Networking

CMSC398W: Practical Tools For Efficient Development

Mohammad Durrani

June 22, 2026

Think-Pair-Share: Docker Images vs. Containers

The Question

What is the main difference between a Docker image and a Docker container?

- A) A container requires more storage space than an image
- B) An image is a running instance, while a container is the template
- C) An image is a read-only template, while a container is a running instance
- D) There is no difference between containers and images

Think-Pair-Share: Docker Images vs. Containers

The Question

What is the main difference between a Docker image and a Docker container?

- A) A container requires more storage space than an image
- B) An image is a running instance, while a container is the template
- C) An image is a read-only template, while a container is a running instance
- D) There is no difference between containers and images

Answer

C: An image is read-only; a container is a running instance of one.

Think-Pair-Share: Explaining Docker to a VM User

The Challenge

Your colleague is comfortable with VMs. Explain two key advantages of Docker containers over traditional VMs, and give a concrete scenario where they matter.

Think-Pair-Share: Explaining Docker to a VM User

The Challenge

Your colleague is comfortable with VMs. Explain two key advantages of Docker containers over traditional VMs, and give a concrete scenario where they matter.

Answer

Many valid answers, for example:

- Startup speed: containers start in seconds vs. minutes; good for scaling a web service under sudden load
- Efficiency: containers share the host kernel, you can run 10 containers where you could only run 2 VMs

Networking Is Everywhere

As a Developer, You're (Pretty Much) Always on a Network

Action	Network Dependency
`git pull`	SSH/HTTPS to GitHub
Call a REST API	HTTP over TCP/IP
Query a database	TCP socket to DB host
Deploy to the cloud	SSH, HTTPS, container registry

Networking Is Everywhere

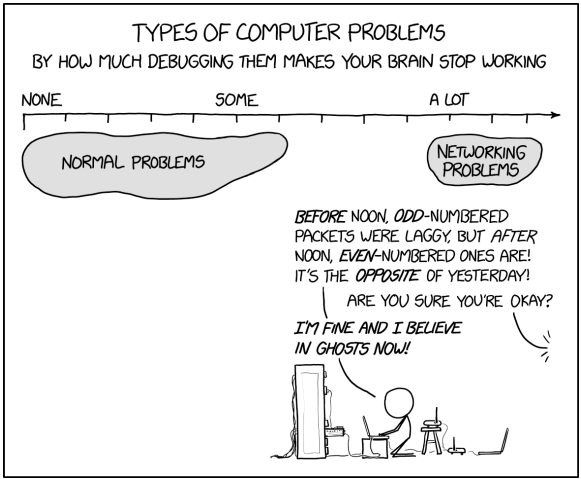
As a Developer, You're (Pretty Much) Always on a Network

Action	Network Dependency
`git pull`	SSH/HTTPS to GitHub
Call a REST API	HTTP over TCP/IP
Query a database	TCP socket to DB host
Deploy to the cloud	SSH, HTTPS, container registry

Why This Matters

When the network breaks, everything breaks, so knowing how to diagnose it is super important.

Class Alignment



When things go wrong with networking, it can be incredibly hard to debug and waste a lot of time. This is why we teach it in this class!

Real World Motivation

Spain's Football Blocking Incident (2026)

Spanish courts ordered ISPs to block Cloudflare IP ranges during La Liga matches to stop piracy streams.

Collateral damage: This was first identified when `docker pull` failed and CI/CD pipelines broke. More improtantly, some GPS tracking apps went offline for millions of users who had nothing to do with piracy.

Real World Motivation

Spain's Football Blocking Incident (2026)

Spanish courts ordered ISPs to block Cloudflare IP ranges during La Liga matches to stop piracy streams.

Collateral damage: This was first identified when `docker pull` failed and CI/CD pipelines broke. More improtantly, some GPS tracking apps went offline for millions of users who had nothing to do with piracy.

Why This Is Instructive

- IP-level blocking (L3) kills all traffic to those IPs indiscriminately
- Users saw TLS errors (L7 symptom of an L3 cause)
- `ping` and `traceroute` showed where packets vanished
- DNS worked fine, whicht ells you that the problem was routing, not name resolution

The layered model helps tell you where to look.

What We'll Cover Today

Networking Tools

Topic	Tools
Network model	OSI layers (conceptual)
Addressing	`ip addr`, IP / subnets
Transport	TCP vs. UDP, ports, `ss` / `netstat`
Connectivity	`ping`, `traceroute`
Name resolution	`dig`
Application layer	HTTP, `curl`

What We'll Cover Today

Networking Tools

Topic	Tools
Network model	OSI layers (conceptual)
Addressing	`ip addr`, IP / subnets
Transport	TCP vs. UDP, ports, `ss` / `netstat`
Connectivity	`ping`, `traceroute`
Name resolution	`dig`
Application layer	HTTP, `curl`

Note

We are not covering how networking works at a protocol level. For that, take CMSC417, in my opinion one of the best courses in the department.

The OSI Model: A Mental Map

Why We Use a Layered Model

Computer networking involves many interacting systems. A layered model lets us isolate problems at the right level instead of treating the network as a black box.

The OSI Model: A Mental Map

Why We Use a Layered Model

Computer networking involves many interacting systems. A layered model lets us isolate problems at the right level instead of treating the network as a black box.

OSI Model (7 Layers)

Layer	Name	Examples
7	Application	HTTP, DNS, SSH, SMTP
6	Presentation	TLS/SSL, encoding
5	Session	Session management
4	Transport	TCP, UDP, ports
3	Network	IP addresses, routing
2	Data Link	MAC addresses, switches
1	Physical	Cables, Wi-Fi radio

OSI Layers and Their Debugging Tools

Layer → Tool Reference

Layer	Focus	Tool
L1 Physical	Cable in? Wi-Fi signal?	(visual check)
L2 Data Link	Local segment reachable?	`arp`
L3 Network	IP address? Route to destination?	`ip addr`, `ping`, `traceroute`
L4 Transport	Port open? Service listening?	`ss` / `netstat`
L7 Application	DNS? TLS cert? Response code?	`dig`, `curl`

Think-Pair-Share: Which Layer?

The Challenge

Two computers are on the same network. They can see each other's MAC addresses, but cannot establish an IP connection. At which OSI layer is the problem?

- A) Layer 1 (Physical)
- B) Layer 2 (Data Link)
- C) Layer 3 (Network)
- D) Layer 4 (Transport)

Think-Pair-Share: Which Layer?

The Challenge

Two computers are on the same network. They can see each other's MAC addresses, but cannot establish an IP connection. At which OSI layer is the problem?

- A) Layer 1 (Physical)
- B) Layer 2 (Data Link)
- C) Layer 3 (Network)
- D) Layer 4 (Transport)

Answer

C, Layer 3 (Network). MAC visibility confirms L1 and L2 are working. The breakdown is at IP-level addressing or routing.

IP Addresses

What Is an IP Address?

An IP address is a unique identifier assigned to a network interface (not the device itself) so that packets can be routed to it.

IP Addresses

What Is an IP Address?

An IP address is a unique identifier assigned to a network interface (not the device itself) so that packets can be routed to it.

IPv4 vs. IPv6

Version	Format	Example	Notes
IPv4	4 octets, decimal	`192.168.1.100`	32-bit, pool nearly exhausted
IPv6	8 groups, hex	`2001:db8::1`	128-bit, effectively unlimited

IP Addresses

What Is an IP Address?

An IP address is a unique identifier assigned to a network interface (not the device itself) so that packets can be routed to it.

IPv4 vs. IPv6

Version	Format	Example	Notes
IPv4	4 octets, decimal	`192.168.1.100`	32-bit, pool nearly exhausted
IPv6	8 groups, hex	`2001:db8::1`	128-bit, effectively unlimited

Special Addresses

`127.0.0.1` (loopback) is your machine talking to itself, and it's always a good first `ping` target when debugging.

Interfaces, Not Devices

An Interface Is an Entry Point

A single machine can have multiple interfaces, each with its own IP address:

- `lo`: loopback interface (`127.0.0.1`), purely local
- `eth0`: wired Ethernet
- `wlan0`: Wi-Fi
- Virtual interfaces created by Docker, VPNs, etc.

Tool: ip addr show / ifconfig

Inspecting Your Interfaces

```
$ ip addr show          # Linux  (shorthand: ip a)
$ ifconfig              # macOS
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 ...
    inet 127.0.0.1/8 scope host lo
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 ...
    link/ether 02:42:ac:11:00:02
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
```

Tool: ip addr show / ifconfig

Inspecting Your Interfaces

```
$ ip addr show          # Linux  (shorthand: ip a)
$ ifconfig              # macOS
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 ...
    inet 127.0.0.1/8 scope host lo
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 ...
    link/ether 02:42:ac:11:00:02
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
```

Key Fields

Field	Meaning	Good Sign
`UP / LOWER_UP`	Interface enabled; physical link connected	Both present
`inet`	IPv4 address assigned	Expected address
`link/ether`	MAC address	Always present

TCP vs. UDP

Layer 4: Getting Data to the Right Application

IP gets packets to the right machine and layer 4 will get data to the right application.

Transmission Control Protocol

- Connection-oriented (handshake first)
- Reliable: ordered, no gaps, retransmits lost data
- Higher overhead

Use when: correctness matters (web, SSH, databases, email)

User Datagram Protocol

- Connectionless (fire and forget)
- Best-effort: packets may drop or reorder
- Low overhead

Use when: speed matters (DNS, video streaming, games)

Think-Pair-Share: TCP or UDP?

The Question

For each use case, which transport protocol is more appropriate and why?

1. Downloading a file from a server
2. A live video call (Zoom/FaceTime)
3. DNS lookup for `google.com`
4. SSH connection to a remote server

Think-Pair-Share: TCP or UDP?

The Question

For each use case, which transport protocol is more appropriate and why?

1. Downloading a file from a server
2. A live video call (Zoom/FaceTime)
3. DNS lookup for `google.com`
4. SSH connection to a remote server

Answers

1. TCP: every byte must arrive correctly
2. UDP: a dropped frame is better than a stall; retransmission is too slow
3. UDP: fast query/response; client just retries if needed
4. TCP: reliability and ordering are essential for a shell session

Ports and Sockets

Port Numbers

A port (0-65535) tells the OS which application should receive incoming data.

Port	Protocol	Service
22	TCP	SSH
53	TCP/UDP	DNS
80	TCP	HTTP
443	TCP	HTTPS
5432	TCP	PostgreSQL
3306	TCP	MySQL

Ports and Sockets

Port Numbers

A port (0-65535) tells the OS which application should receive incoming data.

Port	Protocol	Service
22	TCP	SSH
53	TCP/UDP	DNS
80	TCP	HTTP
443	TCP	HTTPS
5432	TCP	PostgreSQL
3306	TCP	MySQL

Sockets

A socket = IP address + protocol + port. Example: `172.17.0.2:tcp:443`. See `/etc/services` for more well-known ports.

Checking Reachability with ping

What `ping` Does

`ping` sends ICMP Echo Requests to a destination; if the host replies, you have a confirmed Layer 3 path.

Checking Reachability with ping

What `ping` Does

`ping` sends ICMP Echo Requests to a destination; if the host replies, you have a confirmed Layer 3 path.

Key Output Fields

Field	Meaning
`icmp_seq`	Sequence number; gaps mean dropped packets
`ttl`	Time To Live, decremented per hop
`time`	Round-trip latency in ms
`packet loss`	Non-zero means network trouble

ping in Action

Example

```
$ ping 8.8.8.8
PING 8.8.8.8 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=116 time=12.5 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=116 time=12.2 ms
^C
--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss
rtt min/avg/max/mdev = 12.2/12.4/12.5/0.1 ms
```

ping in Action

Example

```
$ ping 8.8.8.8
PING 8.8.8.8 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=116 time=12.5 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=116 time=12.2 ms
^C
--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss
rtt min/avg/max/mdev = 12.2/12.4/12.5/0.1 ms
```

Common Errors

- Request timed out / 100% packet loss: host unreachable or firewall blocking ICMP
- Destination Host Unreachable: routing failure between you and the target

Debugging Strategy with ping

Step-by-Step

Step	Command	Tests
1	`ping 127.0.0.1`	Your own network stack
2	`ping <your IP>`	Your specific interface
3	`ping <gateway IP>`	Local router reachability
4	`ping 8.8.8.8`	Internet connectivity
5	`ping google.com`	DNS resolution + internet

Debugging Strategy with ping

Step-by-Step

Step	Command	Tests
1	`ping 127.0.0.1`	Your own network stack
2	`ping <your IP>`	Your specific interface
3	`ping <gateway IP>`	Local router reachability
4	`ping 8.8.8.8`	Internet connectivity
5	`ping google.com`	DNS resolution + internet

Find Your Gateway

```
$ ip route show | grep default      # Linux
default via 192.168.1.1 dev eth0

$ netstat -rn | grep default        # macOS
default            192.168.1.1    UGScg    en0
```

Tracing the Path with traceroute

What `traceroute` Reveals

`traceroute` maps each router ("hop") between you and the destination, letting you pinpoint where a connection breaks or slows down.

Tracing the Path with traceroute

What `traceroute` Reveals

`traceroute` maps each router ("hop") between you and the destination, letting you pinpoint where a connection breaks or slows down.

Interpreting Output

Symbol	Meaning
Three RTT values	Latency for three probes at that hop
* * *	No reply (firewall or congestion)
Sudden RTT jump	Slow link or congested router
Trace stops early	Likely point of failure

traceroute in Action

Example

```
$ traceroute google.com
 1  _gateway (192.168.1.1)      0.5 ms   0.5 ms   0.5 ms
 2  10.x.x.x                    8.1 ms   8.0 ms   8.0 ms
 3  isp-router.net (A.B.C.D)    9.5 ms   9.4 ms   9.4 ms
 4  * * *                       # no reply (firewall / security policy)
 5  backbone.net (E.F.G.H)     15.2 ms  15.1 ms  15.1 ms
12  lga34s35.1e100.net         11.9 ms  11.8 ms  11.9 ms
```

Why DNS?

The Problem with IP Addresses

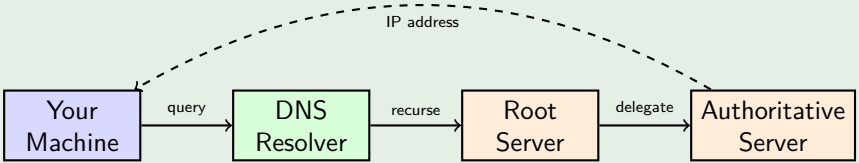
Computers route by IP address, but humans remember names, so DNS translates `www.google.com` into `142.250.191.174`.

Why DNS?

The Problem with IP Addresses

Computers route by IP address, but humans remember names, so DNS translates `www.google.com` into `142.250.191.174`.

Resolution Process



Tool: dig

Domain Information Groper

`dig` lets you query DNS records directly from the command line.

Tool: dig

Domain Information Groper

`dig` lets you query DNS records directly from the command line.

Common Record Types

```
# A record: IPv4 address
$ dig A www.google.com +short
142.250.191.142

# AAAA record: IPv6 address
$ dig AAAA www.google.com +short
2607:f8b0:4004:834::200e

# Query a specific DNS server
$ dig @8.8.8.8 A www.stanford.edu +short
171.67.215.200
```

Debugging DNS

Symptom vs. Cause

Observation	Diagnosis
`ping 8.8.8.8` works, `ping google.com` fails	DNS problem
`dig` returns no answer	Wrong resolver or name doesn't exist
`dig` succeeds but browser fails	Browser-level issue or proxy
`dig @8.8.8.8` works but `dig` alone fails	Local DNS resolver misconfigured

Debugging DNS

Symptom vs. Cause

Observation	Diagnosis
`ping 8.8.8.8` works, `ping google.com` fails	DNS problem
`dig` returns no answer	Wrong resolver or name doesn't exist
`dig` succeeds but browser fails	Browser-level issue or proxy
`dig @8.8.8.8` works but `dig` alone fails	Local DNS resolver misconfigured

Resolver Configuration

```
$ cat /etc/resolv.conf   # Linux: which DNS server are you using?
nameserver 127.0.0.53

$ scutil --dns            # macOS: /etc/resolv.conf is NOT used for DNS
```

Private IPs and NAT

Private Address Ranges

Most devices never get a public IP; instead they use private addresses valid only within their local network:

Range	Typical Use
`10.0.0.0/8`	Large corporate networks
`172.16.0.0/12`	Medium networks
`192.168.0.0/16`	Home / small office

Is the Service Actually Running?

Before You Blame the Network

You will have to ask on the project is the service listening at all? The `ss` (socket statistics) command answers this which is the modern replacement for `netstat` on Linux.

Is the Service Actually Running?

Before You Blame the Network

You will have to ask on the project is the service listening at all? The `ss` (socket statistics) command answers this which is the modern replacement for `netstat` on Linux.

Linux Invocation

```
sudo ss -tulnp
```

Flag	Meaning
`-t`	TCP sockets
`-u`	UDP sockets
`-l`	Listening only
`-n`	Numeric (no hostname resolution)
`-p`	Show owning process

macOS: Use netstat or lsof Instead

macOS Commands

```
# Show all listening TCP/UDP ports (no process names)
sudo netstat -an -p tcp | grep LISTEN
sudo netstat -an -p udp

# Show listening ports WITH process names (closest to ss -tulnp)
sudo lsof -iTCP -sTCP:LISTEN -nP
```

Reading ss Output

Example Output

```
State   Recv-Q  Send-Q  Local Address:Port   Peer Address:Port  Process
LISTEN  0       128     0.0.0.0:22           0.0.0.0:*          sshd (pid=870)
LISTEN  0       511     0.0.0.0:80           0.0.0.0:*          nginx (pid=123
LISTEN  0       128     127.0.0.1:5432       0.0.0.0:*          postgres (pid=
```

Reading ss Output

Example Output

```
State   Recv-Q  Send-Q  Local Address:Port   Peer Address:Port  Process
LISTEN  0       128     0.0.0.0:22           0.0.0.0:*          sshd (pid=870)
LISTEN  0       511     0.0.0.0:80           0.0.0.0:*          nginx (pid=1234
LISTEN  0       128     127.0.0.1:5432       0.0.0.0:*          postgres (pid=9
```

Interpreting Local Address

Local Address	Meaning
`0.0.0.0:<port>`	Listening on all IPv4 interfaces
`[::]:<port>`	Listening on all IPv6 interfaces
`127.0.0.1:<port>`	Local machine only, not reachable from outside
`<specific IP>:<port>`	That interface only

Time to Explore: Checking Your Own Machine

Challenge

1. Run `sudo ss -tulnp` (Linux) or `sudo lsof -iTCP -sTCP:LISTEN -nP` (macOS) and identify one listening service
2. What address is it bound to? Could you reach it from another machine?
3. Run `ip addr show` (Linux) or `ifconfig` (macOS): how many interfaces do you have?
4. Run `ping 127.0.0.1` then `ping 8.8.8.8`. What do you observe?

What Is HTTP?

HyperText Transfer Protocol

HTTP is the L7 protocol behind the web. A client sends a request; the server sends a response.

- Client: browser, `curl`, your app
- Response: HTML, JSON, a file, or an error

What Is HTTP?

HyperText Transfer Protocol

HTTP is the L7 protocol behind the web. A client sends a request; the server sends a response.

- Client: browser, `curl`, your app
- Response: HTML, JSON, a file, or an error

HTTPS

HTTPS = HTTP + TLS. The padlock means traffic is encrypted and the server's certificate has been verified.

HTTP: Request and Response

Part	Example
Method	`GET`, `POST`, `PUT`, `DELETE`
Path	`/api/users/42`
Headers	`Authorization:`, `Content-Type:`
Body	JSON payload (optional)

Part	Example
Status Code	200, 404, 500
Headers	`Content-Type:`, `Location:`
Body	HTML, JSON, image bytes

HTTP: Request and Response

Part	Example
Method	`GET`, `POST`, `PUT`, `DELETE`
Path	`/api/users/42`
Headers	`Authorization:`, `Content-Type:`
Body	JSON payload (optional)

Part	Example
Status Code	200, 404, 500
Headers	`Content-Type:`, `Location:`
Body	HTML, JSON, image bytes

Common Methods

`GET` (read), `POST` (create), `PUT` (replace), `DELETE` (remove), `HEAD` (headers only)

HTTP Status Codes

Status Code Reference

Range	Class	Common Codes
2xx	Success	200 OK, 201 Created, 204 No Content
3xx	Redirect	301 Permanent, 302 Temporary, 304 Not Modified
4xx	Client Error	400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found
5xx	Server Error	500 Internal Error, 502 Bad Gateway, 503 Service Unavailable

HTTP Status Codes

Status Code Reference

Range	Class	Common Codes
2xx	Success	200 OK, 201 Created, 204 No Content
3xx	Redirect	301 Permanent, 302 Temporary, 304 Not Modified
4xx	Client Error	400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found
5xx	Server Error	500 Internal Error, 502 Bad Gateway, 503 Service Unavailable

Think-Pair-Share: What Code Is It?

You call `DELETE /api/posts/5` and the post is deleted. The server has nothing to return. What status code is correct? 204 No Content

Tool: curl

Making HTTP Requests From the Terminal

`curl` is a command-line HTTP client that shows you the raw request and response without a browser in the way.

Tool: curl

Making HTTP Requests From the Terminal

`curl` is a command-line HTTP client that shows you the raw request and response without a browser in the way.

What `curl -v` Shows You

`curl -v` reveals each step of the connection:

Step	What It Confirms
`Trying ...`	DNS resolved to an IP
`Connected to ...`	TCP connection established
`SSL connection using ...`	TLS handshake succeeded
`> ...`	Request headers sent
`< HTTP/1.1 200 OK`	Server responded with status

curl Examples

Common curl Patterns

```
# Simple GET request
$ curl https://api.example.com/users

# Verbose output (see full HTTP conversation)
$ curl -v https://api.example.com/users

# POST with JSON body
$ curl -X POST https://api.example.com/users \
  -H "Content-Type: application/json" \
  -d '{"name": "Alice", "email": "alice@example.com"}'

# Include response headers in output
$ curl -i https://api.example.com/users

# Only fetch headers (uses HEAD method)
```

General Workflow

Flowchart