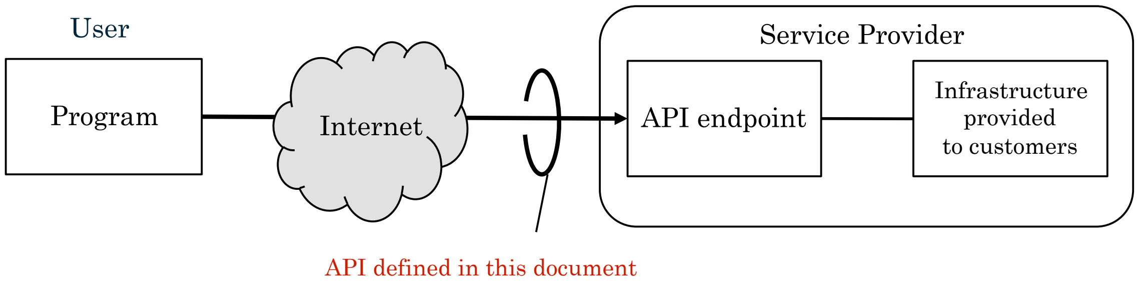
API Design for IaaS Cloud Computing Service

November 13, 2009

FUJITSU LIMITED
Shiodome City Center
1-5-2 Higashi-Shimbashi
Minato-ku, Tokyo
Japan 105-7123
http://www.fujitsu.com

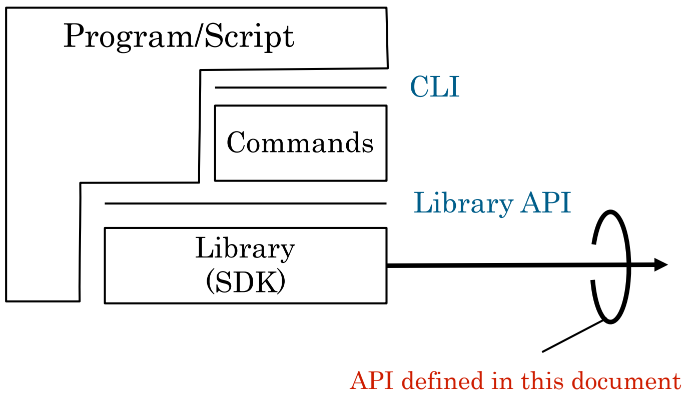
1. Introduction

This document is intended to be an input to the standardization effort for cloud interoperability. It defines the API exposed by cloud computing service providers on the Internet. It covers services that dynamically offer to users an infrastructure that consists of servers (CPUs and memory), storage, and network connectivity between servers. For interoperability discussion, this document focuses on the cloud management model, and does not go into the details of a specific API syntax.
Specific detail of the infrastructure is not mandated by this specification, but compliance to the API is.


User
Program
Internet
Service Provider
API endpoint
Infrastructure provided to customers
API defined in this document

[Figure 1] API defined in this document

In order to accommodate various development and run-time programming environments, this specification allows developers to create libraries for specific programming languages (using a Software Development Kit or SDK) and commands (that can be used via a Command Line Interface or CLI). However, the library API and the CLI are out of the scope of this specification.



[Figure 2] Library API and CLI

The terms used in this document are defined as follows:

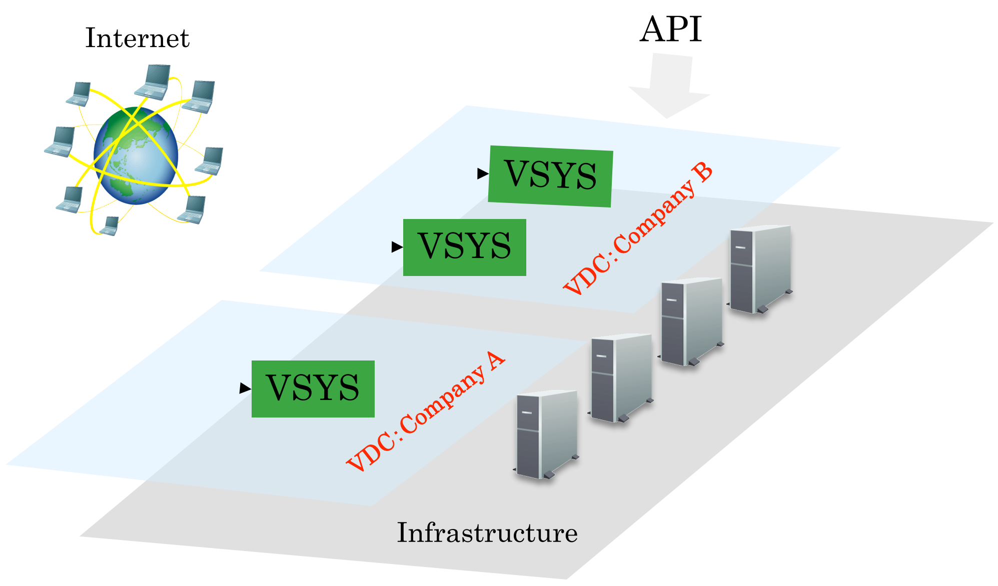
- **Service provider**
  The service provider is an entity that provides cloud computing services.

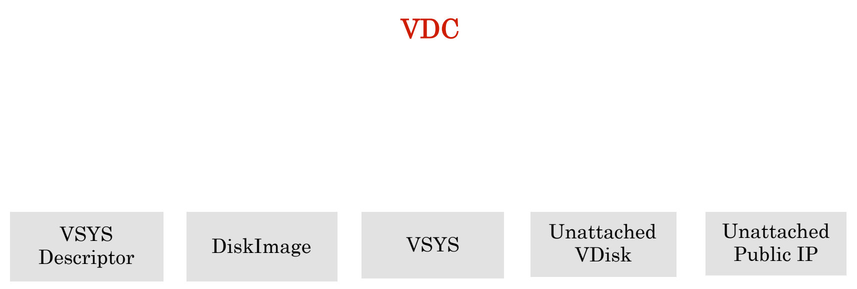
- **User**
  The user is an entity that has the privilege to access the service provider. The user is identified by his/her userID. Each userID is assigned authentication information required to use the API.

- **API endpoint**
  The API endpoint is the location to be accessed when the API is used. It is identified by its URL.

2. VDC

A VDC is a virtual data center that a service provider provides to a user. One user’s VDC is logically separated from other users' VDCs and the user sees the illusion that the user has his/her own data center. [Figure 3] shows resources that can be created in a VDC. Components of VDC, such as VSYS descriptor and VSYS are explained in the following sections.





[Figure 3] VDC

Operations on VDC:

registerVSYSDescriptor: Registers a VSYS Descriptor in the VDC (see Section 3).

listVSYSDesciptor: Returns a list of IDs of the VSYSDescriptors in the VDC. Attributes of the VSYSDescriptors can be obtained at the same time.

createVSYS: Creates a VSYS based on a specified VSYS descriptor. See Section 4 for VSYS. By default, this operation implies PowerOn[1] of all VServers and Extended Function Modules (EFMs) defined in the VSYS descriptor and automatic starting of the VSYS. An option can also be selected so that VSYSes are powered on, but do not start. The start order (of the contained resources) is not specified.

listVSYS: Returns a list of VSYS IDs in the VDC.

createVDisk: Creates a VDisk in the VDC, which can be attached to a VServer (see Section 6).

listVDisk: Returns a list of the IDs of VDisks in the VDC. It will also indicate if the VDisk is associated with a particular VServer.

allocatePublicIP: Allocates a Public IP that is a global IP address (see Section 7).

listPublicIP: Returns a list of the PublicIPs in the VDC. It will also indicate if the PublicIP is associated with a particular VNIC (see Section 8).

registerDiskImage: Registers a DiskImage (see Section 11).

listDiskImage: Returns a list of IDs of the DiskImages in the VDC. ProductIDs for the contents of a DiskImage (see Section 13) are specified when registering.

registerProductID: Registers product information.

listProductID: Returns a list of ProductIDs in the VDC.

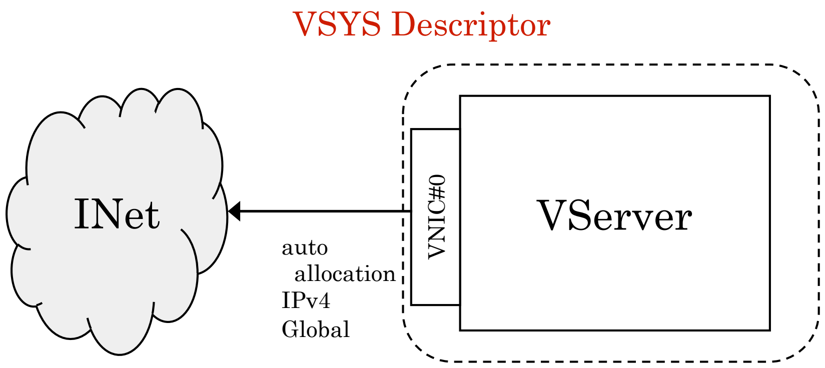
[1] Refer to individual VServer states and EFM states.

3. VSYS Descriptor

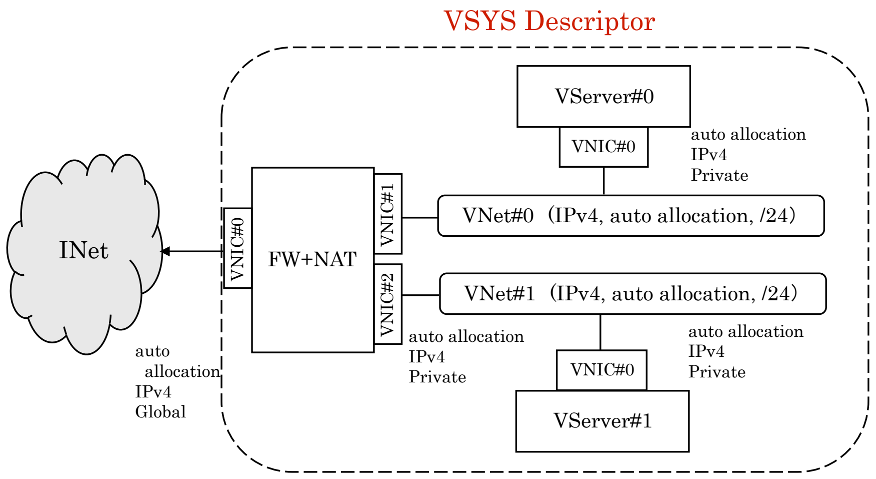
The VSYS descriptor is the configuration definition, which is equivalent to the design of the virtual system (VSYS) that will be created for the user at a data center of the service provider. It defines the resources that make up the VSYS, the connections and relationships among the resources, and configuration information for the resources. Different VSYS descriptors vary in complexity: for example, an empty VSYS descriptor without any resources, a VSYS descriptor containing one server, a VSYS descriptor for a three-tier web system. VSYS descriptors are identified by their IDs.

There is a repository of VSYS descriptors in a service provider. Some VSYS descriptors are provided by the service provider, whereas other VSYS descriptors are derived by users from existing VSYS descriptors provided by a service provider. Some VSYS descriptors are public and are accessible by all users, while others are private and accessible only by the owner. VSYS descriptors may include specifications of restrictions that must be reflected in the created VSYS. For example, a service provider can create VSYS descriptors that only allow a defined network configuration.

Examples of VSYS descriptor are shown in [Figure 4]

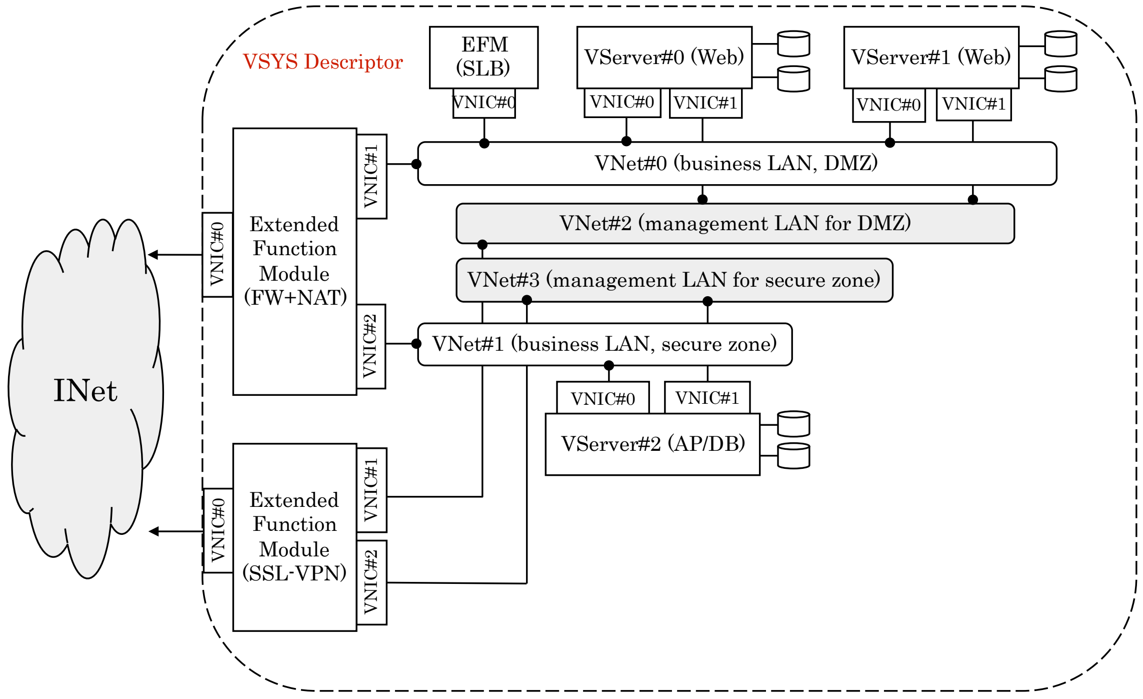


(a) Single VServer configuration



FW: Firewall, NAT: Network Address Translation

(b) Two VNets configuration



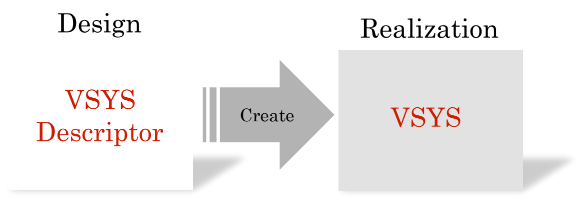
FW: Firewall, NAT: Network Address Translation, SLB: Server Load Balancer,

SSL-VPN: Virtual Private Network using the SSL protocol

(c) Complex configuration with management VNets.

[Figure 4] Examples of VSYS descriptors

To use a working VSYS, the user creates a VSYS based on the appropriate VSYS descriptor.



[Figure 5] VSYS Descriptor and VSYS

The external representation of VSYS descriptor is called the **VSYS package**. Users can export a VSYS descriptor in the form of a VSYS package, or register a VSYS descriptor by importing a VSYS package. The VSYS package is expected to be an extension of OVF[2] (Open Virtual Format; DMTF/DSP0243).

Operations on VSYSDescriptor:

unregisterVSYSDescriptor: Unregisters the VSYSDescriptor from the VDC.

getVSYSDescriptorAttributes: Returns information on all or specific attributes of the VSYSDescriptor.

VSYSDescriptor states:

(none).

Attributes of VSYSDescriptor:

creatorName: The name of the person who created the VSYSDescriptor. He/she may not be a user of this service provider.

registerer: The ID of the user that registered the VSYSDescriptor.

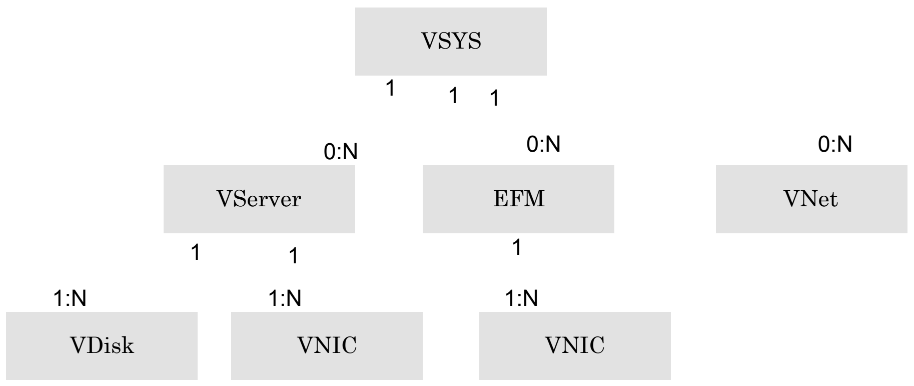
registrationTime: The time that the VSYSDescriptor was registered. Time must be in UTC.

description: A plain language description of the contents of the VSYSDescriptor.

[2] http://www.dmtf.org/standards/published_documents/DSP0243_1.0.0.pdf

4. VSYS

A VSYS is a system that is created on a data center of a service provider and runs as a separate coordinated system. A VSYS is a collection of resources such as VServers, their related type Extended Function Modules (EFMs), VDisks, VNets, and VNICs [Figure 6]. A VSYS can hold an arbitrary number of resources and is usually treated as a unit of management; All VServers in a VSYS can be stopped (made inactive) in one operation; a VSYS typically has a single firewall; a VSYS may be imported from other service providers. In a VDC, a user can have multiple VSYSes.



[Figure 6] Resource Structure

The only way to create a VSYS is to use a VSYS descriptor as the base. Once a VSYS is created, it is possible to add, delete, and modify resources in the VSYS as long as the restrictions defined in the VSYS descriptor are not violated.

<u>Operations on VSYS:</u>

destroyVSYS: Destroys a VSYS. All resources within the VSYS are destroyed. They are lost forever.

startVSYS: Starts all VServers and EFMs in the VSYS. The starting order is not specified.

stopVSYS: Stops all VServers and EFMs in the VSYS instance. The stopping order is not specified.

getVSYSConfiguration: Returns configuration information on the VSYS.

getVSYSAttributes: Returns information on all or specific attributes of the VSYS.

createVServer: Creates a VServer in the VSYS. The DiskImageID (see Section

11) that is used as the initial content for the boot disk must be specified.

listVServer: Returns a list of the IDs of VServers in the VSYS.

createEFM: Creates an EFM (see Section 12) in the VSYS.

listEFM: Returns a list of the IDs of EFMs in the VSYS.

createVNet: Creates a VNet associated with this VSYS.

listVNet: Returns a list of the IDs of VNets in the VSYS.

getINet: Returns the INet provided by the service provider.

VSYS states:

(Refer to individual VServer states and EFM states).

Attributes of VSYS:

VSYSID: A unique identifier of a VSYS.

creator: User ID of the user who created the VSYS.

createTime: Indicates when the VSYS was created. Time must be in UTC.

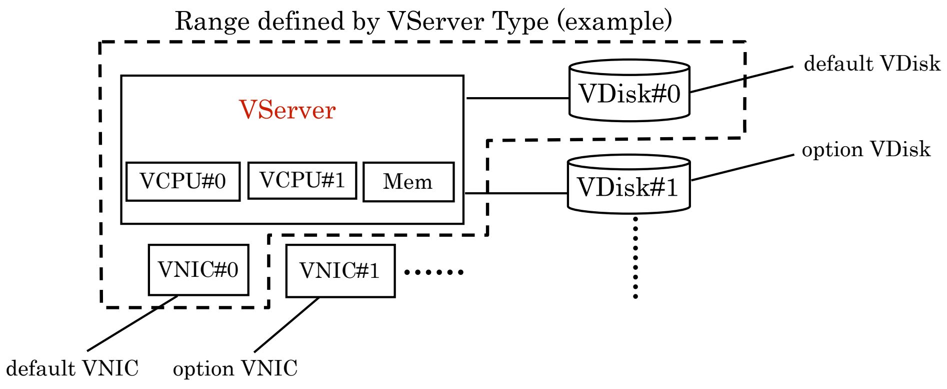
description: As included in the VSYS descriptor.

baseDescriptor: ID of the VSYS descriptor that was used as the base.

5. VServer

A VServer is a server that is available to users on the data center of a service provider. It is implemented as a virtual machine. A VServer consists of VCPUs (virtual CPUs) and memory. VDisks (virtual disks) can be attached to a VServer. VNICs (virtual network interface cards) can be created for a VServer.

The VServer type, which must always be specified when creating a VServer instance, defines the number and performance of VCPUs, the memory capacity, and a combination of default VDisks and default VNICs. The service provider publishes supported VServer types. As part of VServer type definition, the service provider can add restrictions and/or define parameters, such as prohibition of using additional VNICs.



[Figure 7] An example of VServer Type

[Table 1] An example of VServer types
(VServer types are defined by each service provider)

VServer type	Number of VCPUs	Total CPU performance	Machine bits	Memory	Default VDisk	Default VNIC
S-2010	1	0.5 CU	32b and 64b	4GB	1 (10GB)	1 (auto private IP)
M-2010	1	2.0 CU	32b and 64b	16GB	2 (50GB+500GB)	1 (auto private IP)
L-2010	2	4.0 CU	32b and 64b	32GB	2 (100GB+1TB)	1 (auto private IP)

CU (Computing Unit) is expected to be defined by the provider as part of a SLA (Service Level Agreement). For example, 1CU may be defined as approximately corresponding to Intel Xeon 1GHz in 2009.

Zero or more VServers are created according to the configuration definition in the VSYS descriptor when a VSYS is created. After a VSYS is created, additional VServers can be added to the VSYS (unless such addition is prohibited by the VSYS descriptor).

The creation of a VServer is accompanied by (1) the assignment of a VServer instance ID (which is unique within the service provider), (2) allocation of resources to be managed by the VServer, (3) virtual power-on (transition to the *inactive* state), and, usually, (4) transfer of control from the system disk (explained in Section 6, "VDisk") to the OS (transition to the running state). It is possible to leave the OS inactive.

VDisk and VNIC resources can also be added to a VServer that has been created (see the explanations in Sections 6 through 11).

When a stopVServer operation is performed via the API, a shutdown request is issued to the OS of the VServer and the VServer enters the *stopping* state. When the stop operation is completed, the VServer enters the *inactive* state. If the OS fails to complete the shutdown process internally, the VServer stays in the *stopping* state.

It is also possible to start the shutdown process within the VServer OS. However, this shutdown process cannot be observed as a VServer state (the VServer still remains in the running state). Only after the shutdown process within the OS is successfully completed, does the VServer state change to *inactive*.

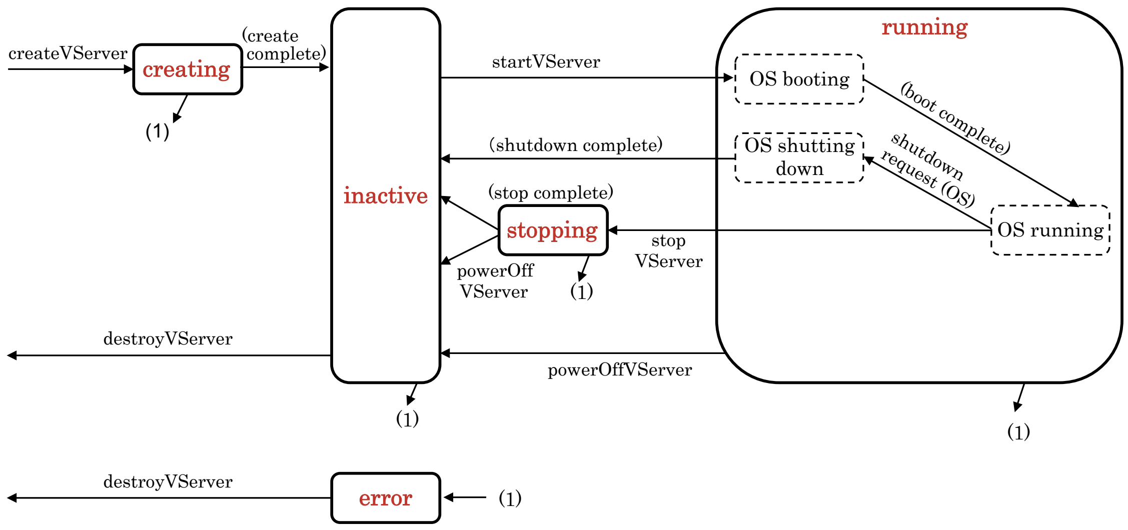
A startVServer operation can only be performed when the VServer is in the *inactive* state. It causes the OS to boot from the system disk of the VServer. A restartVServer operation is equivalent to a stopVServer operation followed by a startVServer operation.

When a VServer is destroyed, the VCPUs, memory, default VDisks, and default VNICs of the VServer no longer exist (the content of the default VDisks is lost too). The optional VDisks attached to the VServer are detached and become unattached VDisks but the content may be preserved (see Section 6, "VDisk," for information on VDisks).

When a powerOffVServer operation is performed, the VCPUs, memory, and VNICs (default and optional) of the VServer no longer exist. When a powerOnVServer operation is performed, these resources are newly allocated. A powerResetVServer operation is equivalent to a powerOffVServer operation followed by a powerOnVServer operation. powerOffVServer is forcibly performed regardless of the VServer OS state. Therefore, if a powerOffVServer operation is performed while the OS is operating, inconsistencies introduced to VDisk content may cause the OS reboot to fail.

The content of VDisks (default and optional) attached to the VServer will be preserved even when a StopVServer or powerOffVServer operation is performed. When the VServer OS reboots, the previous VDisk content can be accessed.

The VServer state transitions are shown in [Figure 8]



[Figure 8] State transition of VServer

When a VServer is created, user configuration data can be passed (as server-specific user data) to the VServer. This configuration data is provided to the VServer using the approach described by OVF Environment.

Depending on the size or characteristics of the configuration data an infrastructure provider may choose to provide extended or dynamic configuration data through an external configuration service. In this case the OVF Environment document must contain a Property entry for the configuration service as follows:

Name: "configuration-service"

Value: the URL of the configuration service

Further, this external configuration service is expected to provide the entire user data from the “/user-data” location; and individual system data items can be obtained from corresponding paths that begin with “/system-data/”.

For example, if the configuration service URL is http://169.254.169.254/ then the VServer can obtain server-specific user data and server-specific system data by accessing the following resources:

http://169.254.169.254/user-data

http://169.254.169.254/system-data/

The VServer OS boots from the VDisk that is specified as the VServer system disk. The system disk content that is used for initial boot is a copy of the DiskImage specified for the VDisk. This means that the OS user authentication method generally depends on the content of this DiskImage.

Services compatible with this API provide a framework that will ensure that when the OS initially boots, the OS will create authentication information (such as random passwords), which can then be retrieved via the API. The program that will safely transfer authentication information from inside the OS to the service provider is incorporated in the DiskImage in advance. The specific transfer method depends on the service provider (for the time being).

<u>Operations on VServer:</u>

- **destroyVServer**: Destroys a VServer.
- **powerOnVServer**: Turns on the power to the VServer.
- **powerOffVServer**: Turns off the power to the VServer.
- **powerResetVServer**: Resets the power to the VServer.
- **startVServer**: Boots the OS.
- **stopVServer**: Shuts down the OS.
- **restartVServer**: Reboots the OS.
- **getVServerStatus**: Returns the status of the VServer.
- **getVServerAttributes**: Returns the attributes of the VServer.
- **getInitialPasswordVServer**: Retrieves initial authentication information from the OS.
- **createVNIC**: Creates a VNIC associated with this VServer.
- **listVNIC**: Returns a list of the VNICs associated with this VServer.

<u>VServer states:</u>

- creating
- inactive
- running
- stopping
- error

<u>VServer attributes:</u>

- **VServerID**: A unique identifier of the VServer.
- **creator**: ID of the user who created the VServer instance
- **createTime**: Indicates when the VServer was created. Time must be in UTC.
- **VServerType**: The type of the VServer, see Table 1 for examples.

VServerUserData: server-specific user data.

VDiskIDs: IDs of attached VDisks.

DiskImageID: The ID of the DiskImage registered with this VServer that was used for the system disk.

VNICIDs: IDs of VNICs attached to this VServer.

6. VDisk

VDisks are disks that are seen as block access devices by the OS running on the VServer. VDisks are classified into two types: default VDisks, which are defined by the VServer type, and optional VDisks, which are all others. The latter type includes VDisks that are created for backup.

VDisks are created (1) when a VSYS is created (createVSYS) based on a VSYS descriptor which has default or optional VDisks in it, (2) when a VServer is created (createVServer) based on a VServer type which has default VDisks, and (3) when an optional VDisk is explicitly created (createVDisk, backupVDisk).

The VDisk reliability (redundancy that would be assured in case of a hardware failure) depends on the service provider and determined by the Service Level Agreement.

A VDisk can be attached to a VServer when the VServer is in the inactive or running state.

When attaching a VDisk to a VServer, the user specifies its sequence number indicating its relative position on the VServer (this position is hereinafter called the VDisk position). The VServer type determines how many default VDisks can be attached to a VServer and what capacity the default VDisk at each VDisk position has. The mapping between the VDisk positions and the device names as seen from the OS is determined depending on the service provider and on the OS running on the VServer. The table below shows an example.

[Table 2] Example of VDisk position to OS device mapping

VDisk Position	OS Device Name
0	xvda
1	xvdb
2	xvdc
...	...

A VServer boots from the VDisk that is at VDisk position #0 (this VDisk is called the system disk).

When a user issues the backupVDisk operation, a separate optional VDisk is created that has the same capacity as the specified (source) VDisk, and then the content of the source VDisk is copied (on a byte-by-byte basis) to the newly created VDisk. The implementation method of backupVDisk may vary. All service providers must support the VDisk backup when the VServer to which the VDisk is attached is in the *inactive* state, or when the VDisk is not attached to any VServer. Some service providers may also support the VDisk backup even when the VServer is in the running state. After a VDisk is backed up, the ID of the source VDisk will remain on the backup VDisk as an attribute. The backup VDisk is not attached to any VServer. Multiple backup VDisks can be created and kept for a single source VDisk.

A VDisk the content of which has already been backed up can be connected to any VServer by an attach operation or discarded by a destroyVDisk operation.

If a VServer is unable to boot, it is possible to examine the cause of this failure by using a backup operation to copy the content of the possibly faulty VDisk and then attaching the copy to another VServer. The copied VDisk can be repaired, detached, and then restored to the original VDisk.

To restore a VDisk, copy the VDisk containing the restore data as is (on a byte-by-byte basis) to the VDisk attached to the VServer. At the restore time, the VServer to which the restore destination VDisk is attached must be in the inactive state.

<u>Operations on VDisk:</u>

destroyVDisk: Destroys the VDisk and its contents.

attachVDisk: Attaches the VDisk to a specified VServer.

detachVDisk: Detaches the VDisk for the VServer to which it is currently attached.

backupVDisk: Backups the VDisk. The contents of the VDisk are copied to a newly created VDisk as a backup.

restoreVDisk: Copies the contents of the specified backup VDisk to the VDisk. All existing contents of the VDisk are replaced by the backup contents.

getVDiskStatus: Returns the status (attached or un attached) of the VDisk.

getVDiskAttributes: Returns the attributes of the VDisk.

<u>VDisk states:</u>

unattached

attached

<u>VDisk attributes:</u>

VDiskID: Unique identifier for this VDisk.

creator: ID of the user who created the VDisk.

createTime: Indicates when the VDisk was created. Time must be in UTC.

size: The size in GB of the VDisk.

attachedTo: ID of the VServer to which the VDisk is currently attached.

DiskImageID: If the VDisk was copied from a DiskImage, this attribute indicates the ID of the original Image.

backupSrcVDisk: If this VDisk was created by a backup operation, this attribute indicates the ID of the VDisk from which this VDisk was backed up.

7. PublicIP

A PublicIP is a global Internet IP address. The address is assigned by the service provider. The user cannot set a numeric value for the address. A PublicIP can be allocated automatically at the time of VNIC creation, or can be explicitly allocated (allocatePublicIP). The explicitly allocated PublicIP can then be attached to a VNIC when the VNIC is created. When a VNIC is destroyed, the automatically allocated PublicIP is freed, but the explicitly allocated PublicIP is only detached and not freed. VNIC is described in the next section.

<u>Operations on PublicIP:</u>

freePublicIP: Releases the PublicIP.

getPublicIPSatus: Returns the status of the PublicIP.

getPublicIPAttributes: Returns the attributes of this PublicIP.

PublicIP states:

unattached

attached

PublicIP attributes:

v4v6Flag: Shows whether the PublicIP is IPv4 or IPv6.

autoAllocateFlag: Shows whether the PublicIP was automatically allocated or explicitly allocated.

allocateUser: The ID of the user who allocated this PublicIP.

allocatedTime: Time at which the PublicIP was allocated.

8. VNIC

A VNIC is a network interface that can be created, associated with a VServer, and can be seen from the OS on the VServer. A VNIC is attached to a VNet (see Section 9) or an INet (see Section 10) when it is created. Once created, the VNIC cannot be disassociated from the VServer nor detached from the VNet until it is destroyed.

VNICs are classified into two types: default VNICs, which are defined by the VServer type, and optional VNICs, which are all others.

VNICs are created (1) when a VSYS is created (createVSYS) based on a VSYS descriptor which has default or optional VNICs in it, (2) when a VServer is created (createVServer) based on a VServer type which has default VNICs, and (3) when an optional VNIC is explicitly created (createVNIC).

The sequence number indicating its relative position on the VServer, called the VNIC position starting from 0, distinguishes a VNIC within a VServer. The VNIC position is determined when a VNIC is created, and is given the smallest unused number. The mapping between the VNIC position and the device name as seen from the OS is determined depending on the service provider and the OS running on the VServer. The following table shows an example.

[Table 3] Example of VNet position to OS device mapping

VNet Position	OS Device Name
0	eth0
1	eth1
2	eth2
...	...

When a VNIC is created, a MAC address (an Ethernet (IEEE 802) MAC address) must be automatically generated and assigned. This MAC address must be unique within the service provider.

The MAC address must not change until the VNIC is destroyed. The MAC address of a VNIC remains unchanged even when the VServer power is turned off and back on.

The VServer OS functionality allows up to one IPv4 address and up to one IPv6 address to be assigned to one VNIC, which by default are defined by the service provider.

The IP address and netmask for the DHCP server (IPv4 and/or IPv6) can also be specified beforehand by the user. The IP address must be within the range of network addresses that can be assigned to the VNET to which the VNIC is attached. The netmask must be the same as that of the VNet. Each service provider determines whether IP addresses can be specified by users or are automatically assigned by the service provider.

VNICs do not support VLAN tags. When a MAC frame that contains a VLAN tag is sent from the OS, it will be discarded before it enters the VNet. No MAC frame that contains a VLAN tag will be received from VNICs.

<u>Operations on VNIC:</u>

destroyVNIC: Detaches the VNIC from the VNet, dissociate it from the VServer, and destroy it.

getVNICAttributes: Returns the attributes of the VNIC.

<u>VNIC states:</u>

(none)

VNIC attributes:

MAC Address: The MAC address of the VNIC.

IPv4Address, IPv4Netmask, IPv4AutoAllocateFlag: IPv4 properties of the VNIC.

IPv6Address, IPv6Netmask, IPv6AutoAllocateFlag: IPv6 properties of the VNIC.

9. VNet

A VNet is a LAN (compatible with the IEEE 802 MAC service) that is available within the VSYS. A VNet can be regarded as a virtual Ethernet switch. When a VNet is created, it is assigned up to one set of an IP subnet network address and netmask for each protocol (IPv4 and IPv6).

If the user is permitted to specify the network address, the user can assign the network address and netmask to the createVNet arguments. Otherwise (in an ordinary case), the user can specify the netmask only and must accept the automatically assigned network address. If the netmask is not specified, it defaults to a netmask in the format in which the host part is 8 bits (e.g., 255.255.255.0 in the case of IPv4).

VNets do not support VLAN tags.

The following server features are provided in each VNet:

- DHCPv4 and/or DHCPv6 server
- Server that provides server-specific information

The user cannot directly access the configuration information of a VNet.

Operations on VNet:

destroyVNet: Destroys this VNet.

getVNetAttributes: Returns the VNet attributes.

VNet state:

(none)

VNet attributes:

All of these below are IPv4 attributes of this VNet:

IPv4NetworkAddress,

IPv4Netmask,
IPv4DefaultGateway,
IPv4AutoAllocateFlag

All of these below are IPv6 attributes of this VNet:
IPv6NetworkAddress,
IPv6Netmask,
IPv6DefaulGateway,
IPv6AutoAllocateFlag

10. INet

The Internet and the provider network segment to access the Internet are outside of the VSYS. The VSYS-side boundary between the VSYS and the Internet access network is a VNIC contained in the VSYS. A specifier, INet, representing an Internet access network, is provided to ensure that the system can be instructed to connect the boundary VNIC to the Internet access network rather than a VNet included in the VSYS (see Fig. 4).

The INet is provided by the service provider. It cannot be created by the user.

<u>Operations on INet:</u>
(none)

<u>INet state:</u>
<u>(none)</u>

<u>INet attributes:</u>
(non)

11. DiskImage

A DiskImage is content (byte string) that is copied to a VDisk and used there. A DiskImage is identified by a DiskImageID that is provided to indicate the content's identity. DiskImageIDs are specific as URIs and at least unique within the service provider.

There is a repository of DiskImages maintained by the service provider. Some DiskImages are provided by the service provider, whereas other DiskImages are registered by users. The users can import a new DiskImage, or can derive a DiskImage from a VDisk. Some DiskImages are public and are accessible from all users, others are private and accessible only by the owner.

A DiskImage is used as the content of the system disk. When a VServer is created, the content of the DiskImageID identified by the specified DiskImageID is copied to the system disk (the default VDisk located at VDisk position #0) of the VServer. The VServer will boot from the system disk.

The supported DiskImage content and formats depend on the service provider and the OS. Examples of content and formats include the PC BIOS format (MBR: Master Boot Record) and the GPT (GUID Partition Table) format. A DiskImage may contain multiple disk partitions, one of which must be the boot partition.

The content of the boot disk partition can be any software stack, which may consist only the OS or span a wide variety of software, including the OS, middleware, and business applications.

<u>Operations on DiskImage:</u>

unregisterDiskImage: Unregisters the DiskImage from the VDC.
getDiskImageAttributes: Returns the attributes of this DiskImage.

<u>DiskImage status:</u>

(none)

<u>DiskImage attributes</u>:

DiskImageID: An identifier unique within the service provider.
size: The size of the DiskImage in GB.
OSType: OS type whether it is a para-virtualized kernel or standard kernel, if a boot image. This distinction is necessary for some VM software.
OSName: The name of the OS contained in the DiskImage, if a boot image.
productID list: ProductIDs can only be specified when registering a DiskImage. When registered from a VDisk, productIDs associated with the VDisk is inherited upon DiskImage registration.

creatorName: The name of the person who created the DiskImage. He/she may not be a user of this service provider.

registerer: The ID of the user that registered the DiskImage.

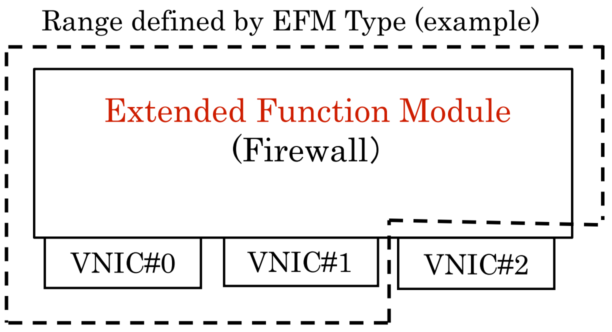
regisreredTime: The time that the DiskImage was registered. Time must be in UTC.

description: A plain language description of the contents of the DiskImage.

licenseInfo: Information pertaining to the licensing required for use of this DiskImage.

12. Extended Function Module

An Extended Function Module (EFM) is a function module that can be located in the VSYS instead of a VServer. It has specific features and provides APIs required for the features. An EFM has VNICs to be attached to VNETs. In some cases, the EFM determines the number of VNICs; in other cases, VNICs can be added.



[Figure 9] An example of EFM Type

Typical features an EFM can provide include firewall, NAT, SSL-VPN, load-balancer, router, and IPv4-v6 gateway features. Some EFMs provide multiple features. The feature sets are identified by EFMTypes, which are provided by a service provider. As far as the Cloud API is concerned, EFMs are only defined by their own interface and can be implemented in various ways (e.g. run on virtual machines or not).

An EFM is a black box in that the user cannot modify its internal configuration. All EFM features can only be managed via the APIs provided by EFM.

<u>Operations on EFM:</u>

destroyEFM: Destroys an EFM.

powerOnEFM: Turns on the power of the EFM.

powerOffEFM: Turns off the power of the EFM.

startEFM: Starts the EFM.

stopEFM: Stops the EFM.

getEFMStatus: Returns the status of the EFM.

getEFMAttributes: Returns the attributes of the EFM.

createVNIC: Creates a VNIC associated with this EFM.

listVNIC: Returns a list of IDs of VNICs associated with this EFM.

(Other operations are dependent on the EFM)

<u>EFM Status:</u>

inactive

running

stopping

error

EFM Attributes:

EFMID: A unique identifier of the EFM.

creator: ID of the user who created the EFM.

createTime Indicates when the EFM was created. Time must be in UTC.

EFMType: The type of the EFM.

VNICIDs: IDs of VNICs attached to this EFM.

(Other attributes are dependent on the EFM)

13. ProductID

ProductID is the identification information that is included in a resource. ProductID can be associated with a DiskImage when registering them. For example, a commercial operating system in a DiskImage is identified by the ProductID.

<u>Operations on ProductID:</u>

unregisterProductID: Unregisters this ProductID.

getProductIDAttributes: Returns the attributes of this ProductID.

ProductID status:

(none)

ProductID attributes:

name: The name of the specific product references by the ProductID.

description: Human readable description of the Product.