

Go Track

PowerUp! SG Tech Traineeship -
Software Engineering



Learning Objectives – Mod 18

At the end of the course, participants should be able to:

- Define Microservices.
- Define the methods of parsing JSON
- Define the methods of consuming Web Services

Microservices

- An approach of breaking down large monolith application into individual applications specialising in a specific service/functionality
- In **Monolith** architecture, all business logic resides in the same applications
 - All application services uses the same database
- In **Microservice** architecture, the services are all running in **separate processes**
 - Different databases for difference functionalities
 - **Services communicate through HTTP**, or protocol like TCP, depending on the nature of the services

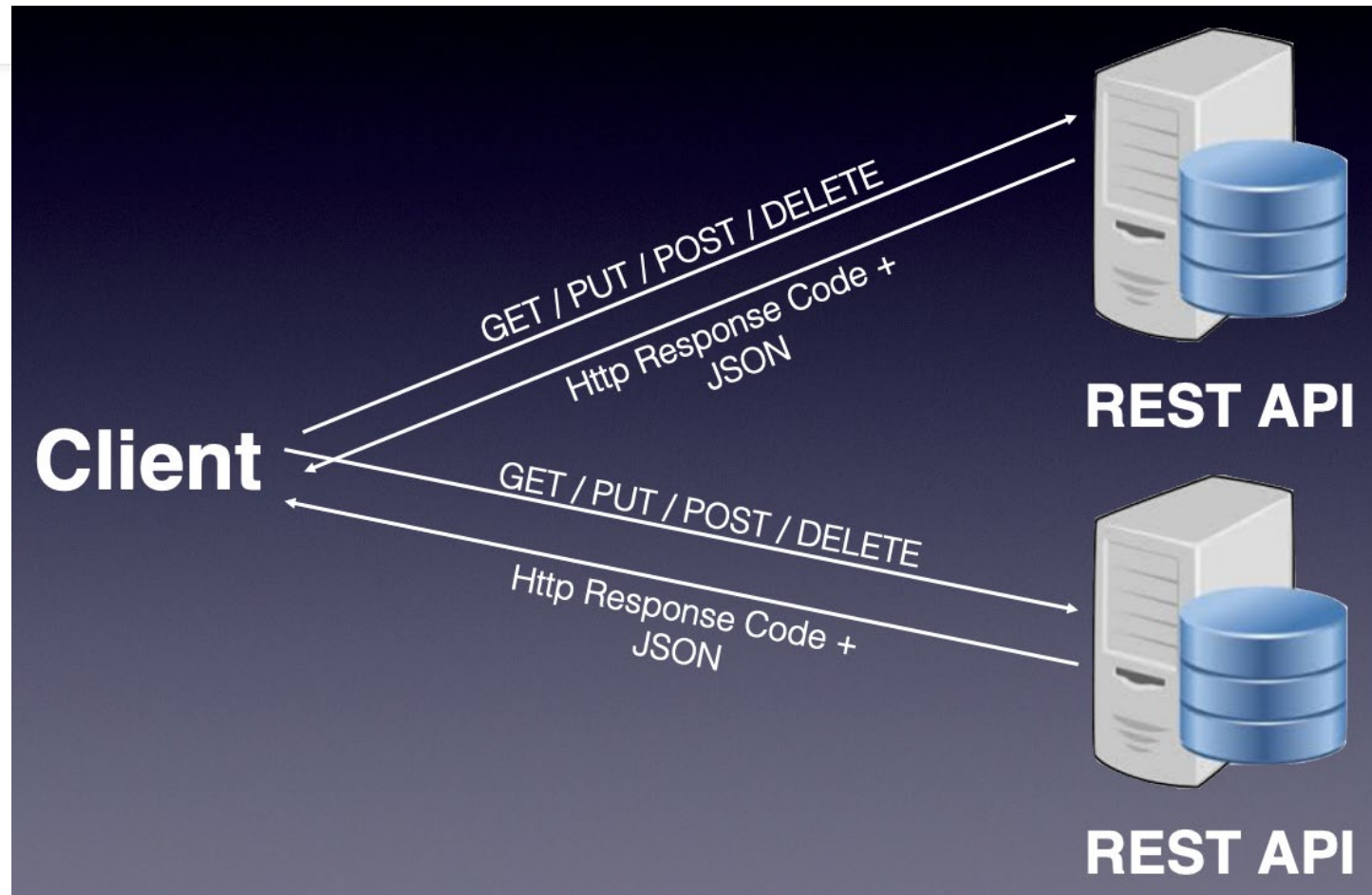
Benefits of Microservices

- Loosely coupled applications means different services can be built using different technologies
- Application scaling is easier as services that requires higher CPU usages can be scaled up accordingly
- Easier maintenance as individual services are responsible for specific functionality

Drawbacks of Microservices

- Since different databases are used for the various services, transactions involving more than one service needs to ensure database consistency
- Perfect splitting of services is difficult to achieve, and needs multiple iteration before the best separation is found
- As services need to communicate through the network, the application is inherently slower due to network latency

Microservices



Javascript Serialized Object Notation (JSON)

- JSON is a **lightweight, text-based** data format based on Javascript.
- Main idea behind is its easily readable by both human and machines.
- Originally defined by Douglas Crockford, but currently defined by RFC 7159, and ECMA-404.
- **Popularly used in REST-based web services**

Note: REST-based web services do not necessarily need to accept or return JSON data.

JSON Data Types

- JSON supports the following data types:
 - Object
 - String
 - Boolean
 - Number
 - Array
 - null

Object

- An **Object** is an **unordered** collection of **key:value** pairs enclosed in a pair of curly braces ({}).
- The following is an example of an empty object:
 - { }

String

- The **key in an object must be a String**, while the value can be either a String, Boolean, Number, Array, null, or another Object.
- The following shows an object with one key:value pair:

```
{ "firstName": "John" }
```
- An object can have multiple key:value pairs, for example:

```
{ "firstName": "John", "lastName": "Doe" }
```

 - Note the comma (,) after John and there is no comma after Doe.

String

- Each key in the Object must be unique. For example, the following example is not a valid JSON string:

```
{ "firstName": "John", "firstName": "Doe" }
```

Boolean

- A Boolean value can either be true or false:

```
{  
  "firstName": "John",  
  "lastName": "Doe",  
  "isMember": true  
}
```

Number

- A Number value can either be an integer, or a floating-point number:

```
{  
  "firstName": "John",  
  "lastName": "Doe",  
  "isMember": true,  
  "weight": 79.5,  
  "height": 1.73,  
  "children": 3  
}
```

Nested Object

- The value of a key can also be another Object, as the following example shows:

```
{  
  "firstName": "John",  
  "lastName": "Doe",  
  "isMember": true,  
  "weight": 79.5,  
  "height": 1.73,  
  "children": 3,  
  "address": {  
    "line1": "123 Street",  
    "line2": "San Francisco",  
    "state": "CA",  
    "postal": "12345"  
  }  
}
```

Array

- An Array is an **ordered sequence** of Objects:

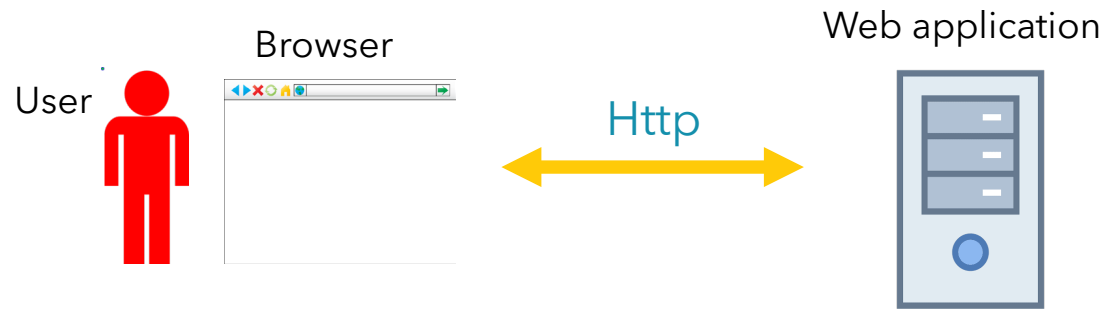
```
{  
  ...  
  "phones": [  
    {  
      "type": "work",  
      "number": "1234567"  
    },  
    {  
      "type": "home",  
      "number": "8765432"  
    },  
    {  
      "type": "mobile",  
      "number": "1234876"  
    }  
  ]  
}
```

Web Services

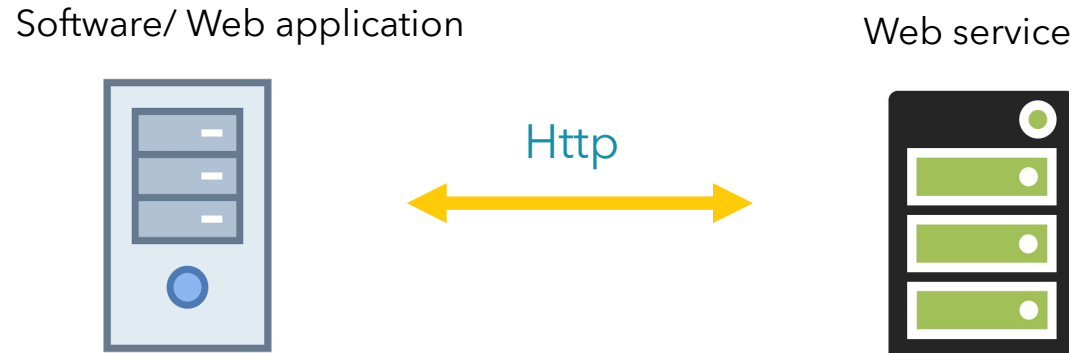
- Web services provide a service to other software programs.
- Instead of human as end user, web service has a software program as the end user.
- Web services, as it is done via Web, communicate over HTTP.

Web Application vs Web Service

Web Application – provides service directly to human as user.



Web Service – provides service to other programs.



Web Services

- There are different types of web services: SOAP-based, REST-based, XML-RPC-based.
- 2 most popular types are SOAP-based and REST-based.
- SOAP-based are used mostly in enterprise systems; REST-based used mainly in publicly available web services.

SOAP-based Web Services

- **SOAP (Simple Object Access Protocol)** is a protocol for exchanging structured data defined in **XML**.
- SOAP-based web services have been around for a while, standardized by WSC working group, very well documented, well supported by the enterprise, have large number of extensions (known as WS-*)
- SOAP-based web services are robust, explicitly described by **WSDL (Web Service Definition Language)**, and have built-in error handling.
- **UDDI (Universal Description, Discovery and Integration)**, a directory server, allows SOAP-based services to be easily discovered.

SOAP-based Web Services

- However, SOAP is highly structured and heavily defined, so known to be unnecessarily complex and cumbersome.
- SOAP XML messages used for the transportation of data can be highly complex.
- Often need other tools to manage them.
- WSDL, although providing a solid contract between client and server, can be cumbersome.
- Every change in the web service requires the WSDL, and thus the SOAP clients to be changed.
- Therefore, often result in version lock-in, as developers are wary of making changes.

REST-based Web Services

- **REST(Representational State Transfer)**-based Web services are more flexible.
- It is not an architecture, but a **design philosophy**.
- It does not require XML; very often REST-based web services use simpler data formats like JSON, leading to **speedier web services**.
- Major difference is that SOAP-based are function-driven, while REST-based are data-driven. SOAP-based tend to be RPC (Remote Procedure Call) styled, REST-based **focuses on resources, HTTP methods are the verbs** working on the resources.

So which one?

- Many developers and companies end up using both, but for different purposes.
- SOAP is often used in internal applications for enterprise integration.
- REST for external, third-party developers.
- Should learn how to tap on both strengths, with SOAP being secure and robust, while REST being simple and speedy.

Learning Objectives – Mod 19

At the end of the course, participants should be able to:

- Define the use of API.
- Examine the use of REST APIs.
- Define containerization.
- Examine and demonstrate the use of containers in containerization

REST

REST - Representational State Transfer

- a simple alternative to SOAP-based services
- relies on HTTP and uses the same HTTP verbs (GET, POST, PUT, DELETE, etc) to retrieve and send data to server
- In OOP, you create models (objects) to represent things and you define functions (methods) and attach them to models, which can be called later.
- REST is evolution of same line of thought! Instead of exposing functions as services to be called, we **expose the models(called resources)** and allow a **few actions (called verbs)** on them.

REST

- When used over HTTP:
 - URL represents a resource
 - HTTP methods are used as *verbs* to manipulate them.
- It is amazing to notice mapping between use of HTTP methods for REST with database CRUD operations.
- However the mapping is not a 1-to-1 mapping, also not the only mapping. (e.g. can use both PUT and POST to create a new resource, and either will be RESTFUL)

Communicating with Servers



- To see what goes on behind the scene, try this:
 - `$ curl -v http://www.google.com`

Examining HTTP in Action

```
Weis-Mac-mini:~ weimenglee$ curl -v google.com
* Rebuilt URL to: google.com/
* Hostname was NOT found in DNS cache
*   Trying 202.65.246.85...
* Connected to google.com (202.65.246.85) port 80 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.37.1
> Host: google.com
> Accept: */*
>
< HTTP/1.1 302 Found
< Cache-Control: private
< Content-Type: text/html; charset=UTF-8
< Location: http://www.google.com.sg/?gfe_rd=cr&ei=twKqVPgvyJbxB-0rgbAH
< Content-Length: 260
< Date: Mon, 05 Jan 2015 03:19:19 GMT
* Server GFE/2.0 is not blacklisted
< Server: GFE/2.0
< Alternate-Protocol: 80:quic,p=0.002
<
<HTML><HEAD><meta http-equiv="content-type" content="text/html; charset=utf-8">
<TITLE>302 Moved</TITLE></HEAD><BODY>
<H1>302 Moved</H1>
The document has moved
<A HREF="http://www.google.com.sg/?gfe_rd=cr&ei=twKqVPgvyJbxB-0rgbAH">here</A>
</BODY></HTML>
* Connection #0 to host google.com left intact
Weis-Mac-mini:~ weimenglee$
```

Request

HTTP Verb



Response

Header

Body

CRUD and HTTP Verbs

Mapping between the use of HTTP methods with database CRUD operations



CRUD	HTTP Verbs
Create	PUT, POST
Read	GET
Update	PUT, POST
Delete	DELETE

Communicating with REST APIs

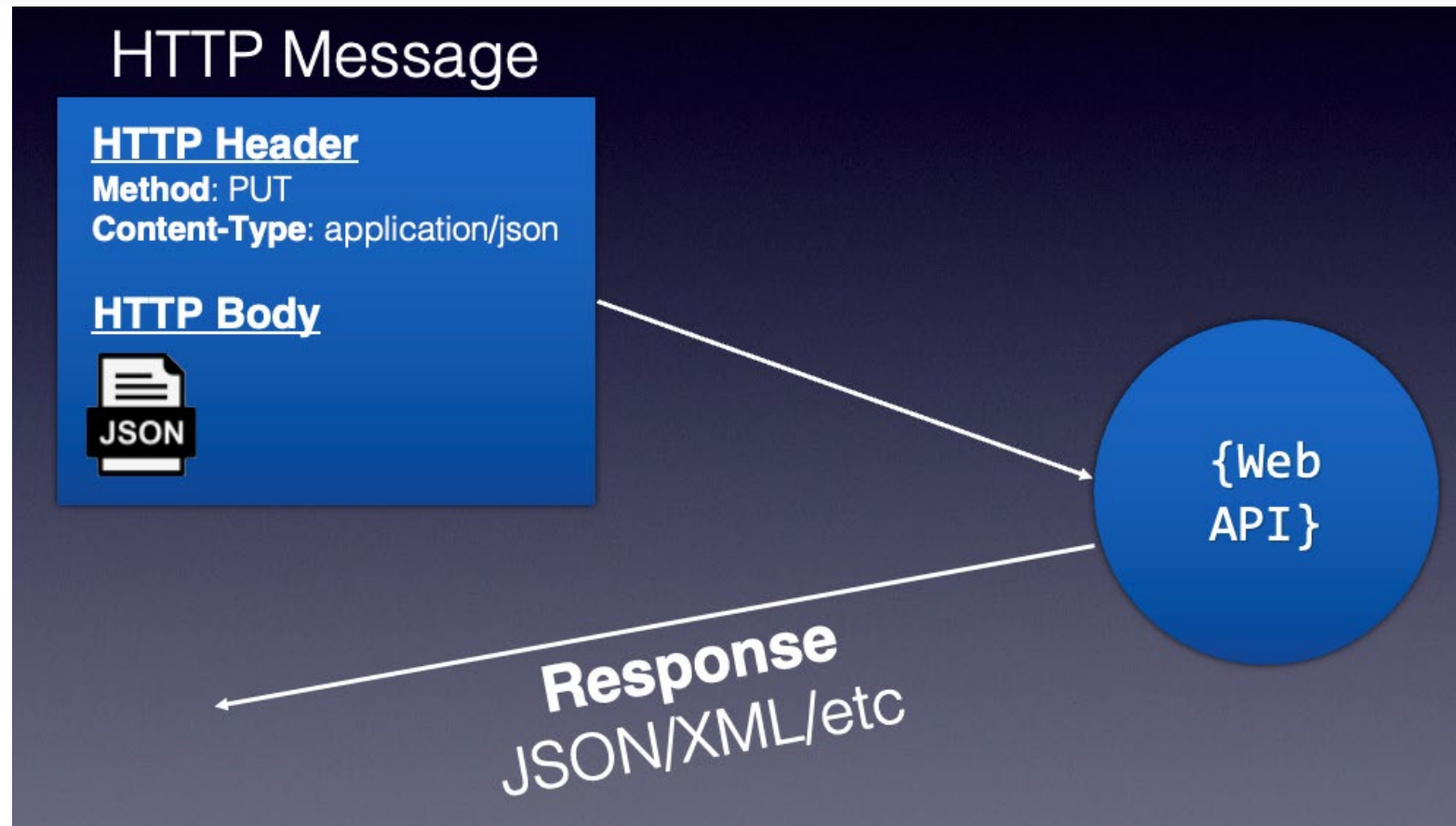


- Send requests to services using different verbs
- Get the response from services

HTTP Messages

- HTTP messages are made up of:
 - header - contains metadata, such as encoding information, HTTP methods, etc
 - header can only contain plain text
 - body - data to transmit over the network
 - body can contain data in any format; specified in the Content-Type field:
 - Content-Type: application/json

HTTP Message



REST Urls

- A REST url identifies a resource:
 - /api/v1/courses/IOS101
 - resources are best thought of as **nouns**;
- The following is not RESTful (because it describes an **action**):
 - /api/v1/courses/add

Format of a REST API URL

`http://www.yourdomain.com/api/v1/courses/IOS101`

Domain **API** **Version** **Service** **Resource**



Example REST URLs

- `http://www.yourdomain.com/api/v1/courses/`

Idempotent

- **GET** - gets all the courses

- `http://www.yourdomain.com/api/v1/courses/IOT201`

- **GET** - gets the detail of course IOT201

Idempotent

- **PUT** - updates or creates a new course

- **POST** - creates a new course

- course code is optional here; if it is not supplied, a new course code is generated

Idempotent

- **DELETE** - deletes the course IOT201

programs

Retrieve
information
of a course

<http://www.yourdomain.com/api/v1/courses/IOS101>

GET

Add or
update an
existing
course

<http://www.yourdomain.com/api/v1/courses/IOS101>



PUT

Add a new
course

<http://www.yourdomain.com/api/v1/courses/IOS101>



POST

Delete an
existing
course

<http://www.yourdomain.com/api/v1/courses/IOS101>

DELETE

{Web
API }

Idempotent Methods

- Idempotent methods achieve the same result, regardless of how many times the request is repeated
 - **GET**, **PUT**, and **DELETE** are all idempotent
 - repeating the above methods does not change the state of the server, no matter how many times the call repeated.
 - **POST** is not idempotent, everytime it is called, POST will create a resource , with a new URL

Rule of Thumb

- Use **POST** to create
- Use **PUT** to update

HTTP Response

- **200 OK**

- This response code indicates that the request was successful.

- **201 Created**

- This indicates the request was successful and a resource was created. It is used to confirm success of a PUT or POST request.

- **400 Bad Request**

- The request was malformed. This happens especially with POST and PUT requests, when the data does not pass validation, or is in the wrong format.

- **404 Not Found**

- This response indicates that the required resource could not be found. This is generally returned to all requests which point to a URL with no corresponding resource.

HTTP Response

- **401 Unauthorized**

- This error indicates that you need to perform authentication before accessing the resource.

- **405 Method Not Allowed**

- The HTTP method used is not supported for this resource.

- **409 Conflict**

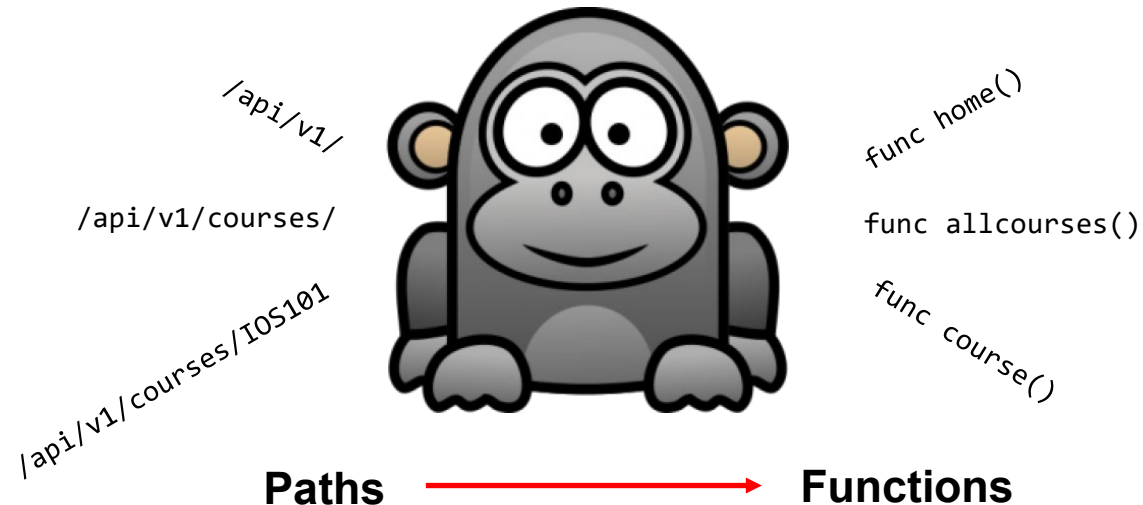
- This indicates a conflict. For instance, you are using a POST request to create the same resource twice.

- **500 Internal Server Error**

- When all else fails; generally, a 500 response is used when processing fails due to unanticipated circumstances on the server side, which causes the server to error out.

Implementing a REST API in Go

- The gorilla/mux package



gorilla/mux maps paths to functions

Mapping Path Variables

Path variable

```
router.HandleFunc("/api/v1/courses/{courseid}", course)
```

Returns a map of path variables

```
func course(w http.ResponseWriter, r *http.Request) {  
    params := mux.Vars(r)  
    fmt.Fprintf(w, "Detail for course " +  
                params["courseid"])  
}
```

Path variable

Query String

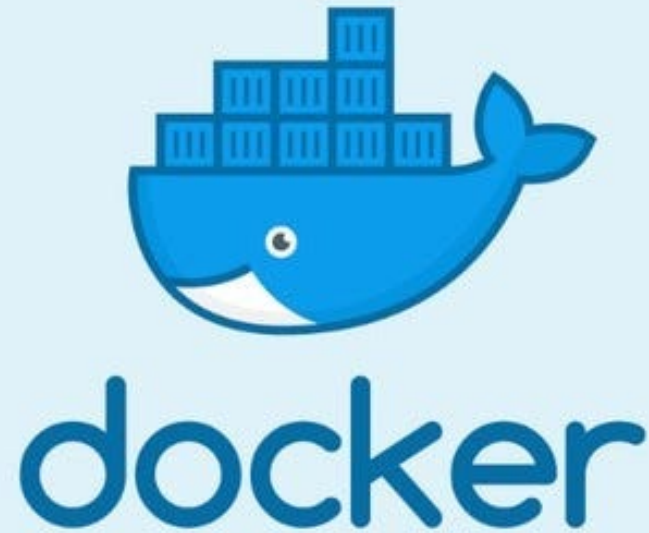
Query String

`/api/v1/courses?country=US&state=CA`

Key Value *Key Value*

A diagram illustrating the structure of a query string. The text "/api/v1/courses?country=US&state=CA" is shown. The portion after the question mark is highlighted in blue. Within this blue area, "country=US" and "state=CA" are each underlined. Below "country=US", the words "Key" and "Value" are written, with arrows pointing from "country" to "Key" and from "US" to "Value". Similarly, below "state=CA", the words "Key" and "Value" are written, with arrows pointing from "state" to "Key" and from "CA" to "Value".

Containerization



What is Docker

Docker is a software platform that simplifies the process of building, running, managing and distributing applications. It does this by virtualizing the operating system of the computer on which it is installed and running.

Docker

- It is released as an open source project by PaaS company dotCloud in 2013.
- Widely adopted by several companies.
- AWS has the EC2 Container Service, Google provides the Google Container engine, many even IBM has thrown in support for Docker.
- Docker is an **open platform** for **building, shipping and running applications on containers**.
- Containers not new, they are around for long time. Docker in fact is based on Unix/Linux containers.
- Docker essentially software for **managing containers**, making it easier for developers to use them.

Why need Docker?

- Say you have 3 different Python applications that you need to host on a single server
- Each of these apps uses a specific version of Python, as well as libraries and dependencies
- Installing different versions of Python in one single machine is problematic

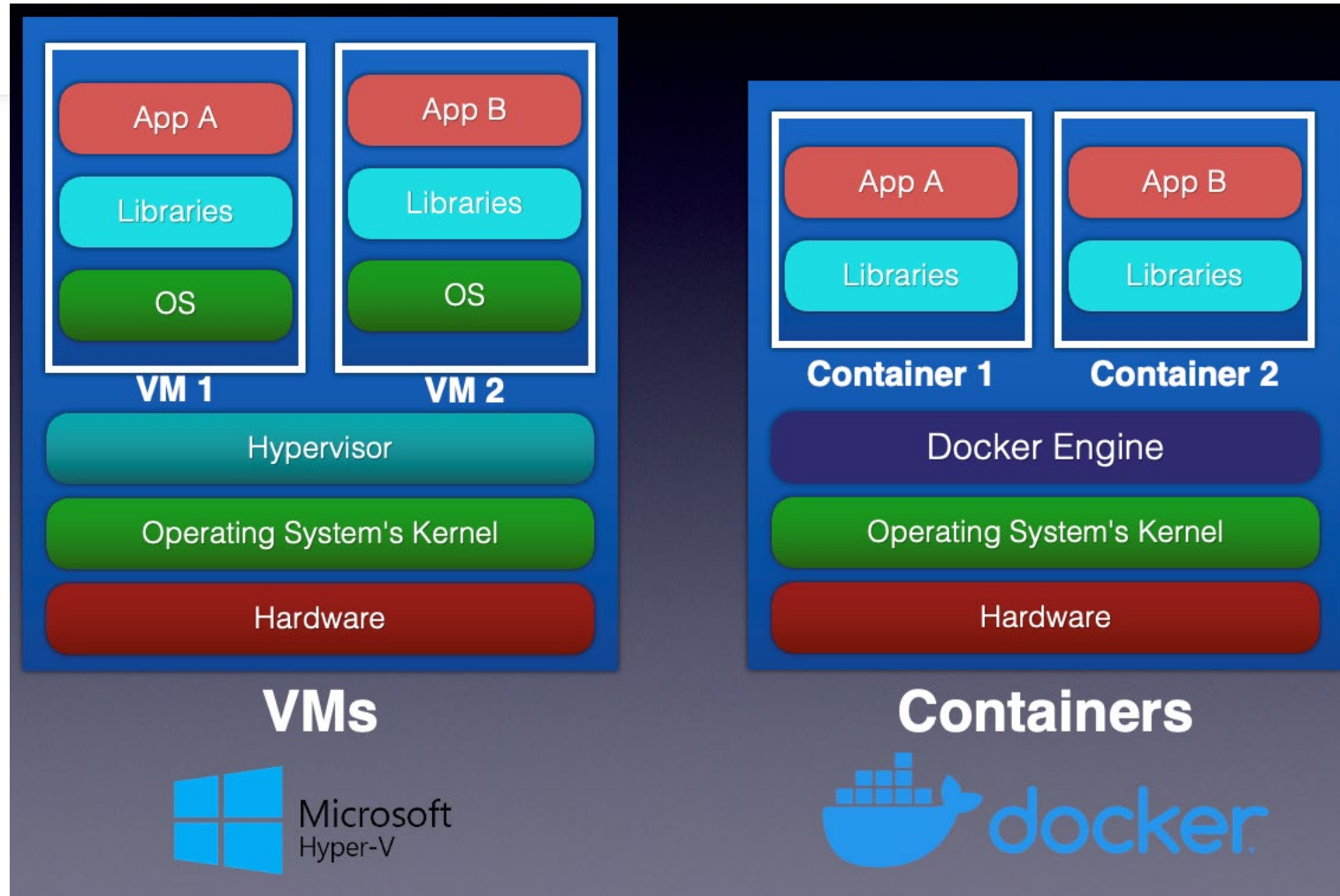
Solution to the Problems

- To solve this problem, you can:
 - Use 3 different physical machines, or
 - Use 3 different virtual machines (VM), running on 1 physical machine
- But neither solution is cheap
 - You need to either have 3 physical machines, or 1 machine that is powerful enough to run 3 VMs

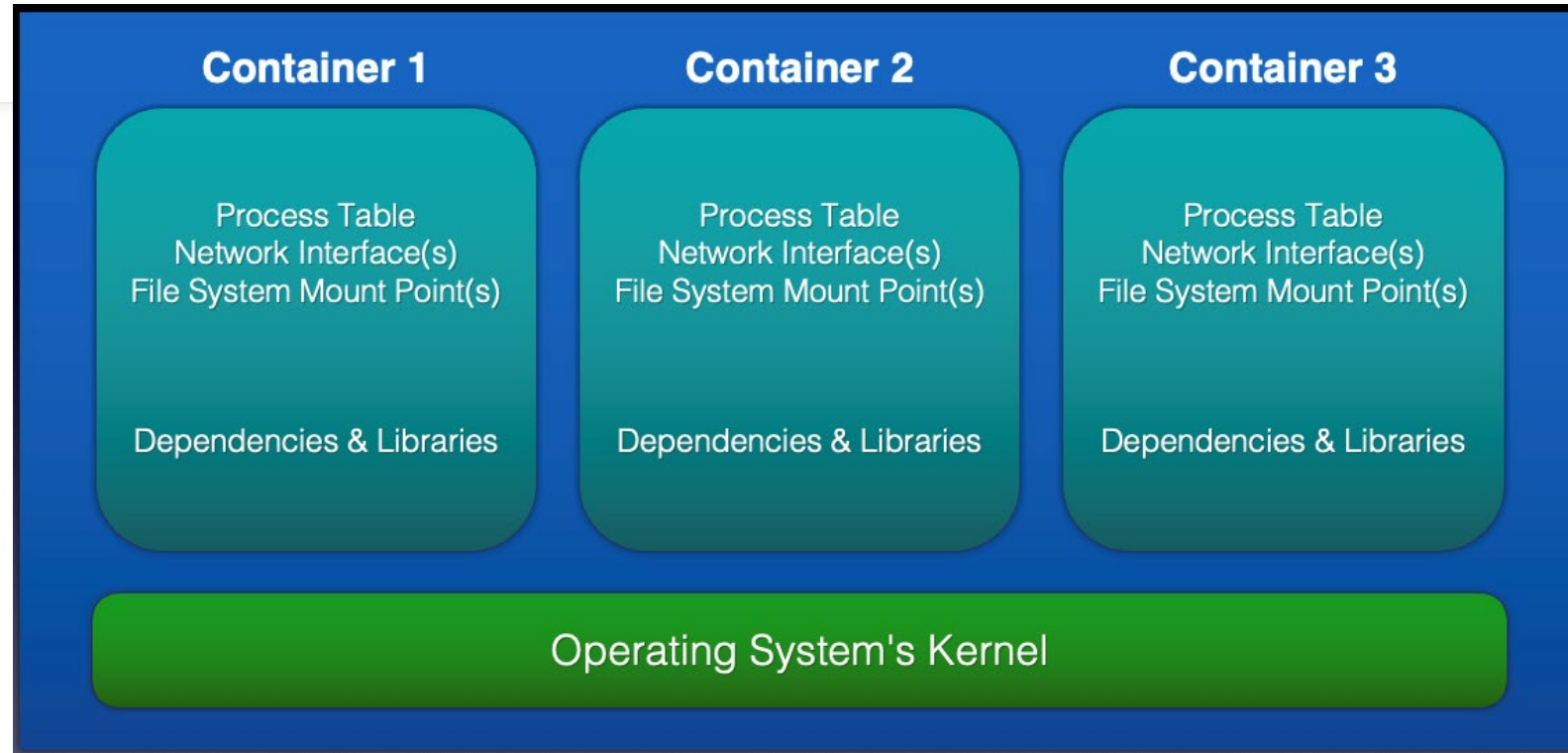
VM vs Container

- VMs and containers provides different virtualization of infrastructure.
- VMs provide emulation of the **complete computer system**, including the entire OS.
- Containers provide virtualization **only at the OS level**, allowing resources of the host computer to be partitioned through multiple isolated instances of user space.
- Therefore, **resource requirements of a container is less** than a VM. A container is also faster to start up and deploy.
- But the application and other programs running in the container believe that they have the OS to themselves! (When they actually don't; they are sharing the same host OS).

VMs vs Containers



How Docker Works



Docker Host

Each container is isolated from the other containers present on the same host.
Thus multiple containers with different application requirements and dependencies can run on the same host.

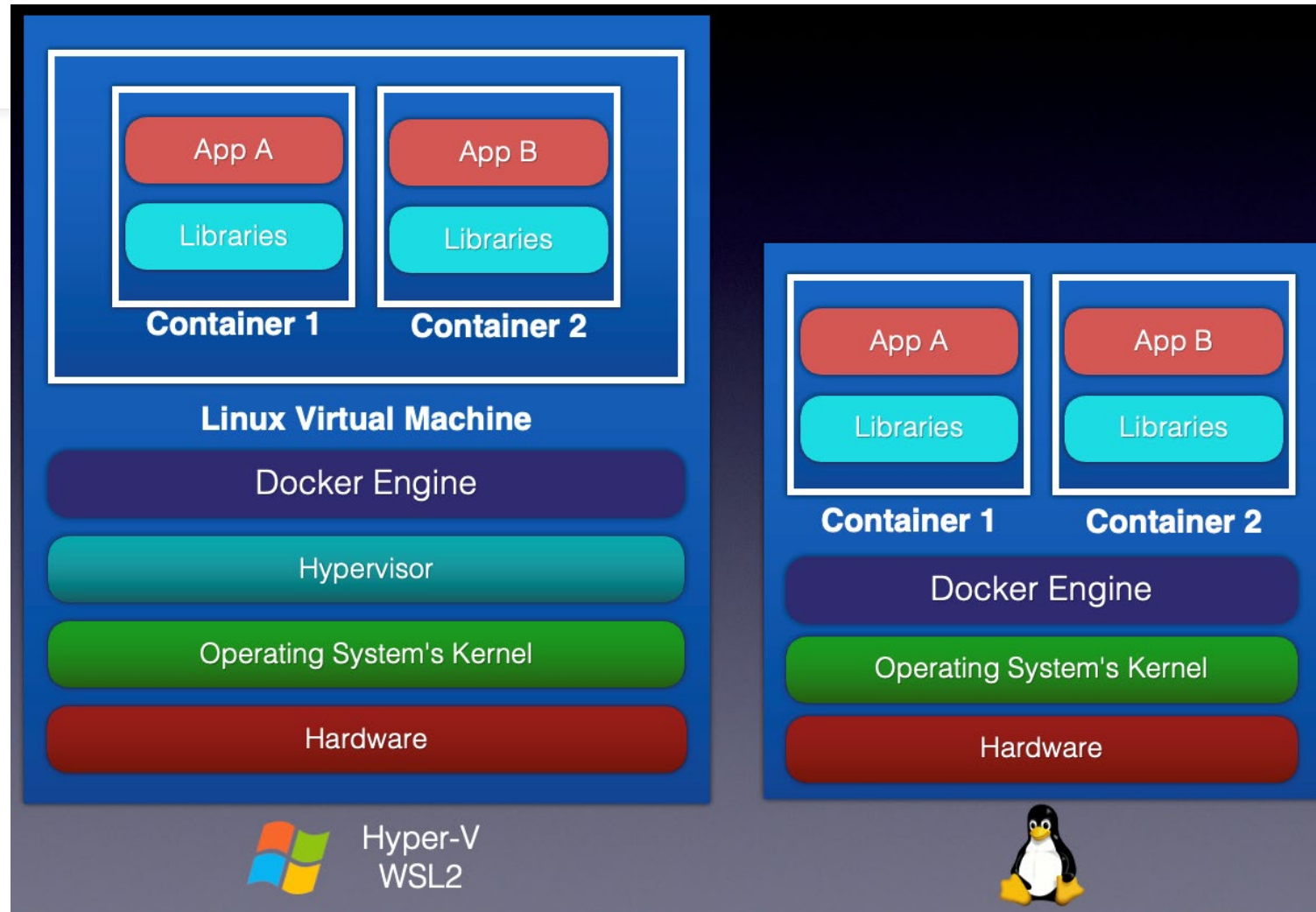
Docker: Containerization

- A Docker Container **does not have any operating system** installed and running on it
- But it would have a virtual copy of the process table, network interface(s), and the file system mount point(s)
 - These have been inherited from the operating system of the host on which the container is hosted and running
 - The **kernel of the host's operating system is shared** across all the containers that are running on it
- Docker **virtualize the operating system of the host** on which it is installed and running, rather than virtualizing the hardware components

Docker on Linux vs Docker on other OS

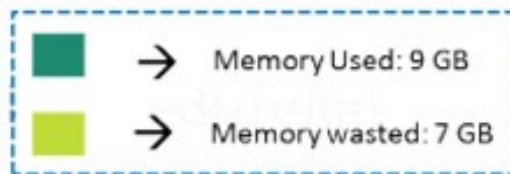
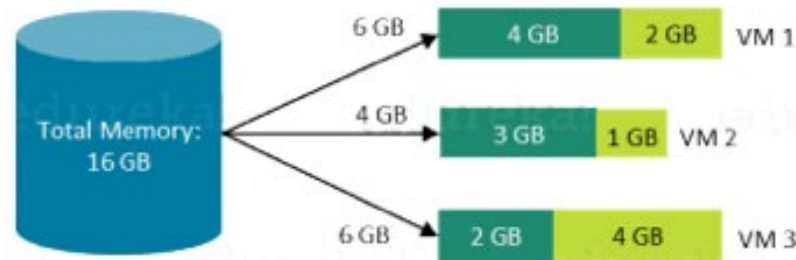
- Docker only works on Linux-based OS.
- For other OS, use of Docker require workarounds to make the tool available.
- In Linux OS, the Docker daemon and the Docker client are installed in same machine.
- In other OSes, Docker can install the client on one type of OS (e.g. Windows) and the daemon installed somewhere else, usually in a VM sitting in that OS (e.g. VirtualBox (which is a **hypervisor**), or HyperV enabled)
- You may go to <https://docs.docker.com/docker-for-windows/install/> to download Docker for the OS you are using.

Docker on Windows (and Mac) vs Docker on Linux



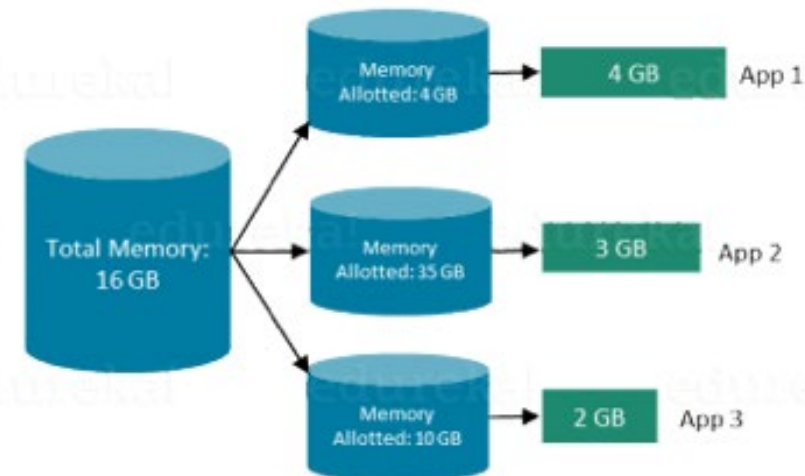
Memory Usage of VMs vs Docker

In case of Virtual Machines



7 Gb of Memory is blocked and cannot be allotted to a new VM

In case of Docker



Only 9 GB memory utilized;
7 GB can be allotted to a new Container

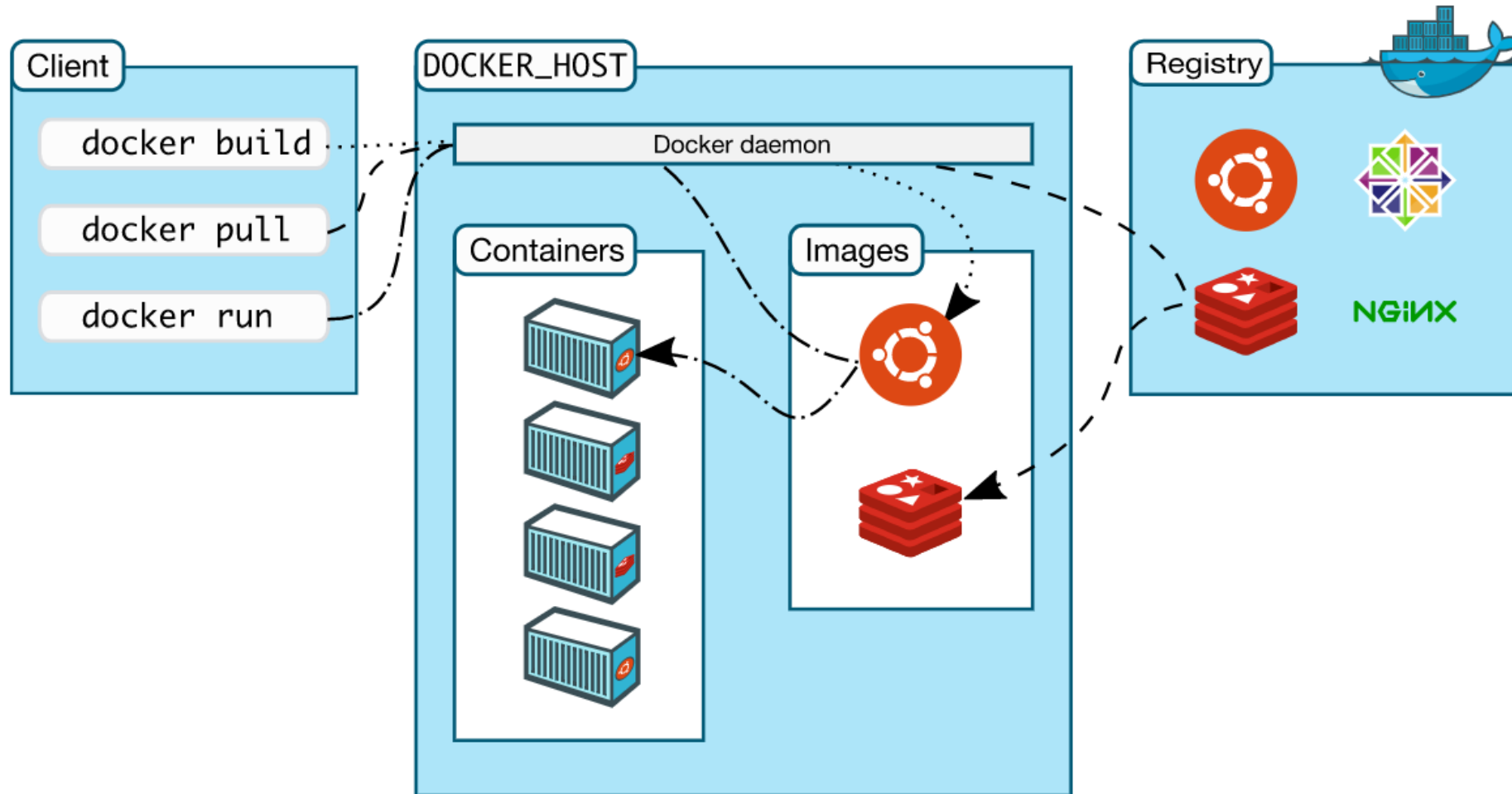
Docker concepts & Components

- The **Docker engine** or **Docker** consists of a number of components.
- First, there is **Docker client**: a command-line interface that allows you to interact with the Docker daemon. We will be using it to test out installation of Docker.
- **Docker daemon**: process that sits on the host OS that answers requests for service, and orchestrates management of containers.
- **Docker image**: read-only template that help to launch containers.
- **Docker container**: built on docker image, lightweight virtualization of programs needed to run an application.






Docker concepts & Components

- Docker images can be built in different ways. One way is to run a set of instructions contained in a file called **Dockerfile**.
- Docker images can be stored locally in same computer as the Docker daemon (also called Docker host), or they can be hosted in a **Docker registry**.
- Docker registry: a repository of Docker images. E.g. **Docker Hub** <https://hub.docker.com>. Docker Hub hosts both public and private Docker images.
- You can also run your own private registry.

Docker Concepts & Components



Docker Image

- A **Docker image** is a read-only template with instructions for creating a Docker container
- Some sample Docker Images:
 - ubuntu 
 - python 
 - golang 
 - nginx 
 - mySQL 
- You can find Docker images from: <https://hub.docker.com/>

Types of Images

Base Images

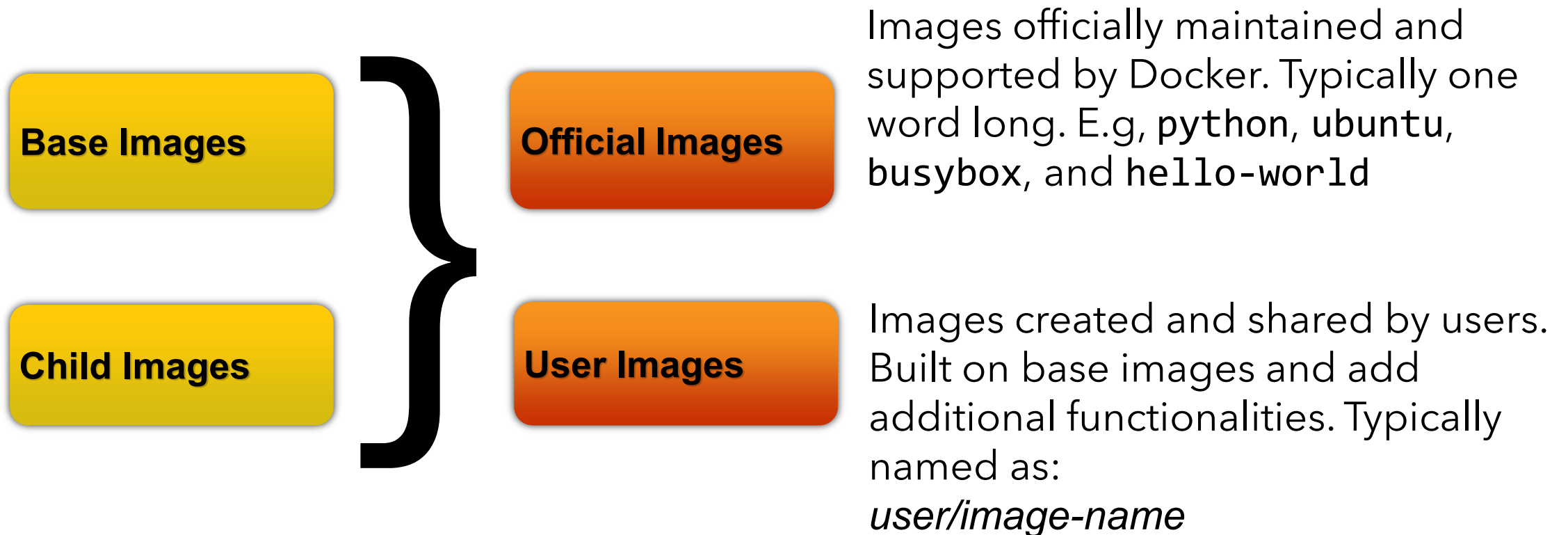
Images that have no parent images, usually images with an OS like Ubuntu, Busybox, or Debian



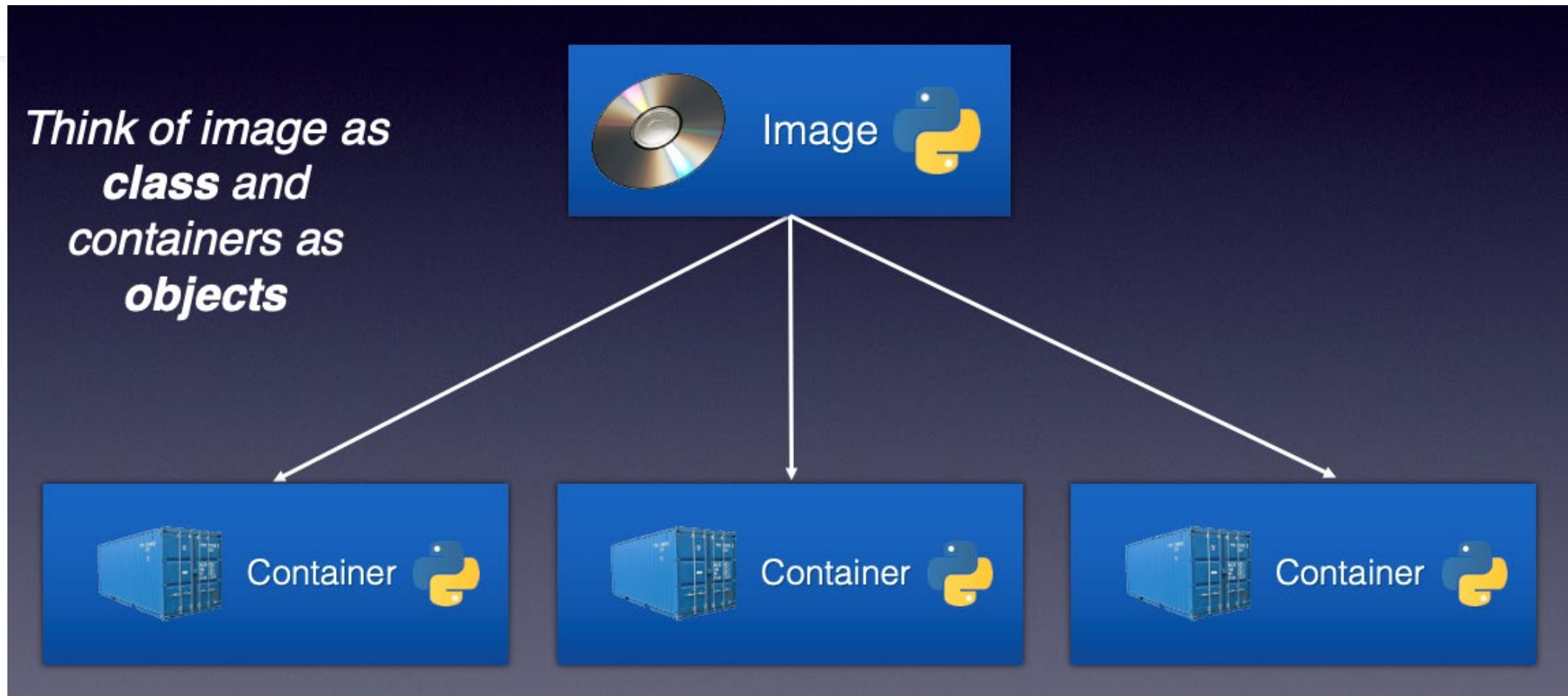
Child Images

Images that build on base images and add additional functionalities

Creating Child Images



Containers are instances of a Docker Image



MySQL

- a database system used on the web
- runs on a server
- ideal for both small and large applications
- very fast, reliable, and easy to use
- uses standard SQL
- compiles on a number of platforms
- free to download and use
- developed, distributed, and supported by Oracle Corporation
- named after co-founder Monty Widenius's daughter: My



Installing MySQL

- For Windows users:
 - Go to: <https://dev.mysql.com/downloads/file/?id=499590>
- For Mac users:
 - Go to: <https://dev.mysql.com/downloads/file/?id=499568>
 - Go to: <https://dev.mysql.com/downloads/file/?id=498743>

