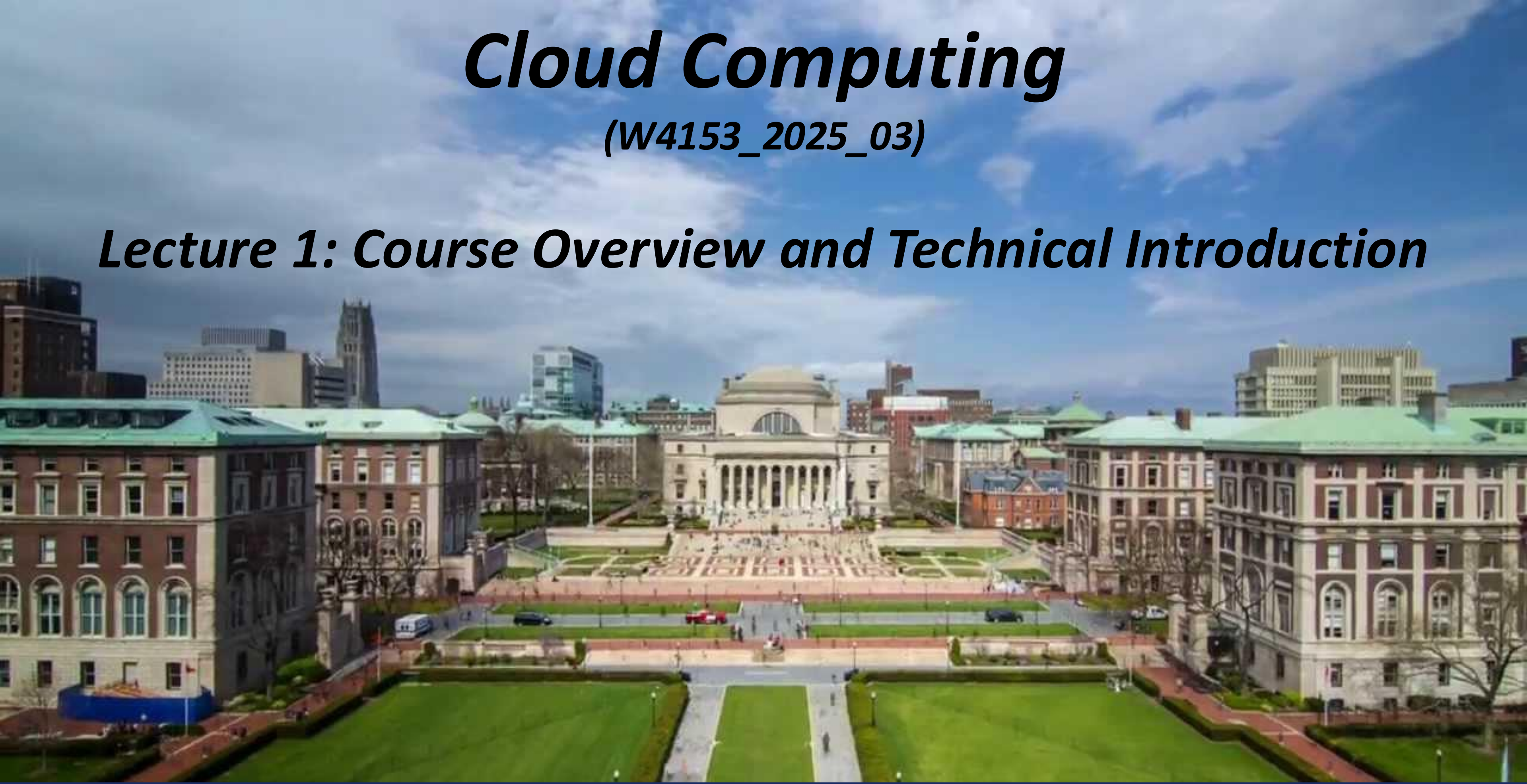


Cloud Computing

(W4153_2025_03)

Lecture 1: Course Overview and Technical Introduction

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

Course Overview

Homework and Grading

- There are no exams. This is a projects course.
- Projects
  - You will form 5 person teams. Each team
    - Will have a "name."
    - Designated focal point/contact point for communication from instructor and TAs.
    - Everyone on the project gets the same grade.
  - You can define your own projects, i.e. the application you want to implement.
  - I will identify a set of concepts presented in lectures.
    Your project will receive "points" for each concept successfully included in your project's implementation.
- Grading
  - The point total achieved on the final project determines the grade.
  - You will submit a team written report and give a presentation/demo.
  - You will implement your project in two week "sprints."
    - You submit a status report at the end of each sprint. I will give you a template.
    - Your team must submit status reports. We will deduct one or two points if your report is incomplete.

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. 19 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.

About Your TAs

- Many of the TAs have taken cloud computing with me.
  They can relate to your experience in the class and help you be successful.

- The remainder have real world experience with cloud, projects, etc.

- I ask them to be your advocates.
  If you are struggling, please contact them and/or me.

- Each team will have an assigned TA and meet with the TA every sprint.

Course Objectives

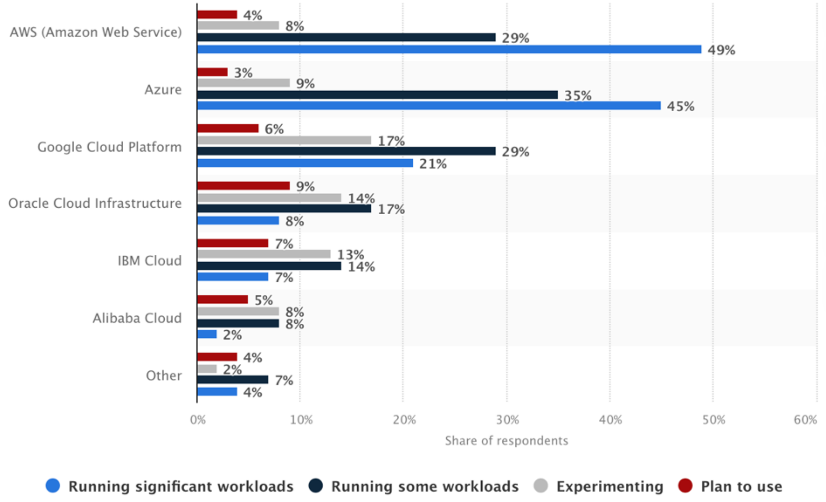
- Cloud computing:
  - Has become mainstream and foundational to modern applications and IT systems.
  - The structure of cloud applications and infrastructure are fundamentally different and more advanced than most students experience in core CS classes.
- By the end of the course:
  - You will understand many of the critical, core cloud and SW concepts and how to apply them to a solution.
  - You will be prepared to use the technology in other courses, projects and employment.
- Additionally,
  - We should have fun. This is an amazing area.
  - You should have a small “capstone” project that you can reference in interviews.

Lectures, Office Hours and Reading Material

- Lectures are Friday, 1:10 PM to 3:00 PM
  - The allocated time slot is 1:10 PM to *3:40* PM.
  - We will use the 3:00 to 3:40 time slot for discussions, help on projects, etc.
  - Also, there is a lot of independent reading, study and research, … …
- I will record and stream the lectures.
- Attendance
  - I travel 3.5 hours round trip to lecture in person.
  - There are students that cannot get into this class because classroom size limits the enrollment in the class.
  - There will be times when you cannot attend in person, and there will be lectures where I cannot attend in person.
  - Basic respect and courtesy means that you should make every effort to attend unless there is a compelling reason not to attend.
  - If attendance drops to ridiculously low levels, which it will, I will do something crazy like take attendance of have a "pop quiz."

Student Resources, Environment and Systems

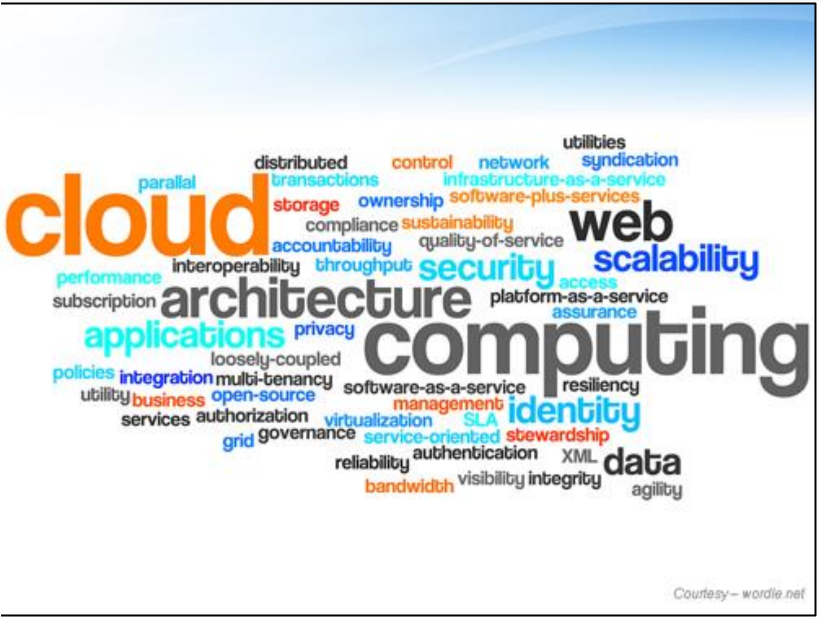
- Environment and systemsThere are 3 primary cloud service p
  - Microsoft Azure
  - Google Cloud Platform
  - Amazon Web Services
- The course will focus on concepts and use the clouds for implementations.
- Managing costs:
  - All clouds have “free tiers” and/or “new user” credits.
    - **You must track and manage your costs.**
    - **You must set cost controls and alerts.**
  - We have credits for Google Cloud Platform.
- I suggest that you also mirror/follow my usage of locally installed SW for development:
  - PyCharm, DataGrip, WebStorm including cloud usage plugins.
  - Docker, Minikube, … …
  - Others will follow during semester.
- Resources
  - There is no textbook.
  - I will provide links to articles, tutorials, etc. for independent work.

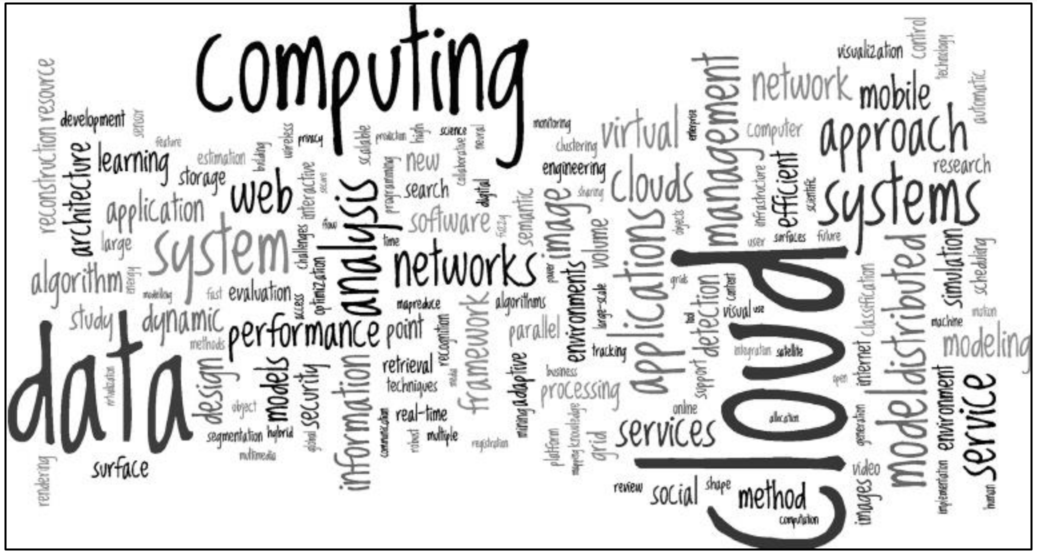


Introduction to Core Concepts

Cloud Computing and Taxonomy

Cloud Computing Concepts





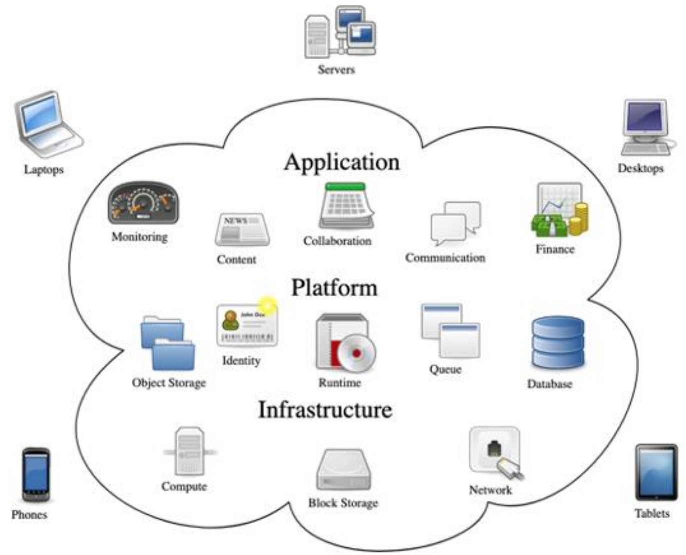
- There are many, many cloud concepts, and many different perspectives on each concept.
- Most cloud service providers offer their own specific versions of concepts and technology – But the core of the concept is similar.
- We explore as many as we can by writing a “complex” application using the technology.
  - 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.

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

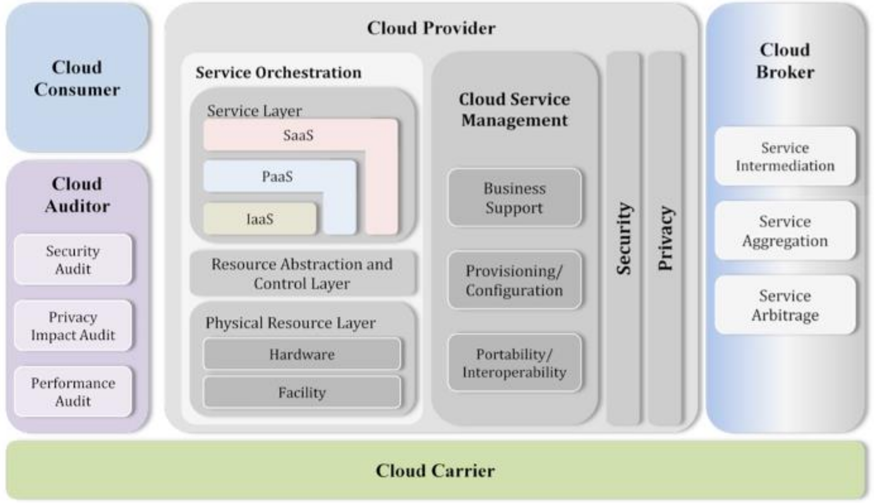
- "Simply put, **cloud computing** is the delivery of computing services—including servers, storage, databases, networking, software, analytics, and intelligence—over the internet ("the cloud") to offer faster innovation, flexible resources, and economies of scale. You typically pay only for cloud services 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)

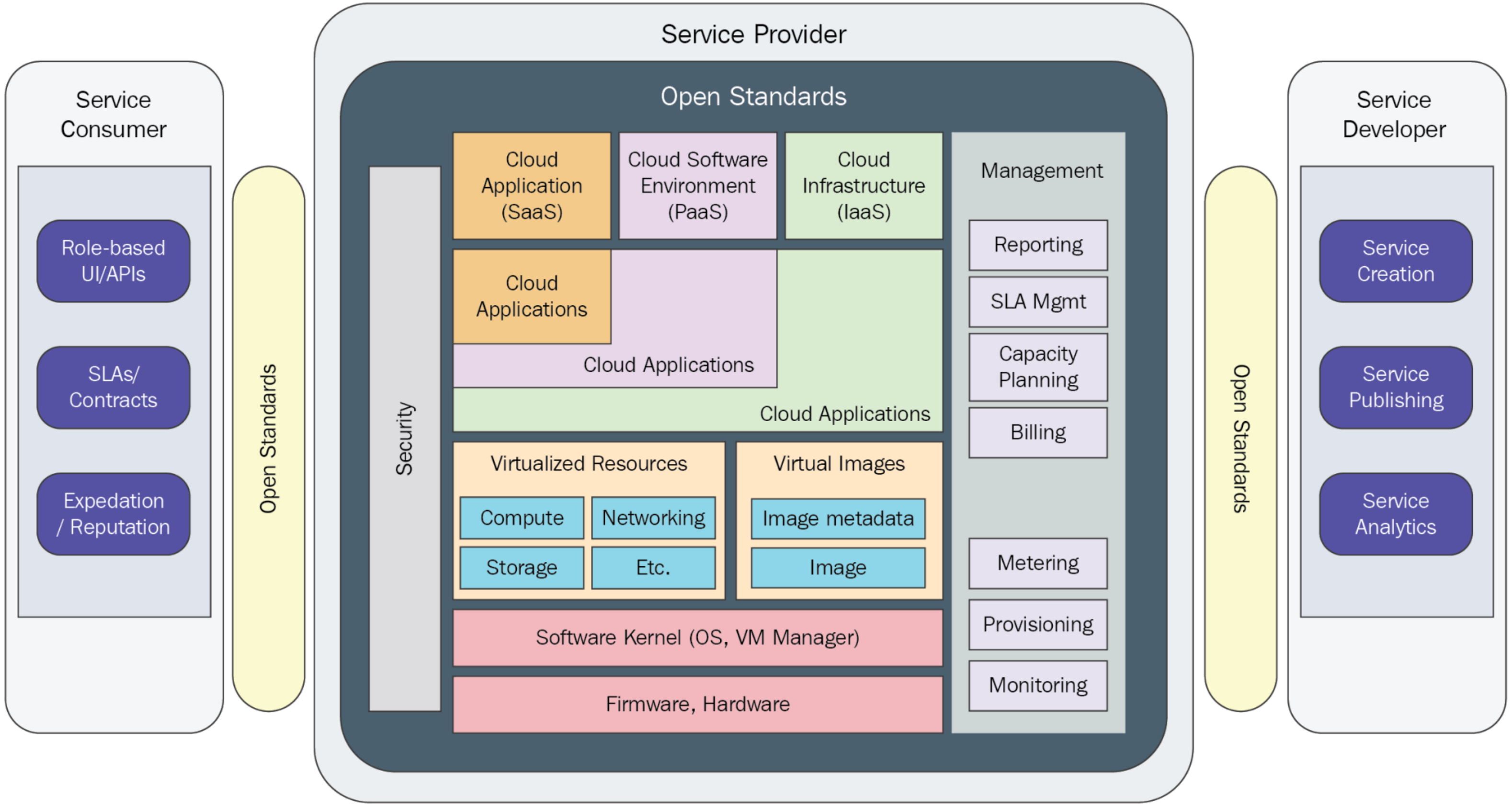
Cloud Computing



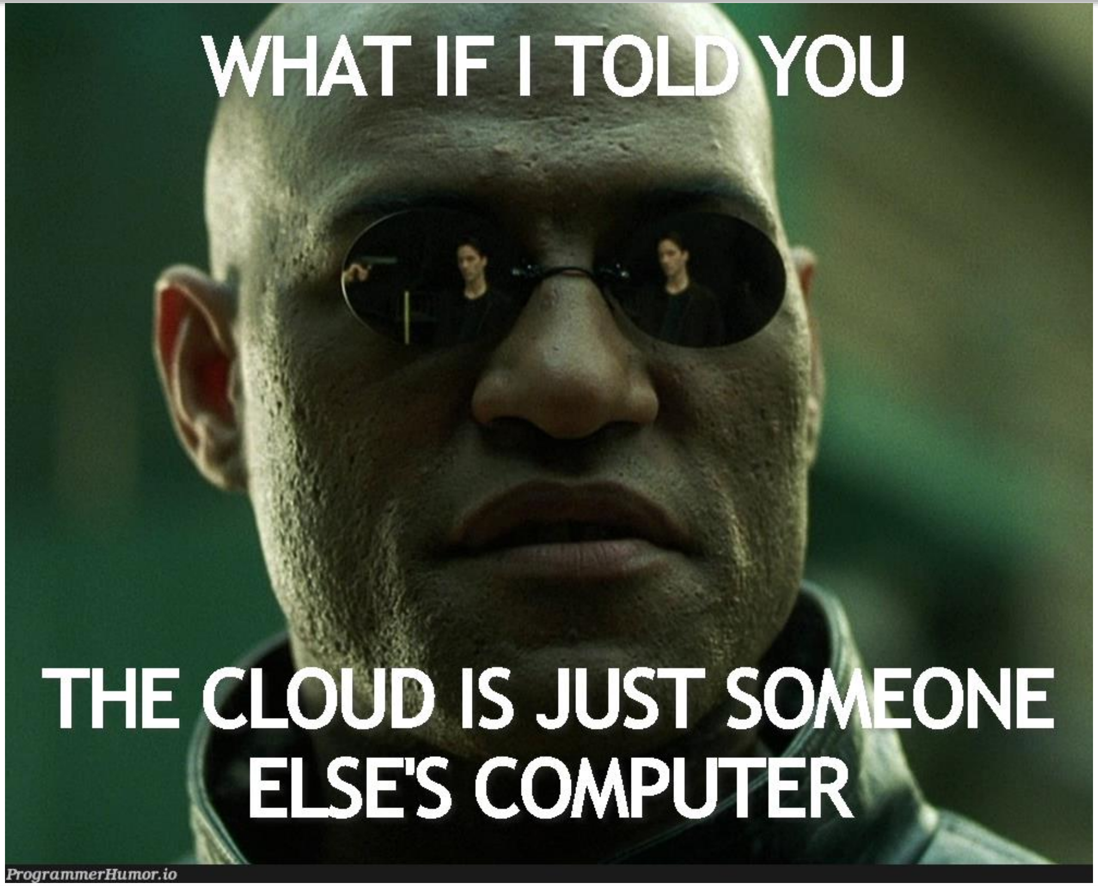
NIST Reference Architecture



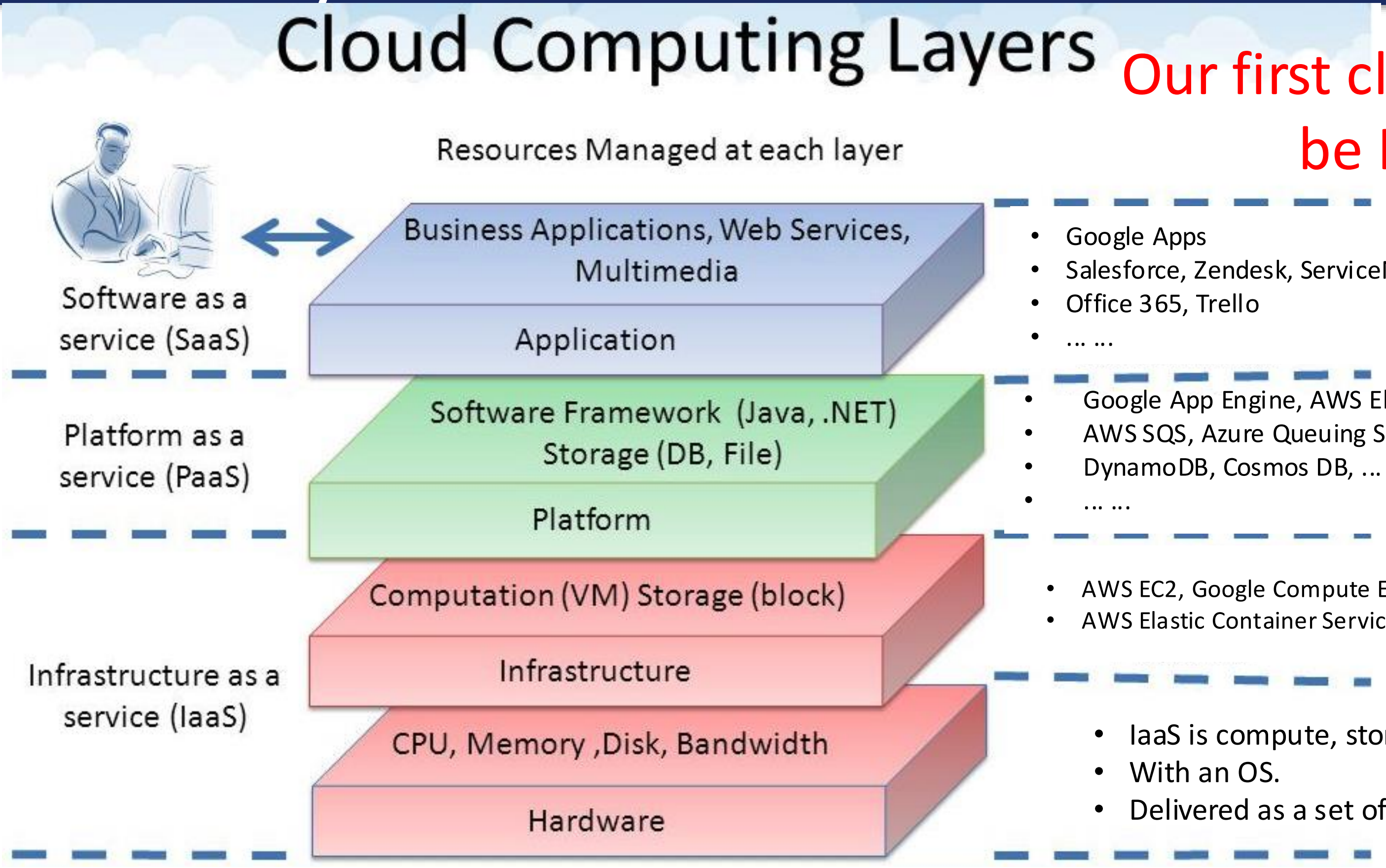
- There are **a lot** of concepts, aspects, types, … …
- Some conceptual models for how it comes together.
- Many applications and solutions build on all kinds of things from many clouds.
- We will see examples of technology and use during the semester.


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

Our first cloud elements will be IaaS (VMs).

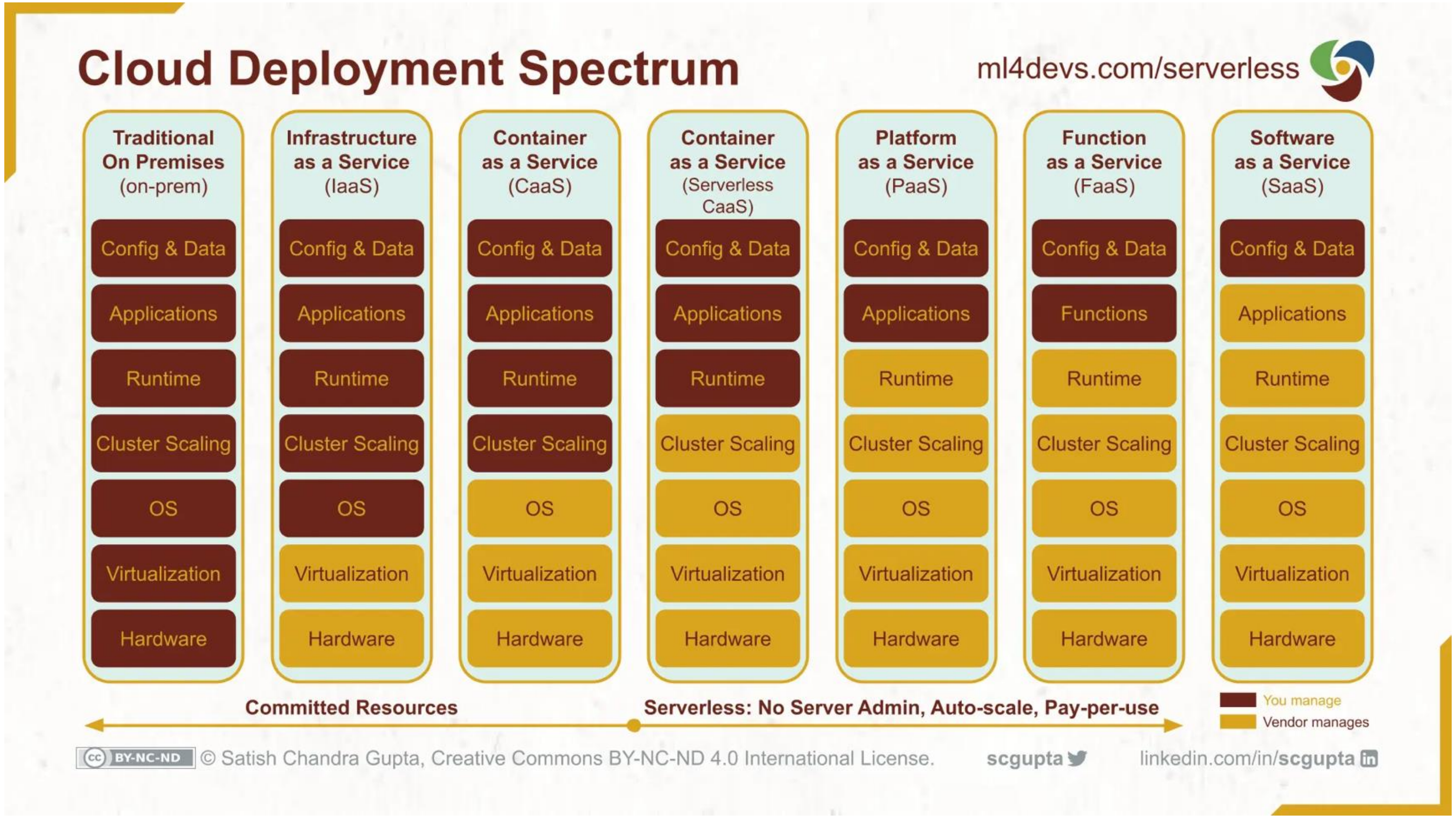
- Google Apps
- Salesforce, Zendesk, ServiceNow, Concur, … …
- Office 365, Trello
- … …

- 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, … …

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

Cloud Layers



Cloud Deployment Spectrum

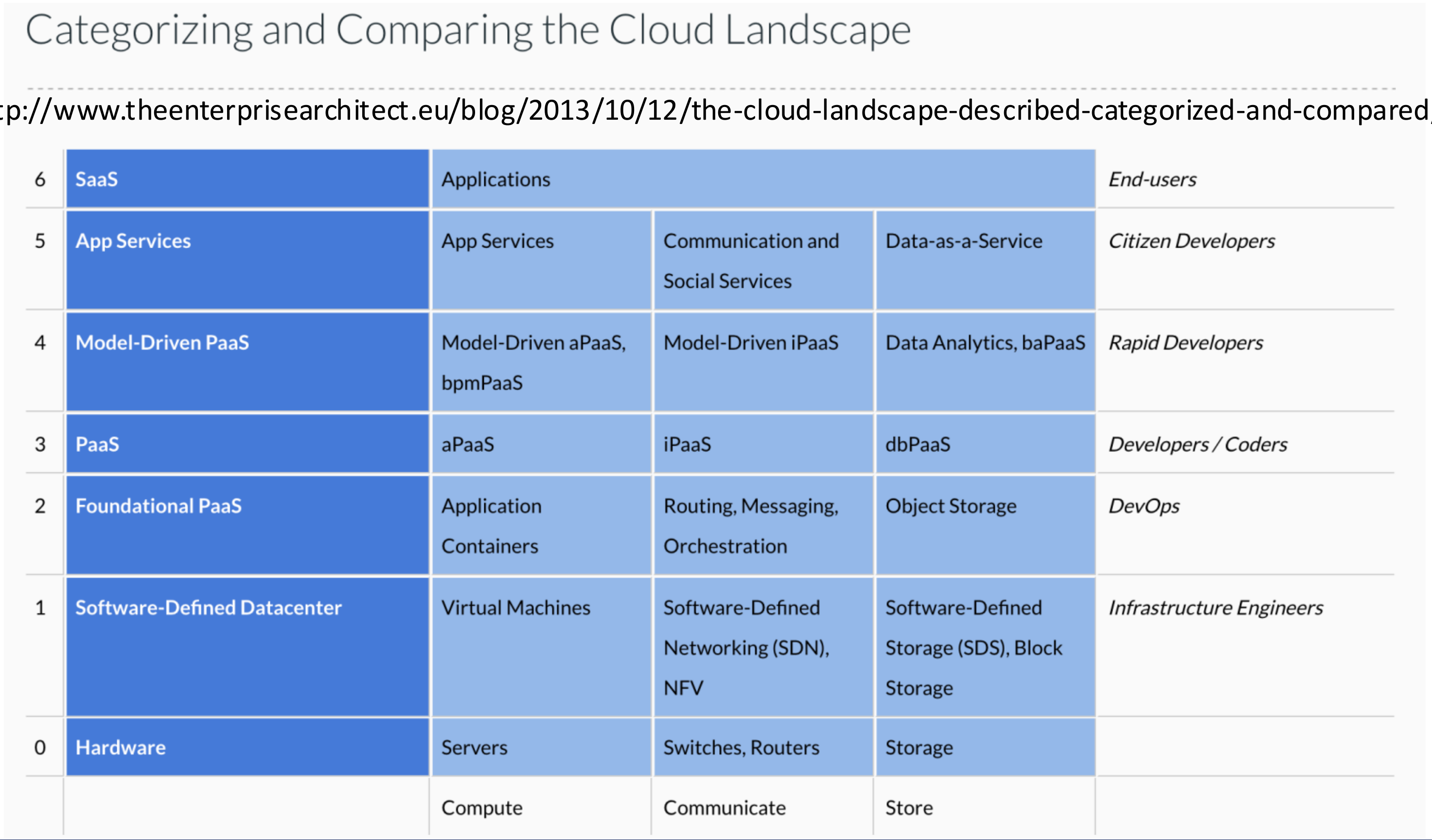
ml4devs.com/serverless

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

Committed Resources ← → Serverless: No Server Admin, Auto-scale, Pay-per-use

🟥 You manage
🟨 Vendor manages

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/

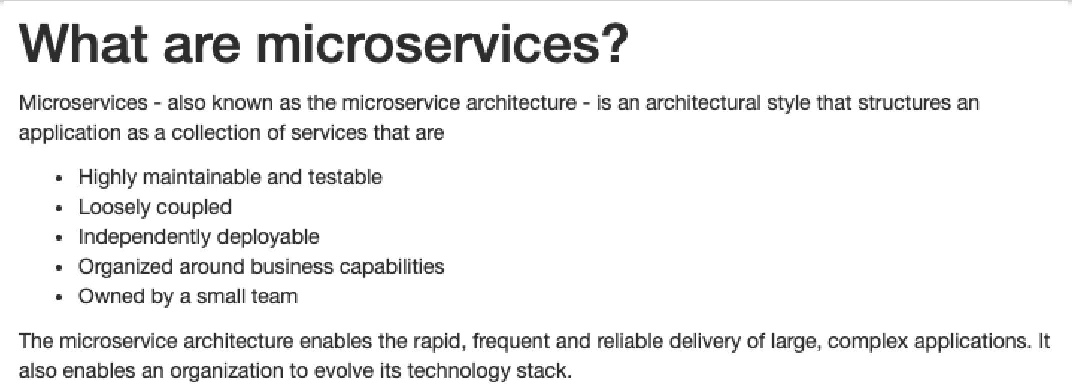
6	SaaS	Applications			*End-users*
5	App Services	App Services	Communication and Social Services	Data-as-a-Service	*Citizen Developers*
4	Model-Driven PaaS	Model-Driven aPaaS, bpmPaaS	Model-Driven iPaaS	Data Analytics, baPaaS	*Rapid Developers*
3	PaaS	aPaaS	iPaaS	dbPaaS	*Developers / Coders*
2	Foundational 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.

Microservices

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



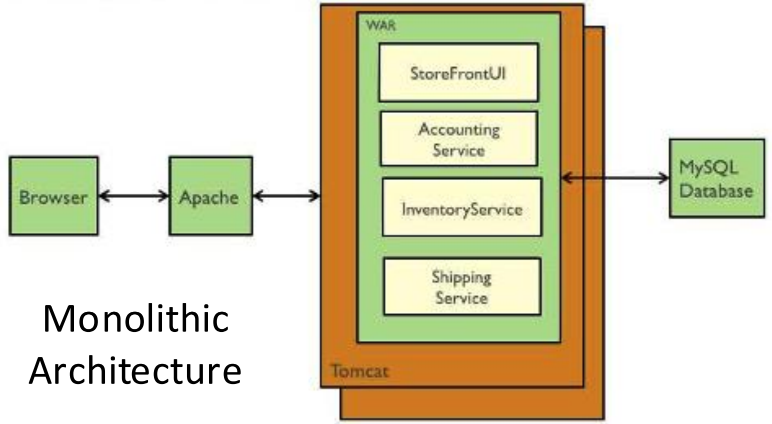
What are microservices?

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

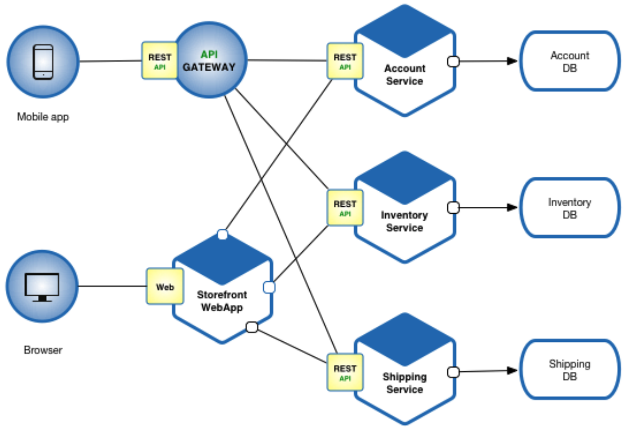
The microservice architecture enables the rapid, frequent and reliable delivery of large, complex applications. It also enables an organization to evolve its technology stack.

- 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)

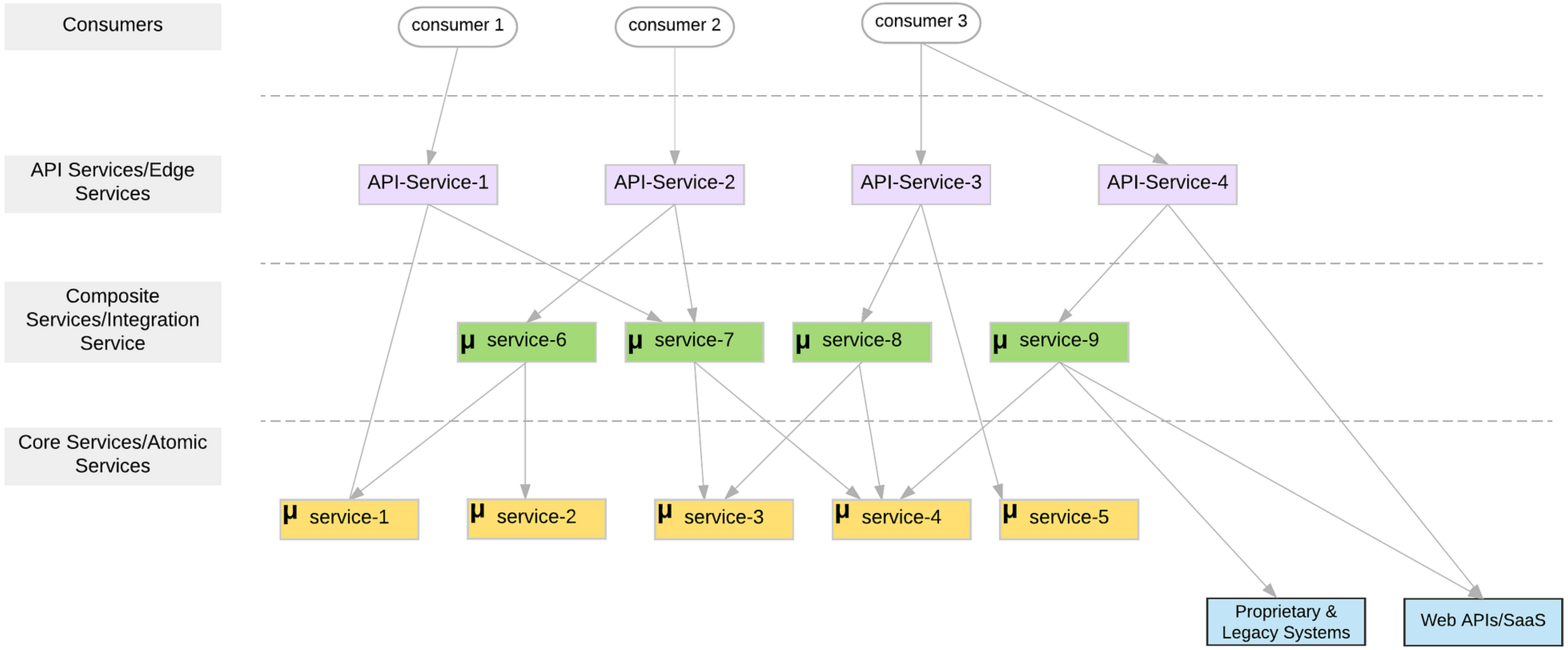
Decoupling
1
Componentization
2
Business Capabilities
3
Autonomy
4
5
Continous Delivery
6
Responsibility
7
Decentralized Governance
8
Agility

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)

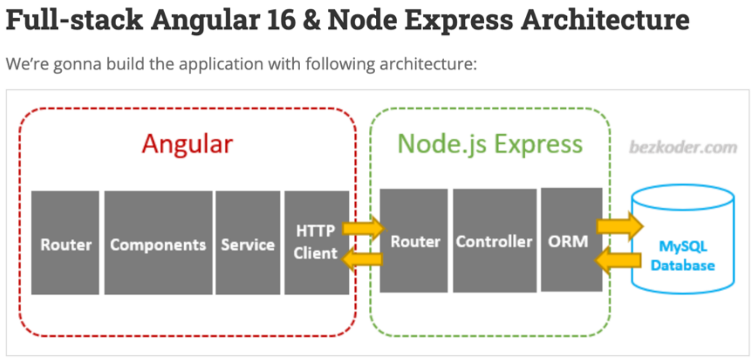


Microservice Details and Perspectives

- 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.

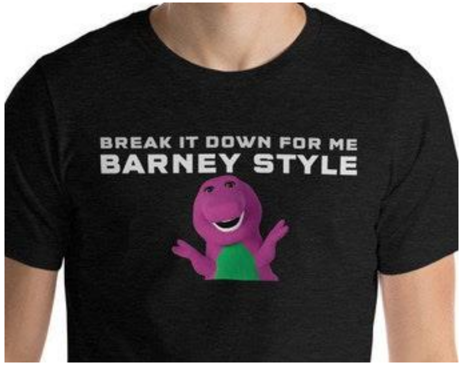
Walkthrough of a Set of Examples

- 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
  –

Representation State Transfer (REST)
Part 1

REST (https://restfulapi.net/)

What is REST

- REST is acronym for **RE**presentational **S**tate **T**ransfer. It is architectural style for **distributed hypermedia systems** and was first presented by Roy Fielding in 2000 in his famous dissertation.
- Like any other architectural style, REST also does have it's own 6 guiding constraints which must be satisfied if an interface needs to be referred as **RESTful**. These principles are listed below.

Guiding Principles of REST

- **Client–server** – By separating the user interface concerns from the data storage concerns, we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components.
- **Stateless** – Each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client.
- **Cacheable** – Cache constraints require that the data within a response to a request be implicitly or explicitly labeled as cacheable or non-cacheable. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests.
- **Uniform interface** – By applying the software engineering principle of generality to the component interface, the overall system architecture is simplified and the visibility of interactions is improved. In order to obtain a uniform interface, multiple architectural constraints are needed to guide the behavior of components. REST is defined by four interface constraints: identification of resources; manipulation of resources through representations; self-descriptive messages; and, hypermedia as the engine of application state.
- **Layered system** – The layered system style allows an architecture to be composed of hierarchical layers by constraining component behavior such that each component cannot "see" beyond the immediate layer with which they are interacting.
- **Code on demand (optional)** – REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts. This simplifies clients by reducing the number of features required to be pre-implemented.

Resources

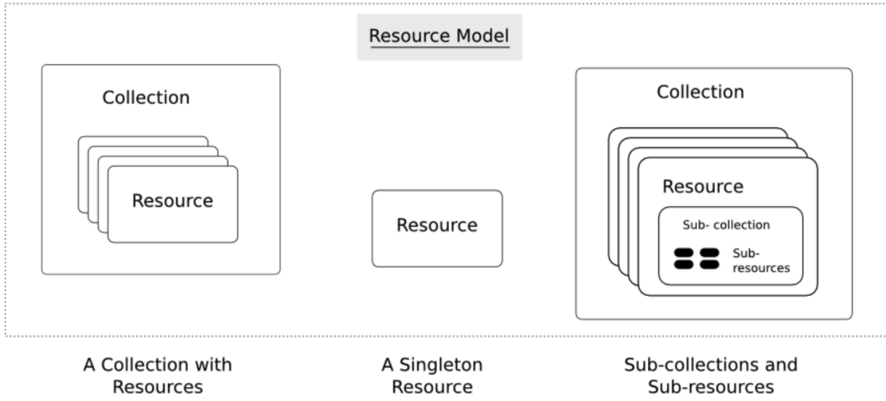
Resources are an abstraction. The application maps to create things and actions.

"A resource-oriented API is generally modeled as a resource hierarchy, where each node is either a *simple resource* or a *collection resource*. For convenience, they are often called a resource and a collection, respectively.

- A collection contains a list of resources of **the same type**. For example, a user has a collection of contacts.
- A resource has some state and zero or more sub-resources. Each sub-resource can be either a simple resource or a collection resource.

For example, Gmail API has a collection of users, each user has a collection of messages, a collection of threads, a collection of labels, a profile resource, and several setting resources.

While there is some conceptual alignment between storage systems and REST APIs, a service with a resource-oriented API is not necessarily a database, and has enormous flexibility in how it interprets resources and methods. For example, creating a calendar event (resource) may create additional events for attendees, send email invitations to attendees, reserve conference rooms, and update video conference schedules. (Emphasis added)
(https://cloud.google.com/apis/design/resources#resources)



https://restful-api-design.readthedocs.io/en/latest/resources.html

REST – Resource Oriented

- When writing applications, we are used to writing functions or methods:
  - openAccount(last_name, first_name, tax_payer_id)
  - account.deposit(deposit_amount)
  - account.close()

  We can create and implement whatever functions we need.
- REST only allows four methods:
  - POST: Create a resource
  - GET: Retrieve a resource
  - PUT: Update a resource
  - DELETE: Delete a resource

  That's it. That's all you get.

"The key characteristic of a resource-oriented API is that it emphasizes resources (data model) over the methods performed on the resources (functionality). A typical resource-oriented API exposes a large number of resources with a small number of methods." (https://cloud.google.com/apis/design/resources)

- A REST client needs no prior knowledge about how to interact with any particular application or server beyond a generic understanding of hypermedia.

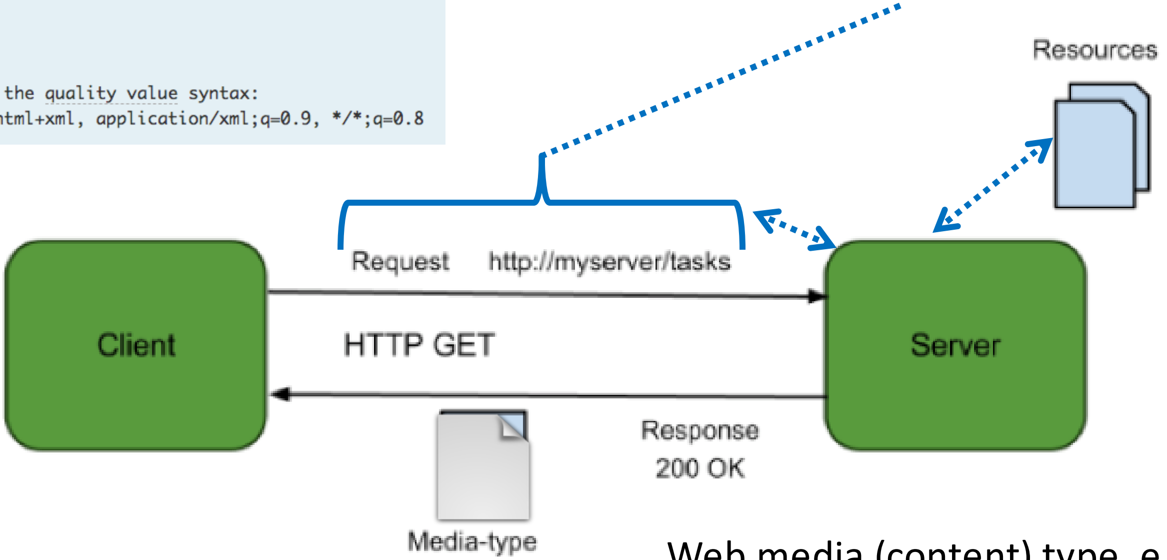
Resources, URLs, Content Types

Accept type in headers.

```
Accept: <MIME_type>/<MIME_subtype>
Accept: <MIME_type>/*
Accept: */*

// Multiple types, weighted with the quality value syntax:
Accept: text/html, application/xhtml+xml, application/xml;q=0.9, */*;q=0.8
```

- Relative URL identifies "resource" on the server.
- Server implementation maps abstract resource to tangible "thing," file, DB row, … and any application logic.



Client may be

Browser

Mobile device

Other REST Service

… …

Web media (content) type, e.g.

- text/html
- application/json

REST Principles

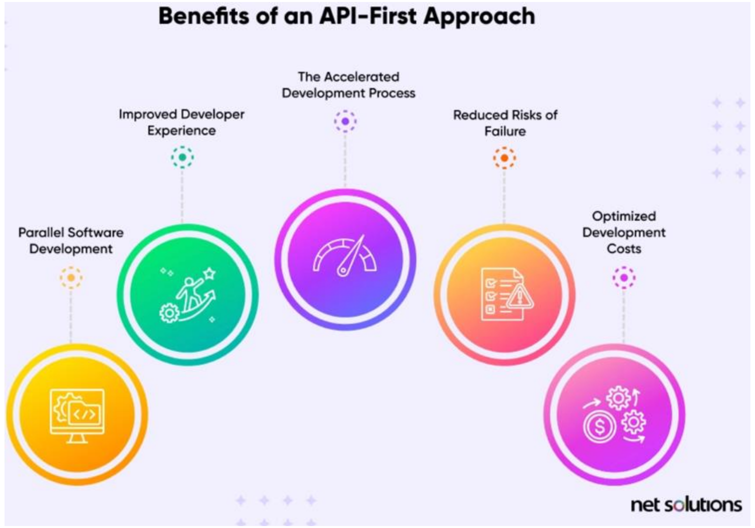
- What about all of those principles?
  - Client/Server
  - Stateless
  - Cacheable
  - Uniform Interface
  - Layered System
  - Code on demand
- Some of these principles
  - Are simple and obvious.
  - Are subtle and have major implications.
  - Stateless can be baffling, but we will cover later.
- We will go into the principles in later lectures, ...
- We will also go into HATEOAS, GraphQL, … …

API First

- “As the connective tissue linking ecosystems of technologies and organizations, APIs allow businesses to monetize data, forge profitable partnerships, and open new pathways for innovation and growth.” (McKinsey)
- “API-first, also called the API-first approach, prioritizes APIs at the beginning of the software development process, positioning APIs as the building blocks of software. API-first organizations develop APIs before writing other code, instead of treating them as afterthoughts. This lets teams construct applications with internal and external services that are delivered through APIs.” (https://www.netsolutions.com/hub/mach-architecture/api-first)

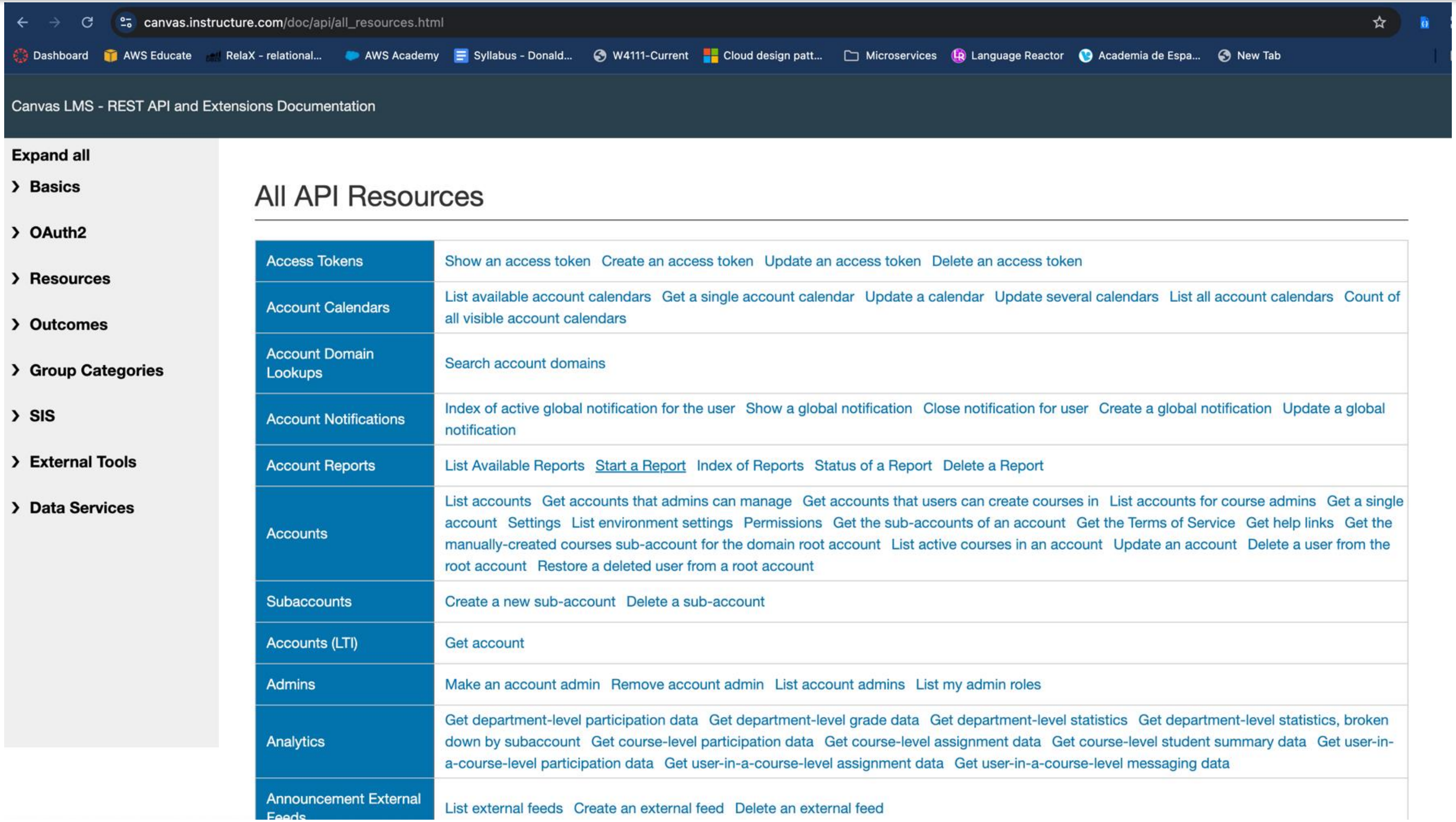
Why API-First Approach?

- By defining the APIs first, there is a modular separation between the services in the system. This increases the reusability of API endpoints across different parts of the system and to external entities as well (Interoperability).
- The API-first approach promotes collaboration as API documentation and contract definitions serve as a common language for stakeholders such as different teams in development (e.g. frontend and backend), V&V teams for test scenarios, etc.
- Creating the API contracts as part of the API-first design approach can be shared with frontend teams where the team can start building user interfaces parallelly with mocked data as per contract models, tested for the user experience. Also, codegen can be used to generate client mocks which can be directly used by client apps such as iOS/Android, etc.
- It gives the flexibility for tech-stack choices. Different services or applications in the system can be in different languages as long as they adhere to API contracts. It can help in identifying the best suitable tools/options in terms of cost and performance.
- With the API-First approach many concerns such as security measures, versioning, request traffic and data volume, and Integration with different services/applications within the system.


Benefits of an API-First Approach
Parallel Software Development
Improved Developer Experience
The Accelerated Development Process
Reduced Risks of Failure
Optimized Development Costs
net solutions

https://medium.com/@rohitranjan.pandey/api-first-design-approach-swagger-a1423db48f76

CourseWorks/Canvas LMS API



canvas.instructure.com/doc/api/all_resources.html

Dashboard | AWS Educate | RelaX - relational... | AWS Academy | Syllabus - Donald... | W4111-Current | Cloud design patt... | Microservices | Language Reactor | Academia de Espa... | New Tab

Canvas LMS - REST API and Extensions Documentation

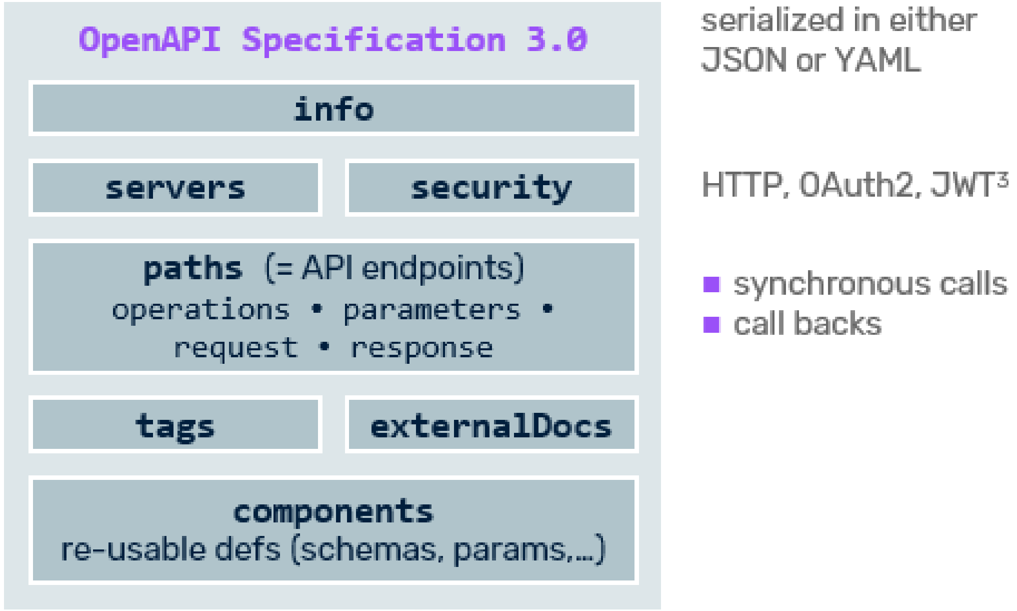
Expand all
- **Basics**
- **OAuth2**
- **Resources**
- **Outcomes**
- **Group Categories**
- **SIS**
- **External Tools**
- **Data Services**

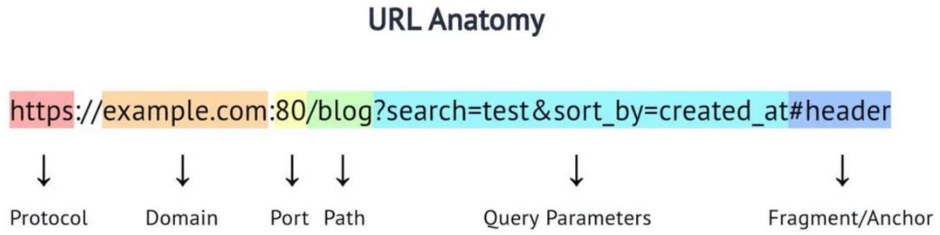
All API Resources

Access Tokens	Show an access token Create an access token Update an access token Delete an access token
Account Calendars	List available account calendars Get a single account calendar Update a calendar Update several calendars List all account calendars Count of all visible account calendars
Account Domain Lookups	Search account domains
Account Notifications	Index of active global notification for the user Show a global notification Close notification for user Create a global notification Update a global notification
Account Reports	List Available Reports Start a Report Index of Reports Status of a Report Delete a Report
Accounts	List accounts Get accounts that admins can manage Get accounts that users can create courses in List accounts for course admins Get a single account Settings List environment settings Permissions Get the sub-accounts of an account Get the Terms of Service Get help links Get the manually-created courses sub-account for the domain root account List active courses in an account Update an account Delete a user from the root account Restore a deleted user from a root account
Subaccounts	Create a new sub-account Delete a sub-account
Accounts (LTI)	Get account
Admins	Make an account admin Remove account admin List account admins List my admin roles
Analytics	Get department-level participation data Get department-level grade data Get department-level statistics Get department-level statistics, broken down by subaccount Get course-level participation data Get course-level assignment data Get course-level student summary data Get user-in-a-course-level participation data Get user-in-a-course-level assignment data Get user-in-a-course-level messaging data
Announcement External Feeds	List external feeds Create an external feed Delete an external feed

OpenAPI 3.0

- OpenAPI is a standard for documenting REST APIs.
- An HTTP request/response has several elements:
  - Protocol
  - Server address and port
  - Path
  - Query parameters
  - Fragment
  - Headers, Cookies
  - Body content and content type
  - Method
- OpenAPI and web application frameworks introduce:
  - Path parameters
  - Models to represent bodies/content
  - Security information
  - Descriptive metadata


OpenAPI Specification 3.0
info
servers
security
paths (= API endpoints)
operations • parameters • request • response
tags
externalDocs
components
re-usable defs (schemas, params,...)
serialized in either JSON or YAML
HTTP, OAuth2, JWT³
synchronous calls
call backs

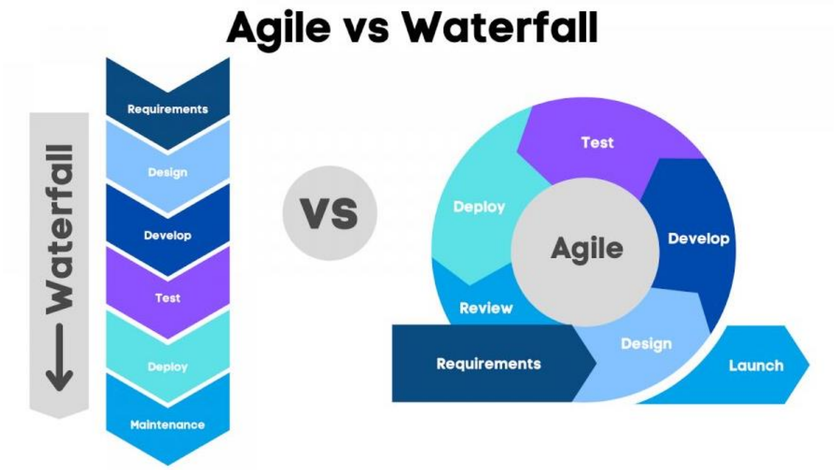

URL Anatomy
https://example.com:80/blog?search=test&sort_by=created_at#header
Protocol
Domain
Port
Path
Query Parameters
Fragment/Anchor

~~Architecture and Modeling~~

Agile Development

Agile Development

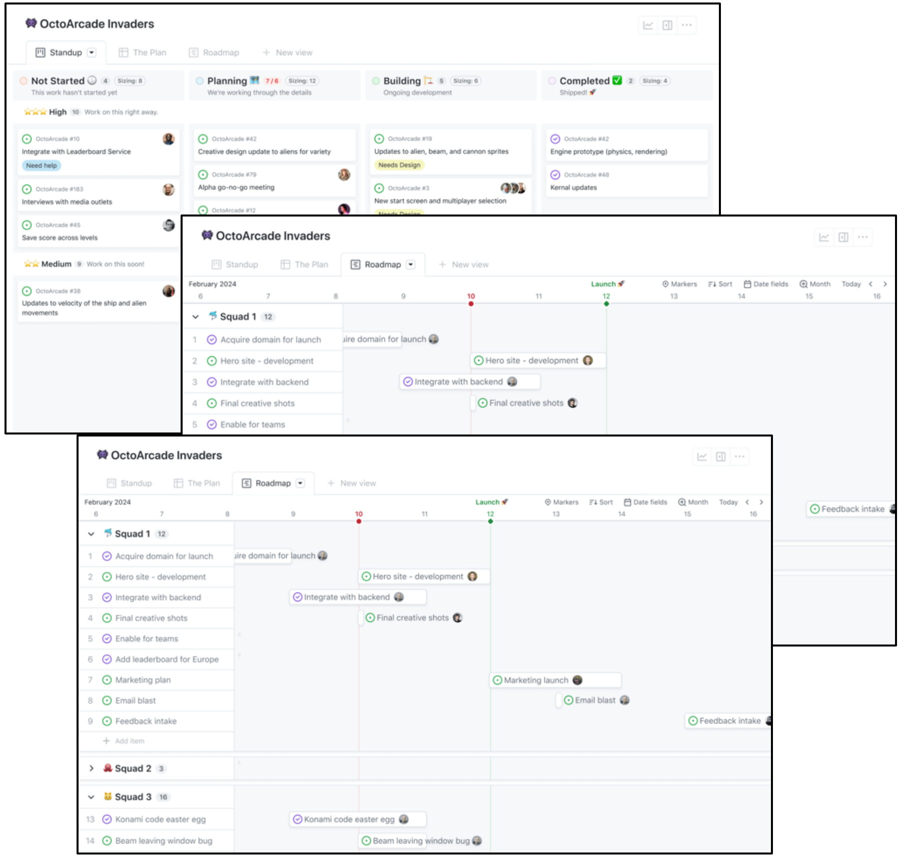
https://www.linkedin.com/pulse/waterfall-vs-agile-which-better-you-why-datacademy-cloud



- 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.			
OctoArcade #10 Integrate with Leaderboard Service — Need help	OctoArcade #42 Creative design update to aliens for variety	OctoArcade #19 Updates to alien, beam, and cannon sprites — Needs Design	OctoArcade #42 Engine prototype (physics, rendering)
OctoArcade #183 Interviews with media outlets	OctoArcade #79 Alpha go-no-go meeting	OctoArcade #3 New start screen and multiplayer selection	OctoArcade #48 Kernal updates
OctoArcade #45 Save score across levels	OctoArcade #12		
⭐⭐ 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

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

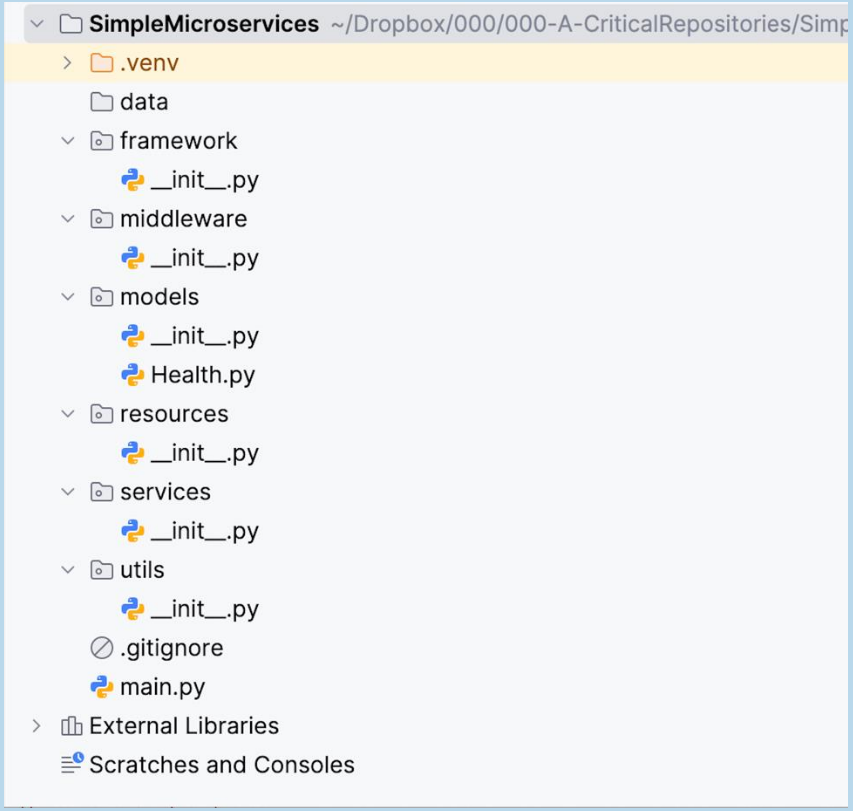
Squad 1 12
1 Acquire domain for launch
2 Hero site - development
3 Integrate with backend
4 Final creative shots
5 Enable for teams
6 Add leaderboard for Europe
7 Marketing plan
8 Email blast
9 Feedback intake
+ Add item

Squad 2 3

Squad 3 16
13 Konami code easter egg
14 Beam leaving window bug

Next Steps

Simple Project Structure – Assignment 1



```
SimpleMicroservices ~/Dropbox/000/000-A-CriticalRepositories/Simp
  .venv
  data
  framework
    __init__.py
  middleware
    __init__.py
  models
    __init__.py
    Health.py
  resources
    __init__.py
  services
    __init__.py
  utils
    __init__.py
  .gitignore
  main.py
External Libraries
Scratches and Consoles
```

- This is an individual assignment.
  - You will implement and test:
    - The FastAPI project that I demoed in class.
    - A GitHub repository for the code.
    - A project and empty Kanban.
  - This will require some independent study and reading, especial for metadata annotation and models.
  - This creates a template project that you can use for microservices that you implement in your project.
- Submission
  - Ed and GradeScope will define the submission format.
  - Due at 11:59 PM 14-SEP-2025