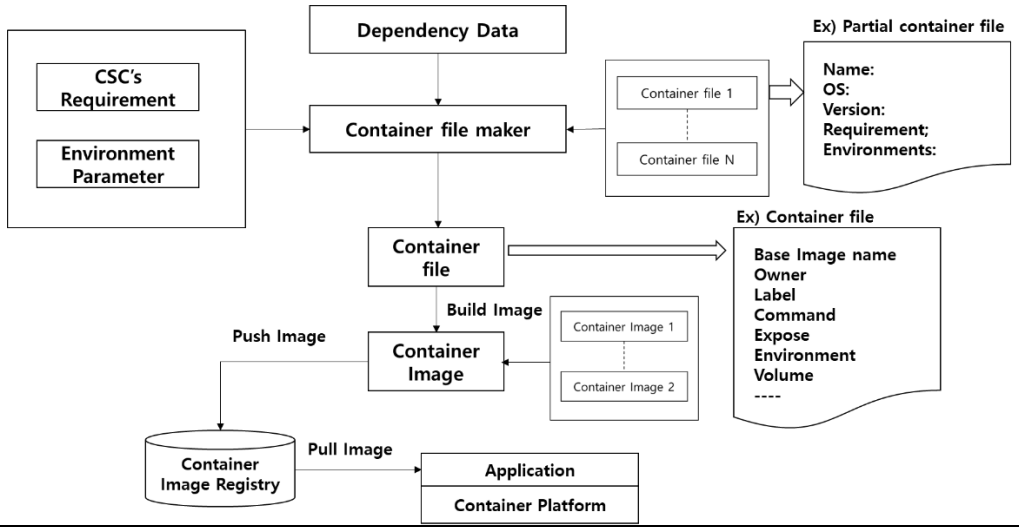
I.9 Building and registering container images

Title	Building and registering container images
Description	This use case shows how to build a container image and register it at 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 those container images can be registered using pushing commends. 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
Figure (optional)	 (1) Building container image and registering container image  (2) Pulling container image and creating container
Pre-conditions (optional)	
Post-conditions (optional)	
Derived requirements	– Clause 8.1.1 requirement (10) – Clause 8.1.1 requirement (11) – Clause 8.1.1 requirement (13) – Clause 8.1.1 requirement (14) – Clause 8.1.1 requirement (15) – 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. 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. To create a container, a user's container image is created with a container file in which a series of commands are described. As shown in the figure, containers are used in combination with various images and the container file is manually written by the developer. 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 on the container's file system and writes it to the script. Images are created according to the content. 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. 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. In the partial container file in the figure, the requirements are the CSC's requirement 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. Additional dependency data has dependency information between requirements during the installation. Container file maker refers to the dependency data and finally create the container file through the combination of requirement, env 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) − Clause 8.1.1 requirement (11) − Clause 8.1.1 requirement (16) − Clause 8.1.1 requirement (18)

Appendix II

An example for illustration for the comparison between container and virtual machine

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

This is the comparison between containers and virtual machines. The numbers in this table can be changed rapidly.

Containers have similar resource isolation and allocation benefits as virtual machines but a different architectural approach allows them to be much more portable and efficient. Each virtual machine includes the application, the necessary binaries and libraries and an entire guest operating system - all of which may be tens 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 operating system. They are also not tied to any specific infrastructure –containers can run on most types of servers and operating systems, and most cloud infrastructures.

Comparison between virtual machines and containers

	Virtual machines	Containers
Software stack	Application + Binary/Library+ OS	Application + Binary/Library
Image size	Tens of GB	Tens to hundreds of MB
Instances per host	Tens of	Hundreds to thousands of
Deployment time	Several minutes	Several seconds

Bibliography

[b-ITU-T H.764] Recommendation ITU-T H.764 (2012), *IPTV services enhanced script language.*

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

_____