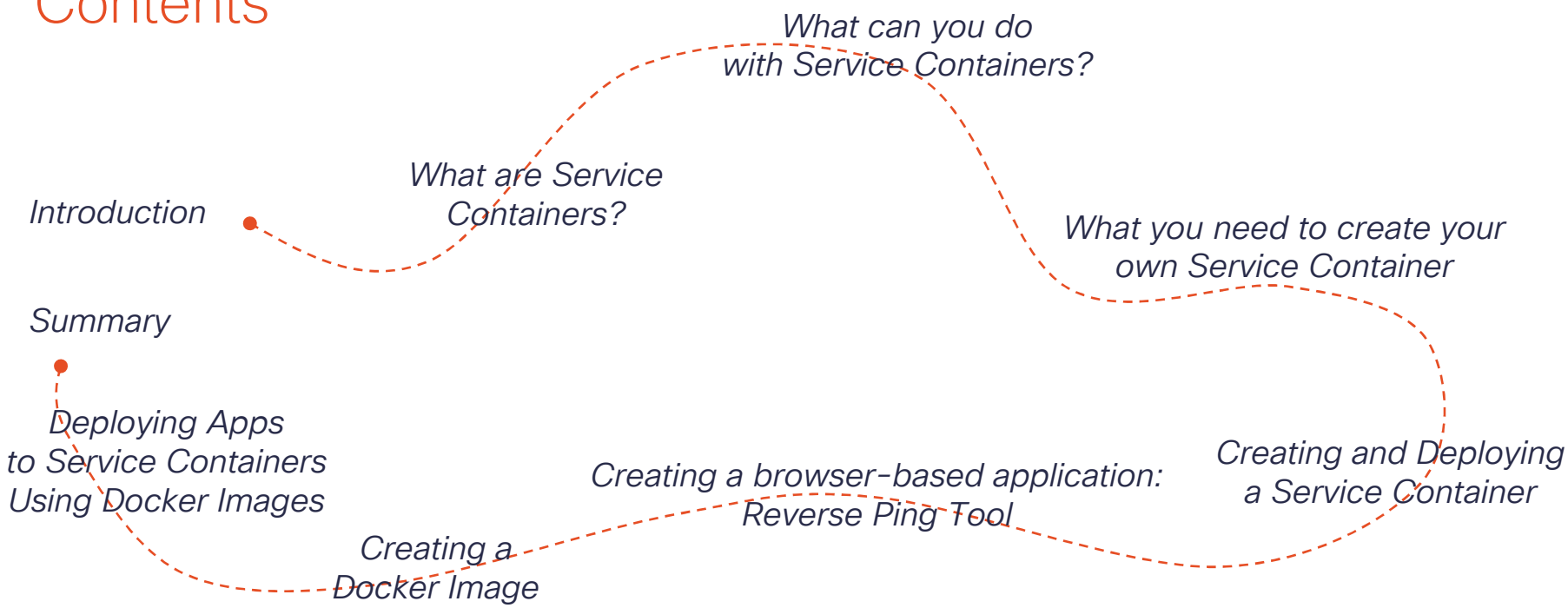
Service Containers Tutorial

Shabaz Yousaf
Cisco

April 2016

Contents



Appendix: Service Container Troubleshooting Guide, Linux Quick Reference

How to Use this Guide

Service Containers Overview

One-hour overview about Service Containers, Benefits, and Packet Flow

Learn How to Create and Deploy Service Containers

55GB of disk space and 16GB or more RAM is needed for your development environment

One full day to complete the tutorial

Learn How to Create Applications and Deploy using Docker

Use a pre-existing Ubuntu Services Container .ova file if you wish to save time and not create it

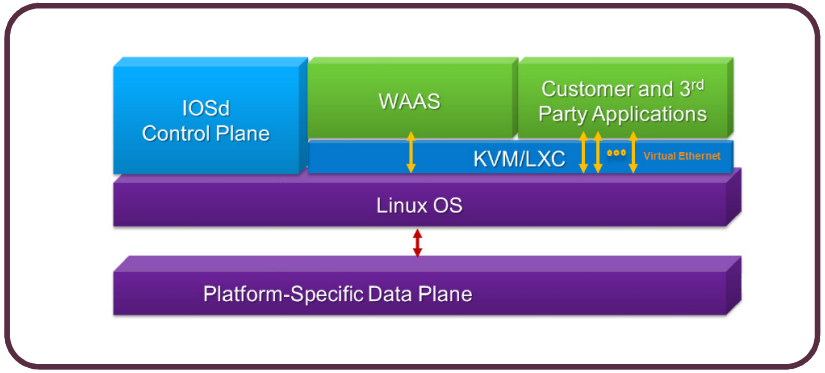
Applications can be directly written inside the Services Container if you don't have sufficient disk space and RAM to do it in a separate development environment

Half a day to complete the tutorial

What are Service Containers?

Brief Summary

- All ISR 4000 series routers have spare CPU cores
- Full applications can run inside the router, inside virtual machines (VMs) with no impact to router performance
- Apps can be Cisco provided, partner or customer provided. They come as .ova files ready to be installed on the router
- The technology is called Cisco Service Containers



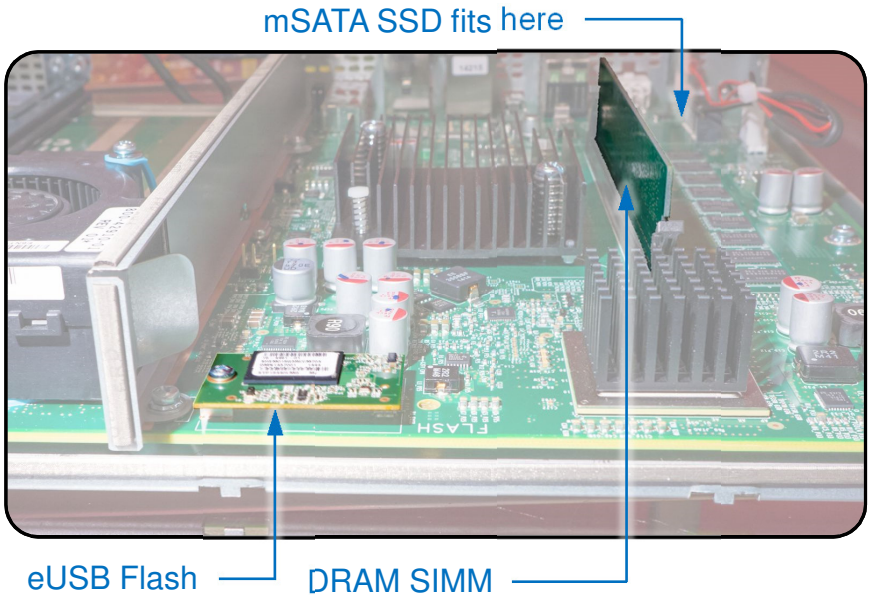
ISR 4000 Series

	4321	4331	4351	4431	4451
Scenario	SOHO	Small branch	Small branch with high compute requirements or expandability	Large branch	Large branch with high compute requirements or expandability
Size	Desktop	1 RU	2 RU	1 RU	2 RU
Built-in routed interfaces	2	3	3	4	4
NIM slots	2	2	3	3	3
SM-X slot capability	-	Single width	Double width (or 2 x single width)	-	Double width (or 2 x single width)
Throughput	100Mbps	300Mbps	400Mbps	1Gbps	2Gbps
Order Code	ISR4321-AX/K9	ISR4331-AX/K9	ISR4351-AX/K9	ISR4431-AX/K9	ISR4451-X-AX/K9

Router Requirements

- All 4000 series are supported
- DRAM and Flash
  - Minimum 8GB DRAM
  - Flash must be greater than DRAM
- Storage options
  - NIM-SSD is supported on all 4000 series
  - NIM-HD is supported but SSD is recommended
  - mSATA SSD is internally supported on all 4300 series

ISR 4321 Router



Virtualization and Container Technologies

KVM

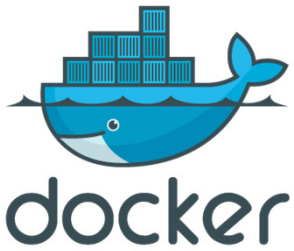
- Full Type 2 Hypervisor
- Can run any guest OS; therefore offers the most flexibility
- Performance is good (uses Intel VT-x)
- KVM is supported on the ISR 4000 routers for use by customers and partners

LXC

- Container for files; not a hypervisor
- Router Linux kernel is shared with apps; therefore considered less secure if unsigned apps were allowed to be run
- Very high performance
- LXC is only available for Cisco signed apps; not for customer apps

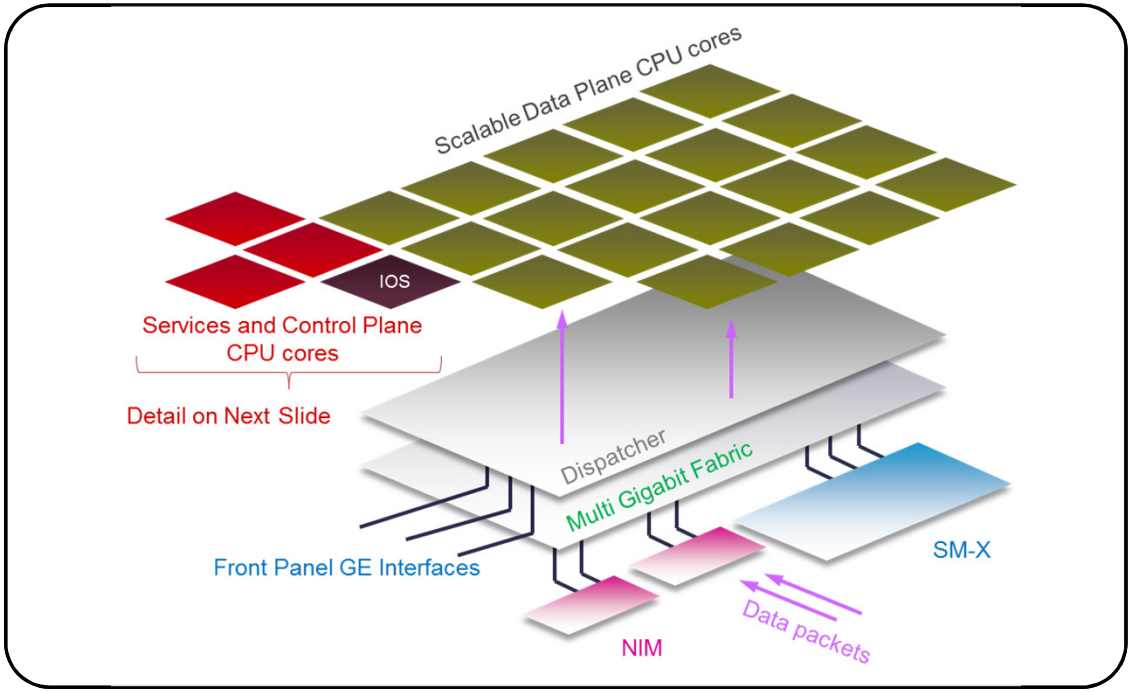
Docker

- Container for files; not a hypervisor
- Applications (Docker Images) can be freely pushed onto Docker Hub or on a private server
- Very high performance
- Easy to download and run apps that are in Docker Images
- Not natively supported today (but can be run inside a KVM VM)

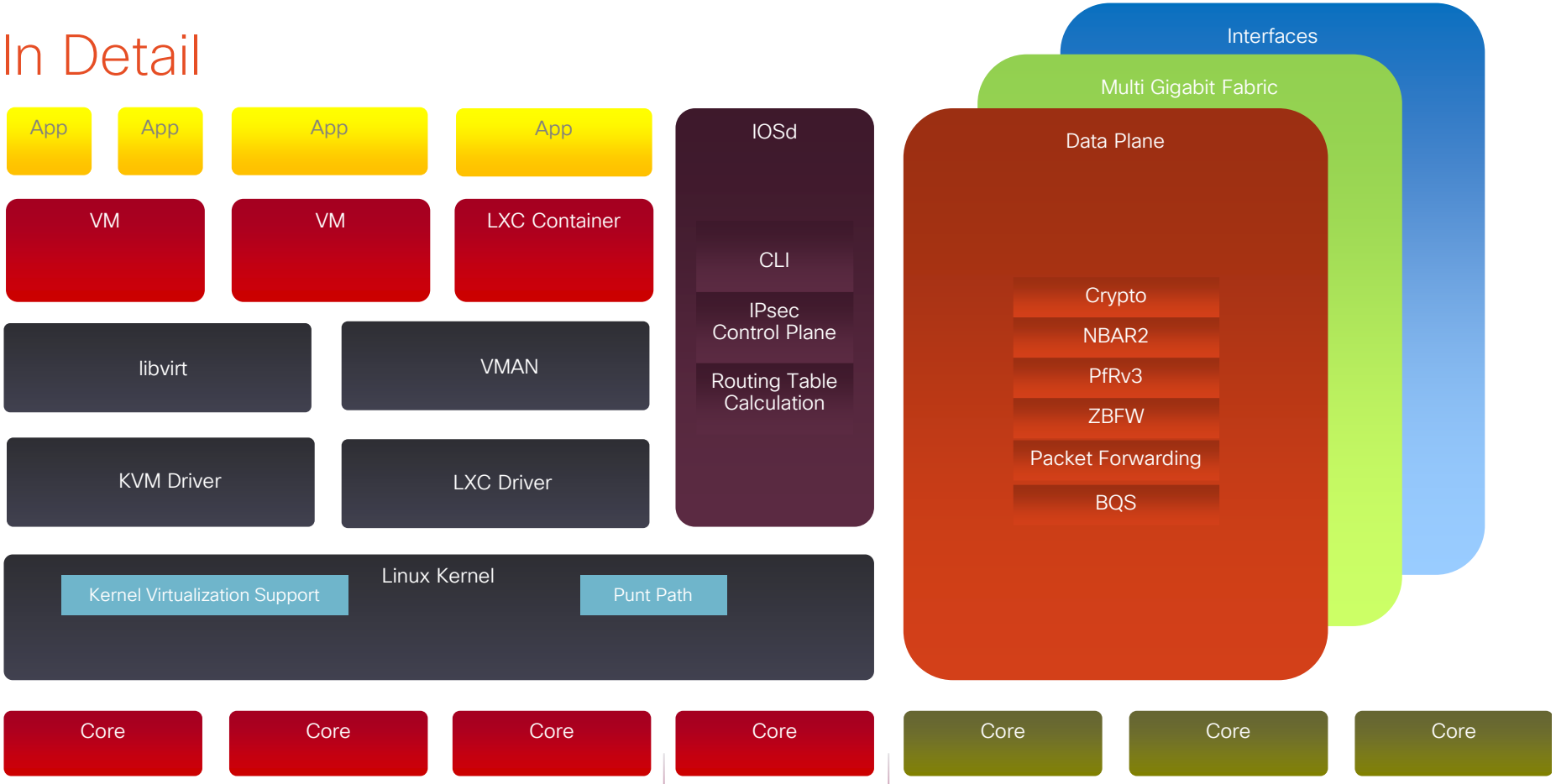


ISR 4000 Series Architecture

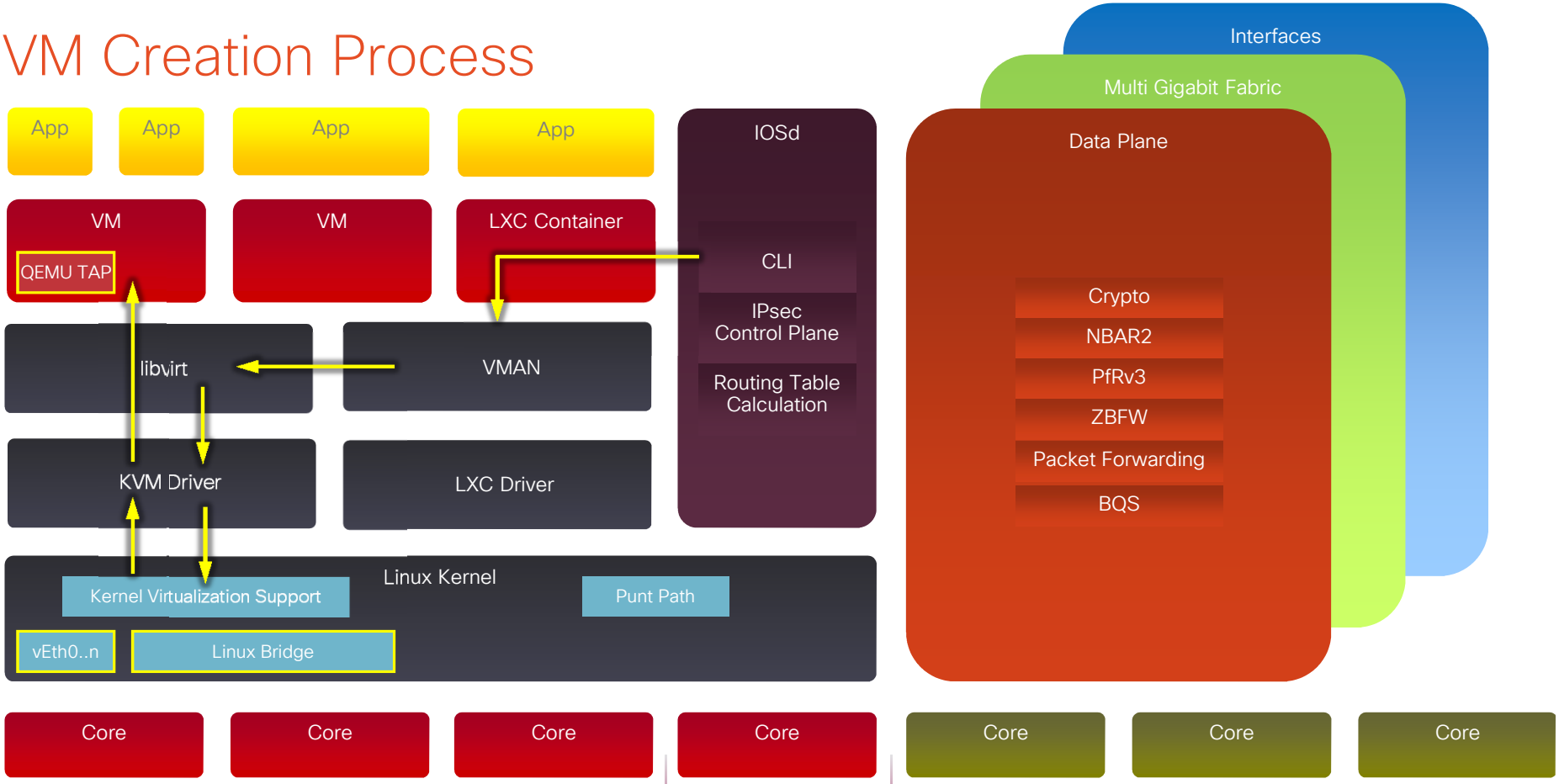
- Multi-core architecture
- IOS-XE: Separated Control and Data Plane
- 1-3 Cores are Service Container Ready



In Detail

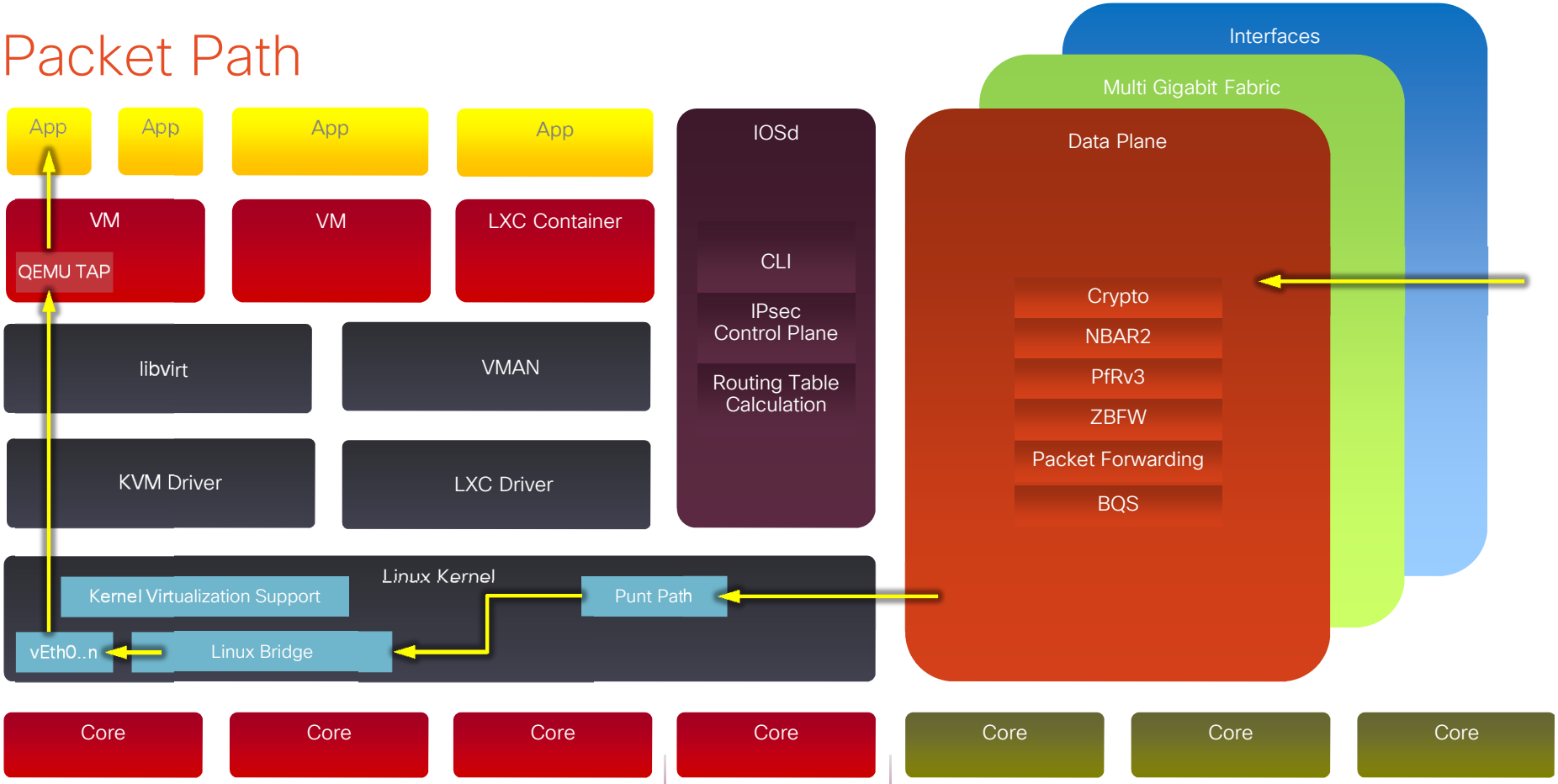


VM Creation Process

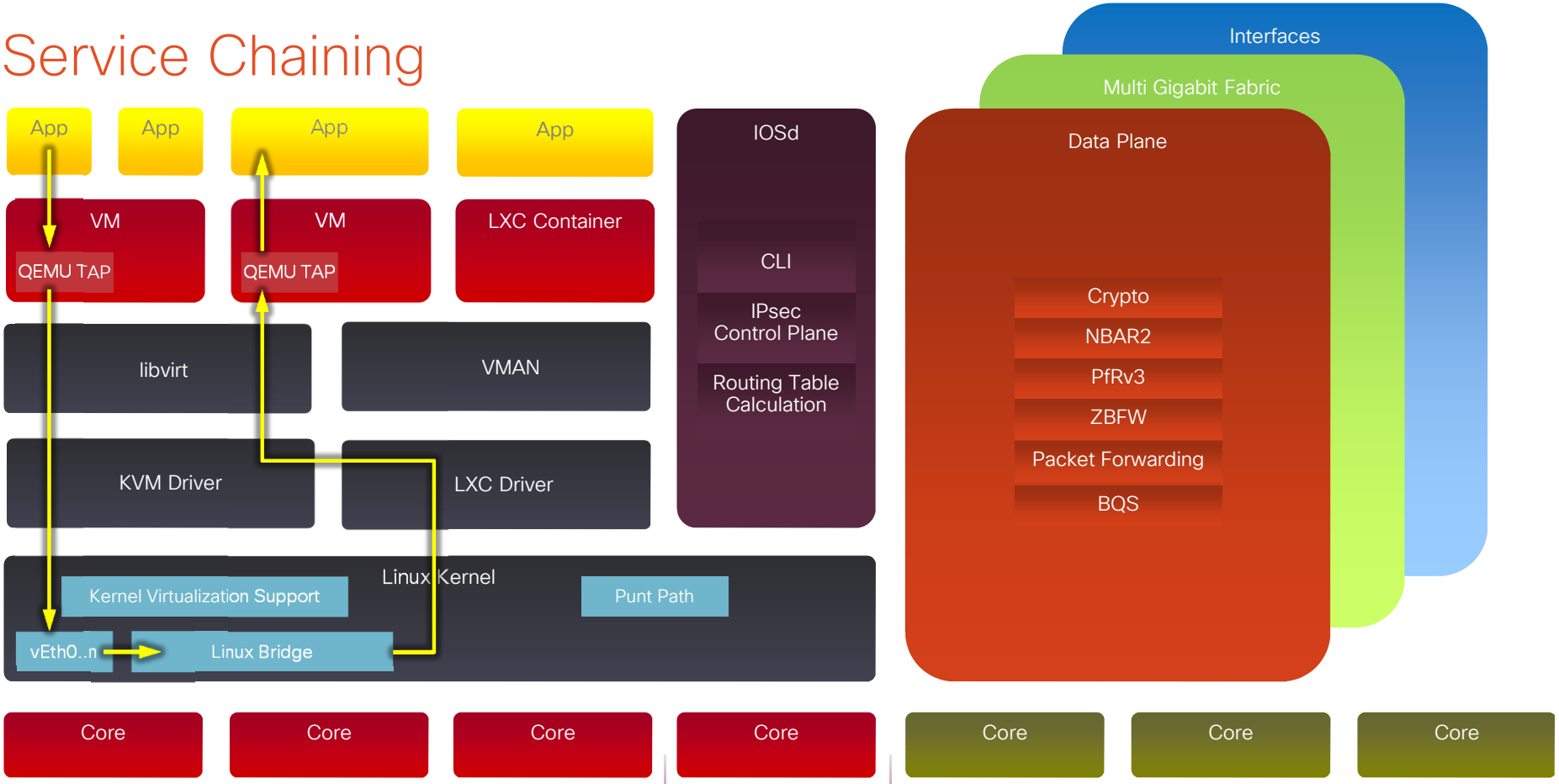


Services Plane | Control Plane | Data Plane

Packet Path



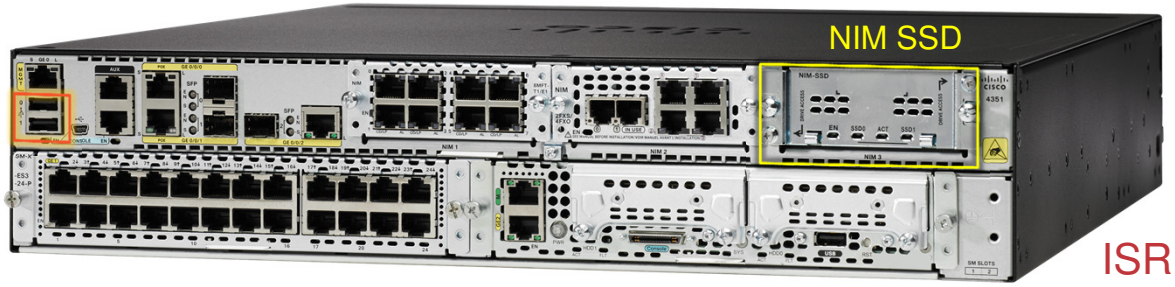
Service Chaining



Brief Install Overview – will be covered in detail later

Install IOS-XE 3.17 or later

Copy the .ova file (e.g. service_container.ova file to the hard disk; a quick way is to copy to a USB memory stick first and plug it into the router



ISR 4351 Router

Use the virtual-service install CLI to install the container
Configure IP addresses and DHCP if desired for the VM
Configure NAT if desired for traffic between the VM and the outside world
Configure the virtual-service
Activate the virtual-service
Done!

Features and Benefits

Feature	Benefit
Standards based hypervisor (KVM)	Low engineering effort to place applications to IOS-XE routers
L2 connectivity to Data Plane	Full visibility of all traffic – ideal for analytics applications
Direct L2 connectivity between VMs if desired	Service chaining will not consume additional data plane resources
64-bit Linux kernel	Ability to allocate up to 12GB amongst VMs for large memory consuming applications
Multi-cored CPU	Up to 3 cores available for VMs
Extended Service Modules (SM-X)	Ability to move applications from Service Containers to UCS-E as performance needs grow with no additional appliances
Network Interface Modules (NIM)	Protected storage – RAID 1
mSATA (ISR 4300 series)	Storage without sacrificing a NIM card slot

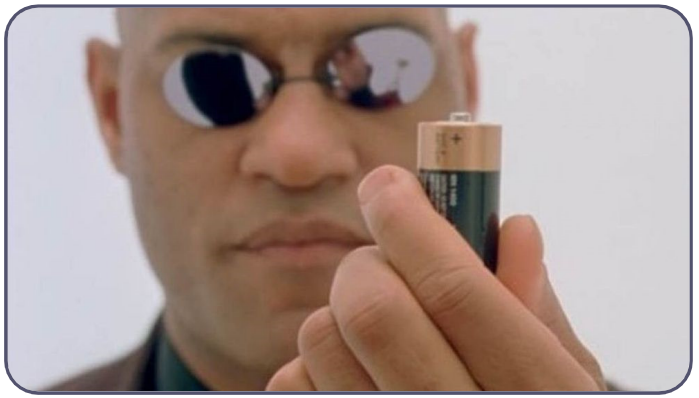
What can you do with Service Containers?

Benefits to MSPs / Partners / System Integrators

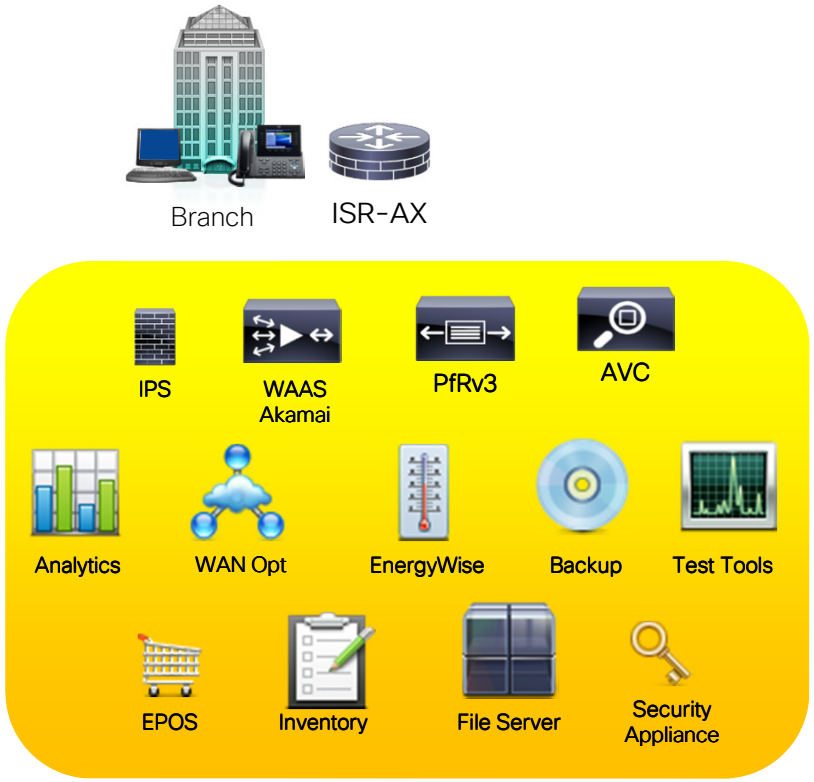
- Opportunity to offer more than WAN connectivity
- Scalable service offerings for all branch sizes
- Reduced costs – no branch hardware changes, and no physical cabling
- Seamless transition to cloud

Benefits to Customers

- Ultra efficient – CapEx and Opex savings
- Space and power saving – no need for separate storage, server, cooling, switch
- Deploy in-house apps
- Seamless cloud transition
- Maintain operations during WAN outages



Example Applications

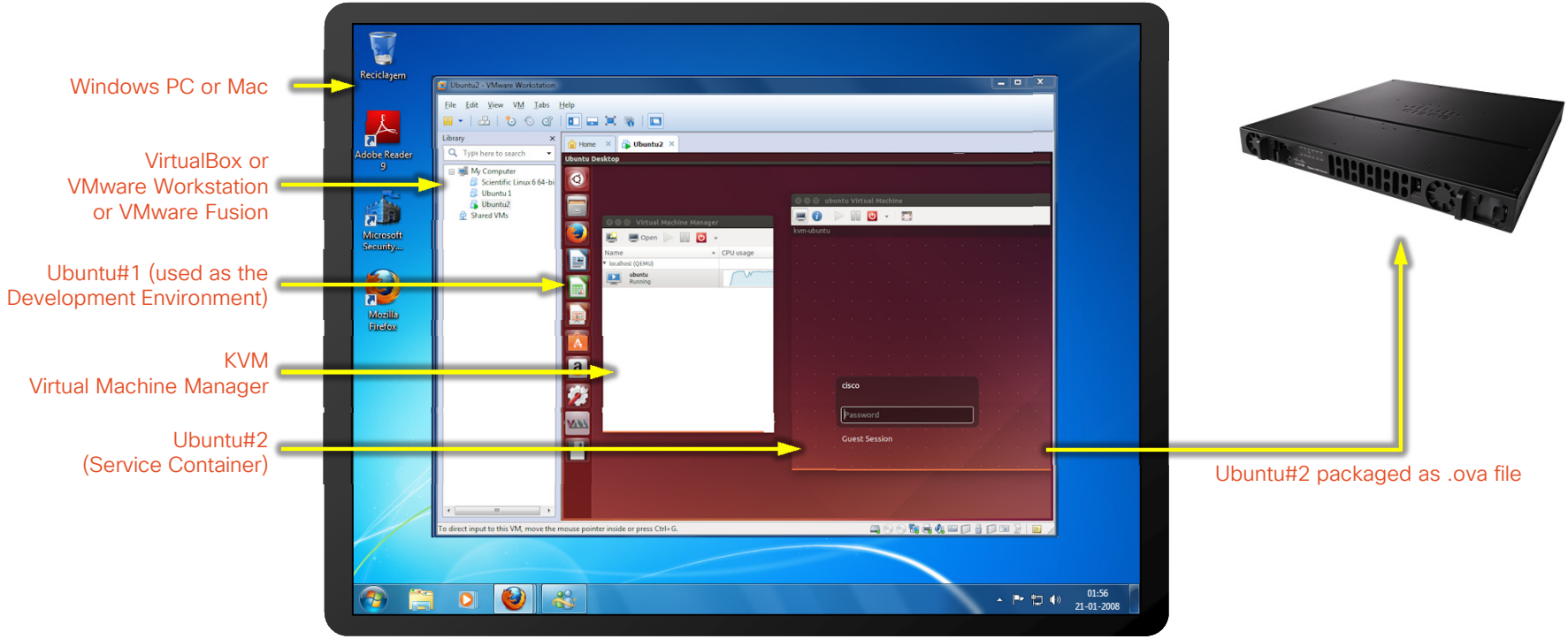


- WAN Optimization (vWAAS)
- Intrusion Prevention (Sourcefire/SNORT)
- Data Analytics, WAN Analytics, Wireshark
- Path instrumentation
- Cloud Storage with Local Sync
- Retail Store Inventory Management
- MSP SLA Monitoring
- Your Killer App

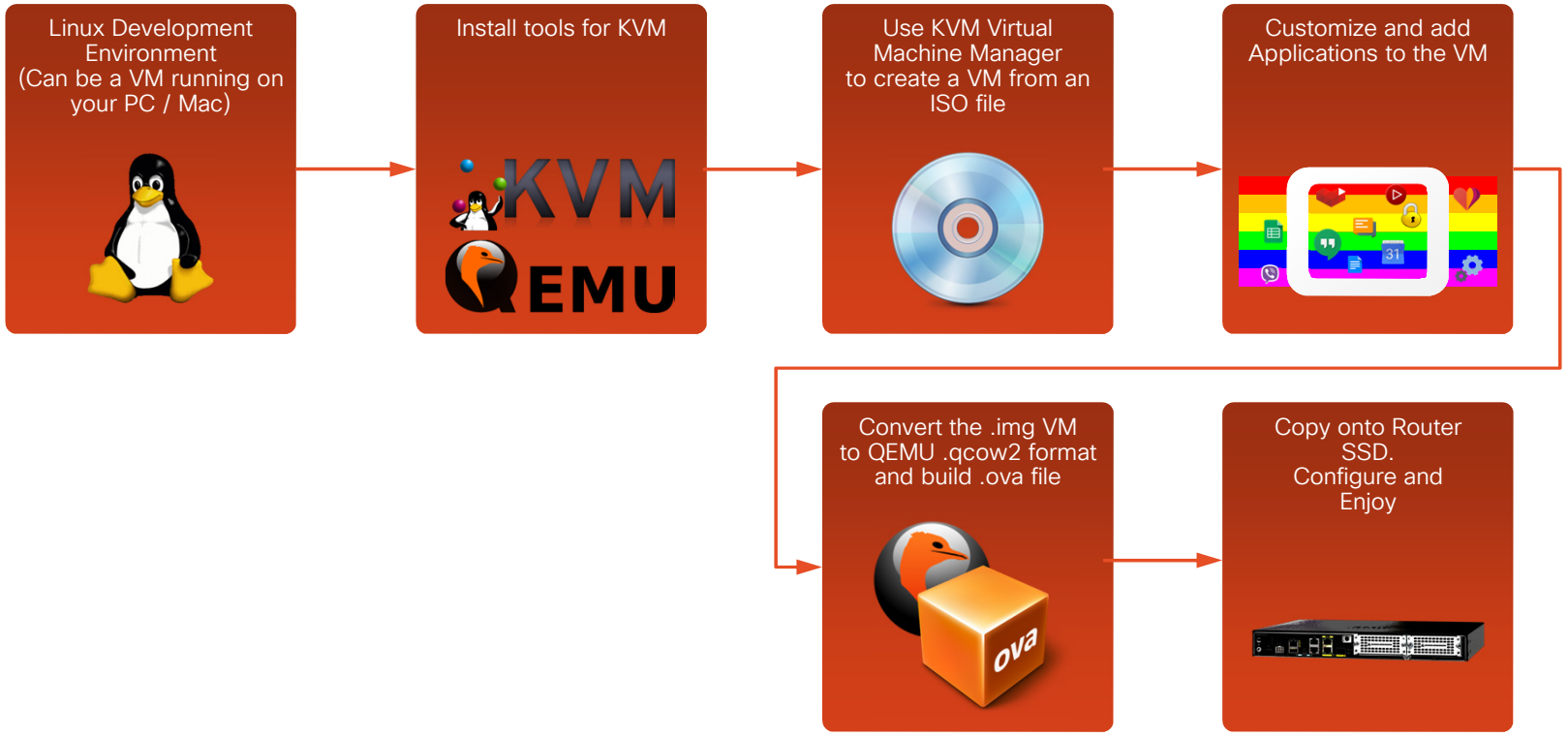
What you need to create your own Service Container

Overview of the Procedure

A Virtual Machine within a Virtual Machine!



Steps Involved



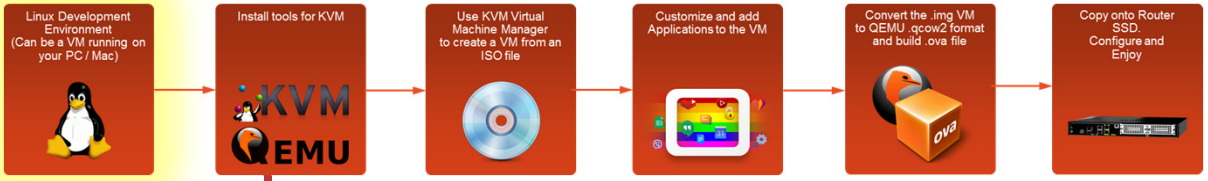
PC Requirements

- At least 8GB more memory than the size of the desired service container memory
- At least 2s+15 GB of free disk space where s is the size of the desired service container disk space
  - For example a 20GB sized service container will require a development PC with at least 2x20+15 = 55GB of free disk space
  - If you don't have such space, consider using USB memory or network storage
- This tutorial creates a 4GB RAM, 20GB disk service container so your development environment needs 12GB (or more) RAM and 55GB of disk space

Router Requirements

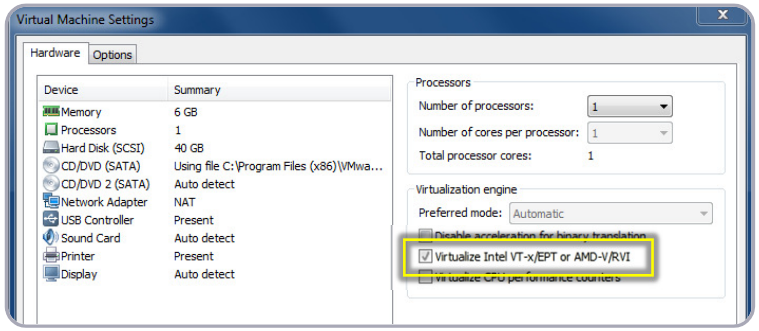
- Any ISR 4000 series router with at least 8GB DRAM and more Flash, and either mSATA SSD (4300 series only) or NIM-SSD or NIM-HD
- 4321 router has only 1 core (4 vcpu) available for service containers
- All other 4000 series have 3 cores (12 vcpu) available

Creating and Deploying a Service Container



You are Here

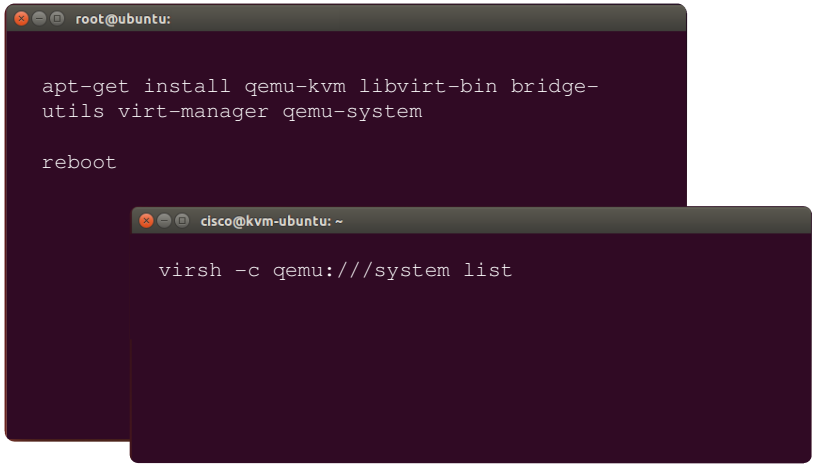
Install Ubuntu



- Download Ubuntu Desktop Linux ISO 64-bit long term support (LTS) 14.04.x from http://www.ubuntu.com/desktop
- Either install it natively on a PC, or run it inside VirtualBox, VMware Workstation or VMware Fusion
- Shut down VM, enable 'Virtualize Intel VT-x/EPT' from VirtualBox/Workstation/Fusion and start up the VM
- Confirm VT-x by typing the following in a terminal (it should return 1):

```
egrep -c '(svm|vmx)' /proc/cpuinfo
```

Install tools for KVM



- As root user type the following:

```
apt-get install qemu-kvm libvirt-bin
bridge-utils virt-manager qemu-system
```

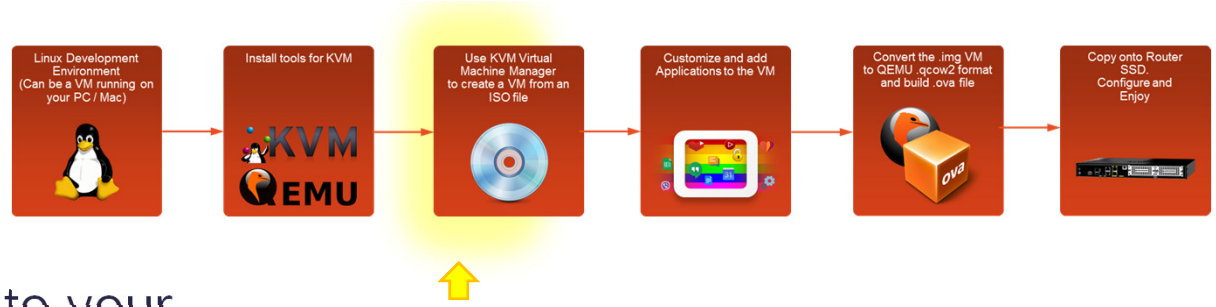
- Reboot the VM
- After it restarts, type the following as a non-root user:

```
virsh -c qemu:///system list
```

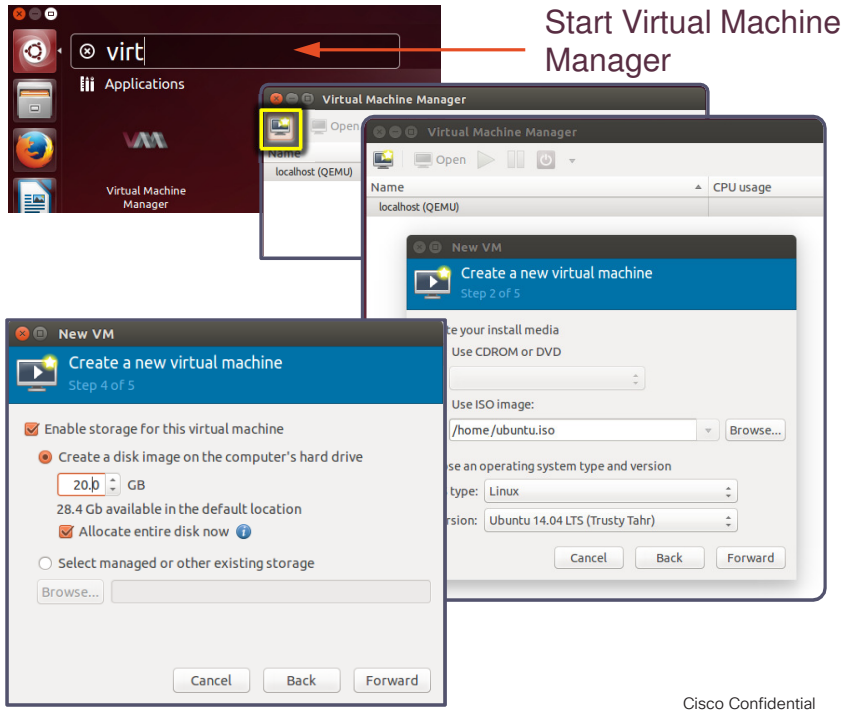
- If successful it will display this:

```
Id    Name                           State
----------------------------------------------------
```

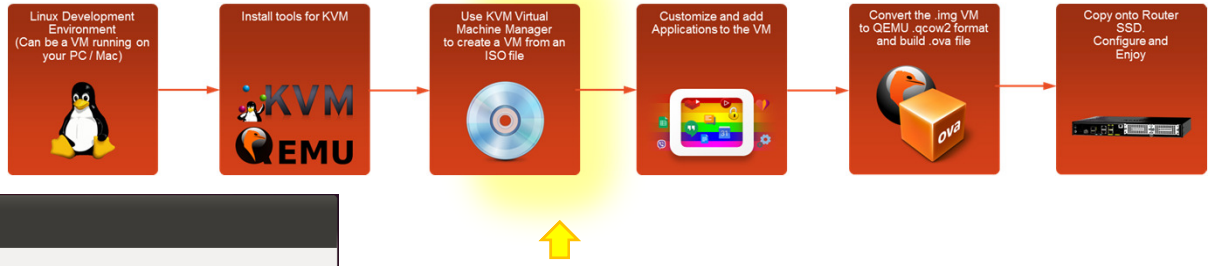
Create a VM from an ISO file 1/2

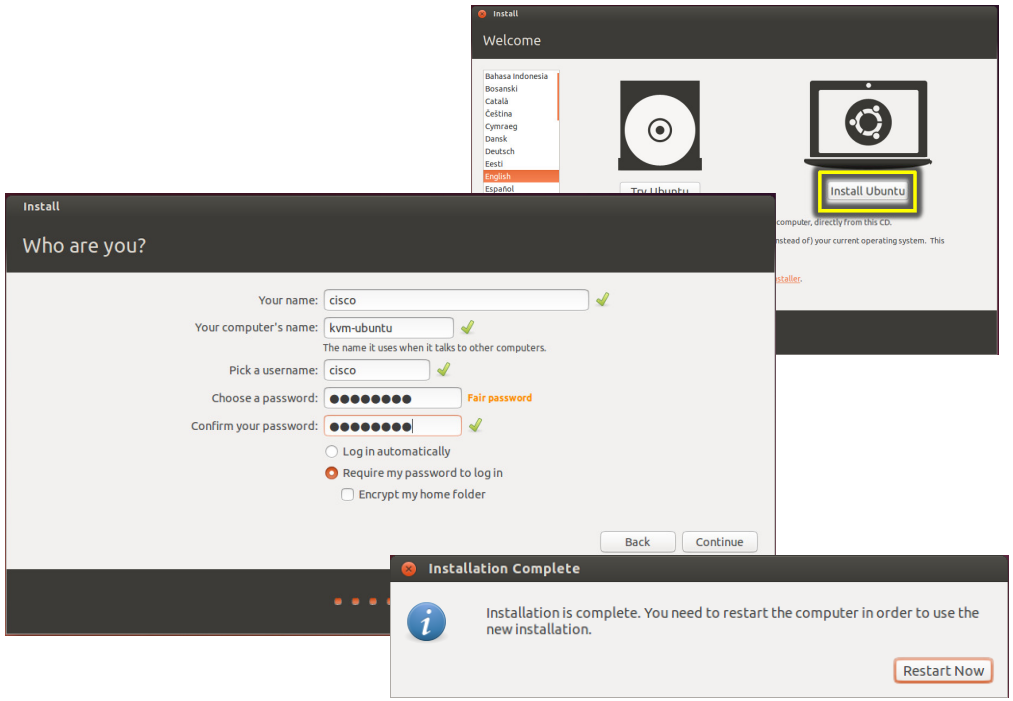


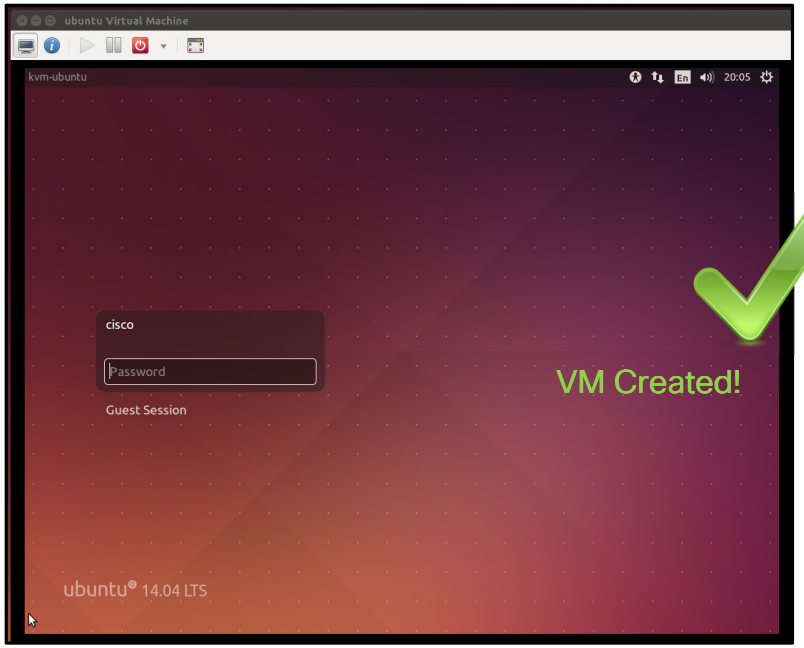
- Copy the Ubuntu ISO into your Linux environment
- Start up Virtual Machine Manager and use it to create a Service Container VM
- Select 4096MB, 1 CPU, 20GB
- Eventually the VM will launch and Ubuntu will begin installing



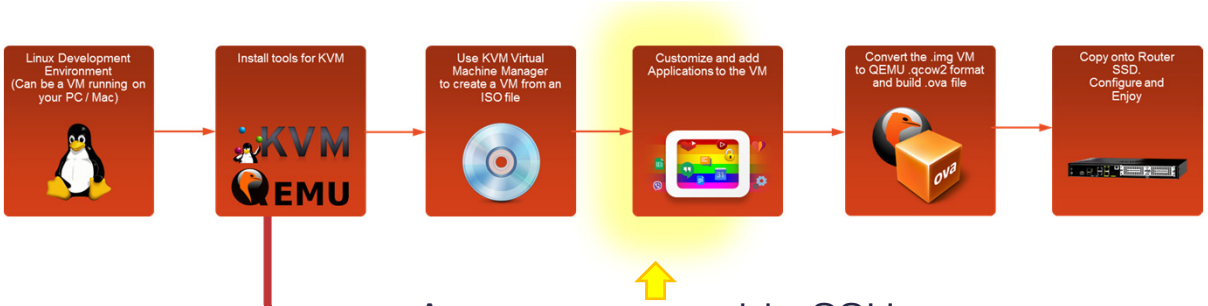
Create a VM from an ISO file 2/2







Customizing the Service Container VM 1/2



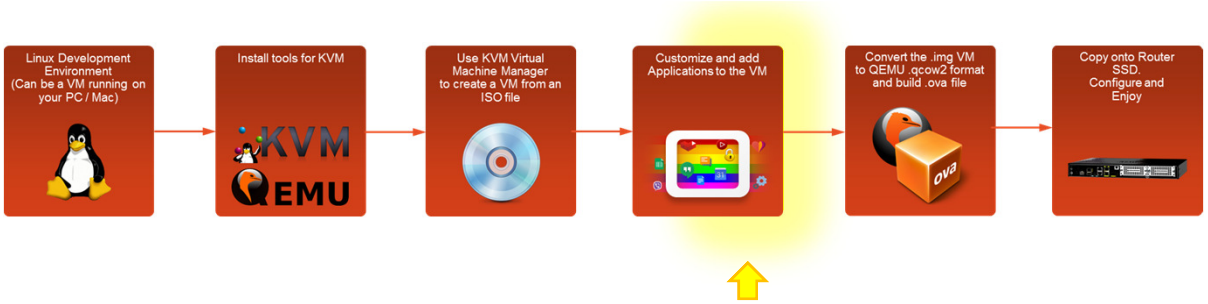
- As root user enable SSH:
  - `apt-get install openssh-server`
  - Still as root user edit /etc/ssh/sshd_config and change `PermitRootLogin without-password` to `PermitRootLogin yes` then save the file and reboot
  - After restart test it! `ssh root@127.0.0.1`
- As root user create a file /etc/init/ttyS0.conf as shown on the left and then type `sudo start ttyS0`

root@ubuntu:

```
# ttyS0 - getty
#
# This service maintains a getty on ttyS0

start on stopped rc RUNLEVEL=[2345] and (
            not-container or
            container CONTAINER=lxc or
            container CONTAINER=lxc-libvirt)

stop on runlevel [!2345]

respawn
exec /sbin/getty -L 9600 ttyS0 vt102
```

/etc/init/ttyS0.conf file contents

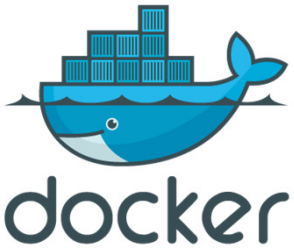
Customizing the Service Container VM 2/2



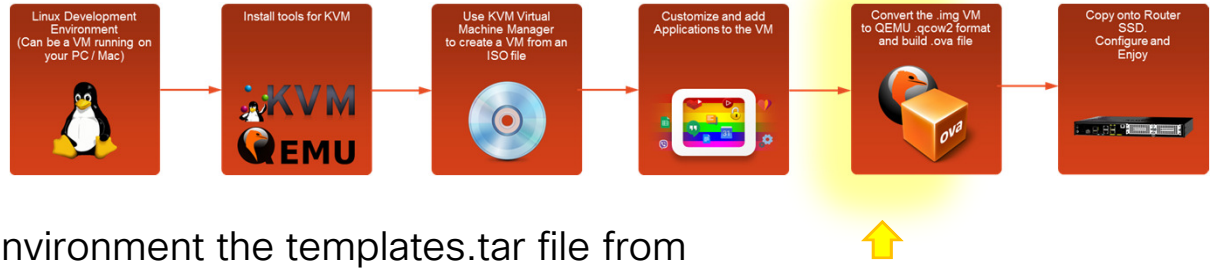
Install your apps! For example Docker: https://docs.docker.com/linux/step_one/

```
cisco@kvm-ubuntu: ~

su
apt-get update
apt-get install curl
exit
curl -fsSL https://get.docker.com | sh
su
usermod -aG docker cisco
reboot
```

```
cisco@kvm-ubuntu: ~

docker run hello-world
Hello from Docker.
This message shows that your installation appears to be
working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the
Docker Hub.
 3. The Docker daemon created a new container from that image
which runs the
    executable that produces the output you are currently
reading.
 4. The Docker daemon streamed that output to the Docker
client, which sent it
    to your terminal.
```



Build the .ova File



Download into the Development Environment the templates.tar file from https://github.com/shabaz123/ServiceContainers/raw/master/templates.tar

Create a folder called container-build

Download templates.tar and uncompress

Create a version.ver file

Converts the KVM .img file into qcow2 format

Packages up a .ova file. Output is called ubuntu.ova

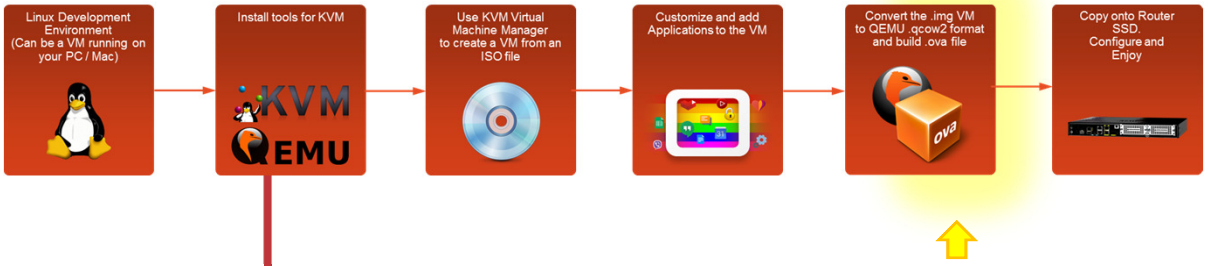
```
mkdir container-build; cd container-build
wget
https://github.com/shabaz123/ServiceContainers/raw/master/templ
ates.tar
tar xvf templates.tar
cp templates/create_ova.sh .
mkdir ubuntu; cd ubuntu
cp ../templates/package.yaml .
grep manifest-version package.yaml
echo 1.0 > version.ver
su
qemu-img convert -p -c -f raw -o compat=0.10 -O qcow2
/var/lib/libvirt/images/ubuntu.img ubuntu.qcow2
chown bob:bob ubuntu.qcow2
exit
./create_ova.sh -mts 200000 -mfs 100000 ubuntu
```

Result

container-build contents

```
create_ova.sh
templates
    create_ova.sh
    package.yaml
    readme.txt
templates.tar
ubuntu
    package.yaml
    ubuntu.mf
    ubuntu.ova
    ubuntu.qcow2
    version.ver
```

What Does the .ova Contain?



- The .ova file is actually just a .tar file
- (A .tar file is a concatenation of multiple files into a single file for convenience)

Example ubuntu.mf file

```
SHA1(package.yaml)= 4d5e21f805c61bc247aa70dfe314e2e0984fba9a
SHA1(ubuntu.qcow2)= a0b7487781dcb4073d06c4f957f09a30a6961f1c
SHA1(version.ver)= 61652cd1568dcf2614df833eba241755eee34e89
```

Example package.yaml file (snippet)

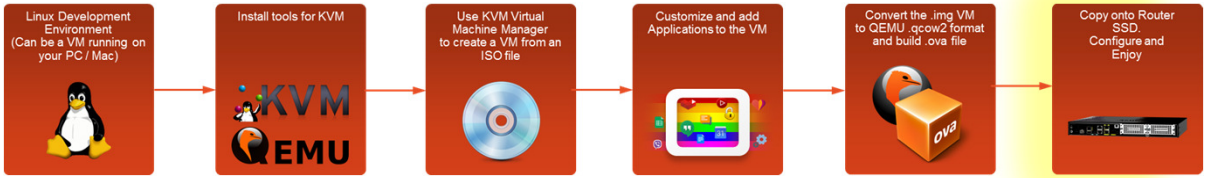
```
manifest-version: 1.0

info:
  name: ubuntu
  description: "KVM Ubuntu 14.04 LTS"
  version: 1.1

app:
  # Indicate app type (vm, paas, lxc etc.,)
  apptype: vm

  resources:
   cpu: 10
   memory: 3999744
```

File	Description
package.yaml	A text file describing the Service Container (amount of memory and CPU needed etc)
version.ver	A file containing a number which matches the number inside package.yaml
ubuntu.qcow2	The Virtual Machine in .qcow2 format
ubuntu.mf	A file containing SHA1 hash values list for all files in the .ova

Copy .ova onto Router SSD

Copy .ova across (USB memory is easiest)

```
copy usb0:ubuntu.ova harddisk:
```



Configure virtual-service

Download full example router configuration from

https://github.com/shabaz123/ServiceContainers/blob/master/example-router-config.txt

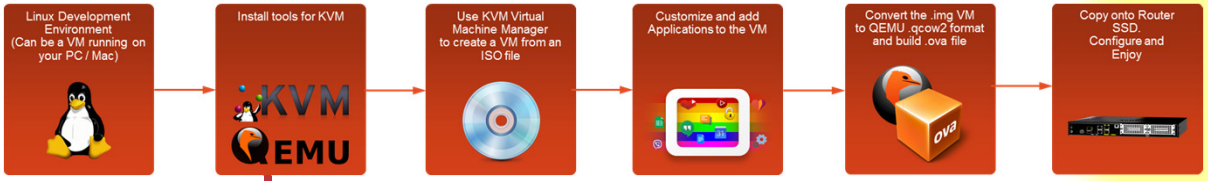
```
interface VirtualPortGroup1
 ip address 10.0.0.1
255.255.255.0
virtual-service
 signing level unsigned
virtual-service install
virtual-service ubuntu
 vnic gateway
VirtualPortGroup1
  guest ip address 10.0.0.2
```

IP address for router side

Allow unsigned VMs to run

IP address for the VM

Install the virtual-service




```
debug virtual-service all

terminal monitor

virtual-service install name
ubuntu package
harddisk:ubuntu.ova

show virtual-service list
```

Installs the .ova file as a virtual-service

Check to see the status

```
virtual-service ubuntu

activate
```

Installs the .ova file as a virtual-service (Check the status again afterwards)

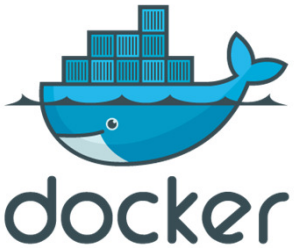
```
ping 10.0.0.2

ssh -l cisco 10.0.0.2
```

Connect to the virtual-service!

Running Docker Apps

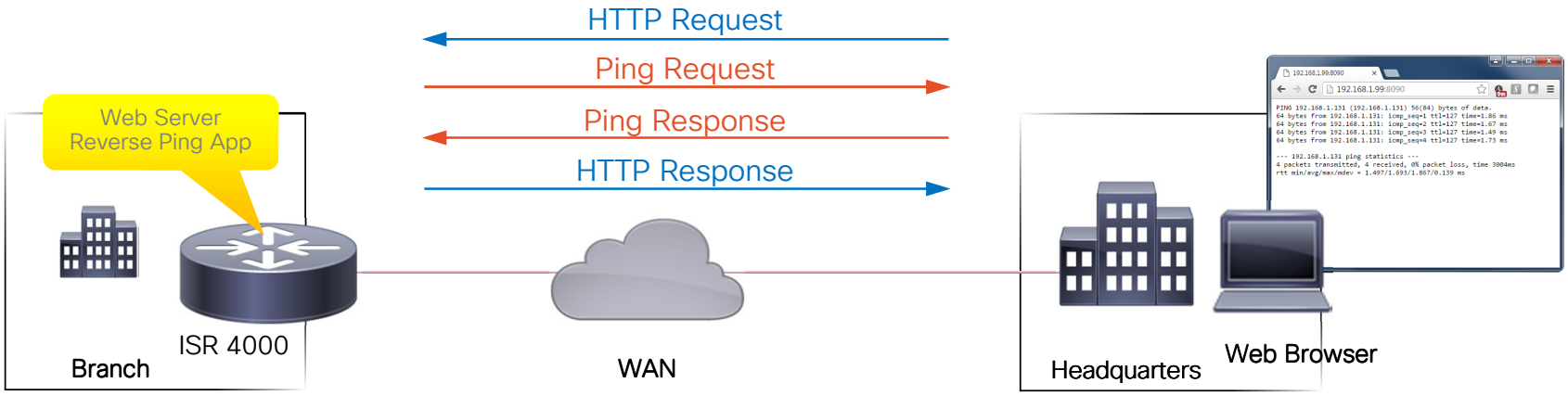
The Service Container has Docker installed; it is possible to run a demo Docker Image by typing the following:

```
docker run docker/whalesay
cowsay boo
```

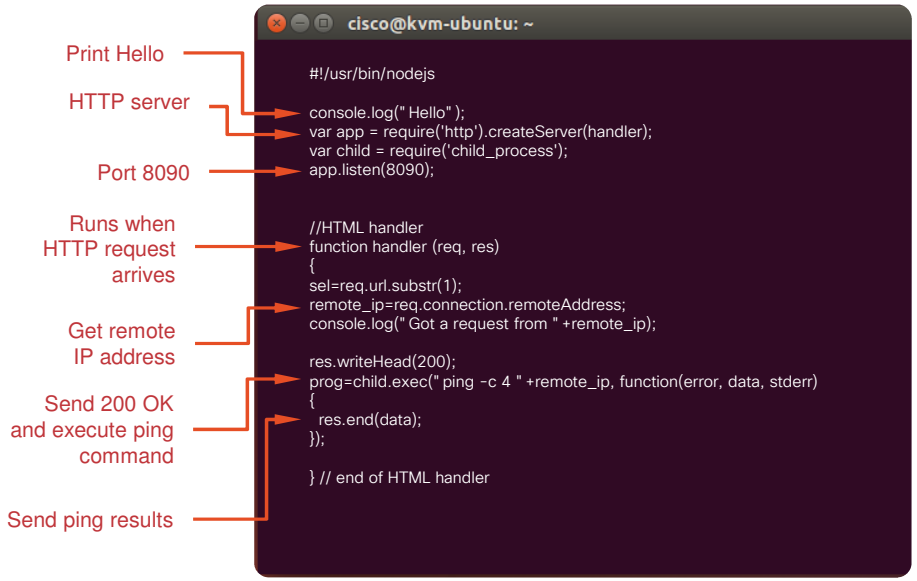


Creating your own browser-based Application

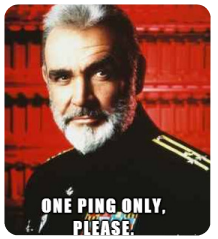
Reverse Ping Tool



Code



- Create an application in the development environment called basic_server.js
  - It is downloadable from https://github.com/shabaz123/ServiceContainers/blob/master/basic_server.js
- Install a platform called Node.js by typing as root user apt-get install nodejs
- As normal user change the permissions on the file by typing chmod 755 basic_server.js
- Test the application by typing ./basic_server.js
- Browse to http://xx.xx.xx.xx:8090 (your development environment address port 8090) and you should get your ping results after a few seconds
- Press Ctrl-C to exit!



Where could you store your applications?

GitHub is ideal if you wish to share your code

Create an account on github.com and then download the desktop app to allow syncing of your code to github.com

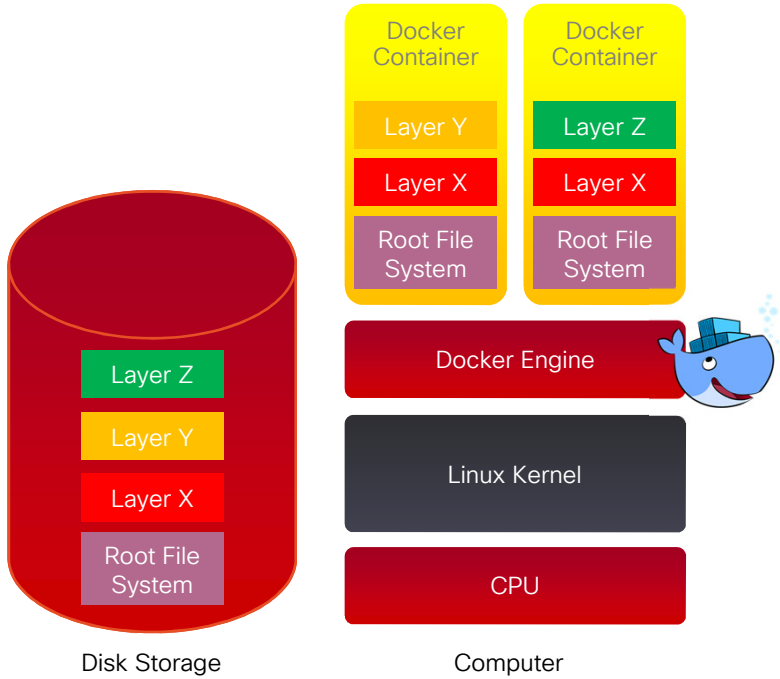
Others can clone and build on your work

github
SOCIAL CODING

Creating a Docker Image

How Docker Works

- A Docker Image is composed of ‘layers’. When run the instance/process is known as a Docker Container.
- When an image is ‘pulled’ from the cloud, these layers are downloaded (unless they have already been previously downloaded)
- A web caching app could have (say) an Ubuntu layer, a ‘wget’ layer (to perform HTTP fetches), and a layer that contains your glue code
- A traffic test tool app might also have an Ubuntu layer and a wget layer (to exercise HTTP traffic) and some different glue code.
- Since both images have Ubuntu and wget layers in common, these common layers only get downloaded once
- Note that the root file system is also part of the Docker Image, and each container gets its own copy. Any malicious container that modifies the /bin or /lib folders in Linux won’t be able to affect other containers



Example Docker Commands

See what images are downloaded

```
docker images
```

Delete an image

```
docker rmi -f <image_name>
```

Start up an image

```
docker run <image_name>
```

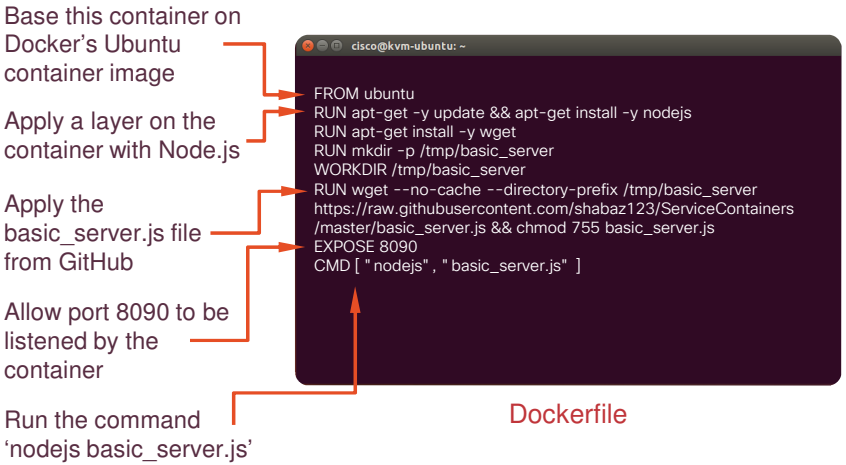
See what containers are running

```
docker ps
```

Stop a container

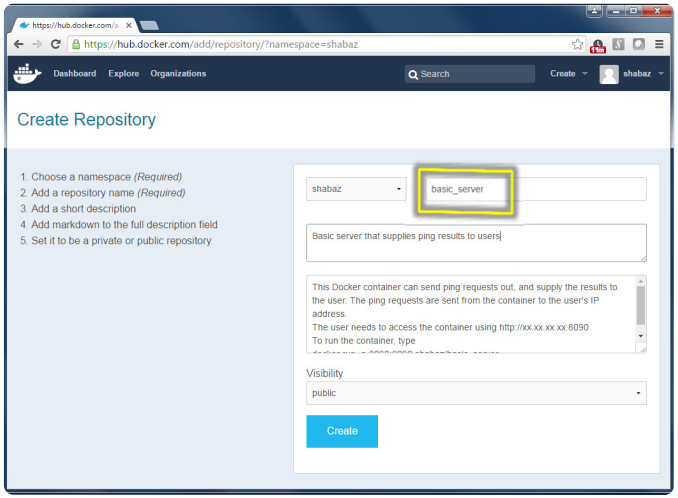
```
docker stop <container_id>
```

Creating a Docker Image 1/2



- Install Docker in your Development Environment the same as before (it was installed in the Service Container VM earlier, follow the same procedure)
- Create a folder and put a file called Dockerfile in there – you can copy it from https://github.com/shabaz123/ServiceContainers/blob/master/Dockerfile
- Build the Container by typing docker build -t basic_server .
- Test it by typing docker run -p 8090:8090 basic_server

Creating a Docker Image 2/2



- Create an account at http://hub.docker.com
- Click on Create Repository and type basic_server in the Enter Name field (it must match the name of the Container that we have just created)
- In the development environment get the Image ID by typing docker images | grep basic_server
- Use your hub.docker.com account username and the Image ID and type:

  docker tag d8dc7cbca23f bob/basic_server:latest

  docker login -u=bob

  docker push bob/basic_server
- To test a pull operation, first delete the images from the development environment:

  docker rmi -f bob/basic_server
- This will fetch it and run the image:

  docker run -p 8090:8090 bob/basic_server

Deploying Apps to Service Containers using Docker Images

Deploying a Docker Image

One line operation!

Log into the Service Container and type:

```
docker run -p 8090:8090 bob/basic_server
```

If you wish it to always start up after reboot then type:

```
docker run --restart=always -p 8090:8090 bob/basic_server
```

Invoking Services from the Router

Since the application relies on HTTP, it is possible to invoke it directly from the router too (doesn't make much sense for this particular app, but could be useful for other apps)

```
tclsh
more http://10.0.0.2:8090/index.html
tclquit
```

Summary

Services Containers Summary

- Technology for deploying in-house, Cisco and third party applications at all branches in an extremely efficient and cost effective manner
- High-flexibility KVM and Linux
- Straightforward to build a Service Container
- Simple applications are quick to write but large applications are possible too
- A browser based application can be created in about 15 lines of code
- Docker makes deploying applications on Cisco routers a 1-line exercise

Appendices

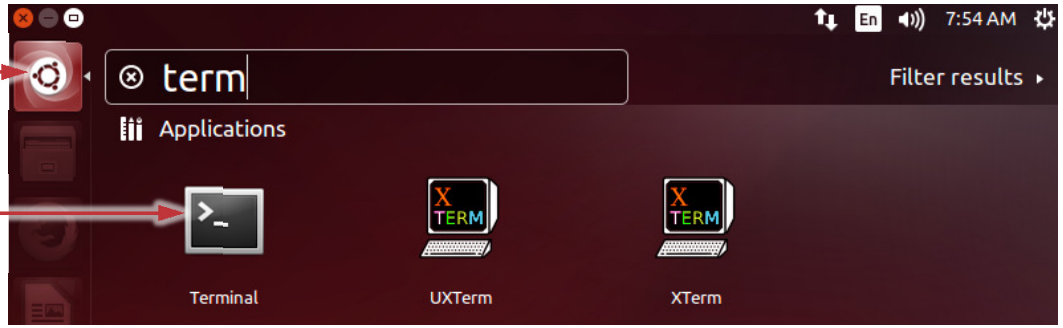
Linux Quick Reference

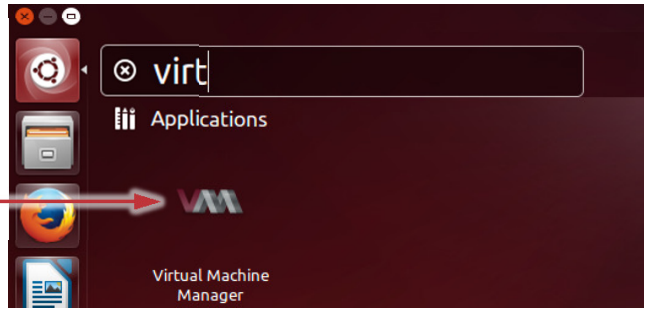
Working with Ubuntu using the Search Button

Click the Search icon and then just type what you want

Terminal

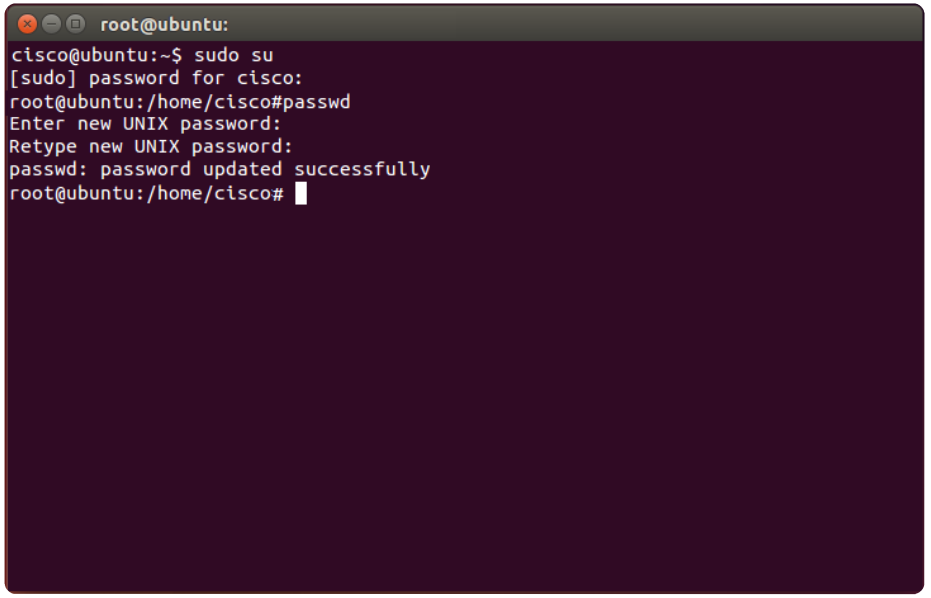
Virtual Machine Manager





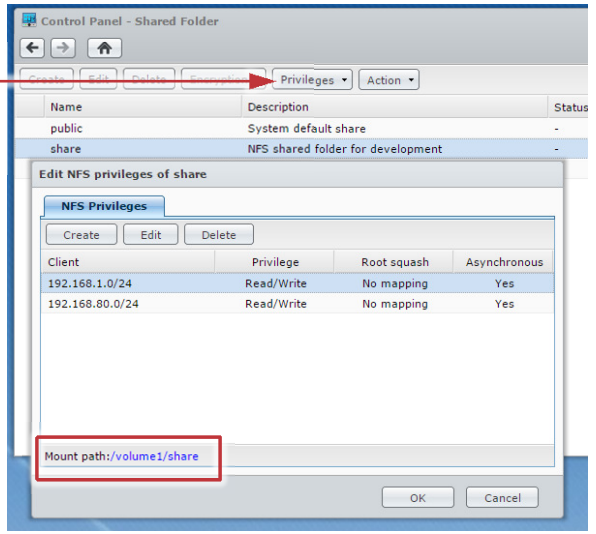
Creating a root user (super-user) password

- Initially there is no root password set
- Become root user by typing sudo su and using your normal user password
- Next type passwd to set the root user password
- Type exit to become normal user again
- From now on you can just type su to become super-user



Mounting a NFS File System

Enable NFS privileges on your Network Storage device




```
root@ubuntu:

mkdir /mnt/share
apt-get install nfs-install
mount -t nfs xx.xx.xx.xx:/volume1/share /mnt/share
```

Quick Linux Command Reference

Introduction – Shell (command prompt)

Once logged into the Linux machine, you will see a terminal view or command prompt. The software that displays the prompt and allows you to type content is called the 'shell'. You can leave the shell at any time by typing 'exit'.

Shell command syntax

Commands typically (but not always) have the syntax

```
command –option –option argument
```

An example would be

```
ls –l *.txt
```

The command 'ls' performs a directory (i.e. folder) listing. The argument '-l' displays in a 'lengthy' format. The '*.txt' argument defines what files to list.

Redirection

You can force the output into a file, using '>'.

For example,

```
ls > listing.txt
```

would place the directory listing output into a file called listing.txt.

History

You can see earlier commands that you have typed by using the `history` command. It is very useful. To pause per page use `history | more`

Command	Example	Description
man	man pwd	Bring up a screen of help information or a manual page for a command. Press space to step through it, and press 'q' to quit. Very useful to quickly obtain information about any command. The example here pulls up information on what the 'pwd' command does.
cd	cd /home	Change directory (folder)
mkdir	mkdir bob mkdir /home/bob	Make a directory (folder)
pwd	pwd	Display the current directory
ls	ls -altr	Directory listing of files. The '-altr' is a combination of 4 options. '-a' lists hidden files too. '-l' displays in a lengthy detailed format. '-t' displays in date stamped order. '-r' reverses the order, so that the most recent file is printed last.
cp	cp bob.txt mytest1.txt	Copies a file
mv	mv bob.txt tests/.	Moves a file into a folder or to a new file name. The example shown here would move the file into the tests folder.
rm	rm bob.txt	Delete a file
cat	cat readme.txt	Displays the contents of a file.
more	more readme.txt ls -altr | more cat readme.txt | more	Displays content, but pauses per page. It can be combined with other commands that print lots of output, using the 'pipe' character '|'. The second example would be useful for viewing the contents of a folder with hundreds of files. The third example does the same as the first example.
su	su su - bob	Change user. This allows you to jump from one user to another. It effectively creates a new shell. The first example allows you to become the root user who has administrator privileges. The second example changes to the user called 'bob'. You can revert back to the original user (i.e. the original shell) by typing 'exit'.
ps	ps -ef	View the running processes. The example lists all processes ('-e') and in a long format ('-f'). This command is useful to see what is running, and also useful if you want to force a process to close using the next command in this table
kill	kill -9 3244	Force a process (program) to be closed. The '-9' is a special signal that usually tells the program to close. The argument 3244 is the process number, found by using the 'ps –ef' command mentioned above.
df	df -k	Check disk space. The '-k' displays the values in kbytes.
history	history	See all the previous commands that were typed

Docker Quick Command Reference

Docker Commands: Creating and Uploading Images

Category	Command	Syntax	Example	Description
Images (Repositories)	docker build	docker build -t <Image Name> .	docker build -t basic_server .	Creates an image based on the Dockerfile in the current folder.
Images (Repositories)	docker tag	docker tag <Image ID> <Username>/<Image Name>:<Tag>	docker tag d8dc7cbca23f bob/basic_server:latest	The example here creates an image based on an existing Image ID. The new image has the name bob/basic_server and a tag named latest
Hub (hub.docker.com or other Registry)	docker login	docker login –u=<Username> docker login –u=<Username> <IP Address>	docker login –u=bob	Log into hub.docker.com (or some other Registry if the address is specified) for subsequent docker push instructions
Images (Repositories)	docker push	docker push <Username>/<Image Name>	docker push bob/basic_server	Stores the Image (repository) to hub.docker.com

Docker Commands: Managing Images and Containers

Category	Command	Syntax	Example	Description
Images (Repositories)	docker images	docker images		List all images
Images (Repositories)	docker rmi	docker rmi -f <Image Name> docker rmi -f <Image ID>	docker rmi -f bob/basic_server docker rmi $(docker images -f " dangling=true" -q)	Delete an image. There is no automatic garbage collection and after multiple deploys there can be 'dangling' images. The second example deletes all such unused images but you will need to delete all exited containers first (see below)
Images (Repositories)	docker pull	docker pull <Image Name>	docker pull bob/basic_server	Fetch an image from the cloud
Containers	docker ps	docker ps docker ps -a		List running Docker containers. To list all containers (including stopped ones) use -a
Containers	docker rm	docker rm <Container ID>	docker rm 6bec2a9e3a22 docker rm $(docker ps -q -f status=exited)	Deletes a Docker container. The second example deletes all exited containers (a new container is created each time the container is stopped and restarted)
Containers	docker stop	docker stop <Container ID>	docker stop 6bec2a9e3a22	Stop a container that is running
Run	docker run	docker run [--restart=always] [-p outside:inside] <Image Name> docker run <Image Name> <Linux Command> docker run -it <Linux Shell Name>	docker run bob/basic_server docker run -p 8090:8090 shabaz/basic_server docker run --restart=always -p 8090:8090 shabaz/basic_server docker run ubuntu echo hello	Runs an image (i.e. creates a container process based on an image, and then runs it) The second example is used to expose a port so that traffic can be sent to the container The third example instructs Docker to automatically run the image whenever the VM restarts. The fourth example runs the image called ubuntu and executes the command echo hello
Run	docker exec	docker exec -it <Container ID> <Linux Command>	docker exec -it 7d82ab2103f8 /bin/bash	Runs a command in the specified running container. For example this is useful for debugging purposes

Docker Commands: Advanced

Category	Command	Syntax	Example	DescriptionContsa
Containers and Images (Repositories)	docker commit	docker commit <Container ID> <New Image Name>	docker commit d8dc7cbca23f new_basic_server	Creates an image based on a current running container (including any changes made to the container) Note: not advisable; preferable to do any changes in a Dockerfile so that layers can be updated over time
Images (Repositories)	docker inspect	docker inspect <Image Name>	docker inspect bob/basic_server	Displays information about the image such as the processor architecture, exposed ports and so on

Thank you.