W4153 – Cloud Computing

Lecture 1: Introduction, Concepts

Columbia | Engineering
The Fu Foundation School of Engineering and Applied Science

W4153 – Cloud Computing

Lecture 1: Introduction, Concepts

We will start in a few minutes.

If you ask me about the waitlist,
I will become very grumpy.

W4153 Introduction to Databases:

Faculty do not manage waitlists
for some courses, including W4153.
The academic admin staff in the
CS Department manages the waitlist,
priorities and enrollment.
You should contact advising email:
ug-advising, ms-advising, or phd-advising
@cs.columbia.edu

Contents

Contents

- Course Overview
  - About your instructor.
  - Course objectives, logistics, structure, grading, … …
- Core Concepts
  - Cloud computing concepts.
  - Microservices.
  - An aside on modeling.
  - Full stack web application.
  - Application development concepts.
- Sprint 0 – Get Ready for the Remainder of the Course
  - Form teams.
  - Individual end-to-end project to validate that you are ready for the coursework.
- Discussion, Questions, "Lab"

Course Overview

About your Instructor

- 38 years in computer science industry:
  - IBM Fellow.
  - Microsoft Technical Fellow.
  - Chief Technology Officer, CA technologies.
  - Dell Senior Technical Fellow.
  - CTO, Co-Founder, Seeka.tv.
  - Ansys (current):
    - Ansys Fellow, Chief SW Architect;
    - VP/GM, Cloud, AI, Solutions and Developer Enablement BU (CASEBU)
- Academic experience:
  - BA, MS, Ph.D., Computer Science, Columbia University.
  - Approx. 18 semesters as an Adjunct Professor.
  - Professor of Professional Practice in CS (2018)
  - Courses:
    - E1006: Intro. to Computing
    - W4111: Intro. to Databases
    - E6998, E6156: Advanced Topics in SW Engineering (Cloud Computing)
- Approx. 65 technical publications; Approx. 12 patents.

I have taught some version of this class 10 times.

Personal:

- Two children:
  - College Sophomore.
  - 2019 Barnard Graduate.
- Hobbies:
  - Krav Maga, Black Belt in Kenpo Karate.
  - Former 1LT, New York Guard.
  - Bicycling.
  - Astronomy.
  - Languages:
    - Proficient in Spanish.
    - Learning Arabic.

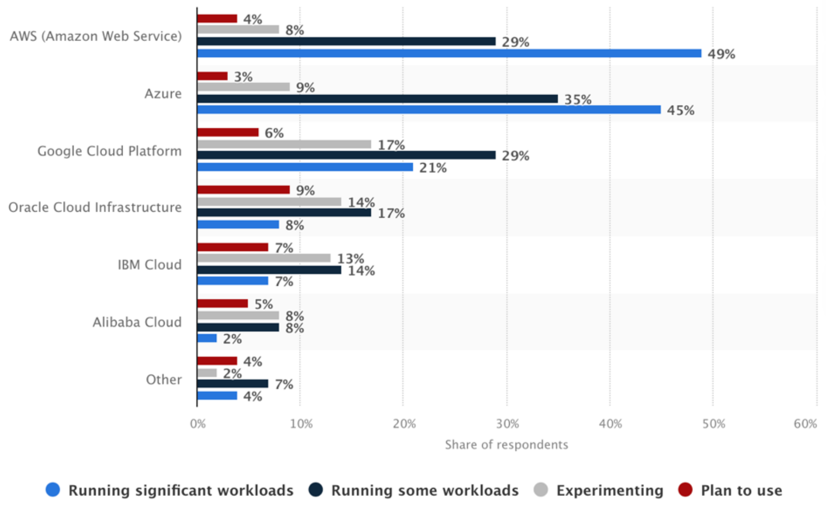
Logistics and Grading

- Logistics
  - Lectures: Friday’s 1:10 to 3:40 PM
    - I can present and cover material *much more quickly* than you can “use and implement.” ➔
      - 1:10 to 2:50-ish will be lecture and presenting/teaching material.
      - 2:50-ish to 3:40 will be “in the lecture room ‘labs’” where you can work on projects, ask questions, discuss details.
    - I will periodically devote a lecture slot to extended offices hours, project team meetings, etc.
  - Office Hours: Friday’s 8:30 AM to 10:00 AM and 4:00 PM to 5:00 PM.

- A total of 100 points is available:
  - Your final team project earns up to 75 points.
  - Written individual homework assignments earn up to 25 points.
  - Earning an A+ requires the TAs and my seeing and recognizing individual extra effort that we reward with extra-credit. We will let you know when we assign extra-credit points.
  - “Extraordinary results come from doing ordinary things extraordinarily well.”
  - “The harder I work, the luckier I get.”

Environment and Systems

- There are 3 primary cloud service providers:
  - Microsoft Azure
  - Google Cloud Platform
  - Amazon Web Services
- The course will focus on concepts and use the clouds for implementations.
- Managing costs:
  - All clouds have "free tiers" and/or "new user" credits. **You must track and manage your costs.**
  - We have credits for Google Cloud Platform.
  - I am working on credits and free usage for AWS and Azure.
- I suggest that you also mirror/follow my usage of locally installed SW for development:
  - PyCharm, DataGrip, WebStorm including cloud usage plugins.
  - Docker
  - Others will follow during semester.



	Plan to use	Experimenting	Running some workloads	Running significant workloads
AWS (Amazon Web Service)	4%	8%	29%	49%
Azure	3%	9%	35%	45%
Google Cloud Platform	6%	17%	29%	21%
Oracle Cloud Infrastructure	9%	14%	17%	8%
IBM Cloud	7%	13%	14%	7%
Alibaba Cloud	5%	8%	8%	2%
Other	4%	2%	7%	4%

Objectives

Core Concepts (Initial, There are More)

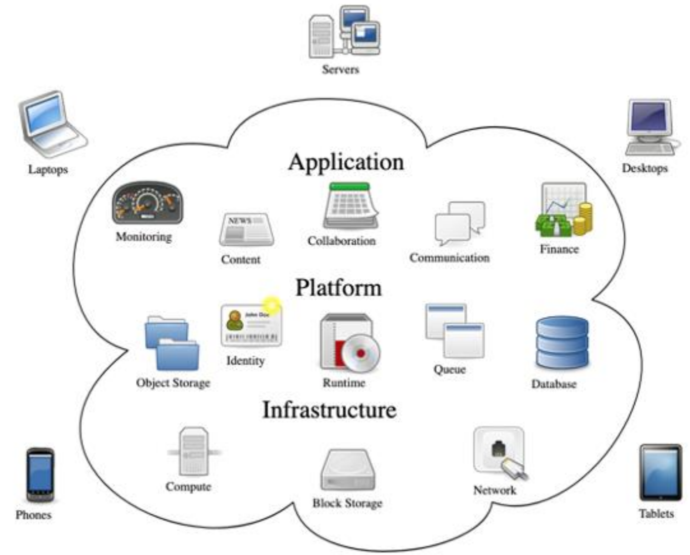
Cloud Computing
(Initial – There are More)

Definitions – As Good as Any of the 10^6 Definitions

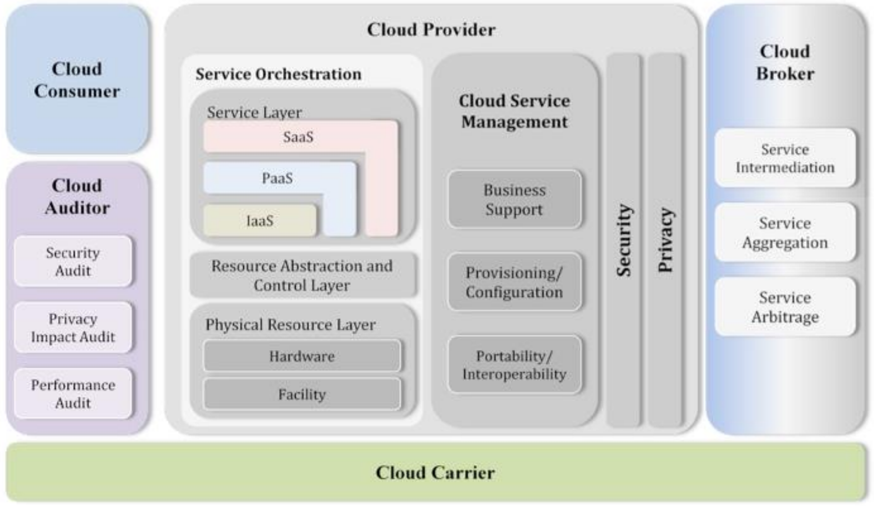
- "Simply put, **cloud computing** is the delivery of computing services—including servers, storage, databases, networking, software, analytics, and intelligence—over the internet ("the cloud") to offer faster innovation, flexible resources, and economies of scale. You typically pay only for cloud services you use, helping you lower your operating costs, run your infrastructure more efficiently, and scale as your business needs change." (https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-cloud-computing)
- "Cloud computing is a delivery model for providing on-demand information technology (IT) infrastructure and services over the Internet. Cloud services are scalable, which means that compute, networking, and storage resources can be adjusted manually or automatically in real time to meet the changing demands of applications and users. Customers can access and use cloud services through freemium, subscription, or consumption-based pricing models.
  Essentially, a cloud computing environment has five important characteristics that differentiate it from a traditional in-house, local computing environment:
  - The customer can provision computing resources by themselves on demand.
  - Resources are provisioned and accessed over the Internet.
  - Resources are pooled together to serve the needs of multiple customers.
  - Resources can be rapidly scaled horizontally or scaled down depending on need.
  - Resource use is controlled by the customer and can be monitored in real-time."

  (https://www.techopedia.com/definition/2/cloud-computing)

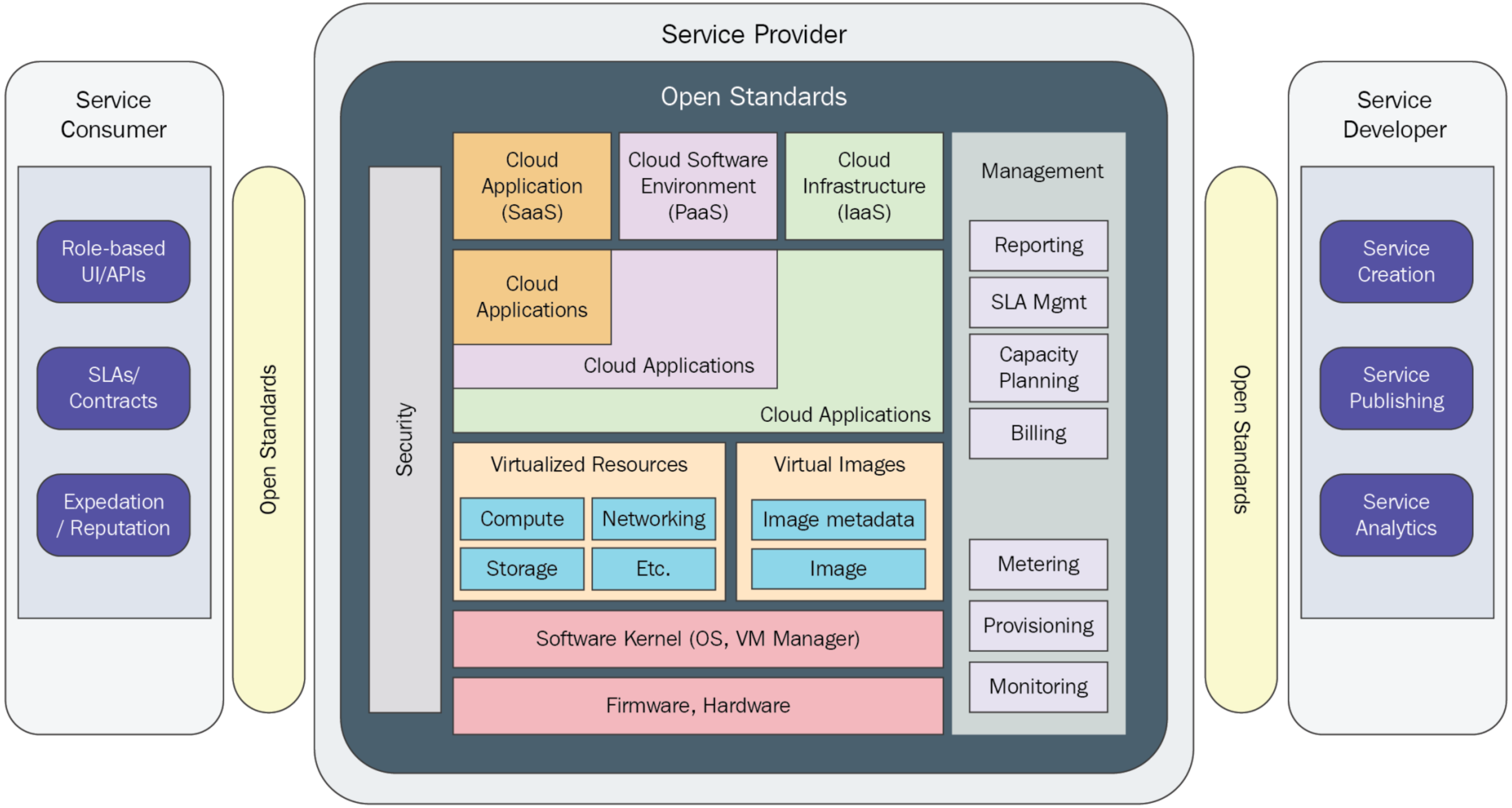
Cloud Computing



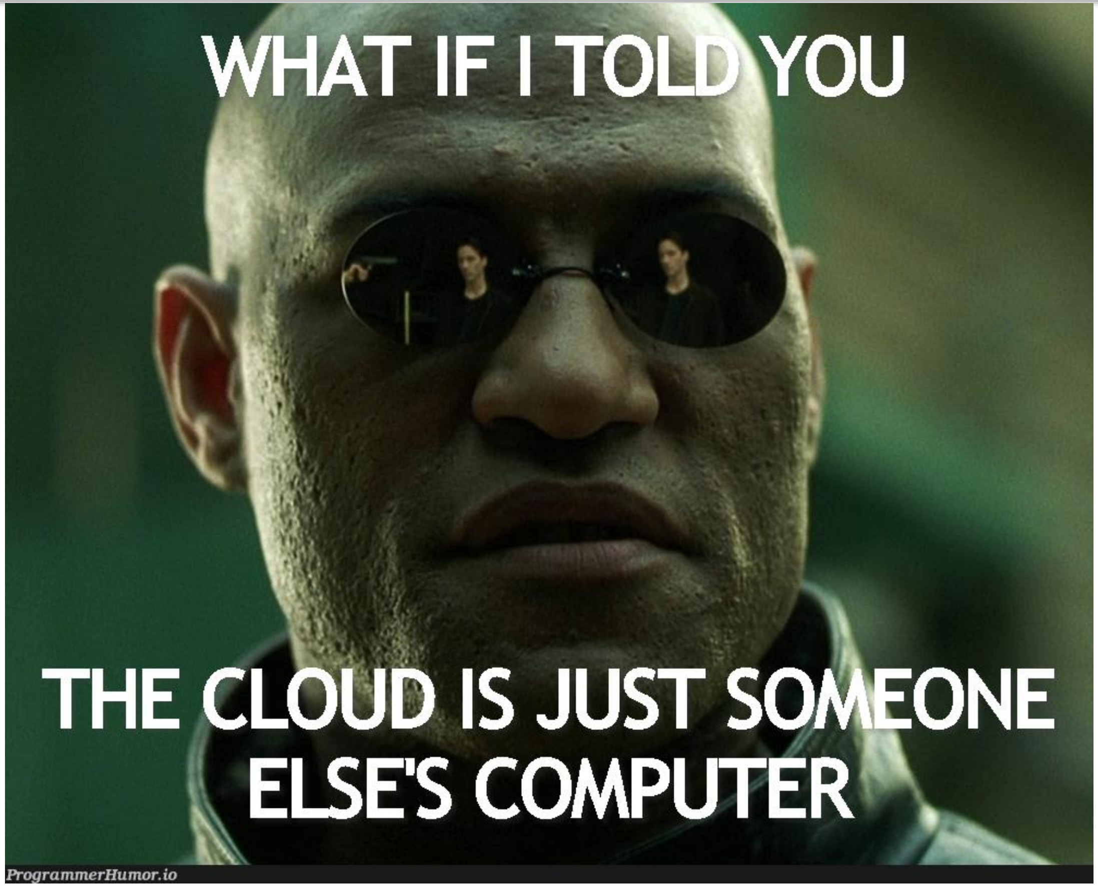
NIST Reference Architecture



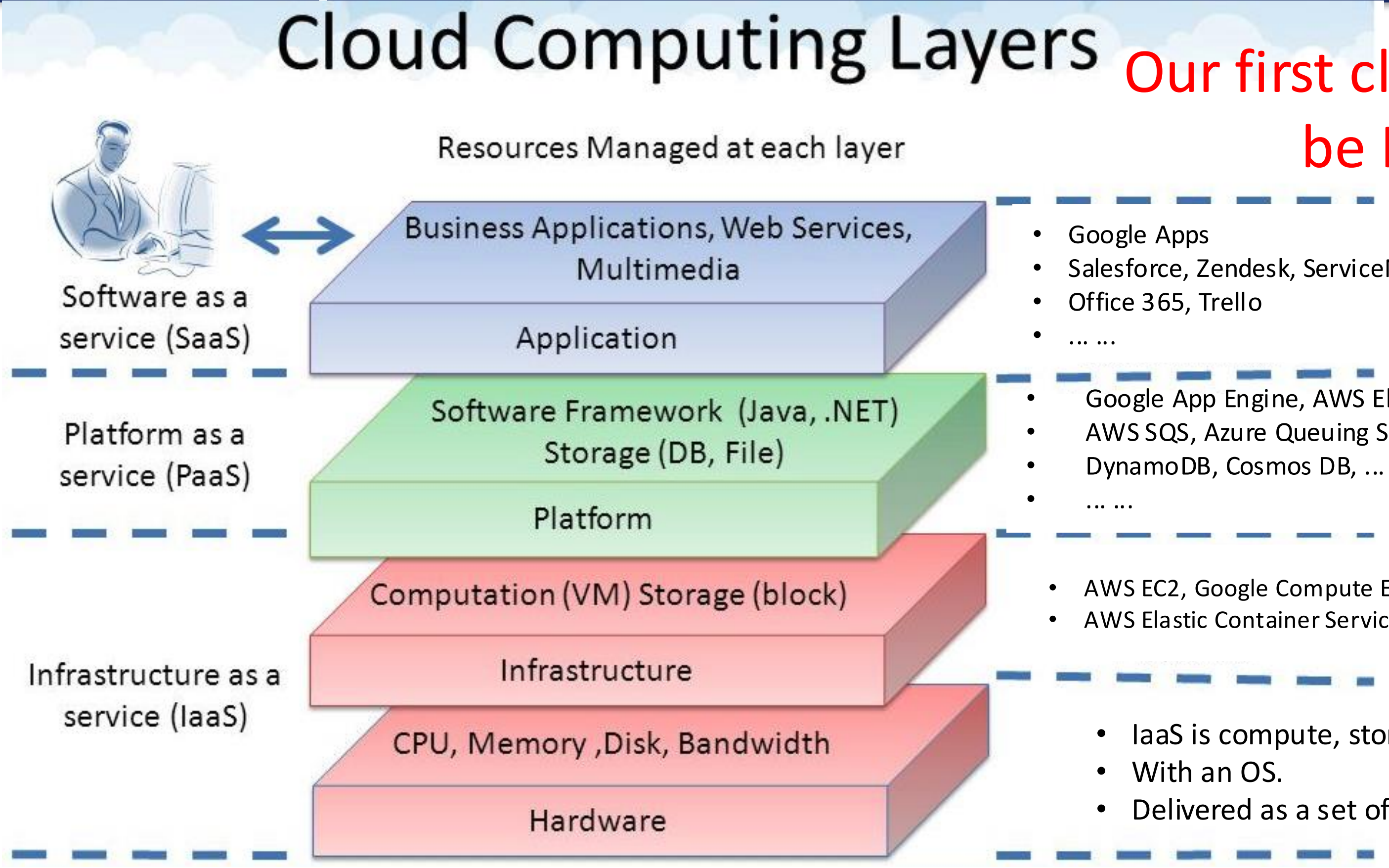
- There are **a lot** of concepts, aspects, types, … …
- Some conceptual models for how it comes together.
- Many applications and solutions build on all kinds of things from many clouds.
- We will see examples of technology and use during the semester.


Service Provider
Open Standards
Service Consumer
Role-based UI/APIs
SLAs/ Contracts
Expedation / Reputation
Open Standards
Security
Cloud Application (SaaS)
Cloud Software Environment (PaaS)
Cloud Infrastructure (IaaS)
Cloud Applications
Cloud Applications
Cloud Applications
Virtualized Resources
Compute
Networking
Storage
Etc.
Virtual Images
Image metadata
Image
Software Kernel (OS, VM Manager)
Firmware, Hardware
Management
Reporting
SLA Mgmt
Capacity Planning
Billing
Metering
Provisioning
Monitoring
Open Standards
Service Developer
Service Creation
Service Publishing
Service Analytics

Definitions – The Best One



Core Layers


Cloud Computing Layers
Resources Managed at each layer
Software as a service (SaaS)
Business Applications, Web Services, Multimedia
Application
Platform as a service (PaaS)
Software Framework (Java, .NET) Storage (DB, File)
Platform
Infrastructure as a service (IaaS)
Computation (VM) Storage (block)
Infrastructure
CPU, Memory ,Disk, Bandwidth
Hardware

- Google Apps
- Salesforce, Zendesk, ServiceNow, Concur, … …
- Office 365, Trello
- … …

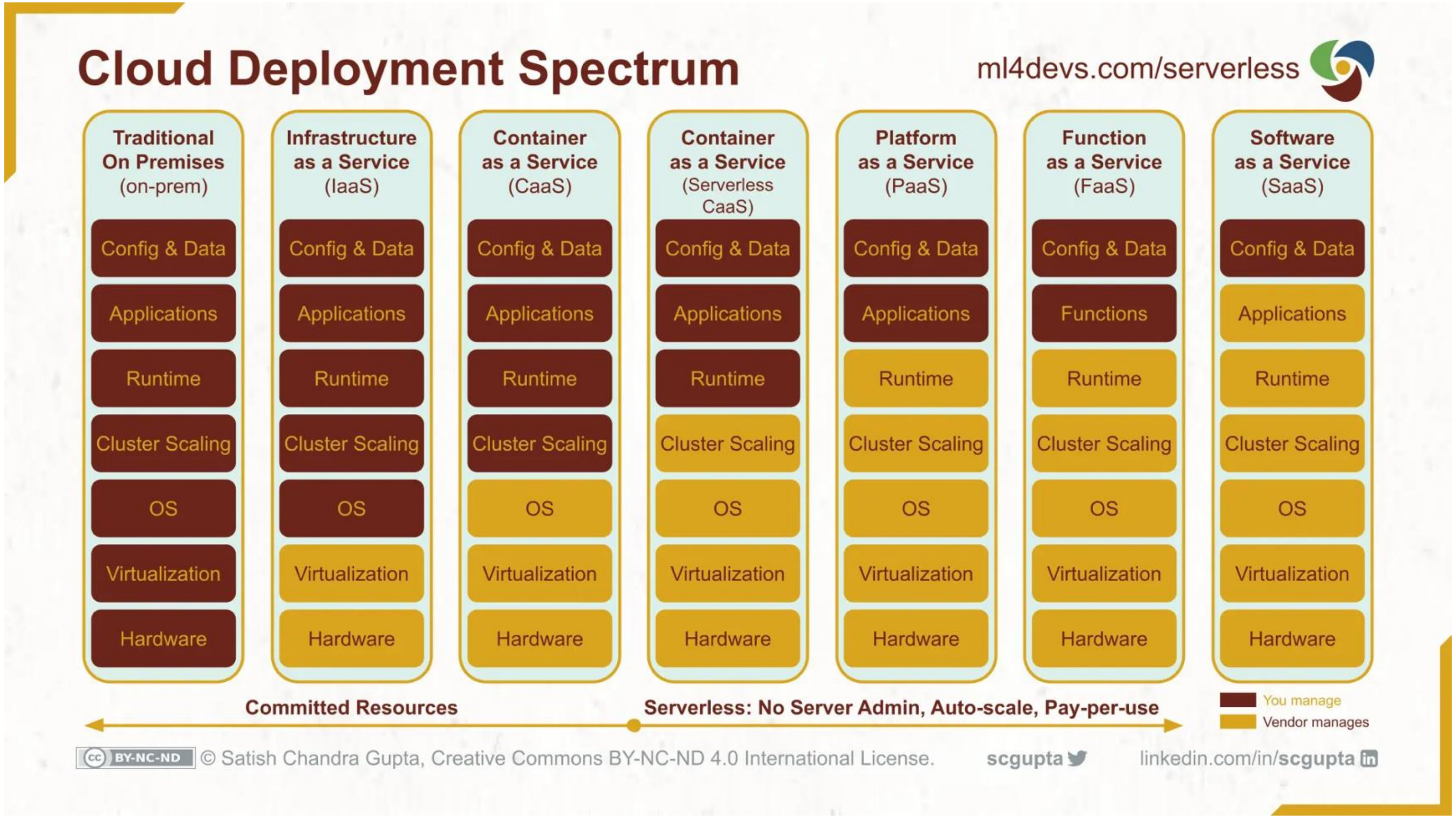
- Google App Engine, AWS Elastic Beanstalk, …
- AWS SQS, Azure Queuing Service, Google PubSub
- DynamoDB, Cosmos DB, … …
- … …

- AWS EC2, Google Compute Engine, Azure VMs, … …
- AWS Elastic Container Service, Azure Kubernetes, … …

Our first cloud elements will be IaaS (VMs).

- IaaS is compute, storage, networking.
- With an OS.
- Delivered as a set of VMs and/or containers.

Cloud Layers



Cloud Deployment Spectrum

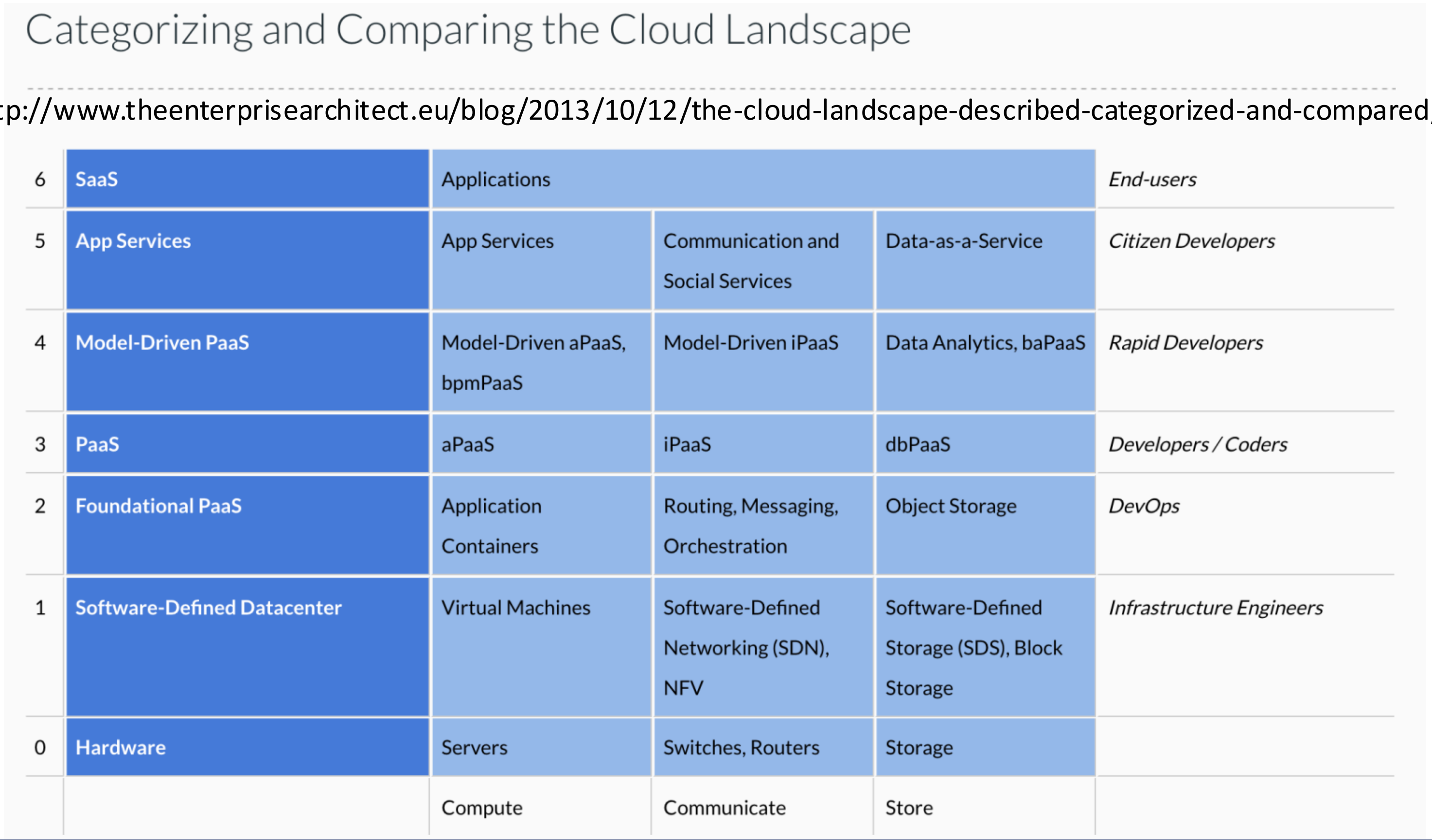
ml4devs.com/serverless

Traditional On Premises (on-prem)	Infrastructure as a Service (IaaS)	Container as a Service (CaaS)	Container as a Service (Serverless CaaS)	Platform as a Service (PaaS)	Function as a Service (FaaS)	Software as a Service (SaaS)
🟥 Config & Data	🟥 Config & Data	🟥 Config & Data	🟥 Config & Data	🟥 Config & Data	🟥 Config & Data	🟥 Config & Data
🟥 Applications	🟥 Applications	🟥 Applications	🟥 Applications	🟥 Applications	🟥 Functions	🟨 Applications
🟥 Runtime	🟥 Runtime	🟥 Runtime	🟥 Runtime	🟨 Runtime	🟨 Runtime	🟨 Runtime
🟥 Cluster Scaling	🟥 Cluster Scaling	🟥 Cluster Scaling	🟨 Cluster Scaling	🟨 Cluster Scaling	🟨 Cluster Scaling	🟨 Cluster Scaling
🟥 OS	🟥 OS	🟨 OS	🟨 OS	🟨 OS	🟨 OS	🟨 OS
🟥 Virtualization	🟨 Virtualization	🟨 Virtualization	🟨 Virtualization	🟨 Virtualization	🟨 Virtualization	🟨 Virtualization
🟥 Hardware	🟨 Hardware	🟨 Hardware	🟨 Hardware	🟨 Hardware	🟨 Hardware	🟨 Hardware

🟥 You manage
🟨 Vendor manages

Committed Resources ⟵ ● ⟶ Serverless: No Server Admin, Auto-scale, Pay-per-use

Cloud Concepts – One Perspective



Categorizing and Comparing the Cloud Landscape

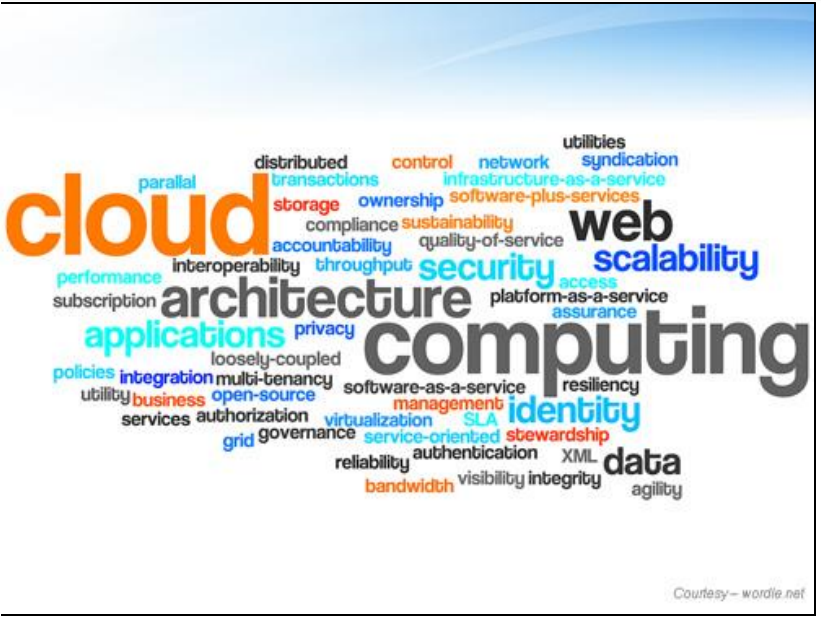
http://www.theenterprisearchitect.eu/blog/2013/10/12/the-cloud-landscape-described-categorized-and-compared/

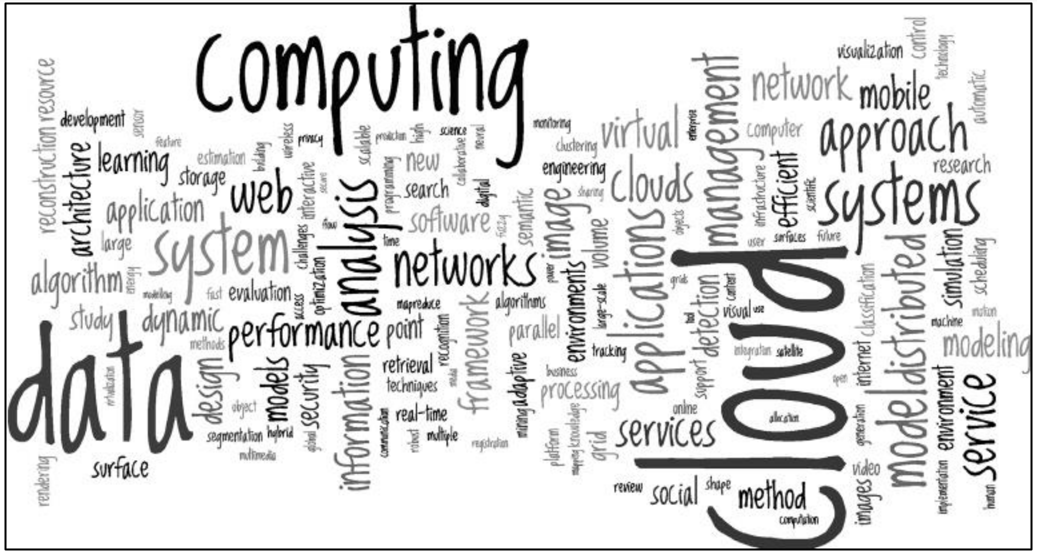
6	SaaS	Applications			*End-users*
5	App Services	App Services	Communication and Social Services	Data-as-a-Service	*Citizen Developers*
4	Model-Driven PaaS	Model-Driven aPaaS, bpmPaaS	Model-Driven iPaaS	Data Analytics, baPaaS	*Rapid Developers*
3	PaaS	aPaaS	iPaaS	dbPaaS	*Developers / Coders*
2	Foundational PaaS	Application Containers	Routing, Messaging, Orchestration	Object Storage	*DevOps*
1	Software-Defined Datacenter	Virtual Machines	Software-Defined Networking (SDN), NFV	Software-Defined Storage (SDS), Block Storage	*Infrastructure Engineers*
0	Hardware	Servers	Switches, Routers	Storage	
		Compute	Communicate	Store	

Why Cloud Computing?

- Economy of scale:
  - There are fixed costs for a data center. One big data center of size N is more cost effective than N data centers.
  - The cloud service provider (CSP) can “buy in bulk” from HW suppliers.
- The average matters:
  - A company must over provision resources to deal with surges in load/requests. Over provisioning by 3x is common before cloud.
  - The CSP can rely on averages. Not all N tenants will have surges at the same time. So, the CSP can over provision by a smaller amount than 3*N.
- Elastic capacity:
  - Tenants do not need to over-provision.
  - They can “request” additional resources when there is a surge and then release. (scale up versus scale down)
- Outsource non-core business functions:
  - A tenant does not want to manage buildings, pay water bills, … …
  - Buying HW, installing, configuring, monitoring and rebooting, … is a total downer.
  - The cloud takes over many systems and applications management and operations tasks.
- Capital versus Operational Cost: Buying a house versus renting a hotel room.
  - On premises, non-cloud resources require capital investment, depreciation, ROI, … …
  - Cloud resources are rented and are operational.
- There are many other reasons that you can just look up because I am too lazy to type.
- Taking a cloud computing course looks really cool on your resume.

Cloud Computing Concepts





- There are many, many cloud concepts, and many different perspectives on each concept.
- Most cloud service providers offer their own specific versions of concepts and technology – But the core of the concept is similar.
- We explore as many as we can by writing a "complex" application using the technology.
  - Each element in the application will be very simple, i.e. not deep application domain logic.
  - The overall application will have many and diverse implementations for elements.

Microservices

Microservices (https://microservices.io/index.html)

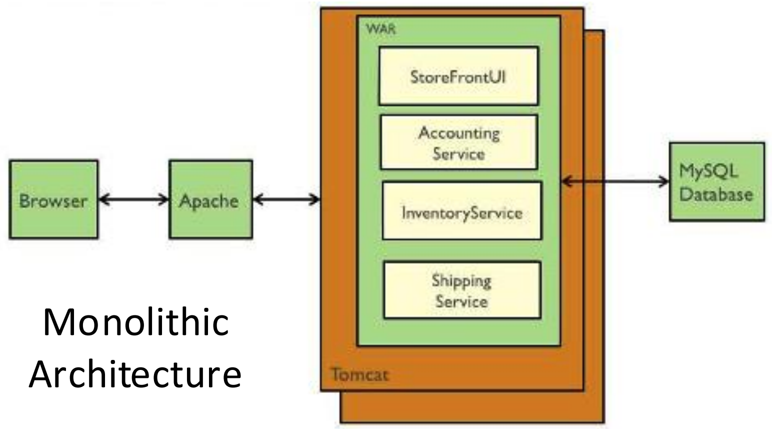
What are microservices?

Microservices - also known as the microservice architecture - is an architectural style that structures an application as a collection of services that are

- Highly maintainable and testable
- Loosely coupled
- Independently deployable
- Organized around business capabilities
- Owned by a small team

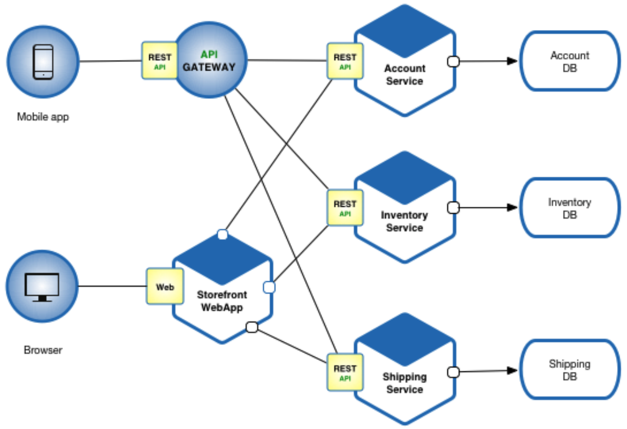
The microservice architecture enables the rapid, frequent and reliable delivery of large, complex applications. It also enables an organization to evolve its technology stack.

- Microservice benefits relative to monoliths are hard to understand without having developed and maintained monoliths.
- Cars are "cool." Cars are totally, super cool if you had to go into town using a horse drawn carriage.



Monolithic Architecture

Microservices Architecture



Microservice Characteristics

(https://dzone.com/articles/what-is-microservices-an-introduction-to-microserv)

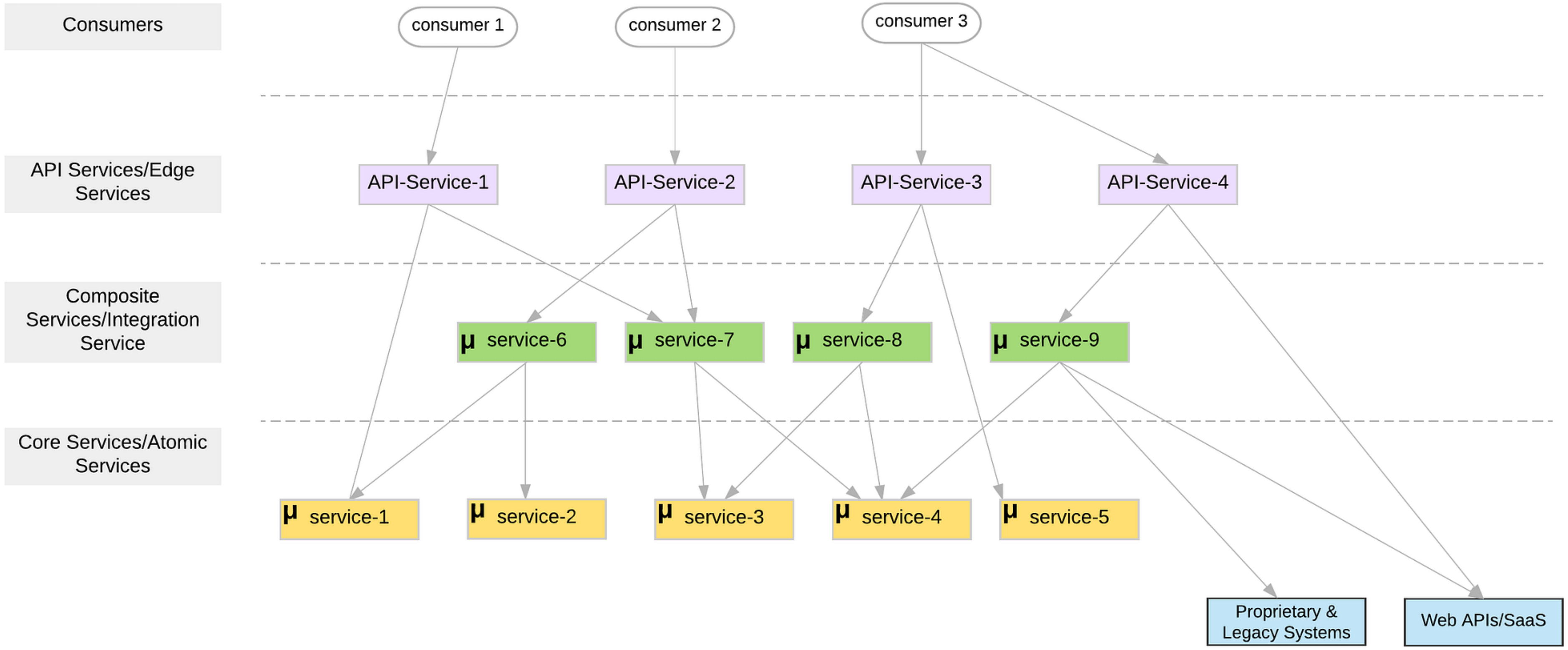
Decoupling
1
Componentization
2
Business Capabilities
3
Autonomy
4
5
Continous Delivery
6
Responsibility
7
Decentralized Governance
8
Agility

Microservice Characteristics

- **Decoupling** - Services within a system are largely decoupled, so the application as a whole can be easily built, altered, and scaled.
- **Componentization** - Microservices are treated as independent components that can be easily replaced and upgraded.
- **Business Capabilities** - Microservices are very simple and focus on a single capability.
- **Autonomy** - Developers and teams can work independently of each other, thus increasing speed.
- **Continous Delivery** - Allows frequent releases of software through systematic automation of software creation, testing, and approval.
- **Responsibility** - Microservices do not focus on applications as projects. Instead, they treat applications as products for which they are responsible.
- **Decentralized Governance** - The focus is on using the right tool for the right job. That means there is no standardized pattern or any technology pattern. Developers have the freedom to choose the best useful tools to solve their problems.
- **Agility** - Microservices support agile development. Any new feature can be quickly developed and discarded again.

Microservice Layers

(https://medium.com/microservices-in-practice/microservices-layered-architecture-88a7fc38d3f1)



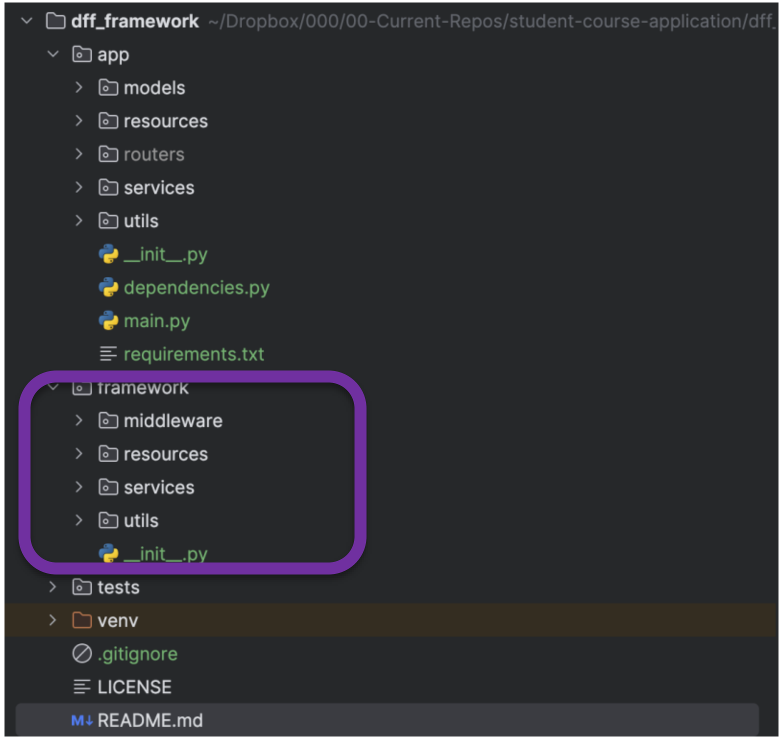
Microservice Details and Perspectives

- Some sites for good, introductory overviews and more information about microservices:
  - https://martinfowler.com/articles/microservices.html
  - https://microservices.io/
  - https://aws.amazon.com/microservices/
- What is true of any:
  - Mapping, classification, taxonomy, … …?
  - Technology definition, … …?
  - Enumeration of technology pros and cons … …?
- Answers:
  - "… … it is … …
    More honor'd in the breach than the observance"
  - If you have three subject matter experts, you have 7 conflicting opinions.
  - "The most dangerous thing in the world is a 2nd LT with a map."
    Like any map, it is both a way of orientating yourself and a way of having a spectacular disaster because you are studying the map instead of thinking.

Our First Microservice

- A simple microservice for getting basic information about courses (sections).
- We use
  - FastAPI for the (web) application server.
  - We use REST and OpenAPI.
  - Follow a design pattern for "large" applications, with modifications.
- ➔ that our project will have:
  - main.py
  - Routers
  - Resources
  - Models

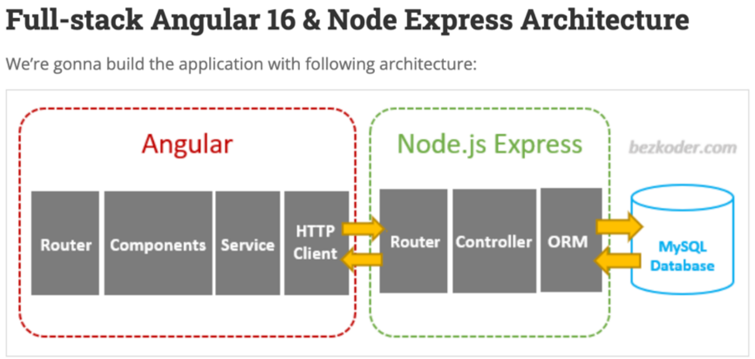
  We will go into more details in later lectures.
- dff_framework:
  - Is a bunch of random ways I think about building apps.
  - We would install as dependencies, not include in the project. I put together for teaching simplicity.



```
dff_framework ~/Dropbox/000/00-Current-Repos/student-course-application/dff_
  app
    models
    resources
    routers
    services
    utils
    __init__.py
    dependencies.py
    main.py
    requirements.txt
  framework
    middleware
    resources
    services
    utils
    __init__.py
  tests
  venv
  .gitignore
  LICENSE
  README.md
```

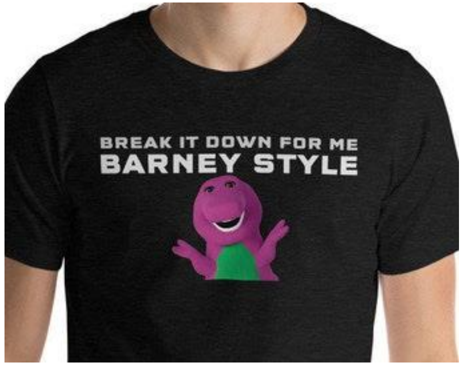
Walkthrough of a Set of Examples

- I found a got set of starter examples at BezCoder.com:
  - Full Stack: https://www.bezkoder.com/category/full-stack/.
  - I will quickly walk through an example



Full-stack Angular 16 & Node Express Architecture

We're gonna build the application with following architecture:



  - This shows core concepts and has a lot of step-by-step guides.
- I tend to use Python/FastAPI in my examples and lectures but do not provide a tutorial that "breaks it down Barney Style."
  - ~/Dropbox/000/00-Current-Repos/W4153/angular-17-crud-example
  - ~/Dropbox/000/00-Current-Repos/W4153/nodejs-express-mysql
  -

Very First Step (Individual)

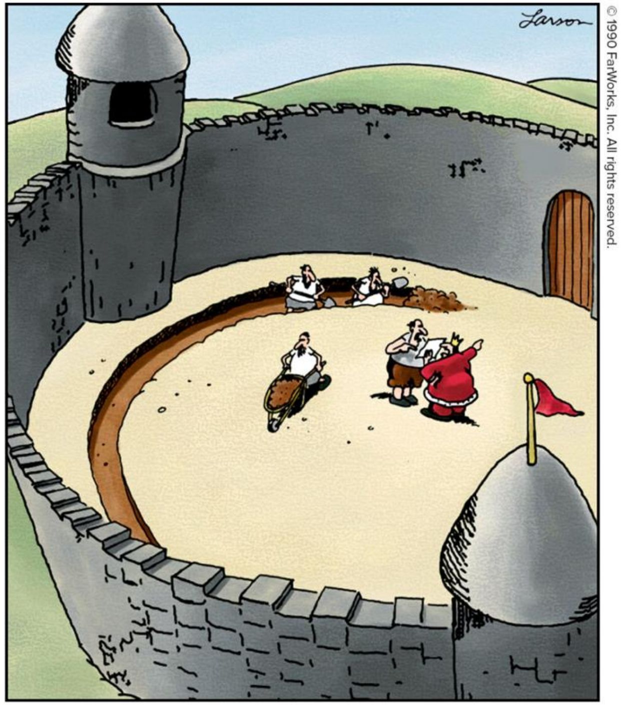
- Create 3 GitHub projects (use any names you want) – Mine are:
  - W4153-P1-UI (https://github.com/donald-f-ferguson/W4153-P1-UI.git)
  - W4153-P1-Application (https://github.com/donald-f-ferguson/W4153-P1-Application.git)
  - W4153-P1-Database (https://github.com/donald-f-ferguson/W4153-P1-Database.git)
- We will
  - Develop each project locally.
  - Unit test locally.
  - Integration test locally.
  - Manually deploy to the cloud platform.
  - Test again.

An Aside on Modeling

The Value of Just Enough Modeling

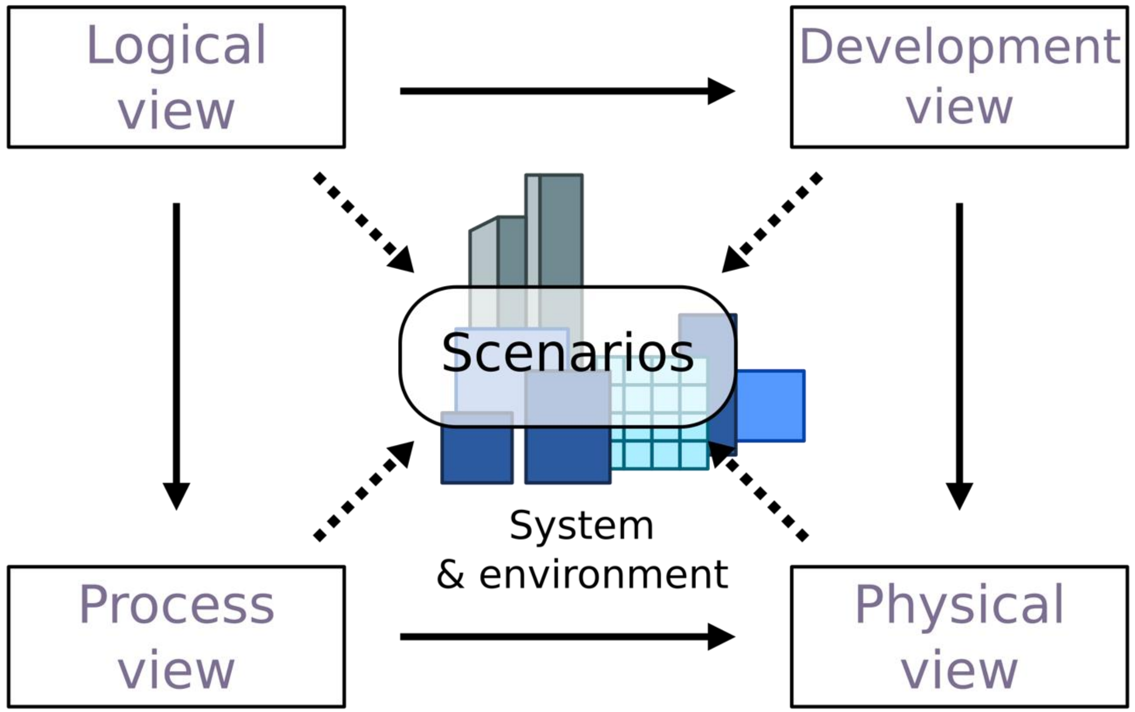
- Weinberg's Second Law: "If builders built buildings the way programmers wrote programs, **then the first woodpecker that came along would destroy civilization."**
- Give your project, code, … a little upfront thought. Focus on simple steps, test and then next step.
- Most students and teams build their systems and write their code like a slam poet on a triple espresso.
- Just enough modeling and thought, with simple steps and frequent testing seems slow, but … …

"Slow is smooth. Smooth is fast."



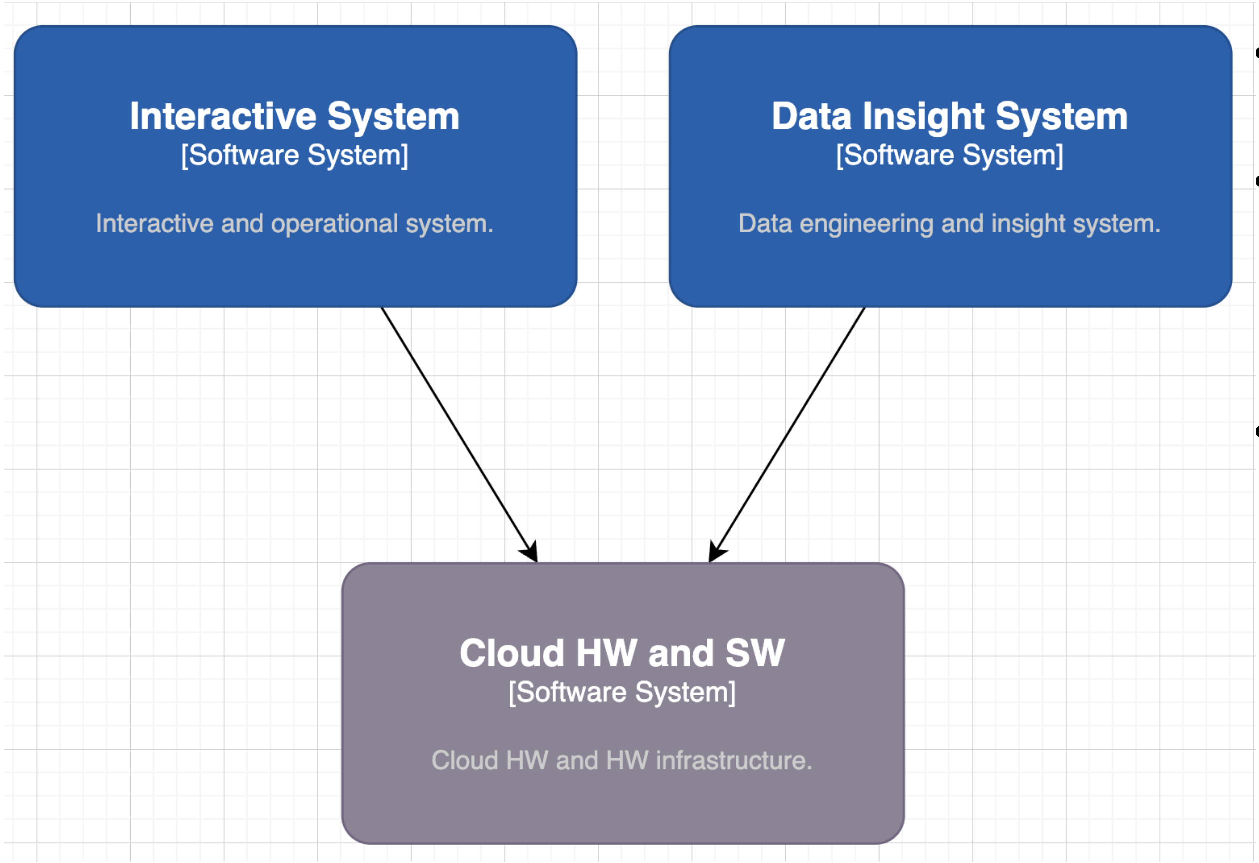
Suddenly, a heated exchange took place between the king and the moat contractor.

4+1 Architecture Model



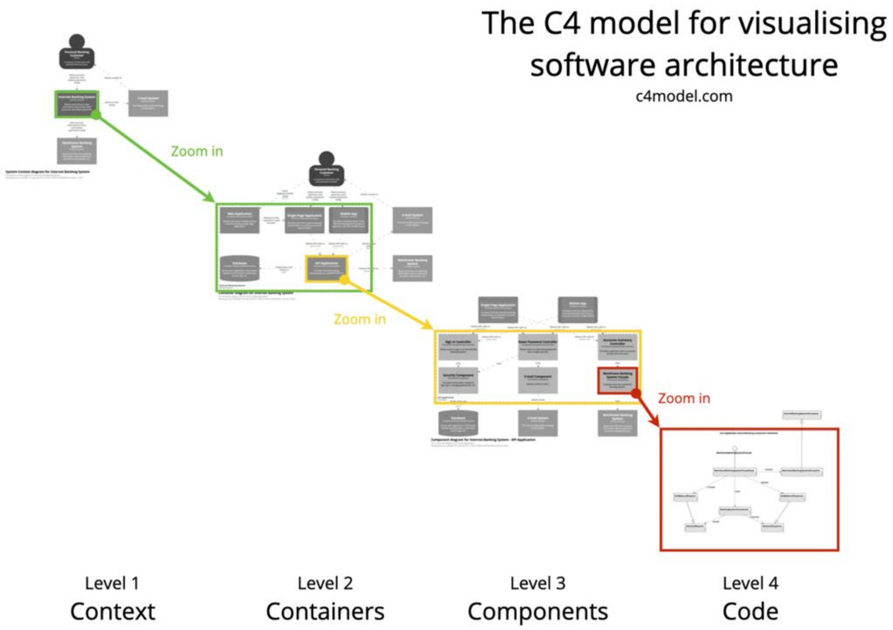
- I try to apply "just enough" modeling in 5 areas.
- The 4+1 Architecture Model is a simple, conceptual approach.
- I have a preferred "notation" for each "view."

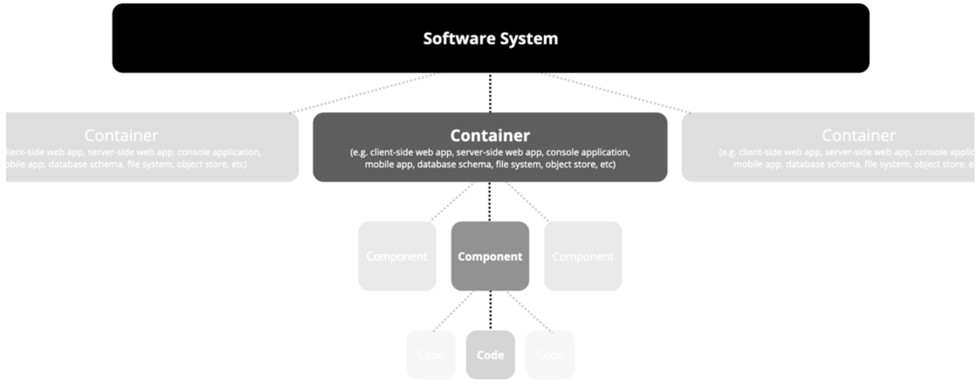
Class Project Solution – Logical View



- I use C4 and UML for the logical model.
- Many complex solutions have two major elements:
  - Operational/Interactive
  - Analytical/Data Analysis
- We place more emphasis on interactive because there are topics courses on cloud and data.

C4 Model (https://c4model.com/)



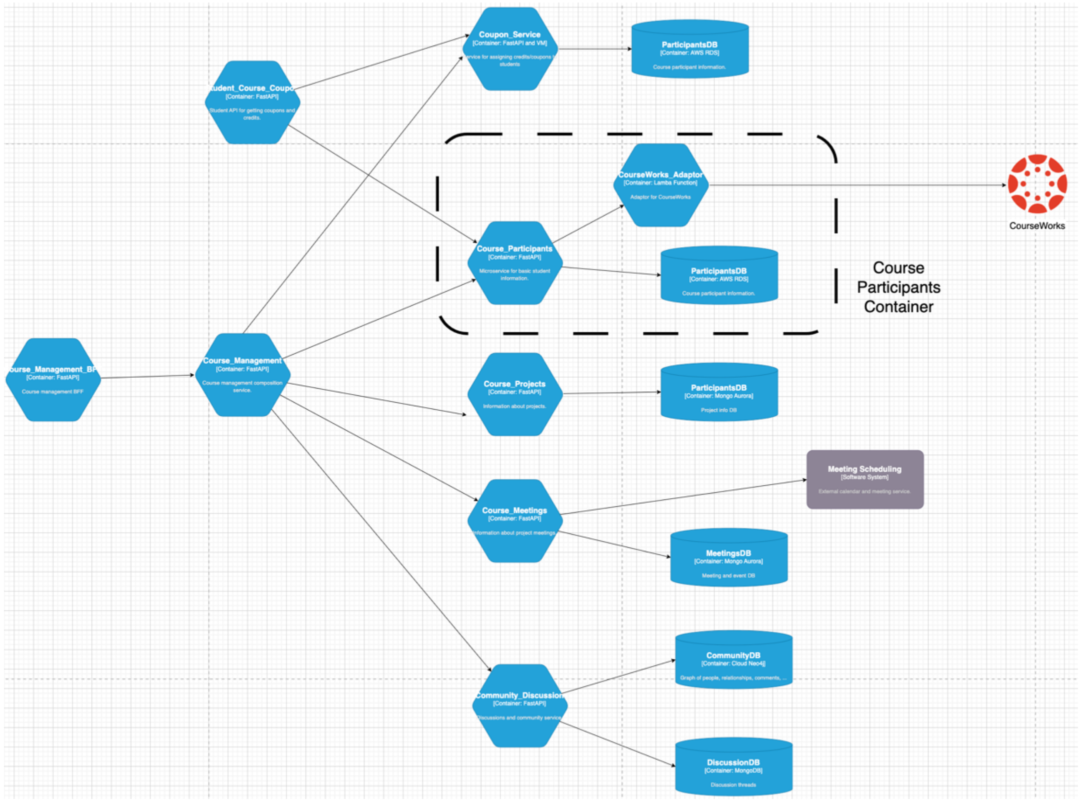

Software System
Container
Container
(e.g. client-side web app, server-side web app, console application, mobile app, database schema, file system, object store, etc)
Container
Component
Component
Component
Code
Code
Code

A **software system** is made up of one or more **containers** (applications and data stores), each of which contains one or more **components**, which in turn are implemented by one or more **code** elements (classes, interfaces, objects, functions, etc).

- Incremental, progressive design and refinement.
- Again, “just enough” to think it through and explain.

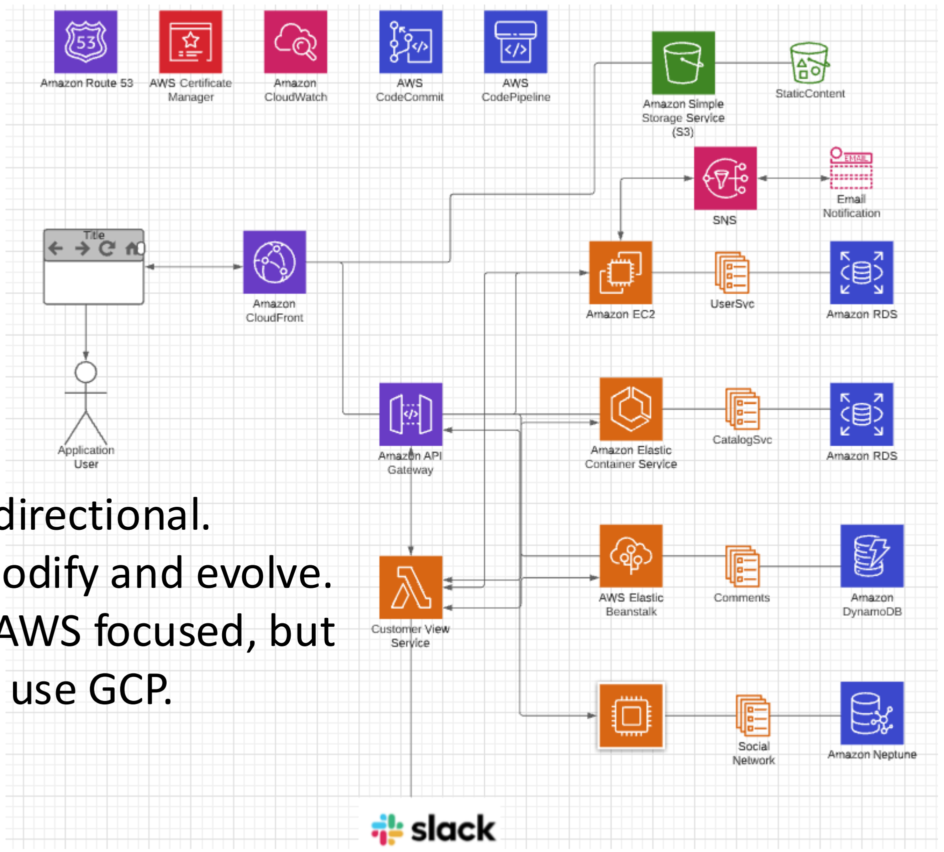
(Partial) Logical View – Interactive System

- Some of the distinctions are arbitrary or in the ”eye of the beholder.”
  - System vs Container
  - Container vs Component
  - etc.
- A container is not synonymous with a “Docker container.”
- I have been in SW for 40 years and we still cannot define terms like “component.”
- Just use common sense.
- In general, containers in the cloud are *microservices.”*



Physical View

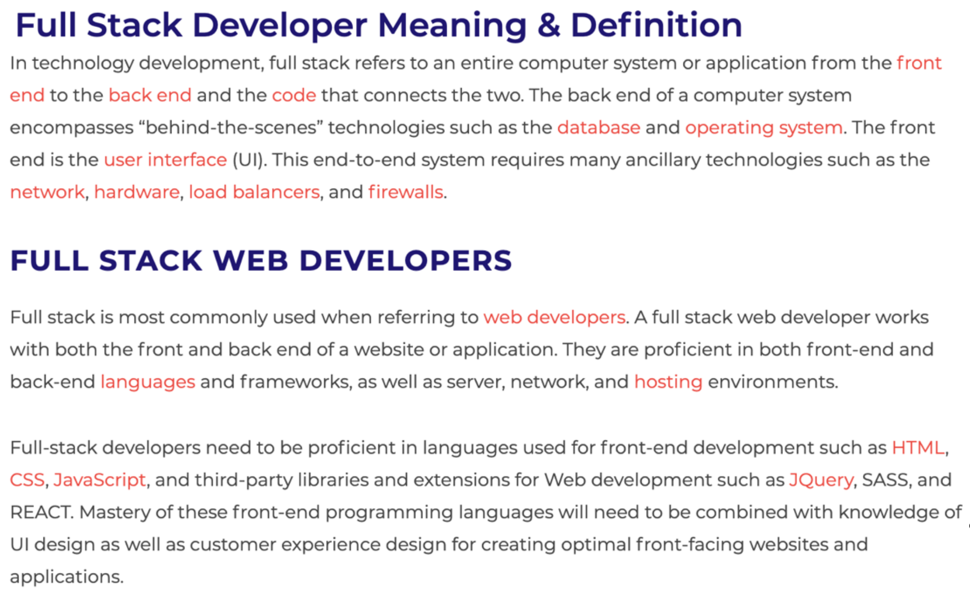
- Note: Cannot show all of the technology and connections with creating spaghetti.
- Databases/Storage:
  - RDS
  - DynamoDB
  - MongoDB (or Document DB)
  - Neptune (or Neo4j)
  - S3
- Networking/Communication:
  - Route 53 (DNS)
  - Certificate Management
  - CloudFront
  - VPC
  - API Gateway
- Compute:
  - EC2
  - Elastic Beanstalk
  - Elastic Container Service (or Docker)
  - Lambda Functions
- Integration/collaboration:
  - SNS, SQS
  - Email
  - Slack
- Continuous Integration/Continuous Deployment:
  - Code Commit, Code Pipeline
  - Or GitHub and actions



- This is directional.
- I will modify and evolve.
- This is AWS focused, but we will use GCP.

Full Stack Web Application

Full Stack Application



Full Stack Developer Meaning & Definition

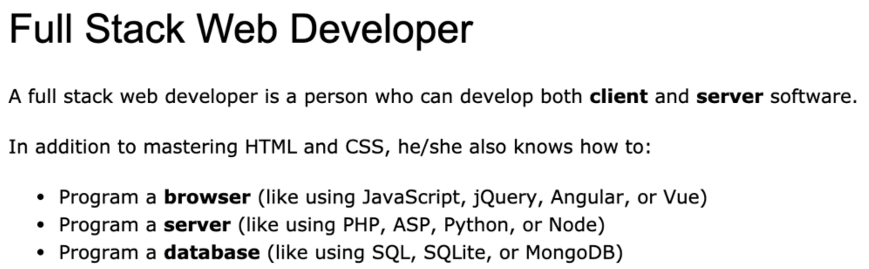
In technology development, full stack refers to an entire computer system or application from the front end to the back end and the code that connects the two. The back end of a computer system encompasses "behind-the-scenes" technologies such as the database and operating system. The front end is the user interface (UI). This end-to-end system requires many ancillary technologies such as the network, hardware, load balancers, and firewalls.

FULL STACK WEB DEVELOPERS

Full stack is most commonly used when referring to web developers. A full stack web developer works with both the front and back end of a website or application. They are proficient in both front-end and back-end languages and frameworks, as well as server, network, and hosting environments.

Full-stack developers need to be proficient in languages used for front-end development such as HTML, CSS, JavaScript, and third-party libraries and extensions for Web development such as JQuery, SASS, and REACT. Mastery of these front-end programming languages will need to be combined with knowledge of UI design as well as customer experience design for creating optimal front-facing websites and applications.

https://www.webopedia.com/definitions/full-stack/



Full Stack Web Developer

A full stack web developer is a person who can develop both **client** and **server** software.

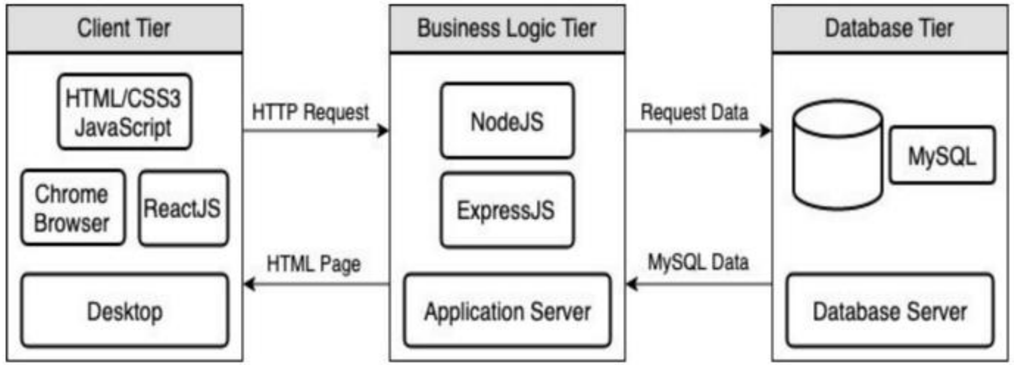
In addition to mastering HTML and CSS, he/she also knows how to:

- Program a **browser** (like using JavaScript, jQuery, Angular, or Vue)
- Program a **server** (like using PHP, ASP, Python, or Node)
- Program a **database** (like using SQL, SQLite, or MongoDB)

https://www.w3schools.com/whatis/whatis_fullstack.asp

- There are courses that cover topics:
  - COMS W4153: Advanced Software Engineering
  - COMS W4111: Introduction to Databases
  - COMS W4170 - User Interface Design
- This course will focus on cloud realization, microservices and application patterns, … …
- Also, I am not great at UIs … … We will not emphasize or require a lot of UI work.

Full Stack Web Application



M	=	Mongo
E	=	Express
R	=	React
N	=	Node

I start with FastAPI and MySQL, but all the concepts are the same.

https://levelup.gitconnected.com/a-complete-guide-build-a-scalable-3-tier-architecture-with-mern-stack-es6-ca129d7df805

- My preferences are to replace React with Angular, and Node with Flask.
- There are three projects to design, develop, test, deploy, ...
  1. Browser UI application.
  2. Microservice.
  3. Database.
- We will initial have two deployments: local machine, virtual machine.
  We will ignore the database for step 1.

Some Terms

- A web application server or web application framework: “A web framework (WF) or web application framework (WAF) is a software framework that is designed to support the development of web applications including web services, web resources, and web APIs. Web frameworks provide a standard way to build and deploy web applications on the World Wide Web. Web frameworks aim to automate the overhead associated with common activities performed in web development. For example, many web frameworks provide libraries for database access, templating frameworks, and session management, and they often promote code reuse.” (https://en.wikipedia.org/wiki/Web_framework)

- REST: “REST (Representational State Transfer) is a software architectural style that was created to guide the design and development of the architecture for the World Wide Web. REST defines a set of constraints for how the architecture of a distributed, Internet-scale hypermedia system, such as the Web, should behave. The REST architectural style emphasises uniform interfaces, independent deployment of components, the scalability of interactions between them, and creating a layered architecture to promote caching to reduce user-perceived latency, enforce security, and encapsulate legacy systems.[1]

  REST has been employed throughout the software industry to create stateless, reliable web-based applications.” (https://en.wikipedia.org/wiki/REST)

Some Terms

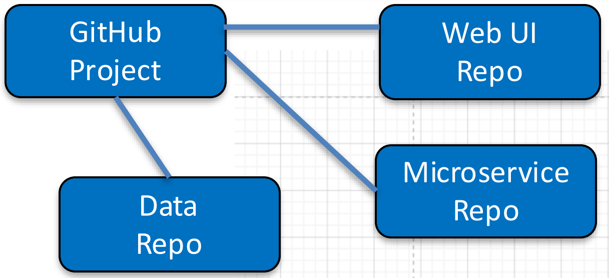
- OpenAPI: “The OpenAPI Specification, previously known as the Swagger Specification, is a specification for a machine-readable interface definition language for describing, producing, consuming and visualizing web services.” (https://en.wikipedia.org/wiki/OpenAPI_Specification)

- Model: “A model represents an entity of our application domain with an associated type.” (https://medium.com/@nicola88/your-first-openapi-document-part-ii-data-model-52ee1d6503e0)

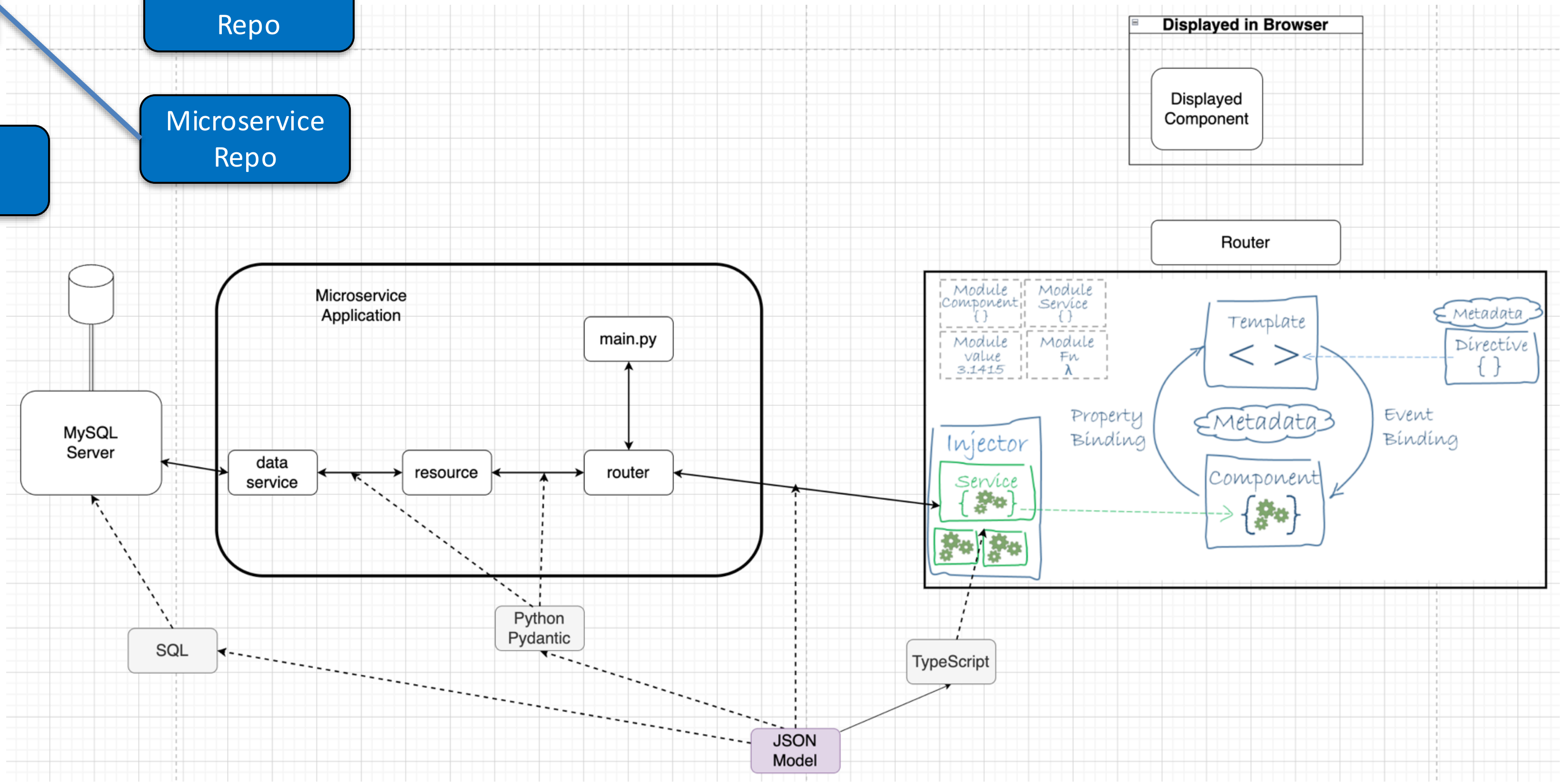
- Routers: “What fastapi docs says about routers: If you are building an application or a web API, it’s rarely the case that you can put everything on a single file. FastAPI provides a convenience tool to structure your application while keeping all the flexibility.” (https://medium.com/@rushikeshnaik779/routers-in-fastapi-tutorial-2-adf3e505fdca)

- Summary:
  – These are general concepts, and we will go into more detail in the semester.
  – FastAPI is a specific technology for Python.
  – There are many other frameworks applicable to Python, NodeJS/TypeScript, Go, C#, Java, … …
  – They all surface similar concepts with slightly different names.

Full Stack Application Structure



https://medium.com/@bhavikagarg8/angular-architecture-overview-1e7cc7483a0

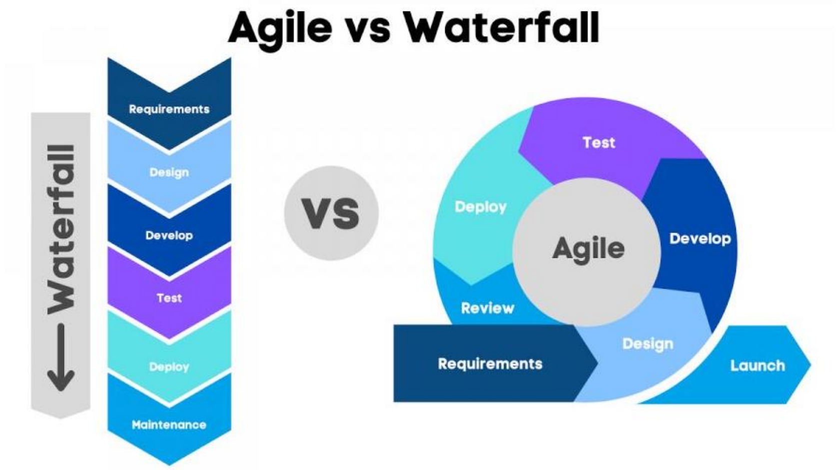


Code Walkthrough

Application Development Concepts

Agile Development

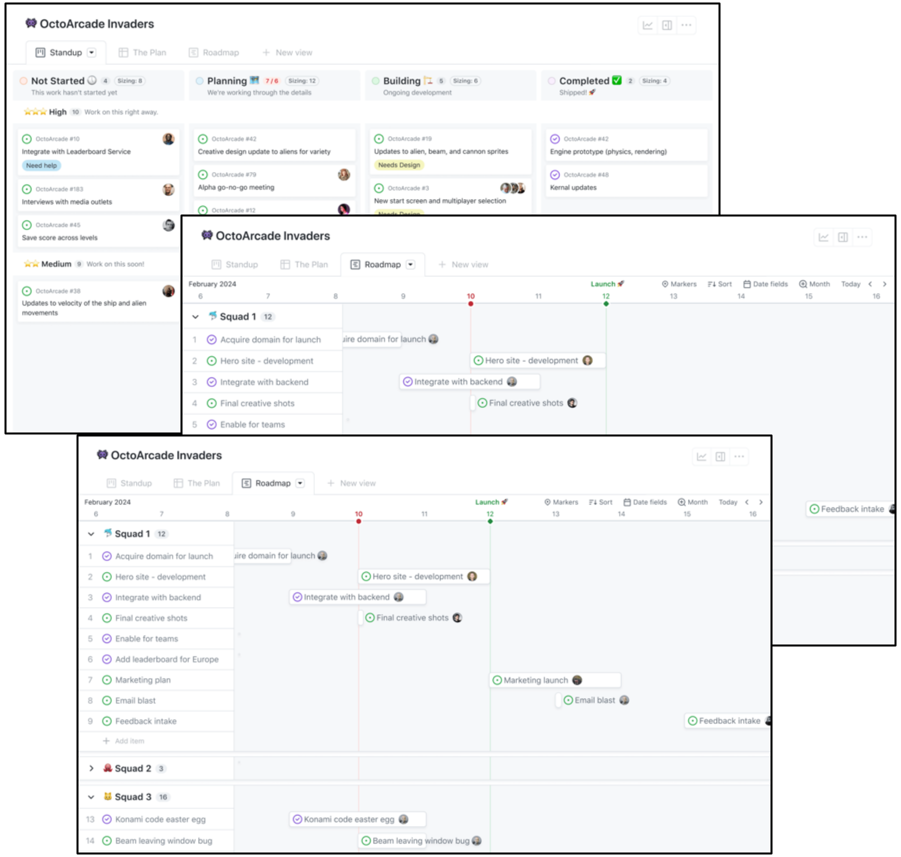
https://www.linkedin.com/pulse/waterfall-vs-agile-which-better-you-why-datacademy-cloud



- SW projects traditionally/ previously followed a waterfall model:
  - 12-18 months between releases.
  - Proceed through phases one at a time.
- Modern development follows a form of *agile development.*
  - A sequence of short sprints, with a working system at the end of each sprint.
  - Continuous, iterative, incremental refinement and improvement.
- In this course, project teams will execute in two-week sprints.
  - Start sprint by defining sprint objectives and user stories.
  - End of sprint status report and demo, which we will review.

Agile and GitHub Projects

- Teams will use GitHub Projects to track and manage their work.
  - Ideas
  - Issues
  - Backlog
  - Sprint
  - … …
- The "project" will reference subprojects and code repos.
- We will also get some experience with pull requests and CI/CD.



OctoArcade Invaders

Standup | The Plan | Roadmap | + New view

Not Started 4 Sizing: 8 — This work hasn't started yet
Planning 7 / 6 Sizing: 12 — We're working through the details
Building 5 Sizing: 6 — Ongoing development
Completed ✅ 2 Sizing: 4 — Shipped! 🚀

⭐⭐⭐ High 10 Work on this right away.

- Not Started
  - OctoArcade #10 — Integrate with Leaderboard Service — Need help
  - OctoArcade #183 — Interviews with media outlets
  - OctoArcade #45 — Save score across levels
- Planning
  - OctoArcade #42 — Creative design update to aliens for variety
  - OctoArcade #79 — Alpha go-no-go meeting
  - OctoArcade #12
- Building
  - OctoArcade #19 — Updates to alien, beam, and cannon sprites — Needs Design
  - OctoArcade #3 — New start screen and multiplayer selection — Needs Design
- Completed
  - OctoArcade #42 — Engine prototype (physics, rendering)
  - OctoArcade #48 — Kernal updates

⭐⭐ Medium 9 Work on this soon!

- OctoArcade #38 — Updates to velocity of the ship and alien movements

OctoArcade Invaders

Standup | The Plan | Roadmap | + New view

February 2024 — 6 7 8 9 10 11 12 13 14 15 16 — Launch 🚀 — Markers | Sort | Date fields | Month | Today

Squad 1 12
1. Acquire domain for launch
2. Hero site - development
3. Integrate with backend
4. Final creative shots
5. Enable for teams
6. Add leaderboard for Europe
7. Marketing plan
8. Email blast
9. Feedback intake

+ Add item

Squad 2 3

Squad 3 16

13. Konami code easter egg
14. Beam leaving window bug

First (Individual) Project

First Sprint “HW 0” done Individually

- TBD