Web

- Client-server architecture (request/reply oriented)
- Domain Name System (DNS) ≙ server that maps names to IP addresses
- Client ≙ make queries on behalf of users
- Server ≙ await and respond to queries, serve many clients
- Frameworks support client-side and server-side app development
- Peer-to-peer ≙ each participant is both a client and a server

TCP / IP

- IPv4 identity a physical network with four octets
  - Octet = 8 bit byte
  - 127.0.0.1 ≙ localhost (even if not connected to internet)
- IP ≙ no-guarantee, best-effort service that delivers packets from one IP to another
- TCP ≙ make IP reliable by detecting 'dropped' packets, data arriving out of order, transmission errors, slow networks, etc.
- TCP ports allow multiple TCP apps on the same computer

Routing

- Route ≙ HTTP method + URI
  - eg: GET http://srch.com/main/search?q=cloud&lang=en#top
  - GET ≙ method
  - http ≙ protocol
  - srch.com ≙ host:port
  - /main/search ≙ path
  - ?q=cloud&lang=en ≙ optional query params
  - #top ≙ optional fragment
- Types:
  - HEAD ≙ get metadata
  - GET ≙ get data
  - POST ≙ send data
  - DELETE
- If you fetch a webpage, relative links cannot be loaded since you only fetch that specific page not the entire directory
  - Can use `curl` for this

Responses

- 2xx ≙ all is well
- 3xx ≙ resource moved
- 4xx ≙ access problem
- 5xx ≙ server error

Cookies

- Used because HTTP is stateless
  - Every HTTP request is independent of all prior requests BESIDES cookies
- Set in `set-cookie` header
  - Client is tasked with maintaining these cookies
  - Only on first visit to a page

Service-Oriented Architecture

- Web1 $\longrightarrow$ web2 $\triangleq$ JS could dynamically update the page without loading a new page
  - Done with AJAX (Async. JS and XML)
  - Noting is XML specific
- Microservice $\triangleq$ independently-deployable component that supports message-based communication
  - Often not designed to be called from 'top level'
  - May provide only part of a page (requires AJAX)
  - May introduce performance issues (b/c async loading)
  - Must manage partial failure
  - More interfaces to keep track of (but REST helps)
  - Devs have to learn about operations and vice versa

RESTful APIs

- Callee $\triangleq$ endpoint (base URI + path)
- Operations $\triangleq$ path portion of URI
- How args are passed $\triangleq$ part of URI or JSON/XML payload
- How values are returned $\triangleq$ JSON or XML data structure (can technically be anything)
- How errors are indicated $\triangleq$ HTTP status codes and returned data structure if it contains error message
- REST $:=$ representational state transfer
  - Canonical way of mapping URIs for remote procedure calls
  - Everything the server manages is a resource
  - Actions done on resource $\triangleq$ {create, read, update, delete, index}
  - Side effects always use POST or PUT

JSON (JavaScript Object Notation)

- Primitive types: string, numeric, array, hash
- Usually API response or payload is a single top-level JSON object with well-defined slots

Display Tiers

- Presentation $\triangleq$ front-end layer
- Logic $\triangleq$ core capabilities
- Data $\triangleq$ storage