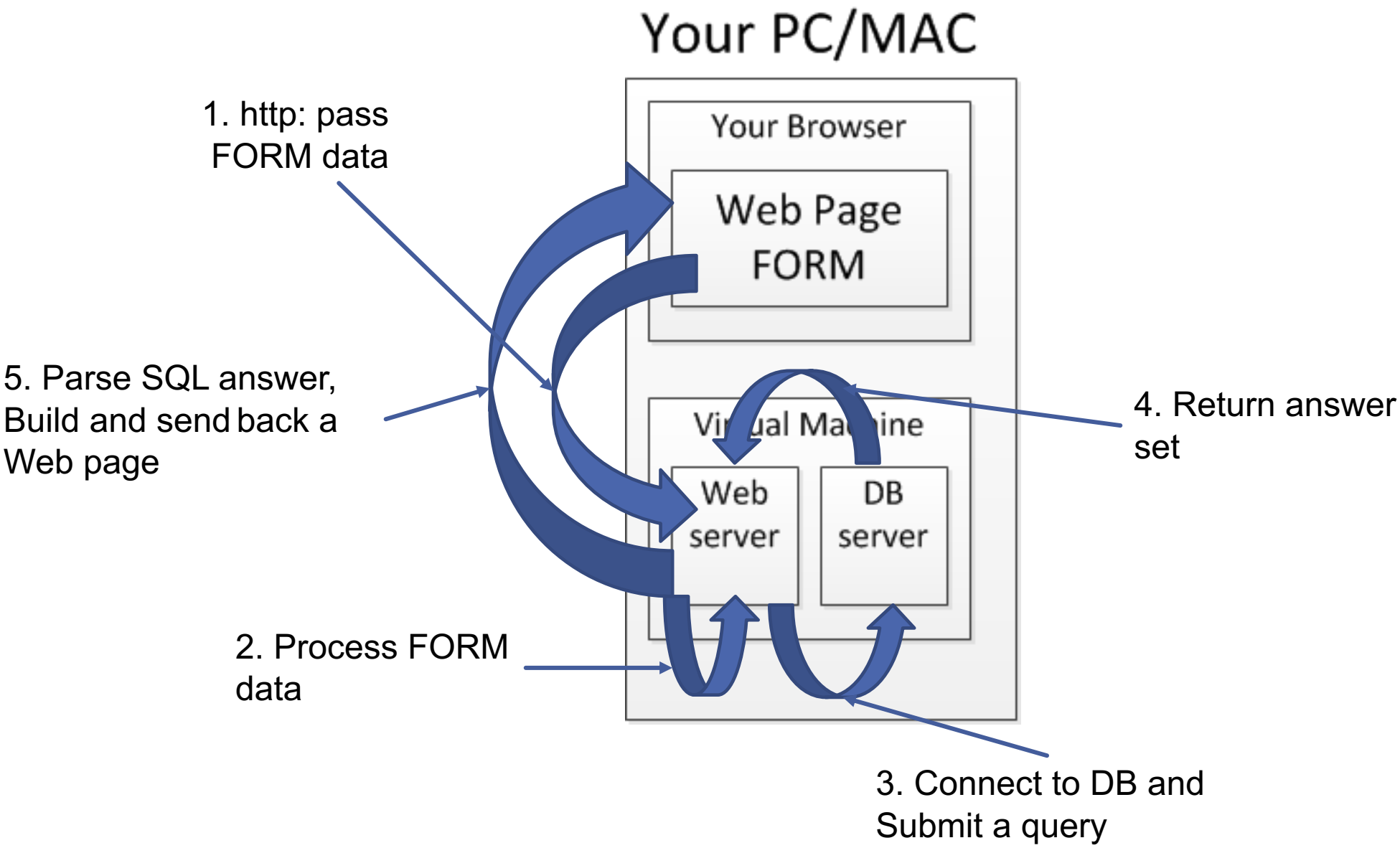
MIDTERM 2 REVIEW

WEEK 13 – 11/18/2019


Your PC/MAC
Your Browser
Web Page
FORM
Virtual Machine
Web
server
DB
server
1. http: pass
FORM data
2. Process FORM
data
3. Connect to DB and
Submit a query
4. Return answer
set
5. Parse SQL answer,
Build and send back a
Web page

NODE JS BASICS

NodeJS can process HTTP requests from your browser:

- Node.js code can generate dynamic page content – it *creates* the HTML on the fly
- Node.js code can create, open, read, write, delete, and close files on the server
- Node.js code can collect and process form data from an HTML page
- Node.js code can read, add, change, delete data in your database

HTML FORMS

- How are they used?
  - Use the browser's window as a data entry screen
  - Collect information from the user
  - Pass it to the web server via http
  - Invoke a server-side script
  - Passes ***form data*** as input to the script
  - Script on server parses out the form data

HTML FORMS

- `<form>` tag has several attributes – two are required
- ACTION
  - `<form action="http://URL">` name of a program on the web server
    - URL specifies the location of the executable file on the web server
  - `<form action="mailto:mailrecipient">` sends an email
- METHOD
  - `<form method="POST" >` or `<form method="GET">`
    - **POST** when you have large amount of data being sent, encryption available, a two-step process
    - **GET** for small amounts, no security – all in one step

      `<form enctype=`
      - multipart/form-data (default)
      - text/plain (used only for mailto)

NODEJS CONCEPTS

- NodeJS programs handle HTTP: calls from the client
  - Two types of calls:
    - req – the HTTP: request coming from the client
    - res – the HTTP: response being sent back to the client
- ROUTING
  - How your NodeJS application code responds to the client request
  - Defined by blocks/sections of code within your NodeJS program (will illustrate with code demos)
  - Each section handles a URL and a method (GET or POST)

ROUTING

- For example, a block of code may do
  - `app.get(/)` = process a get request for the "index" html file
  - `app.post(/)` = process a post request for the "index" html file
  - `app.get(/URL...)` = process a get request for the file "URL"
  - `app.post(/URL...)` = process a get request for the file "URL"

- Further reading: https://expressjs.com/en/guide/routing.html

INTEGRATING UI AND DB USING NODEJS

Demonstration of Integration using a NodeJS program `HandleForm.js` with Express framework routing

- Program Steps:
  1. Does a res.send to send the HTML form page to the browser
  2. Does a app.get to receive the HTTP GET from the form, passing back a PlayerId when SUBMIT is pressed in the browser
  3. Builds a database connection string
  4. Connects to the database
  5. Runs a query getting the row for that PlayerId
  6. Parse out the results
  7. Build a web page and send it back to the browser
  8. Leave the Node web server running, listening on port 8080

HTML FORM

```
<html>
   <body>
      <form action = "http://127.0.0.1:8080/process_get" method = "GET">
         <div align=center>
          <h1>PlayerId:</h1> <input type = "text" name = "playerId">  <br><br>
          <input type = "submit" value = "Submit">
          </div>
      </form>
   </body>
</html>
```

TYPES OF CLOUD NETWORKS

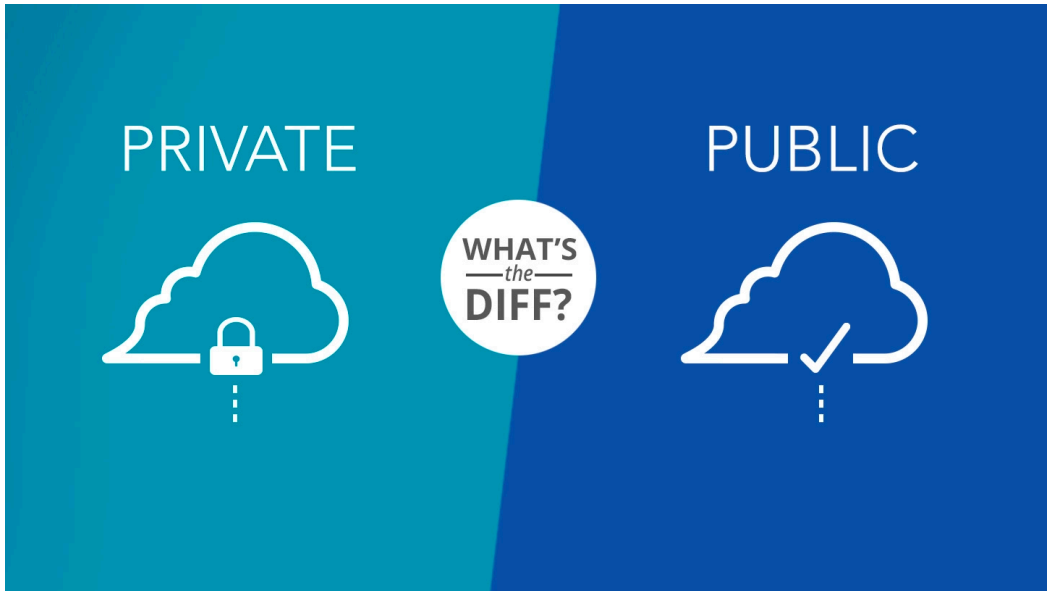
Private – my private cloud in my own data center

Public – a shared environment hosted by a provider

CHARACTERISTICS OF PUBLIC CLOUD SERVICES

- Offsite hosting
- Pay per use (setup/initial, plus ongoing)
- Shared space
- Massively Scalable
- On-Demand Provisioning
- Rapid Deployment
- Lowers innovation barriers
- Leading edge architecture

CHARACTERISTICS OF PRIVATE CLOUD SERVICES



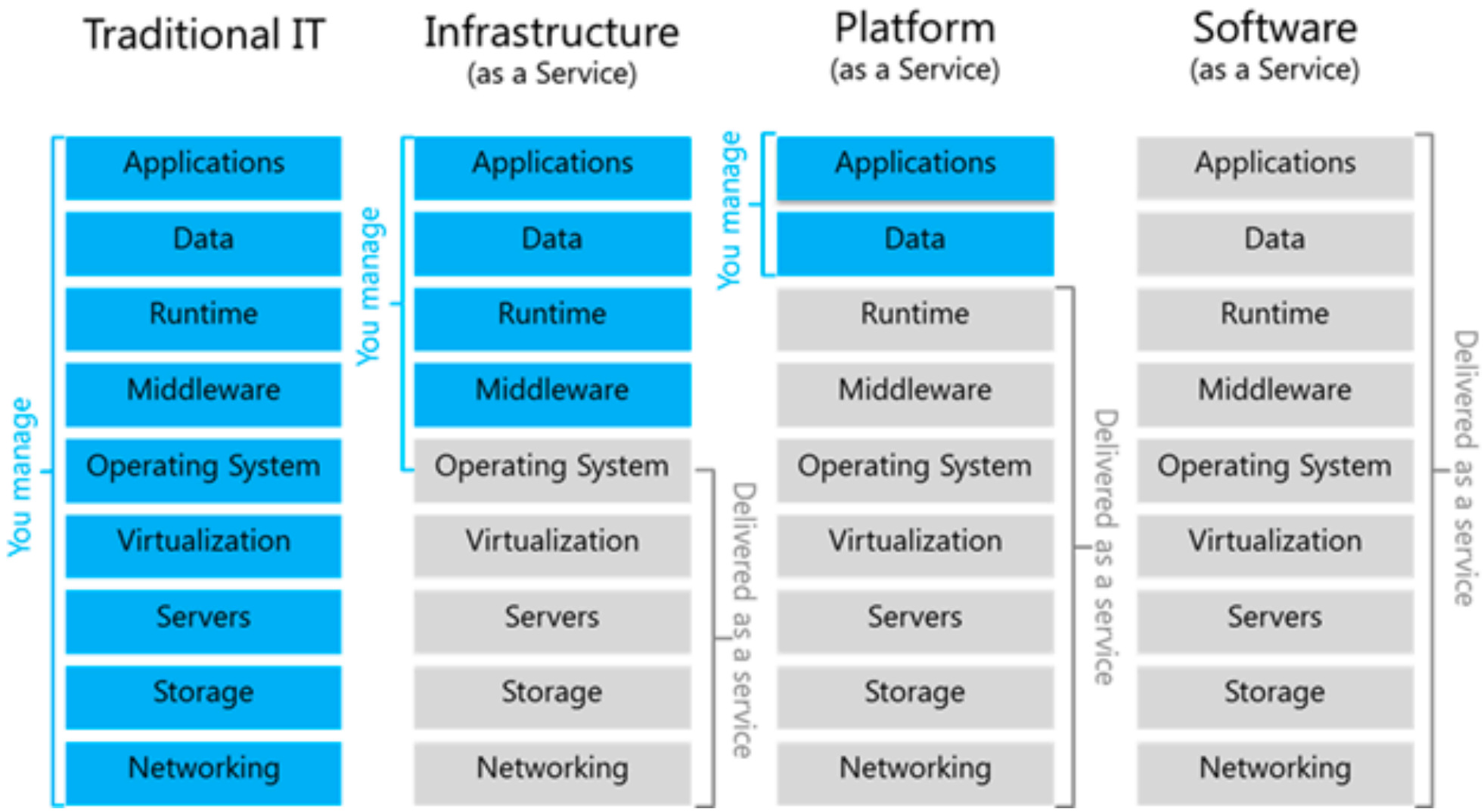
- Private: Leverage the advantages, with few disadvantages
  - Massively Scalable
  - On-Demand Provisioning
  - Rapid Deployment
  - More secure
  - Better Monitoring
  - BUT → Still requires significant internal infrastructure

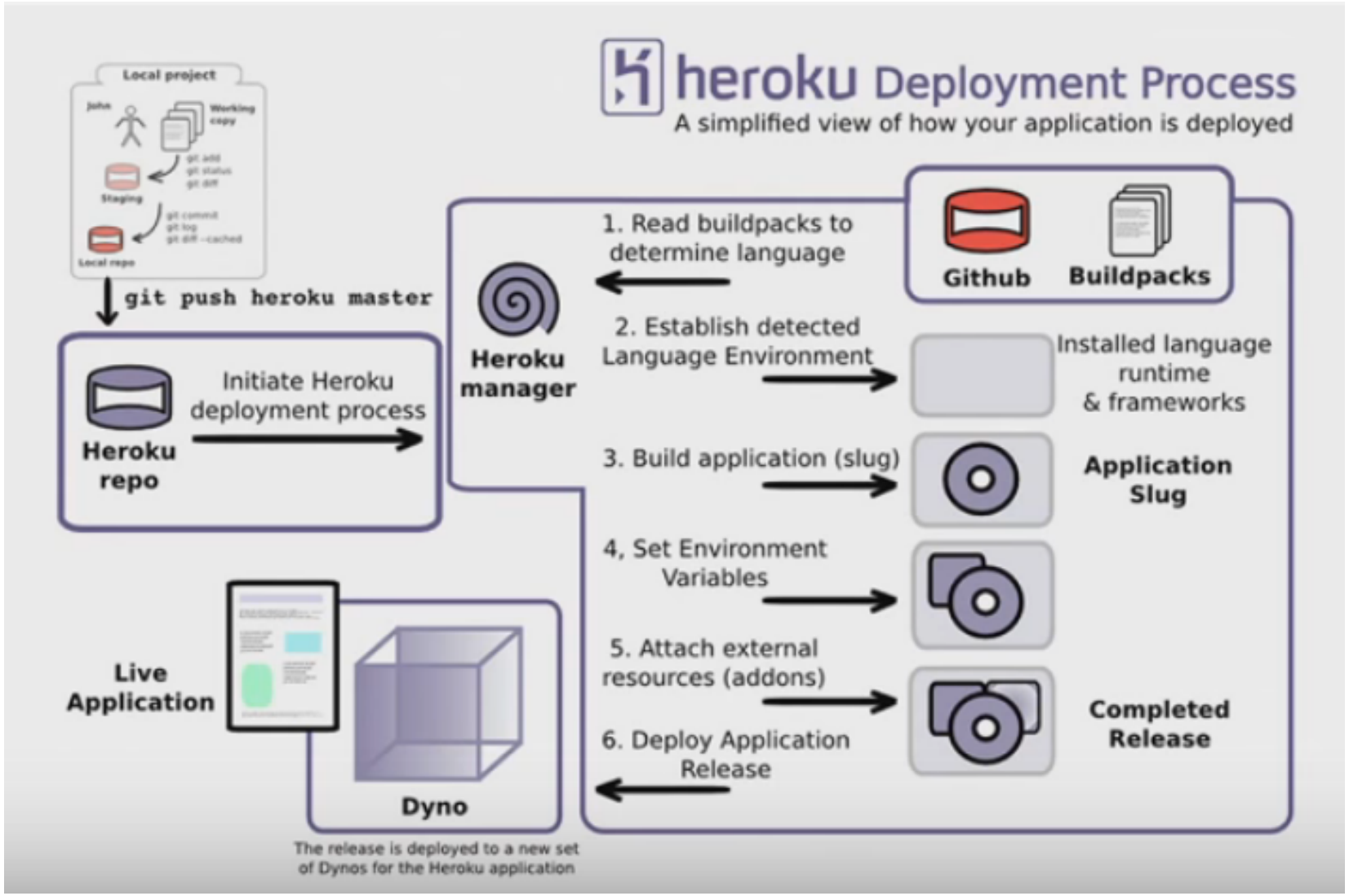
CLOUD SERVICE MODELS

- IaaS - Infrastructure-as-a-Service
  - A cloud service providing infrastructure - computers, networking resources, storage. Typically virtual, but could be could be physical.

- PaaS - Platform-as-a-Service
  - A cloud service that hides the infrastructure (users don't see the servers, storage, switches, etc.) Provides a software development platform. Users can develop and run an application on a PaaS: the system ensures the app has the necessary infrastructure to run and scale.

- SaaS - Software-as-a-Service
  - A cloud service providing users access to software in a self-service, on-demand fashion. This could be a single application or an entire suite.


Traditional IT
Infrastructure
(as a Service)
Platform
(as a Service)
Software
(as a Service)
Applications
Data
Runtime
Middleware
Operating System
Virtualization
Servers
Storage
Networking
You manage
Delivered as a service


heroku Deployment Process
A simplified view of how your application is deployed
Local project
John
Working copy
git add
git status
git diff
Staging
git commit
git log
git diff --cached
Local repo
git push heroku master
Heroku repo
Initiate Heroku deployment process
Heroku manager
1. Read buildpacks to determine language
Github
Buildpacks
2. Establish detected Language Environment
Installed language runtime & frameworks
3. Build application (slug)
Application Slug
4, Set Environment Variables
5. Attach external resources (addons)
6. Deploy Application Release
Completed Release
Live Application
Dyno
The release is deployed to a new set of Dynos for the Heroku application

CONTINUOUS INTEGRATION - FOLLOWS THE AGILE APPROACH

- Definition:
  - Integrate & build the system several times a day
  - Integrate every time a task is completed
- Lets you know every day the status of the full system
- Continuous integration and relentless testing go hand-in-hand.
- By keeping the system integrated at all times, you increase the chance of catching defects early and improving the quality and timeliness of your product.
- Continuous integration helps everyone see what is going on in the system at all times.

CONTINUOUS INTEGRATION

"Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily, leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible."

- Martin Fowler

https://en.wikipedia.org/wiki/Martin_Fowler

*If **testing** is good, why not do it all the time? (continuous testing)*
*If **integration** is good, why not do it several times a day? (continuous integration)*
*If **customer involvement** is good, why not show the business value and quality we are creating as we create it (continuous reporting)*

WHY DO CONTINUOUS INTEGRATION?

- Speed Up Each Feature/Sprint
- Automate Deployment
- Guarantee that Necessary Tests Are Being Run
- Track Build and Test Status
- Immediate Bug Detection
- Yields more Bug-Free Code
- A "Deployable" system at any given point
- Record of the history/evolution of the project

FOWLER'S 10 BEST PRACTICES FOR CI

1. Maintain a Single Source Repository
2. Automate the Build
3. Make your Build Self-testing
4. Everyone Commits Every day
5. Every Commit should Build the Mainline on an Integration Machine
6. Keep the Build Fast
7. Test in a Clone of the Production Environment
8. Make it easy for Anyone to get the Latest Executable
9. Everyone can see what's Happening
10. Automate Deployment

TWO PRIMARY TYPES OF DOCUMENTATION

- External to your code
  - A feature/design specification document
  - A "wiki" type of website, document repo
- Internal to your code
  - Comments
  - Good coding habits

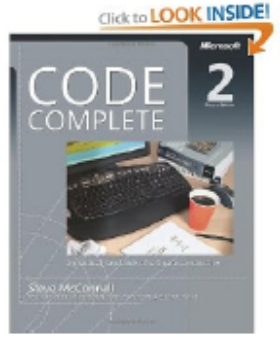
WHAT MUST WE DOCUMENT?

- What is the code doing to the data
- The meaning of variables
- Functions, methods, called procedures

"SELF-DOCUMENTING" CODE

- Program structure
- Variable Naming
- Class and Method Names
- Named Constants instead of Literals
- Minimized control flow
- Reduced data structure complexity

Self-Documenting Code



- Code should be written for humans
  - Compiler will keep the machine happy
- Quality Comments
  - Bad comments are worse than none at all!
- Naming Scheme
  - Make a name count!
- Coding Style
  - Decide on one and enforce its use
- Documentation Extraction Systems
  - JavaDoc, Doxygen, rdoc, etc…

Effective comments DO NOT repeat the code!

YOUR ORGANIZATION SHOULD DEFINE AND ENFORCE

- Standards for names
  - Variables – lower case, words separated by "_"
  - Methods/Functions – camelCase, no "_"
- Avoid numbers as differentiators
  - grade1, grade2, grade3
- Use a name that is self-explanatory – no comments needed
- Standard abbreviations
  - "dept" for "department"
  - "cust" for "customer"

HTTP – HYPERTEXT TRANSFER PROTOCOL

HTTP – a request/response protocol

- It is **STATELESS**
- The client submits a request, HTTP responds with the requested resource and a return code
  - Resources may be static or dynamic
  - Resources may redirect, include other resources, etc.

HTTP Methods

- GET - Retrieves the URI
- POST - Submits a resource to the URI - Like submitting a FORM to be processed by a script
- PUT - Stores a resource under the URI
- DELETE - Deletes the URI

PASSING DATA TO/FROM THE WEB SERVER

XML - EXTENSIBLE MARKUP LANGUAGE

JSON - JAVA SCRIPT OBJECT NOTATION

WEB SERVICES

- If you want to use a web service, you must use an API (application programming interface)
- Defines everything you need to know to talk to a web service:
  1. Message format: SOAP, XML, JSON, etc.
  2. Request syntax: URI, Parameters & Data types
  3. Actions on the server: named methods, HTTP verbs
  4. Security: authentication (username & password)
  5. Response format: SOAP, XML, JSON, etc.

- The web service hides its complexity behind the API

REST

- The “architectural style” is an abstract concept - it defines the characteristics and features you would find in a house built according to that style
- It is NOT the same as the house itself.
- REST is an abstract concept that defines the characteristics and features you would find in a web service request built according to the REST style
- REST is not really a protocol – it is a set of standards used to define Web Services

CHARACTERISTICS OF A REQUEST/RESPONSE FOLLOWING THE REST STYLE

- Resources follow the rules
  - URI (identifies the resource being requested)
  - Uniform Interface Methods (GET, PUT, POST, etc.)
  - Uniform Interface Representation (XML, JSON, HTML)
- Protocols offer features
  - Client-Server (like HTTP)
  - Stateless (each request is independent)
  - Layered (may pass through intermediaries)
  - Cacheable (intermediaries may cache for performance)

WEB SERVICES

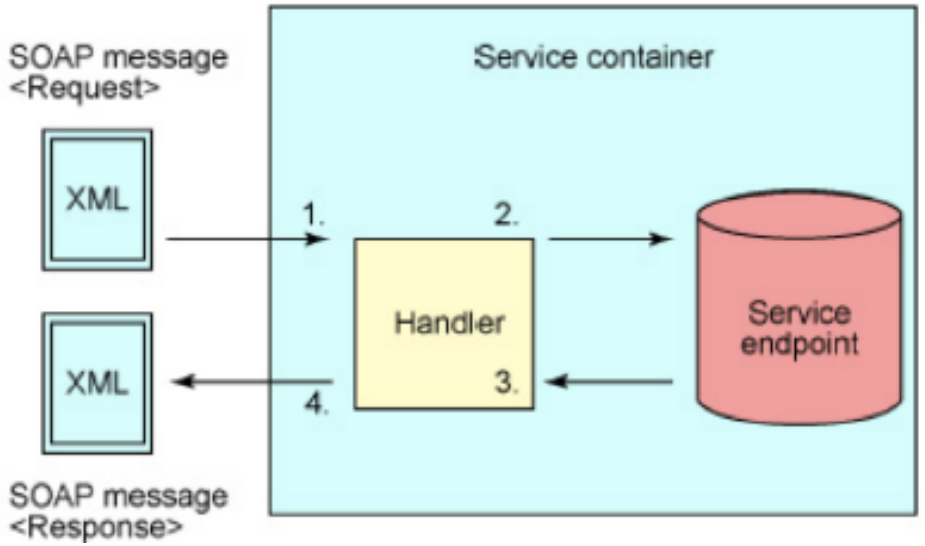
- WSDL (Web Service Description Language) is an XML document that defines "contract" between client and service and is static by its nature.
- SOAP builds an XML based protocol on top of HTTP or some other protocol according to the rules described in the WSDL for that Web Service.

SOAP

- A SOAP message is an XML document containing the following elements:
  - An **Envelope** element that identifies the XML document as a SOAP message
  - A **Header** element that contains header information
  - A **Body** element that contains call and response information
  - A **Fault** element containing errors and status information

SOAP Handlers

- **Handlers** are pluggable classes that can be associated with a Web service or Web service client to provide pre-processing or post-processing of XML messages.
  - Ex: logging XML traffic through a Web service
  - Ex: measure performance by accessing the SOAP header to insert initial and finish times between two checkpoints



PRINCIPAL TYPES OF IP LAW

- Copyright
- Patent
- Trademark
- Trade secret

COPYRIGHT

- Key aspects to remember
  - Doesn't protect just an idea – it has to be "fixed" in a tangible medium
    - You have to write the code, and the code is what is protected
  - Typically doesn't protect a word or phrase (not considered an artistic work)
  - Work made for hire: employer owns works within the scope of employment
    - Doesn't apply to consultants

PATENT

- Key aspects to remember
  - You don't have protection unless you file it
  - Must be a new idea ("sufficiently novel")
    - It cannot be known by the general public before filing (careful about those public presentations – use an NDA)
  - You have to be able to actually do it ("reduce it to practice")
  - Patents are expensive; there are lots of patents by companies like IBM, HP, Oracle and others in the software space
  - Only an individual can be the inventor – not the company; so companies have to get the inventor to assign the idea and patent application.

TRADEMARK

- Key aspects to remember
  - Must be used in commerce
  - Must not be confusingly similar to someone else's mark in the same space
  - You may not have to file to protect it; but filing can get nationwide protection

TRADE SECRET

- Key aspects to remember
  - Must be secret – must use reasonable efforts to keep it secret
    - Nondisclosure agreements
  - No filing required
  - No requirement regarding novelty or originality, just secrecy