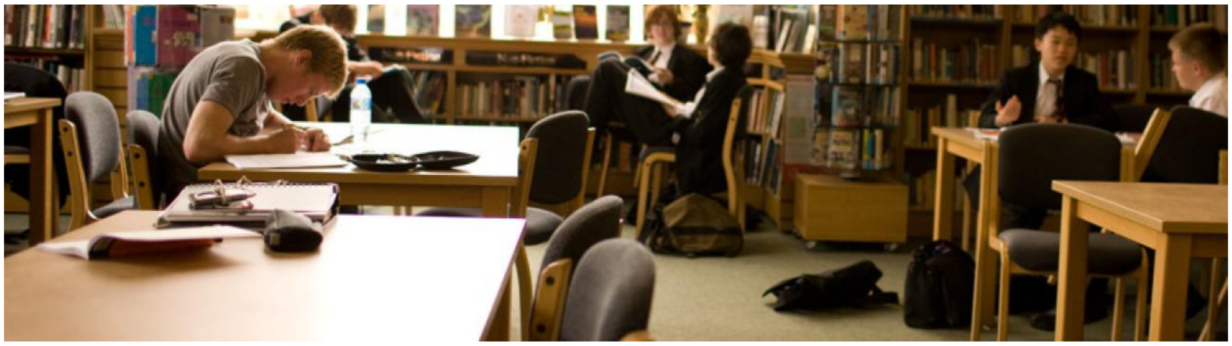
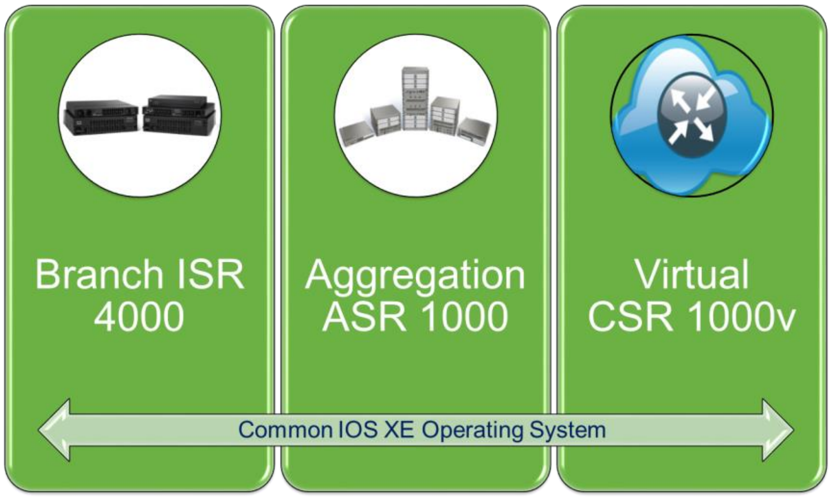
White Paper

IOS XE Virtual Service Development Guide



The Cisco Enterprise routing portfolio consists of multiple products targeted at different segments of the network. Even though these appear to be very different products at first glance, they all share a common architecture and operating system. Cisco IOS XE is the common architecture unifying the 4000 Series Integrated Services Routers, 1000 Series Aggregation Services Routers and the 1000 Series Cloud Services Router.

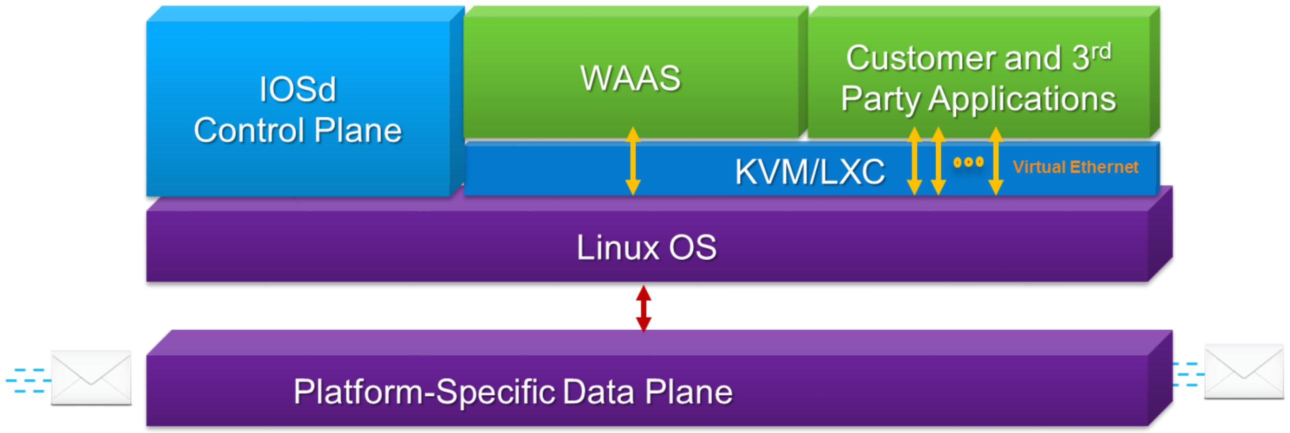
All of these network elements share a common architecture with a platform specific data plane and a common control plane. At their core these are based on an open-source Linux architecture. One common aspect to Linux environments is their ability to host both virtual machines (KVM) and Linux Containers (LXC) guest applications in a protected environment. This capability has always been available for Cisco developed applications in IOS XE platforms. These applications were required to carry a digital signature from Cisco identifying them as genuine before they could be installed.



That all changes after IOS XE release 3.17 in November 2015. With that release Cisco introduces a configuration option allowing customers to bypass the required digital signature when installing new KVM applications[1].

IOS-XE Virtual Service Architecture

IOS-XE routing platforms all share a common software architecture. This means that the architecture and code base is the same for the 4000 Series ISR in the branch, 1000 Series ASR at the headend, and the Cloud Services Router in the cloud.


IOSd Control Plane
WAAS
Customer and 3rd Party Applications
KVM/LXC
Virtual Ethernet
Linux OS
Platform-Specific Data Plane

Features and applications are consistent across all three.

Where these platforms differ is in the Data Plane. Platforms implement IOS-XE Data Plane software in the way that makes the most sense for their deployment. An ASR1K will use a Quantum Flow Processor ASIC, an ISR4K implements the same functionality in an off-the-shelf CPU, and a CSR uses multiple threads in a virtual environment. Packet forwarding and features are consistent across the portfolio, but isn't important for virtual service hosting.

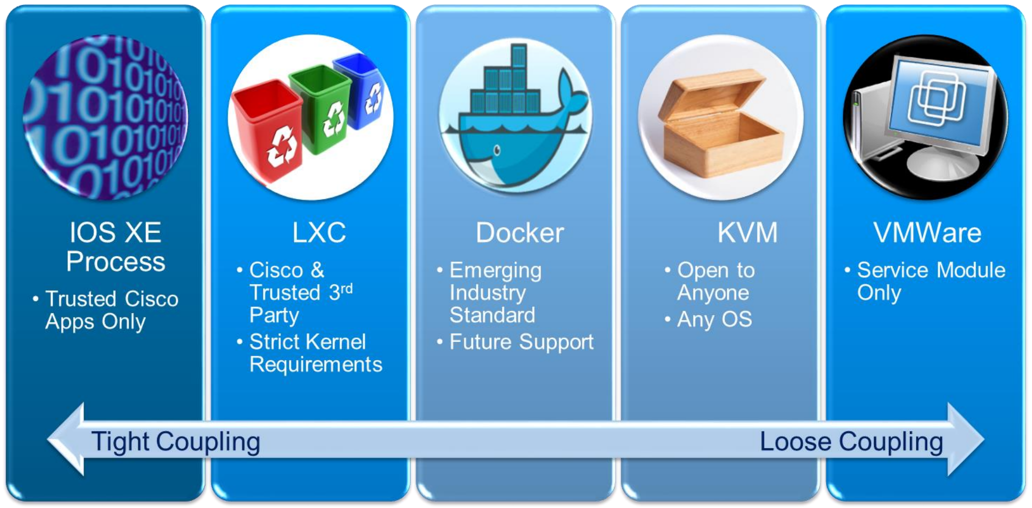
When it comes to virtual service hosting, all IOS-XE platforms share exactly the same architecture. IOS-XE runs a Linux environment outside of the Data Plane. Within this environment reside the control plane processes as well as all other processes necessary for the proper functioning of the box. However, even in the busiest router, it is very rare that the control plane of the system is very active at all. There might be brief periods of activity, such as initially building a large routing table, but for the most part the control plane is idle while the data plane does the work of moving packets through the system.

That leaves a significant amount of CPU time available to do other things. Since IOS-XE is a Linux environment, hosting applications in containers or virtual machines is a very straightforward concept. Using standard open-source tools like vman, libvirt and

[1] At the time of this writing, only KVM applications are allowed without a Cisco digital signature. In the future there may be options for LXC or even Docker container support based on customer interest.

qemu, IOS-XE provides a hosting environment familiar to developers in the Linux world.

APPLICATION HOSTING COMPARISON



Today there exists a spectrum of choices in the application hosting world. These range from very tight coupling and dependencies between the guest application and the host and very loose coupling in the form of virtualization. Many of these are options in networking platforms today including from Cisco.

- Native Process: This option consists of running an application natively within the Linux operating system. Applications can be added with standard package management tools like RPM and YUM. While some platforms do support this option, the potential security concerns and the requirements for the application developer to maintain compatibility with any IOS-XE changes make this unattractive. It is not an option on IOS-XE platforms.

- Linux Containers (LXC): LXC is a lightweight virtualization technology that provides some separation and protection between the guest application and the host system. However, common resources like the kernel and core components are shared between the two. This means that component compatibility must always be maintained between the two. This requirement makes LXC unattractive for application developers even though it does offer some performance advantages over other virtualization options. LXC hosting is available to signed Cisco applications on IOS-XE platforms, but is not available to non-Cisco applications.

- Docker: Docker is an emerging virtualization technology which solves some of the compatibility issues associated with LXC. While Docker containers are not currently supported in IOS-XE, it is being looked at as a possible addition if there is sufficient interest.
- Kernel Virtual Machine (KVM): KVM is the standard virtualization technology within Linux. Guest applications maintain their own kernel and completely separate resources from the host. This separation provides additional levels of security while increasing flexibility for application developers. This is the only hosting option available to non-Cisco applications on IOS-XE platforms.
- VMWare: Technically this category could include other commercial hypervisors such as HyperV or Xen. These are generally heavier weight and targeted at data center applications. These are supported on server blades such as the UCS E-Series, but are generally too heavyweight for hosting applications within IOS-XE platforms. Support for Linux hosts is generally lacking while some also include licensing fees.

Architectural Overview

IOS-XE Virtualization Components

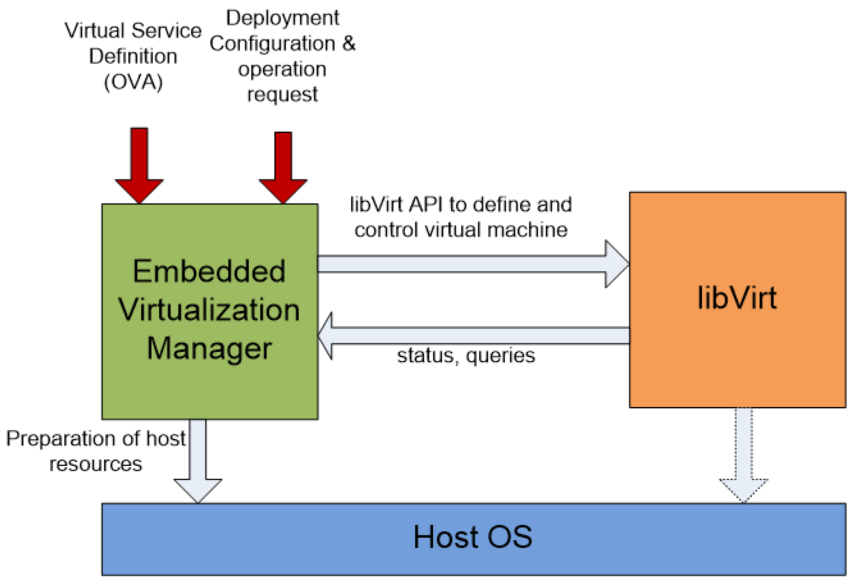

Virtual Service Definition (OVA)
Deployment Configuration & operation request
Embedded Virtualization Manager
libVirt API to define and control virtual machine
libVirt
status, queries
Preparation of host resources
Host OS

Table 1. Platform Capabilities

Platform	Intel X86 Processor	CPU for KVM	System Memory	Memory for KVM	Storage for KVM[2]
ISR4451	Intel Gladden 4 core 2GHz	3 cores (equivalent)	4-16GB	0-12GB	NIM-SSD(200GB, 400GB), NIM-HD(500GB, 1TB)
ISR4431	Intel Gladden 4 core 1GHz	3 cores (equivalent)	4-16GB	0-12GB	NIM-SSD(200GB, 400GB), NIM-HD(500GB, 1TB)
ISR 4351	Intel Rangeley 8 core 2.4GHz	3 cores (equivalent)	4-16GB	0-12GB	MSATA(50GB, 200GB), NIM-SSD, NIM-HD
ISR 4331	Intel Rangeley 8 core 2.0GHz	3 cores (equivalent)	4-16GB	0-12GB	MSATA(50GB, 200GB), NIM-SSD, NIM-HD
ISR 4321	Intel Rangeley 4 core 2.4GHz	1 core (equivalent)	4-12GB	0-8GB	MSATA(50GB, 200GB), NIM-SSD, NIM-HD
ASR 1001-X					
ASR 1002-X					
ASR 1000 RP2					
CSR 1000v	Various				

[2] NIM-SSD, NIM-HD and MSATA capacities subject to change. Refer to Cisco.com for the latest options.

Building Your First Service Container Application

Building applications is hard. Really hard.

Fortunately there's almost no additional work to building a Service Container application beyond what it takes to build the application itself. If you're taking advantage of open-source or previously developed applications built for other KVM deployments building a Service Container application is a simple matter of packaging.

The industry standard package for a virtual machine is known as an Open Virtual Appliance or OVA. In reality OVA is a packaging standard that says nothing about the contents of the package. For those familiar with common Unix tools, an OVA file is actually a TAR archive with all of the files necessary to deploy a virtual machine on a target hypervisor.

Because hypervisors vary greatly, the contents of an OVA file need to be customized to provide exactly what the hypervisor requires. An OVA designed for a VMWare system is going to include different contents than an OVA intended for HyperV or Xen. Even within KVM the requirements for an OVA package can vary from one system to another.

All OVA files will have some common requirements. At a minimum they will need a binary file or files for any disks, hard drives or CD/DVD drives, available to the virtual machine. The OVA will also need to include a description of what the virtual machine hardware configuration looks like. This is often an XML or formatted text file that describes the number of CPUs, amount of DRAM and interfaces such as Ethernet or serial ports available to the virtual machine.

Cisco Service Container OVA packages are relatively simple. There are only a few required files listed in the table below. Most of these are simple text files that can be created in a few minutes. The most complex file is the virtual disk image. This represents the binary read-only (ISO) or read-write (QCOW2 or RAW) disk image for the virtual machine.

For reference, this OVA package is the same format used for Cisco IOX applications. To create the actual OVA file, simply use the Linux tar application, or a compatible packaging program on other operating systems, to generate a TAR archive with the .ova file name extension.

Critical OVA Contents

Category	Description	Usage	Origin
package.yaml	Virtual Machine definition defined in YAML format.	Used by Virtualization Manager to provision the virtual service	Provided by s/w developer for virtual service.
***.mf**	Manifest file that contains SHA1 hash for each file in the OVA	Used by Virtualization Manager to verify the integrity of the files in OVA	Automatically generated by script or created using tools such as openssl.
***.ver**	Simplified compatibility check with Virtualization Infrastructure	Used by Virtualization Manager to perform simple compatibility check.	Simple text file provided by s/w developer.
***.img**	HDD image files (qcow2, raw)	Used to package pre-installed images or pre-allocated empty storage for usage by virtual machine.	Provided by s/w developer.
***.ISO**	ISO image files	Used to pass CDROM images or root file systems	Provided by s/w developer.

Virtual Service Definition File

When a virtual service is installed into an IOS XE system, the host needs to know exactly what the service requires. In the KVM world, this is typically provided by a specially formatted XML file which passes parameters to the libvirt process. This file describes things such as CPU, memory and storage requirements, network interfaces, pointers to disk images and any serial console options. The libvirt process is extremely sensitive to the formatting of this XML file and is constantly evolving the capabilities of this file making the XML file very specific to the version of libvirt used and very sensitive to typos in the XML formatting. This is great for the open source community, but it’s lousy for predictibality when writing applications for wide distribution in network devices.

For that reason, Cisco developed a YAML formated definition file. YAML is a human-friendly standard format that is significantly easier to deal with than libvirt format XML. Behind the scenes, the IOS XE processes responsible for installing an application are actually parsing this YAML file into a libvirt XML file taking care of all of the formatting and context specific to the version used in the system. The YAML format used for IOS XE virtual services is identical to that used for IOX applications as well as future open virtualization services from Cisco.

YAML Virtual Service Definition File

```
manifest-version: <manifest version>

info:
  name: <application name>
  description: <application description string>
  version: <application version>
  author-name: <application author/vendor>
  author-link: <application author/vendor
website>

app:
  apptype: <vm/app type>

  resources:
   cpu: <cpu share %>
   memory: <memory in megs>
   vcpu: <no. of vcpus>

   disk:
    - target dev: <disk name>
      file: <image name>
      upgrade-model: <ha-sync | local>
      share-model: <core>
      capacity: <disk capacity in megs>
    - ...

   interfaces:
    - target-dev: <interface name>
      type: <management>
    - ...

   serial:
    - serial
    - console
    - syslog
    - tracelog

  startup:
    runtime: <kvm>
    boot-dev: <boot device>
```

Description of Fields in the YAML Virtual Service Definition File

manifest-version: The version of the descriptor file which may be different from the version specified in the package version file.

info section

This section contains the fields that define the "application" running in the virtual service. The application a is program or group of programs that are designed for the end user and runs within the virtual service. Current supported version is "1.0".

name: Application name

version: Application version

description: A string describing the application

author-name: Application author name

author-link: Application author website for reference

app section

apptype: Only 'vm' is supported.

resources section

This section contains all the fields that describe the virtual service. Use "show virtual-service" in IOS-XE exec mode to determine the current resource quota on the system.

memory: Amount of RAM in KB allocated to the virtual service. The sum of all memory of active virtual services cannot exceed the amount specified in the quota.

vcpu: Number of vcpus assigned to the virtual service. This field is only valid for KVM. Defaults to 1 if not specified.

cpu: Percent of system CPU share assigned to the virtual service (minimum guarantee). The sum of all cpu shares of active virtual services cannot exceed the amount specified in the quota.

disk section

This section is used to specify the disk resources for the virtual service. Up to three disk devices can be specified. For each disk, either a disk image file or the capacity has to be specified.

target-dev: Name used for the disk device.

file: Disk image file could either be iso, qcow2 or raw format.

capacity: Size of disk to allocate in MB.

upgrade-model: ha-sync - Sync this disk with the standby route processors. This is not supported for iso or boot devices.

local - Does not get synced to the standby route processor.

(defaults to local if upgrade model not specified)

interface section

Two type of interfaces are supported, virtualPortGroup and the management interface. The virtualPortGroup interface will act as the default gateway to the guest's interface. The management interface will need to be in the same subnet as the guest interface, but will not act as the gateway. The order in which these interfaces are defined is also maintained in the guest.

target-dev: Name used for interface device.

alias: Alias name of interface device (optional).

type: The Interface type is set to 'management' to specify that management interface. If not specified, the virtualPortGroup will be used (optional).

serial section

console: Guest ttyS0 will be the console (optional).

aux: Guest ttyS1 will be the aux port (optional).

syslog: Guest ttyS2 will be used for syslog events (optional).

logger: Guest ttyS3 will be used for debug log events (optional).

The guest tty serial port numbers will always remain in sequential order, regardless of which serial port is configured in the YAML descriptor file.

startup section

runtime: Only 'kvm' is currently supported.

boot-dev: Reference to the disk boot device. Can be 'cdrom' or 'hd'. Only one boot device is currently allowed.

Example YAML Virtual Service Definition Files

Bootable harddisk image in OVA, one Virtual Port Group interface and one management interface:

```
# KVM Linux test distribution descriptor file
manifest-version: 1.0

info:
  name: kvm_linux
  description: "KVM Linux Test Distro"
  version: 1.0
  author-name: Cisco Systems, Inc.
  author-link: "http://www.cisco.com"

app:
  apptype: vm

  resources:
   cpu: 6
   memory: 262144
   vcpu: 1

   disk:
    - target-dev: hda
      file: linux.img

   interfaces:
    - target-dev: net1
    - target-dev: net2
      type: management

   serial:
    - console
    - aux
    - syslog
    - tracelog

  startup:
    runtime: kvm
    boot-dev: hd
```

Bootable iso image and harddisk image in OVA, carve additional 2GB harddisk, two VirtualPortGroup interfaces:

```
manifest-version: 1.0

info:
  name: kvm_linux_2
  description: "KVM Linux Test Distro"
  version: 1.0
  author-name: Cisco Systems, Inc.
  author-link: "http://www.cisco.com"

app:
  # Indicate app type
  apptype: vm

  resources:
   cpu: 6
   memory: 262144
   vcpu: 1

   disk:
    - target dev: hdc
      file: linux.iso
    - target dev: sda
      file: kvm_storage_4000MB.img
      upgrade-model: ha-sync
    - target dev: sdb
      capacity: 2000

   interfaces:
    - target-dev: net1
    - target-dev: net2

   serial:
    - serial
    - console
    - syslog
    - tracelog

  # Specify runtime and startup
  startup:
    runtime: kvm
    boot-dev: cdrom
```

Developer Workflow

Working with Virtual Services is almost identical to working with KVM virtual machines. There are just a few additional components needed in the packaging process. The development workflow and common development tools used for KVM can still be used when developing, or modifying existing applications, for the Virtual Service Environment.

Several open source tools exist that make the final packaging process easier. If this is your first time developing a KVM application, these will come in very handy. If you have experience working in the VMWare world, some of these concepts will be familiar but different when moving to a Linux environment. There is no requirement to use these tools when developing Virtual Services for IOS XE platforms. They're mentioned here simply to give new developers a head start.

Tool	Uses	Example
virt-manager	GUI Linux VM Manager for crafting KVMs.	GUI
qemu-img	Converting disk image formats.	qemu-img convert -p -c -f raw -O qcow2 <raw.img> <qcow2.img>
openssl	Generates the manifest file.	qemu-img convert -p -c -f raw -O qcow2 <raw.img> <qcow2.img>
tar	Packages the files into an OVA.	tar -cvf VM.ova vm.qcow2 *.yaml vm.mf
create_ova.sh	Cisco script for packing an OVA in one step.	create_ova.sh [<options>] <directory>

Working with the Cisco Packaging Script

Cisco provides a simple script (create_ova.sh) to help developers with packaging a virtual service compatible OVA. Use of this script is entirely optional and will generally be more useful for high-volume developers. This script takes the raw files for the application, generates the manifest file and builds the OVA tar package.

Usage:

```
create_ova.sh [<options>] <directory>
```

Options:

- -mts or -max_total_size - (default 600) Specify the maximum directory size before compression is considered.

- -mfs or -max_file_size - (default 10) If max_total_size is exceeded, compress each file larger than this threshold.

  Together -mts and -mfs provide a heuristic to calculate whether or not to compress large files. If total directory size exceeds -max_total_size, compress files greater than -max_file_size. Specifying '-max_total_size 0' will force compression on all files greater than -max_file_size Setting -max_total_size to a very high value will avoid compression

  Examples:

- **create_ova.sh Test** - Create unsigned OVA package, compress files if necessary using Directory 'Test'
- **create_ova.sh -mts 500 -mfs 20 Test** - Create unsigned OVA package, compress files greater than 20M if total directory size of 'Test' is greater than 500M

  The OVA package will be created inside the specified directory. Example procedure for creating OVA package shown below:

```
mkdir test
cp linux.img test
echo 1.7 > test/test.ver
# Note: edit test/package.yaml and add required yaml parameters for virtual se
create_ova.sh test
```

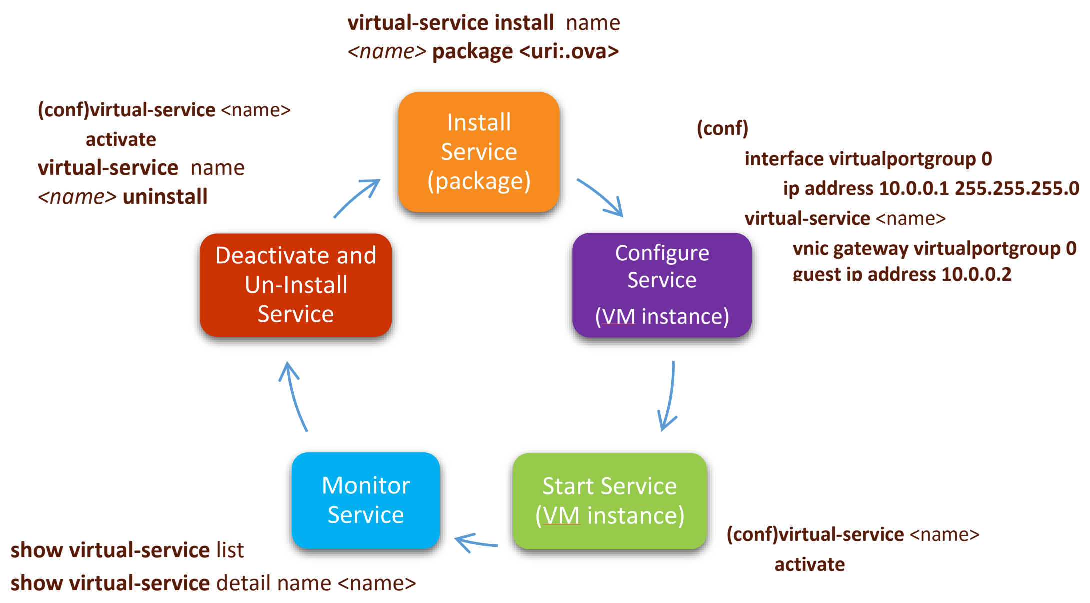
If an existing ova is untared and repackaged, be sure none of the files are compressed before running the packing script or the resulting manifest file will prevent the ova package from being used on the router.

Service Container Deployment

Phase	Trigger	Actions	Virtual Service Instance State
Pre-Installation		1. Gather and prepare system resources 2. Establish internal communication infrastructures.	✓ Host is ready to accept new virtual service.
Installation	VMan received a request to install a virtual service package.	1. Unzip and unpack the virtual service definition from its OVA package. 2. For signed application, perform SHA2 code signing check using the artifacts in the OVA (.cert, .mf) and a hidden Cisco public key. 3. Validates the machine definition specified in the OVA and performs preliminary resource check (for warnings). 4. Parses the machine definition and creates internal objects for manageability. 5. Process tiered resource profiles requests.	✓ Validated that package is Cisco signed. ✓ Validated integrity of OVA content. ✓ Validated and parsed machine definition and binds it to a virtual service “instance name”
Configuration	VMan received a request to configure instance of the virtual service.	1. Perform validation and necessary network provisioning for configured guest IP address (if applicable) 2. Perform resource check and reservation for selected profile.	✓ Virtual service is configured
Activation	VMan received a request to activate	1. Carves our storage resource from host system as per need. 2. Commit CPU, Memory, Storage and Networking resources as needed. 3. Update the machine definition XML and request libvirt to start virtual machine. 4. Service to console, aux, logging and tracing ports as needed.	✓ Virtual service is activated.
Post Activation		1. Perform monitoring services 2. Process lifecycle control services	

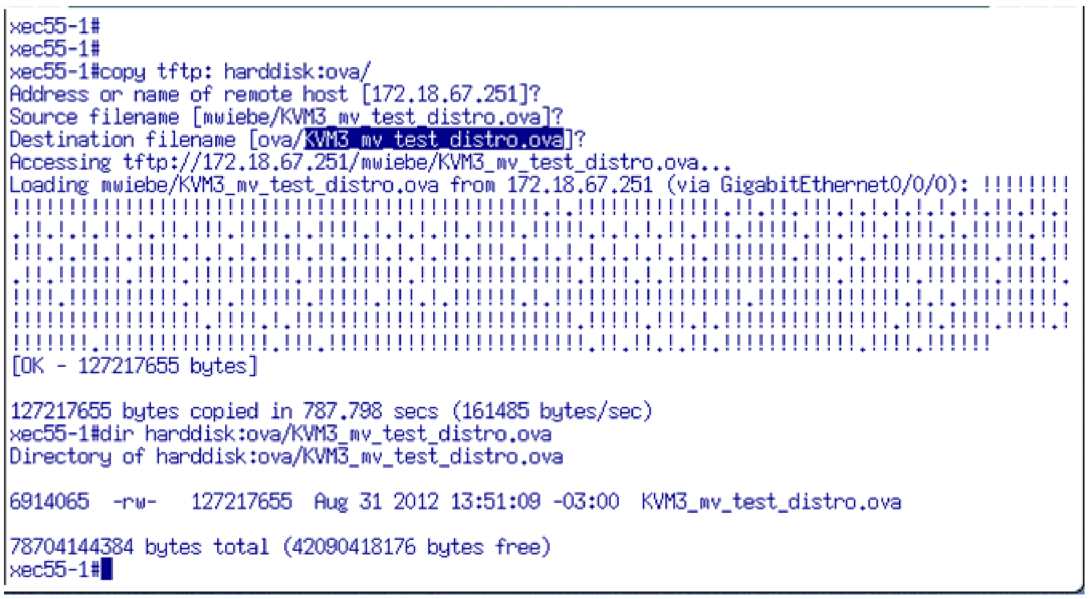
Deploying a KVM Virtual Service in an IOS-XE Router

In this document we’re going to cover the end-user interactions for installing and managing virtual services using the command line interface. There are other graphical management tools available today and in the future that will take advantage of APIs provided by the system. These are generally outside the scope of this document targeted at developers. Just be aware that options such as Prime Infrastructure and Cisco Fog Director are options for dealing with virtual services in addition to the command line.

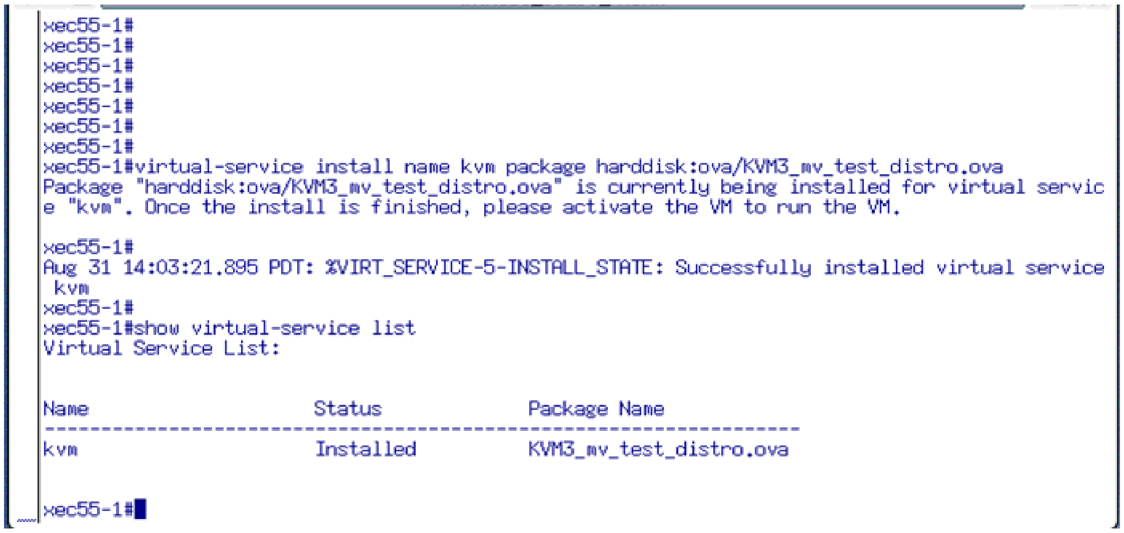

virtual-service install name <name> package <uri:.ova>
Install Service (package)
(conf)
interface virtualportgroup 0
ip address 10.0.0.1 255.255.255.0
virtual-service <name>
vnic gateway virtualportgroup 0
guest ip address 10.0.0.2
Configure Service (VM instance)
Start Service (VM instance)
(conf)virtual-service <name>
activate
Monitor Service
show virtual-service list
show virtual-service detail name <name>
Deactivate and Un-Install Service
(conf)virtual-service <name>
activate
virtual-service name <name> uninstall

This section uses an example KVM virtual machine to demonstrates bringing up of a generic guest OS within the host OS.

<u>Step #1: Copy the package to target</u>

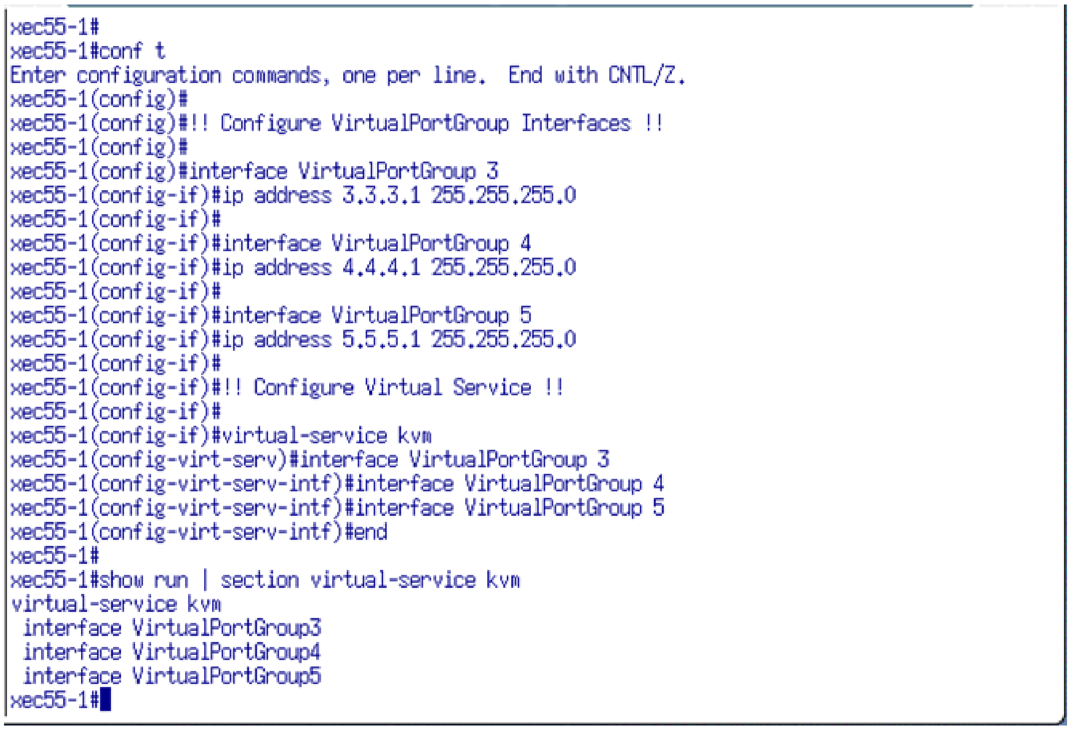

```
xec55-1#
xec55-1#
xec55-1#copy tftp: harddisk:ova/
Address or name of remote host [172.18.67.251]?
Source filename [mwiebe/KVM3_mv_test_distro.ova]?
Destination filename [ova/KVM3_mv_test_distro.ova]?
Accessing tftp://172.18.67.251/mwiebe/KVM3_mv_test_distro.ova...
Loading mwiebe/KVM3_mv_test_distro.ova from 172.18.67.251 (via GigabitEthernet0/0/0): !!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!.!.!!!!!!!!!!!!!!.!!.!!.!!!.!.!.!.!.!.!!.!!.!!.!
.!!.!.!.!!.!.!!.!!!.!!!!.!.!!!!.!.!.!.!!.!!!!.!!!!!.!.!.!.!!.!!!.!!!!!.!!.!!!.!!!!.!.!!!!!.!!!
!!!.!.!!.!.!!!!.!.!.!.!!!!.!!.!!!.!.!.!!.!!!.!.!.!.!.!.!.!.!!.!!!!!!!.!!!.!.!!!!!!!!!!!.!!!.!!
.!!.!!!!!!.!!!!.!!!!!!!!!.!!!!!!!!.!!!!!!!!!!!!!!.!!!!.!.!!!!!!!!!!!!!!!.!!!!!!.!!!!!!.!!!!!.
!!!!.!!!!!!!!!!.!!!.!!!!!!.!!!!!.!!!.!.!!!!!!.!.!!!!!!!!!!!!!!!!.!!!!!!!!!!!!!.!.!.!!!!!!!!!.
!!!!!!!!!!!!!!!!.!!!!.!.!!!!!!!!!!!!!!!!!!!!!!!!.!!!!!.!!!.!.!!!!!!!!!!!!!!!.!!!.!!!!.!!!!.!
!!!!!!!.!!!!!!!!!!!!!!!.!!!.!!!!!!!!!!!!!!!!!!!!!!.!!.!!.!.!!.!!!!!!!!!!!!.!!!!.!!!!!!
[OK - 127217655 bytes]

127217655 bytes copied in 787.798 secs (161485 bytes/sec)
xec55-1#dir harddisk:ova/KVM3_mv_test_distro.ova
Directory of harddisk:ova/KVM3_mv_test_distro.ova

6914065  -rw-   127217655  Aug 31 2012 13:51:09 -03:00  KVM3_mv_test_distro.ova

78704144384 bytes total (42090418176 bytes free)
xec55-1#
```

<u>Step #2 : Install the package</u>



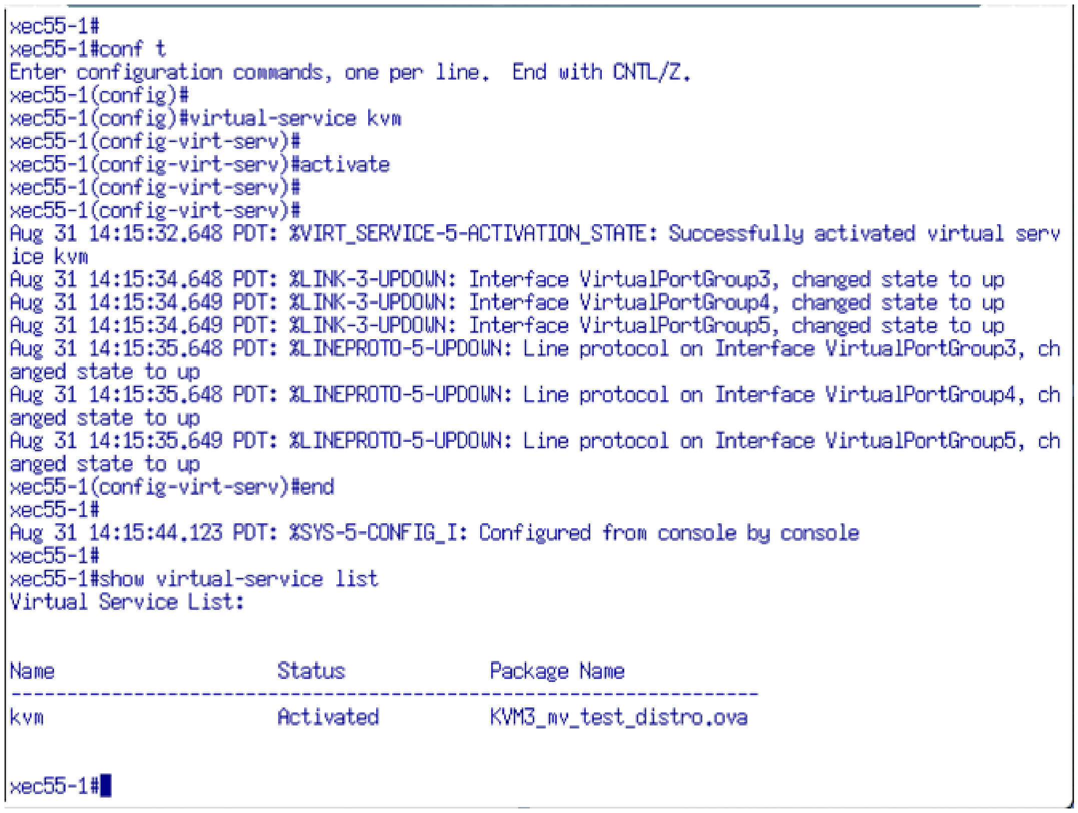
```
xec55-1#
xec55-1#
xec55-1#
xec55-1#
xec55-1#
xec55-1#
xec55-1#
xec55-1#virtual-service install name kvm package harddisk:ova/KVM3_mv_test_distro.ova
Package "harddisk:ova/KVM3_mv_test_distro.ova" is currently being installed for virtual servic
e "kvm". Once the install is finished, please activate the VM to run the VM.

xec55-1#
Aug 31 14:03:21.895 PDT: %VIRT_SERVICE-5-INSTALL_STATE: Successfully installed virtual service
 kvm
xec55-1#
xec55-1#show virtual-service list
Virtual Service List:


Name                     Status             Package Name
-------------------------------------------------------------------
kvm                      Installed          KVM3_mv_test_distro.ova


xec55-1#
```

Step #3 : Configure the service



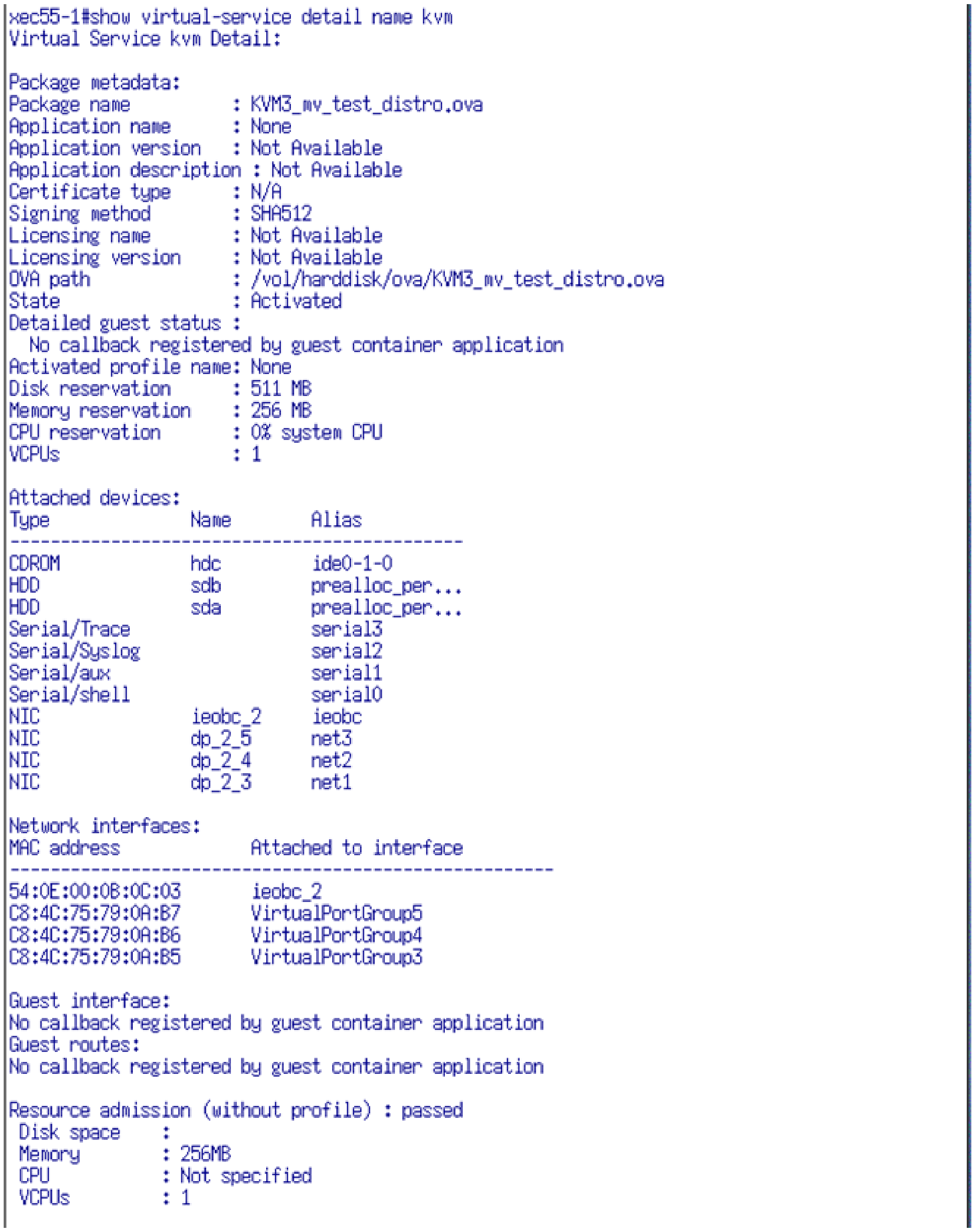
```
xec55-1#
xec55-1#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
xec55-1(config)#
xec55-1(config)#!! Configure VirtualPortGroup Interfaces !!
xec55-1(config)#
xec55-1(config)#interface VirtualPortGroup 3
xec55-1(config-if)#ip address 3.3.3.1 255.255.255.0
xec55-1(config-if)#
xec55-1(config-if)#interface VirtualPortGroup 4
xec55-1(config-if)#ip address 4.4.4.1 255.255.255.0
xec55-1(config-if)#
xec55-1(config-if)#interface VirtualPortGroup 5
xec55-1(config-if)#ip address 5.5.5.1 255.255.255.0
xec55-1(config-if)#
xec55-1(config-if)#!! Configure Virtual Service !!
xec55-1(config-if)#
xec55-1(config-if)#virtual-service kvm
xec55-1(config-virt-serv)#interface VirtualPortGroup 3
xec55-1(config-virt-serv-intf)#interface VirtualPortGroup 4
xec55-1(config-virt-serv-intf)#interface VirtualPortGroup 5
xec55-1(config-virt-serv-intf)#end
xec55-1#
xec55-1#show run | section virtual-service kvm
virtual-service kvm
 interface VirtualPortGroup3
 interface VirtualPortGroup4
 interface VirtualPortGroup5
xec55-1#
```

Step #4: Start the service

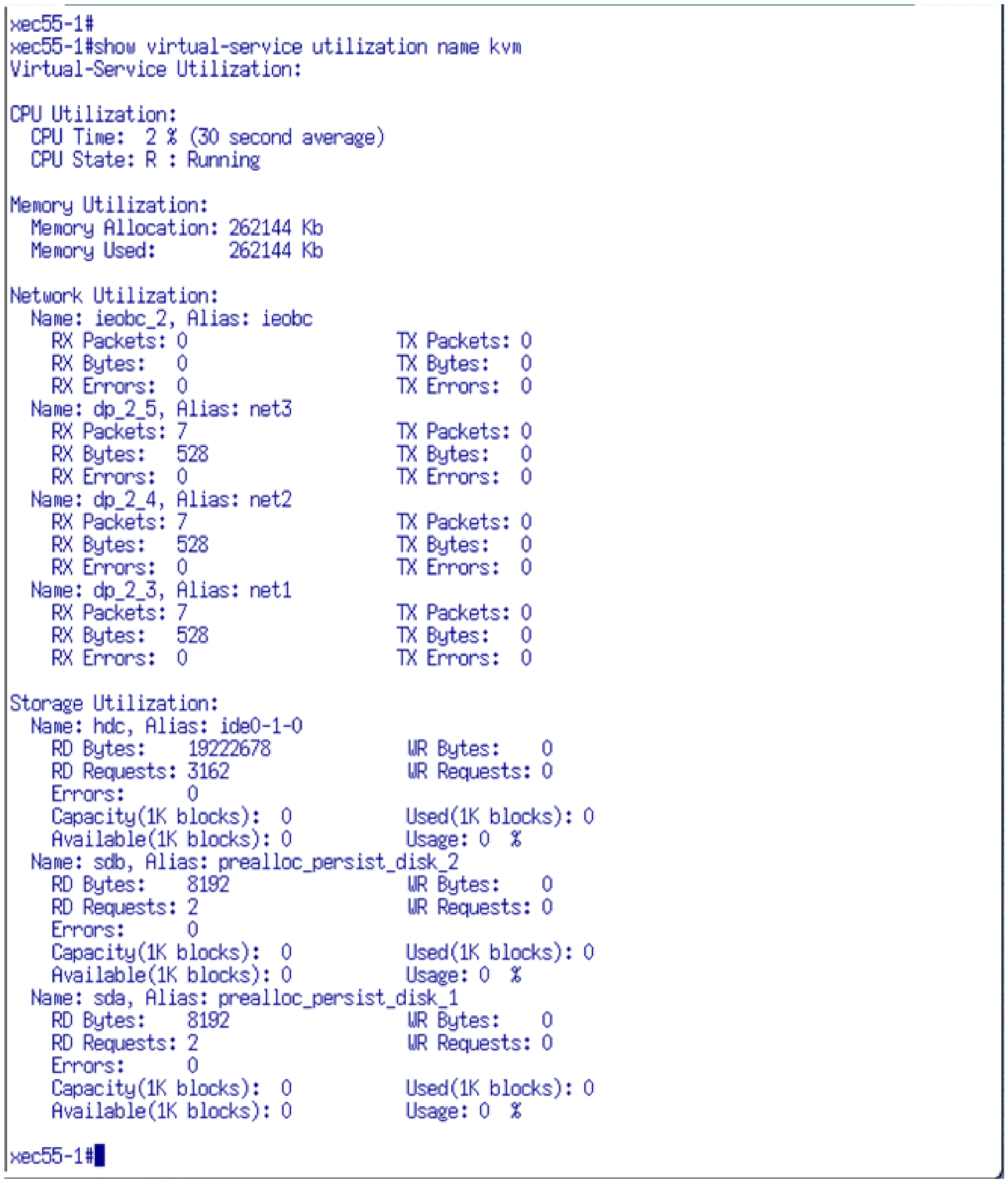


```
xec55-1#
xec55-1#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
xec55-1(config)#
xec55-1(config)#virtual-service kvm
xec55-1(config-virt-serv)#
xec55-1(config-virt-serv)#activate
xec55-1(config-virt-serv)#
xec55-1(config-virt-serv)#
Aug 31 14:15:32.648 PDT: %VIRT_SERVICE-5-ACTIVATION_STATE: Successfully activated virtual serv
ice kvm
Aug 31 14:15:34.648 PDT: %LINK-3-UPDOWN: Interface VirtualPortGroup3, changed state to up
Aug 31 14:15:34.649 PDT: %LINK-3-UPDOWN: Interface VirtualPortGroup4, changed state to up
Aug 31 14:15:34.649 PDT: %LINK-3-UPDOWN: Interface VirtualPortGroup5, changed state to up
Aug 31 14:15:35.648 PDT: %LINEPROTO-5-UPDOWN: Line protocol on Interface VirtualPortGroup3, ch
anged state to up
Aug 31 14:15:35.648 PDT: %LINEPROTO-5-UPDOWN: Line protocol on Interface VirtualPortGroup4, ch
anged state to up
Aug 31 14:15:35.649 PDT: %LINEPROTO-5-UPDOWN: Line protocol on Interface VirtualPortGroup5, ch
anged state to up
xec55-1(config-virt-serv)#end
xec55-1#
Aug 31 14:15:44.123 PDT: %SYS-5-CONFIG_I: Configured from console by console
xec55-1#
xec55-1#show virtual-service list
Virtual Service List:


Name                   Status            Package Name
------------------------------------------------------------------
kvm                    Activated         KVM3_mv_test_distro.ova


xec55-1#
```

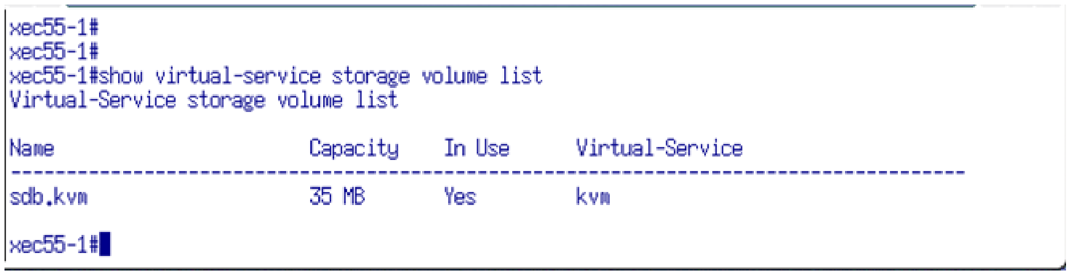
Step #5: Monitor the service



```
xec55-1#show virtual-service detail name kvm
Virtual Service kvm Detail:

Package metadata:
Package name           : KVM3_mv_test_distro.ova
Application name       : None
Application version    : Not Available
Application description : Not Available
Certificate type       : N/A
Signing method         : SHA512
Licensing name         : Not Available
Licensing version      : Not Available
OVA path               : /vol/harddisk/ova/KVM3_mv_test_distro.ova
State                  : Activated
Detailed guest status :
  No callback registered by guest container application
Activated profile name: None
Disk reservation       : 511 MB
Memory reservation     : 256 MB
CPU reservation        : 0% system CPU
VCPUs                  : 1

Attached devices:
Type                 Name        Alias
---------------------------------------------
CDROM                hdc         ide0-1-0
HDD                  sdb         prealloc_per...
HDD                  sda         prealloc_per...
Serial/Trace                     serial3
Serial/Syslog                    serial2
Serial/aux                       serial1
Serial/shell                     serial0
NIC                  ieobc_2     ieobc
NIC                  dp_2_5      net3
NIC                  dp_2_4      net2
NIC                  dp_2_3      net1

Network interfaces:
MAC address            Attached to interface
------------------------------------------------------
54:0E:00:0B:0C:03      ieobc_2
C8:4C:75:79:0A:B7      VirtualPortGroup5
C8:4C:75:79:0A:B6      VirtualPortGroup4
C8:4C:75:79:0A:B5      VirtualPortGroup3

Guest interface:
No callback registered by guest container application
Guest routes:
No callback registered by guest container application

Resource admission (without profile) : passed
 Disk space     :
 Memory         : 256MB
 CPU            : Not specified
 VCPUs          : 1
```

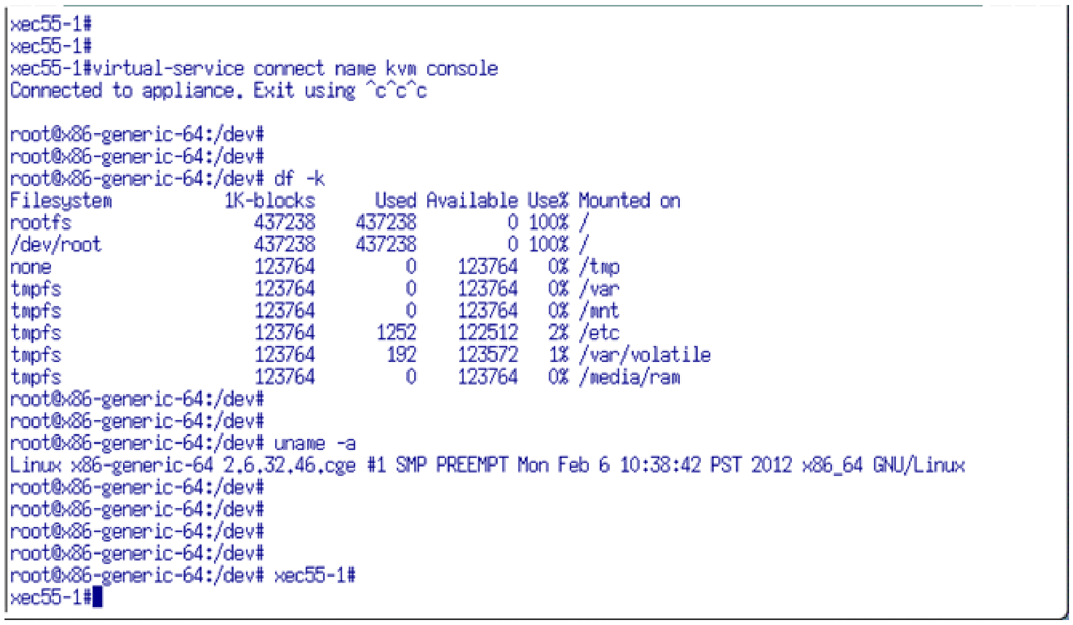


```
xec55-1#
xec55-1#show virtual-service utilization name kvm
Virtual-Service Utilization:

CPU Utilization:
  CPU Time:  2 % (30 second average)
  CPU State: R : Running

Memory Utilization:
  Memory Allocation: 262144 Kb
  Memory Used:       262144 Kb

Network Utilization:
  Name: ieobc_2, Alias: ieobc
    RX Packets: 0                  TX Packets: 0
    RX Bytes:   0                  TX Bytes:   0
    RX Errors:  0                  TX Errors:  0
  Name: dp_2_5, Alias: net3
    RX Packets: 7                  TX Packets: 0
    RX Bytes:   528                TX Bytes:   0
    RX Errors:  0                  TX Errors:  0
  Name: dp_2_4, Alias: net2
    RX Packets: 7                  TX Packets: 0
    RX Bytes:   528                TX Bytes:   0
    RX Errors:  0                  TX Errors:  0
  Name: dp_2_3, Alias: net1
    RX Packets: 7                  TX Packets: 0
    RX Bytes:   528                TX Bytes:   0
    RX Errors:  0                  TX Errors:  0

Storage Utilization:
  Name: hdc, Alias: ide0-1-0
    RD Bytes:     19222678          WR Bytes:     0
    RD Requests: 3162               WR Requests: 0
    Errors:       0
    Capacity(1K blocks):  0         Used(1K blocks): 0
    Available(1K blocks): 0         Usage: 0  %
  Name: sdb, Alias: prealloc_persist_disk_2
    RD Bytes:     8192              WR Bytes:     0
    RD Requests: 2                  WR Requests: 0
    Errors:       0
    Capacity(1K blocks):  0         Used(1K blocks): 0
    Available(1K blocks): 0         Usage: 0  %
  Name: sda, Alias: prealloc_persist_disk_1
    RD Bytes:     8192              WR Bytes:     0
    RD Requests: 2                  WR Requests: 0
    Errors:       0
    Capacity(1K blocks):  0         Used(1K blocks): 0
    Available(1K blocks): 0         Usage: 0  %

xec55-1#
```



```
xec55-1#
xec55-1#
xec55-1#show virtual-service storage volume list
Virtual-Service storage volume list

Name                      Capacity    In Use      Virtual-Service
-------------------------------------------------------------------------------------
sdb.kvm                   35 MB       Yes         kvm

xec55-1#
```

Step #6: Console into the service



```
xec55-1#
xec55-1#
xec55-1#virtual-service connect name kvm console
Connected to appliance. Exit using ^c^c^c

root@x86-generic-64:/dev#
root@x86-generic-64:/dev#
root@x86-generic-64:/dev# df -k
Filesystem           1K-blocks      Used Available Use% Mounted on
rootfs                  437238    437238         0 100% /
/dev/root               437238    437238         0 100% /
none                    123764         0    123764   0% /tmp
tmpfs                   123764         0    123764   0% /var
tmpfs                   123764         0    123764   0% /mnt
tmpfs                   123764      1252    122512   2% /etc
tmpfs                   123764       192    123572   1% /var/volatile
tmpfs                   123764         0    123764   0% /media/ram
root@x86-generic-64:/dev#
root@x86-generic-64:/dev#
root@x86-generic-64:/dev# uname -a
Linux x86-generic-64 2.6.32.46.cge #1 SMP PREEMPT Mon Feb 6 10:38:42 PST 2012 x86_64 GNU/Linux
root@x86-generic-64:/dev#
root@x86-generic-64:/dev#
root@x86-generic-64:/dev#
root@x86-generic-64:/dev#
root@x86-generic-64:/dev# xec55-1#
xec55-1#
```

For More Information

Read more about the Cisco IP Phone 7905G, or contact your local account representative.

Read more about the Cisco End-of-Life Policy.

Subscribe to receive end-of-life/end-of-sale information.