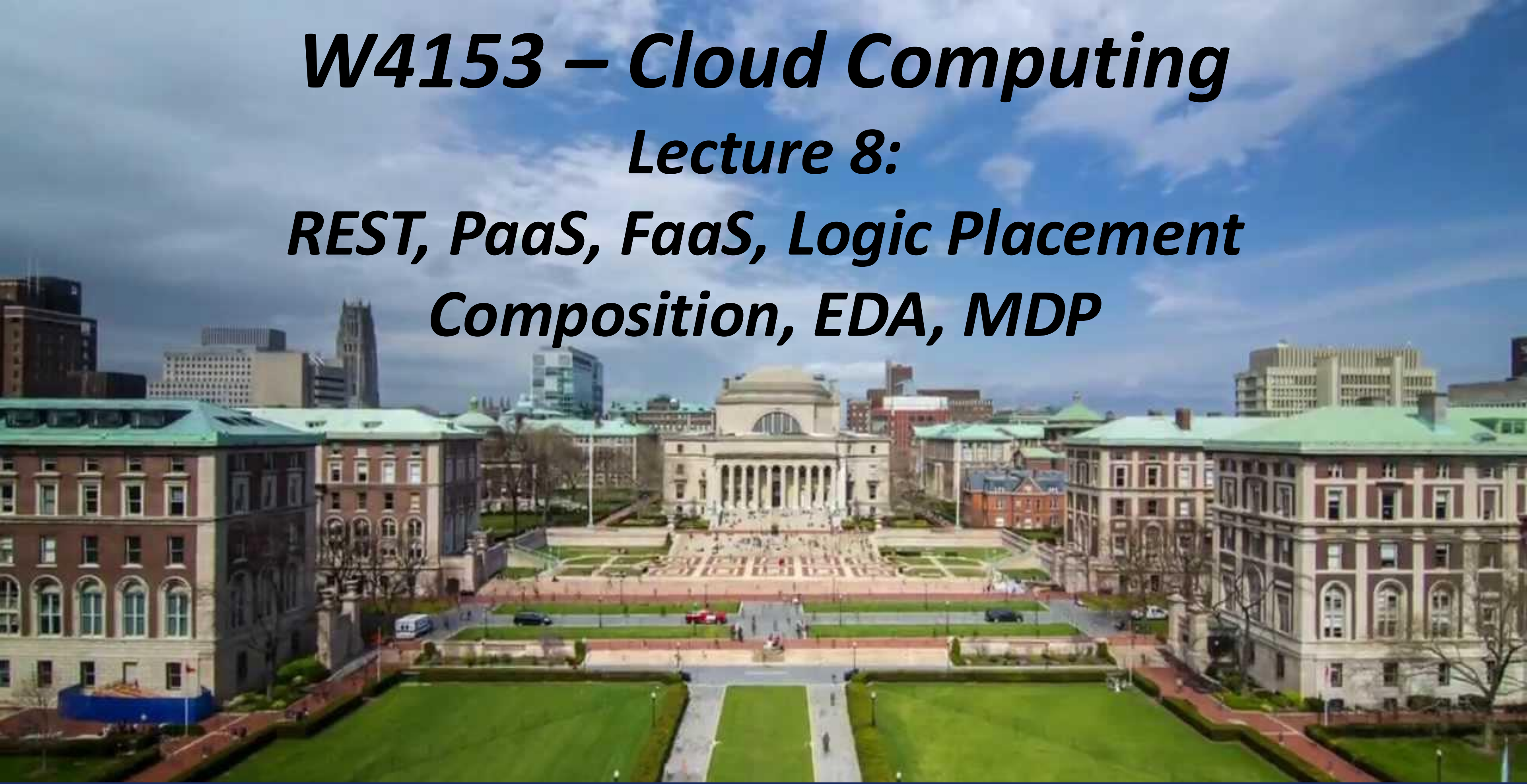
W4153 – Cloud Computing

Lecture 8:

REST, PaaS, FaaS, Logic Placement Composition, EDA, MDP



COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Contents

- EDA Medium Article: https://medium.com/@sujit.code90/what-is-event-driven-architecture-eda-1763ec03cacc

Contents

- A note on grading.
- REST continued:
  - Review principals of REST and HATEOAS
  - Idempotent
  - Linked resources ➔ Initial motivation for GraphQL
- Platform-as-a-Service, Function-as-a-Service
  - Nested doll model
  - PaaS overview
  - FaaS overview
- Application Development Concepts
  - ”My” development framework and style
  - ~~Overall microservice project structure~~
  - ~~Where does application logic go? UI? Microservice? Database?~~
- ~~Composition continued~~
  - ~~Architecture of my next logical sprint~~
  - ~~Orchestration vs Choreography~~
  - ~~Models: synchronous code, asynchronous code, event driven, queued, “workflow”~~
- Sprint 2 discussion

- I published some examples on GitHub/Ed.

- Next Lecture.
- I want to give you time to get the basic composition pulled together.

A Note on Grading

Grading Policy

- The review and grading of your final project *determines your grade.*
  - What project or application scenario you choose does not matter.
  - We cover a set of topics in class.
  - Your “correctly” using and applying the concepts in your project/application determines your grade on the final project.
  - We will provide a “checklist” to guide you.
  - The documented sprint requirements incrementally define the final checklist.
- Why do we require the “homework assignments” and “grade them.”
  - It allows us to assess how well the class and teams are progressing.
  - It allows us to provide incremental, continuous feedback.
  - **There is no reason to request a regrade.**
    Requesting an explanation or details does not make sense.

REST Continued

Reminder

REST (https://restfulapi.net/)

What is REST

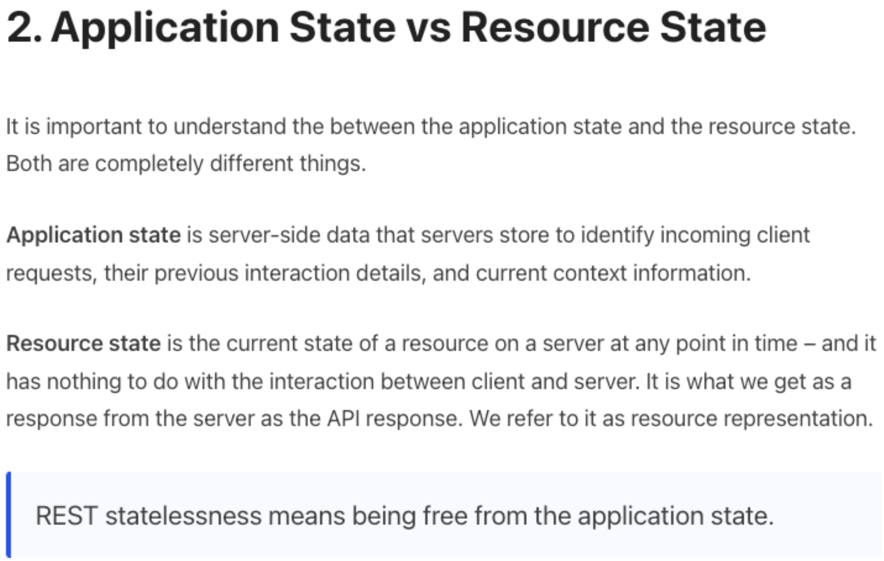
- REST is acronym for **RE**presentational **S**tate **T**ransfer. It is architectural style for **distributed hypermedia systems** and was first presented by Roy Fielding in 2000 in his famous dissertation.
- Like any other architectural style, REST also does have it's own 6 guiding constraints which 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.

Stateless

REST is Stateless (https://restfulapi.net/statelessness/)



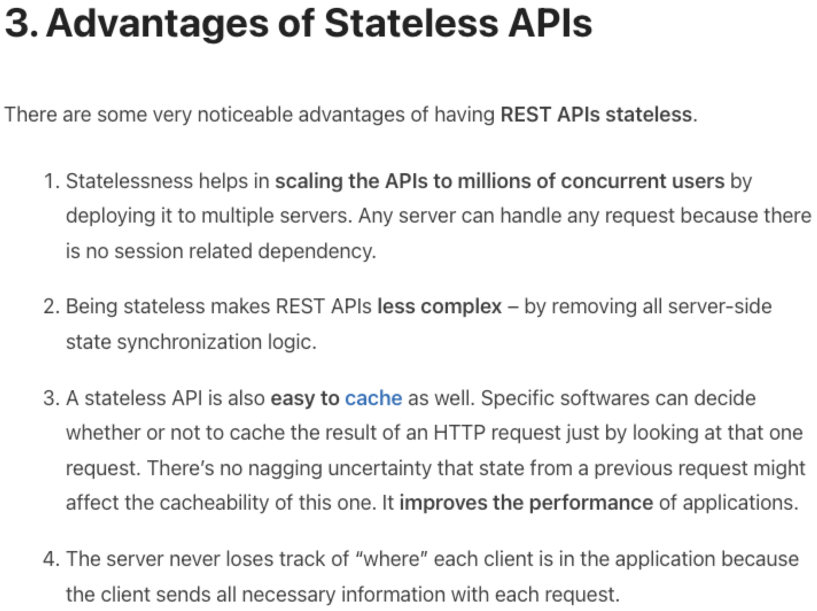
2. Application State vs Resource State

It is important to understand the between the application state and the resource state. Both are completely different things.

Application state is server-side data that servers store to identify incoming client requests, their previous interaction details, and current context information.

Resource state is the current state of a resource on a server at any point in time – and it has nothing to do with the interaction between client and server. It is what we get as a response from the server as the API response. We refer to it as resource representation.

> REST statelessness means being free from the application state.



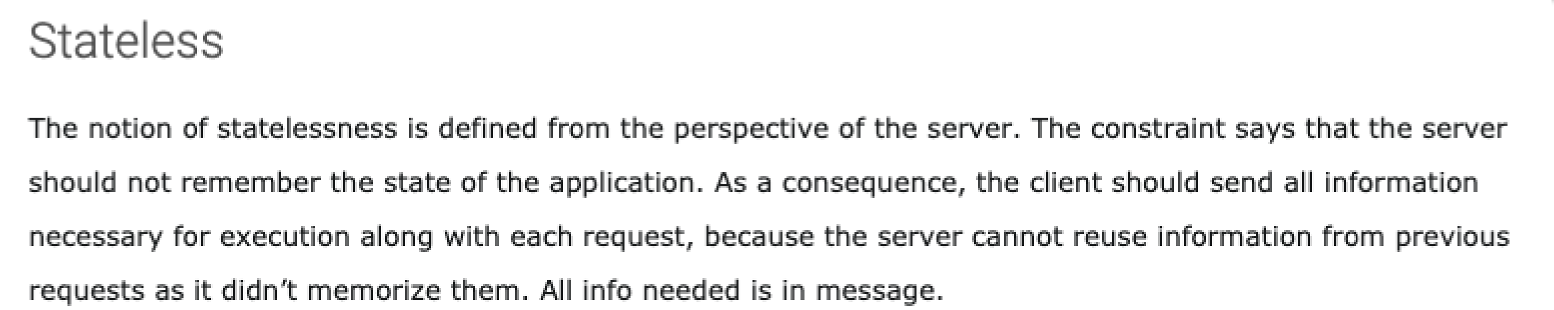
3. Advantages of Stateless APIs

There are some very noticeable advantages of having **REST APIs stateless**.

1. Statelessness helps in **scaling the APIs to millions of concurrent users** by deploying it to multiple servers. Any server can handle any request because there is no session related dependency.
2. Being stateless makes REST APIs **less complex** – by removing all server-side state synchronization logic.
3. A stateless API is also **easy to** cache as well. Specific softwares can decide whether or not to cache the result of an HTTP request just by looking at that one request. There's no nagging uncertainty that state from a previous request might affect the cacheability of this one. It **improves the performance** of applications.
4. The server never loses track of "where" each client is in the application because the client sends all necessary information with each request.

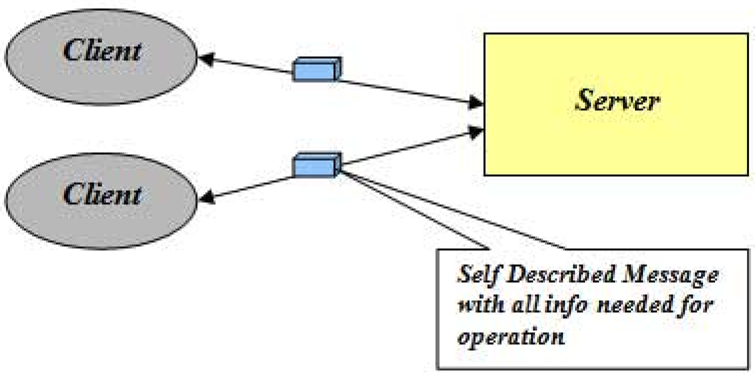
Stateless

http://mrbool.com/rest-architectural-elements-and-



Stateless

The notion of statelessness is defined from the perspective of the server. The constraint says that the server should not remember the state of the application. As a consequence, the client should send all information necessary for execution along with each request, because the server cannot reuse information from previous requests as it didn't memorize them. All info needed is in message.



Many platforms provided server-side functions to save session/conversation/application state.

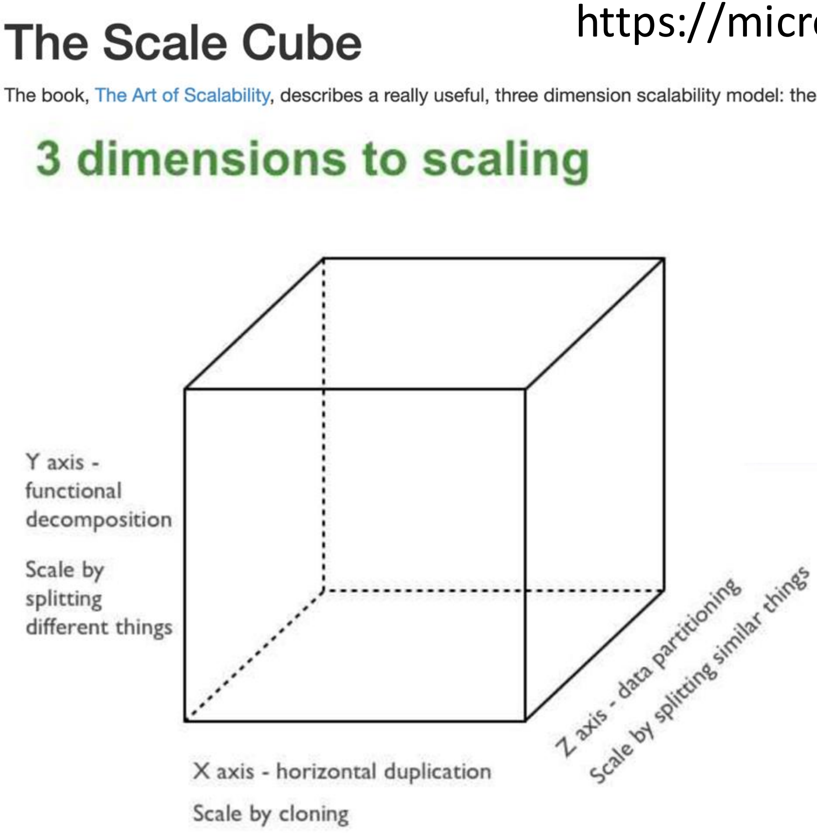
- Caching results of previous DB queries.
- Current navigation position in a complex, multi-page application.
- Transaction/Saga status
- Logging context
- etc.

REST keeps the information in various headers or cookies. The browser re-provides the information on each request.

This is another use for JWT.

Microservices Scalability Cube

https://microservices.io/articles/scalecube.html



The Scale Cube

The book, The Art of Scalability, describes a really useful, three dimension scalability model: the scale cube.

3 dimensions to scaling
Y axis - functional decomposition
Scale by splitting different things
X axis - horizontal duplication
Scale by cloning
Z axis - data partitioning
Scale by splitting similar things

- X-axis scaling consists of running multiple copies of an application behind a load balancer. If there are N copies then each copy handles 1/N of the load. This is a simple, commonly used approach of scaling an application.

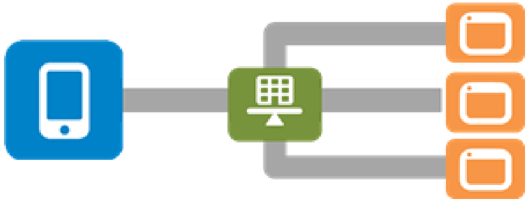
- Y-axis axis scaling splits the application into multiple, different services. Each service is responsible for one or more closely related functions.

- When using Z-axis scaling each server runs an identical copy of the code. In this respect, it’s similar to X-axis scaling. The big difference is that each server is responsible for only a subset of the data. Some component of the system is responsible for routing each request to the appropriate server

XYZ – Scaling

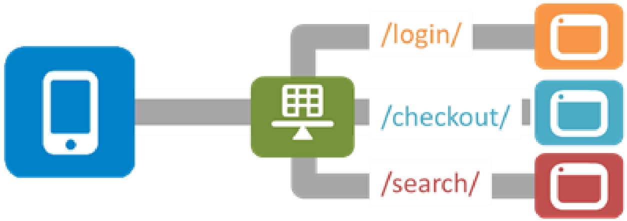
X-AXIS SCALING

Network name: Horizontal scaling, scale out



Y-AXIS SCALING

Network name: Layer 7 Load Balancing, Content switching, HTTP Message Steering



/login/

Service A

/checkout/

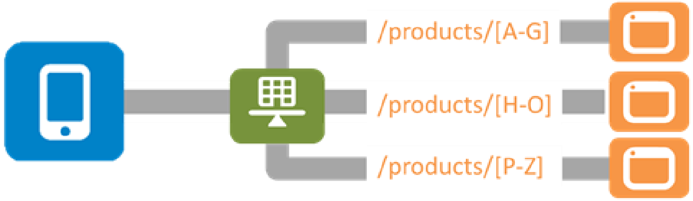
Service B

/search/

Service C

Z-AXIS SCALING

Network name: Layer 7 Load Balancing, Content switching, HTTP Message Steering



/products/[A-G]

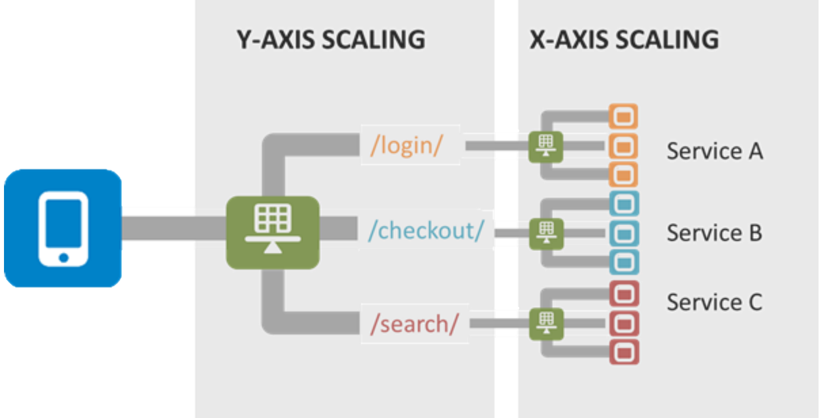
Products A-G

/products/[H-O]

Products H-O

/products/[P-Z]

Products P-Z



Y-AXIS SCALING

X-AXIS SCALING

/login/

Service A

/checkout/

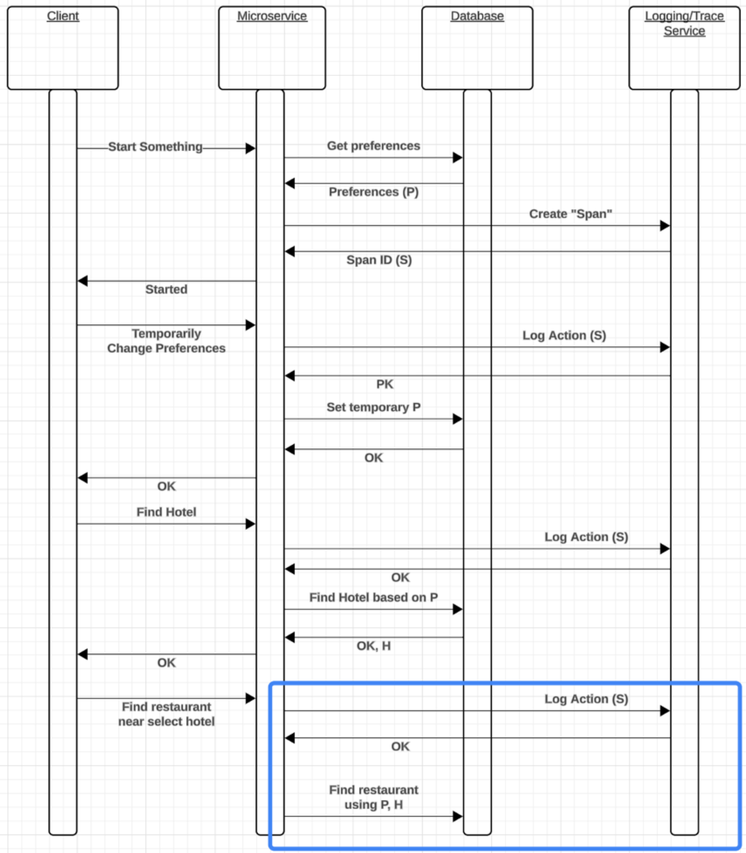
Service B

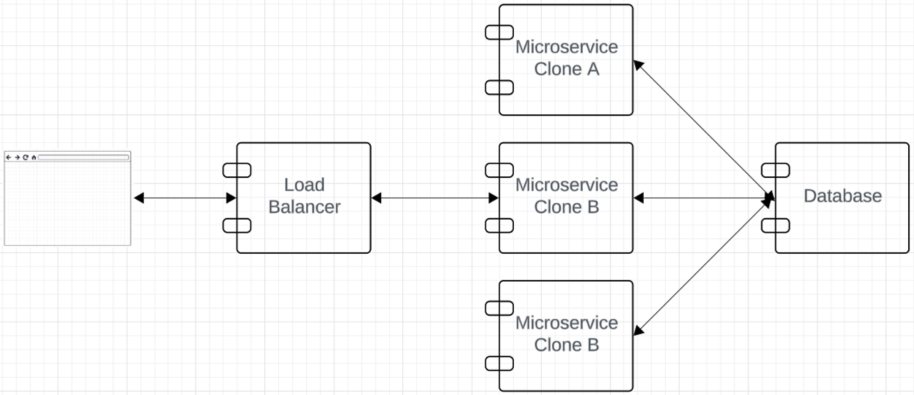
/search/

Service C

- There are three dimensions.
- A complete solution/environment typical is a mix and composition of patterns.
- Application design and data access determines opitions.

Conversation/Session State

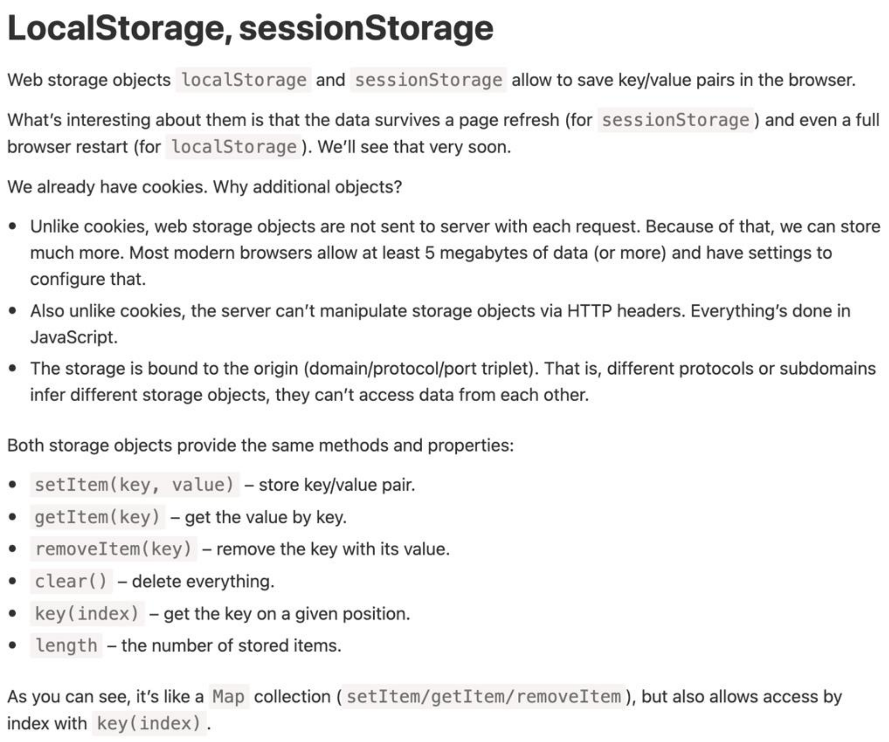




- The client/user and microservice need to agree on S, P and H
- Where is it stored?
  - I cannot keep in in the application "memory" because of routing and high availability.
  - I could associate S, P and H with the user ID or IP address, and then read/write from a database on each operation, but this is expensive.
  - I can "give it to the client" and make it "give it back to me" on every request.

If I have lots and lots of active clients, the easiest way to scale is to keep the state in the "client."

Browser Storage



LocalStorage, sessionStorage

Web storage objects `localStorage` and `sessionStorage` allow to save key/value pairs in the browser.

What's interesting about them is that the data survives a page refresh (for `sessionStorage`) and even a full browser restart (for `localStorage`). We'll see that very soon.

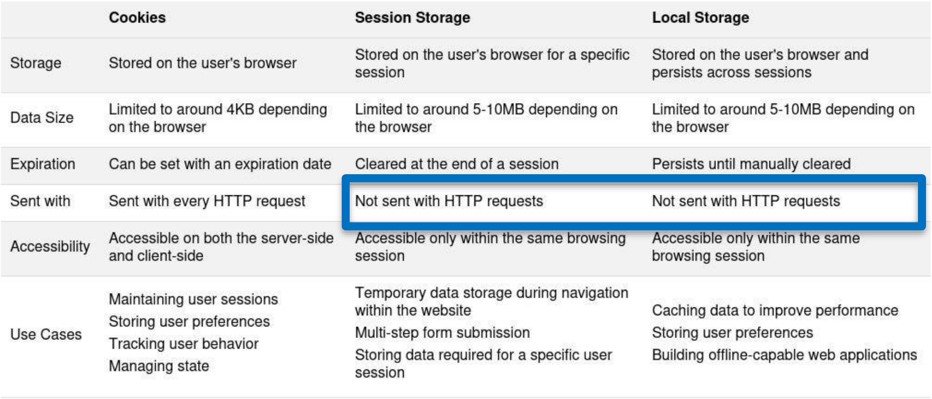
We already have cookies. Why additional objects?

- Unlike cookies, web storage objects are not sent to server with each request. Because of that, we can store much more. Most modern browsers allow at least 5 megabytes of data (or more) and have settings to configure that.
- Also unlike cookies, the server can't manipulate storage objects via HTTP headers. Everything's done in JavaScript.
- The storage is bound to the origin (domain/protocol/port triplet). That is, different protocols or subdomains infer different storage objects, they can't access data from each other.

Both storage objects provide the same methods and properties:

- `setItem(key, value)` – store key/value pair.
- `getItem(key)` – get the value by key.
- `removeItem(key)` – remove the key with its value.
- `clear()` – delete everything.
- `key(index)` – get the key on a given position.
- `length` – the number of stored items.

As you can see, it's like a `Map` collection (`setItem/getItem/removeItem`), but also allows access by index with `key(index)`.



	Cookies	Session Storage	Local Storage
Storage	Stored on the user's browser	Stored on the user's browser for a specific session	Stored on the user's browser and persists across sessions
Data Size	Limited to around 4KB depending on the browser	Limited to around 5-10MB depending on the browser	Limited to around 5-10MB depending on the browser
Expiration	Can be set with an expiration date	Cleared at the end of a session	Persists until manually cleared
Sent with	Sent with every HTTP request	Not sent with HTTP requests	Not sent with HTTP requests
Accessibility	Accessible on both the server-side and client-side	Accessible only within the same browsing session	Accessible only within the same browsing session
Use Cases	Maintaining user sessions Storing user preferences Tracking user behavior Managing state	Temporary data storage during navigation within the website Multi-step form submission Storing data required for a specific user session	Caching data to improve performance Storing user preferences Building offline-capable web applications

Not *automatically* sent with *all* HTTP requests, but … …

- The client application TypeScript can add to requests, and update based on responses.
- The server can specify in the API definition that there is a "token" that is sent to the API and then back to the client.
- The token can be "opaque" and "signed" for integrity by the server.
- This is the same mechanism/process for the *Access Token.*

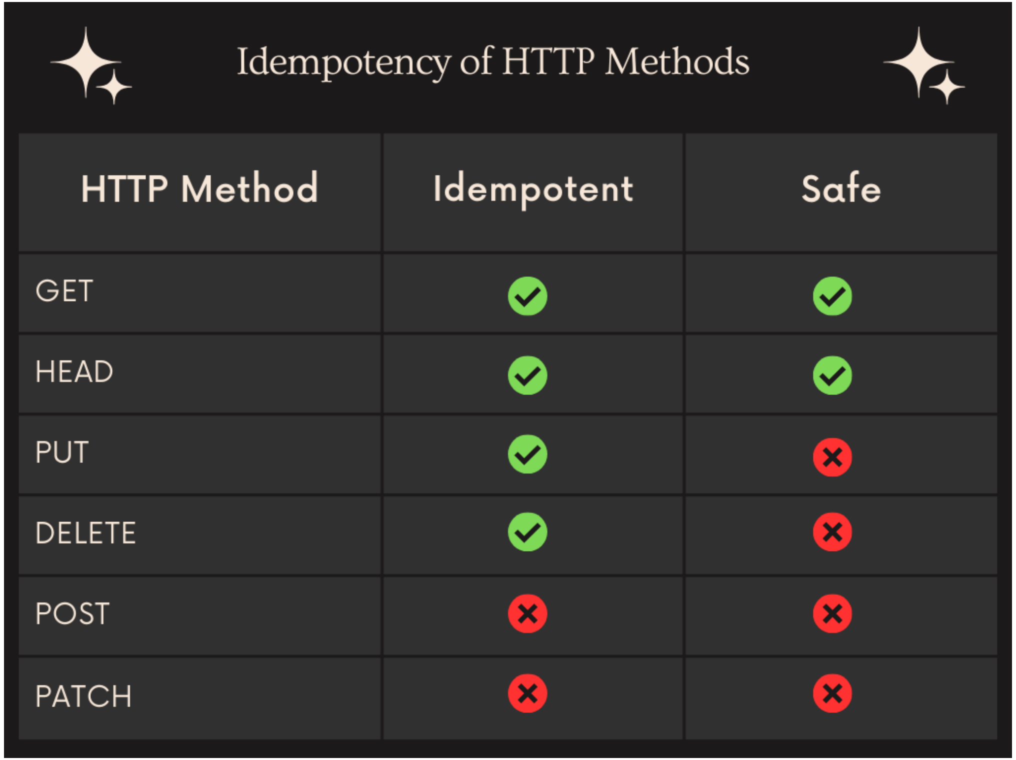
Idempotent

Idempotent

- From Wikpedia (https://en.wikipedia.org/wiki/Idempotence)
  - "Idempotence is the property of certain operations in mathematics and computer science whereby they can be applied multiple times without changing the result beyond the initial application."
  - "In the Hypertext Transfer Protocol (HTTP), idempotence and safety are the major attributes that separate HTTP methods. Of the major HTTP methods, GET, PUT, and DELETE should be implemented in an idempotent manner according to the standard, but POST doesn't need to be. GET retrieves the state of a resource; PUT updates the state of a resource; and DELETE deletes a resource. As in the example above, reading data usually has no side effects, so it is idempotent (in fact nullipotent). Updating and deleting a given data are each usually idempotent as long as the request uniquely identifies the resource and only that resource again in the future."
- Why does this matter? Ultimately, if I do a PUT and do not get an answer, my code has no way of distinguishing between:
  - The request got "lost" and was not processed.
  - The request was processed and the response got lost.
  - But more fundamentally, the network is flaky and loses messages, loses responses, retransmit messages, … …

Idempotent

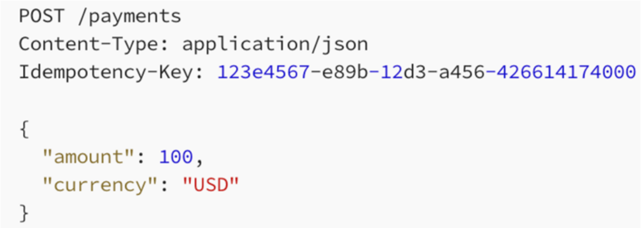
https://restfulapi.net/idempotent-rest-apis/



Idempotency of HTTP Methods

HTTP Method	Idempotent	Safe
GET	✓	✓
HEAD	✓	✓
PUT	✓	✗
DELETE	✓	✗
POST	✗	✗
PATCH	✗	✗

- There are several techniques for ensuring correct implementation of idempotent processing.
- Examples:
  - Nonce (Idempotent Key)
  - Condition processing based on resources current state.
- Idempotent Key



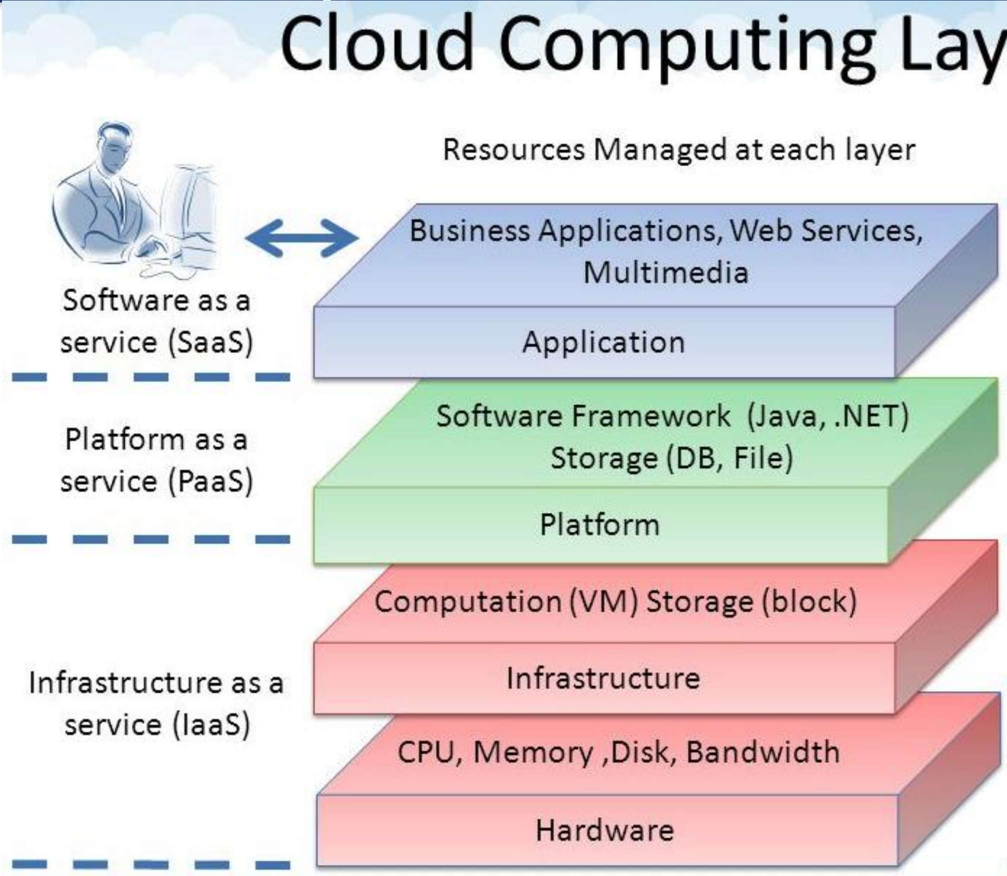
```
POST /payments
Content-Type: application/json
Idempotency-Key: 123e4567-e89b-12d3-a456-426614174000

{
  "amount": 100,
  "currency": "USD"
}
```

  - Client generates the UUID.
  - If the server has not processed the key, process and “remember” the response.
  - If the server has processed the key, return the previous answer.

Platform-as-a-Service
Function-as-a-Service

Reminder
Context

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).

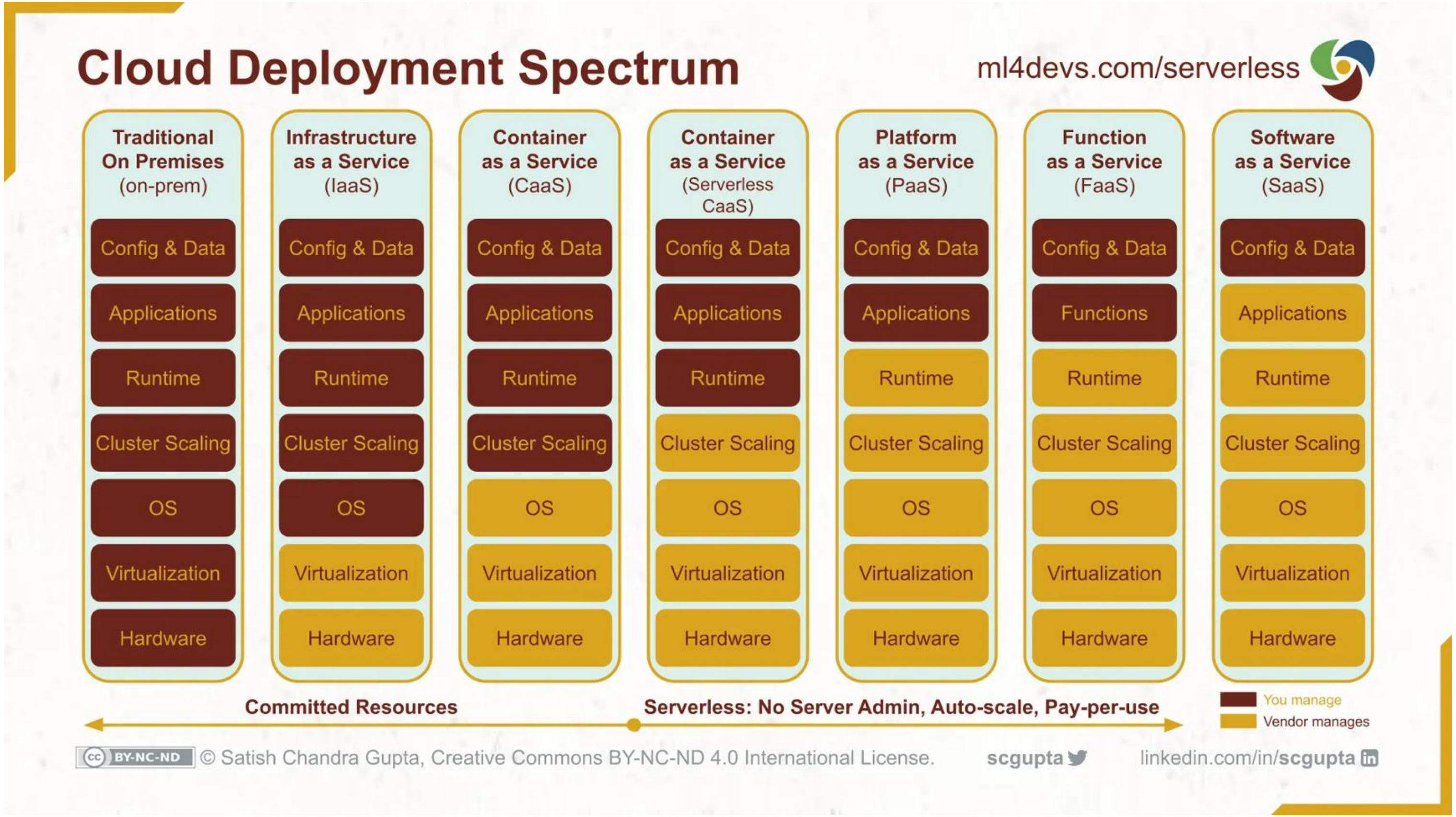
- Google Apps
- Salesforce, Zendesk, ServiceNow, Concur, … …
- Office 365, Trello
- … …

- 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

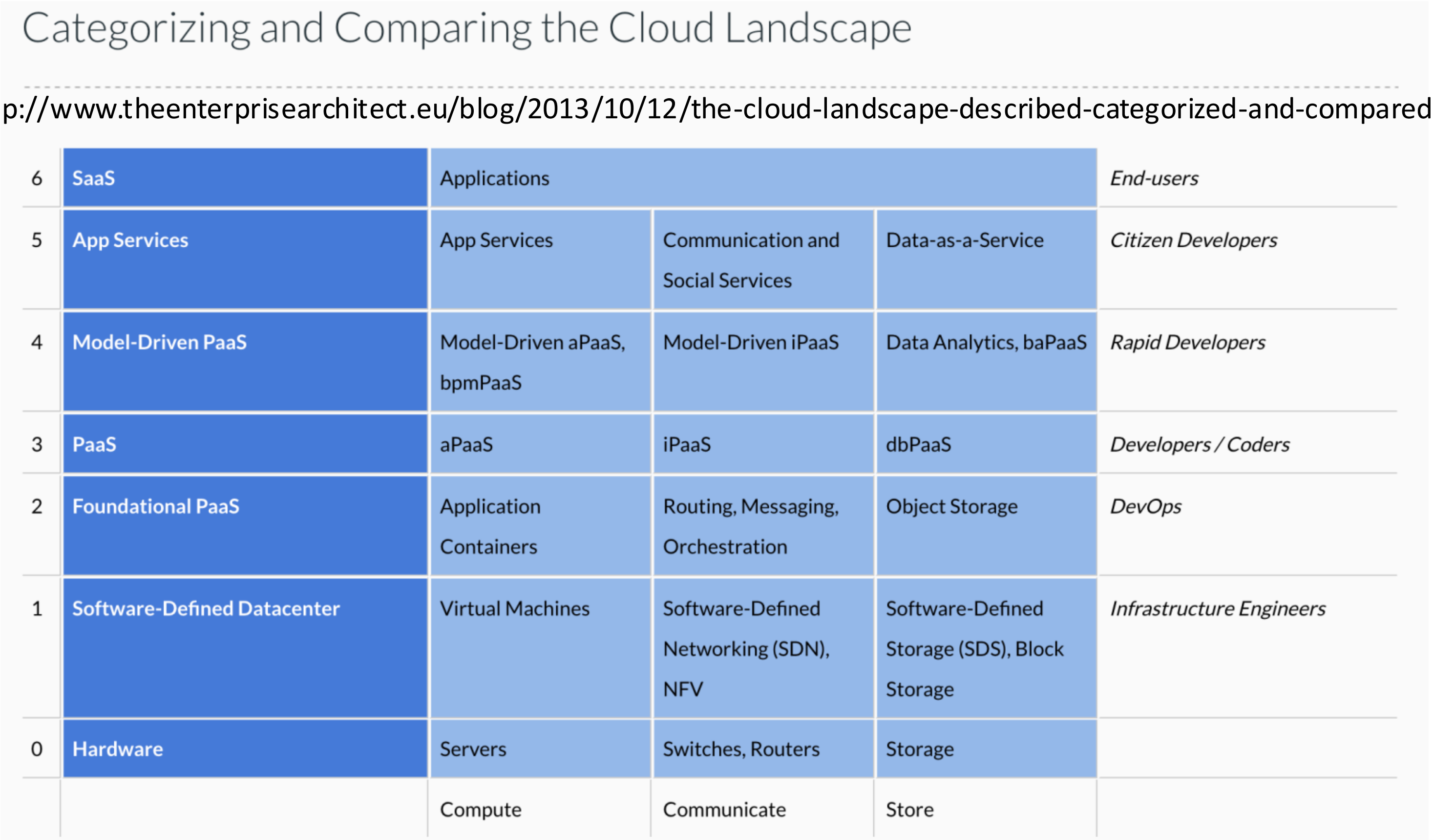
Traditional On Premises (on-prem)	Infrastructure as a Service (IaaS)	Container as a Service (CaaS)	Container as a Service (Serverless CaaS)	Platform as a Service (PaaS)	Function as a Service (FaaS)	Software as a Service (SaaS)
🟫 Config & Data	🟫 Config & Data	🟫 Config & Data	🟫 Config & Data	🟫 Config & 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

scgupta linkedin.com/in/scgupta

Cloud Concepts – One Perspective



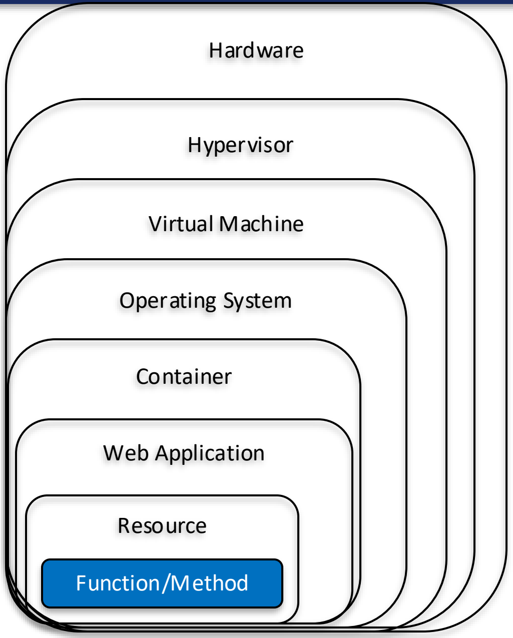
Categorizing and Comparing the Cloud Landscape

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	

Computer Models

- At the very core, you write a function or method.
- A set of functions or methods become a resource.
- A resource runs in a web application and uses required software, e. g. packages, libraries.
- A web application run in some kind of container, which provides isolation, ensures pre-reqs, etc.
- Containers run in an operating system.
- An operating systems runs on HW, which may be a virtual machine and virtual resources.
- Virtual resources run on hypervisors, which
- Run on hardware.
- This is like a *matryoshka doll*. Ultimately, I decide:
  - Which level of nesting is the package I am willing to/want to produce and deploy.
  - I provide that as a package/bundle and provide a declaration/manifest of what I want created and managed automatically for me.
- This is over simplified and the boundaries blur.



Platform-as-a-Service

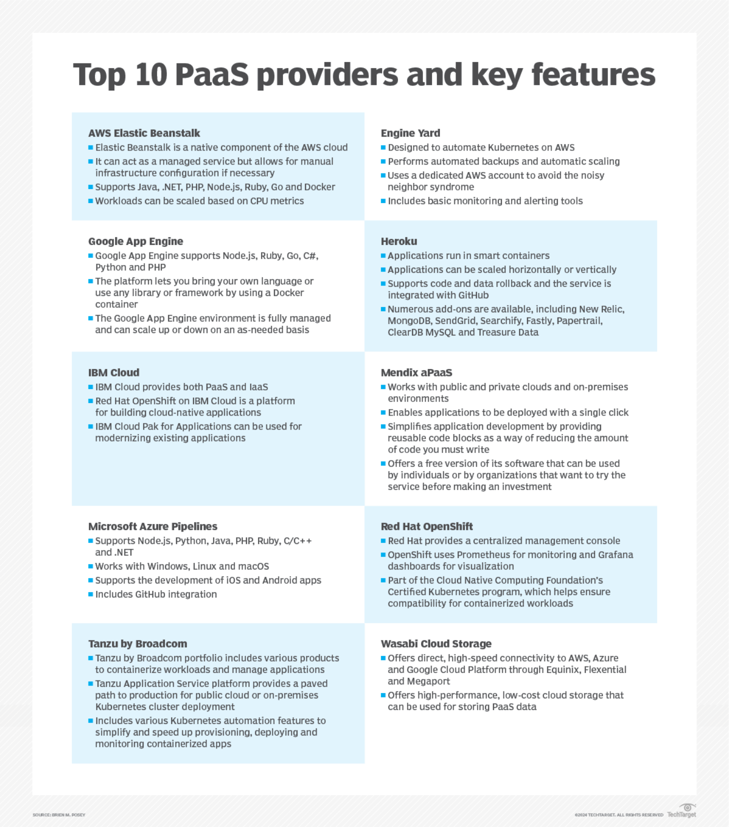
PaaS

The Top 10 Platform as a Service (PaaS) Solutions include:

https://expertinsights.com/insights/the-top-platform-as-a-service-paas-solutions/

1. AWS Elastic Beanstalk
2. Google Cloud Run
3. Heroku by Salesforce
4. IBM Cloud Code Engine
5. Microsoft Azure App Service
6. Oracle Cloud Infrastructure
7. Red Hat OpenShift
8. Salesforce Einstein 1 Platform
9. SAP Business Technology Platform (BTP)
10. VMware Tanzu

- The concept is common and widely used.
- I have no idea if these lists are correct.
- There are subtle differences in the terms.



Top 10 PaaS providers and key features

AWS Elastic Beanstalk

- Elastic Beanstalk is a native component of the AWS cloud
- It can act as a managed service but allows for manual infrastructure configuration if necessary
- Supports Java, .NET, PHP, Node.js, Ruby, Go and Docker
- Workloads can be scaled based on CPU metrics

Engine Yard

- Designed to automate Kubernetes on AWS
- Performs automated backups and automatic scaling
- Uses a dedicated AWS account to avoid the noisy neighbor syndrome
- Includes basic monitoring and alerting tools

Google App Engine

- Google App Engine supports Node.js, Ruby, Go, C#, Python and PHP
- The platform lets you bring your own language or use any library or framework by using a Docker container
- The Google App Engine environment is fully managed and can scale up or down on an as-needed basis

Heroku

- Applications run in smart containers
- Applications can be scaled horizontally or vertically
- Supports code and data rollback and the service is integrated with GitHub
- Numerous add-ons are available, including New Relic, MongoDB, SendGrid, Searchify, Fastly, Papertrail, ClearDB MySQL and Treasure Data

IBM Cloud

- IBM Cloud provides both PaaS and IaaS
- Red Hat OpenShift on IBM Cloud is a platform for building cloud-native applications
- IBM Cloud Pak for Applications can be used for modernizing existing applications

Mendix aPaaS

- Works with public and private clouds and on-premises environments
- Enables applications to be deployed with a single click
- Simplifies application development by providing reusable code blocks as a way of reducing the amount of code you must write
- Offers a free version of its software that can be used by individuals or by organizations that want to try the service before making an investment

Microsoft Azure Pipelines

- Supports Node.js, Python, Java, PHP, Ruby, C/C++ and .NET
- Works with Windows, Linux and macOS
- Supports the development of iOS and Android apps
- Includes GitHub integration

Red Hat OpenShift

- Red Hat provides a centralized management console
- OpenShift uses Prometheus for monitoring and Grafana dashboards for visualization
- Part of the Cloud Native Computing Foundation's Certified Kubernetes program, which helps ensure compatibility for containerized workloads

Tanzu by Broadcom

- Tanzu by Broadcom portfolio includes various products to containerize workloads and manage applications
- Tanzu Application Service platform provides a paved path to production for public cloud or on-premises Kubernetes cluster deployment
- Includes various Kubernetes automation features to simplify and speed up provisioning, deploying and monitoring containerized apps

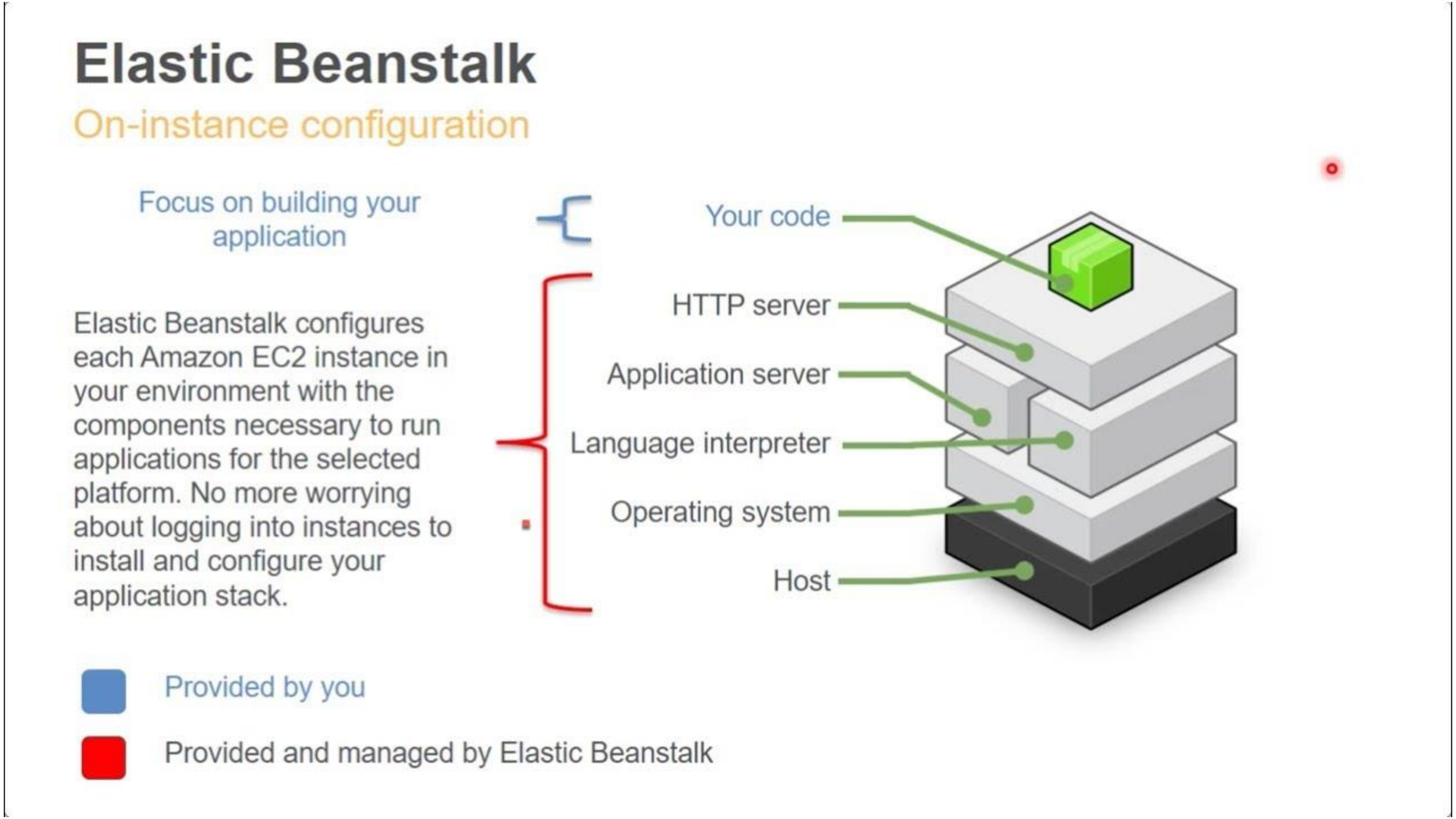
Wasabi Cloud Storage

- Offers direct, high-speed connectivity to AWS, Azure and Google Cloud Platform through Equinix, Flexential and Megaport
- Offers high-performance, low-cost cloud storage that can be used for storing PaaS data

SOURCE: BRIEN M. POSEY

https://www.techtarget.com/searchcloudcomputing/feature/Top-10-PaaS-providers-and-what-they-offer-you

Elastic Beanstalk



Elastic Beanstalk

On-instance configuration

Focus on building your application

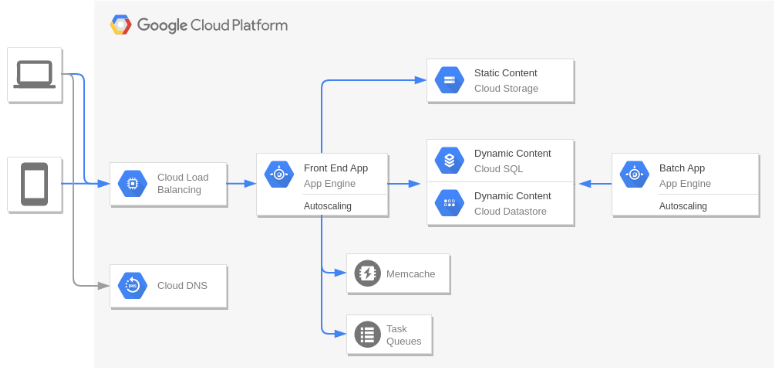
Elastic Beanstalk configures each Amazon EC2 instance in your environment with the components necessary to run applications for the selected platform. No more worrying about logging into instances to install and configure your application stack.

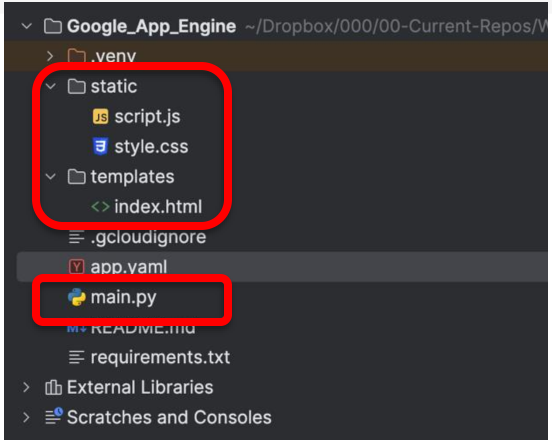
🟦 Provided by you

🟥 Provided and managed by Elastic Beanstalk

Google Cloud App Engine

Larger Application Architecture

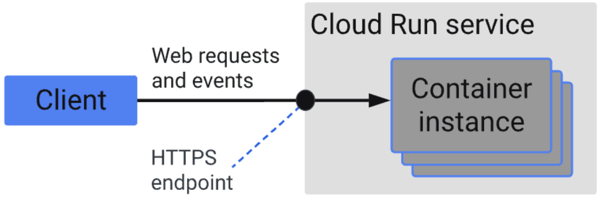


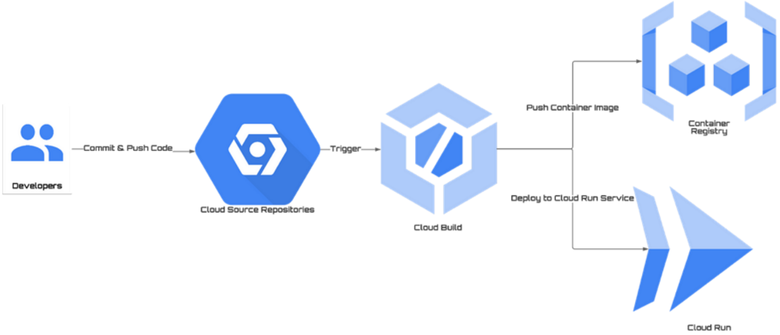


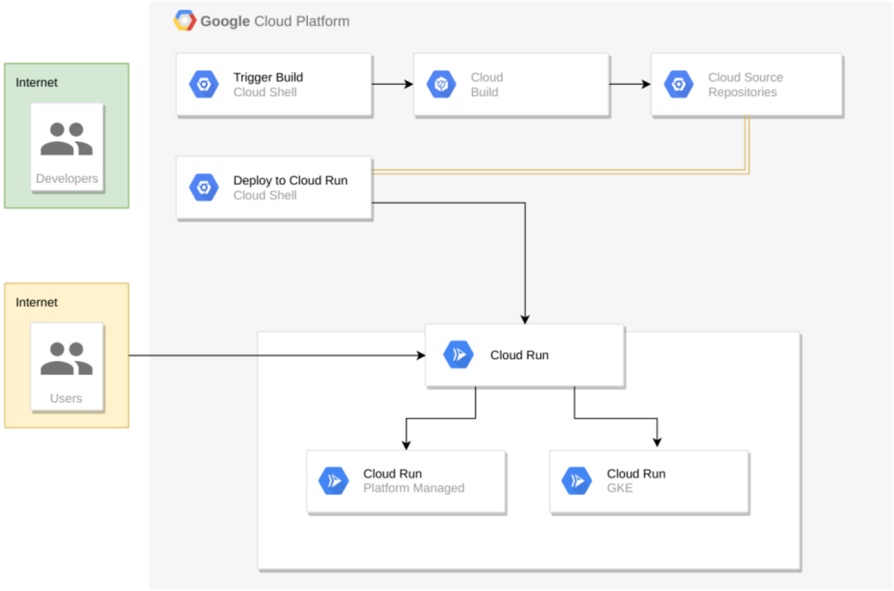
```
Google_App_Engine ~/Dropbox/000/00-Current-Repos/W
  .venv
  static
    script.js
    style.css
  templates
    index.html
  .gcloudignore
  app.yaml
  main.py
  README.md
  requirements.txt
External Libraries
Scratches and Consoles
```

- In this simple example, the application is:
  - /static/script.js, /static/style.css
  - /templates/index.html
  - main.py
- The directory/folder as a whole is the "deployment package" or the "inner doll."
- The declaration of what is needed from the encapsulating/containing dolls is in:
  - requirements
  - app.yaml
- The deployment:
  - Zips the files into a package.
  - Uploads.
  - Triggers a build and deployment workflow.
  - You can do with with the gcloud cli.

Cloud Run







- Cloud Run (https://cloud.google.com/run/docs/overview/what-is-cloud-run?hl=en) is similar but uses containers and a container registry.
- AWS Elastic Beanstalk added a similar model as containers became more common.

A Ton of Tutorials

- Google App Engine
  - https://cloud.google.com/appengine/docs/standard/python3/building-app
  - https://itnext.io/deploy-your-fastapi-app-in-serverless-google-app-engine-f85580bd4157
  - https://medium.com/analytics-vidhya/deploying-fastapi-application-in-google-app-engine-in-standard-environment-dc061d3277a
  - https://tutlinks.com/deploy-fastapi-app-on-google-cloud-platform/
- Google Cloud Run
  - https://cloud.google.com/run/docs/quickstarts/build-and-deploy/deploy-python-service?hl=en
  - https://codelabs.developers.google.com/cloud-run-starter-app#0
- Elastic Beanstalk
  - https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/GettingStarted.html
  - https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/tutorials.html

Function-as-a-Service
(Next Lecture)

Serverless and Function-as-a-Service

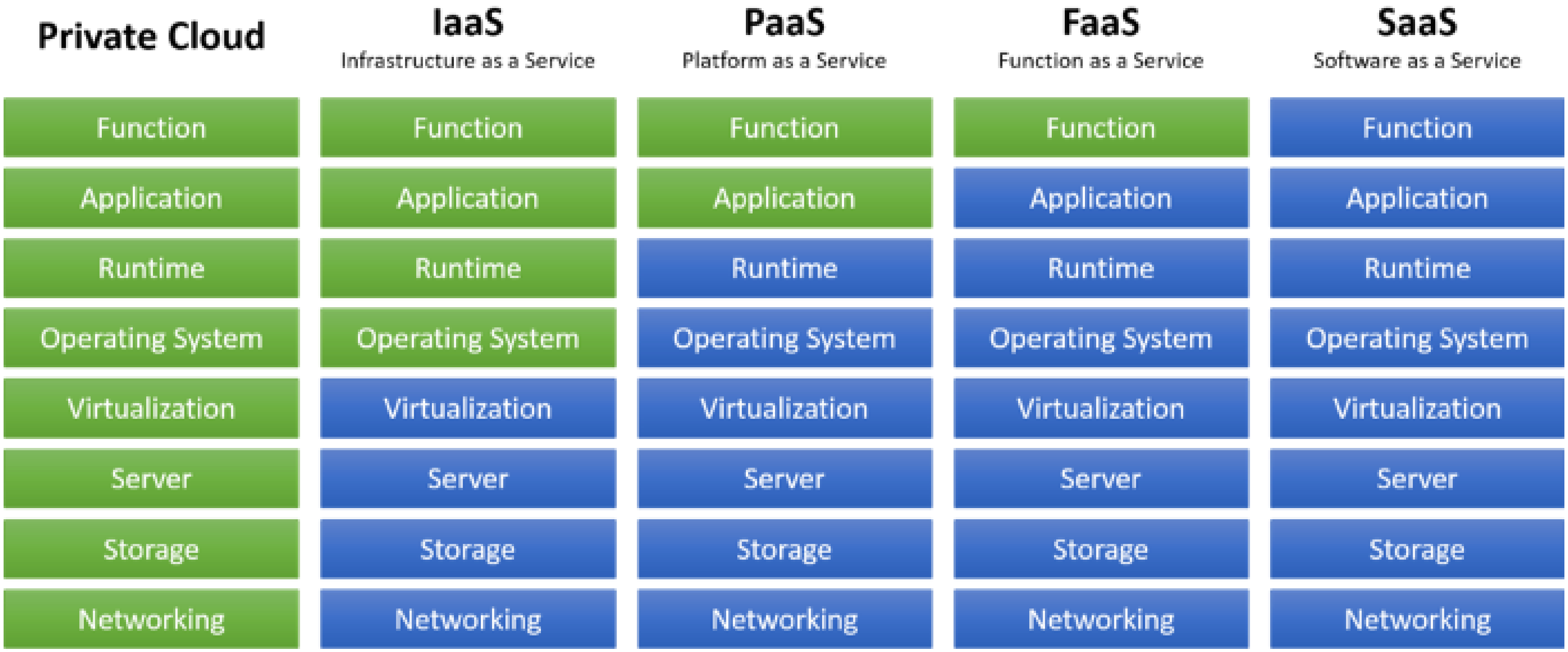
- **Serverless computing** is a cloud computing execution model in which the cloud provider runs the server, and dynamically manages the allocation of machine resources. Pricing is based on the actual amount of resources consumed by an application, rather than on pre-purchased units of capacity.[1] It can be a form of utility computing.

  Serverless computing can simplify the process of deploying code into production. Scaling, capacity planning and maintenance operations may be hidden from the developer or operator. Serverless code can be used in conjunction with code deployed in traditional styles, such as microservices. Alternatively, applications can be written to be purely serverless and use no provisioned servers at all.[2] This should not be confused with computing or networking models that do not require an actual server to function, such as peer-to-peer (P2P)."

- **Function as a service** (**FaaS**) is a category of cloud computing services that provides a platform allowing customers to develop, run, and manage application functionalities without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app.[1] Building an application following this model is one way of achieving a "serverless" architecture, and is typically used when building microservices applications."

- That was baffling:
  - IaaS, CaaS – You control the SW stack and the cloud provides (virtual) HW.
  - PaaS – The cloud hides the lower layer SW and provides an application container (e.g. Flask) with "Your code goes here." You are aware of container.
  - FaaS – The cloud provides the container, and you implement functions (corresponding to routes in REST).

Serverless and Function-as-a-Service



Private Cloud	IaaS Infrastructure as a Service	PaaS Platform as a Service	FaaS Function as a Service	SaaS Software as a Service
🟩 Function	🟩 Function	🟩 Function	🟩 Function	🟦 Function
🟩 Application	🟩 Application	🟩 Application	🟦 Application	🟦 Application
🟩 Runtime	🟩 Runtime	🟦 Runtime	🟦 Runtime	🟦 Runtime
🟩 Operating System	🟩 Operating System	🟦 Operating System	🟦 Operating System	🟦 Operating System
🟩 Virtualization	🟦 Virtualization	🟦 Virtualization	🟦 Virtualization	🟦 Virtualization
🟩 Server	🟦 Server	🟦 Server	🟦 Server	🟦 Server
🟩 Storage	🟦 Storage	🟦 Storage	🟦 Storage	🟦 Storage
🟩 Networking	🟦 Networking	🟦 Networking	🟦 Networking	🟦 Networking

Managed by the customer 🟩

Managed by the provider 🟦

https://medium.com/@tanmayct/serverless-architecture-function-as-a-service-19e127b8c990

What is Serverless Good/Not Good For … …

Serverless is **good** for

short-running
stateless
event-driven

- Microservices
- Mobile Backends
- Bots, ML Inferencing
- IoT
- Modest Stream Processing
- Service integration

Serverless is **not good** for

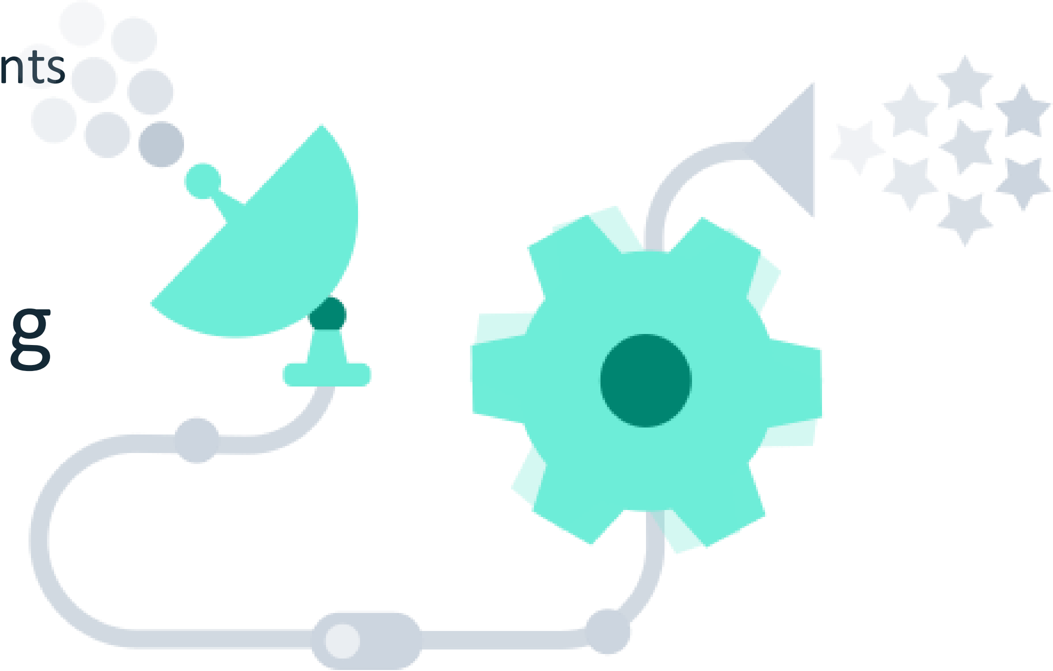
long-running
stateful
number crunching

- Databases
- Deep Learning Training
- Heavy-Duty Stream Analytics
- Numerical Simulation
- Video Streaming

What triggers code execution?

Runs code **in response** to events

Event-programming model



(Some) Current Platforms for Serverless

APACHE OpenWhisk

AWS Lambda

OpenLambda

Iron.io

Azure Functions

fission

Kubernetes

IBM Cloud Functions

Red-Hat

Google Functions

fn

Summary of Models for Deploying your Application

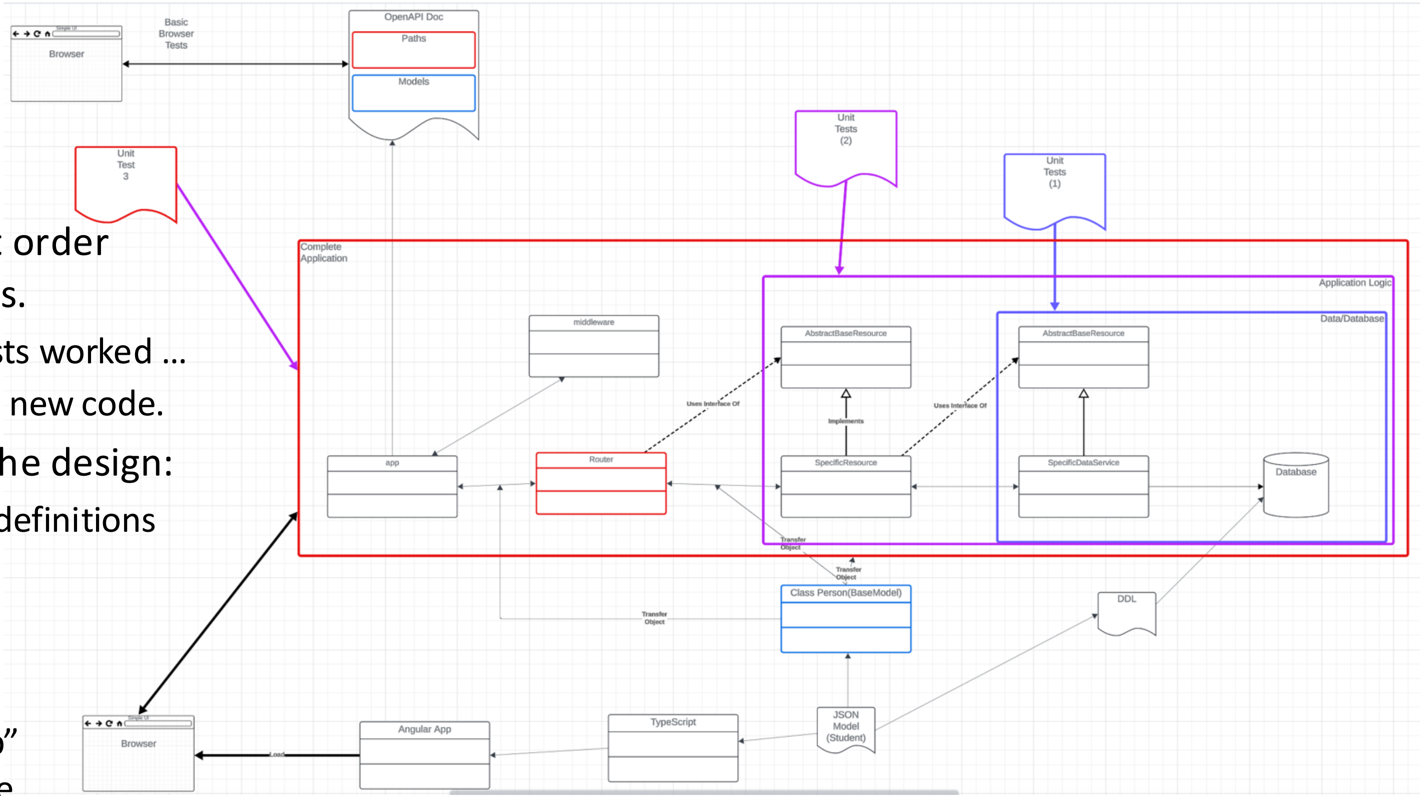
Summary (TBD)

Application Development Concepts

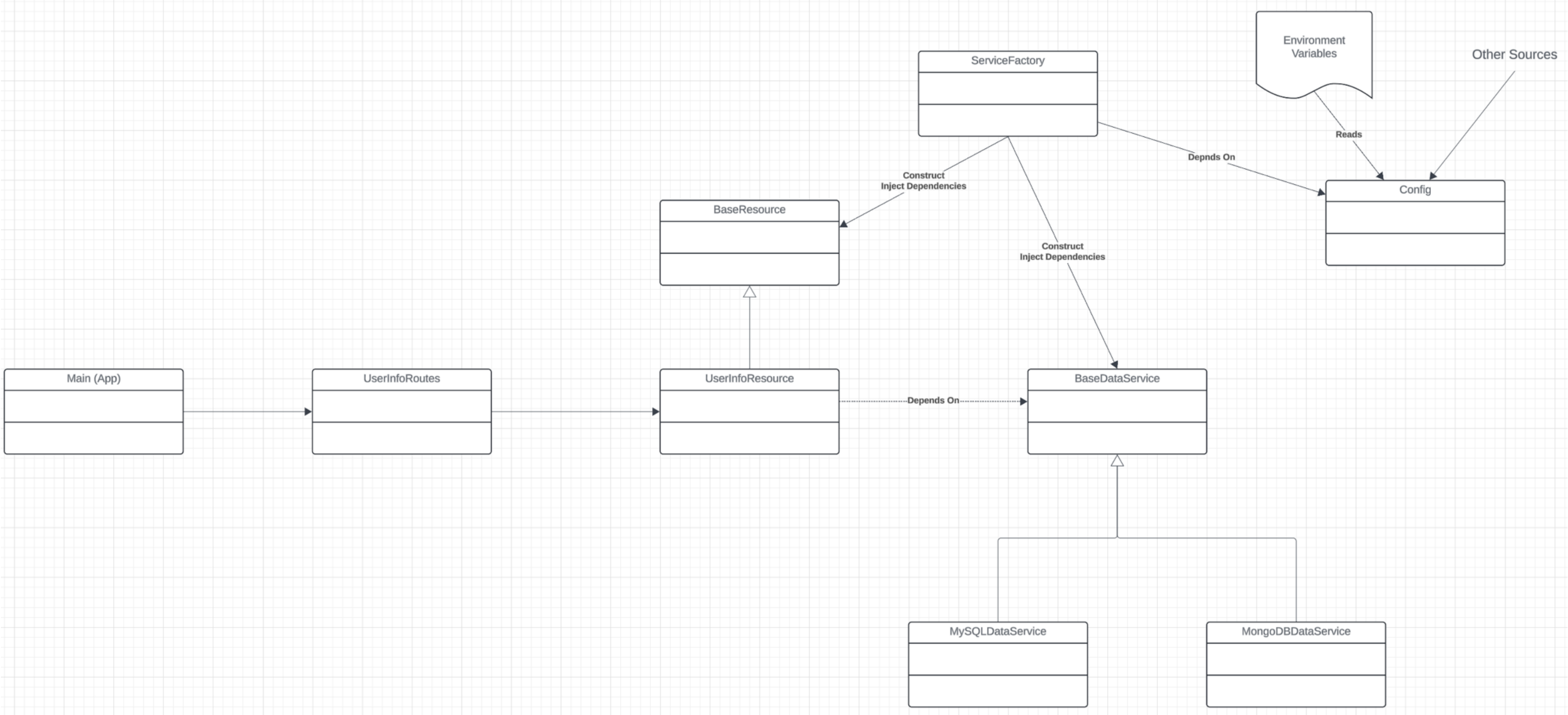
My Framework or Style

My Framework/Style

- Three elements:
  - Data layer
  - Application layer
  - Full application
- I can unit test in that order with standalone tests.
  - If the previous tests worked …
  - The error is in the new code.
- Models are core to the design:
  - Defines resource definitions
  - Define "forms"
- I build and test:
  - The UI separately
  - Can use "mock up" server to decouple.



Framework


ServiceFactory
Environment Variables
Other Sources
Reads
Depnds On
Construct
Inject Dependencies
BaseResource
Config
Construct
Inject Dependencies
Main (App)
UserInfoRoutes
UserInfoResource
Depends On
BaseDataService
MySQLDataService
MongoDBDataService

Sprint 2 – Discussion

Sprint 2 – Objectives

- For all microservices: (As a team – define and implement a project template for all Microservices)
  - OpenAPI documentation.
  - Good response codes and error codes.
  - Links for HATEOS

- At least one microservice implementing a complete REST interface and accessing a database. Complete means:
  - All methods and on all paths work/Have basic functionality for CRUD working.
    - /api/courses: GET, POST
    - /api/courses/{call_no}: GET, PUT, DELETE
  - On collection resources, e.g. /api/courses
    - Query string with parameters.
    - Pagination.
    - 201 Created with a link header for a POST. Implement GET on link header for created resource.

- Simple web UI that calls some of the services.

Sprint 2 – Objectives

- One composite microservice/resource, e.g. /api/student_infos, /api/student_infos/{uni}:
  - Calls/delegates to atomic services for GET and POST.
  - 202 Accepted and implementing an asynchronous update to a URL.
  - Implement one method synchronously and one asynchronously using threads, co-routines, …
  - Basic support for navigation paths, including query parameters, e.g. /student_infos/{uni}/courses, /student_infos/{uni}/teams
  - Middleware on each resource that implements before and after logging for all methods/paths.
  - Getting the URLs/dependencies from the environment variables.
- Deploy the composite and components and test.
- OAuth2, OpenIDC using Google login for composite service.
- Overall project structure in GitHub:
  - Linked repositories.
  - First pass at a Trello board with a backlog.