Part I Outline

- Module 1: Introduction to IoT
  - A Motivating Example
  - IoT Concept
  - IoT Trend
  - IoT Applications
  - IoT Application Enablers
  - IoT Challenges
- Module 2: General Network Architecture for IoT
  - IoT Network Architecture
  - **3-Layer Model:** Functional Stack, **Compute Stack**
  - Cloud Computing
  - Communications with Cloud: HTTP, REST, CoAP, MQTT
- Module 3: IoT Devices
  - Sensors and Actuators
  - Connected Smart Objects
  - IP as the IoT Network Layer
  - Information Acquisition

Venues for Computing

IoT Data Management and Compute Stack

- In most cases, the processing location is outside the smart object. A natural location for this processing activity is the cloud. Smart objects need to connect to the cloud, and data processing is centralized

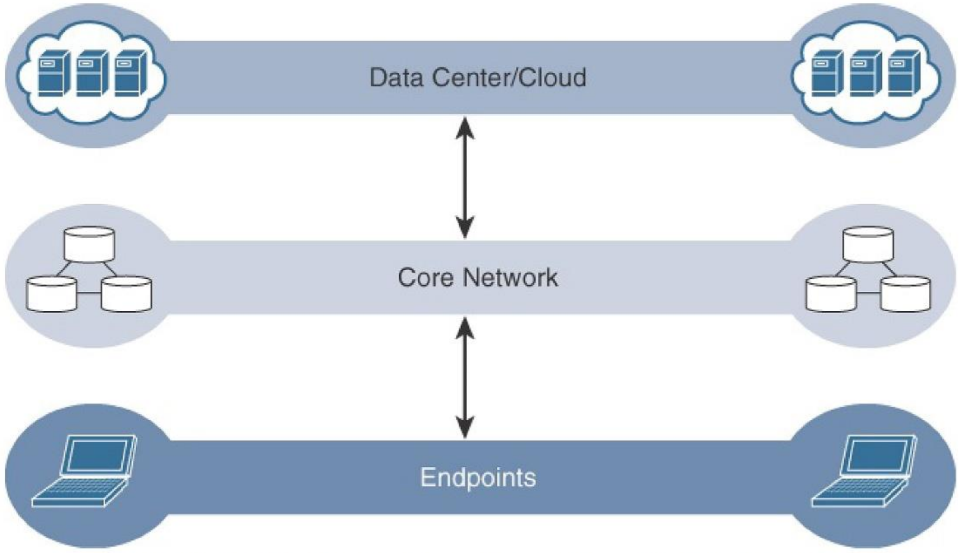


Figure 2-14 *The Traditional IT Cloud Computing Model*

Layer 1: Edge Computing

- Edge Computing
  - Provide computing capability close to IoT devices

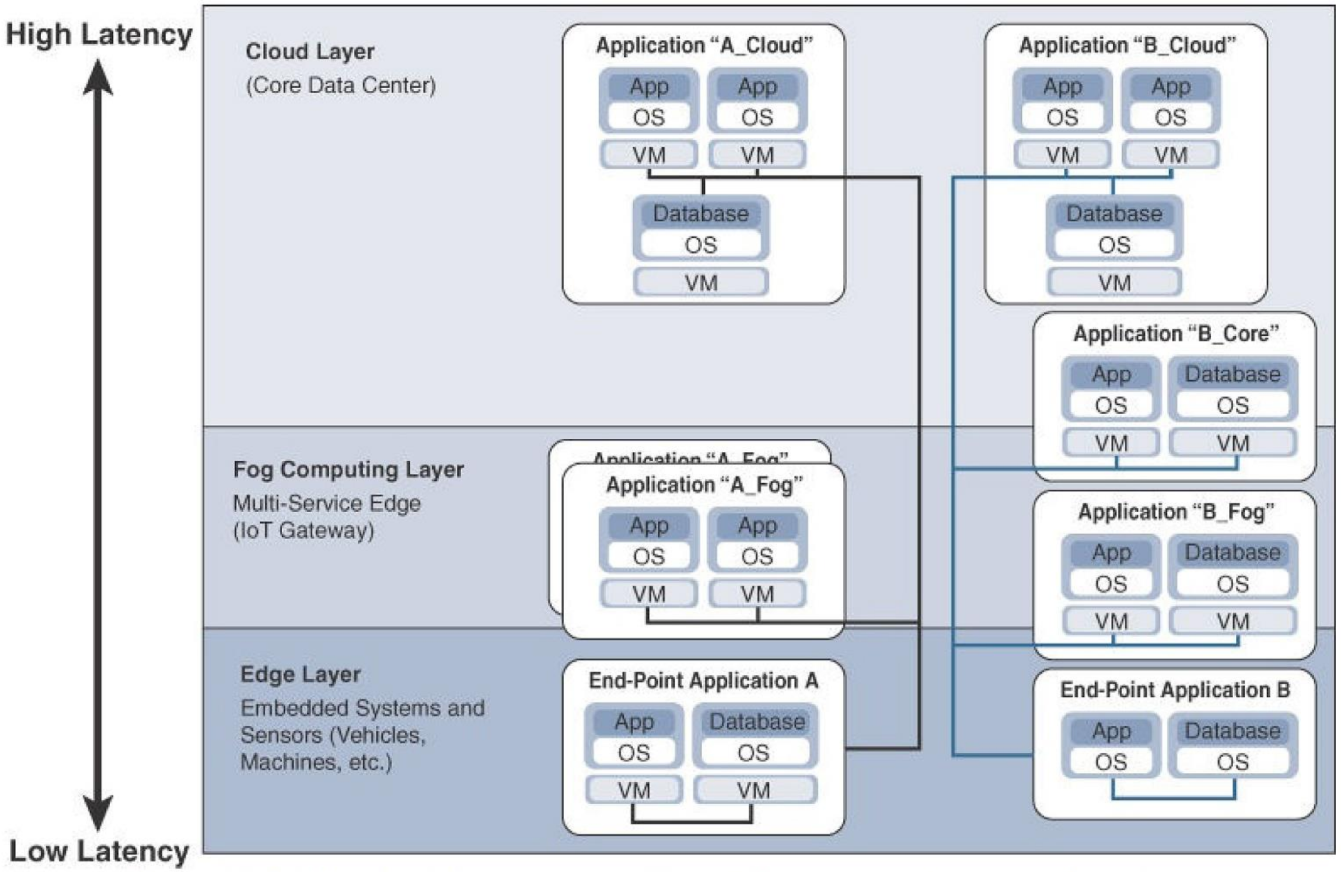


Figure 2-16 *Distributed Compute and Data Management Across an IoT System*

Layer 1: Edge Computing (cont'd)

- Most time-sensitive data is analyzed on the edge nodes
  - Smart home sensor detects extremely high temperature and smoke in a house, indicating fire alarm
- Data can wait seconds or minutes for action is passed along to aggregation node for analysis and action
  - Adjust temperature when there are more/fewer people in the house
- Data that is less time sensitive is sent to the cloud for historical analysis, big data analytics, and long-term storage
  - Historical data of temperature from sensors is analyzed to adjust operations of heating system and air condition system to reduce energy usage and maximize comfortability

Layer 2: Fog Computng

- Fog Computing
  - Any device with computing, storage, and network connectivity can be a fog node (controllers, switches, routers, embedded servers, and IoT gateways)
  - Analyzing IoT data close to where it is collected minimizes latency, offloads gigabytes of network traffic from the core network, and keeps sensitive data inside the local network

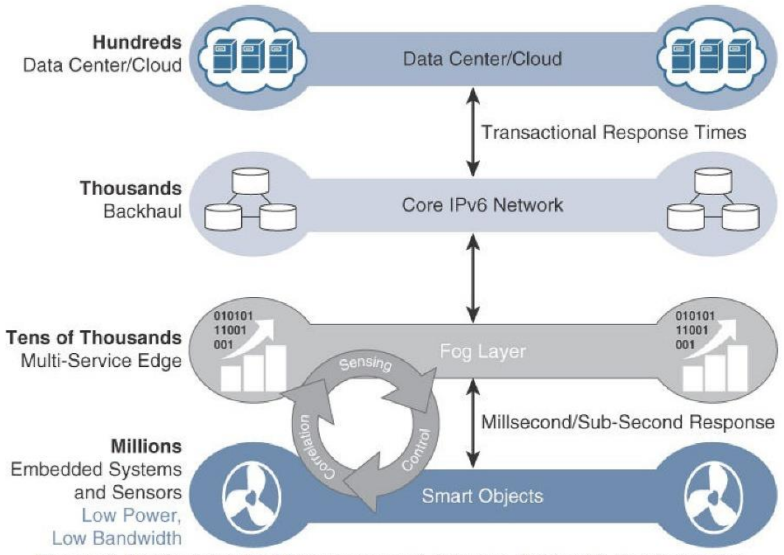


Figure 2-15 *The IoT Data Management and Compute Stack with Fog Computing*

Layer 2: Fog Computing (cont'd)

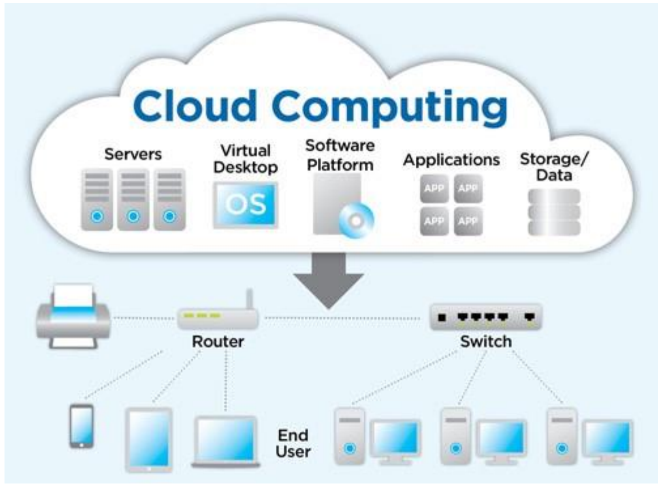
- Fog Computing: Benefits
  - Contextual location awareness and low latency
  - Geographic distribution (distributed structure)
  - Deployment near IoT endpoints
  - Wireless communication between the fog and the IoT endpoint
  - Use for real-time interactions
    - Important fog applications involve real-time interactions rather than batch processing. Preprocessing of data in the fog nodes allows upper-layer applications to perform batch processing on a subset of the data

Part I Outline

- Module 1: Introduction to IoT
  - A Motivating Example
  - IoT Concept
  - IoT Trend
  - IoT Applications
  - IoT Application Enablers
  - IoT Challenges
- Module 2: General Network Architecture for IoT
  - IoT Network Architecture
  - 3-Layer Model: Functional Stack, Compute Stack
  - **Cloud Computing**
  - Communications with Cloud: HTTP, REST, CoAP, MQTT
- Module 3: IoT Devices
  - Sensors and Actuators
  - Connected Smart Objects
  - IP as the IoT Network Layer
  - Information Acquisition

What is Cloud Computing?

- Delivery of computing **services**
  - servers
  - storage
  - analytics
  - databases
  - networking
  - and much more...

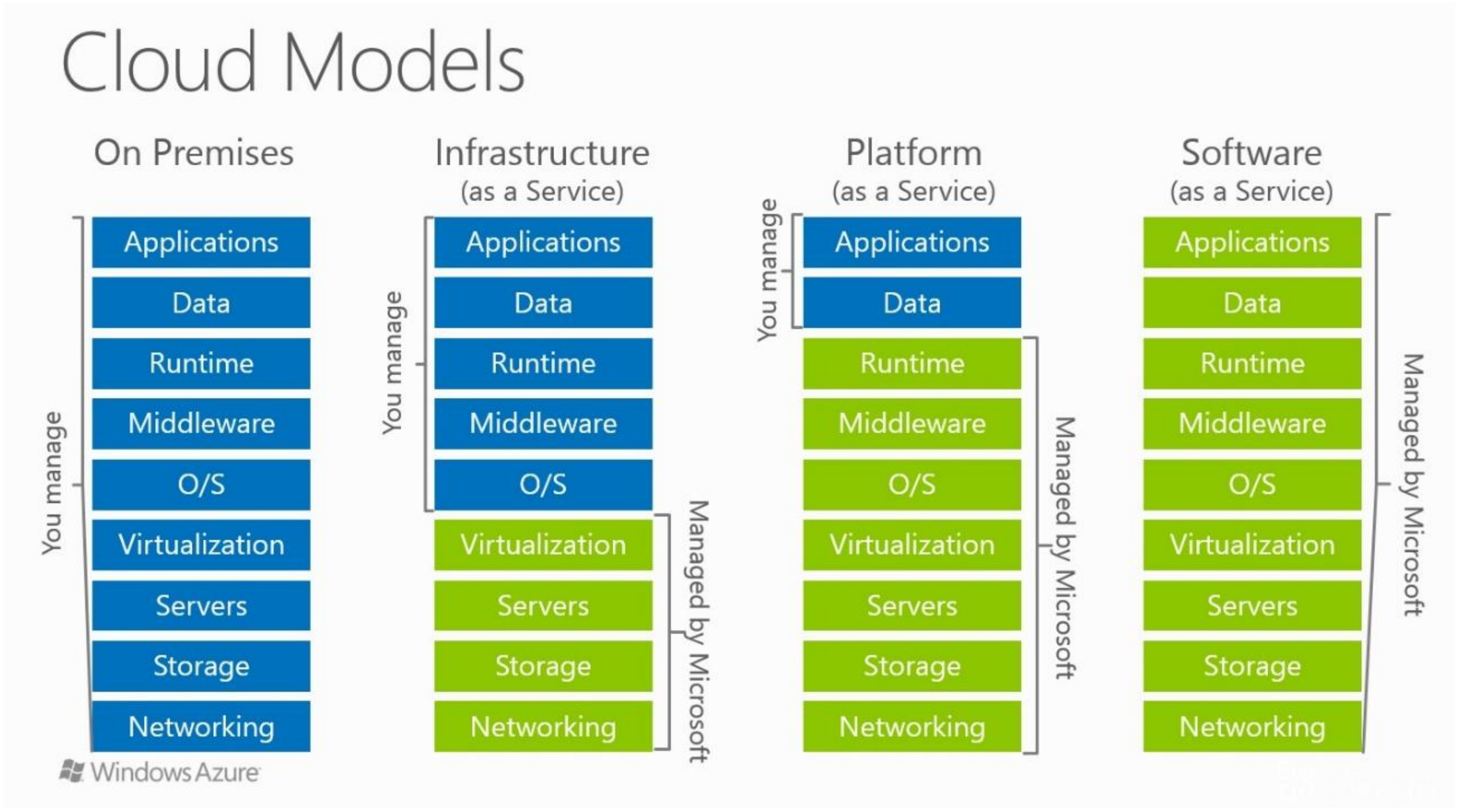


- Another definition: network-based computing taking place over the Internet, while hiding complexity of underlying infrastructure using simple APIs

Advantages of Cloud Computing

- Advantages:
  - New applications
  - Anytime/anywhere access
  - Homogeneity
  - Virtualization
  - Resilient
  - Cost
  - Sharing, collaboration
  - Management/maintenance
  - Security
  - …

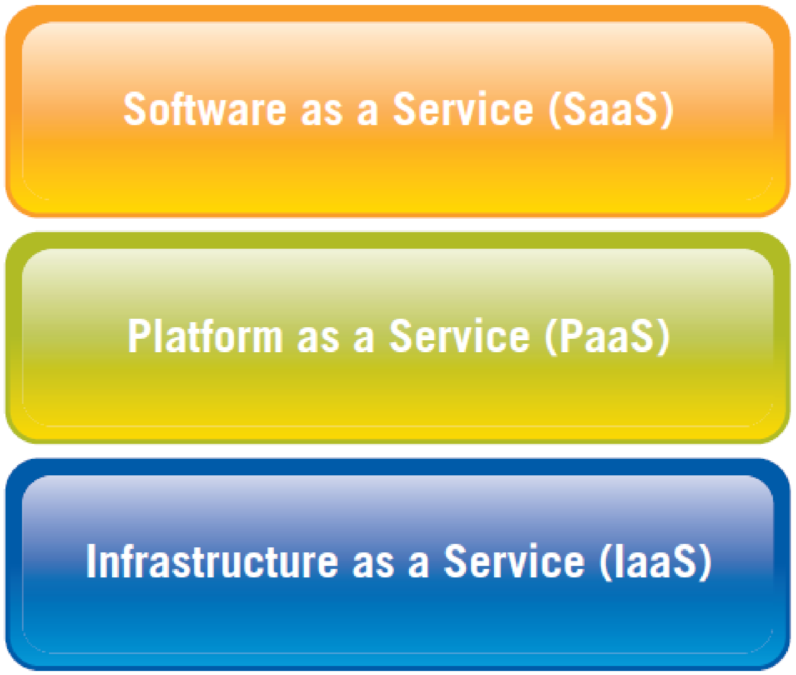
Cloud Models: IaaS, PaaS, SaaS



Terminology

- **Virtualization:** creation of a virtual resource such as a server, desktop, operating system, file, storage, or network
- **Middleware:** software that acts as a bridge between an operating system or database and applications, especially on a network
- **Runtime:** software designed to support the execution of computer programs

IaaS, PaaS, SaaS


Software as a Service (SaaS)
Platform as a Service (PaaS)
Infrastructure as a Service (IaaS)

Enduser application is delivered as a service. Platform and infrastructure is abstracted, and can deployed and managed with less effort.

Application platform onto which custom applications and services can be deployed. Can be built and deployed more inexpensively, although services need to be supported and managed.

Physical infrastructure is abstracted to provide computing, storage, and networking as a service, avoiding the expense and need for dedicated systems.

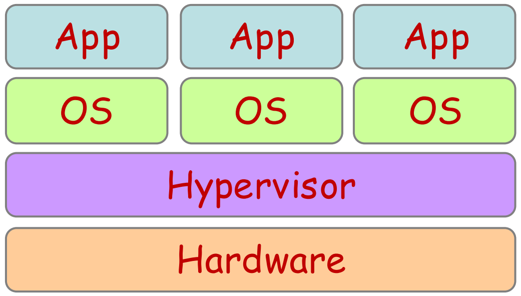
- Simple examples
  - IaaS: barebones computer
  - PaaS: computer + OS (incl. development environment)
  - SaaS: complete solution including application(s)

Products

- IaaS: Amazon Web Services (AWS), Microsoft Azure, Google Compute Engine
- PaaS: Google App Engine, Heroku, OpenShift, AWS Elastic Beanstalk
- SaaS: Google Apps, Dropbox, Cisco Webex, Salesforce, Concur, GoToMeeting

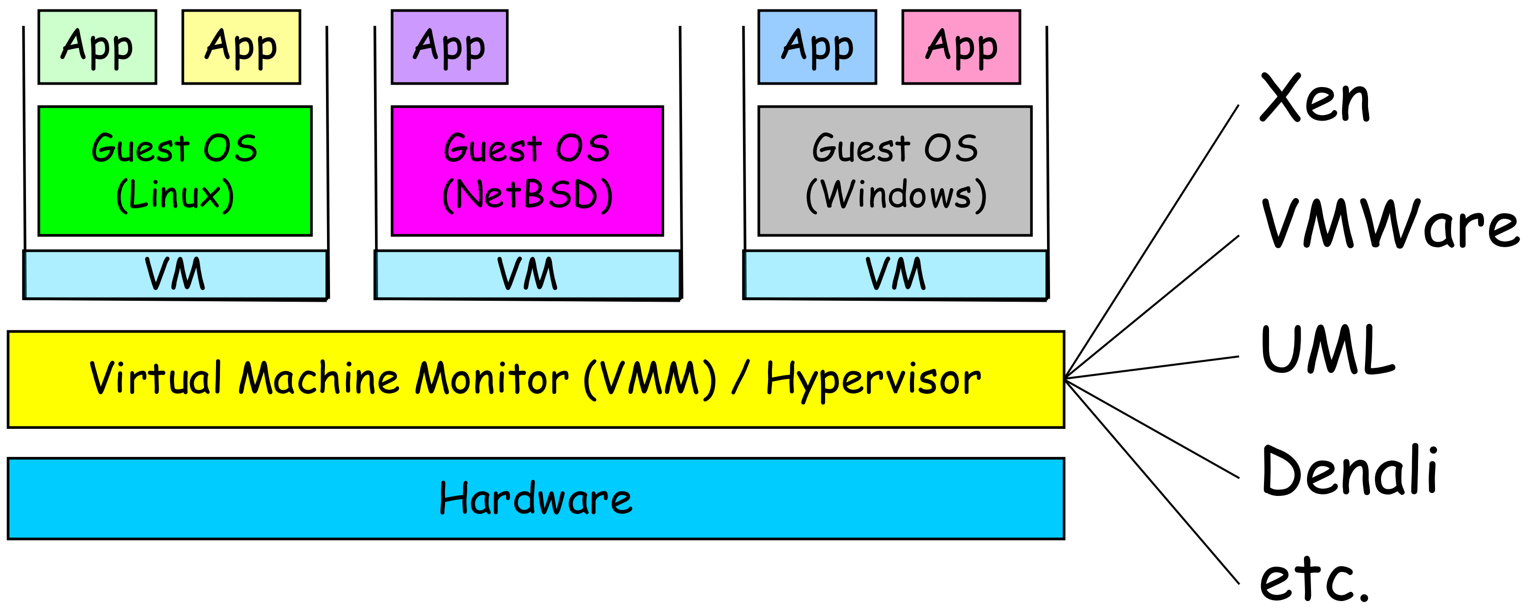
Virtualization

- Virtual workspaces:
  - An abstraction of an execution environment that can be made dynamically available to authorized clients by using well-defined protocols
  - Resource quota (e.g., CPU, memory share)
  - Software configuration (e.g., OS, provided services)
- Implemented on Virtual Machines (VMs):
  - Abstraction of a physical host machine
  - Hypervisor intercepts and emulates instructions from VMs, and allows management of VMs
  - VMWare, Xen, etc.



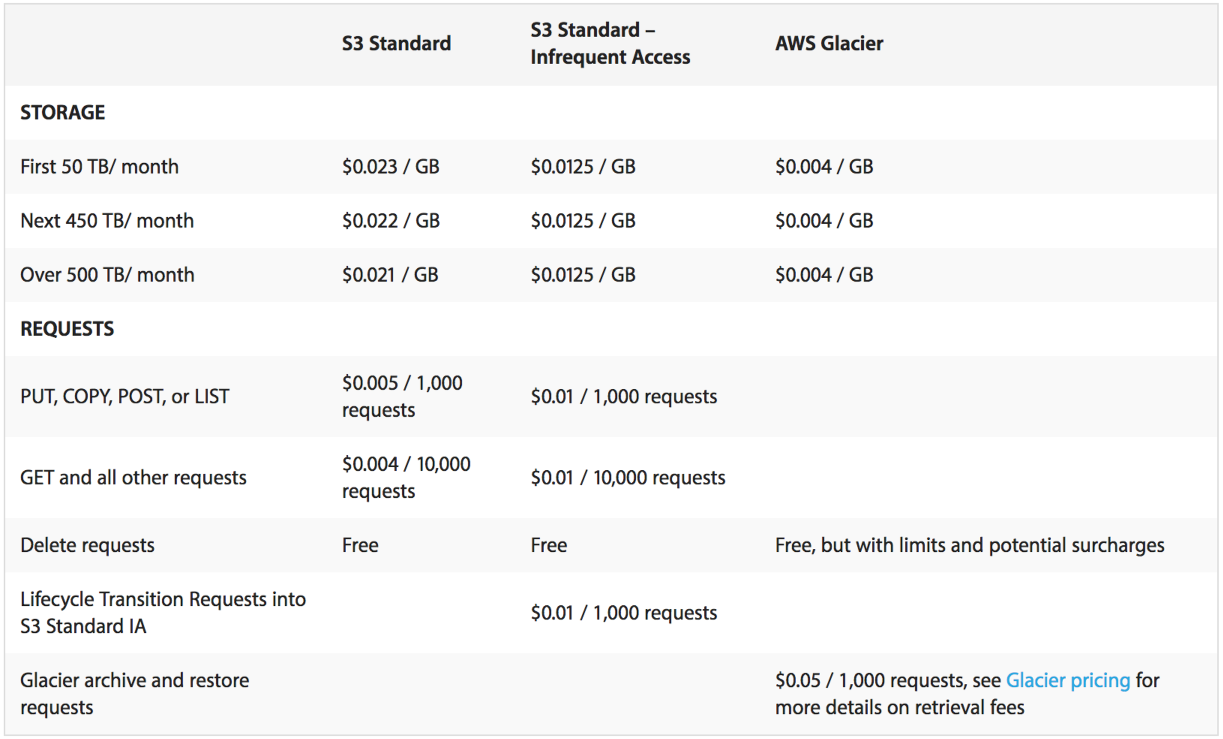
Virtualized Stack

Virtual Machines



Cloud Example: S3

- Amazon Simple Storage Service (S3)
- Unlimited storage
- Pay for what you use



	S3 Standard	S3 Standard – Infrequent Access	AWS Glacier
STORAGE			
First 50 TB/ month	$0.023 / GB	$0.0125 / GB	$0.004 / GB
Next 450 TB/ month	$0.022 / GB	$0.0125 / GB	$0.004 / GB
Over 500 TB/ month	$0.021 / GB	$0.0125 / GB	$0.004 / GB
REQUESTS			
PUT, COPY, POST, or LIST	$0.005 / 1,000 requests	$0.01 / 1,000 requests	
GET and all other requests	$0.004 / 10,000 requests	$0.01 / 10,000 requests	
Delete requests	Free	Free	Free, but with limits and potential surcharges
Lifecycle Transition Requests into S3 Standard IA		$0.01 / 1,000 requests	
Glacier archive and restore requests			$0.05 / 1,000 requests, see Glacier pricing for more details on retrieval fees

Cloud Example: EC2

- Amazon Elastic Compute Cloud (EC2)
  - Virtual computing environments (“instances”)
  - Pre-configured templates for instances
  - Launch as many virtual servers as needed (“elastic”)
  - Xen and KVM hypervisor

Do You Use Clouds?

Clouds for IoT

AWS IoT

ThingSpeak

Samsung ARTIK™ Cloud

Microsoft Azure IoT Platform

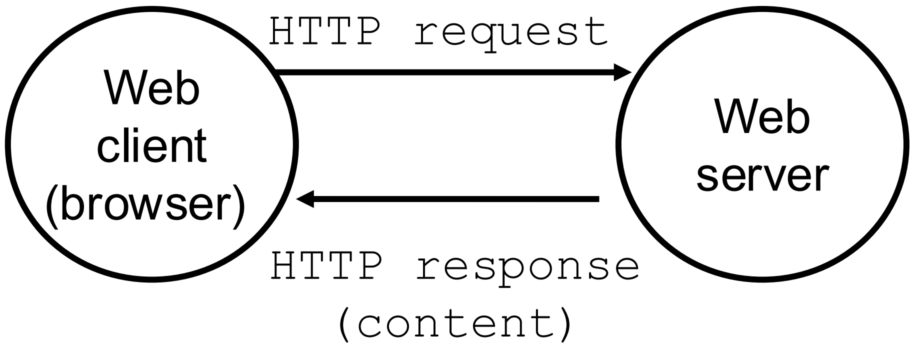
IBM Watson IoT™

Part I Outline

- Module 1: Introduction to IoT
  - A Motivating Example
  - IoT Concept
  - IoT Trend
  - IoT Applications
  - IoT Application Enablers
  - IoT Challenges
- Module 2: General Network Architecture for IoT
  - IoT Network Architecture
  - 3-Layer Model: Functional Stack, Compute Stack
  - Cloud Computing
  - **Communications with Cloud: HTTP**, REST, CoAP, MQTT
- Module 3: IoT Devices
  - Sensors and Actuators
  - Connected Smart Objects
  - IP as the IoT Network Layer
  - Information Acquisition

Hypertext Transfer Protocol (HTTP)

- Clients and servers communicate using the **HyperText Transfer Protocol (HTTP)**
  - Client and server establish TCP connection
  - Client requests content
  - Server responds with requested content
  - Client and server close connection (usually)



Web Content

- Web servers return **content** to clients
  - a sequence of bytes with an associated MIME (Multipurpose Internet Mail Extensions) type
- Example MIME types

– `text/html`	HTML document
– `text/plain`	Unformatted text
– `application/postscript`	Postscript document
– `image/gif`	Binary image encoded in GIF format
– `image/jpeg`	Binary image encoded in JPEG format

Static & Dynamic Content

- The content returned in HTTP responses can be either **static** or **dynamic**
  - Static content: content stored in files and retrieved in response to an HTTP request
    - Examples: HTML files, images, audio clips
  - Dynamic content: content produced on-the-fly in response to an HTTP request
    - Example: content produced by a program executed by the server on behalf of the client
- Bottom line: all web content is associated with a **file** that is managed by the server

URLs

- Each file managed by a server has a unique name called a URL (Universal Resource Locator)
- URLs for static content:
  - `http://www.cse.nd.edu:80/index.html`
  - `http://www.cse.nd.edu/index.html`
  - `http://www.cse.nd.edu`
    - Identifies a file called `index.html`, managed by a web server at `www.cse.nd.edu` that is listening on port 80
- URLs for dynamic content:
  - `http://www.cse.nd.edu:8000/cgi-bin/adder?15000&213`
    - Identifies an executable file called `adder`, managed by a web server at `www.cse.nd.edu` that is listening on port 8000, that should be called with two argument strings: `15000` and `213`

HTTP Transaction

`unix> telnet www.aol.com 80`	*Client: open connection to server*
`Trying 205.188.146.23...`	*Telnet prints 3 lines to the terminal*
`Connected to aol.com.`	
`Escape character is '^]'.`	
`GET / HTTP/1.1`	*Client: request line*
`host: www.aol.com`	*Client: required HTTP/1.1 HOST header*
	Client: empty line terminates headers.
`HTTP/1.0 200 OK`	*Server: response line*
`MIME-Version: 1.0`	*Server: followed by five response headers*
`Date: Mon, 08 Jan 2001 04:59:42 GMT`	
`Server: NaviServer/2.0 AOLserver/2.3.3`	
`Content-Type: text/html`	*Server: expect HTML in the response body*
`Content-Length: 42092`	*Server: expect 42,092 bytes in the resp body*
	Server: empty line (“\r\n”) terminates hdrs
`<html>`	*Server: first HTML line in response body*
`...`	*Server: 766 lines of HTML not shown.*
`</html>`	*Server: last HTML line in response body*
`Connection closed by foreign host.`	*Server: closes connection*
`unix>`	*Client: closes connection and terminates*

HTTP Request

- HTTP request is a *request line*, followed by zero or more *request headers*

- Request line: `<method> <uri> <version>`
  - `<version>` is HTTP version of request (`HTTP/1.0` or `HTTP/1.1`)
  - `<uri>` is typically URL for proxies, URL suffix for servers
  - `<method>` is either `GET`, `POST`, `OPTIONS`, `HEAD`, `PUT`, `DELETE`, or `TRACE`

HTTP Request (cont'd)

- HTTP methods:
  - `GET`: Retrieve static or dynamic content
    - Arguments for dynamic content are in URI
    - Workhorse method (99% of requests)
  - `POST`: Retrieve dynamic content
    - Arguments for dynamic content are in the request body
  - `OPTIONS`: Get server or file attributes
  - `HEAD`: Like `GET` but no data in response body
  - `PUT`: Write a file to the server
  - `DELETE`: Delete a file on the server
  - `TRACE`: Echo request in response body
    - Useful for debugging

HTTP Response

- HTTP response is a *response line* followed by zero or more *response headers*
- Response line:
- `<version> <status code> <status msg>`
  - <version> is HTTP version of the response
  - <status code> is numeric status
  - <status msg> is corresponding English text
    - 200 OK Request was handled without error
    - 403 Forbidden Server lacks permission to access file
    - 404 Not found Server couldn't find the file
- Response headers: `<header name>: <header data>`
  - Provide additional information about response
  - `Content-Type:` MIME type of content in response body
  - `Content-Length:` Length of content in response body