

Big Data & Cloud Ecosystems

ניהול ו創ית נתונים עתק בסביבה ענן

Lesson 1



יוסי זגורו

Yossi.Zaguri@gmail.com

052-4668866

מטרות הקורס

- הכרת מושגים, תהליכיים וטכנולוגיות לטיפול בנתוני עתק
 - KDD
 - אטגרי Big Data
 - ארכיטקטורה וגישות לאחסון ו恢存 מאסות נתונים
- הקניית יכולות שימוש בתשתיות מחשב ענן
 - שירותי תוכנה וספקים מרכזיים
 - הגדרת סביבה בענן לעיבוד נתונים עתק
- הבנת האטגרים ויישום אלגוריתמים נבחרים בלמידה מכונה
 - גישות חישוביות לכירית נתונים
 - מימוש משימות ניתוח והפקת ידע

מושגי יוד



If you wish to converse with me,
define your terms.

~ Voltaire

AZ QUOTES



רקע ותזכורת



- על ההבדלים בין עיצוב וארכיטקטורה.
- עקרון SOC ומודולציה.
- ניהול Dependencies.
- מהי תוכנה ומהו שירות תוכנה.

רקע ותזכורת

- ביזור ומשמעותו.

- המעבר ממודל Scale Up ל-Scale Out.

- תהליכי פיתוח תוכנה, DevOps ופעילות CI/CD

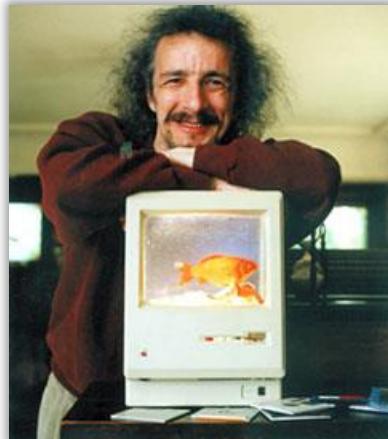
- מה משמעות המונח "וירטואלי" ?



נתונים, מידע וידע

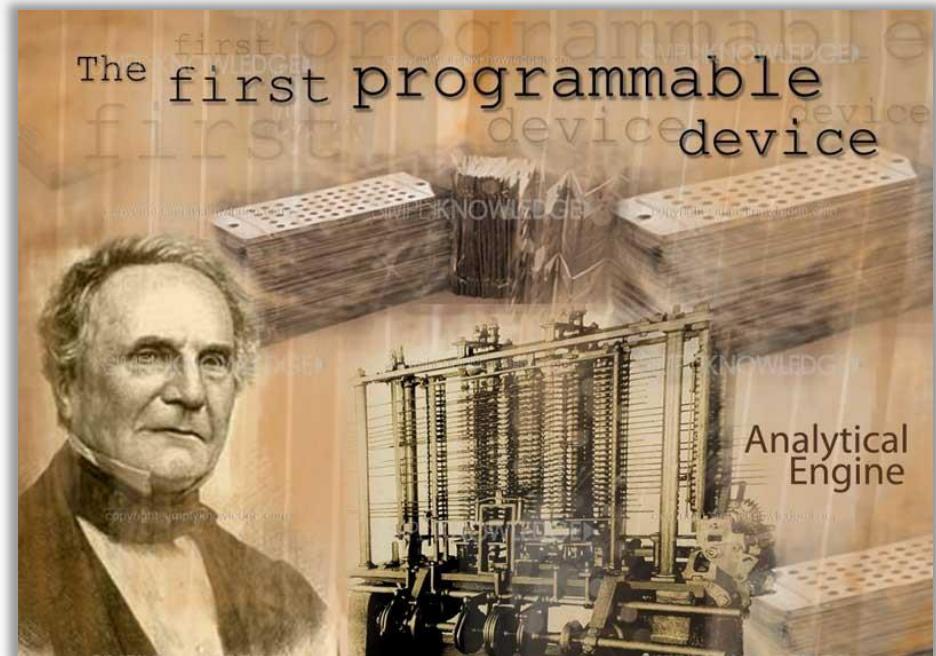
“Errors using inadequate data are much less than those using no data at all.”

Charles Babbage

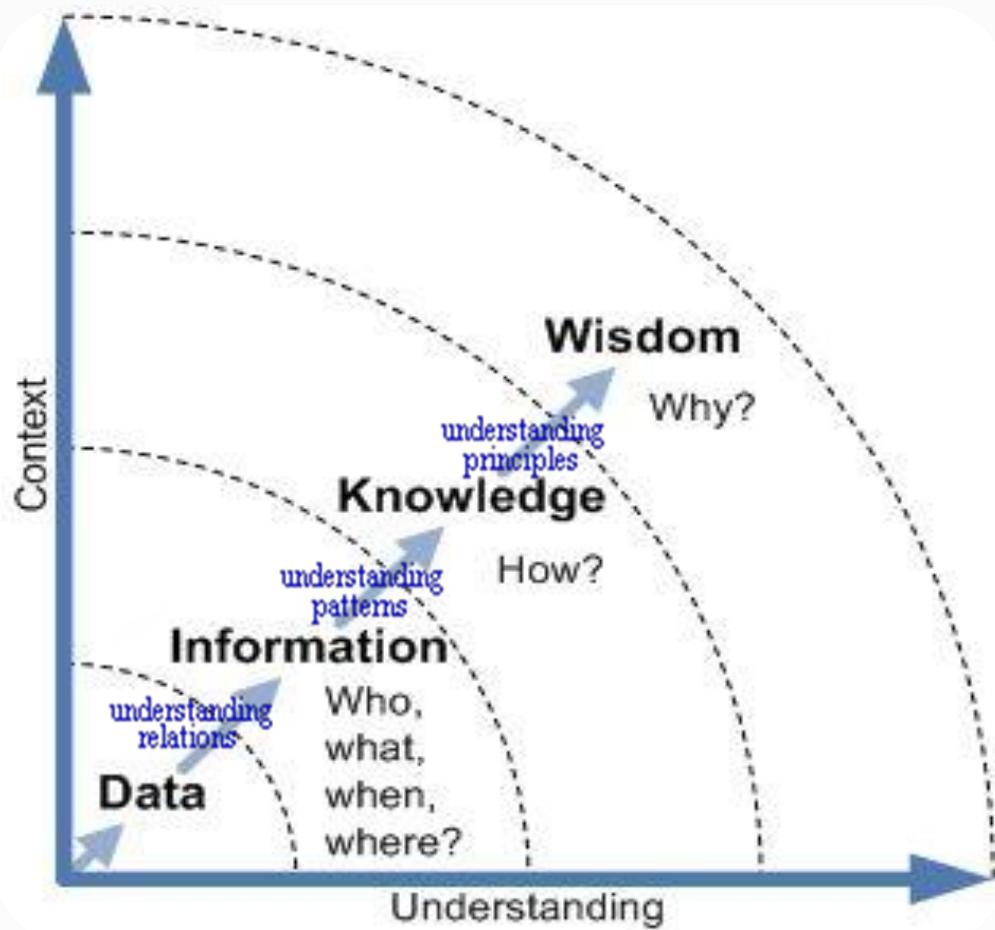


“Data is not information, information is not knowledge, knowledge is not understanding, understanding is not wisdom.”

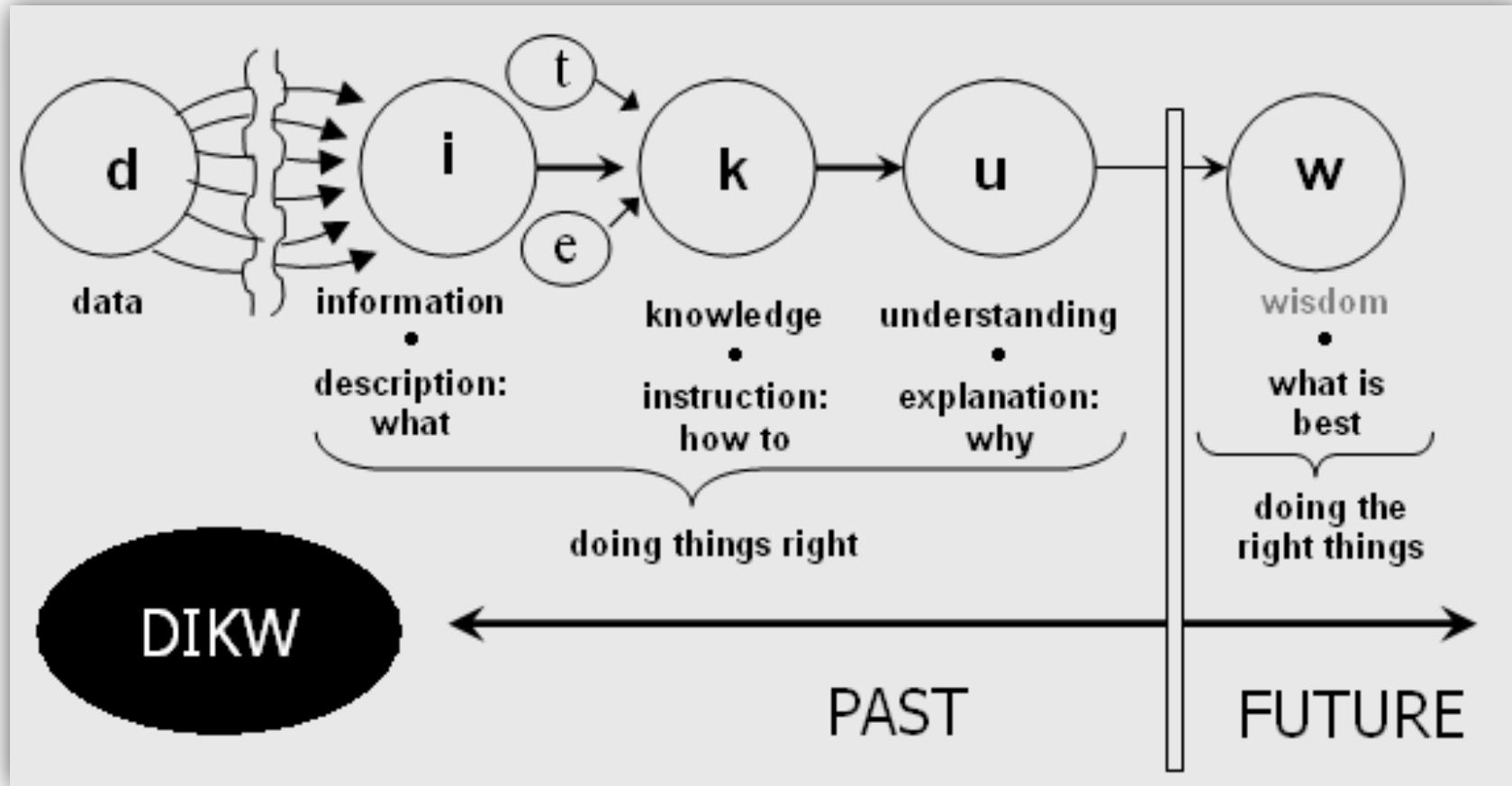
Clifford Stoll



מודל DIKW



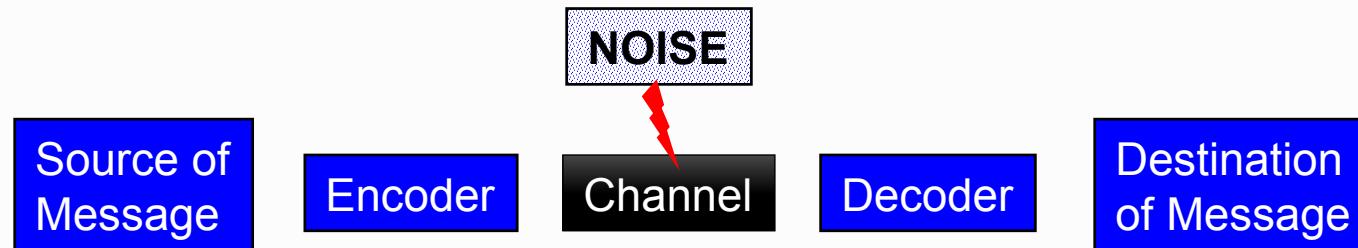
מודל DIKW



Measuring Information

Information Theory is a branch of science that deals with the analysis of a communications system.

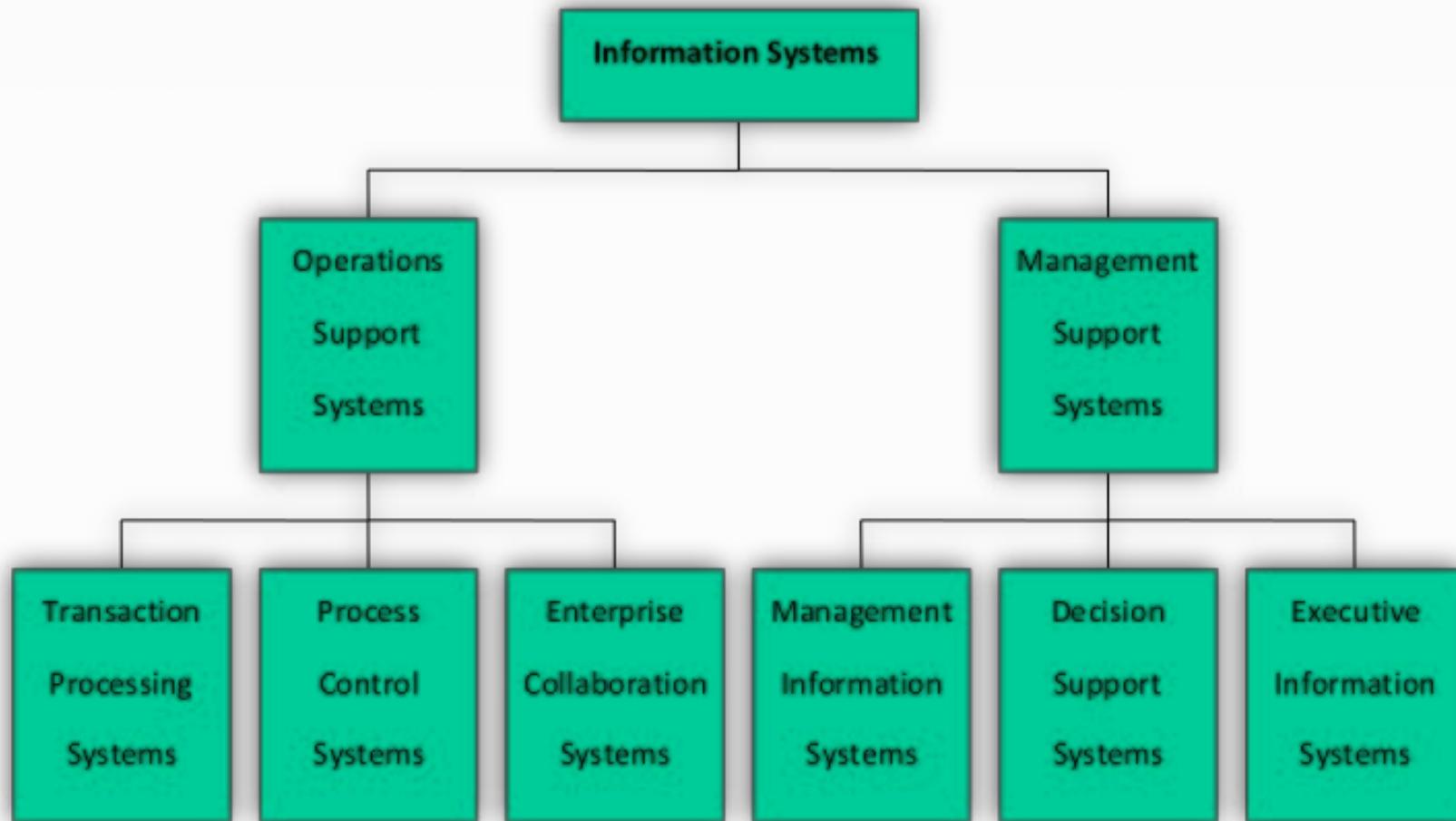
Channels can vary from a file to network protocol.



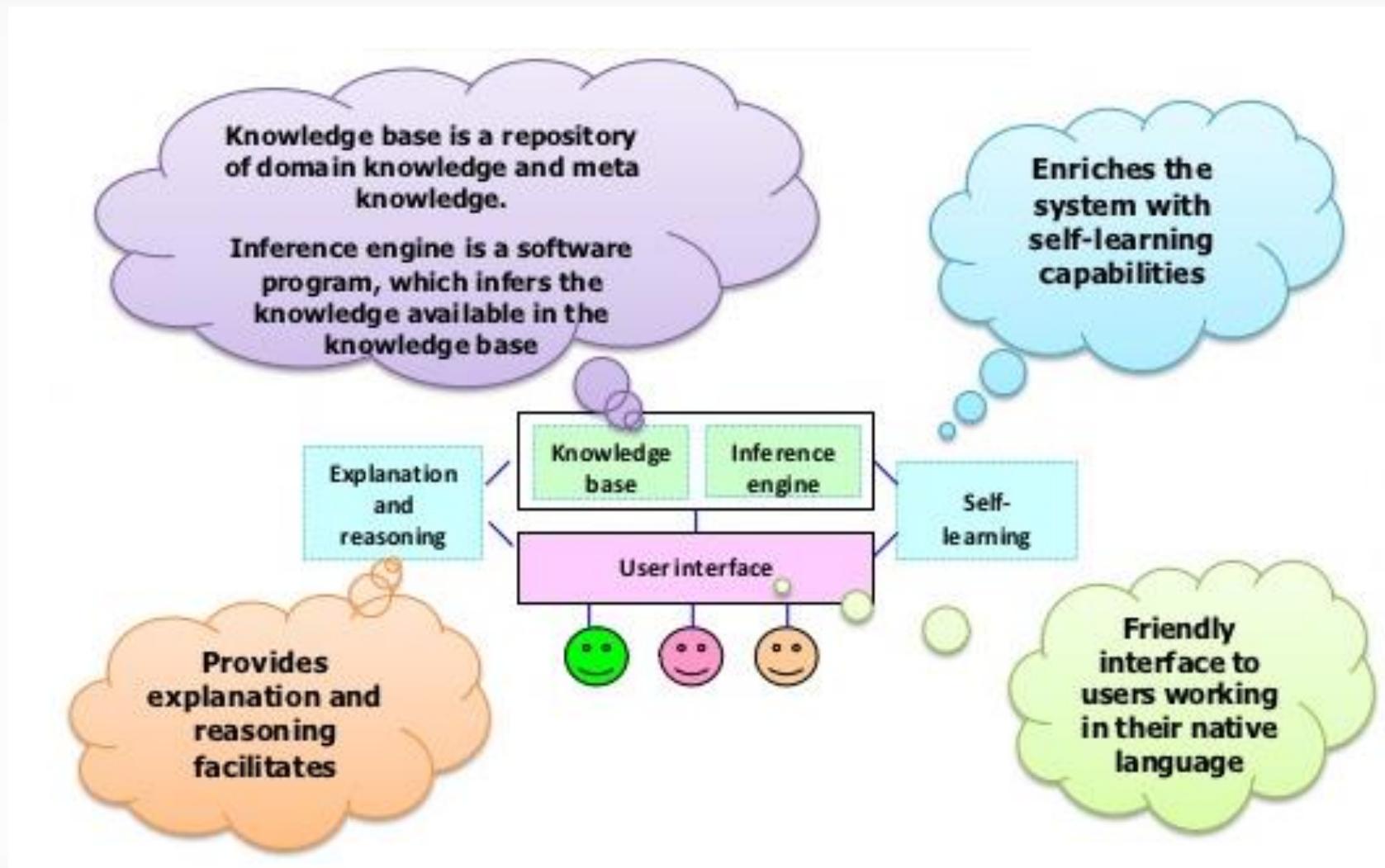
Claude Shannon Published a landmark paper in 1948 that was the beginning of the branch of information theory.

In course context, we are interested in quantifying Information levels each data element may convey

Information Systems



Knowledge Based Systems



Knowledge & AI

Principle Of Rationality

If an agent has knowledge that one of its actions will lead to one of its goals, then the agent will select this action.

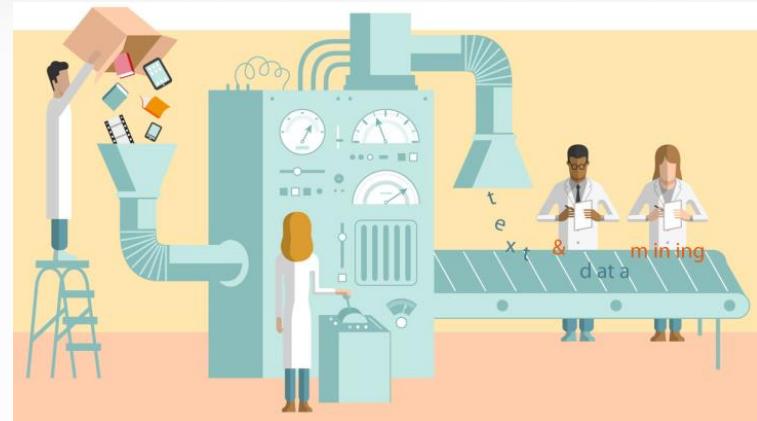
No guarantee that the knowledge is in fact sufficient to determine the right actions or even that it exists.

Knowledge makes the PR work as a *law of behavior*.

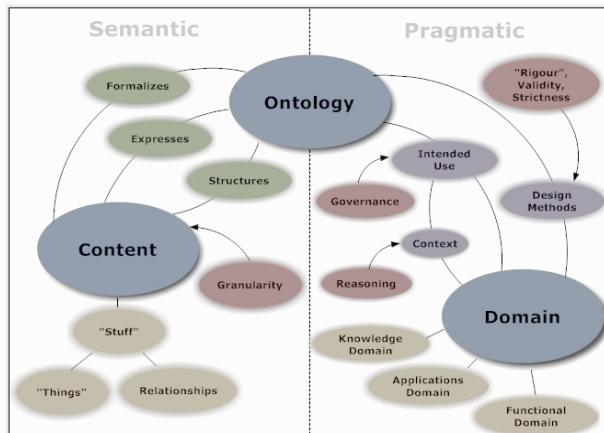
In this Context, **Knowledge** is whatever can be ascribed to the agent such that its behavior can be computed according to the PR.

In our course, we will reveal and represent **Knowledge** using Machine Learning methods, similar to those used in AI to make artificial agents *learn from experience* and improve performance over time, choosing actions according to the principle of rationality.

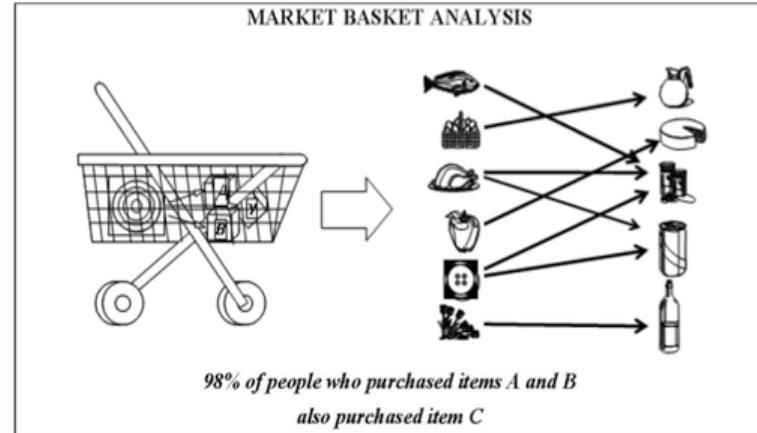
Acquiring & Representing Knowledge



Ontology

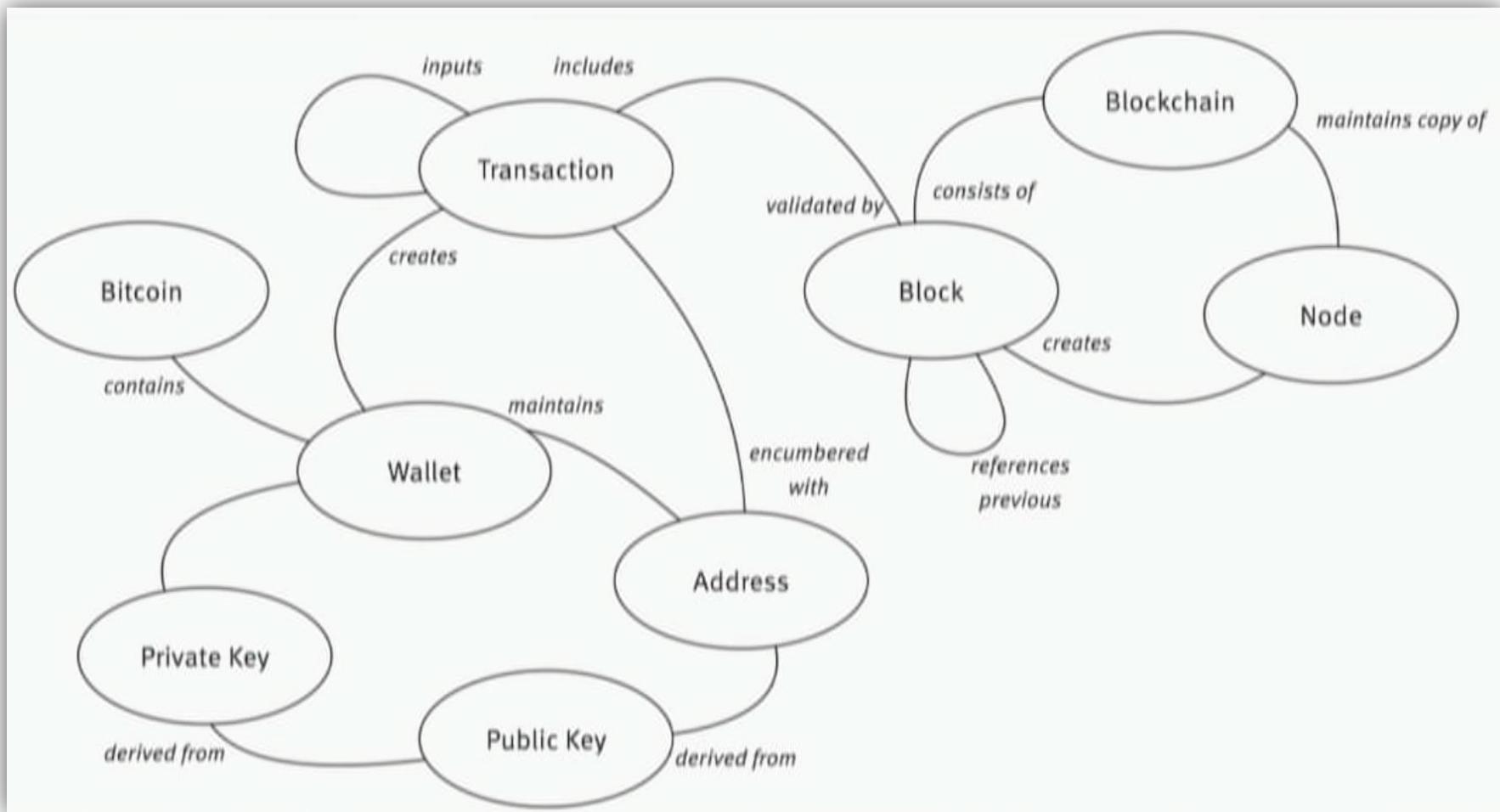


Association Rules



What is Knowledge Representation ?

Ontology Example: Bitcoin



Big Data & Cloud Ecosystems

ניהול וקרית נתונים עתק בסביבה ענן

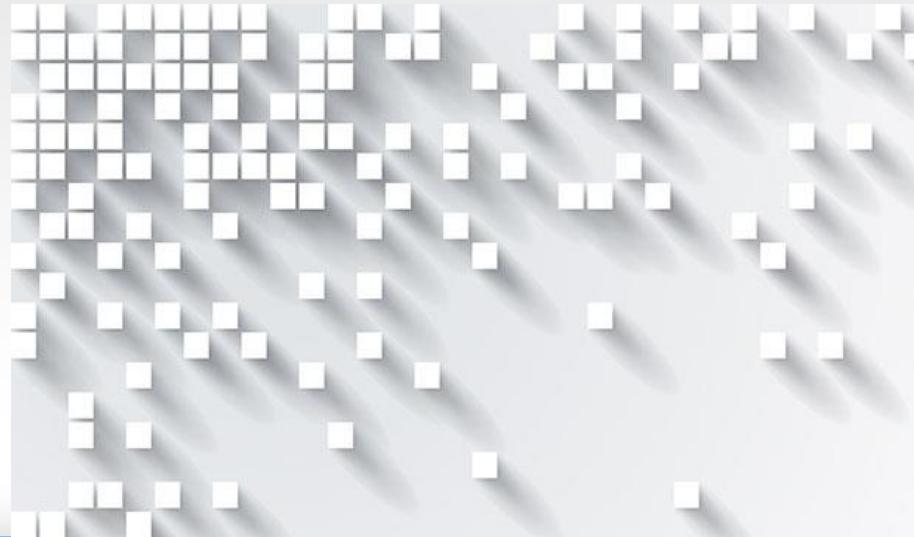
Lesson 2-4



יוסי זגורו

Yossi.Zaguri@gmail.com

052-4668866



Software Services

Services ?

שירות קלטי

- סוג של תופעה "לא מוחשית" שתוצאתה צריכה שאינה כרוכה בבעלויות.

שירותי תוכנה

Cross Platform Software Component

- שלב באבולוציה יצרת רכיבי תוכנה לשימוש חוזר
- אחד משניים : שירות אחסון או שירות עיבוד.
- יכול להיות במסגרת ענן, או לא.

שירותי ענן

רמות גרנולציה שונות

- מאקרו : IaaS, PaaS, SaaS
- מיקטו : Web Service

Software Services

- יחידת פונקציונליות המספקת עיבוד או נתונים.
- "הירושה" של Software Component.
- מתאפיינת בצדמוד נמור וביכולת לשרת מגוון פלטפורמות וטכנולוגיות.
- מתקשרת בינהן באמצעות מסרים הכללים Meta Data.

Software Services

❖ פראדיגמת עיצוב & תוכנות (בדומה ל-O.O)

❖ עקרונות עיצוב שירות תוכנה (Don Box) מאט תוכנה

❖ גבולות מוגדרים מפורשות

“Service-orientation is based on a model of explicit message passing rather than implicit method invocation”

❖ אוטונומיות

“Service-oriented development departs from object-orientation by assuming that atomic deployment of an application is the exception, not the rule.”

Software Services

▫ אינטראקטיבית מבוססת סכימות וחוזים, לא מחלקות

“Classes are convenient abstractions as they share both structure and behavior in a single named unit. Service-oriented development has no such construct. Rather, services interact based solely on schemas (for structures) and contracts (for behaviors)”

תאימות נגזרת מהגדרות Policy

Object-oriented designs often confuse structural compatibility with semantic compatibility. Structural compatibility is based on contract and schema. Semantic compatibility based on explicit statements of capabilities & requirements in the form of policy Expression.

Policy expressions indicate which conditions and guarantees (called assertions) must hold true to enable the normal operation of the service

Cloud Services

IaaS

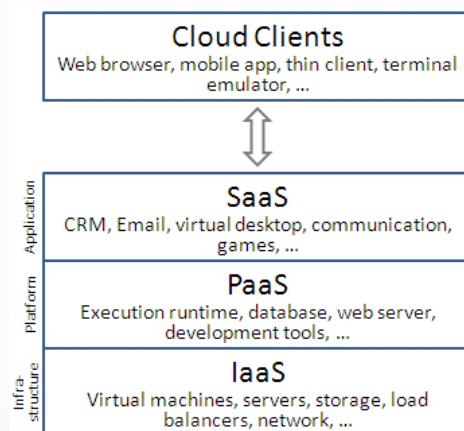
- מכונות וירטואליות, אחסון, נתבים וירטואליים...

PaaS

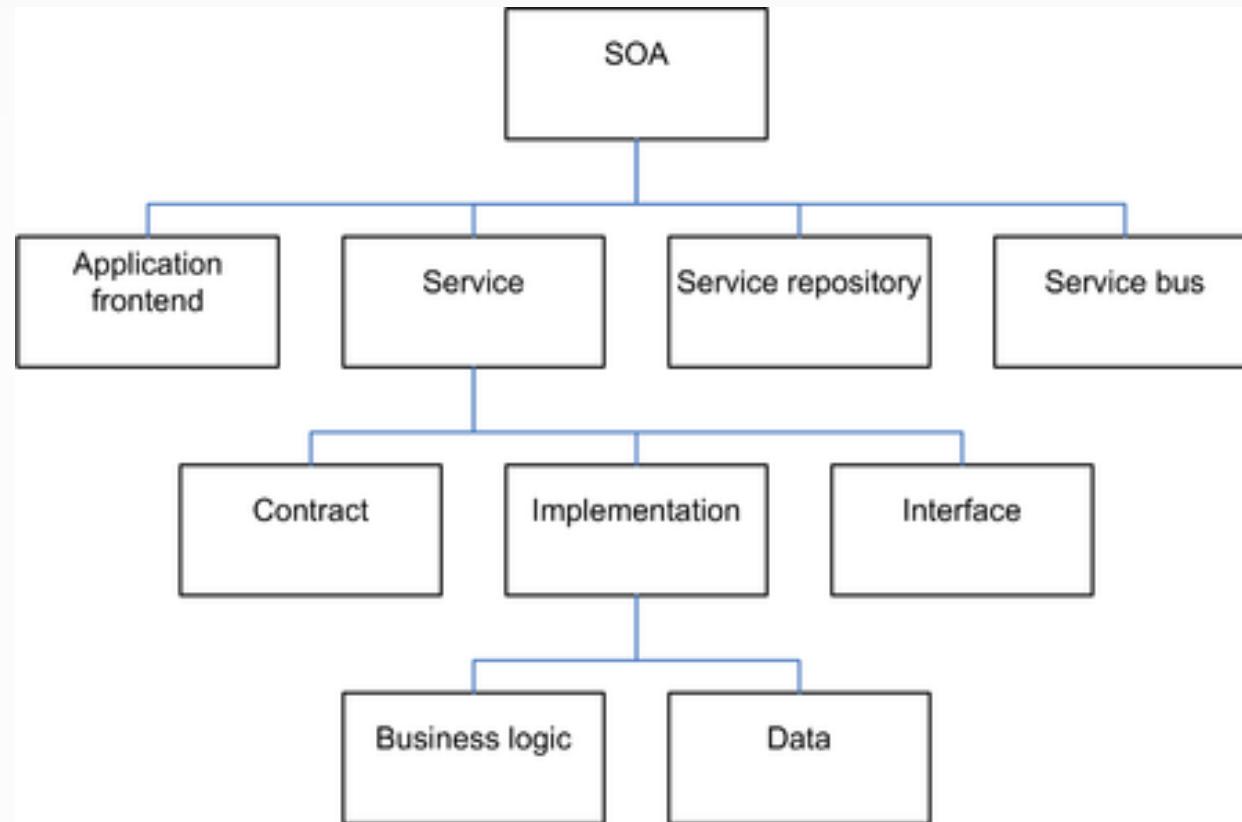
- תשתיות לפיתוח אתרים
- אירוח שירותים ענן
- אחסון וניהול נתונים (טבלאות נתונים , Blobs , DB רלציוני)
- שירותים ניהול משתמשים וזהויות (Service Bus , תורים , ...)

SaaS

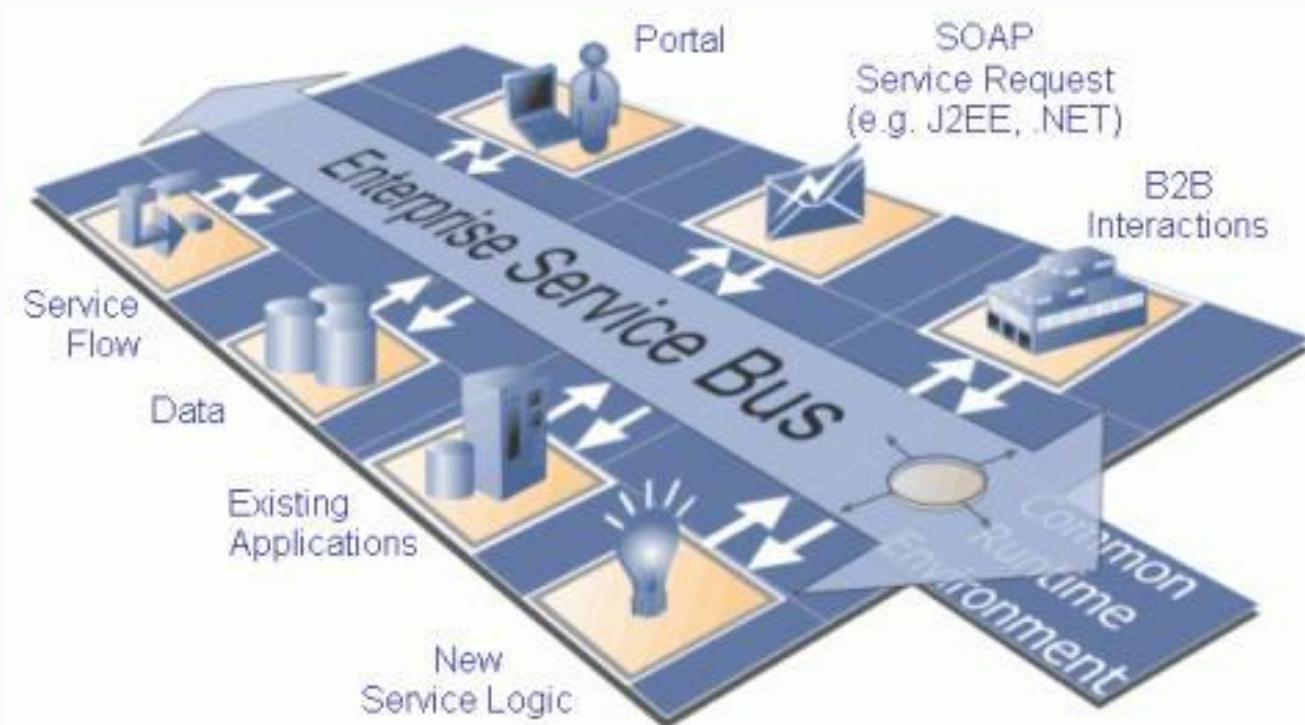
... SalesForce, Goolge Docs, Office 365



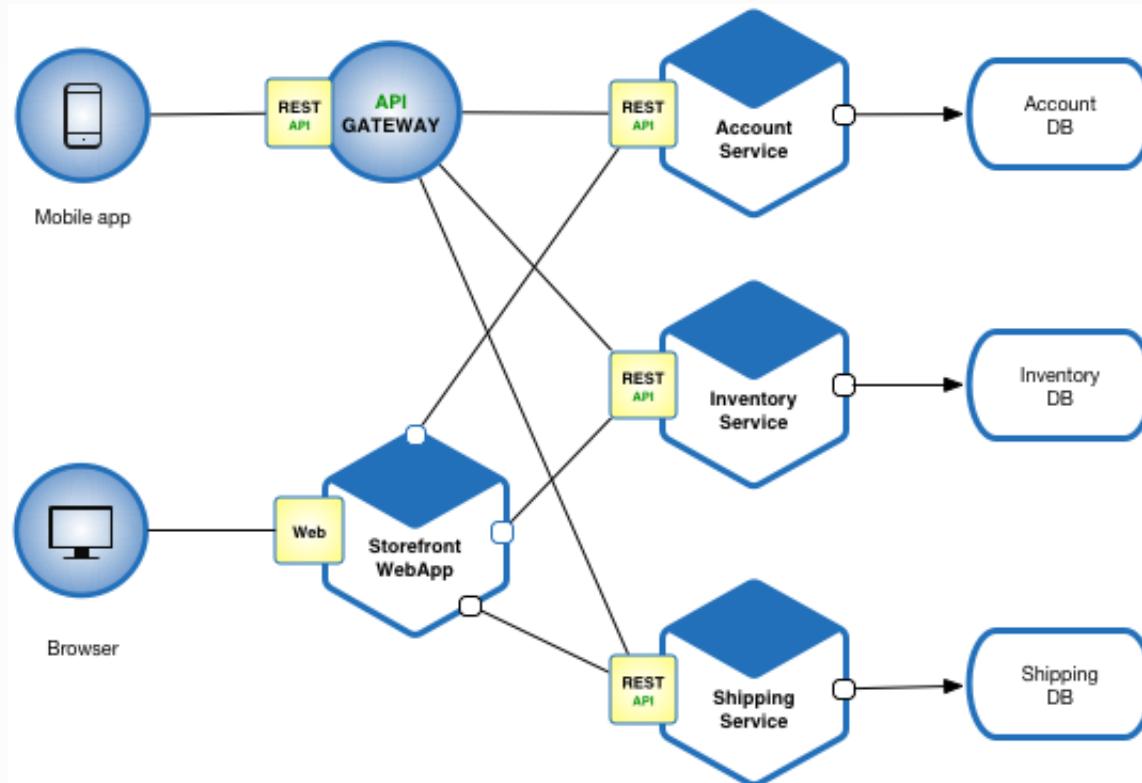
Service Oriented Architecture



Service Oriented Architecture



Microservices Architecture



<http://microservices.io/patterns/microservices.html>

Micro Services Pros

- Microservice architecture gives developers the freedom to independently develop and deploy services
- A microservice can be developed by a fairly small team
- Code for different services can be written in different languages (though many practitioners discourage it).
- Easy integration and automatic deployment (using open-source continuous integration tools such as Jenkins, Hudson, etc.)
- Easy to understand & modify for developers, can help a new team member become productive quickly.
- The code is organized around business capabilities.
- When change is required in a certain part of the application, only the related service can be modified and redeployed—no need to modify and redeploy the entire application.
- Better fault isolation: if one microservice fails, the other will continue to work (although one problematic area of a monolith application can jeopardize the entire system).
- Easy to scale and integrate with third-party services.
- No long-term commitment to technology stack, Can adopt new one's easily.

Micro Services Cons

- Due to distributed deployment, testing can become complicated and tedious.
- Increasing number of services can result in information barriers.
- The architecture brings additional complexity as the developers have to mitigate fault tolerance, network latency, and deal with a variety of message formats as well as load balancing.
- Being a distributed system, it can result in duplication of effort.
- When number of services increases, integration and managing whole products can become complicated
- In addition to several complexities of monolithic architecture, the developers have to deal with the additional complexity of a distributed system.
- Developers have to put additional effort implementing mechanism of communication between the services
- Handling use cases that span more than one service without using distributed transactions is not only tough but also requires communication and cooperation between different teams.
- Partitioning the application into microservices is very much an art.

What are Microservices ?

Communication & Protocols



REST & HTTP

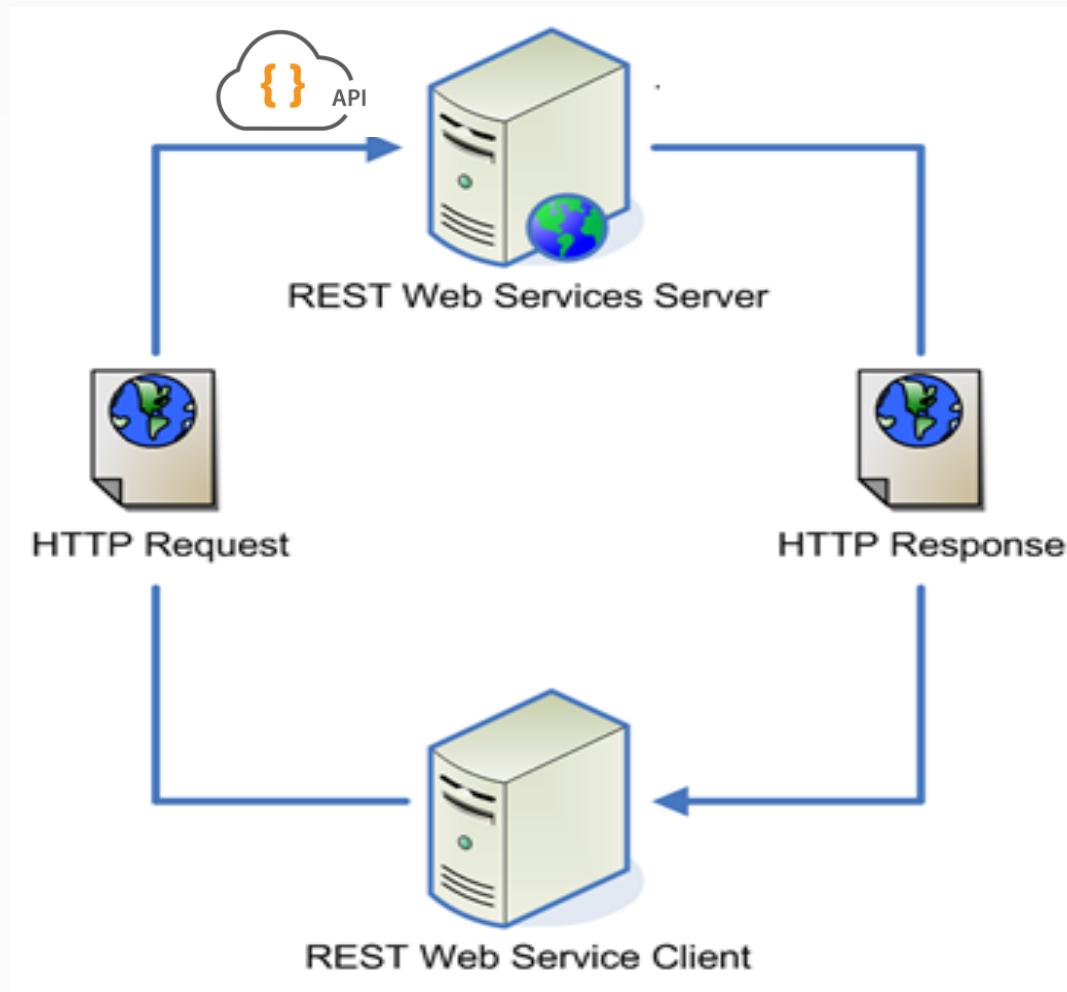
What is REST

REST

What is Http

HTTP

REST API



Data Exchange Formats



```

src,Eqid,Version,Datetime,Lon,Magnitude,Depth,NST,Region
ak,10654594,1,"Tuesday, February 12, 2013 09:08:48 UTC",62.7099,-151.8627,2.0,1.90,18,"Central Alaska"
ak,10654587,1,"Tuesday, February 12, 2013 08:53:37 UTC",59.9869,-152.6854,2.6,112.40,18,"Southern Alaska"
us,c000f5w2,4,"Tuesday, February 12, 2013 08:51:32 UTC",-3.2074,129.1016,4.3,10.00,24,"Seram, Indonesia"
ak,10654581,1,"Tuesday, February 12, 2013 08:38:51 UTC",63.0651,-150.8104,1.7,97.80,20,"Central Alaska"
ak,10654575,1,"Tuesday, February 12, 2013 08:20:55 UTC",60.7567,-150.5577,1.7,29.50, 6,"Kenai Peninsula, Alaska"
nc,71935890,0,"Tuesday, February 12, 2013 07:56:39 UTC",38.7468,-122.7002,1.0,1.80, 9,"Northern California"
nn,00402618,9,"Tuesday, February 12, 2013 07:49:08 UTC",36.6200,-117.0137,1.4,10.20,16,"Central California"
ak,10654569,1,"Tuesday, February 12, 2013 07:40:42 UTC",63.5432,-147.5335,1.3,0.20, 8,"Central Alaska"
ci,11243162,0,"Tuesday, February 12, 2013 07:22:01 UTC",33.2982,-116.5497,1.0,54.00,12,"Southern California"
ak,10654562,1,"Tuesday, February 12, 2013 07:21:20 UTC",62.0603,-145.1054,1.7,17.90,11,"Central Alaska"
ak,10654559,1,"Tuesday, February 12, 2013 07:14:50 UTC",62.0639,-145.2827,1.8,17.30,14,"Central Alaska"
nc,71935855,1,"Tuesday, February 12, 2013 06:56:41 UTC",38.8398,-122.8278,1.4,2.30,27,"Northern California"
ak,10654548,1,"Tuesday, February 12, 2013 06:40:46 UTC",63.3990,-151.4516,2.1,0.10,13,"Central Alaska"
ak,10654546,1,"Tuesday, February 12, 2013 06:31:36 UTC",63.0453,-151.4555,1.3,0.00, 8,"Central Alaska"
uw,60505786,2,"Tuesday, February 12, 2013 05:53:56 UTC",45.2953,-121.7413,1.7,7.00,12,"Mount Hood area, Oregon"
ak,10654534,1,"Tuesday, February 12, 2013 05:50:22 UTC",63.5352,-149.5146,1.1,100.70, 4,"Central Alaska"
nc,71935825,0,"Tuesday, February 12, 2013 05:46:57 UTC",38.8288,-122.8548,1.0,2.00,18,"Northern California"
ak,10654528,1,"Tuesday, February 12, 2013 05:24:55 UTC",64.1457,-145.6067,1.4,20.90, 5,"Central Alaska"
nc,71935805,1,"Tuesday, February 12, 2013 05:19:49 UTC",36.5478,-121.1385,2.3,6.70,34,"Central California"
uw,60505766,2,"Tuesday, February 12, 2013 04:30:36 UTC",47.8732,-121.6530,1.5,10.30,17,"Washington"
ci,11243130,0,"Tuesday, February 12, 2013 04:23:40 UTC",35.4408,-118.3158,1.5,8.30,29,"Central California"
ci,11243122,0,"Tuesday, February 12, 2013 04:16:50 UTC",35.4382,-118.3115,1.8,6.80,46,"Central California"
hv,60449182,2,"Tuesday, February 12, 2013 04:10:33 UTC",19.4307,-155.3082,2.4,6.00,25,"Island of Hawaii, Hawaii"
ak,10654508,1,"Tuesday, February 12, 2013 04:00:29 UTC",60.0486,-152.2403,3.5,98.80,51,"Southern Alaska"

```



```

<Books>
  <Book ISBN="0553212419">
    <title>Sherlock Holmes: Complete Novels...</title>
    <author>Sir Arthur Conan Doyle</author>
  </Book>
  <Book ISBN="0743273567">
    <title>The Great Gatsby</title>
    <author>F. Scott Fitzgerald</author>
  </Book>
  <Book ISBN="0684826976">
    <title>Undaunted Courage</title>
    <author>Stephen E. Ambrose</author>
  </Book>
  <Book ISBN="0743203178">
    <title>Nothing Like It In the World</title>
    <author>Stephen E. Ambrose</author>
  </Book>
</Books>

```

Data Exchange Formats



```
one,two,three
1,2,3
4,3,2
```

CSV to JSON

```
[{"one": "1", "two": "2", "three": "3"},  
 {"one": "4", "two": "3", "three": "2"}]
```



XML to JSON

XML	JSON
<pre>... > <name>Barry & Associates, Inc.</name> <phone>612-321-8156</phone> <street1>14597 Summit Shores Dr</street1> <street2></street2> <city>Burnsville</city> <state>MN</state> <postalcode>55306</postalcode> <country>United States</country> < ... ></pre>	<pre>{ "name": "Barry & Associates, Inc.", "phone": "612-321-8156", "street1": "14597 Summit Shores Dr", "street2": "", "city": "Burnsville", "state": "MN", "postalcode": "55306", "country": "United States" }</pre>



{ JSON }
JavaScript Object Notation

DEMO



<https://www.codementor.io/nodejs/tutorial/how-to-use-json-files-in-node-js>

Big Data & Cloud Ecosystems

ניהול ו創ית נתונים עתק בסביבה ענן

Lesson 5 - 13



יוסי זגורו

Yossi.Zaguri@gmail.com

052-4668866

Implementing RESTful Services



Node.js Demo

<https://www.tutorialspoint.com/nodejs/index.htm>

http://courseware.codeschool.com/node_slides.pdf



Software Containers

Containers

Who is Malcom McLean ?

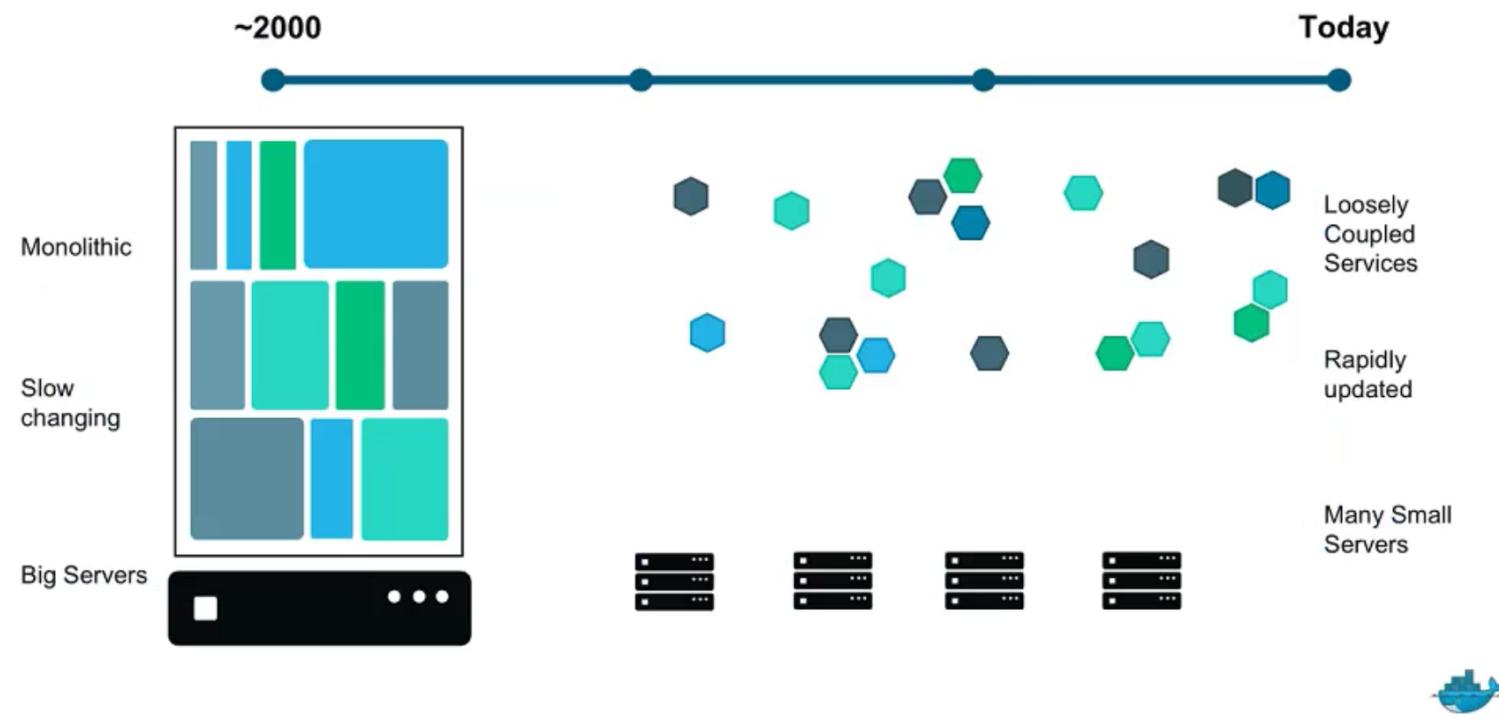


הקופסא ששינה את העולם

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

מגמות בארכיטקטורת מערכות תוכנה

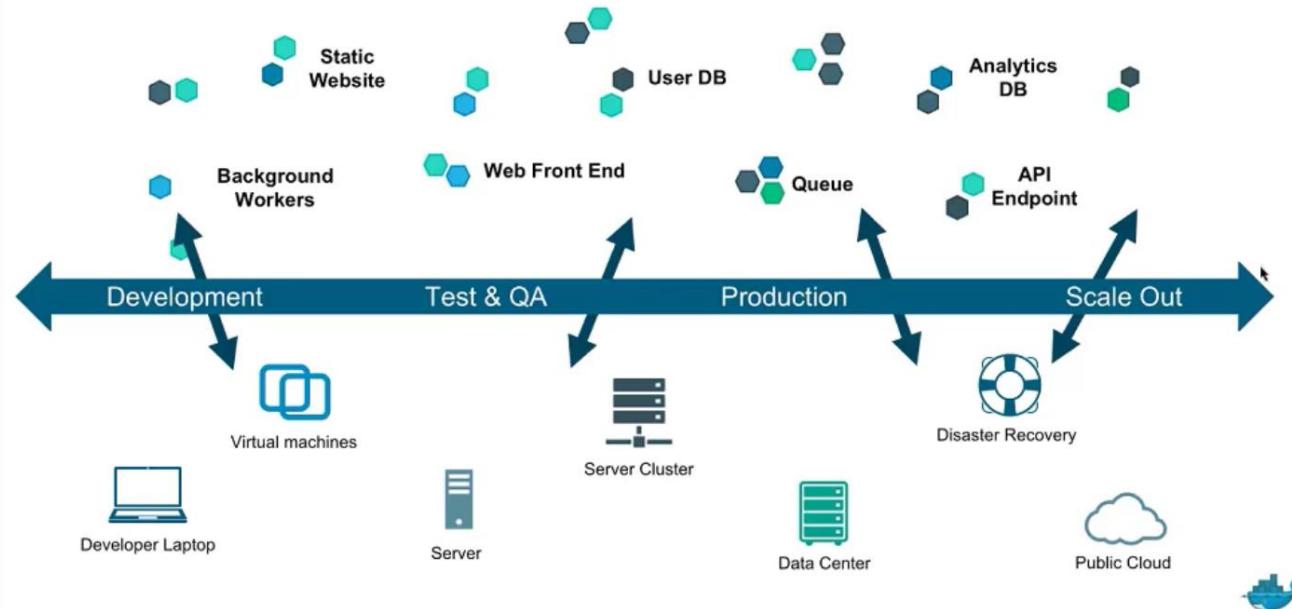
Transforming the Application Landscape



מערכות מונוליטיות לתלcid שירותי תוכנה

אתגרים במודול שירותי

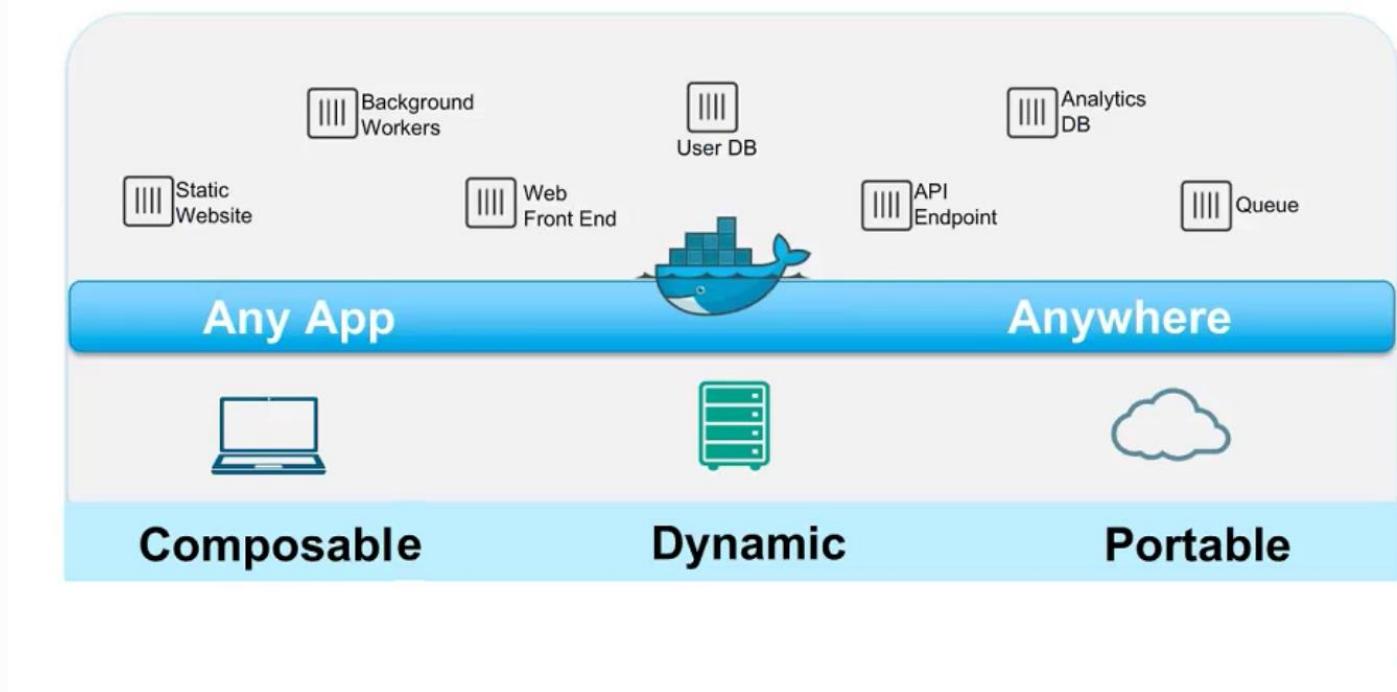
The New Challenge of Distributed Apps



איך להעביר את התוכנה מפיתוח לkiemן, ואיך לבצע ? Scaling ?

הפתרון : מיכליים

Distributed Application Solution

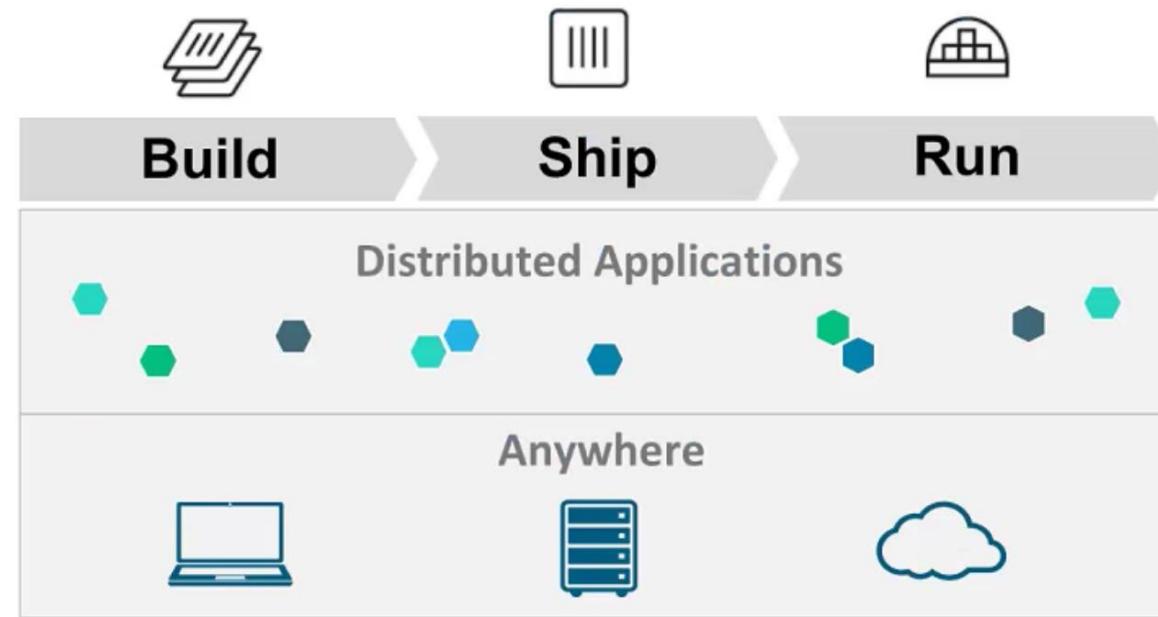


איך להעביר את התוכנה מפיתוח ליתור, ואח"כ לבצע ? Scaling ?

Docker כמיון חכם

- מסיע במבנה מערכת מבוזרת כך שאפשר להריץ אותה בכל מקום, גם מחשב נייד
- יצרת חבילת שהיא Composable, Dynamic, Portable עם כל התלויות.

The Docker mission



כמיכל Docker

Let's start with some basic concepts



Docker Image

The basis of a Docker container



Docker Container

The standard unit in which the application service resides



Docker Engine

Creates, ships and runs Docker containers deployable on physical or virtual host locally, in a datacenter or cloud service provider



Docker Trusted Registry

Dedicated image store and distribution service deployed in your firewall



Docker

“Container adoption is being driven by the promise that containers deliver the ability to ‘build once and run anywhere,’ allowing increased server efficiency and scalability for technology managers.”

Robert Stroud, Analyst



<http://containerjournal.com/>

Docker

<http://www.dwmkerr.com/learn-docker-by-building-a-microservice/>

```
docker run --name db -d -e MYSQL_ROOT_PASSWORD=123 -p 3306:3306 mysql:latest
```

This starts a MySQL instance running, allowing access through port 3306 using the root password 123.

1. `docker run` tells the engine we want to run an image (the image comes at the end, `mysql:latest`)
2. `--name db` names this container `db`.
3. `-d` (or `--detach`) detach - i.e. run the container in the background.
4. `-e MYSQL_ROOT_PASSWORD=123` (or `--env`) environment variables - tells docker we want to provide an environment variable. The variable following it is what the MySQL image checks for setting the default root password.
5. `-p 3306:3306` (or `--publish`) tells the engine that we want to map the port 3306 from inside the container to out port 3306.

The last part is really important - even though that's the MySQL default port, if we don't tell docker explicitly we want to map it, it will block access through that port (because containers are isolated until you tell them you want access).

Docker

The image is a promotional graphic for Docker. At the top center is the Docker logo, featuring a blue whale carrying a stack of shipping containers. Below the logo is the slogan "Build, Ship, Run, Any App Anywhere". A large central arrow points from left to right, representing the flow from development to operations. On the left side, under the heading "From Dev", there is a icon of a laptop with code symbols (< >) and the text "Any App". Below this are icons for various application frameworks: React, Microsoft ASP.NET, .NET, Java, Ruby, Python, Node.js, and a "MORE" button. On the right side, under the heading "To Ops", there is an icon of a server with monitoring graphs and the text "Any OS". Below this are icons for Windows and Linux operating systems. At the bottom, under the heading "Anywhere", there are icons for Physical (server racks), Virtual (two overlapping squares), and Cloud (a cloud shape). The background features a blue gradient with white diagonal stripes.

Build, Ship, Run, Any App Anywhere

From Dev

Any App

To Ops

Any OS

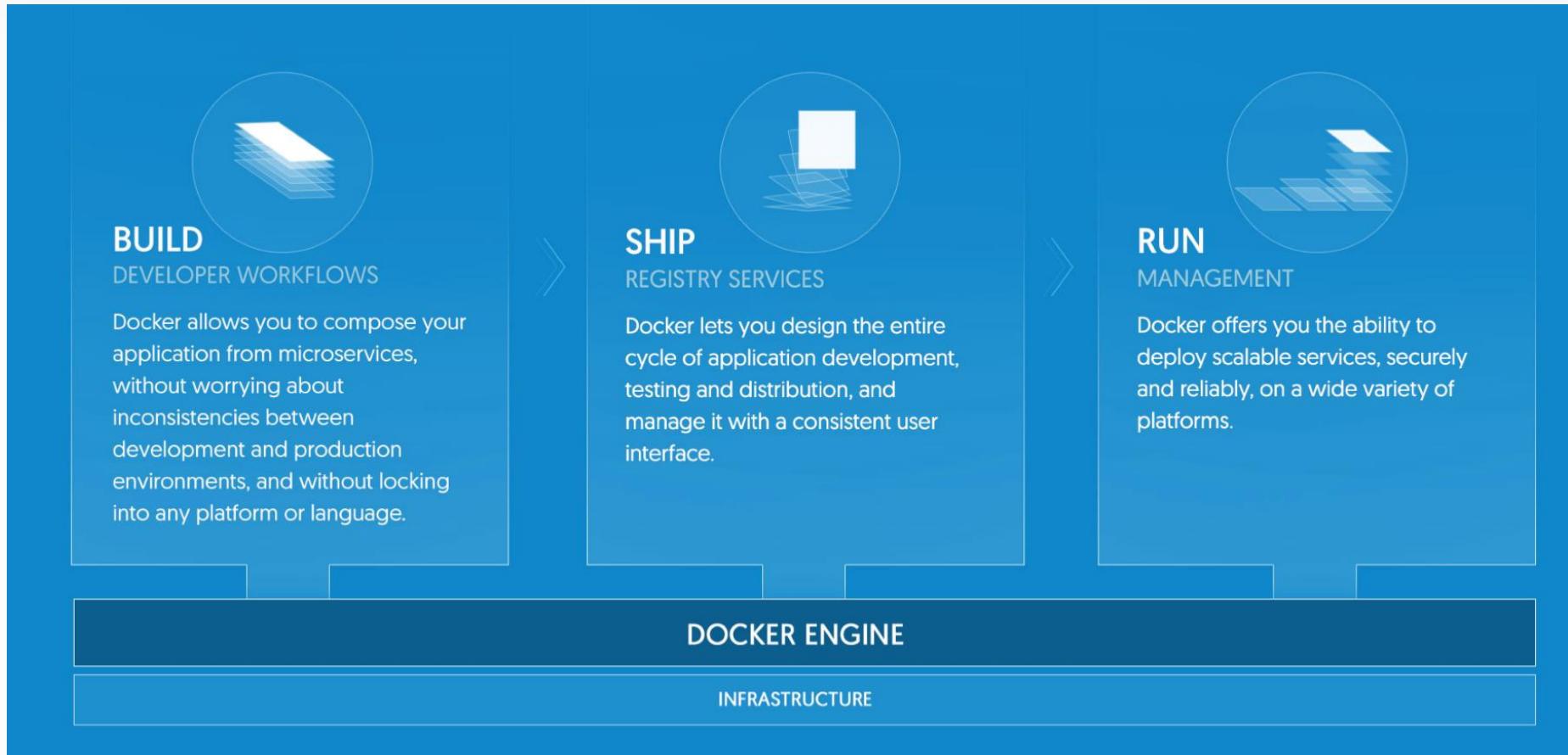
Anywhere

Physical

Virtual

Cloud

Docker



Docker ID

An open-source project that automates the deployment of applications inside software containers

	docker
Original author(s)	Solomon Hykes
Developer(s)	Docker, Inc.
Initial release	13 March 2013; 3 years ago
Stable release	1.11.2 ^[1] / 31 May 2016; 36 days ago
Written in	Go ^[2]
Operating system	Linux ^[a]
Platform	x86-64 with modern Linux kernel
Type	Operating-system-level virtualization
License	Apache License 2.0
Website	www.docker.com ↗

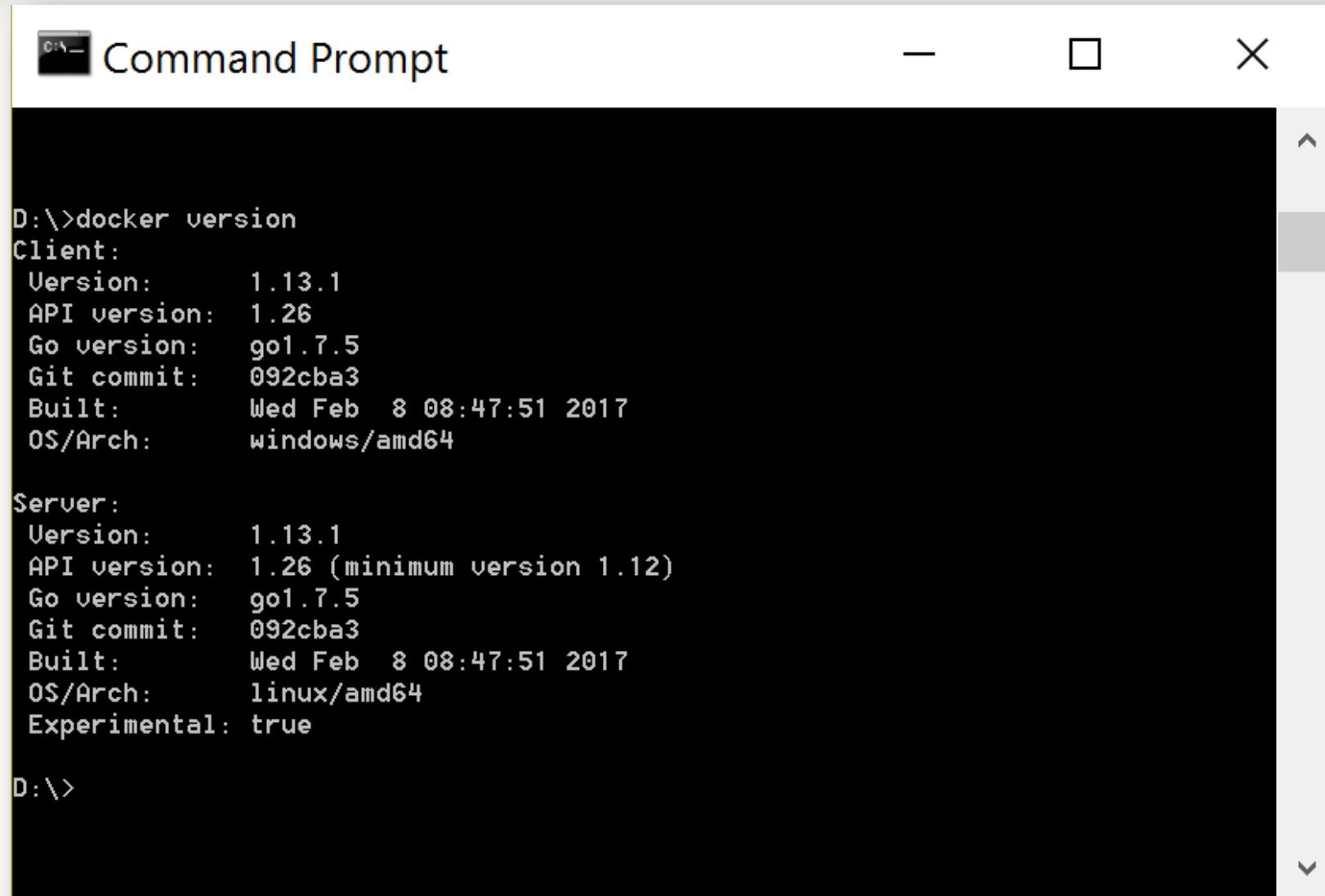
Open platform for developers and sysadmins to build, ship, and run distributed applications

Docker Images

 nginx	official	3.5K STARS	10M+ PULLS	 DETAILS
 busybox	official	737 STARS	10M+ PULLS	 DETAILS
 ubuntu	official	4.3K STARS	10M+ PULLS	 DETAILS
 redis	official	2.4K STARS	10M+ PULLS	 DETAILS
 docker	registry	950 STARS	10M+ PULLS	 DETAILS
 swarm	official	410 STARS	10M+ PULLS	 DETAILS
 mongo	official	2.1K STARS	10M+ PULLS	 DETAILS

<https://hub.docker.com/explore/>

Docker Version



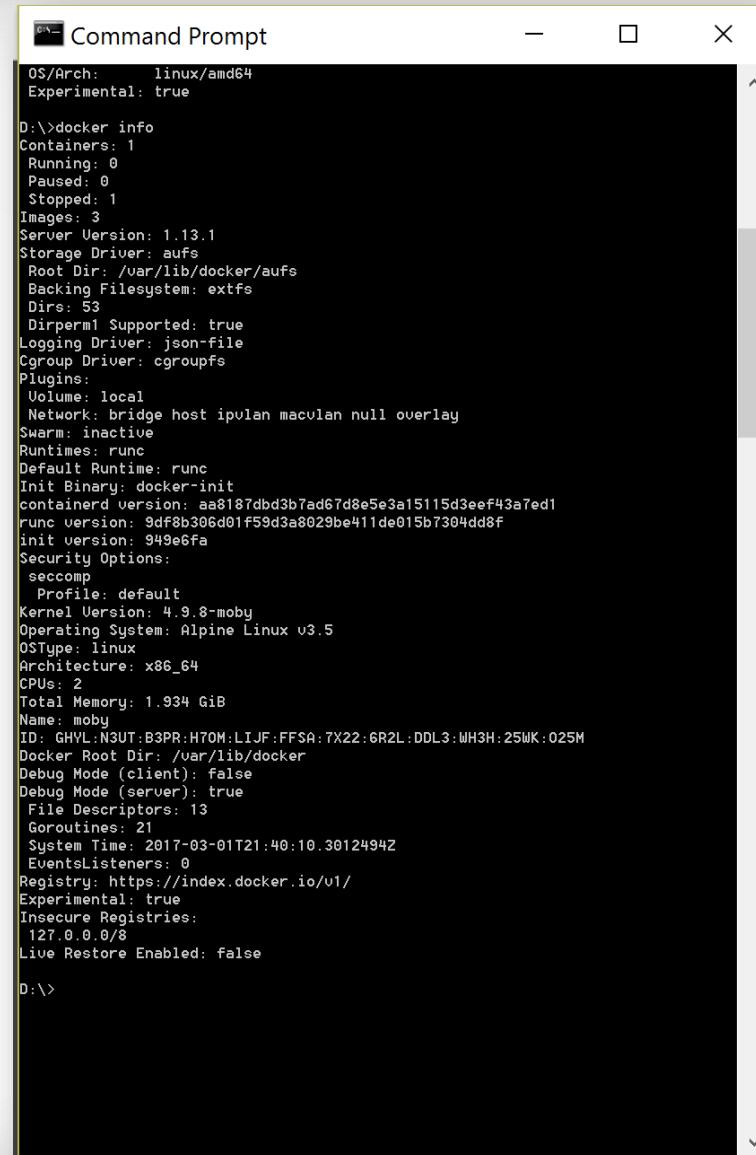
The screenshot shows a Windows Command Prompt window titled "Command Prompt". The window contains the output of the "docker version" command. The output is divided into Client and Server sections, both of which show the same version information: Version 1.13.1, API version 1.26, Go version go1.7.5, Git commit 092cba3, Built on Wed Feb 8 08:47:51 2017, and OS/Arch as windows/amd64 for the Client, and linux/amd64 for the Server. The Experimental field is set to true. The prompt at the bottom of the window is D:\>.

```
D:\>docker version
Client:
  Version: 1.13.1
  API version: 1.26
  Go version: go1.7.5
  Git commit: 092cba3
  Built: Wed Feb 8 08:47:51 2017
  OS/Arch: windows/amd64

Server:
  Version: 1.13.1
  API version: 1.26 (minimum version 1.12)
  Go version: go1.7.5
  Git commit: 092cba3
  Built: Wed Feb 8 08:47:51 2017
  OS/Arch: linux/amd64
  Experimental: true

D:\>
```

Docker Info



```
OS/Arch:      linux/amd64
Experimental: true

D:\>docker info
Containers: 1
Running: 0
Paused: 0
Stopped: 1
Images: 3
Server Version: 1.13.1
Storage Driver: aufs
  Root Dir: /var/lib/docker/aufs
  Backing Filesystem: extfs
  Dirs: 53
  Dirperm1 Supported: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
  Volume: local
  Network: bridge host ipplan maculan null overlay
Swarm: inactive
Runtimes: runc
Default Runtime: runc
Init Binary: docker-init
containerd version: aa8187dbd3b7ad67d8e5e3a15115d3eef43a7ed1
runc version: 9df8b306d01f59d3a8029be411de015b7304dd8f
init version: 949e6fa
Security Options:
  seccomp
    Profile: default
Kernel Version: 4.9.8-moby
Operating System: Alpine Linux v3.5
OSType: linux
Architecture: x86_64
CPUs: 2
Total Memory: 1.934 GiB
Name: moby
ID: GHVL:N3UT:B3PR:H70M:LIJF:FFSA:TX22:6R2L:DDL3:WH3H:25WK:025M
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): true
File Descriptors: 13
Goroutines: 21
System Time: 2017-03-01T21:40:10.3012494Z
EventsListeners: 0
Registry: https://index.docker.io/v1/
Experimental: true
Insecure Registries:
  127.0.0.0/8
Live Restore Enabled: false

D:\>
```

Docker Hello World

```
docker run -it hello-world  
Hello from Docker!
```

This message shows that your installation appears to be working correctly.

Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

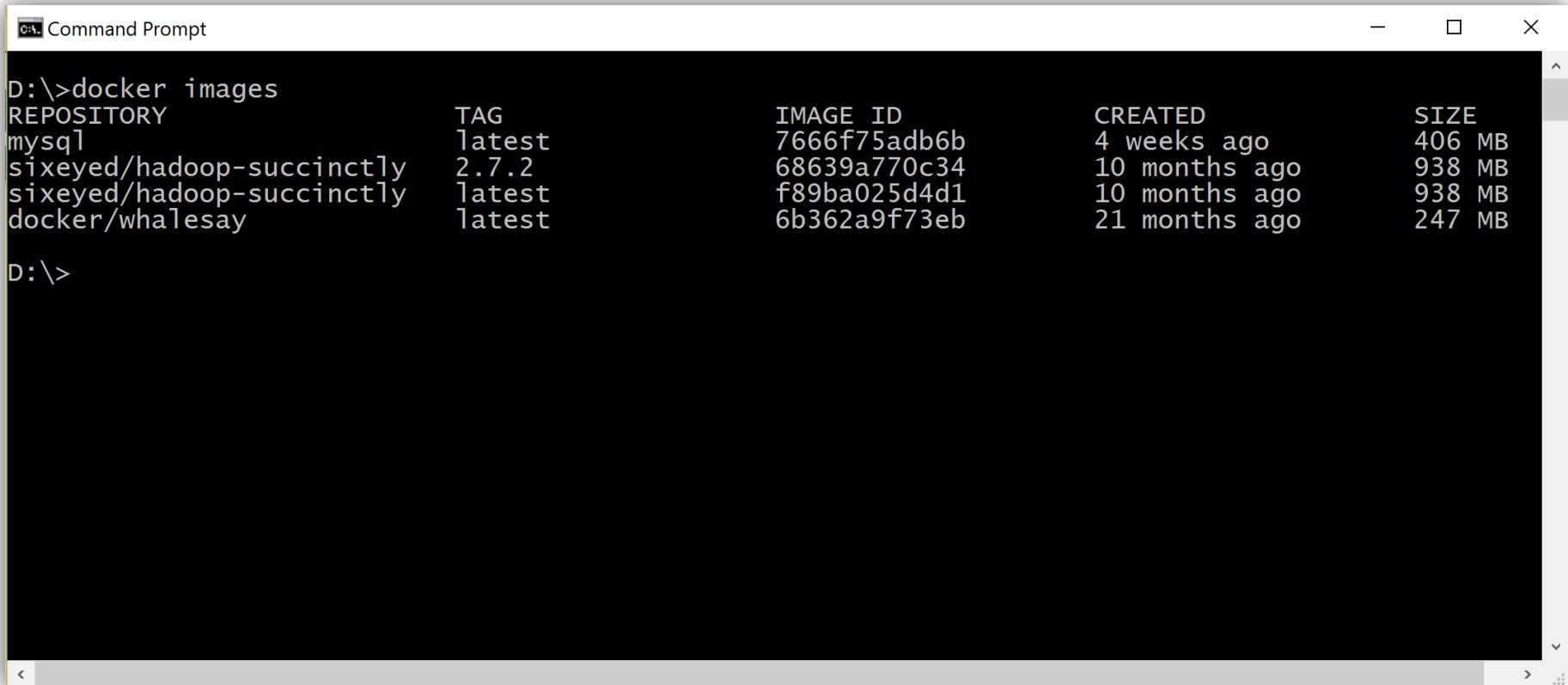
<https://docs.docker.com/engine/userguide/>

Docker Hello World

```
docker run docker/whalesay cowsay Hello world
```

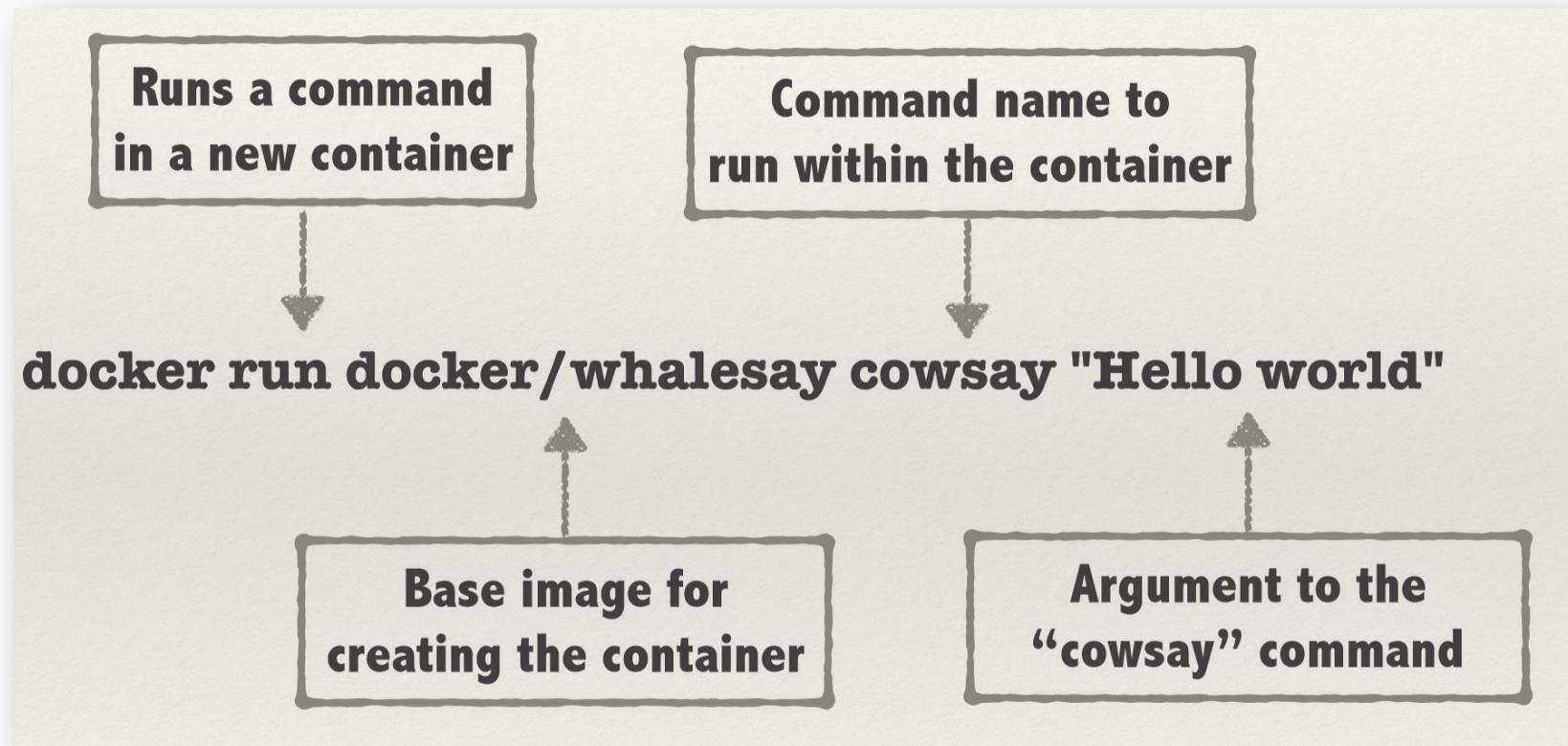
Docker Hello World

docker images



```
D:\>docker images
REPOSITORY          TAG        IMAGE ID      CREATED       SIZE
mysql               latest     7666f75adb6b   4 weeks ago   406 MB
sixeyed/hadoop-succinctly  2.7.2    68639a770c34   10 months ago  938 MB
sixeyed/hadoop-succinctly  latest    f89ba025d4d1   10 months ago  938 MB
docker/whalesay      latest     6b362a9f73eb   21 months ago  247 MB
D:\>
```

Docker Hello World



<http://www.westerndevs.com/docker/windows-docker/>

Docker Facts

`docker ps -a`

Show all containers on the system (without ‘-a’ only currently running).

An *image* is a filesystem and parameters to use at runtime.
It doesn’t have state and never changes.

A *container* is a running instance of an image.

Depending on how it was built, an image might run a simple, single command and then exit

https://docs.docker.com/engine/getstarted/step_two/

<http://www.westerndevs.com/docker/windows-docker/>

Build & Publish your Image

https://docs.docker.com/engine/getstarted/step_four/

<https://docs.docker.com/engine/userguide/engine-baseimages/#creating-a-simple-base-image-using-scratch>

docker build –t “imagename”

<https://hub.docker.com/r/microsoft/iis/>

docker tag

<https://stefanscherer.github.io/run-linux-and-windows-containers-on-windows-10/>

docker login

<https://blog.docker.com/2016/09/build-your-first-docker-windows-server-container/>

docker image remove



DEMO

<http://www.dwmkerr.com/learn-docker-by-building-a-microservice/>

APACHE
HBASE




CouchDB
relax


riak


mongoDB

HYPERTABLE INC


Neo4j



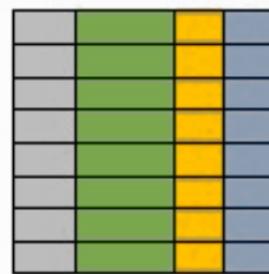
redis



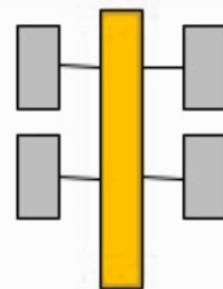
NoSQL DB's

DB Types

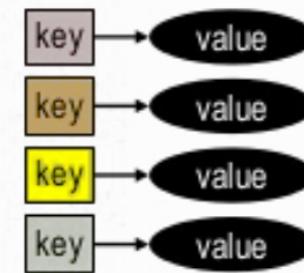
Relational



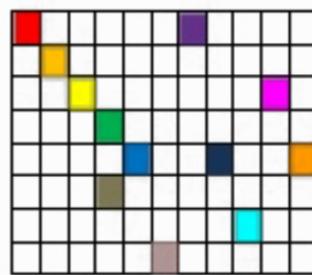
Analytical (OLAP)



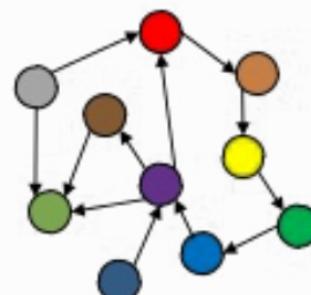
Key-Value



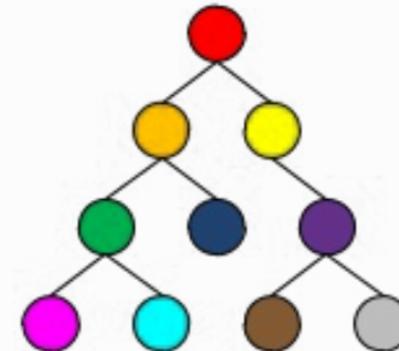
Column-Family



Graph

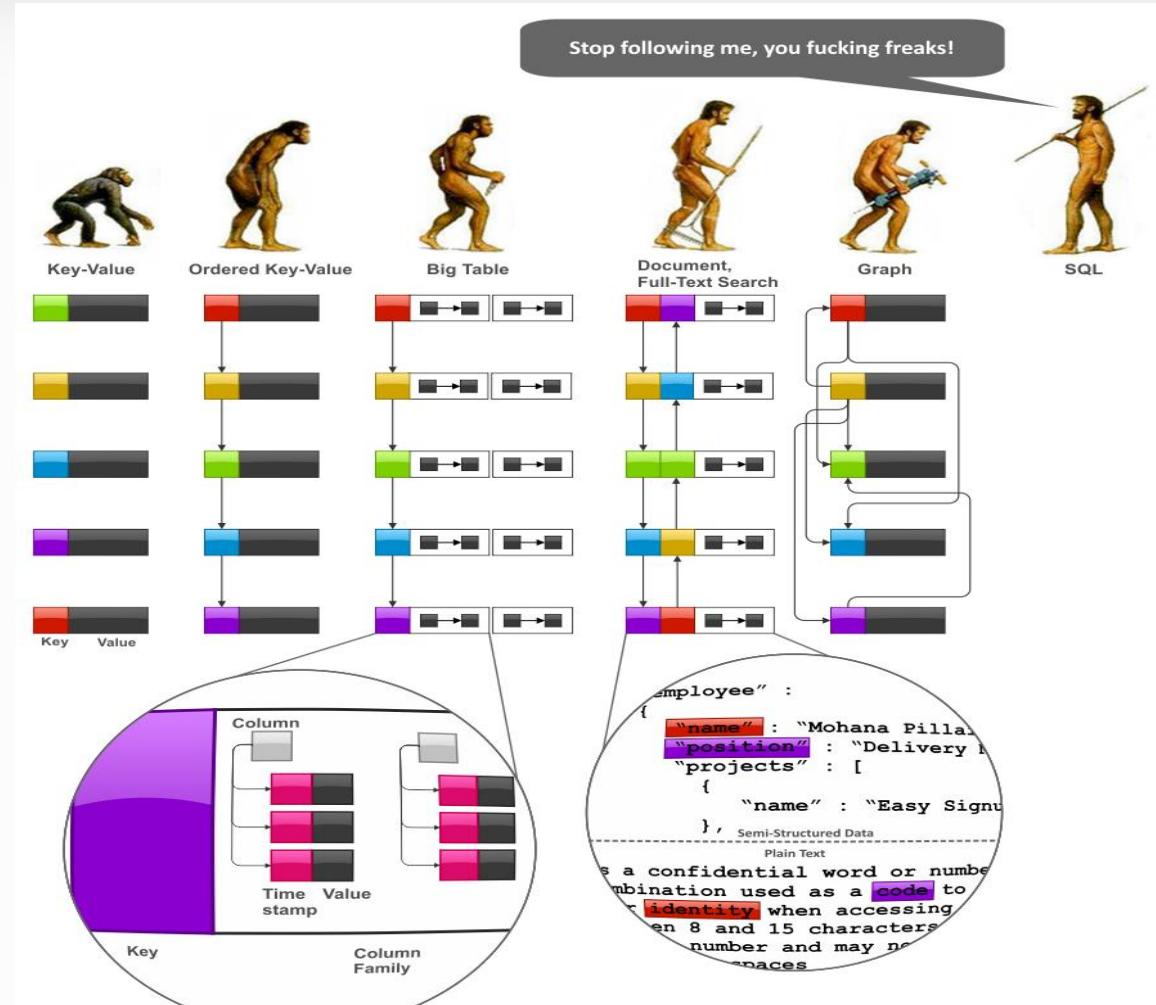


Document



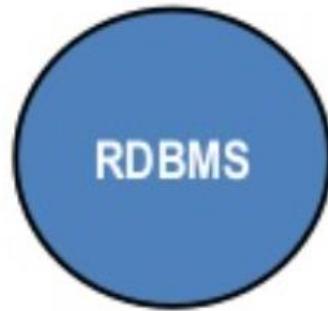
NoSQL DB's are often compared by *non-functional* criteria trade-offs ([CAP](#))

DB Types



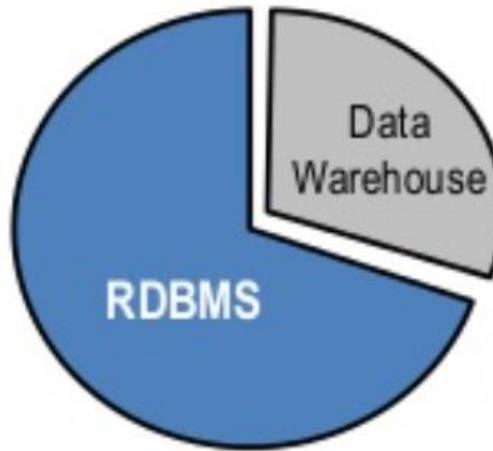
Alternative models tend to be less expressive than the relational one, with the exception of the graph model, which is actually more expressive.

DB Challenges



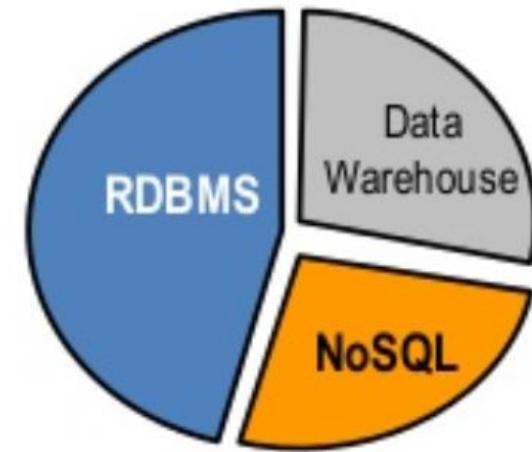
80' -90'

Manage
Transactions



2000'

Do
Analytics

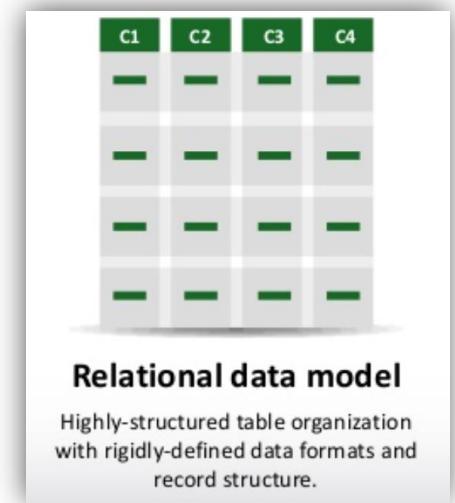


2010....

Cope with
Scalability

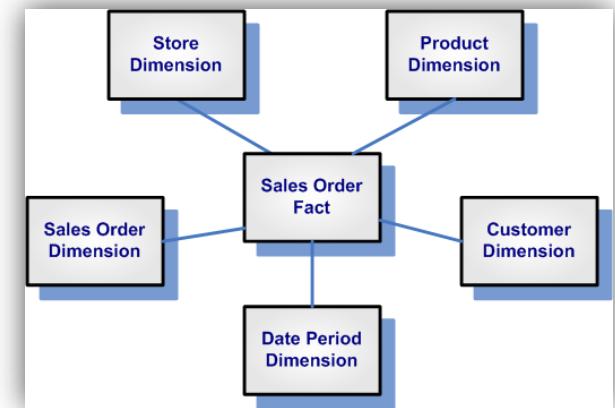
Relational DB

- Unit of Storage is a Row.
- Data Schema defined before storing data.
- Standard API (= SQL).
- Extensive use of Join Operations.
- Serves a “Relation” in a form of a Table.
- ACID Transactions.
- Not Scale Well.

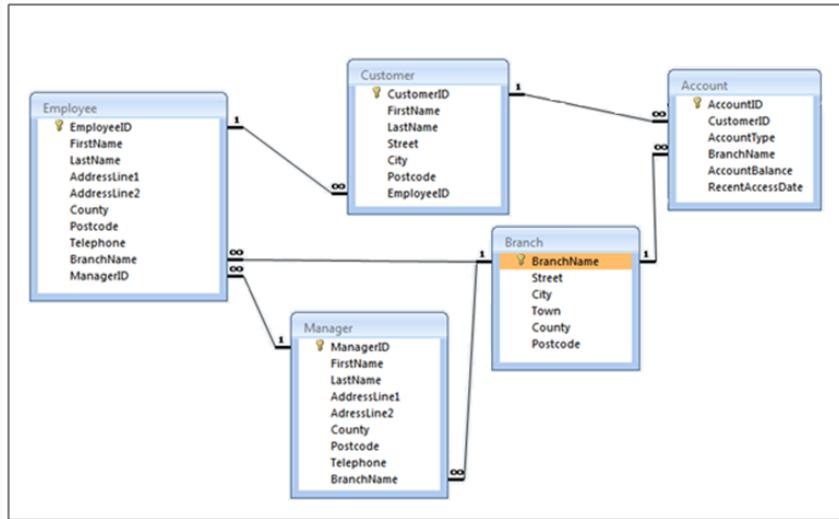


Analytical DB

- Based on Star Schema.
- Optimization set to fast Reading.
- MDX Query language.



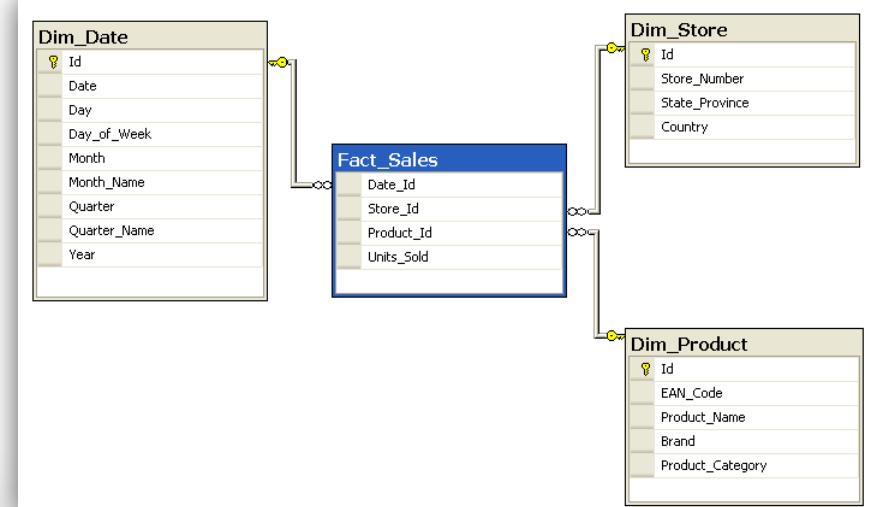
OLTP vs. OLAP



Data normalization is primarily important in the transactional, online transactional processing (OLTP) database world, in which data modifications (CRUD operations) occur rapidly and randomly throughout the stored data

In contrast, **DATA WAREHOUSE** contains large amount of denormalized and precalculated, summarized data—to avoid the performance penalty of ad hoc joins.

In a data warehouse, updates happen periodically under extremely controlled circumstances. End users' updates to data in data warehouses are uncommon



Distribution Tech Advances

MapReduce: Simplified Data Processing on Large Clusters

Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach,
Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber
chang,wilsonh,kerr,m3b,tushar,fikes,gruber}@google.com

Dynamo: Amazon's Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati,
Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall
and Werner Vogels

Amazon.com

For a quarter of a century, the relational database (RDBMS) has been
the **Dominant** model for database management

RDBMS Limitations

- RDBMS - Slow and Expensive.
- RDBMS - Scale up -> buy bigger machines.
- NoSQL - Scale out -> distributing DB across multiple hosts.
- When transaction rates & availability needs to go up and DB moves into cloud or virtualized environment, **Scaling out** on commodity hardware has **Significant economical advantages**.
- RDBMS storing massive volumes of data is constrained due to the poor capability to Scale, NoSQL systems such as Hadoop, handles “big data” much better.

- RDBMS Schemas lead to non flexible data models and change management is a big issue.
- NoSQL data models are more relaxed or even absent of restrictions at all:
 - Key-Value or Document DB can store virtually **ANY** structure.
 - BigTable DB like HBASE, while being more rigid, also allow adding new columns with minimum effort.

NoSQL Common Features

- Horizontal Scalability (= scaling out).
- Smart use of hashing & caching.
- Parallel execution of queries, “leading” the query to where data is being stored, not the other way around.
- Using Simple APIS (= REST)

Dealing with Consistency

In RDB we have ACID

- **Atomic** All operations in a transaction succeed or every operation is rolled back.
- **Consistent** On transaction completion, the database is structurally sound.
- **Isolated** Transactions do not contend with one another. Contentious access to state is moderated by the database so that transactions appear to run sequentially.
- **Durable** The results of applying a transaction are permanent, even in the presence of failures.

Dealing with Consistency

In NOSQL we have BASE

- **Basic availability** The store appears to work most of the time.
- **Soft-state** Stores don't have to be write-consistent, nor do different replicas have to be mutually consistent all the time.
- **Eventual consistency** Stores exhibit consistency at some later point (e.g., lazily at read time).

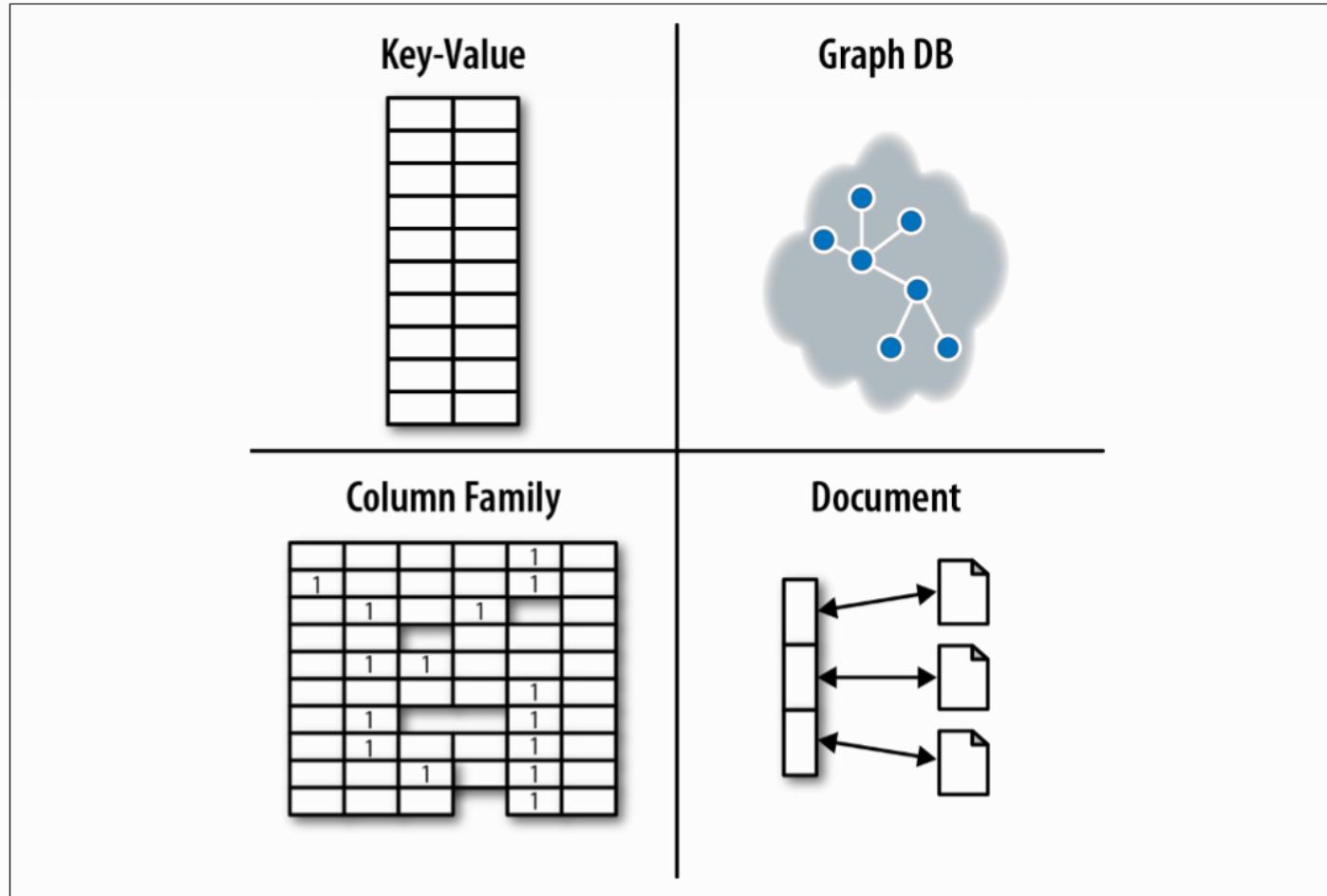
Riak consistency at read time

Datomic consistent at snapshots

In NOSQL world, ACID transactions have gone out of fashion as stores loosen the requirements for immediate consistency, data freshness, and accuracy in order to gain other benefits, like scale and resilience

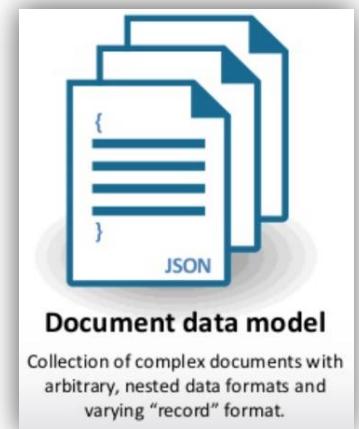
A BASE store values availability (A core building block for scale), but does not offer guaranteed consistency of replicas at write time.

NoSQL DB's

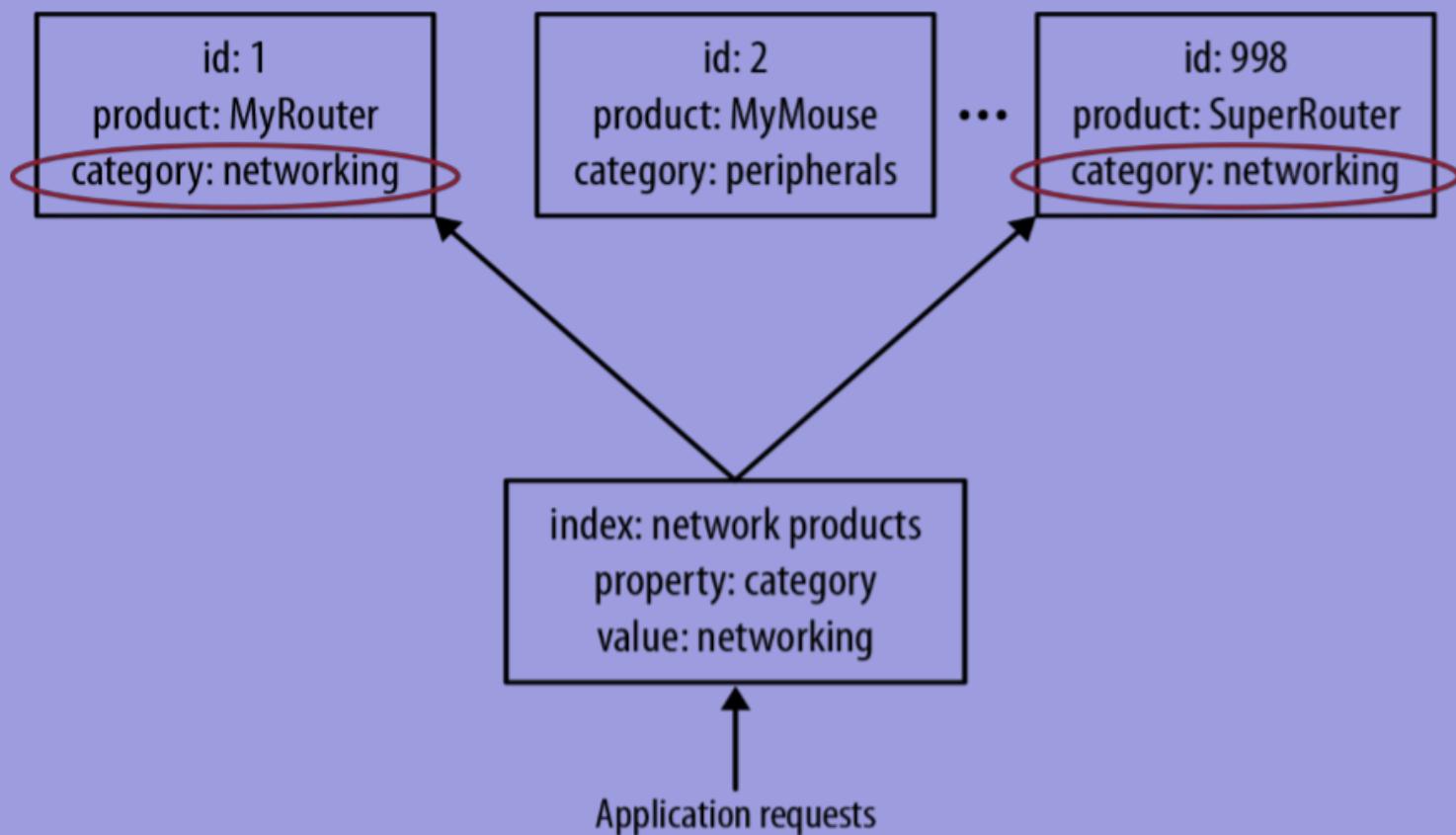


Document DB

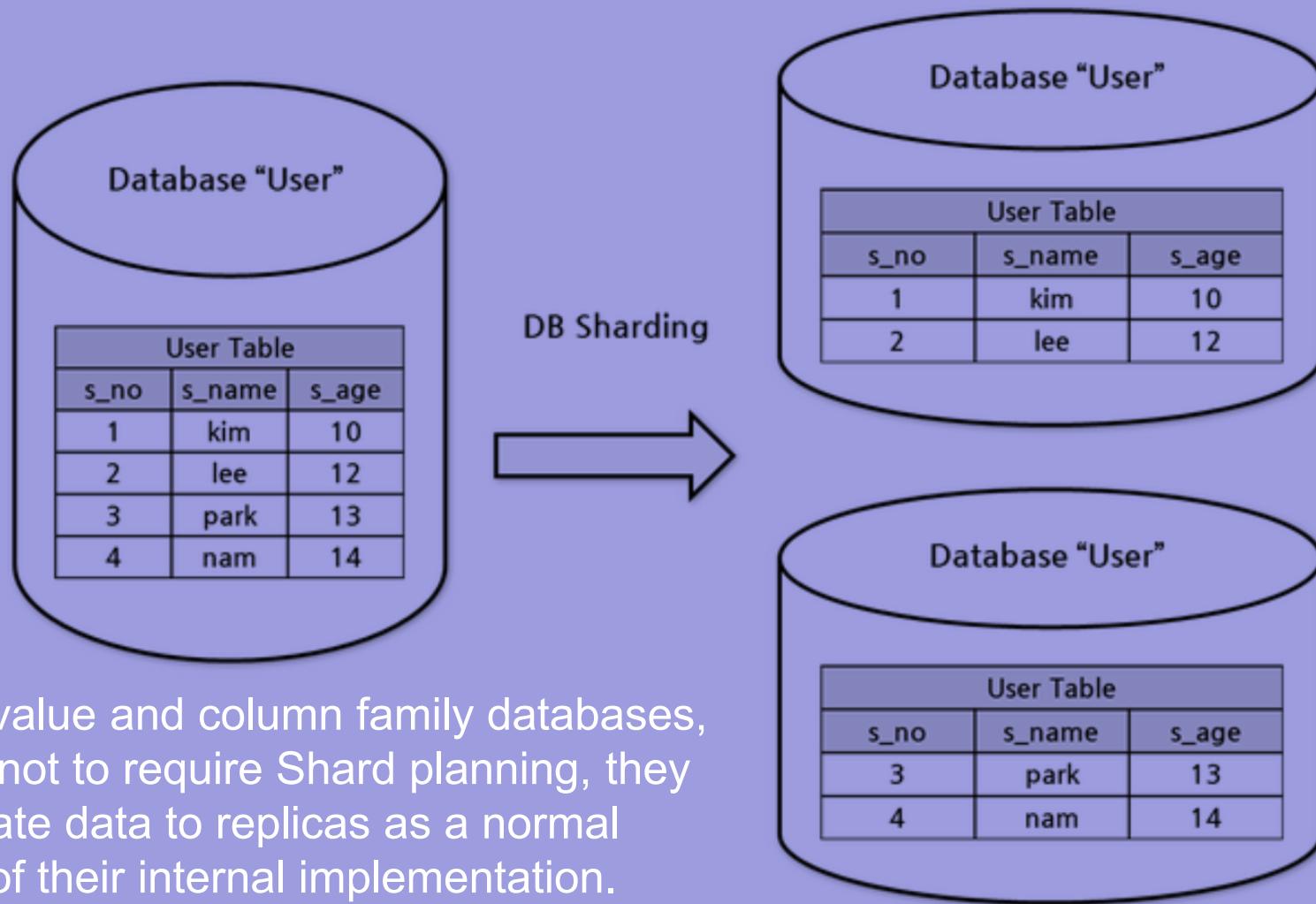
- Immediately familiar paradigm for developers used to working with hierarchically structured documents.
- Documents tend to comprise maps and lists, allowing for natural hierarchies - like JSON and XML formats.
- At simplest level, documents can be stored & retrieved by ID.
- Document store can act like a key-value store.
- In general, document stores rely on indexes to facilitate access based on document attributes.



Document DB



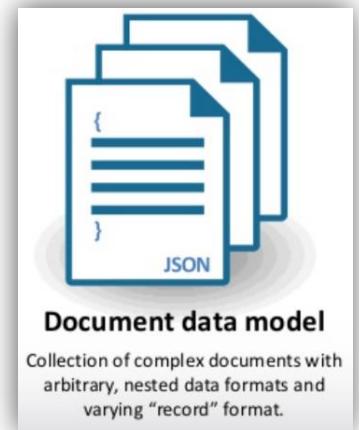
Document DB



Key-value and column family databases, tend not to require Shard planning, they allocate data to replicas as a normal part of their internal implementation.

Document DB

- Document DB ensure writes to a **single** document are atomic (administrator safe levels option). Support for operating across **sets** of documents atomically is emerging, but not matured yet.
- Documents are not connected (except through indexes), resulting numerous **optimistic concurrency** control mechanisms help reconcile concurrent contending writes for a single document without locks.
- In CouchDB documents held in a multimaster database ,automatically replicates, concurrently accessed, across instances without interference from users.





mongoDB®



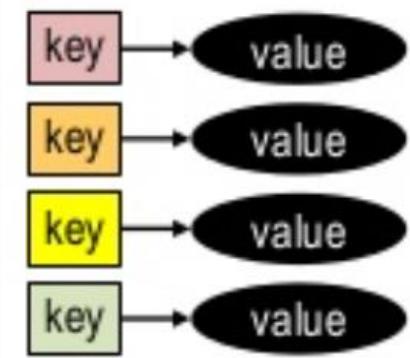
DEMO

<https://docs.mongodb.com/getting-started/node/introduction/>

<https://docs.mongodb.com/getting-started/node/import-data/>

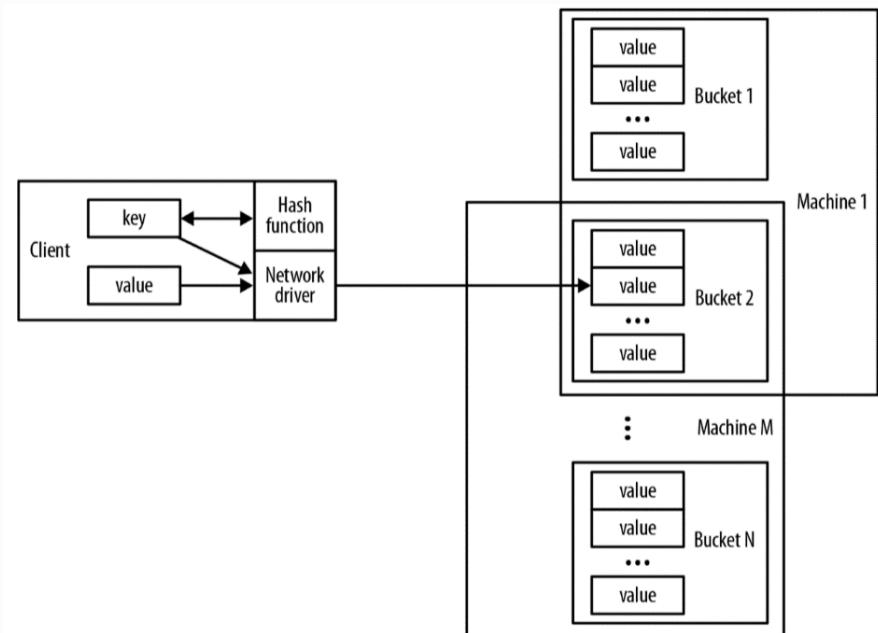
Key-Value

- Very simplistic, but very powerful model
- Key -> blobs/text/any data type.
- Scalable
- Simple API
- Query can not be based on value of content.
- Acts like large, distributed hash-map data structures that store and retrieve opaque values by key.



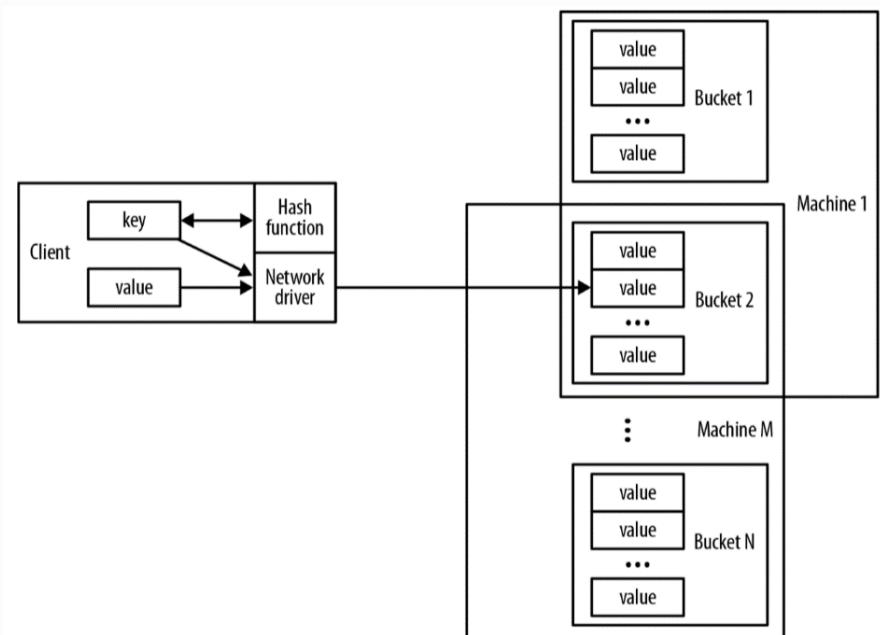
Key-Value

- Key space of the hash-map is spread across numerous buckets on the network.
- For fault-tolerance reasons each bucket is replicated on several machines.
- The formula for number of replicas required is given by $R = 2F + 1$, where F is the number of failures we can tolerate



Key-Value

- Key space of the hash-map is spread across numerous buckets on the network.
- For fault-tolerance reasons each bucket is replicated on several machines.
- The formula for number of replicas required is given by $R = 2F + 1$, where F is the number of failures we can tolerate



Key-Value

- Replication algorithm seeks to ensure that machines aren't exact copies of each other, allowing the system to load balance while a machine & buckets recover.
- From the client's point of view, key-value stores are easy to use. A client stores a data element by hashing a domain-specific identifier (key).
- Hash function is crafted such that it provides a uniform distribution across the available buckets, thereby ensuring that no single machine becomes a hotspot.

Key-Value

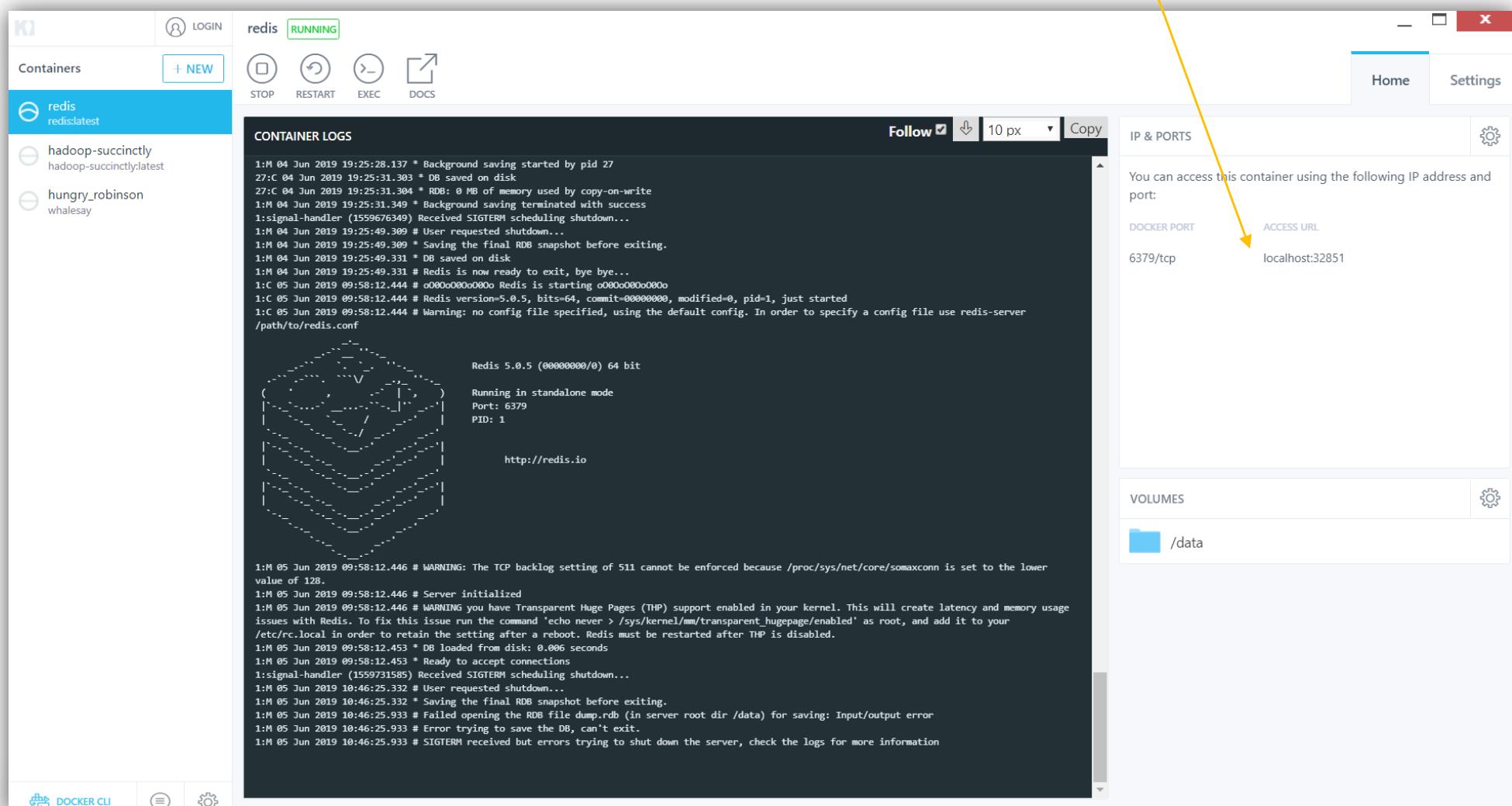
- The key-value data model is **similar** to the document data model. What differentiates them is the level of insight each offers into its data.
- In theory, key-value stores are oblivious to the information contained in their values. Some offer special tools for XML or JSON.
- To retrieve sets of useful information from across individual records, we typically use an external processing infrastructure, such as **MapReduce**.

Key-Value

- Key-value stores offer operational and scale advantages.
- Amazon's Dynamo database, designed for a nonstop shopping cart service, optimized for high availability and scale.
- Amazon team : “ they should work even if disks are failing, network routes are flapping, or data centers are being destroyed by tornados.”

redis with Node.js

Load redis Image to Docker Container, Notice Access Port



redis with Node.js

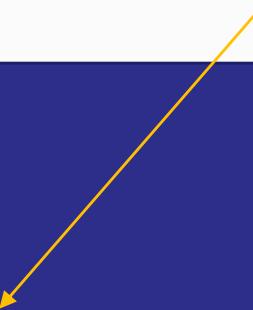
Notice Access Port

```
const path = require("path");
const express = require("express");

const redis = require("redis");
const app = express();

let PORT = 3000;

const client = redis.createClient(32850, "127.0.0.1");
// change port according to local settings
```



```
// -----
// emulate data storage
// -----
// redis-cli --scan --pattern '*'

client.set("111", "milk");
client.set("222", "hunny");
client.set("333", "bread");

// -----
// create app routes
// -----

app.get("/products/:key", function (req, res) {
  res.set('Content-Type', 'text/json');
  client.get(req.params.key, function (err, reply) {
    if (reply)
      res.send(reply);
    else
      res.send("not found");
  });
});

// -----
// Set application port
// -----

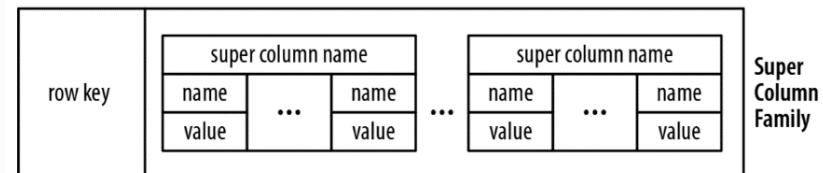
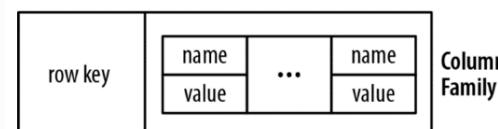
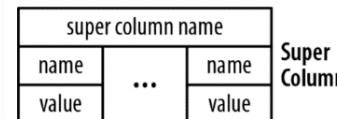
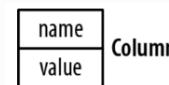
app.listen(PORT, function() {
  console.log(`App running on port ${PORT}`);
});
```

Resources:

<https://redis.io/commands>
<https://try.redis.io/>
https://github.com/NodeRedis/node_redis
<https://scotch.io/tutorials/how-to-optimize-node-requests-with-simple-caching-strategies>
<https://www.sitepoint.com/using-redis-node-js/>

Column Family

- Following Google's [BigTable](#).
- Data model based on sparsely populated table, whose rows can contain arbitrary columns, the keys for which provide natural indexing.



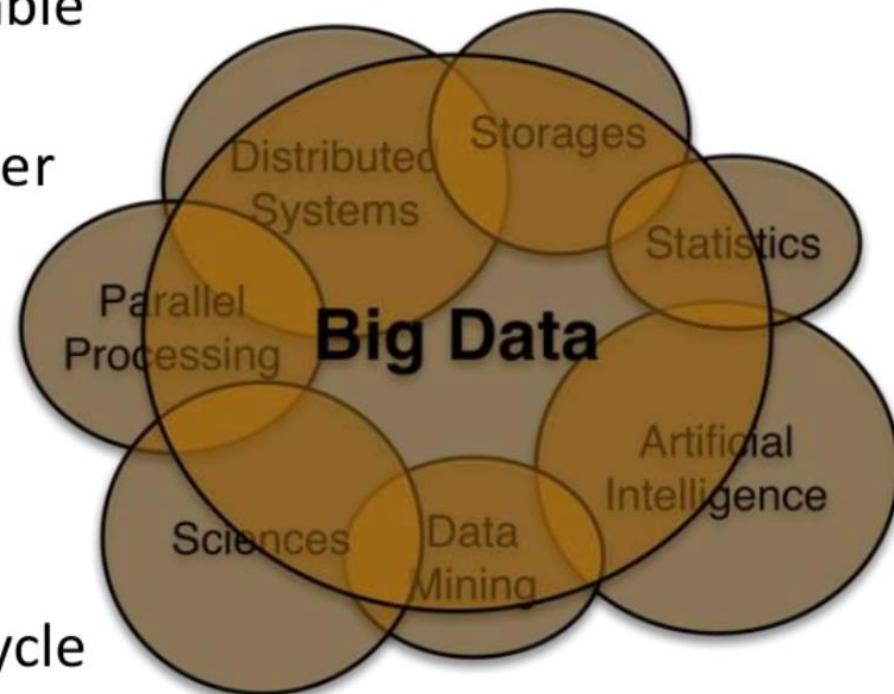


Big Data

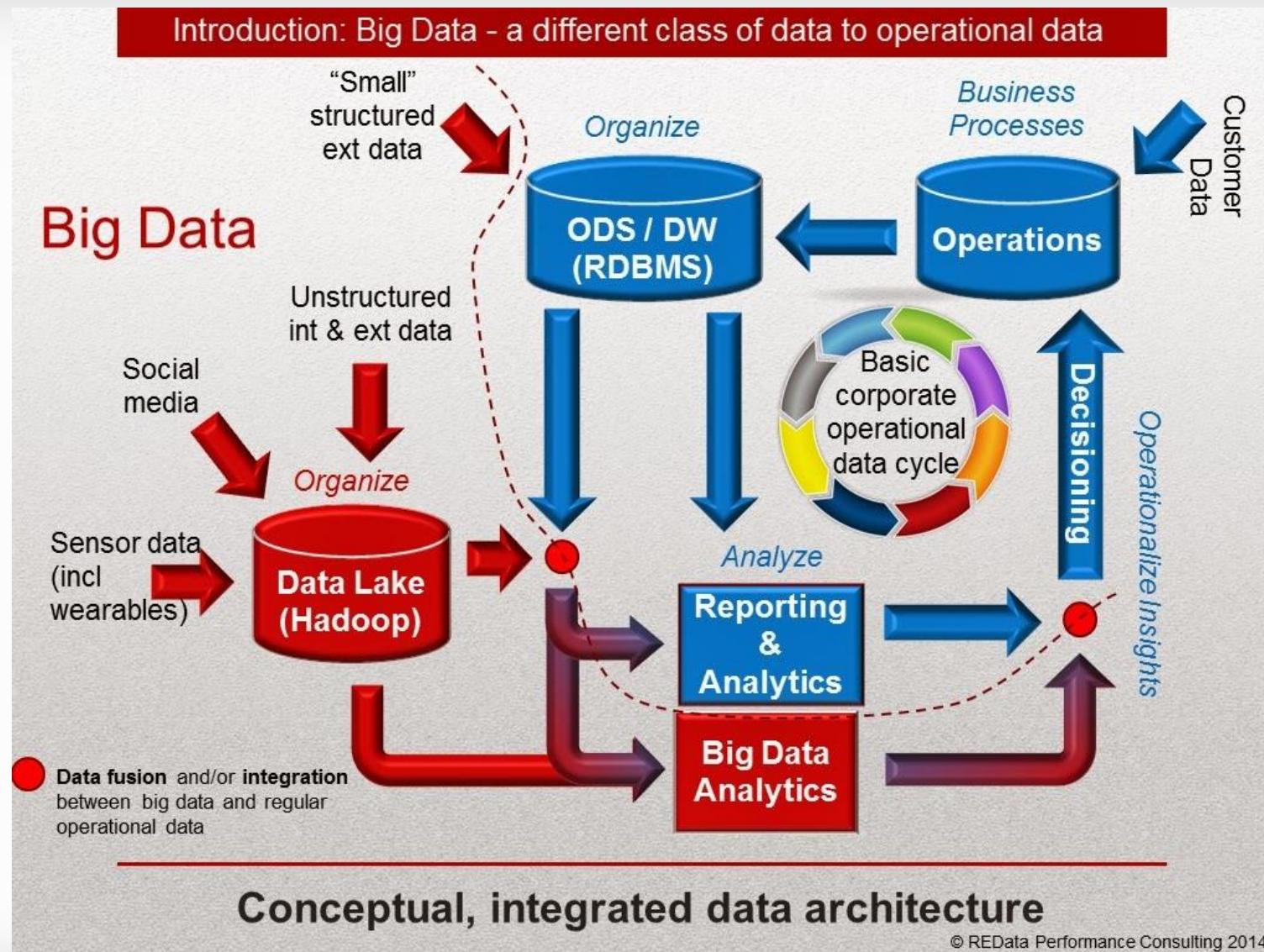
Define Big Data

What is Big data?

- There is lot of data available
 - E.g. Internet of things
- We have computing power
- We have technology
- Goal is same
 - To know
 - To Explain
 - To predict
- Challenge is the full lifecycle



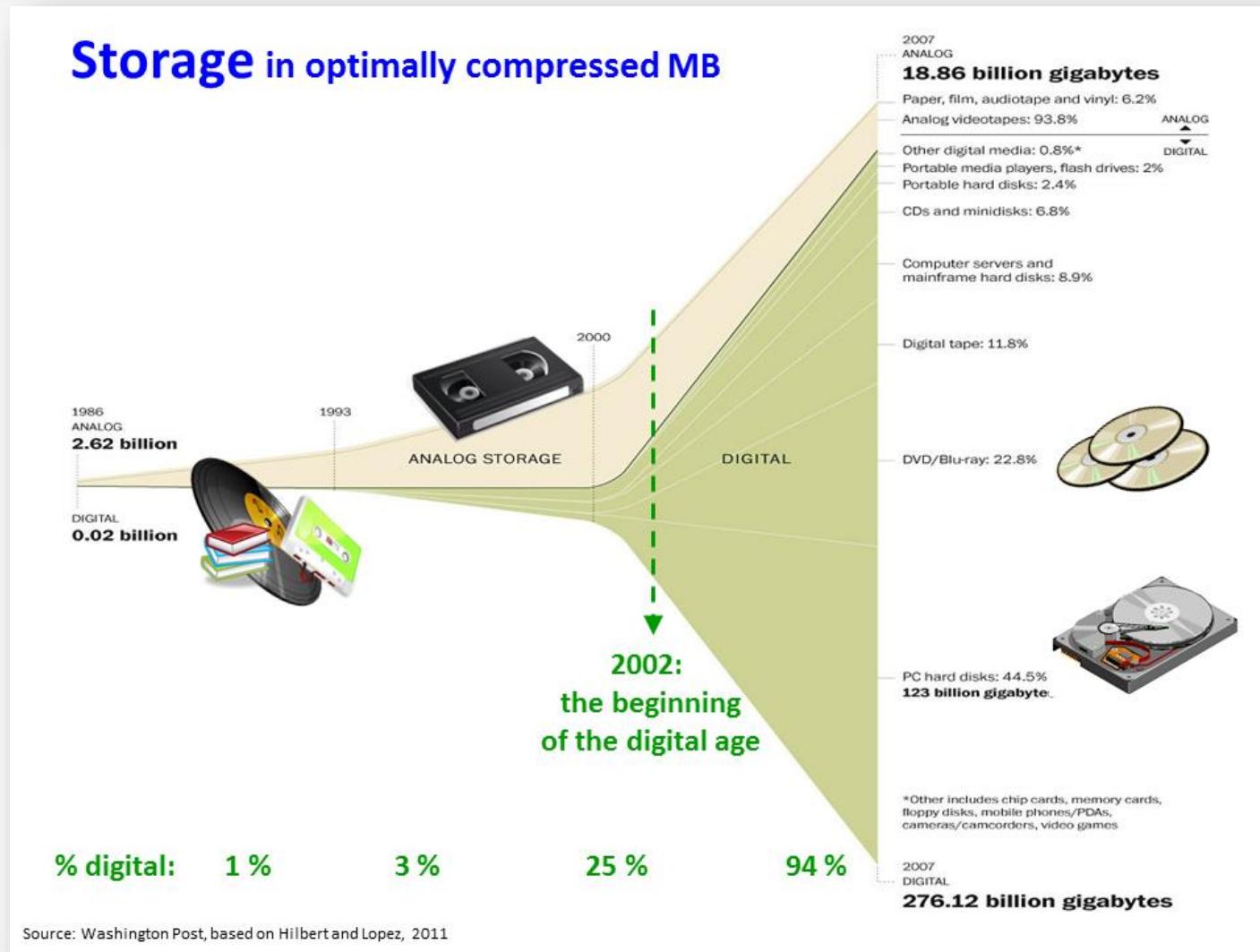
Define Big Data



הבדלים בין Big/Small Data

1. מטרה : נאוסף למטרה מוגדרת אחת לעומת אבולוציה מטרות עם הזמן
2. מקום : מקום אחד בפורמט אחד לעומת מבוזר על גבי מספר פלטפורמות ופורמטים.
3. מבניות : בד"כ מובנה לעומת מעורב.
4. אורך חיים: נאגר וROLLOUT למספר שנים לעומת מצבר לאורך זמן לא ידוע.
5. ייחידות מידת : אחיד ומוגדר לעומת רבגוני.
6. שחזור: ניתן ליצירה מחדש מוחודשת לעומת לא ניתן.
7. אינטראספקציה : נתונים המתארים עצםם בבהירות (מהו האובייקט נמדד, מה המימד ומהו הערך) לעומת מגוון, חוזרות, כפליות וחוסרים.
8. יכולת ניתוח : בביטחון אחד לעומת להcin, לצמצם, לפצל, להמיר.

אתגרי Big Data



אתגרי Big Data

How Much Data is Produced Every Day?



2.5 Exabytes are produced every day

Which is equivalent to:

- 🎵 530,000,000 millions songs
- 📱 150,000,000 iPhones
- 💻 5 million laptops
- 📚 250,000 Libraries of Congress
- ▶️ 90 years of HD Video

LEVEL

<https://www-01.ibm.com/software/data/bigdata/what-is-big-data.html>

Big Data

Where this Big Data originates, what makes the Big Data?

- sensors used to gather climate information
- posts to social media sites
- digital pictures and videos
- software logs, cameras
- microphones
- scans of government documents
- GPS trails
- purchase transaction records
- cell phone GPS signals
- traffic
- and many more.

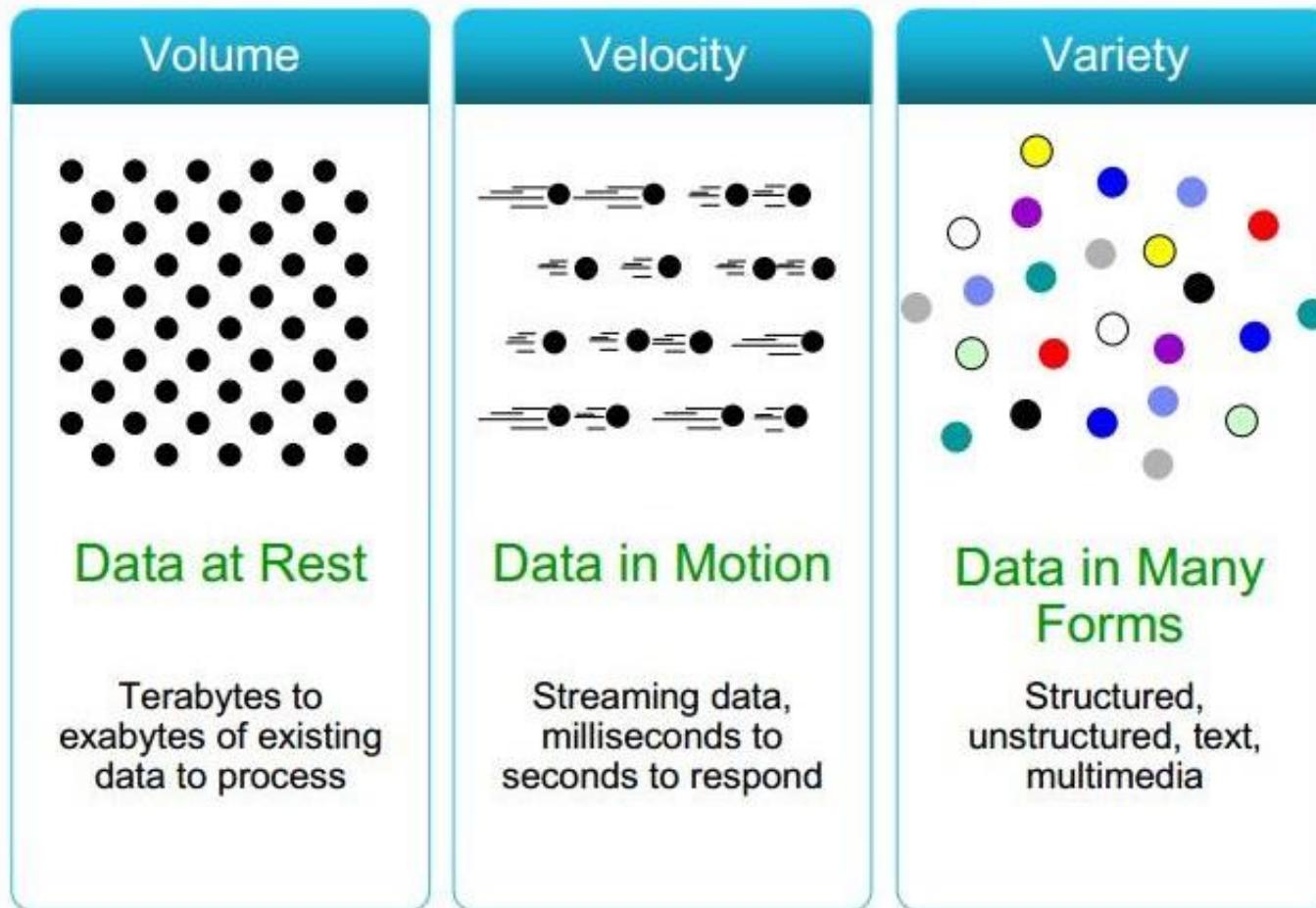
<http://saphanatutorial.com/hadoop-online-training-hadoop-basics-2-2/>

Big Data

- Big data **includes both structured and unstructured data**.
- If you look the current trend, Every day we create 2.5 quintillion bytes of data so much that **90% of the data in the world today has been created in the last two years alone**. Big data deals with data in petabyte and exabyte.
- **Big data is difficult to work with using most relational database management systems** and desktop statistics and visualization packages since it requires "massively parallel software running on tens, hundreds, or even thousands of servers".
- Big data is more than simply a matter of size; **it is an opportunity to find insights in new and emerging types of data and content**, to make your business more agile, and to answer questions that were previously considered beyond your reach.

Big Data

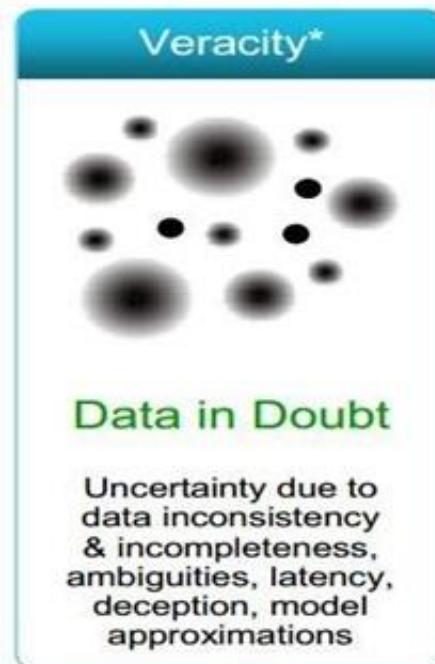
2001, Doug Laney (currently with Gartner) articulated the now mainstream definition of big data as the 3V's of big data



Big Data

There is a 4th challenge - veracity

There is one common problem to trust on the data, data accuracy and data sources



Big Data – What Changed

- Unlocking Value of ALL Data
 - Big Data: Decisions based on all your data
 - Big Data: Decisions based on all your data Video and Images
 - Machine-Generated Data
 - Social Data
 - Documents
- Traditional Architecture: Decisions based on database data from Transactions

Do you have big Data ?

- Volume: how much data do you have?
- Velocity: how quickly do you receive data?
- Variety: does your data come in different forms?
- Veracity: can you trust the content and context of your data?

Having a large amount of data doesn't necessarily mean you have Big Data. If you have 1 TB of log files but the log entries are all in the same format and you're only adding new files at the rate of 1 GB per month, Hadoop might not be a good fit. With careful planning, you could store all that in an SQL database and have real-time query access without the additional overhead of Hadoop.

- Volume



- Velocity



- Variety



- Veracity



Big Data

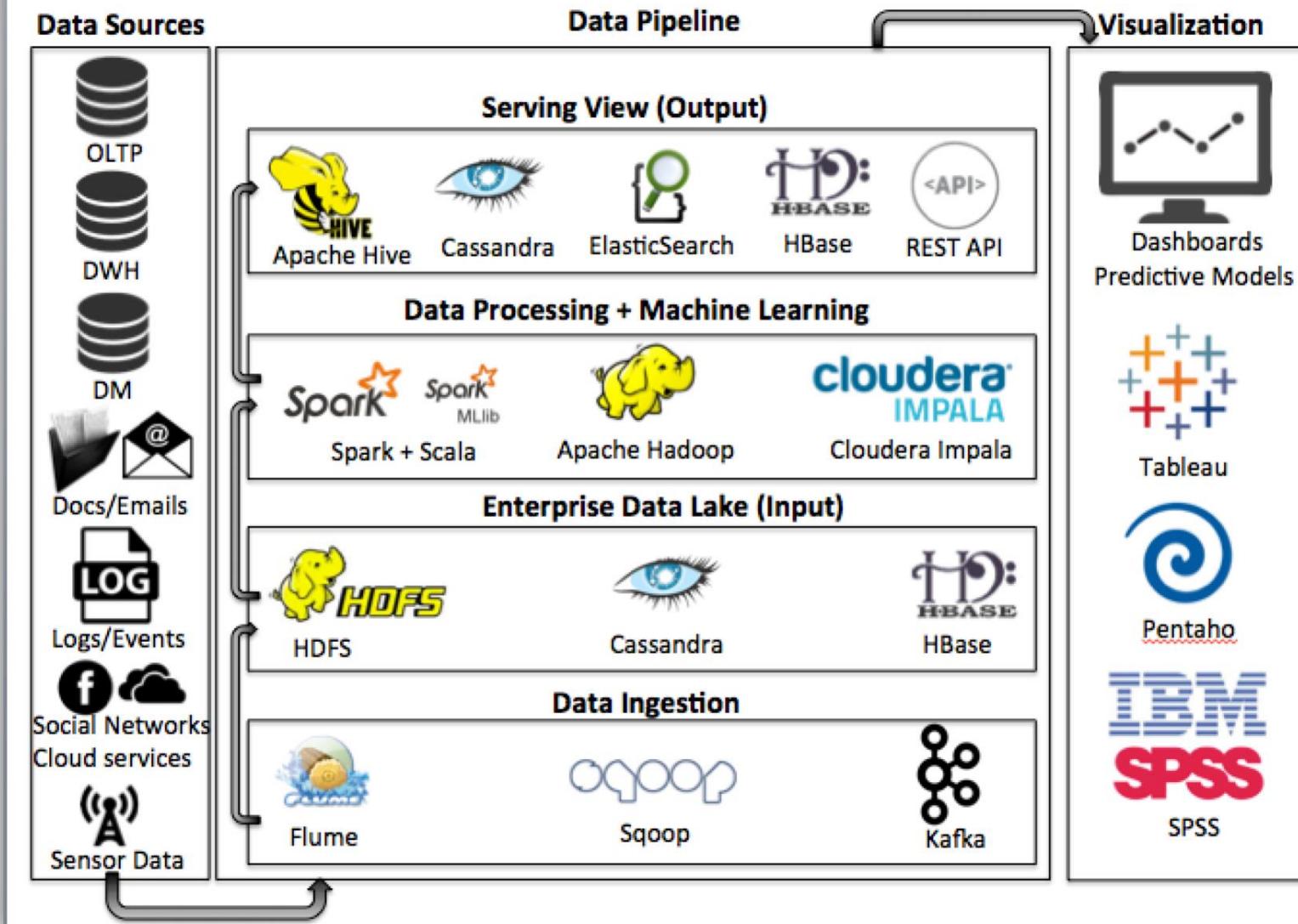
Big Data overwhelms traditional databases, storage and more

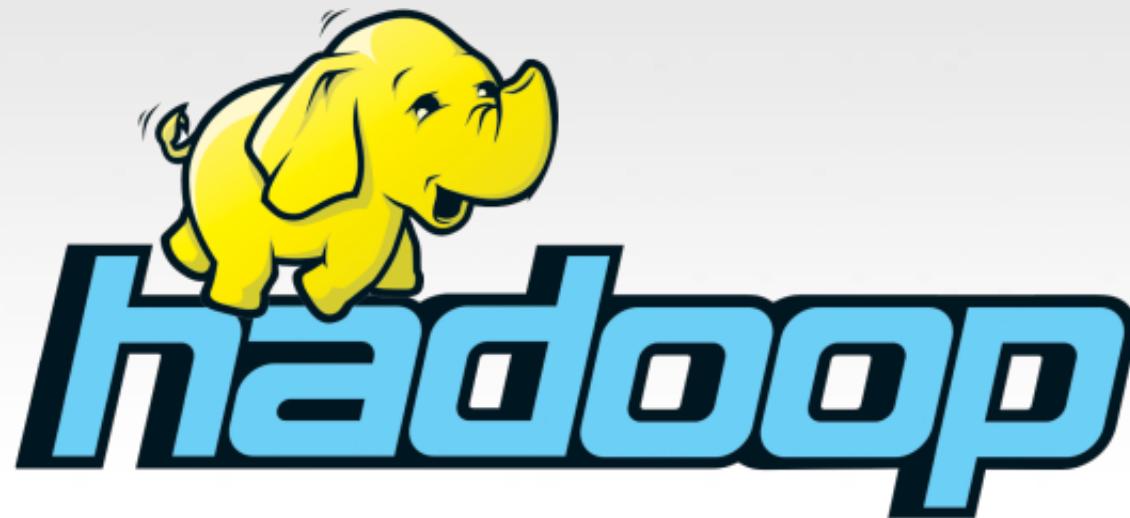
The other approach would be to use non-relational data store - **HADOOP**

To avoid single device bottlenecks, that they inhibit scaling, Hadoop uses map-reduce to spread analytical processing across armies of commodity servers.

Data Pipeline

Modern data pipeline architecture





Hadoop

What is Hadoop

Hadoop is an open-source framework to store and process Big Data in a distributed environment. It contains two modules, one is MapReduce and another is Hadoop Distributed File System (HDFS).

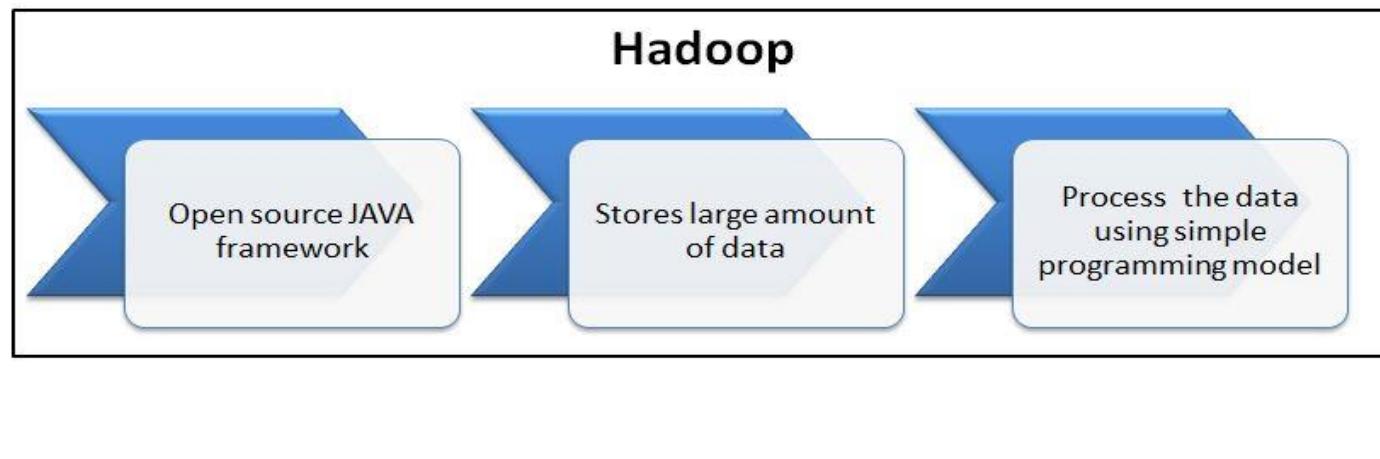
- **MapReduce:** It is a parallel programming model for processing large amounts of structured, semi-structured, and unstructured data on large clusters of commodity hardware – Batch processing in nature.
- **HDFS:** Hadoop Distributed File System is a part of Hadoop framework, used to store and process the datasets. It provides a fault-tolerant file system to run on commodity hardware

<https://www.tutorialspoint.com/hadoop/index.htm>

What is Hadoop

Apache Hadoop is an open source software platform managed by the Apache Software Foundation has proven to be very helpful in storing and managing vast amounts of data cheaply and efficiently. Hadoop is an open source JAVA framework.

- Which allows you to store large amount of data on clusters of low cost commodity hardware.
- Provides you the capability to process that distributed data using simple programming model.



What is Hadoop

Hadoop is NOT a database:

Hadoop is an efficient distributed file system and not a database. It is designed specifically for information that comes in many forms, such as social media, server log files or other documents. Anything that can be stored as a file can be placed in a Hadoop repository.

Hadoop is a Big Data platform with two functions:

1. Storing huge amounts of data in safe reliable Storage.
2. Running complex queries over that data in an efficient way.

Both storage and compute run on the same set of servers in Hadoop, and both functions are fault tolerant, which means you can build a high-performance cluster from commodity hardware

What is Apache Hadoop

Hadoop's key feature is that it just ***keeps scaling***. As your data needs grow, you can expand your cluster by adding more servers.

The same query will run in the exact same way on a cluster with 1,000 nodes as it will on a cluster with five nodes (only much faster).

Hadoop distributes ***storage*** on one hand and distributes ***compute*** on the other.

Hadoop is at the core of a whole host of other Big Data tools:

- HBase for real-time Big Data.
- Hive for querying Big Data using an SQL-like language.
- Oozie for orchestrating and scheduling work.

Origin of Apache Hadoop

The origin of Apache Hadoop Projects is from Google [White paper](#) series on [BigTable](#), [MapReduce](#) and GFS.

Google has written white papers on these topics but the code and implementation was never done.

Later on Yahoo and many other contributors implemented Google's white paper and that's how Hadoop framework developed.

Doug Cutting, Hadoop's creator, named the framework after his child's stuffed toy elephant.



Few Facts about Hadoop

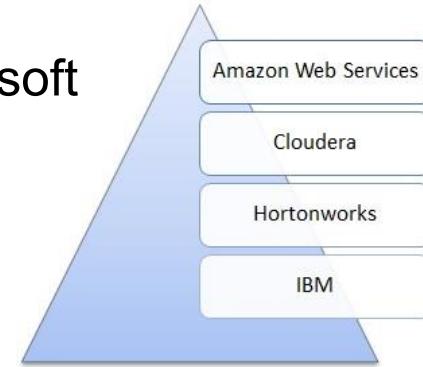
- Hadoop is top level Apache project initiated and led by YAHOO
- Hadoop is one of the most popular environment to work with Big data and to solve Big data problems.
- Hadoop is not a database but a file system, yetIt is not just a distributed file system but a complete open source framework.
- Hadoop core component are HDFS(Hadoop Distributed File System) and Map Reduce.
- The Hadoop framework itself is mostly written in the Java programming language, with some native code in C and command line utilities written as shell-scripts
- All the modules in Hadoop are designed with a fundamental assumption that hardware failures (of individual machines, or racks of machines) are common and thus should be automatically handled in software by the framework.
- A standard 10 node cluster can have a 1 TB of RAM total and do the entire processing in memory (distributed) using MapReduce2/YARN
- Job=Application
- Tasks= Map & Reduce



Few Facts about Hadoop

Many companies from IBM to Amazon Web Services, Microsoft and Teradata have packaged Hadoop into more easily-consumable distributions or services.

Each company has their own strategies but the key differentiator for all of these is that Hadoop has the ability to distribute workloads across potentially thousands of servers,
" Making big data manageable data "



Few Facts about Hadoop

1. Economical (Cost Effective):

1. No license
2. It's an open source framework ,
3. If you buy it from Cloudera, Hortonworks., Mapaa, which are different distributions of Hadoop then also its totally free.



2. Flexible:

1. Hadoop is schema-less, and can absorb any type of data, structured or not, from any number of sources.
2. Data from multiple sources can be joined enabling deeper analyses than any one system can provide.

3. Scalable:

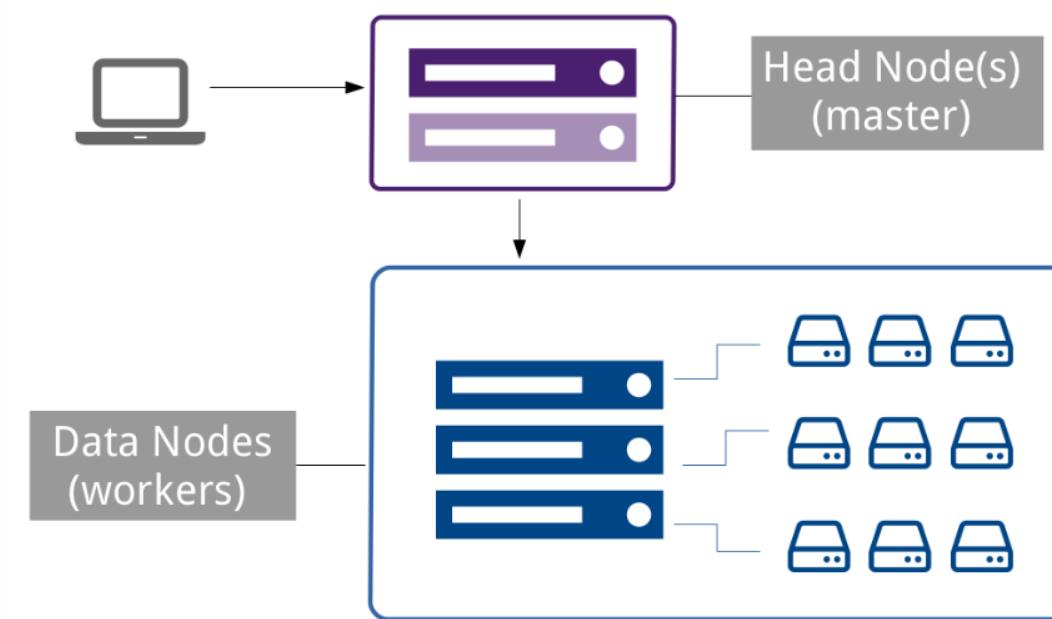
1. New nodes can be added without needing to change data formats, how data is loaded, how jobs are written, or the applications on top.
2. You can add as many number of nodes as you want depending upon the data size and business requirement.
3. Can process really large data (petabytes)



How Hadoop Works

Hadoop is a distributed technology

Hadoop cluster is a classic master/worker architecture,
in which the clients primarily make contact with the master.



How Hadoop Works

The master knows where the data is distributed among the Worker nodes.

The master coordinates queries, splitting them into multiple tasks among the Worker nodes.

The worker nodes store the data and execute tasks sent to them by the master.

When storing a large file in Hadoop, the file gets split into many pieces, called blocks, and the blocks are shared among the slaves in the cluster.

By default, three copies of each block are stored =>1GB file on your cluster is split into eight 128 MB blocks, each replicated on three nodes.



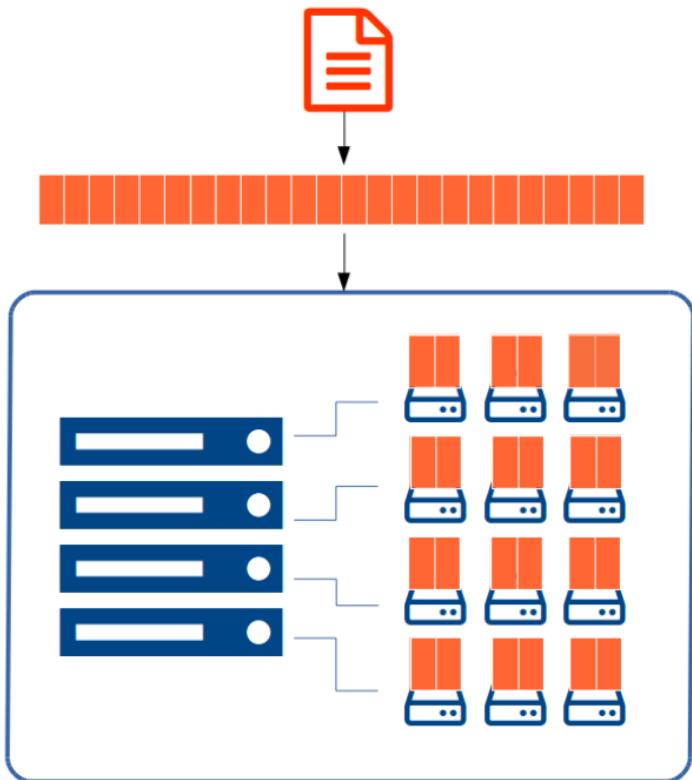
How Hadoop Works

1 GB file be stored as 24 blocks of 128 MB, in a cluster with four data nodes, each node will Store six blocks across its disks.

Massively Parallel Processing (MPP) - split a large job into many small tasks Running concurrently, each using it's local data.

Query over 1 TB data, split into 256 MB blocks. Hadoop splits it into 4,096 .each take About 90 seconds.

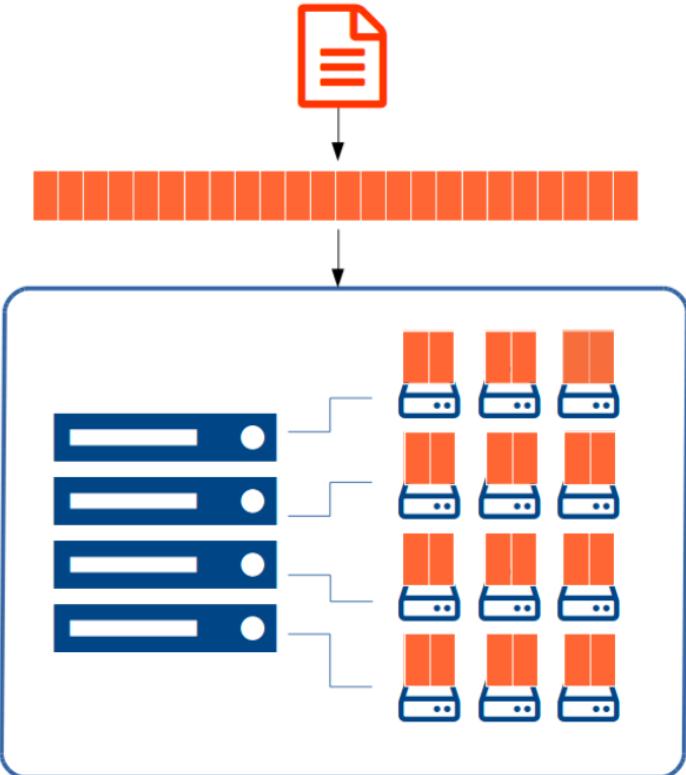
Data Nodes	Total CPU Cores	Total Concurrent Tasks	Best Compute Time
1	1	1	100 hours
12	192	160	38 minutes
50	1200	1000	6 minutes



How Hadoop Works

The more nodes you add, the more likely it is that a node will be allocated a task for Which it does not locally Store the data=> average task completion time going up.

Nodes will be forced to read data from other nodes across the network. Hadoop minimizes the impact of that, with 4 YARN ***Yet Another Resource Negotiator***.



How Hadoop Works

Two parts to Hadoop:

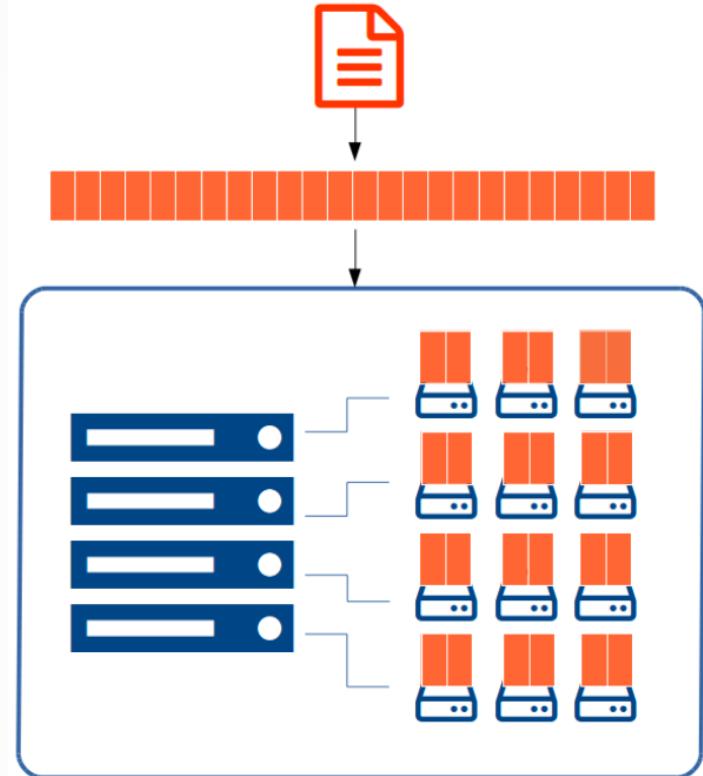
- 1) Distributed file system
- 2) Intelligent job scheduler.

The two Work together to provide high availability at storage level and high parallelism at compute level, enabling high- performance Big Data processing.

Hadoop must understand the nature of the work order to split & distribute it.

It means that in order to query Hadoop, you will need to use set patterns and a specific programming framework.

The main pattern is called **MapReduce** ,and map/reduce queries are typically written in Java.



Setting up Hadoop

Tools in the Hadoop ecosystem Can run in three modes:

1. Local.
2. Pseudo –distributed : Best for exploration.
3. Distributed.

```
docker run -it --rm -p 50070:50070 -p 50010:50010 -p 8088:8088 -p  
8042:8042 -p 19888:19888 -h hadoop --name hadoop sixeyed/hadoop-  
succinctly:2.7.2
```

Search : hadoop-succinctly:2.7.2



Setting up Hadoop

```
Command Prompt - docker run -it --rm -p 50070:50070 -p 50010:5... - Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\>docker run -it --rm -p 50070:50070 -p 50010:50010 -p 8088:8088 -p 8042:8042 -p 19888:19888 -h hadoop --name hadoop sixeyed/hadoop-succinctly:2.7.2

Unable to find image 'sixeyed/hadoop-succinctly:2.7.2' locally
2.7.2: Pulling from sixeyed/hadoop-succinctly
efd26ecc9548: Already exists
a3ed95cae02: Already exists
d1784d73276e: Already exists
72e581645fc3: Already exists
1e6509b4af69: Already exists
a3283d54b740: Already exists
5308fe3d04fe: Already exists
e25dbeb649151: Pull complete
377be167a613: Pull complete
29cdf8493072: Pull complete
74f93e2fc4b3: Pull complete
e57065caf5c8: Pull complete
497diad0309b: Pull complete
dabb3de7070a: Pull complete
b74d5bfa4c8b: Pull complete
cae603f1369c: Pull complete
1e3dc6fe89e6: Pull complete
94f8f060d117: Pull complete
7281ad082554: Pull complete
cecb92f4fee6c: Pull complete
38aefeaef5277: Pull complete
Digest: sha256:cdb84ebf4ae8d8cbccf272f026d5b74476ca6aaa34462ed4b78344380937859
Status: Downloaded newer image for sixeyed/hadoop-succinctly:2.7.2
[ ok ] Starting OpenBSD Secure Shell server: sshd.
/opt/hadoop/bin/start-servers.sh: line 5: /opt/hadoop/etc/hadoop/hadoop-env.sh: Permission denied
Starting namenodes on [localhost]
localhost: Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.
localhost: starting namenode, logging to /opt/hadoop-2.7.2/logs/hadoop-root-name
node-hadoop.out
localhost: Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.
localhost: starting datanode, logging to /opt/hadoop-2.7.2/logs/hadoop-root-data
node-hadoop.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: Warning: Permanently added '0.0.0.0' (ECDSA) to the list of known hosts .
0.0.0.0: starting secondarynamenode, logging to /opt/hadoop-2.7.2/logs/hadoop-ro
ot-secondarynamenode-hadoop.out
starting yarn daemons
starting resourcemanager, logging to /opt/hadoop-2.7.2/logs/yarn--resourcemanag
er-hadoop.out
localhost: Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts .
localhost: starting nodemanager, logging to /opt/hadoop-2.7.2/logs/yarn-root-nod
emanager-hadoop.out
root@hadoop:/hadoop-setup#
```



Testing Hadoop

```
hadoop fs -mkdir -p /user/root/input
```

```
hadoop fs -put $HADOOP_HOME/etc/hadoop input
```

```
hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.2.jar grep input output 'dfs[a-z.]+'
```

Load Hadoop

Select “exe” from kitematic

```
hadoop fs -mkdir -p /user/input
```

Run node.js to copy file

```
View using http://localhost:32806/explorer.html#/
```



MapReduce

MapReduce can be used in a variety of ways to bring efficiency into the ways you are processing data.

We can leverage MapReduce in multiple ways and derive deep insights through queries that were previously difficult or impossible to express.

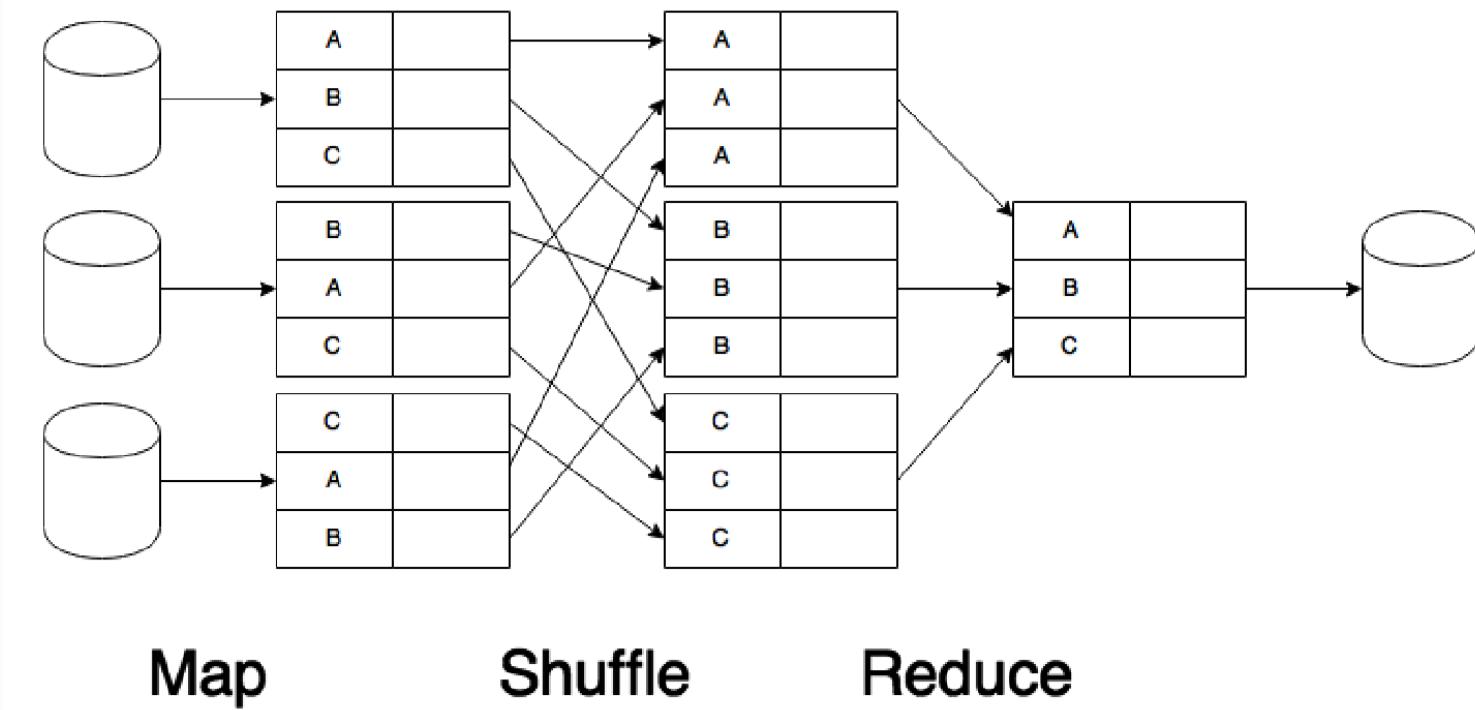
Examples:

- Fraud Detection: Gaming company catches cases of fraud previous queries could not detect. company reduced its fraud analytics cycle time from one week to 15 minutes, with query response dropping from 90 minutes to 90 seconds.
- Graph Analysis: Social media company to understand how its users are connected and enhance the networks of its community.

<http://thejackalofjavascript.com/mapreduce-in-mongodb>



Distributed MapReduce



MapReduce Original Google's Paper

“MapReduce is a programming model and an associated implementation for processing and generating large datasets.

Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key.

Many real world tasks are expressible in this model, as shown in the paper. “

JeffreyDean and Sanjay Ghemawat

[MapReduce: Simplified Data Processing on LargeClusters](#)



MapReduce

- Sharing Behavior: Reduce query times as it analyzes the items that people share online to understand sharing behavior.
- Search Behavior: An online media company uses the MapReduce function to better understand the paths its users follow after conducting a search to improve search results.
- Machine Learning: Research show that algorithms that fit the Statistical Query model can be written in a certain "summation form," which allows them to be easily **parallelized** on multicore computers.

The researchers adapt Google's map-reduce paradigm to demonstrate this parallel speed up technique on a variety of learning algorithms including locally weighted linear regression (LWLR), k-means, logistic regression (LR), naive Bayes (NB), SVM, ICA, PCA, gaussian discriminant analysis (GDA), EM, and backpropagation (NN).

MapReduce Program Structure

Three parts will be implemented as classes:

1) **Driver** : Configures how the program will run in Hadoop.

2) **Mapper** : Fed input data from Hadoop and produces intermediate output.

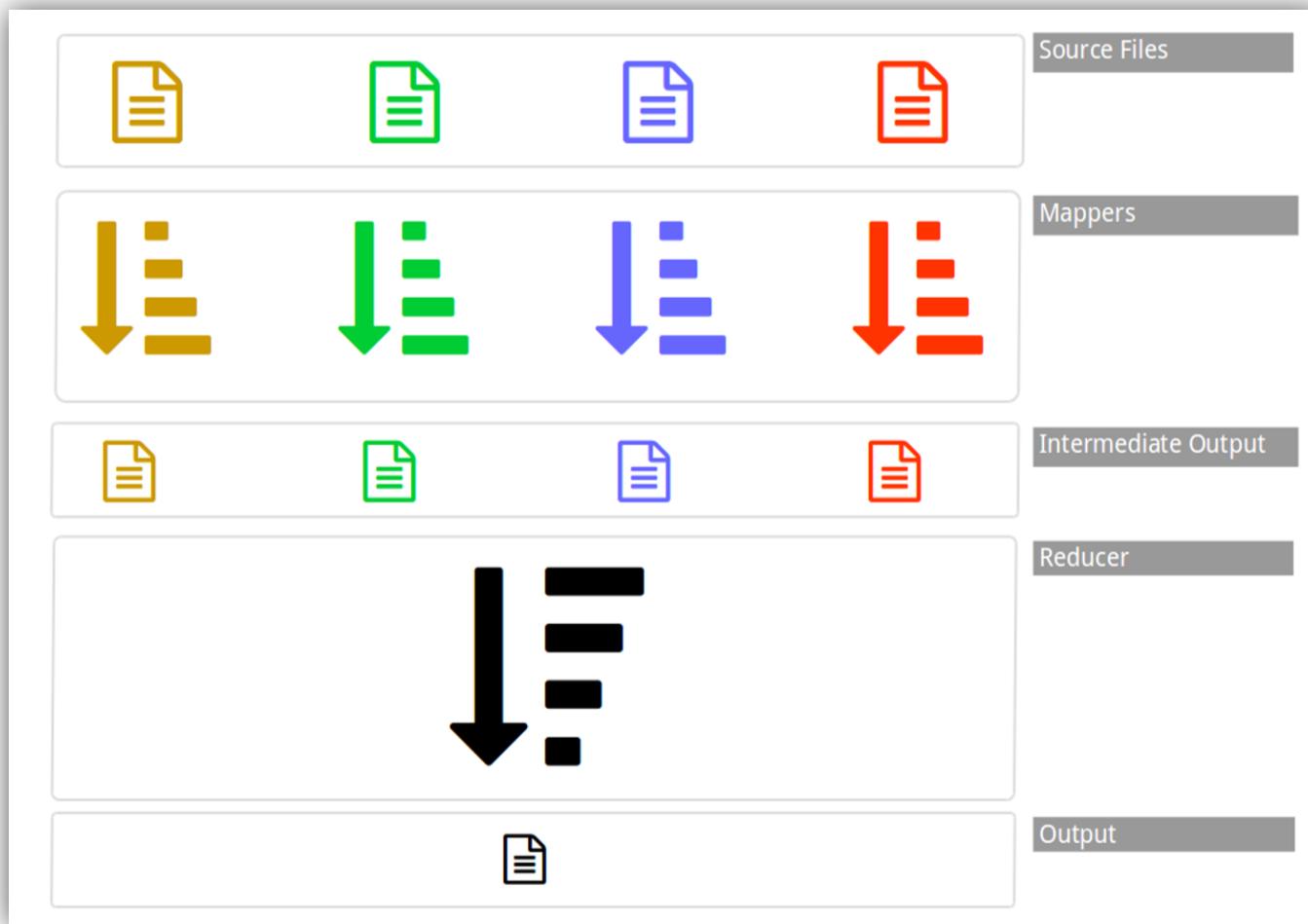
- Runs in parallel - each input file (for large input files, each file block) has a dedicated mapper task that will run on one of the data nodes.
- Each mapper is given only a small subset of the total input data, and its job is to read through the input and transform it into intermediate output.
- Each mapper produces only a part of the total output.

3) **Reducer** : Fed the intermediate output from mappers, produces final output.

- Reducer builds final results by aggregating many intermediate outputs from the mappers into a single result (sort and merge).
- Simple jobs have only one reducer task.



Data Flow in MapReduce



MapReduce Data Considerations

- When data moves between the mappers and reducer, it might cross the network, so all the values must be **Serializable**.
- MapReduce framework provides its own set of classes that are built for efficient serialization (i.e. Text instead of String).
- Hadoop must be explicitly told the data types of our classes for validating the job is properly configured before starting.
- The driver executes this configuration, specifying which mapper and reducer to use and the type of inputs and outputs.
- Inputs and outputs are **always a key-value pair** : you specify the type of the key and the type of the value in the driver.



HDFS



Hadoop Distributed File System

What Is HDFS

- HDFS is the storage part of the Hadoop platform.
- HDFS brings reliability and scale on commodity hardware.
 - No Special Hardware needed.
 - Nodes don't need to have the same specifications.
 - Add new nodes to a running cluster while increasing your storage and compute power without any downtime.
 - Resilience :Servers can go offline, disks can fail without loss of data.
 - Redundancy : (default) Hadoop replicates Data three times in the cluster with Rack Awareness (in large installations).
- HDFS Access data in a random manner, not Sequential.
- HDFS is intended as a write-once, read-many file system that significantly simplifies data access for the rest of the platform

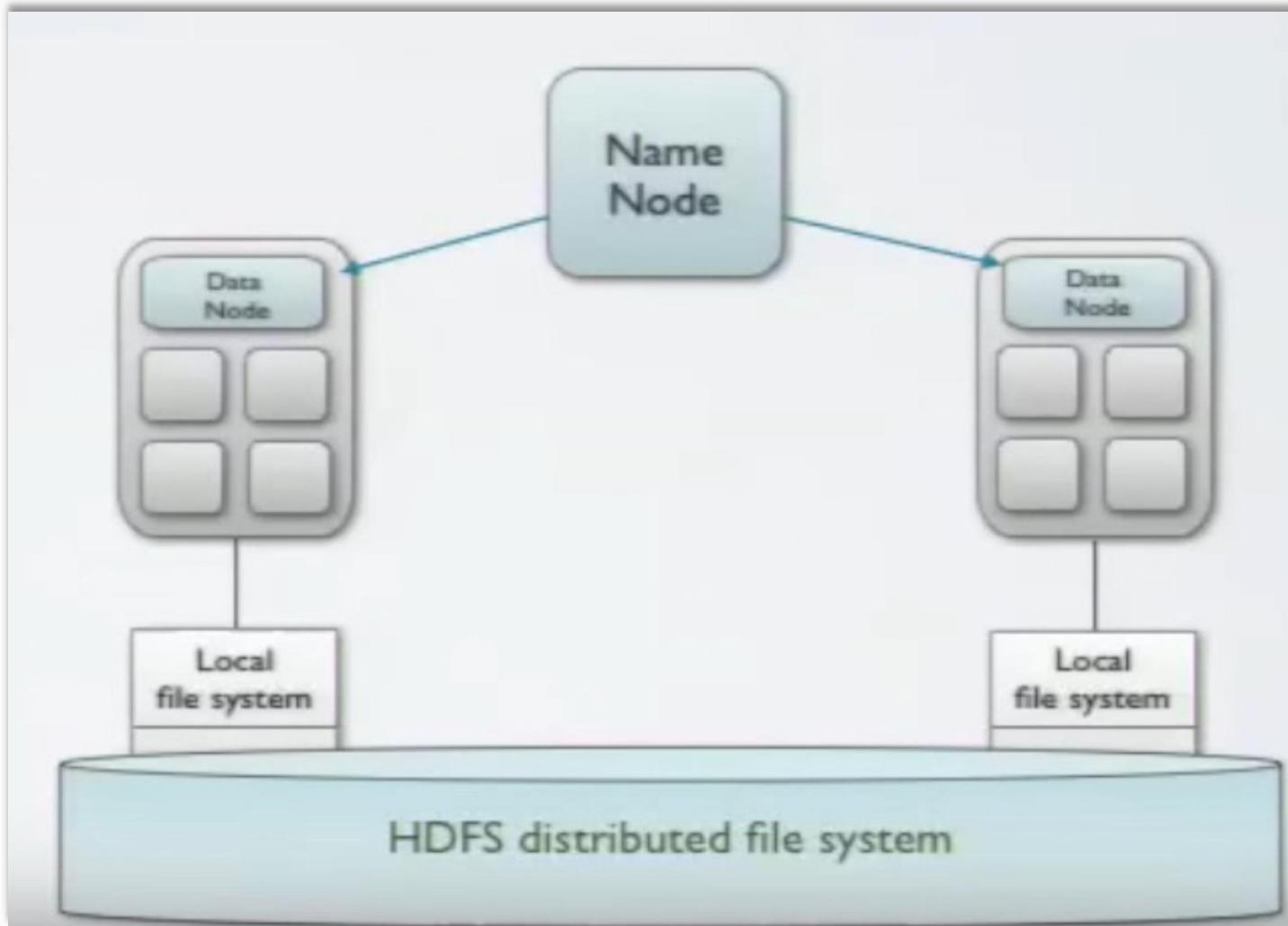


HDFS Node name

- HDFS is closed system: . When you read or write data in Hadoop, you must do it through the HDFS interface.
- The data is distributed among many data nodes, but the index specifying which file is on which node is stored centrally.
- HDFS uses a master/slave architecture in which the master is called the “**Name node**” and the slaves are called “data nodes”.
- **Name node** is a single-point-of-failure.
- Whenever accessing data in HDFS, it's via the name node, which owns the HDFS-equivalent of a file allocation table, called the file system namespace.
- If **Node name** is lost it's near impossible to get the cluster operational again.

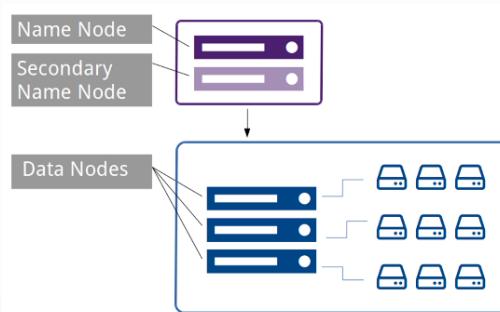


HDFS Node name



HDFS Name Nodes

- Hadoop clusters have a secondary name node that has a replicated file index from the primary name node.
- The secondary is a passive node—if the primary fails, you'll need to manually switch to the secondary, which can take tens of minutes.
- Customer wise, Hadoop cluster is a single, large-file store. details of how files are physically stored on data nodes is abstracted.
- Every file has a unique address, with HDFS scheme & nested folder layout:
`hdfs://[namenode]/logs/2016/04/19/20/web-app-log-201604192049.gz.`



HDFS Commands

- Storage Command Line operations begin with fs (“file system”).
- Hadoop fs -ls : list all objects in HDFS.
- Hadoop fs –cat: read contents of one-or-more files.
- HDFS is hierarchical and can store directories and files.
- Like linux: objects have owner & group, and settings like read/write/execute.
- Users in HDFS have their own home directory.
- .

```
# hadoop fs -mkdir -p /user/root/ch03
# hadoop fs -ls
Found 1 items
drwxr-xr-x - root supergroup 0 2016-04-
15 16:44 ch03
# hadoop fs -ls /user/root
Found 1 items
drwxr-xr-x - root supergroup 0 2016-04-
15 16:44 /user/root/ch03
```

Create directory called 'ch03' in home directory for root user and check that the folder is where we expect it to be.



HDFS Commands

- **put** operation copies files from local fs to HDFS.
- **moveFromLocal** deletes the local source files after copying them
- **get** Copies file or directory in HDFS to local file system path.
- **appendFromLocal** adds contents of local files to existing files in HDFS
- **tail** reads the last 1KB of data in a file.
- **stat** sees useful stats on a file, including the degree of replication.
- **count** tells how many files and folders are in the hierarchy.
- More: https://www.tutorialspoint.com/hadoop/hadoop_command_reference.htm

Browse File System: <http://localhost:50070/explorer.html#/>

The screenshot shows a web browser window displaying the HDFS Web UI at <http://localhost:50070/explorer.html#/>. The title bar says "localhost:50070/explorer.html#/" and the tabs include "Hadoop", "Overview", "Datanodes", "Snapshot", "Startup Progress", and "Utilities". The main content area is titled "Browse Directory" and shows a table of files in the root directory. The table has columns: Permission, Owner, Group, Size, Last Modified, Replication, Block Size, and Name. There are two entries:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwx-----	root	supergroup	0 B	3/5/2017 1:03:28 AM	0	0 B	tmp
drwxr-xr-x	root	supergroup	0 B	3/5/2017 1:00:46 AM	0	0 B	user

At the bottom left, it says "Hadoop, 2015."



HDFS Commands

- **put** operation copies files from local fs to HDFS.
- **moveFromLocal** deletes the local source files after copying them
- **get** Copies file or directory in HDFS to local file system path.
- **appendFromLocal** adds contents of local files to existing files in HDFS.
- **tail** reads the last 1KB of data in a file.
- **stat** sees useful stats on a file, including the degree of replication.
- **count** tells how many files and folders are in the hierarchy.
- More: https://www.tutorialspoint.com/hadoop/hadoop_command_reference.htm



HDFS Append

- HDFS is intended as a **write-once, read-many file system** that significantly simplifies data access for the rest of the platform.
- Data can not be edited in Hadoop. But Can be appended to existing files.
- Hadoop/HDFS is used for batch processing over data that records something (an event or a fact), and that data is static.
 - Example: Hadoop that records a day's worth of financial trades- those are a series of facts that will never change, so there's no need to edit the file.
 - If a trade was booked incorrectly, it will be rebooked, which is a separate event that will be recorded in another day's file.
- In order to append data without altering the original file, HDFS splits the new data into blocks and adds them to the original blocks into which the file was split.



HDFS API's & UI

- HDFS supports Java API and REST API for programmatic access.
- HDFS supports web UI to explore the file system, it is built into HDFS.

<http://localhost:50070/explorer.html#/>

The screenshot shows a web browser window displaying the HDFS Web UI at <http://localhost:50070/explorer.html#/>. The page title is "Browse Directory". The top navigation bar includes links for Hadoop, Overview, Datanodes, Snapshot, Startup Progress, and Utilities. The main content area shows a table of files in the root directory:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwx-----	root	supergroup	0 B	3/5/2017 1:03:28 AM	0	0 B	tmp
drwxr-xr-x	root	supergroup	0 B	3/5/2017 1:00:46 AM	0	0 B	user

At the bottom of the page, there is a footer note: "Hadoop, 2015."



YARN- Resource Negotiator

Hadoop Compute has three parts – Coupled !

- Resource manager
- Job scheduler
- Task monitor

Hadoop clusters can run different types of jobs

- Not Just MapReduce
- Due to YARN (new from version 2.0)

YARN is a black box for Hadoop programs

- submit MapReduce job with its configuration to the cluster and let YARN do the rest.



YARN- Resource Negotiator

- Hadoop was written as MapReduce Engine.
- Because it runs on a cluster, it had Cluster Management Component, But Coupled to MapReduce.
- Result = you have a cluster dedicated for Batch processing Big Data Stores – Solely.



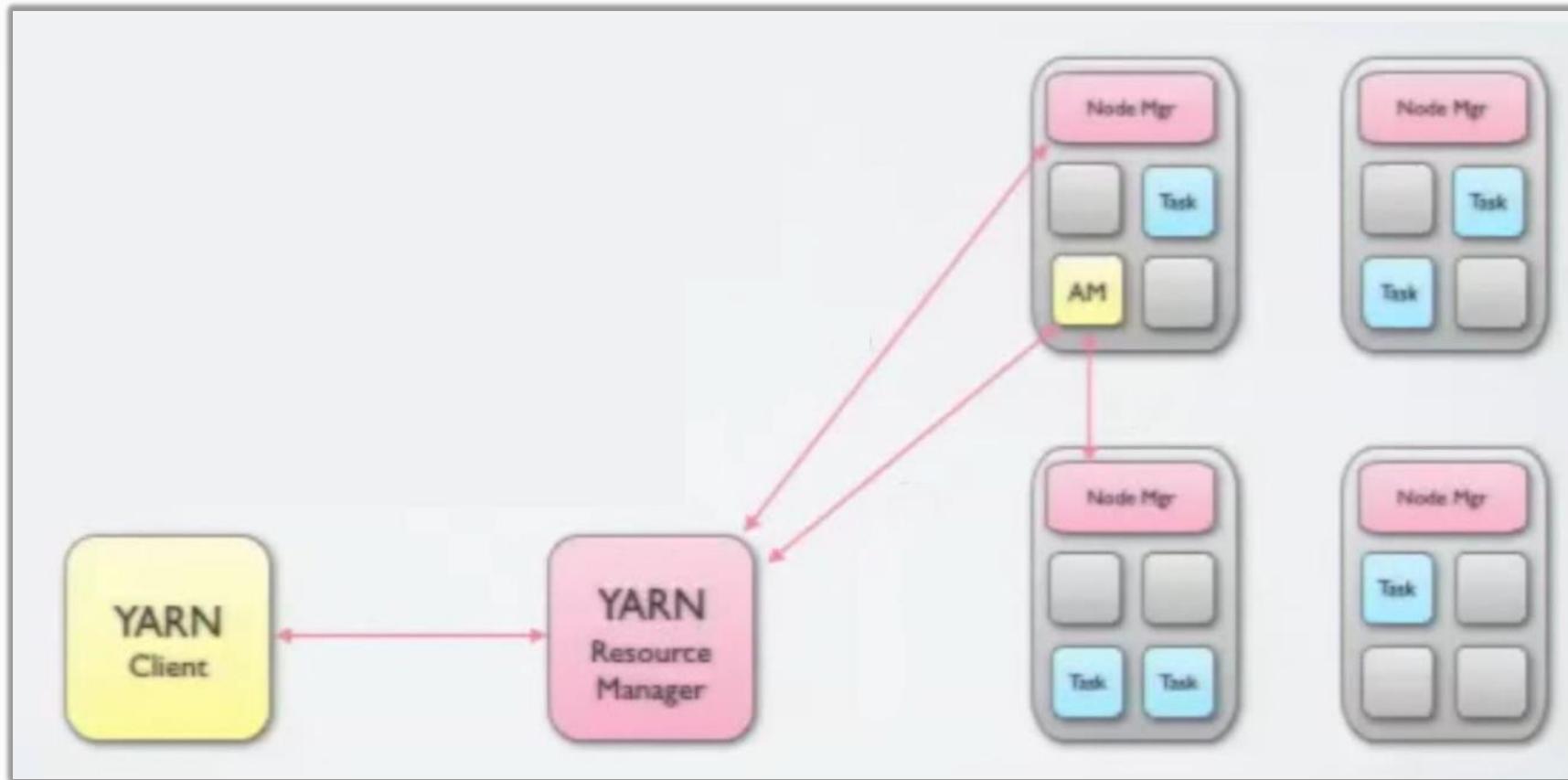
YARN- Resource Negotiator

- In order to utilize the cluster more efficiently, The Cluster Resource management was Decoupled from Application Management (MapReduce) = Enter YARN.
- YARN enables running many types of Applications on the cluster.
- YARN adds one machine to the cluster, knowing all about the resources in the cluster.
- Each application adds one machine = Application Master
- The AM talks with YARN to get resources and start tasks.

YARN- Resource Negotiator

- YARN treats each node (Compute machine) as a collection of slots.
- Slot = Container that can run a task.
- Each Node has Node Manager to manage Slots.
- Application master asks YARN for slots and run tasks.

YARN- Resource Negotiator



YARN-Resource Negotiator

Due To YARN

- MapReduce needs **only** custom mapper, reducer & driver.
- No need to specify how to break up the job or where to run tasks.

YARN is an abstracted job schedule.

- In Hadoop cluster, YARN may Schedule MapReduce Jobs along side Hive Queries or run Spark.

YARN provides resilience & scalability.

- A job submitted to YARN is expected to complete.



Monitoring jobs in YARN

Browse Services: <http://localhost:8088/cluster>

← → ⌂ | localhost:8088/cluster |      ...

 All Applications

Logged in as: dr.who

Cluster Metrics																
Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	Vcores Used	Vcores Total	Vcores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes	
3	0	0	3	0	0 B	8 GB	0 B	0	8	0	1	0	0	0	0	

Scheduler Metrics		Scheduler Type		Scheduling Resource Type		Minimum Allocation			Maximum Allocation		
Capacity Scheduler	[MEMORY]	<memory:1024, vCores:1>		<memory:8192, vCores:8>							

Show 20 entries

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI	Blacklisted Nodes
application_1488668362570_0003	root	grep-search	MAPREDUCE	default	Sun Mar 5 01:14:31 +0200 2017	Sun Mar 5 01:15:01 +0200 2017	FINISHED	FAILED		History	N/A
application_1488668362570_0002	root	grep-sort	MAPREDUCE	default	Sun Mar 5 01:04:15 +0200 2017	Sun Mar 5 01:04:32 +0200 2017	FINISHED	SUCCEEDED		History	N/A
application_1488668362570_0001	root	grep-search	MAPREDUCE	default	Sun Mar 5 01:03:29 +0200 2017	Sun Mar 5 01:04:12 +0200 2017	FINISHED	FAILED		History	N/A

Showing 1 to 3 of 3 entries



Hadoop Streaming

- Hadoop itself is native Java platform.
- Yet Support for building MapReduce components in other languages Exists – named **Hadoop Streaming**.
- Simple approach – Hadoop invokes executable as part of a task, rather than hosting a JAR file in a JVM.
- Hadoop Streaming uses stdin and stdout streams to communicate with the executing process.



Hadoop Streaming

- Streaming allows to use MapReduce without having to write Java code for mappers and reducers.
- Mix & Match: analytical code in Python, MapReduce code in .NET, Java mapper with a C++ reducer, R mapper with Python reducer.



Hadoop Streaming

```
hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop
-streaming-2.7.2.jar -input input/hadoop
-output output4
-mapper /bin/cat -reducer /usr/bin/wc
```

This command runs a MapReduce program with the cat command as the mapper, which will concatenate all the input lines into one output, and with the wc command as the reducer, which runs a word count over the intermediate output.

```
hadoop fs -cat output4/part-00000
```

```
2022 7600 76858
```



Streaming Interface

Data is transferred through Hadoop as key-value pairs.

Input is fed from Hadoop to the executable in a series of tab-separated lines where:

- The string before the first tab character is the key.
- The string after the first tab character is the value.
- Each key-value pair is sent to stdin as a single line.
- An empty string sent to stdin represents the end of the input.



Java Word Count Demo

<https://wiki.apache.org/hadoop/WordCount>



Hadoop & Node.js

<https://www.packtpub.com/books/content/introduction-using-nodejs-and-hadoop-store-distributed-data>

<https://www.mongodb.com/blog/post/using-mongodb-hadoop-spark-part-1-introduction-setup>



MongoDB & Node.js

<https://docs.mongodb.com/ecosystem/tutorial/getting-started-with-hadoop>

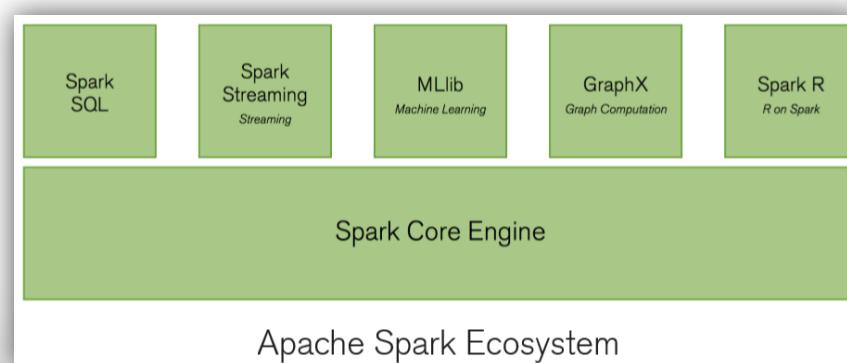




Spark

Introducing Spark

- Spark was initially designed for interactive queries and iterative algorithms, these were two major use cases not well served by batch frameworks such as Hadoop's MapReduce.
- Spark excels in scenarios requiring fast performance, such as iterative processing, interactive querying, large-scale batch operations, streaming, and graph computations.
- Spark allows programmers to develop complex, multi-step data pipelines using a directed acyclic graph (DAG), also supports in-memory data sharing across DAG's.



Introducing Spark

- Spark is designed as an execution engine that works with data both in-memory and on-disk.
- Spark takes MapReduce to the next level with less expensive shuffles during data processing.
- Spark holds intermediate results in memory not writing them to disk, useful especially when a process needs to iteratively work on the same dataset.
- Spark operators perform external operations when data does not fit in memory. it can be used for processing datasets that are larger than the aggregate memory in a cluster.

Spark Properties

- **Simplicity:** Easy-to-use APIs for operating on large datasets. This includes a collection of sophisticated operators for transforming and manipulating semi-structured data.
- **Speed:** Exploiting in-memory optimizations, Spark shown up to 100x higher performance than MapReduce running on Hadoop.
- **Unified Framework:** Packaged with higher-level libraries, including support for SQL queries, machine learning, stream and graph processing. These libraries increase productivity and can be combined to create complex workflows.

Spark RDD

- RDD (Resilient Distributed Dataset) is the core data structure used in the Spark framework - analogous to relational database table or collection in MongoDB.
- RDDs are immutable - they can be modified with a transformation, but the transformation returns a new RDD while the original RDD remains unchanged.
- RDDs supports TWO types of operations:
 - Transformation - return a new RDD after processing with function like map, filter, flatMap, groupByKey, reduceByKey, aggregateByKey, pipe, or coalesce etc.
 - Action - evaluates the RDD and returns a new value, including reduce, collect, count, first, take, countByKey, and foreach.

Spark & MongoDB

- DB like MongoDB natively offers rich analytics capabilities, yet integrating the Spark engine can extend the real-time processing of operational data managed by it.
- DB like MongoDB exposes operational data to Spark's distributed processing layer to provide fast, real-time analysis.
- Spark supports over 100 different operators and algorithms for processing data. For example, Spark offers native support for advanced machine learning algorithms including k-means clustering and Gaussian mixture models.
- Combining Spark queries with MongoDB indexes allows data to be filtered, avoiding full collection scans and delivering low-latency responsiveness with minimal hardware and database overhead.

<https://github.com/mongodb/mongo-spark>

Spark vs. Hadoop

Daytona GraySort 2014 Competition Results

	2013 Record: Hadoop	2014 Record: Spark
Data Size	102.5 TB	100 TB
Elapsed Time	72 minutes	23 minutes
Number of Nodes	2100	206
Sort rate	1.42 TB/min	4.27 TB/min
Sort rate per node	0.67 GB/min	20.7 GB/min

Spark vs. Hadoop

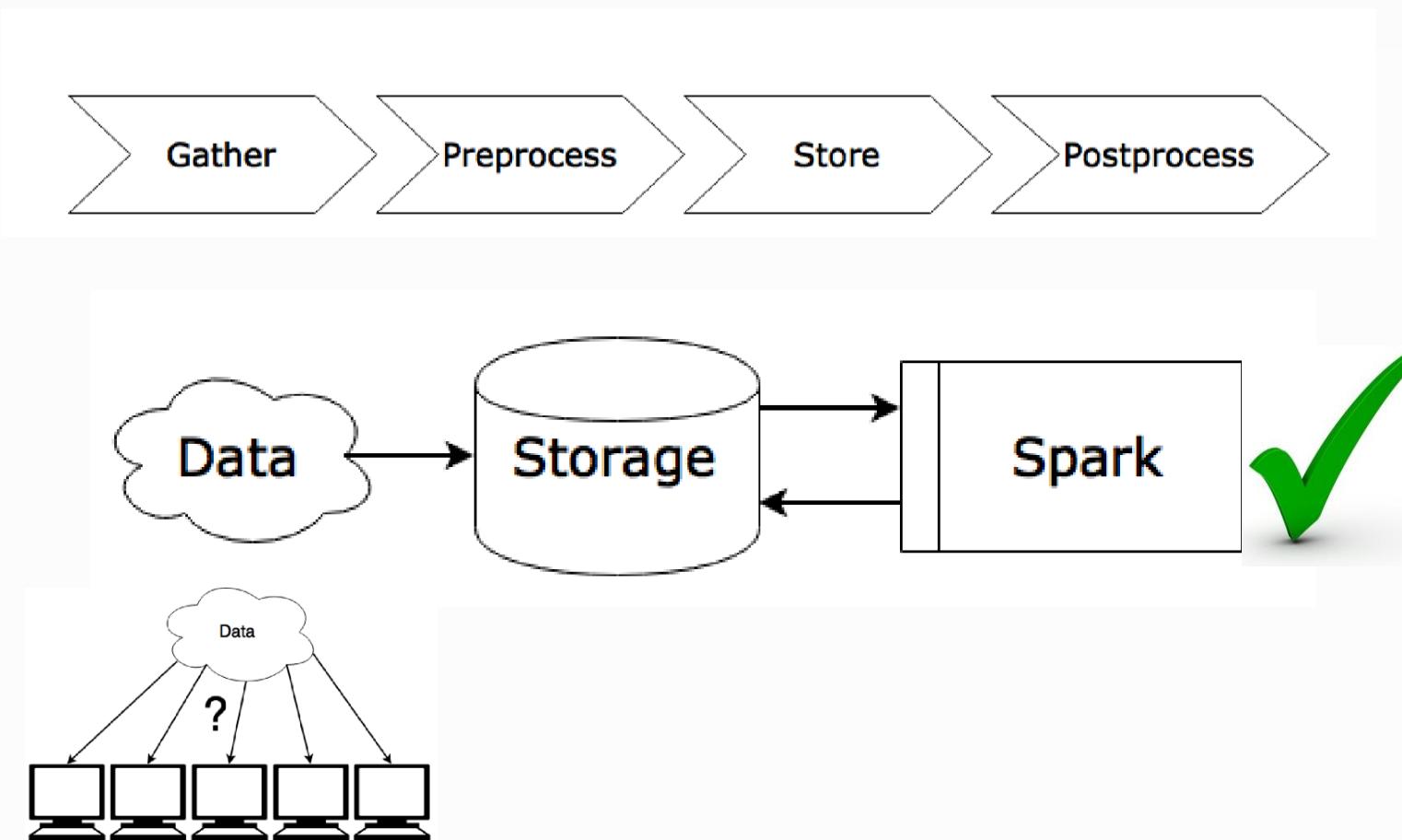
- Spark promise of speeds up to 100 times faster than Hadoop MapReduce.
- Spark process data Fast Because it runs in-memory on the cluster, and it isn't tied to Hadoop's MapReduce two-stage paradigm.
- Spark processes data in-memory while Hadoop MapReduce persists back to the disk after a map or reduce action, so Spark should outperform Hadoop MapReduce.
- Spark can run as a standalone or on top of Hadoop YARN, where it can read data directly from HDFS.
- Spark needs **a lot of memory**. Much like standard DBs, it loads a process into memory and keeps it there until further notice, for the sake of caching.

Spark vs. Hadoop

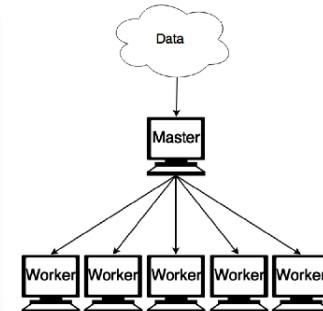
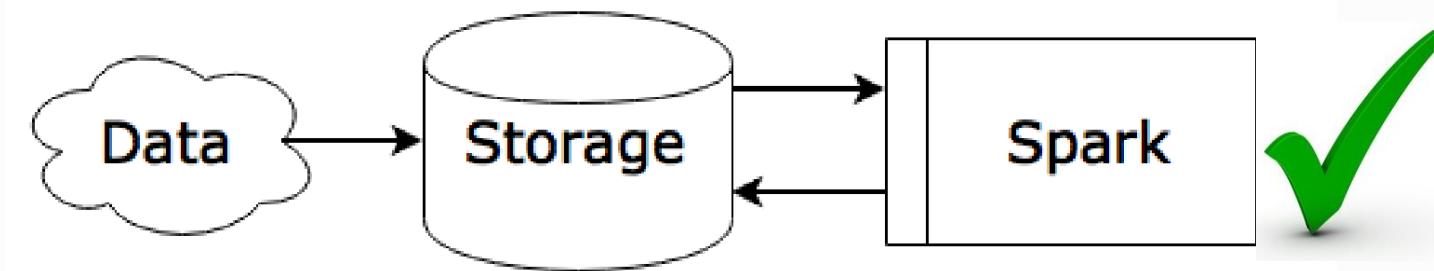
- MapReduce kills its processes as soon as a job is done, so it can easily run alongside other services with minor performance differences.
- Spark has the upper hand as long as we're talking about iterative computations that need to pass over the same data many times.
- When it comes to one-pass ETL-like jobs, data transformation or integration, then MapReduce is BETTER : it was designed for it.
- Thanks to its high performance, Spark can do real-time processing as well as batch processing. Hadoop MapReduce is great only for batch processing.

Spark performs better when all the data fits in the memory, especially on dedicated clusters; Hadoop MapReduce is designed for data that doesn't fit in the memory and it can run well alongside other services.

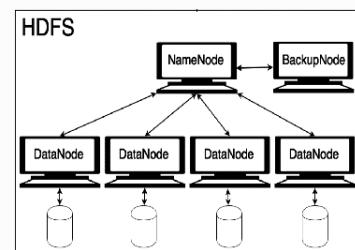
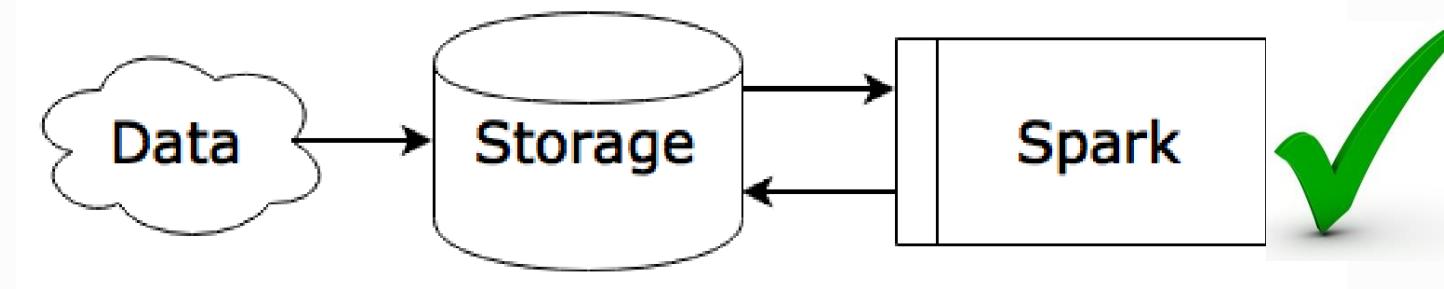
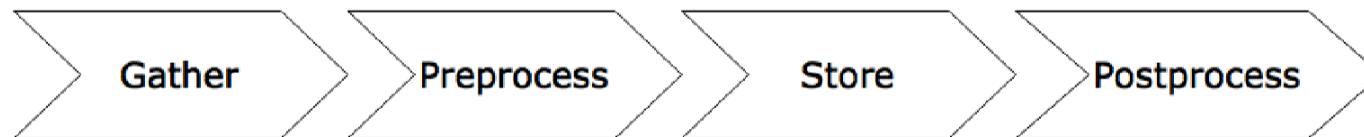
Using Spark



Using Spark

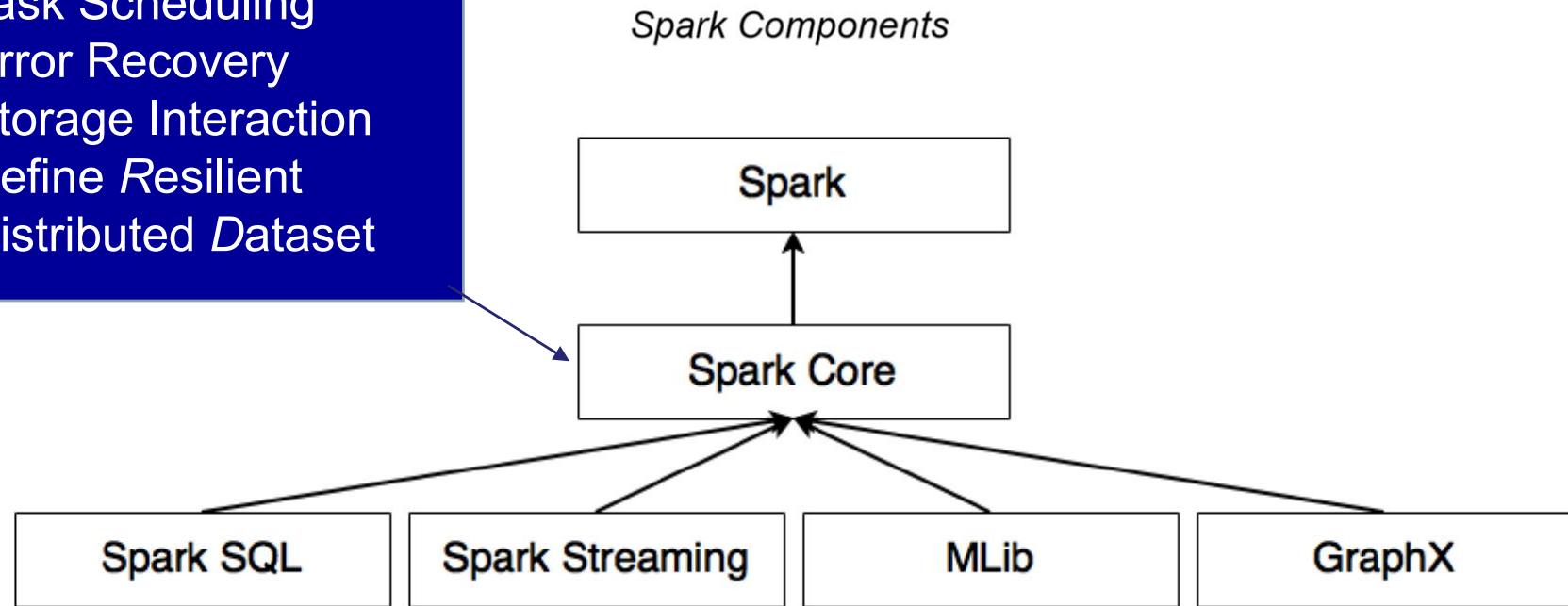


Using Spark



Spark Modules

- Memory Management
- Task Scheduling
- Error Recovery
- Storage Interaction
- Define Resilient Distributed Dataset



Spark Demo

```
docker run -it -v d:/HDFSLocal:/HDFSLocal -p 8088:8088 -p 8042:8042 -h  
sandbox sequenceiq/spark:1.6.0 bash
```

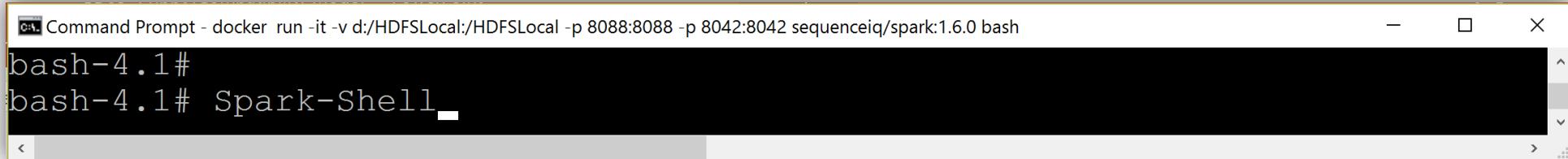
You MUST put files to be processed in HDFS, we will use a shared folder (**d:/HDFSLocal** is windows folder) to interface with Linux Container, and transfer the files to HDFS using the following under bash cli

```
hadoop fs -mkdir /[folder]
```

```
hadoop fs -put /HDFSLocal/data.txt /[folder]/data.txt
```

```
hadoop fs -rm [filename] - Optional
```

Spark Demo Scala

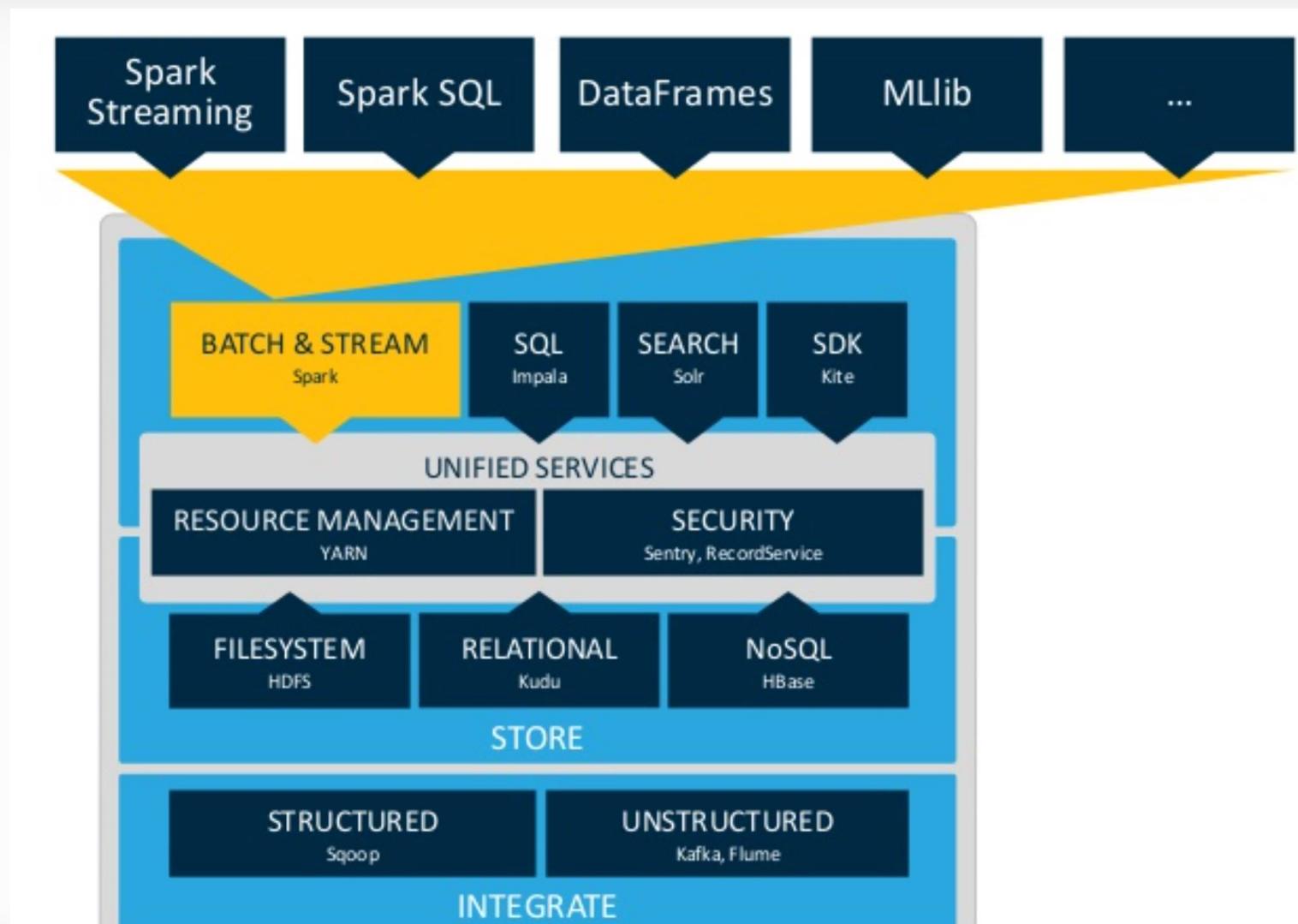


Command Prompt - docker run -it -v d:/HDFSLocal:/HDFSLocal -p 8088:8088 -p 8042:8042 sequenceiq/spark:1.6.0 bash
bash-4.1#
bash-4.1# Spark-Shell

```
val readmeFile = sc.textFile("/inputText/SparkWordsToCount.txt")
val starMentions = readmeFile.filter(line => line.contains("star"))
starMentions.count()
readmeFile.take(10)
```

```
println("lets count using MapReduce")
val readmeFile = sc.textFile("/inputText/SparkWordsToCount.txt")
val fm = readmeFile.flatMap(line => line.split(" "))
val m = fm.map(word => (word, 1))
m.take(5)
val cts = m.reduceByKey((a,b) => a + b)
val sorted =cts.sortBy(item => item._2, false)
sorted.take(5).foreach(println)
```

Spark & Friends



Spark with Java

<https://www.syncfusion.com/resources/techportal/details/ebooks/spark>

Node.js + Spark

<https://github.com/henridf/apache-spark-node>

Spark & MongoDB

<https://www.mongodb.com/collateral/apache-Spark-and-mongodb-turning-analytics-into-real-time-action>



Using MongoDB + Spark + Hadoop

Demo

<https://github.com/henridf/apache-spark-node>

Using Kafka + Spark + Hadoop

<https://bigdatapath.wordpress.com/2019/01/16/real-time-big-data-pipeline-with-hadoop-spark-kafka/>

<https://supergloo.com/spark-streaming/spark-streaming-kafka-example/>

Introduction To Machine Learning



ML Crash Course

What is Machine Learning

- **Herbert Alexander Simon:**
“Learning is any process by which a system improves performance from experience.”
- “Machine Learning is concerned with computer programs that automatically improve their performance through experience.”



Herbert Simon
[Turing Award 1975](#)
[Nobel Prize in Economics 1978](#)

Learning in ML

Learning = Improving with experience at some task

Improve over task T

With respect to performance measure P

Based on experience E

ML vs. Programming

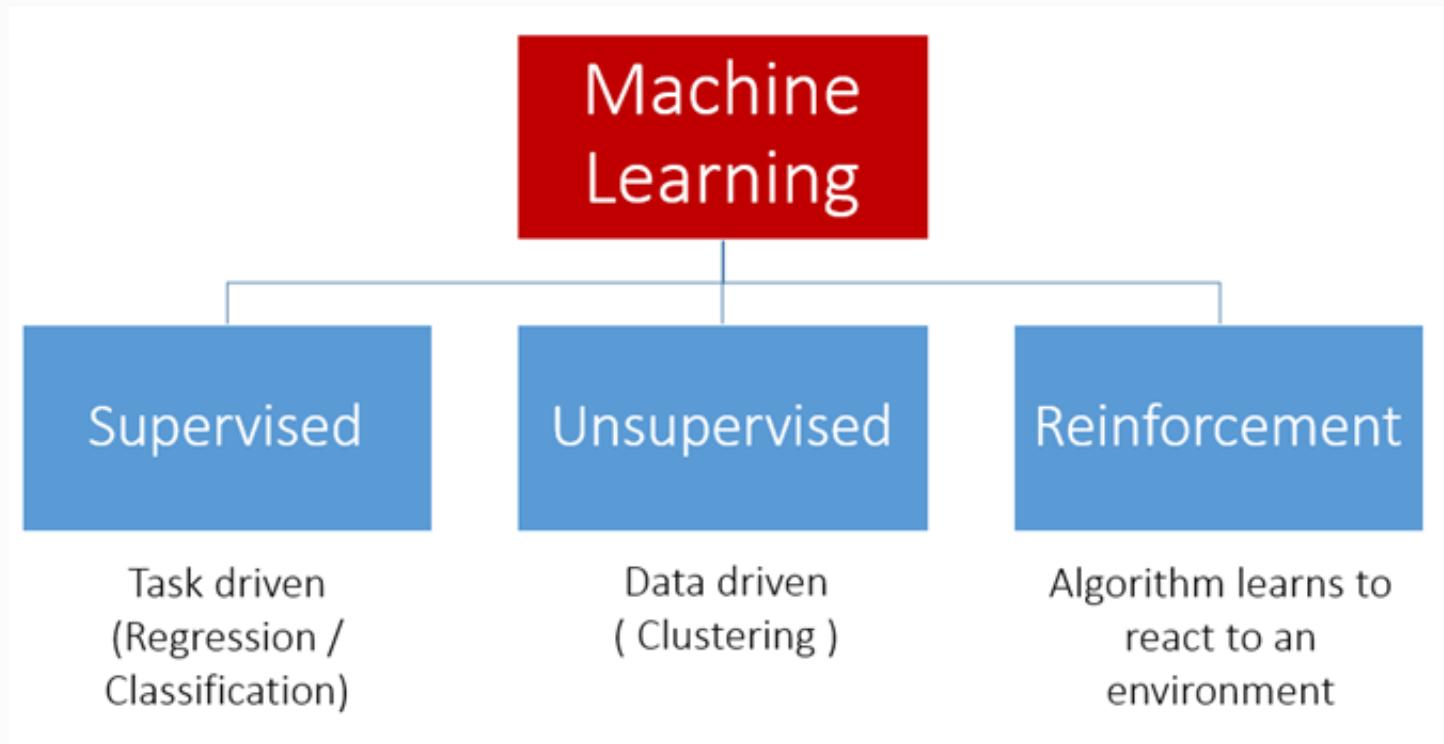
Traditional Programming



Machine Learning

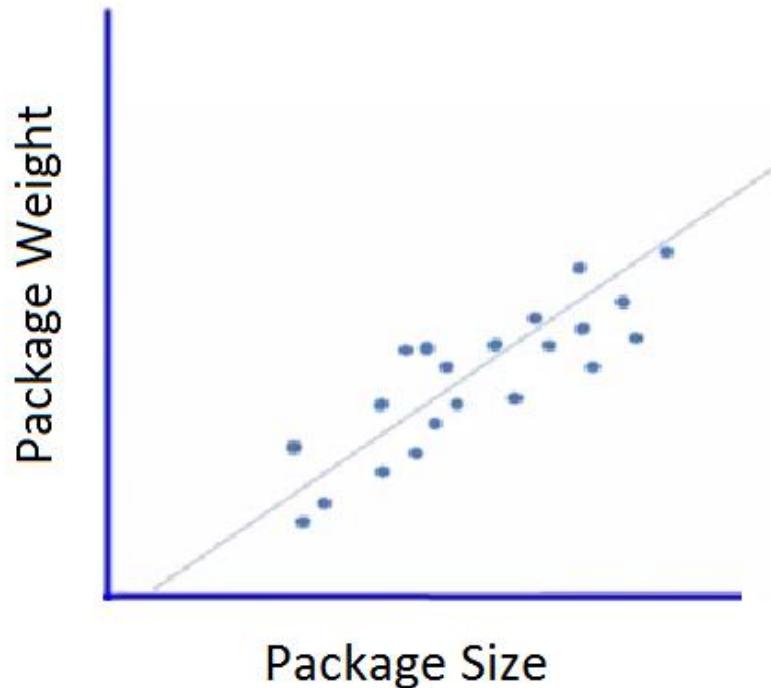


Types of ML



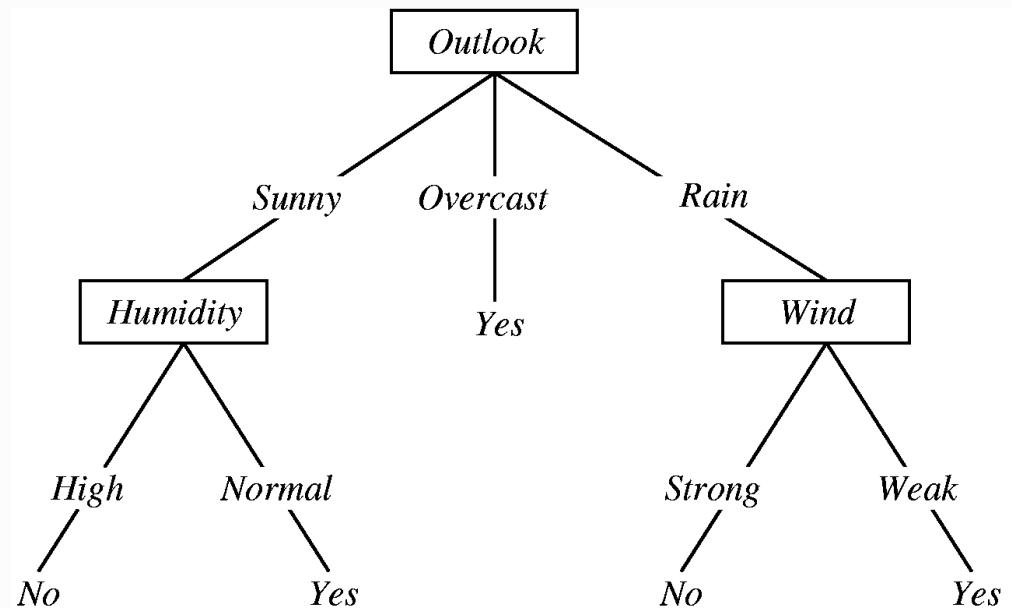
דוגמא ללמידה מפקחת : רגסיה לינארית

- חיפוש יחס בין משתנה תלוי מספרי לבין משתנים אחד או יותר בלתי תלויים.
- תוכנות המטריה היא נומרית.
- endum לביצוע חיזוי לערכו של המשתנה התלוי על בסיס ערכי תוכנות אחרות.



דוגמא ללמידה מפקחת : עץ החלטה

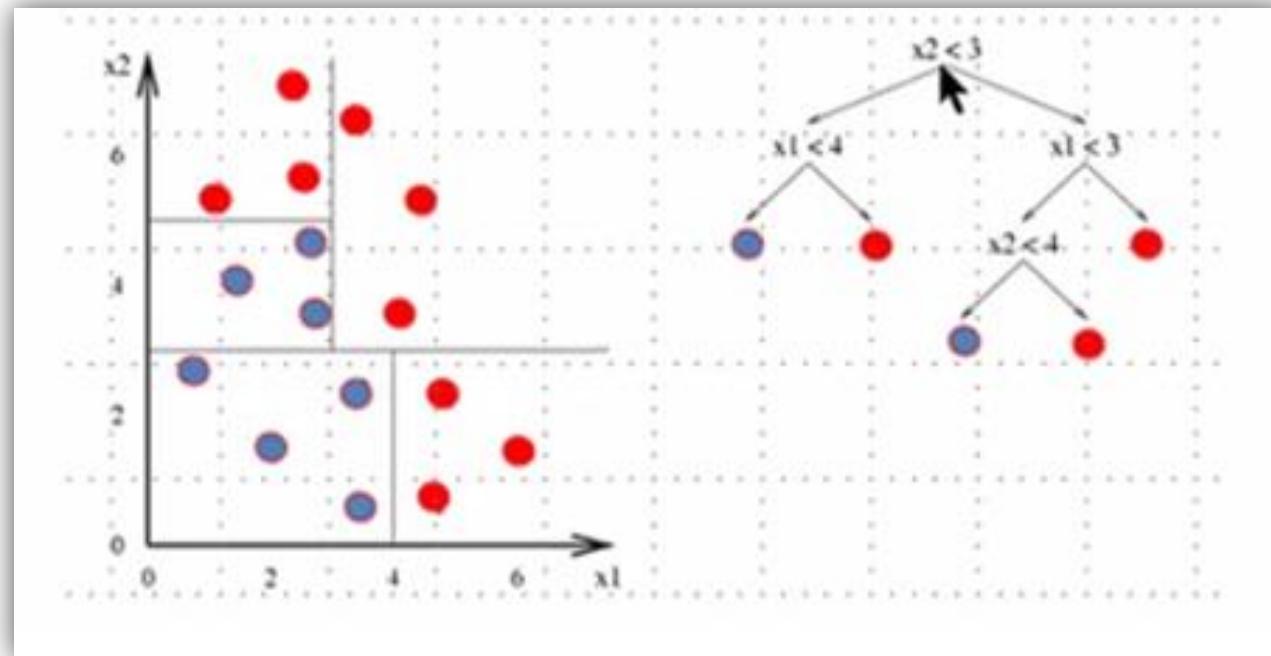
Outlook	Temperature	Humidity	Wind	Play Tennis
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No



<https://www.npmjs.com/package/decision-tree>

עצי החלטה

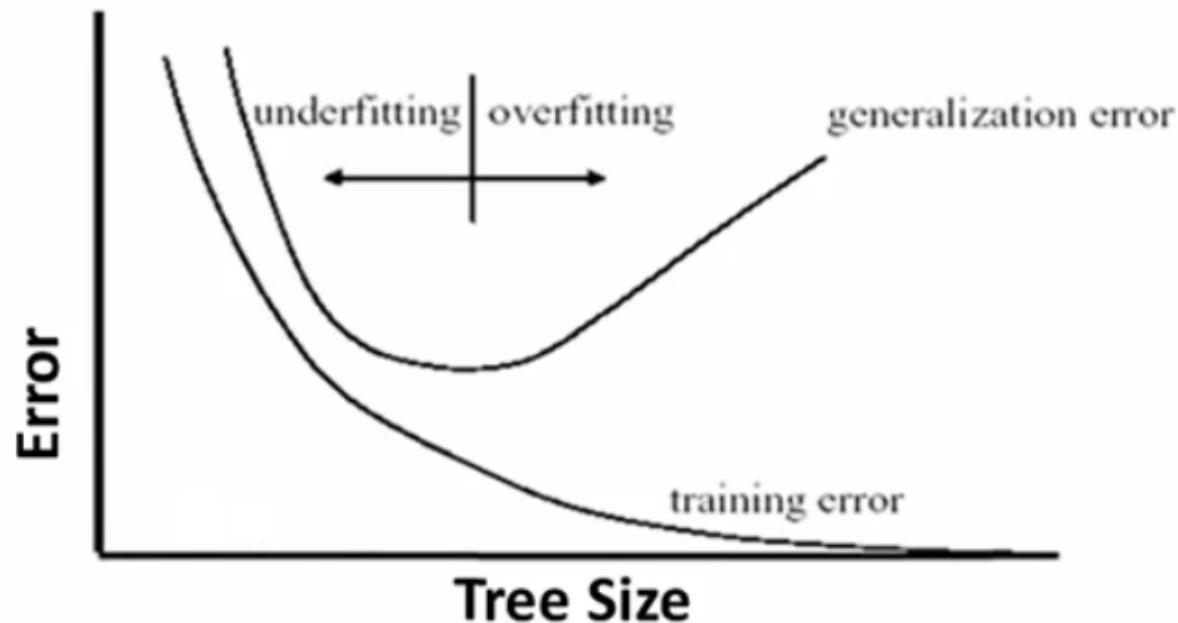
- צומת פנימית מייצגת פיצול על ערכי תכונה.
- קשת מייצגת תוצאה של פיצול.
- עליים מייצגים תיוג או התפלגות.
- עצי החלטה מחלקים את מרחב התכונות למלבנים, ומעניקים לכל מלבן תיוג מ�ור האופציות הנתונות (אדום וכחול בדוגמא).
- שיטת ייצרת העץ היא לרוב בגישה Top Down



<http://www.cs.utexas.edu/~dana/MLClass/decision-trees.pdf>

התאמות יתר ו忽ר

Overfitting and underfitting



Overtraining: means that it learns the training set too well – it overfits to the training set such that it performs poorly on the test set.

Underfitting: when model is too simple, both training and test errors are large

Model Evaluation

Learned Models Should Be Evaluated.

Confusion matrix Is a common method, showing the ways your classification model is confused when it makes predictions.

Confusion Matrix		Play Golf			
		Yes	No		
	Yes	7	2	<u>Positive Predictive Value</u>	0.78
	No	2	3	<u>Negative Predictive Value</u>	0.60
		<u>Sensitivity</u>	<u>Specificity</u>	<u>Accuracy</u> = 0.71	
		0.78	0.60		

Compute Services

ML-as-a-Service



Machine Learning & BigML

BigML SaaS

- Machine Learning commoditized into a Service

- Web Interface, CLI & RESTful API

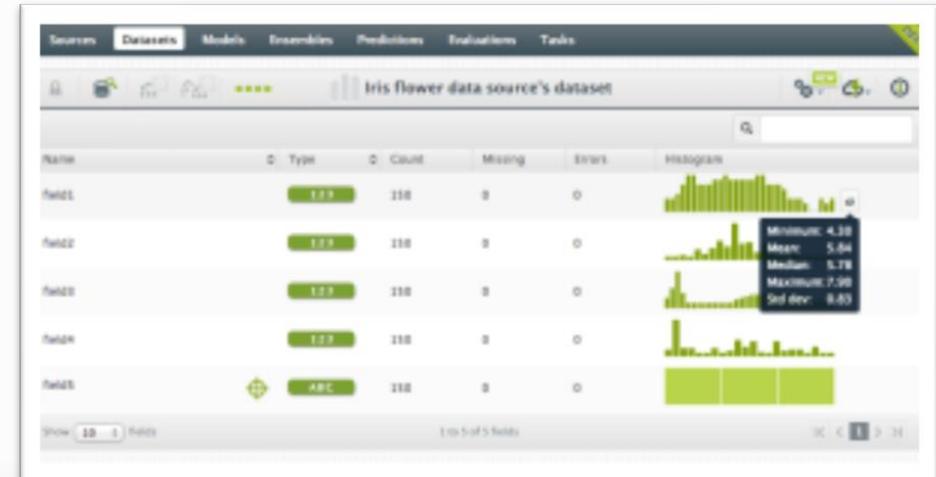
- Pipeline of Steps

Data Source → Dataset → Model → Ensembles



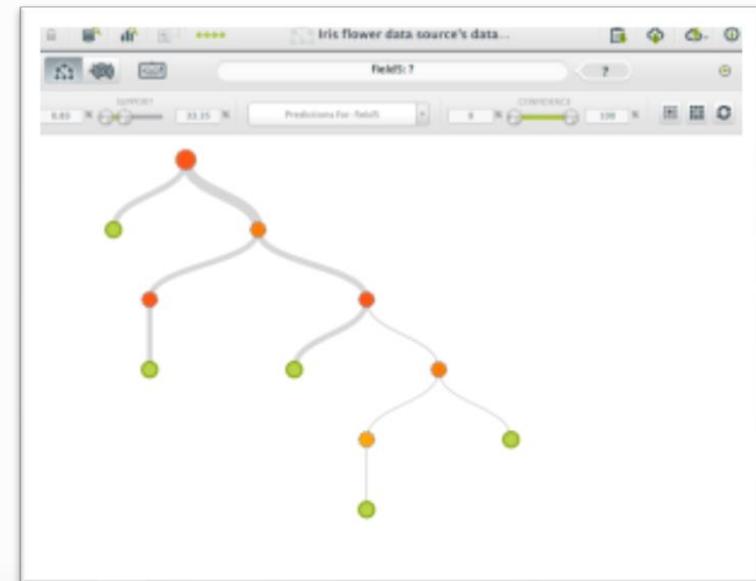
BigML Dataset

- Views on data source, used as basis for building models
- Specify the target attribute (class in classification or output in regression).
- Data is summarized with bar graphs and five-number summaries.
- Can be split into training and test for controlled evaluation of a models performance.



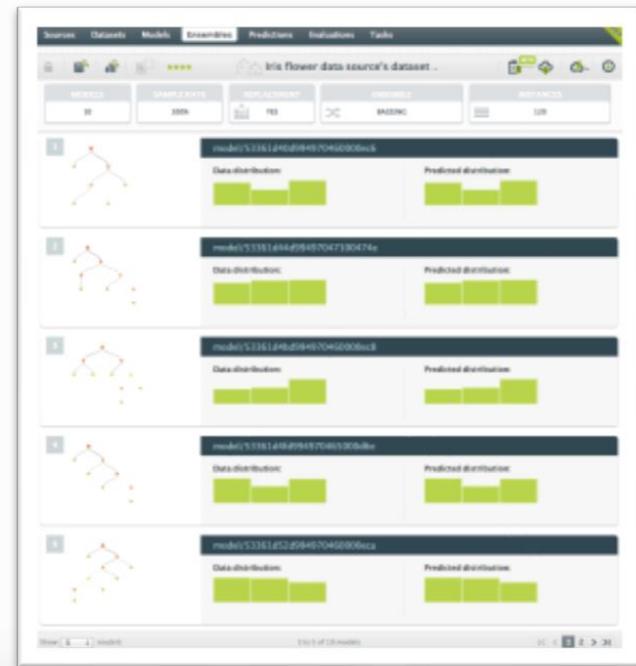
BigML Model

- Analytical Representation of knowledge learned from a dataset, for example decision trees.
- Decision trees are interactive
- Evaluation measures include confidence & support at each node
- You can walk through the tree and see the rule antecedents build up.
- Model can be downloaded in several languages including javascript & Python.



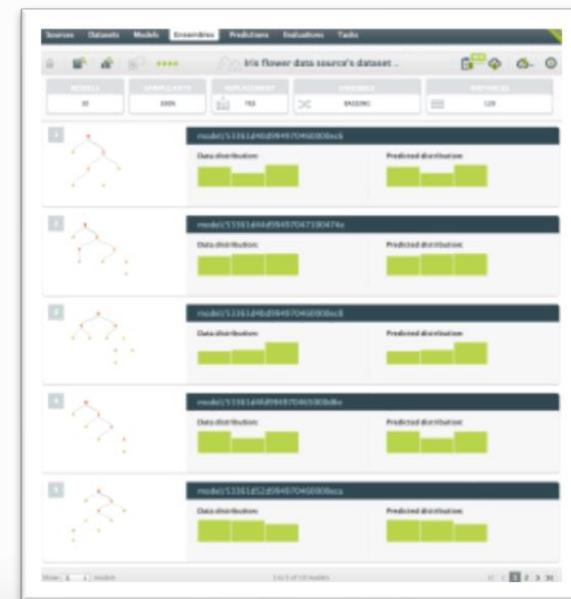
BigML Ensembles

- Models comprised of sub-models
- Less useful for descriptive models and more useful for predictive models
- Hopefully providing increased accuracy from the combination of predictions from varied perspectives of the problem domain



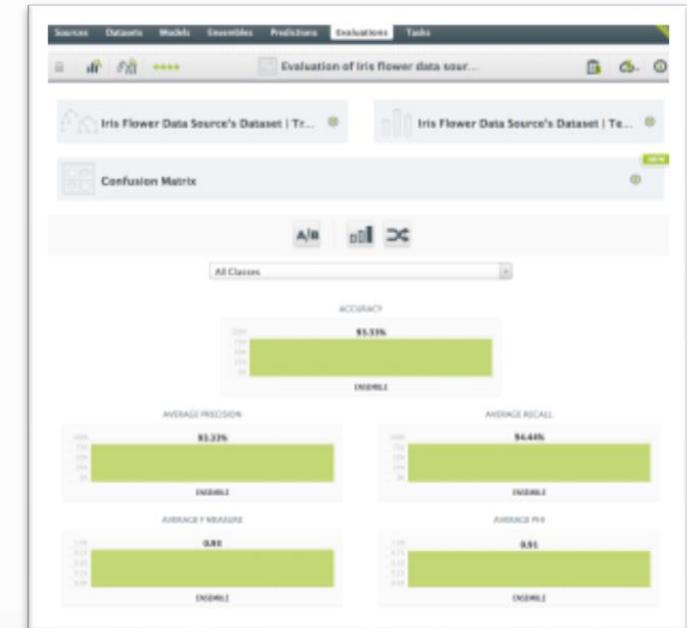
BigML Predictions

- Model can be used to generate predictions:
 - Can be question-by-question through the branches of the decision tree model that was constructed (like a decision support time), via sliders for specifying an input instance, one at a time.
 - Can be via batch predictions, the results of which can be downloaded to file.



BigML Evaluation

- Evaluation presents the estimation of a model's performance based on a dataset.
- If you split a dataset into training and test, you can estimate the capability of the model on unseen data.
- Measures include classification accuracy, precision, recall and others.
- The performance is also summarized in graphs.
- The performance of models can also be compared side-by-side.



Web Interface Features

- ➊ *1-Click:* Complete useful tasks in one-click, from a selected dataset one-click creates a model, an ensemble or a split of the dataset into training and test sets.
- ➋ *Interactive Trees:*
 - ➌ Decision trees are favorable because they can be visualized and easily understood by subject r experts.
 - ➌ Unambiguously present how a decision is made in the context of the domain (unlike NN or SVM).
 - ➌ Making the trees interactive is a natural next step, playing with the visualization and performing what-if's and relating it back to the domain.
- ➌ *Downloadable Trees:* You can download the rules for a model or the tree itself in a programming language of your choice.

Web Interface Features

- *Sunburst View:* The sunburst view of a model provides a new way of thinking & exploring rules developed in a decision tree.



BigML Use Cases

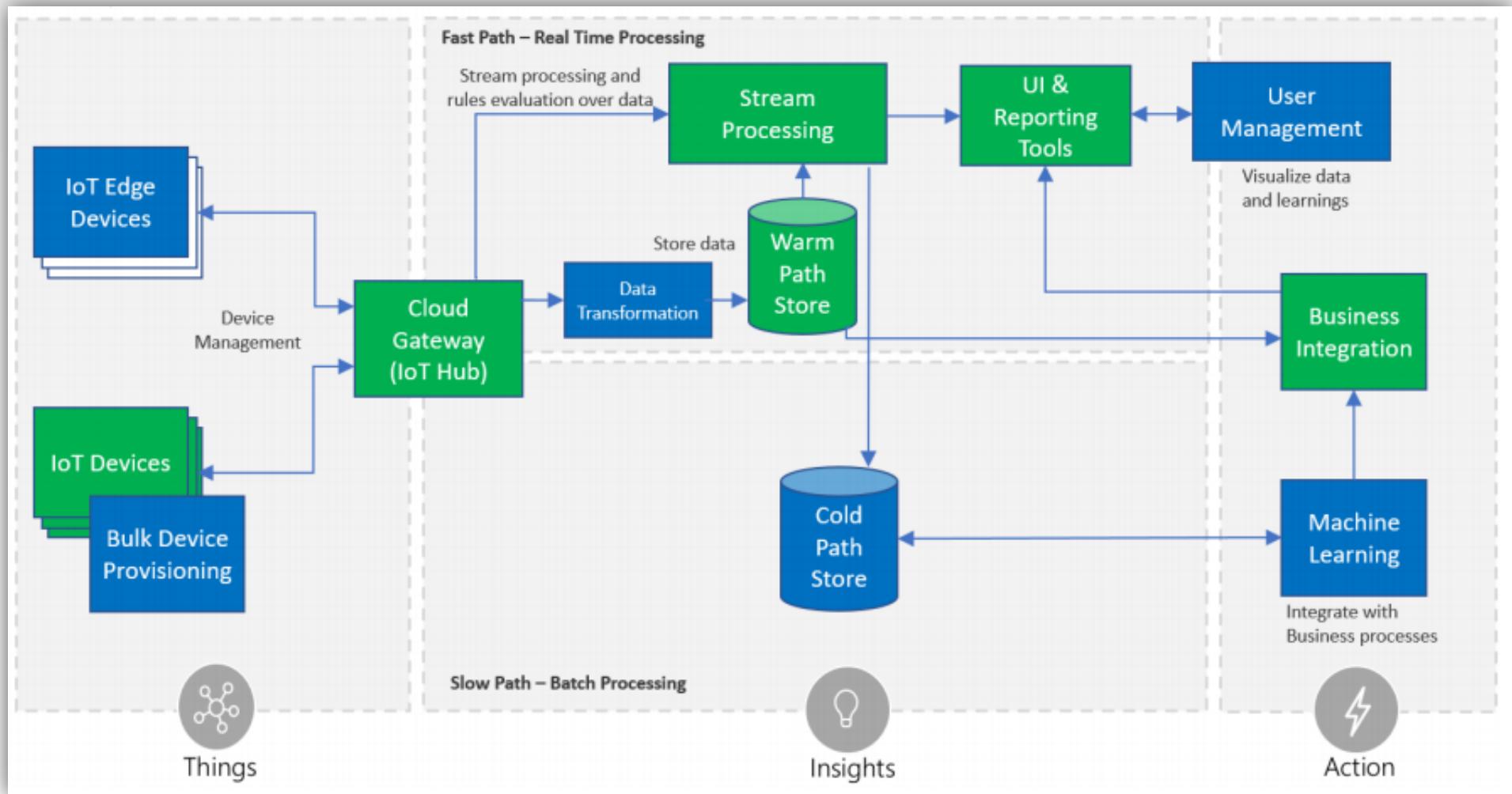
- ***Descriptive Model:*** Using the website, A business analyst can visualize complex data by constructing model to explain the relationships between attributes & predicted attribute or to play what-if scenarios.
- ***Predictive Model:*** A business analyst has complex problem that would like to predict from past examples. predictive model can be constructed on the website and predictions made in batch and downloaded as a CSV file for analysis and application.
- ***Periodic Predictions:*** When predictions are needed periodically in an ongoing manner. The model could be maintained on the BigML platform (updated when needed), called remotely to make predictions as needed via the Bigmler command line interface.
- ***Integration:*** The service could be integrated into a script, internal website or desktop application for decision support. This would require the use of API and the model would be best maintained on the platform.



Architectural Patterns For Big Data Systems

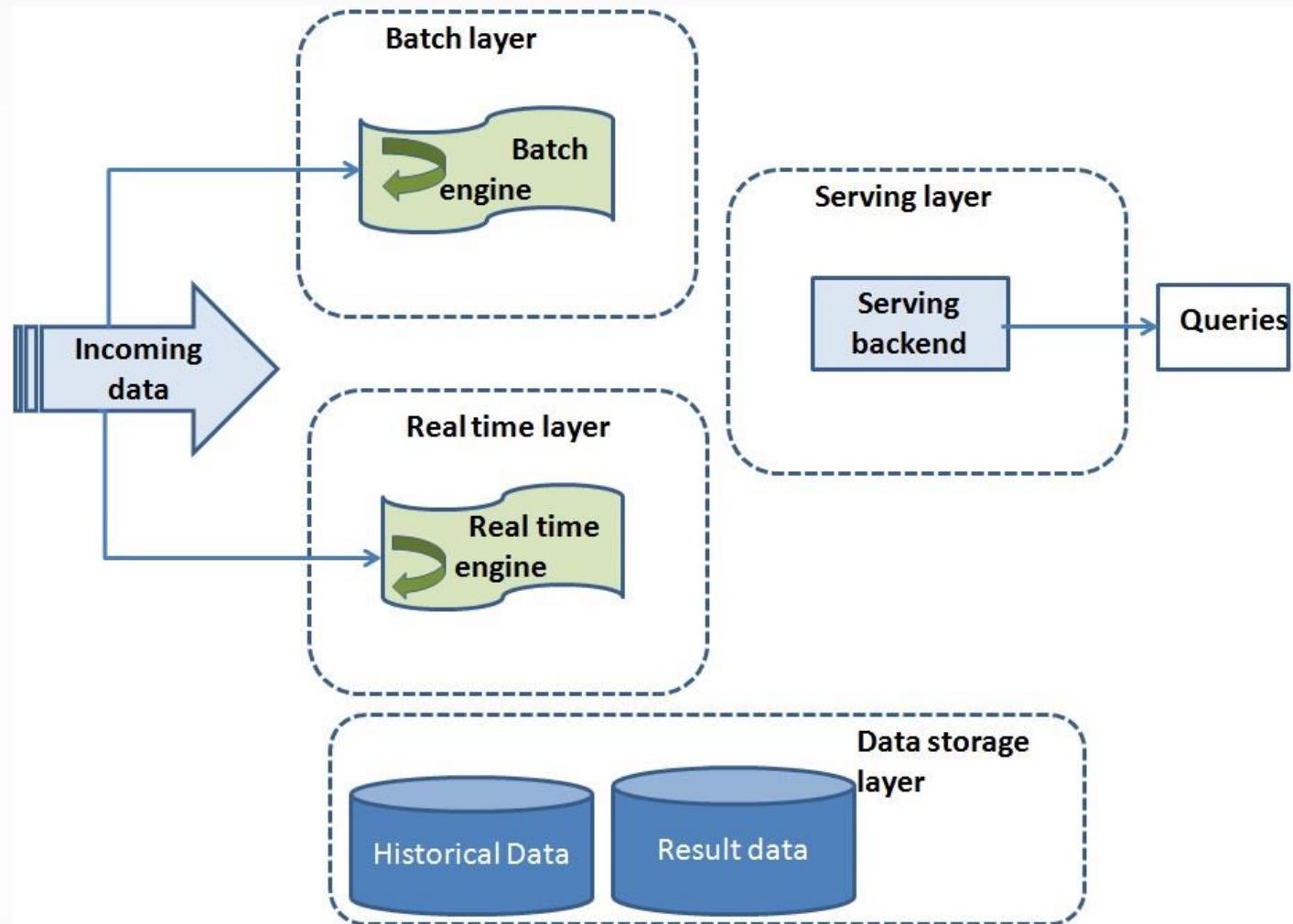


Lambda Architecture

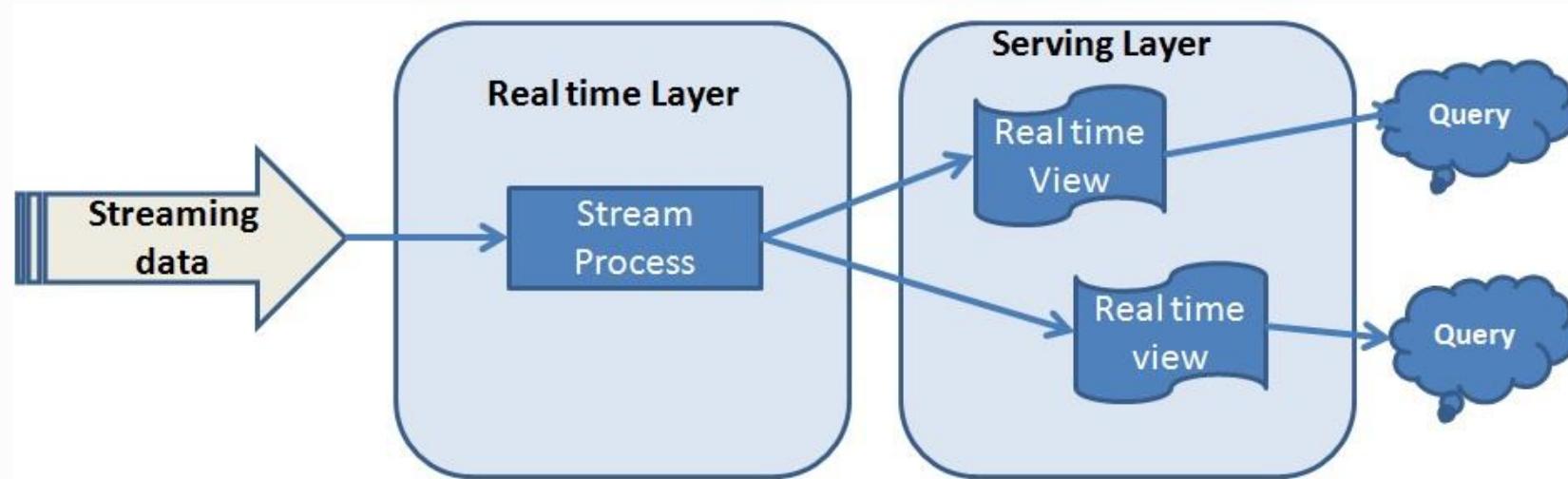


Source: http://download.microsoft.com/download/A/4/D/A4DAD253-BC21-41D3-B9D9-87D2AE6F0719/Microsoft_Azure_IoT_Reference_Architecture.pdf

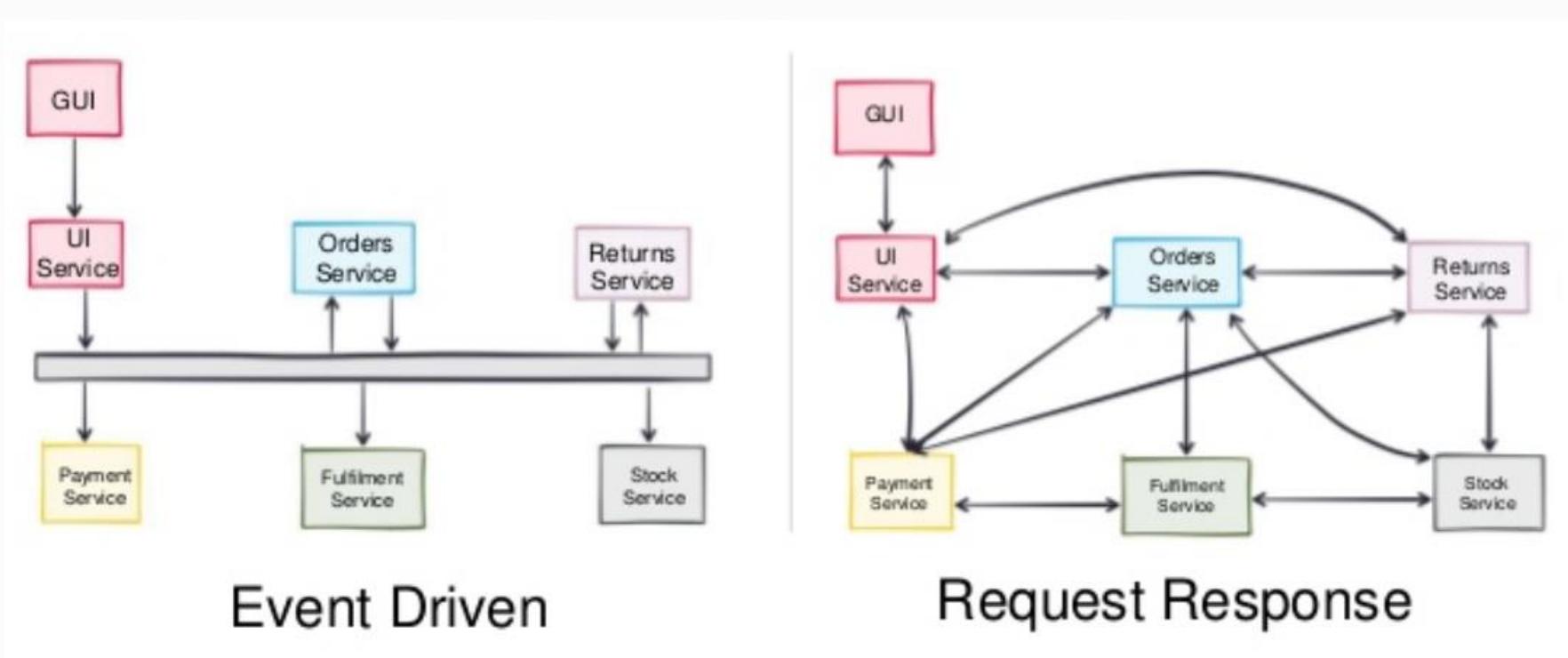
Lambda Architecture Essentials



Kappa Architecture Essentials



Connecting Services



REAL-TIME MESSAGE PROCESSING SYSTEMS

The Need for Kafka

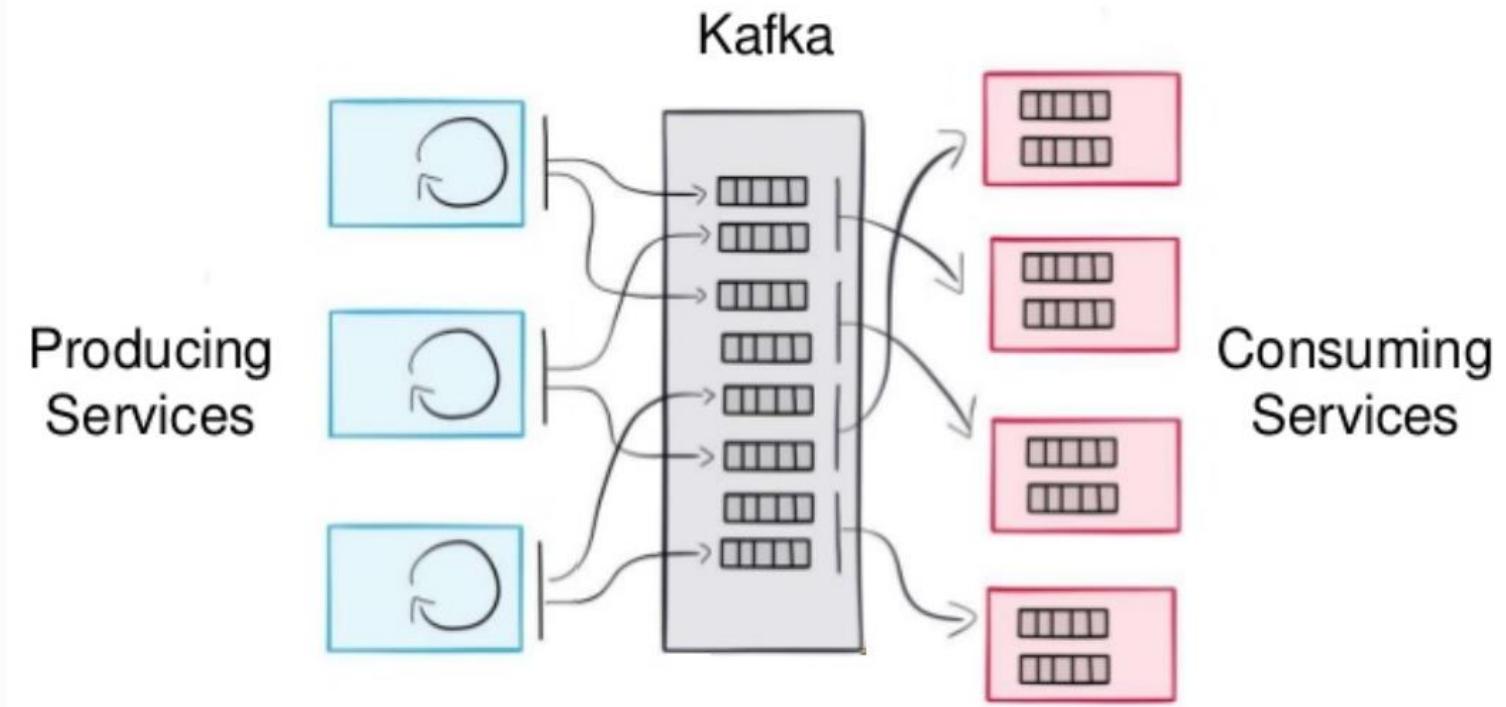
- Real-time message processing or stream processing systems are a need today. There are many examples in the world of IOT, Mobile push messaging, social media, etc.
- Heart of such systems is an enterprise class stream processing system such as Kafka, Amazon Kinesis and Azure event hub. They are usually persistent, fault tolerant and distributed.
- Business logic are modeled as micro services, each having its own database, mostly noSQL such as cassandra or hbase. Scaling needs of micro services are done through scale-out (horizontal farm – copy of the code – Scale cube – z axis).
- There will be message producing services and message consuming services connected to the stream processing system.
- Topics and Queues leveraged all over to ensure the event driven disconnected architecture.

Connecting Services

Scalable, Fault Tolerant, Concurrent, Strongly Ordered, Stateful

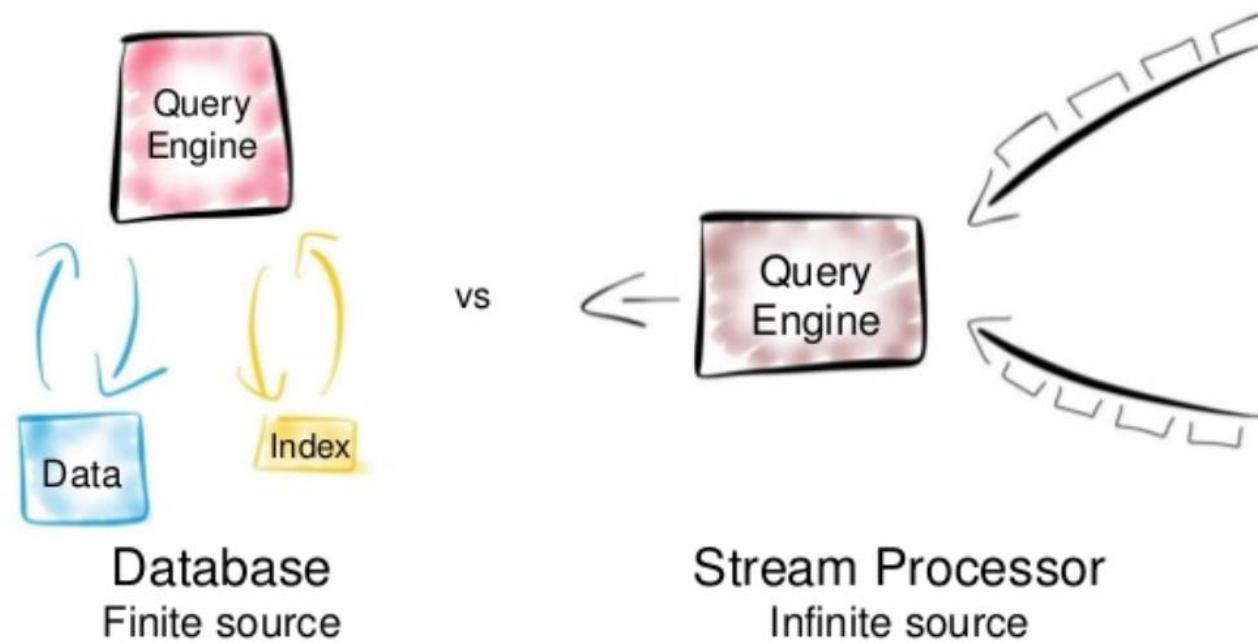


Connecting Services

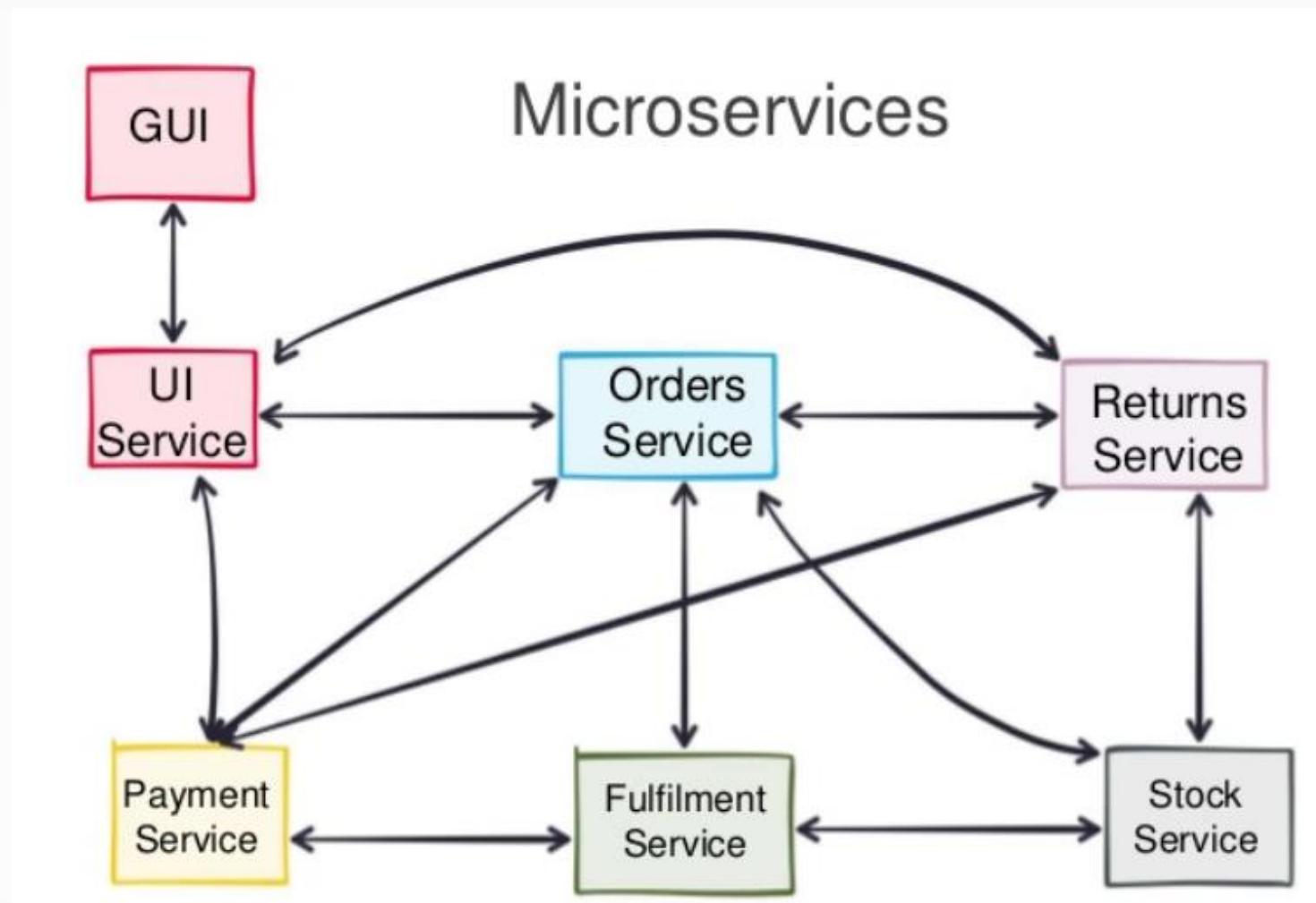


Stream Processing vs. DBMS

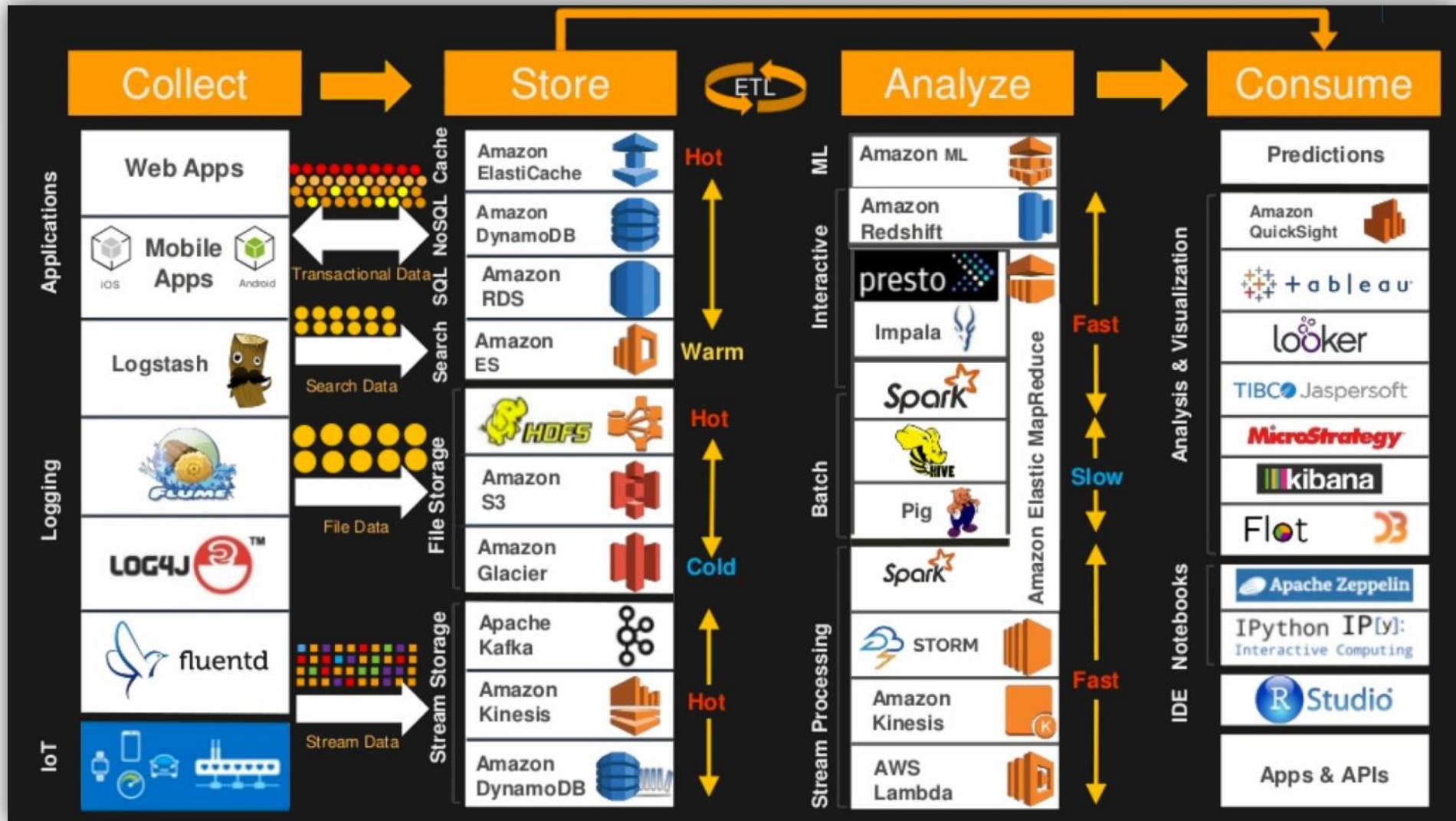
What is stream processing engine?



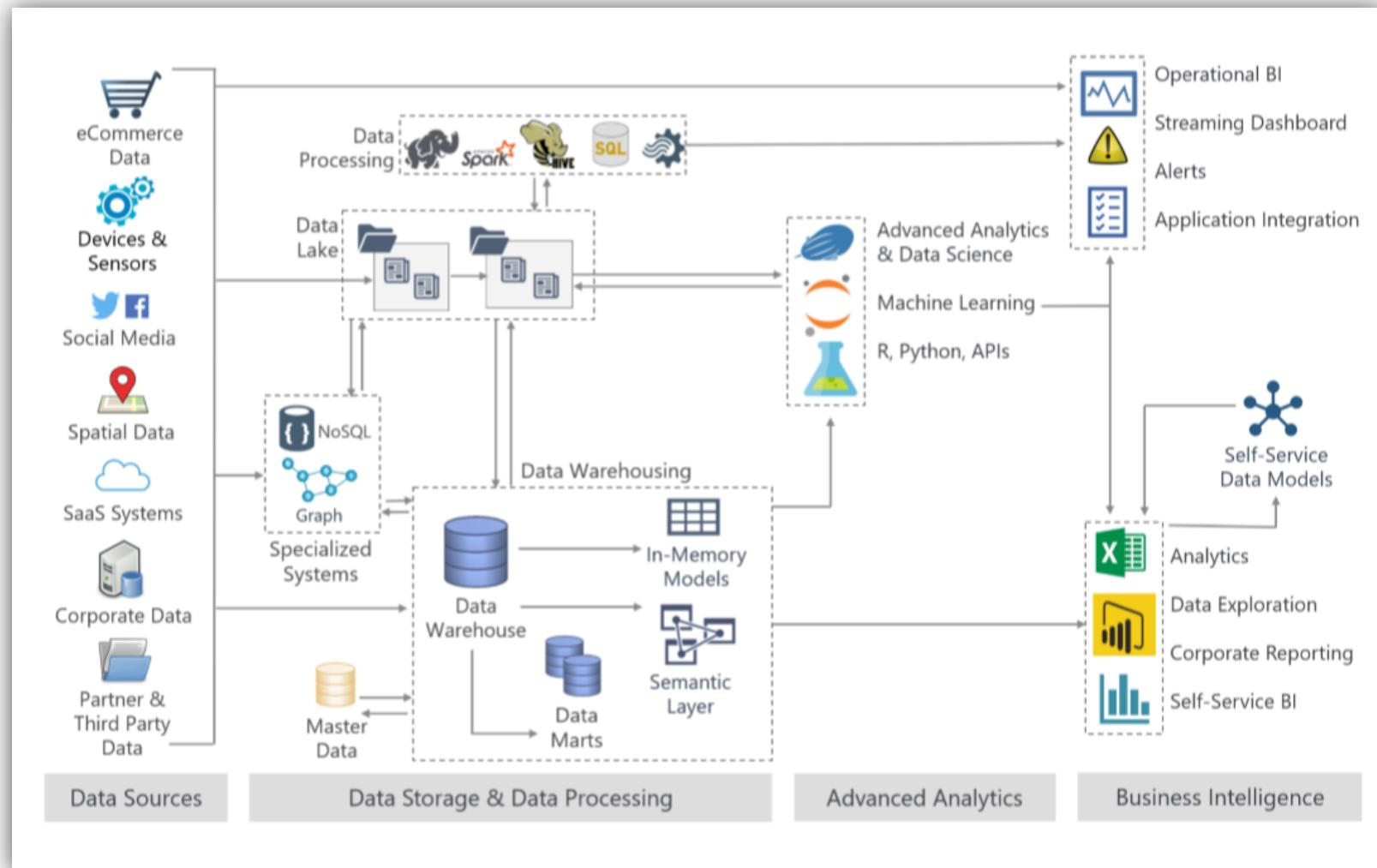
REAL-TIME MESSAGE PROCESSING SYSTEMS



Overall Reference Architecture



General Architecture



Node.js & Service Based Software Development

Yossi Zaguri
Yossiza@ariel.ac.il
052-4668866



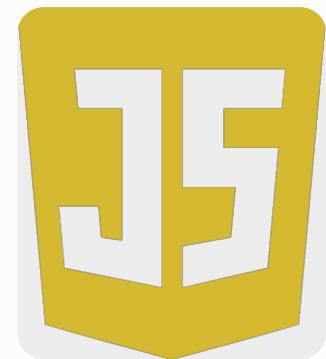
Topics

- What is Node.js
- Node.js ecosystem
- IoC, event loop and Callback methods
- HTTP & Hello world
- Express.js & Web API development
- Implementing MVC & Front Controller Patterns
- Implementing Pipes and Filters Pattern
- Incorporating MongoDB/DBaaS



Topics

- Implementing O.O & Functional paradigms
- O.O prototype concepts
- “this” binding methods
- JavaScript Closures
- Asynchronous programming methods
- Implementing Namespace & IIFE Patterns
- Introduction to Node.js internals
& code optimization

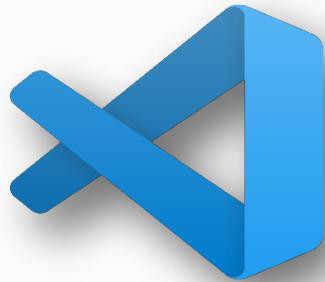


Topics

- Cloud & Service Orientation paradigm
- Web Services vs. Cloud Services
- In-Depth Micro Services Pattern
- Docker & building Node.js Image
- Serverless Development with Node.js
- Review of:
 - Message Broker
 - Gatekeeper
 - Gateway Routing
 - Gateway Aggregation
 - Event Sourcing
 - CQRS Patterns



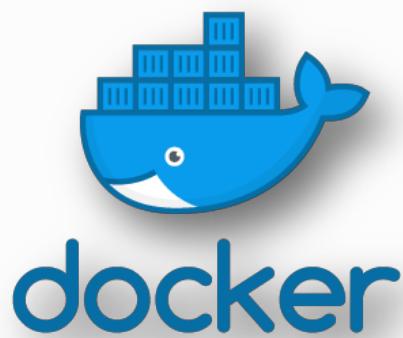
Before we start



<https://code.visualstudio.com/>



<https://nodejs.org/en/download/>



<https://docs.docker.com/desktop/windows/install/>



<https://www.mongodb.com/atlas/database>



Node.js



VS Code and Node.js



DEMO

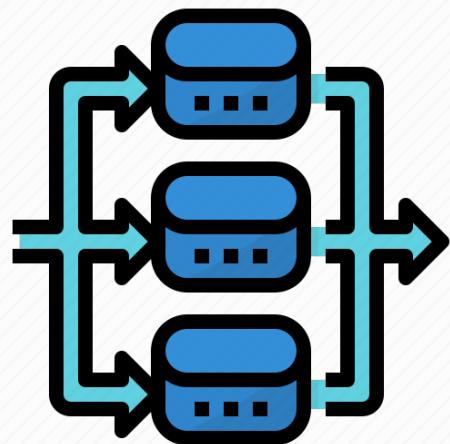
<https://code.visualstudio.com/docs/nodejs/nodejs-tutorial>

What is Node.js

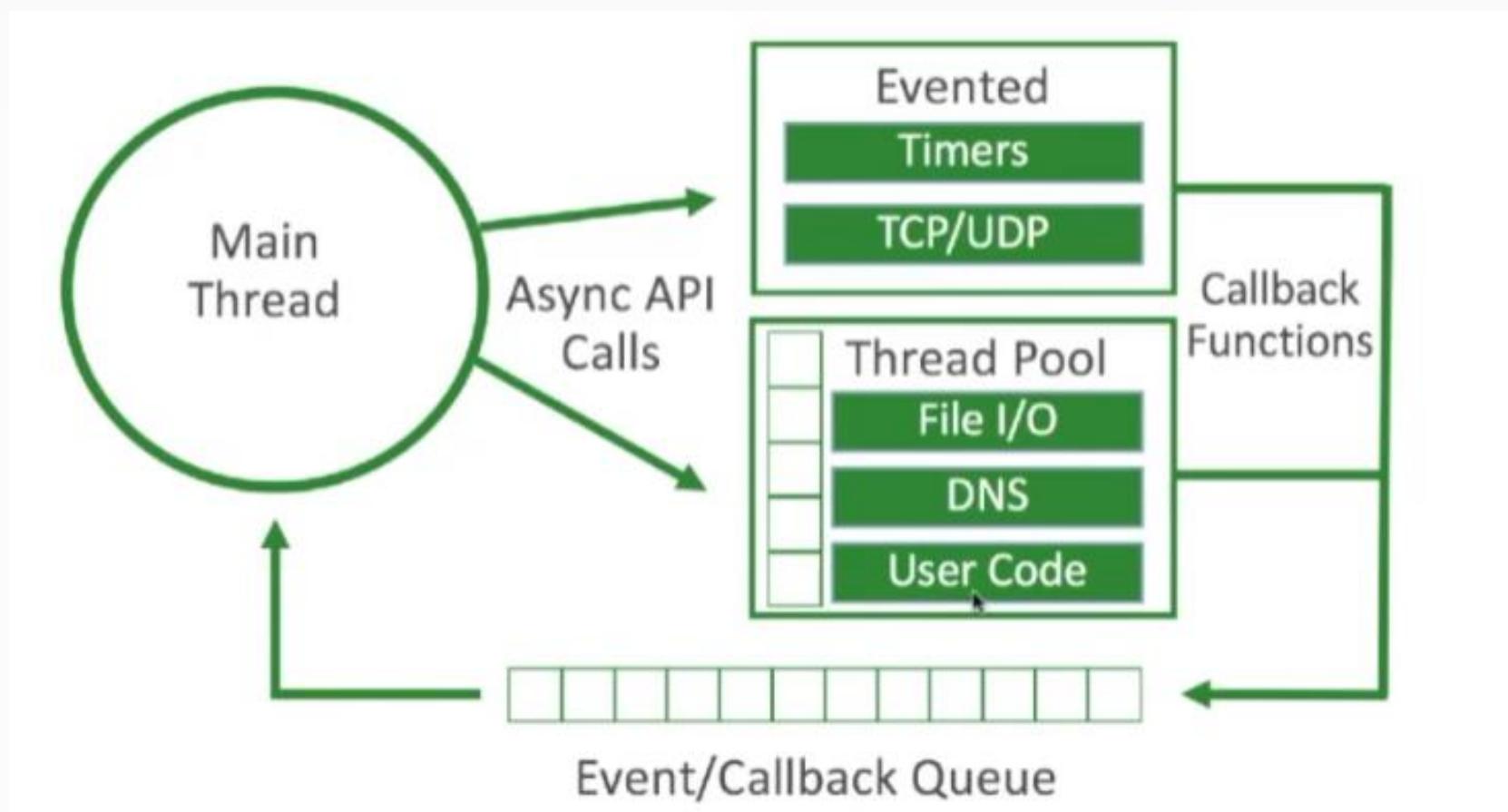
- ➊ Open-source & cross-platform JavaScript runtime environment, developed initially by Ryan Dahl.
- ➋ Runs the V8 JavaScript engine from Google Chrome.
- ➌ Runs in a single process, without creating a new thread for every request.
- ➍ Provides a set of asynchronous I/O primitives in its standard library that prevent JavaScript code from blocking.
- ➎ Libraries are written using non-blocking paradigms, making blocking behavior the exception rather than the norm.

What is Node.js

- ➊ I/O operations (network, access DB or FS), instead of blocking the thread and wasting CPU cycles waiting -> Node.js will resume operations ONLY when the response comes back.
- ➋ Thus Node.js can handle thousands of concurrent connections with a single server without the burdening programmers with thread concurrency management -> reducing this source of potential Bugs.



What is Event Loop

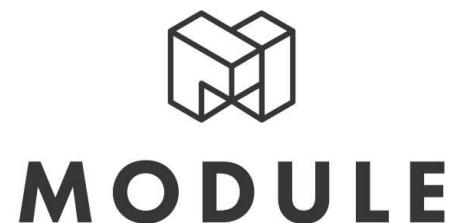


Event Loop & async

- When starting Node.js initializes an Event loop, allowing to perform non-blocking I/O operations despite the fact that it is single-threaded.
- Yet modern kernels are multi-threaded handling multiple operations by executing in the background.
- So Node.js Offers operations to the system kernel whenever possible
- When an operation completes, the kernel tells Node.js that the appropriate callback may be added to the **poll** queue to eventually be executed.

Node.js Modules

- In general : SoC Implementation via functions/classes/packages/sub systems etc.
- In Node.js : Logical (and Physical!) encapsulation of code in a single unit, managed as a separate unit of work.
- *Exports* a function or an object



Export & Import Module

```

1 var exports = module.exports={};

Defining a module

exports.AddNumber=function(a,b)
{
    2 return a+b;
};

3
Creating a function
in our module

Returning a value
back to the calling
function

```

```

var Addition=require('./Addition.js');

1 Using require to
include the Addition
module

2 console.log(Addition.AddNumber(1,2));
Calling the
AddNumber function
in our module

```

Callback functions

- In general : An Implementation of IoC pattern
- Objects creation and the flow of the application delegated to a container or framework
- Increases the modularity of the program
- Leads to Loose Coupling between software parts (modules)
- Grate way to develop libraries or Infrastructure

```
var callback = function(data) {  
    console.log('got data: '+data);  
};  
  
var doSomething = function(callback) {  
    callback('simpler is better');  
};  
  
doSomething(callback);
```



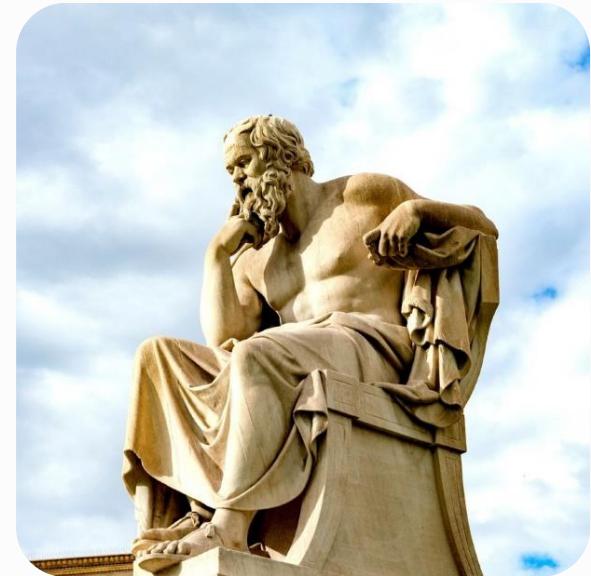
Node.js Dev Spirit



Node.js Philosophy

A set of principles and guidelines that are generally accepted by the community.

An ideology of doing things that influences the evolution of a platform, and how applications are developed and designed.



Node.js Philosophy

➊ Small core

- The rest is “User space” -> ecosystem of modules living outside the core

➋ Small modules

- Code size & Scope
- Physical *module* (file) is the fundamental means to structure the code
- Reusable libraries called *packages* and typically have well-focused dependencies.
- Solves “dependency hell” by making sure that each installed package will have its own separate set of dependencies, thus enabling a program to depend on a lot of packages without conflicts.

➌ Consequences:

- Easier to understand and use
- Simpler to test and maintain
- Perfect to share with the browser
- Boosting DRY (Don't Repeat Yourself)

Small is beautiful

Make each module do only one thing, yet do it well

Node.js Philosophy

➊ Focused Interfaces

- Node.js modules usually expose only a minimal set of functionality.
- Usually expose only one piece of functionality, such as a function or a constructor
- API becomes clearer to use and is less exposed to erroneous usage
- Advanced aspects or secondary features become properties of the exported Entity, helping the user to identify what is important and what is secondary.

➋ Modules usually are created to be “Aggregated” rather than extended

➌ Simplicity and pragmatism

- Takes less effort to implement
- Allows faster shipping with less resources
- Easier to adapt, understand and to maintain

“Favor Aggregation over Inheritance” (GOF)

“Simplicity is the ultimate sophistication.” (Leonardo da Vinci)



Managing Packages



NPM

- Npm is the package manager for Node. Registry hosts over 1,000,000 open-source packages.
- CLI tool that aids installing packages and manage versions & dependencies.
- Install using npm : `npm install <package>`
- Example : **npm install colors**
- Npm is also used to manage metadata about a project in a file named `package.json`
- To create `package.json` : **npm init**
- To run Package: **npm run package** (later)

Let's Try it

```
var colors = require('colors/safe');

console.log(colors.green('hello')) // outputs green text
console.log(colors.red.underline('i like cake and pies')) // red underlined text
console.log(colors.inverse('inverse the color')) // inverses the color
console.log(colors.rainbow('OMG Rainbows!')) // rainbow
console.log(colors.trap('Run the trap')) // Drops the bass
```



HTTP & REST



HTTP

- Client-server network protocol that made World-Wide Web what it is.
- Application-Level Protocol
- Peer-to-Peer communication
- Message Based (textual cross platform information units)
- Standard & Minimal Request types (called http verbs or method including :get , post, put, delete etc.)

HTTP Request & Response

```
GET /httpgallery/introduction/ HTTP/1.1
Accept: /*
Accept-Language: en-gb
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko
Host: www.httpwatch.com
Connection: Keep-Alive
```

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/8.0
Date: Mon, 04 Jan 2015 12:04:43 GMT
X-Powered-By: ASP.NET
X-AspNet-Version: 4.0.30319
Cache-Control: no-cache, no-store
Expires: -1
Content-Type: text/html; charset=utf-8
Content-Length: 14990

<!DOCTYPE html> <html>...
```

HTTP hello world

```
const http = require('http')

const hostname = '127.0.0.1'
const port = 3000

const server = http.createServer((req, res) => {
  res.statusCode = 200
  res.setHeader('Content-Type', 'text/plain')
  res.end('Hello World\n')
})

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`)
})
```

Whenever a new request is received, the [request event](#) is called, providing two objects a request (an [http.IncomingMessage](#) object) and a response (an [http.ServerResponse](#) object).

REST

- ➊ A set of architectural Principles/constraints to apply in network-based systems design
- ➋ Revolved around the concept of “Resource”
- ➌ Principles :
 - ➍ Give every “thing” an ID
 - ➎ Link things together
 - ➏ Use standard methods
 - ➐ Resources with multiple representations
 - ➑ Communicate statelessly
- ➒ RESTful API is one of the most popular Web Services Interface type



Express.js



What is Express.js

- Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.
- With a myriad of HTTP utility methods and middleware at your disposal, creating a robust API is quick and easy.
- Many popular frameworks are based on Express.



Express “hello world”

```
md myapp
```

```
cd myapp
```

```
npm init
```

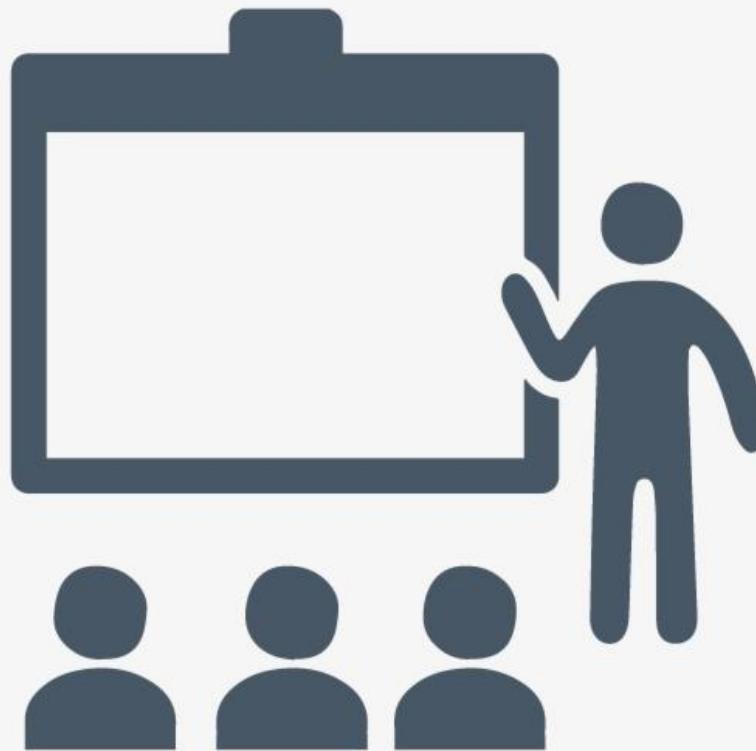
```
npm install express
```

```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.listen(port, () => {
  console.log(`Example app listening at http://localhost:${port}`)
})
```

Live Demos



<https://expressjs.com/>

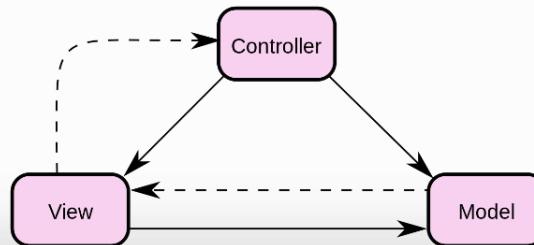


MVC

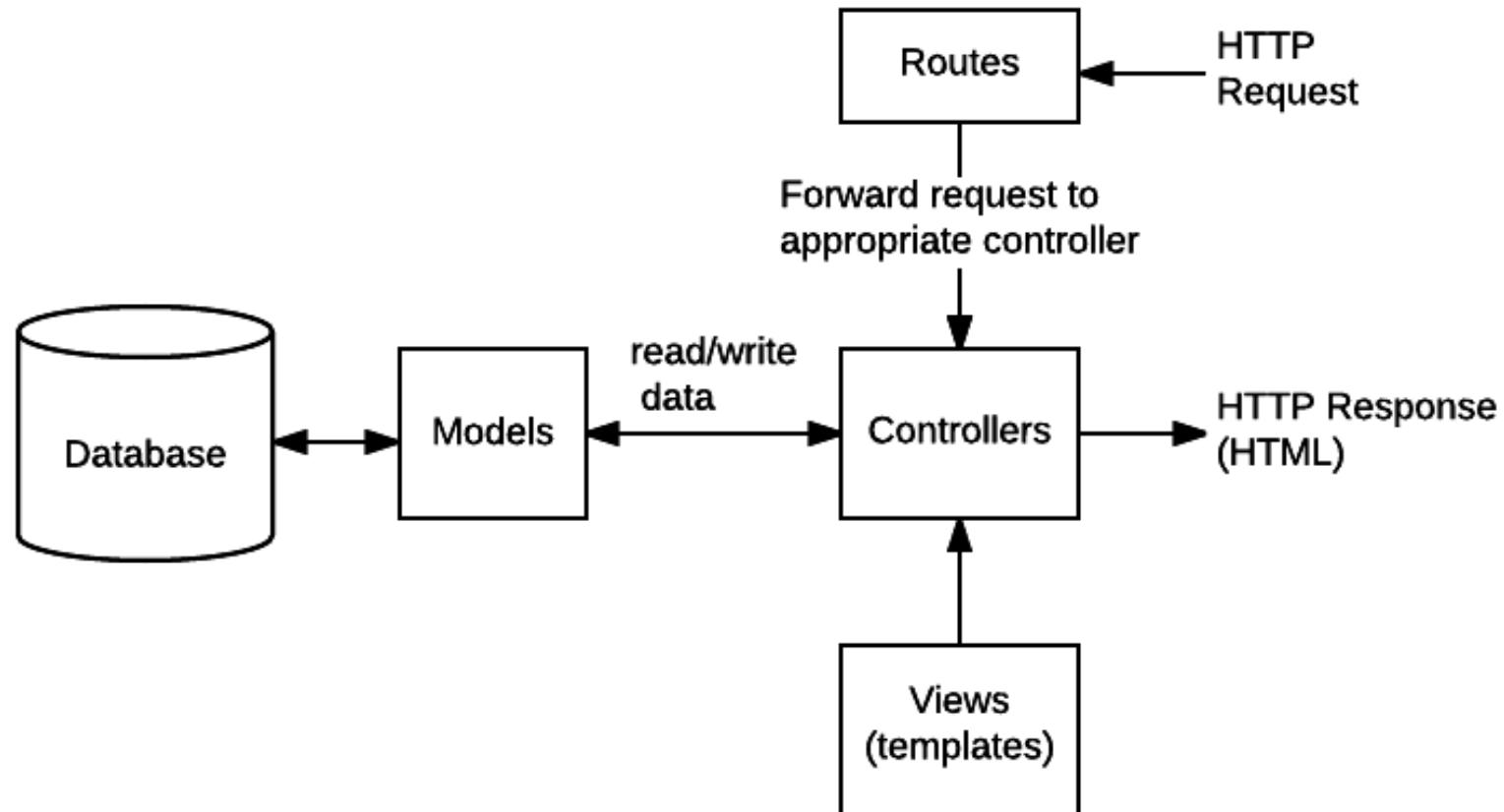


What is MVC

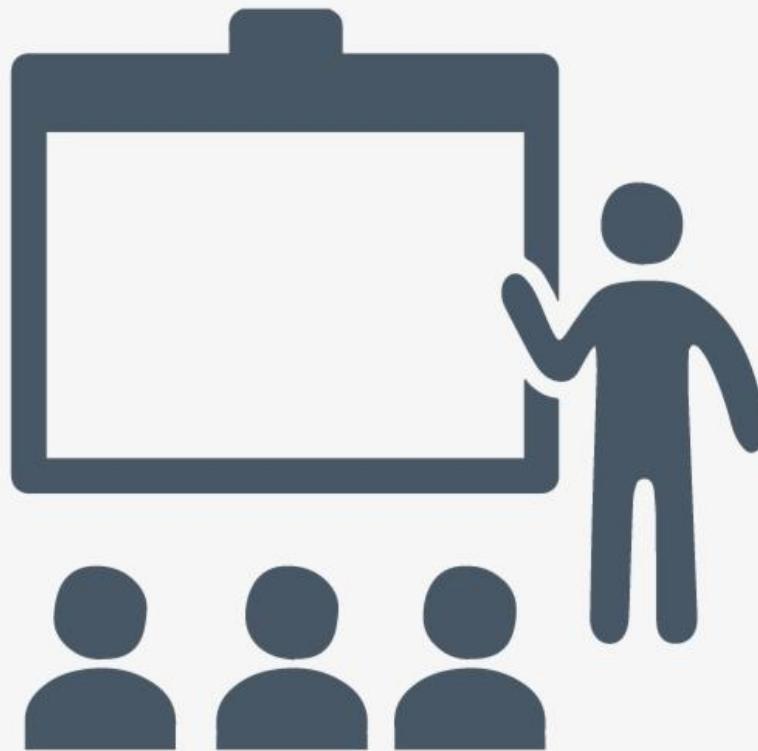
- Model-View-Controller (MVC) is one of the most popular architectural patterns (Is it an architecture indeed ?)
- We can describe the MVC architecture in simple terms:
 - Model*: The **Module** that deals with the database or any data-related functionality.
 - View*: The **Module** responsible of the interaction/display with the user. In our case it will be a VIEW ENGINE.
 - Controller: The Module holding the main logic and managing execution flow . It will get a request, call models to get the data, then send the data to the view engine, getting html document to be sent to the user.



Flow of processing



Live Demos



<https://expressjs.com/>

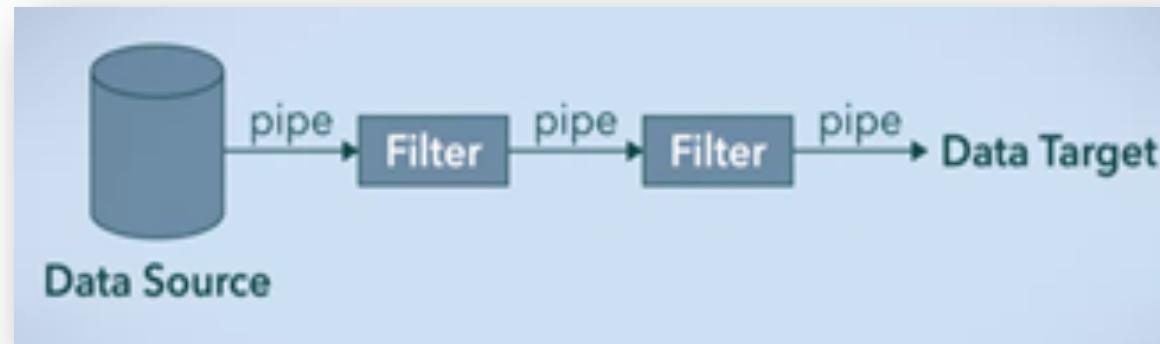


Pipeline Pattern



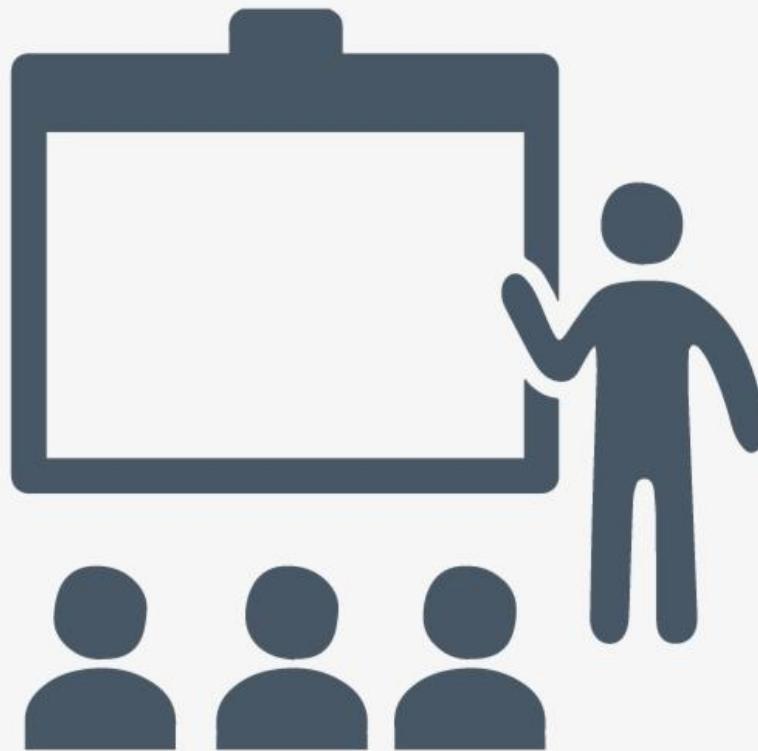
What are Pipes & Filters

- Architectural pattern having independent entities called *filters* (components) which perform transformations on data processing the input data, and *pipes*, which serve as connectors for the stream of data being transformed, each connected to the next component in the pipeline.
- Decompose a task that performs complex processing into a series of separate elements that can be reused. This can improve performance, scalability, and reusability by allowing task elements that perform the processing to be deployed and scaled independently.



<https://docs.microsoft.com/en-us/azure/architecture/patterns/pipes-and-filters>

Live Demos



<https://expressjs.com/en/guide/writing-middleware.html>

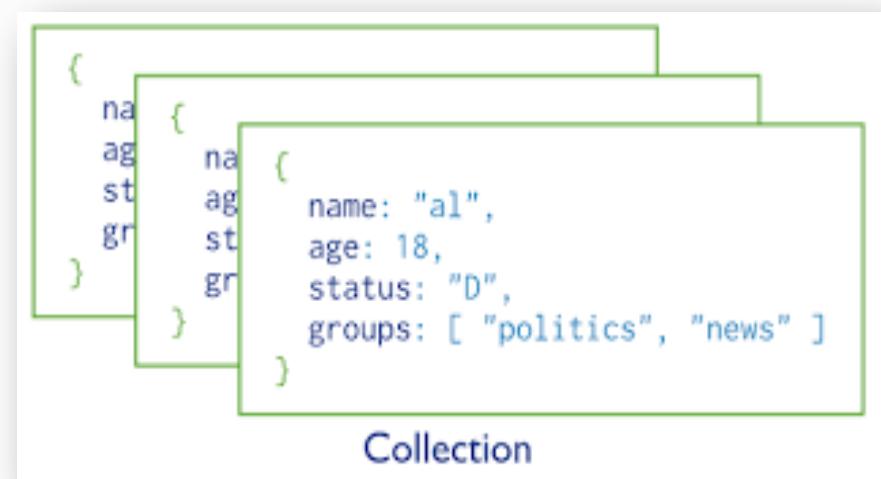
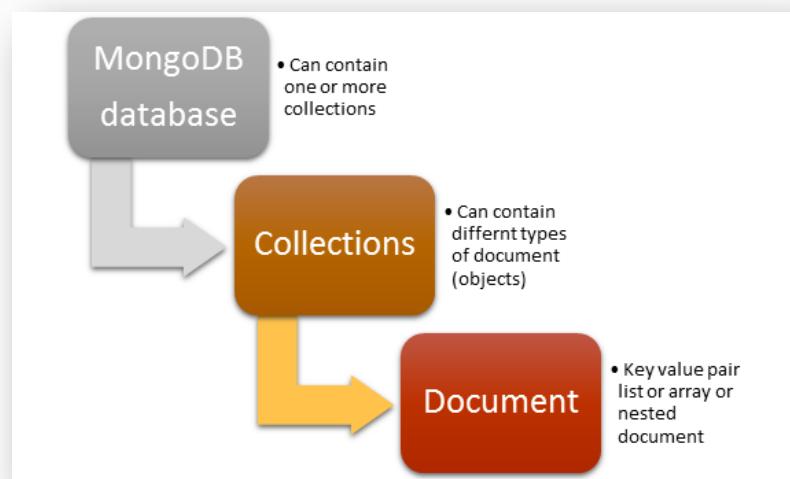


Implement Models using MongoDB



What Is MongoDB

- (Very)Popular NoSQL DBMS.
- DB → Collection → Document
- Document is basically a json entity
- (Very) Highly Scalable



Create DB

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/mydb";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  console.log("Database created!");
  db.close();
});
```

To work using Schemas install mongoose instead
of mongodb driver

Create Collection

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  dbo.createCollection("customers", function(err, res) {
    if (err) throw err;
    console.log("Collection created!");
    db.close();
  });
});
```

Follow ing w 3schools.com

Create Collection

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  dbo.createCollection("customers", function(err, res) {
    if (err) throw err;
    console.log("Collection created!");
    db.close();
  });
});
```

Follow ing w 3schools.com

Insert Object

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  var myobj = { name: "Company Inc", address: "Highway 37" };
  dbo.collection("customers").insertOne(myobj, function(err, res) {
    if (err) throw err;
    console.log("1 document inserted");
    db.close();
  });
});
```

Follow ing w 3schools.com

Insert Array

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  var myobj = [
    { name: 'John', address: 'Herzelia'},
    { name: 'Peter', address: 'Ramat Gan'},
    { name: 'Amy', address: 'Arad'}
  ];
  dbo.collection("customers").insertMany(myobj, function(err, res) {
    if (err) throw err;
    console.log("Number of documents inserted: " + res.insertedCount);
    db.close();
  });
});
```

The Result Object

```
{  
  result: { ok: 1, n: 3 },  
  ops: [  
    { name: 'John', address: 'Herzelia'},  
    { name: 'Peter', address: 'Ramat Gan'},  
    { name: 'Amy', address: 'Arad'},,  
  ],  
  insertedCount: 3,  
  insertedIds: [  
    58fdbf5c0ef8a50b4cdd9a84,  
    58fdbf5c0ef8a50b4cdd9a85,  
    58fdbf5c0ef8a50b4cdd9a86  
  ]  
}
```

Data Query

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  var query = { address: "Arad" };
  dbo.collection("customers").find(query).toArray(function(err, result) {
    if (err) throw err;
    console.log(result);
    db.close();
  });
});
```

Data Projection

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  dbo.collection("customers").find({}, { projection: { _id: 0, name: 1, address: 1
} }).toArray(function(err, result) {
  if (err) throw err;
  console.log(result);
  db.close();
});
});
```

Follow ing w 3schools.com

Delete Data

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  var myquery = { address: /^O/ };
  dbo.collection("customers").deleteMany(myquery, function(err, obj) {
    if (err) throw err;
    console.log(obj.result.n + " document(s) deleted");
    db.close();
  });
});
```

Follow ing w 3schools.com

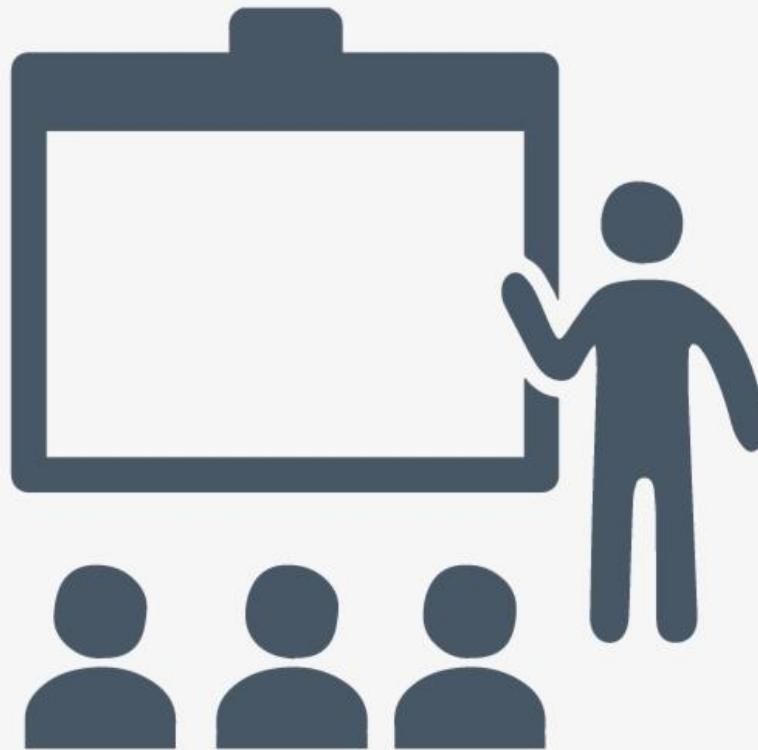
Update Data

```
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://127.0.0.1:27017/";

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("mydb");
  var myquery = { address: "Arad" };
  var newvalues = { $set: {address: "Negev" } };
  dbo.collection("customers").updateOne(myquery, newvalues, function(err,
res) {
    if (err) throw err;
    console.log("1 document updated");
    db.close();
  });
});
```

Follow ing w 3schools.com

Live Demos



<https://zellwk.com/blog/crud-express-mongodb/>



JS – Intro & Programming Paradigms



Built In Types - primitive types

Boolean

- false => 0, -0, null, false, NaN, undefined , empty string ("")
- true => everything else , including “false” string or any object

Number

- 64-bit floating point
- No integer type
- Includes special values NaN & Infinity

String

- No character type
- Literals quoted using the ' or "
- Immutable
- Escapement is done with the “\” character

Built In Types - Primitive types

Null

- The Null type has exactly one value: null

BigInt

- Represent integers with arbitrary precision
- Safely store and operate on large integers even beyond the safe integer limit for Numbers (Number.MAX_SAFE_INTEGER)
- `const y = x + 1n => 9007199254740993n`

Symbol

- Unique and immutable primitive value
- `let Sym1 = Symbol("Sym")`
- Must cast in order to treat as string => `console.log(Sym.toString())`
- `Symbol.keyFor(Symbol.for("tokenString")) === "tokenString" // true`

Built In Types - Special Values

➊ Special values

- ➊ null – An object
 - ➊ value represents a reference that points to a nonexistent or invalid object or address.
 - ➊ null is not as "primitive" as it first seems although its behavior is seemingly primitive: Every Object is derived from null value, and therefore typeof operator returns object : -> (typeof null) === 'object' // true
- ➊ undefined - a privative , it's type is Undefined
- ➊ undefined is the lack of a type and value, null is the lack of a value
- ➊ Common conceptual Mistake
 - ➊ if (foo == null) { alert('foo really has no type. Object imposed'); }
 - ➊ if (foo === undefined) { alert('foo has undefined value.'); }

➋ Everything else is variations on object type:

- ➊ Dates, Regular Expressions & More added by engine implementation These are really just objects

Language Concepts - Global Object

- The global object is essential, and you can't avoid it.
- It contains things available on every module like require, exports, and __dirname
- It also contains timer functions like setTimeout and setInterval as well as your favorite, the console
- On the browser, you create a global variable (with var) which will be available everywhere (global scope)
- With NodeJs things are different, *the top level is the module itself*, any “global” variable you create is local to the module and you choose what the module exports.

Language Concepts - Hashtable

- ⌚ JS is loosely typed → don't use type names in declarations
- ⌚ Define Hash Table:
 - ⌚ var x = {};
- ⌚ Subscript notation to add, replace, retrieve elements:
 - ⌚ x["name"] = "Rina Levi";
- ⌚ Dot notation is more convenient :
 - ⌚ x.city = "Tel Aviv";
- ⌚ Can be used when subscript is string constant in form of a legal identifier
- ⌚ Because objects & hashtables are the same, this is Equal to above :
`var x = new Object()`

Object ↔ Hashtable ↔ Array

Language Concepts - Hashtable

- Member enumeration capability *built* into the **for..in statement**

```
for (var n in x) {  
    if (x.hasOwnProperty(n))  
        alert(x[n])  
}
```

Language Concepts - Object

- Object → Referenceable container of name/value pairs.
- Object literal notation: object description with a set of comma-separated name/value pairs inside curly braces
 - Names can be identifiers or strings followed by a colon
 - Values can be literals or expressions of any type

```
var Obj = {name: "Rina", 'last': 'Levi', city: 'TA', level: 20};
```

- New members can be added to any object any time by assignment:
`Obj.nickname = 'Rinale';`

Object ↔ Hash Table ↔ Array

Language Concepts - Arrays

- ➊ Two ways to make a new array:
 - ➊ var myArray = []
 - ➋ var myArray = new Array()
- ➋ Arrays implemented as objects
- ➋ Arrays are not typed.
 - ➊ Can contain numbers, strings, booleans, objects, functions & arrays
- ➋ You can mix strings, numbers & objects in the same array
- ➋ Arrays literal notation:
 - ➊ myList = ['oats', 'peas', 'beans', 'barley'];
 - ➋ slides = [{url: 'slide1.html', title: 'First'}, {url: 'slide2.html', title:'Second'}];

Object ↔ Hash Table ↔ Array

Language Concepts - Arrays

- New item can be added by assignment:

- `x[2]=30`
- `a[i + j] = f(a[i], a[j])`
- `var myarray=[x+y, 2, Math.round(z)]`

- Undefined values allowed :

- `var myarray=[0,,,5]`

- Multi Dimensional Array:

- `var myarray=[["New York", "LA", "Seattle"], China, Japan]`
- `myarray[0][1]` -> returns "LA"

Object ↔ Hash Table ↔ Array

Language Concepts – Function

Function = constructor

- var Average = new Function("x", "y", "return (x + y)/2");
- Less efficient than declaring function
- Actually Average is An object that has a value
- Function object can be called just as if it were a function by specifying the variable name

Function != function

Language Concepts - function

• *function* = operator

- Declare block of code as in “other” languages
- You CAN pass a fixed number of parameters
- Excess parameters are ignored
- function has access to “arguments” array
- functions are First class objects
- Functions have lexical scoping - inner level can access outer levels
- Can hold their own Data & Methods , acting like a class
- Inner functions (Closures) & anonymous functions EXIST
- Every function returns a value. default “undefined”, constructors=> this

Function != function

Language Concepts – prototype

- ➊ Objects contain a hidden link property pointing to the prototype member of the constructor of the object
- ➋ Prototype link chains:
 - ➌ When items accessed from object by dot notation or subscript notation, if item not found in the object then the link object is examined all the way up, returning “undefined” if not found
 - ➍ Provides sort of inheritance
- ➎ Members can be added to the prototype by assignment

```
function Demo() {}  
Demo.prototype = new Ancestor();  
Demo.prototype.foo = function () {};
```

Language Concepts - this

- ➊ sayHello()
 - ➊ this => global object
- ➋ person.sayHello()
 - ➊ this=> person
- ➌ new person()
 - ➊ this=> the new object
- ➍ Arrow functions Does not have its own bindings to *this* (or super)
- ➎ Use call / apply / bind methods to adjust referencing

```
const materials = [ 'Hydrogen', 'Helium', 'Lithium', 'Beryllium']

console.log(materials.map(material => material.length));
```

Language Concepts – closure

- ➊ An expression (typically a function) that can have free variables together with an environment that binds those variables
- ➋ Using the *function* keyword inside another function creates a closure !

```
function sayHello(name) {  
    var text = 'Hello ' + name;  
    var sayAlert = function() { alert(text); }  
    sayAlert();  
}  
SayHello("rina");
```

Language Concepts – closure

```
function sayHello (name) {  
    var text = 'Hello ' + name; // local variable  
    var sayAlert = function() { alert(text); }  
    return sayAlert;  
}  
var sayHelloAgain=sayHello('rina');  
sayHelloAgain();
```

```
function sayHello(name) {  
    var text = 'Hello ' + name;  
    var sayAlert = function() { alert(text); }  
    sayAlert();  
}
```

Language Concepts – closure

```
function setupSomeGlobals() {  
    // Local variable ends up within closure  
    var num = 100;  
    // Store references to functions as global variables  
    gAlertNumber = function () { alert(num); }  
    gIncreaseNumber = function () { num++; }  
    gChangeNumber = function (x) { num=x; }  
}
```

```
setupSomeGlobals();  
gChangeNumber(200);  
gAlertNumber();  
gIncreaseNumber();  
gAlertNumber();
```

```
setupSomeGlobals();  
gAlertNumber();  
gIncreaseNumber();  
gAlertNumber();
```

num is shared among all functions
Each call created new set of variables (closure)

Language Concepts – closure

```
function newClosure(someNum, someRef) {  
    var num = someNum;  
    var anArray = [1, 2, 3];  
    var ref = someRef;  
    return function (x) {  
        num += x;  
        anArray.push(num);  
        alert('num:' + num +',anArray:' + anArray.toString()  
            +',ref.someVar:' + ref.someVar);  
    }  
}  
  
var firstClosure = newClosure(10, { someVar: 'first Closure' });  
var secondClosure = newClosure(20, { someVar: 'second Closure' });  
firstClosure(100);  
secondClosure(200);
```

No closure per function definition, rather per call
Each call creates new closure

Language Concepts – closure

```
function buildList(list) {
    var result = [];
    for (var i = 0; i < list.length; i++) {
        var item = 'item' + list[i];
        result.push(function () { alert(item); });
    }
    return result;
}
function testList() {
    var fnlist = buildList([1, 2, 3]);
    for (var j = 0; j < fnlist.length; j++) {
        fnlist[j]();
    }
}
testList();
```

```
function sayHello() {
    var sayAlert = function ()
        { alert('Hello'); }
    var Hello = 'שלום';
    return sayAlert;
}
sayHello();
```

Loop Warning: i & item have only single copy
 Variable declaration order not important



JS – Object Orientation



Create Object - Literally

```
var person = {
    firstName:'Dani',
    lastName: 'Choen',

    //method
    getFunction : function(){
        return (`The name of the person is ${person.firstName} ${person.lastName}` )
    },
    //object within object
    phoneNumber : {
        mobile:'12345',
        landLine:'6789'
    }
}
console.log(person.getFunction());
console.log(person.phoneNumber.landLine);
```

Create Object – Object Inheritance & Cloning

```
const coder = {  
    isStudying : false,  
    profession:'programmer',  
    printIntroduction : function(){  
        console.log(`My name is ${this.name}. Am I studying?  
        ${this.isStudying}.`)  
    }  
}  
  
const me = Object.create(coder);  
  
// "name" is a property set on "me", but not on "coder"  
me.name = 'Dani';  
me.isStudying = true;  
console.log(me.profession);
```

```
let objClone = { ...obj }; // pass all key:value pairs from an object
```

Object.create() method creates a new object, using an existing object as the *prototype* of the newly created object.

Create Object - constructor

```
//using a constructor
function person(first_name,last_name){
    this.first_name = first_name;
    this.last_name = last_name;
}
//creating new instances of person object
let person1 = new person('Dani', 'Choen');
let person2 = new person('Rahul',Ganon');

console.log(person1.first_name);
console.log(` ${person2.first_name} ${person2.last_name}`);
```

Create Object – Class

```
class Vehicle {
    constructor(name, maker, engine) {
        this.name = name;
        this.maker = maker;
        this.engine = engine;
    }
    getDetails(){
        return (`The name of the bike is ${this.name}.`)
    }
}
// Making object with the help of the constructor
let bike1 = new Vehicle('Hayabusa', 'Suzuki', '1340cc');
let bike2 = new Vehicle('Ninja', 'Kawasaki', '998cc');

console.log(bike1.name);      // Hayabusa
console.log(bike2.maker);     // Kawasaki
console.log(bike1.getDetails());
```

Unlike other Object Oriented Language there is no classes in JavaScript - only Object. JavaScript is a prototype based object oriented language, which means it doesn't have classes rather it define behaviors using constructor function and then reuse it using the prototype.

It is primarily syntactical sugar over JavaScript's existing prototype-based inheritance.

Create Object - Class Inheritance

```
class person{
    constructor(name){
        this.name = name;
    }
    // method to return the string representing the object
    toString(){
        return (`Name of person: ${this.name}`);
    }
}
class student extends person{
    constructor(name,id){
        //super keyword to for calling above class constructor
        super(name);
        this.id = id;
    }
    // function override
    toString(){
        return (`${super.toString()},Student ID: ${this.id}`);
    }
}
let student1 = new student('Mukul',22);
console.log(student1.toString());
```

Create Object - static Constructor

```
class ClassWithStaticInitializationBlock {  
    static staticProperty1 = 'Property 1';  
    static staticProperty2;  
    static { this.staticProperty2 = 'Property 2'; }  
}
```

```
// output: "Property1"  
console.log(ClassWithStaticInitializationBlock.staticProperty1);
```

```
// output: "Property 2"  
console.log(ClassWithStaticInitializationBlock.staticProperty2);
```

Module returning Object

```
// These variables will stay in the local scope of this module (person.js)
var firstName, lastName, age;

// Make sure your argument name doesn't conflict with variables set above
exports.setFirstName = function (fname) {
    firstName = fname;
};

exports.setLastName = function (lname) {
    lastName = lname;
};

exports.setAge = function (yrsold) {
    age = yrsold;
};

// You're returning an object with property values set above
exports.getPersonInfo = function () {
    return {
        firstName:firstName,
        lastName:lastName,
        age:age
    };
};
```

```
var person = require('./person.js');
person.setFirstName('Steve');
person.setLastName('Jobs');
person.setAge(56);

// Outputs first name, last name, and age as
// an object literal
console.log(person.getPersonInfo());
```

Getter & Setter

```
let obj = {
  get propName() {
    console.log("get")
  },
  set propName(value) {
    console.log("set")
  }
};
```

```
let user =
{
  name: "John",
  surname: "Smith",
  get fullName()
  {
    return `${this.name} ${this.surname}`;
  },
  set fullName(value) {
    [this.name, this.surname]=value.split(" ");
  }
}

user.fullName = "Alice Cooper"
console.log(user.name)
console.log(user.surname)
```

Accessor descriptors

```
let user = {  
    name: "John",  
    surname: "Smith"  
};  
  
Object.defineProperty(user, 'fullName', {  
    get() {  
        return `${this.name} ${this.surname}`;  
    },  
  
    set(value) {  
        [this.name, this.surname] = value.split(" ");  
    }  
});  
  
console.log(user.fullName); // John Smith  
  
for(let key in user) console.log(key)
```

Spread syntax (...)

```
function sum(x, y, z) {  
    return x + y + z;  
}  
  
const numbers = [1, 2, 3];  
  
console.log(sum(...numbers));  
// expected output: 6
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_syntax



JS – Functional Programming



Functional Programming

- Functional Programming describes only operations
- No use of temporary variables to store intermediate results
- Emphasis is to capture "what and why" rather than the "how"
- Emphasizes definition of functions rather than the implementation of state machines, in contrast to procedural programming, which emphasizes execution of sequential commands.
- Knowledge Management systems benefit greatly as it simplifies development
- *Concepts:* Anonymous functions, different ways to call functions, pass functions as arguments to other functions.

Functional Programming

Classic

```
function sum(x,y,z) {  
    return (x+y+z); }
```

Anonymous

```
function(x,y,z) {  
    return (x+y+z); }
```

- ➊ var sum = function(x,y,z) { return (x+y+z); }(1,2,3); console.log(sum);
- ➋ (console.log)((function (x, y, z) { return (x + y + z) })(1, 2, 3));
- ➌ function as an expression

- ➍ function calc(action,data) { console.log(action(data));} calc(function (x){ return x*2;},5);
 - ➎ function as first class object
- ➏ Hence:
 - ➐ Functions need not have names all the time.
 - ➑ Functions can be assigned to variables like other values.
 - ➒ A function expression can be written and enclosed in parentheses
 - ➓ Functions can be passed as arguments to other functions.

Functional Programming

Too Static

```
function printArray(array) {  
  for (var i = 0; i < array.length; i++)  
    console.log(array[i]);  
}
```

More General

```
function forEach(array, action)  
{ for (var i = 0; i < array.length; i++)  
  action(array[i]);  
}  
forEach(["rina", "dina"], alert);
```

```
function sum(numbers) {  
  var total = 0;  
  forEach(numbers, function (number) {  
    total += number; });  
  return total;  
}  
console.log(sum([1, 10, 100]));
```

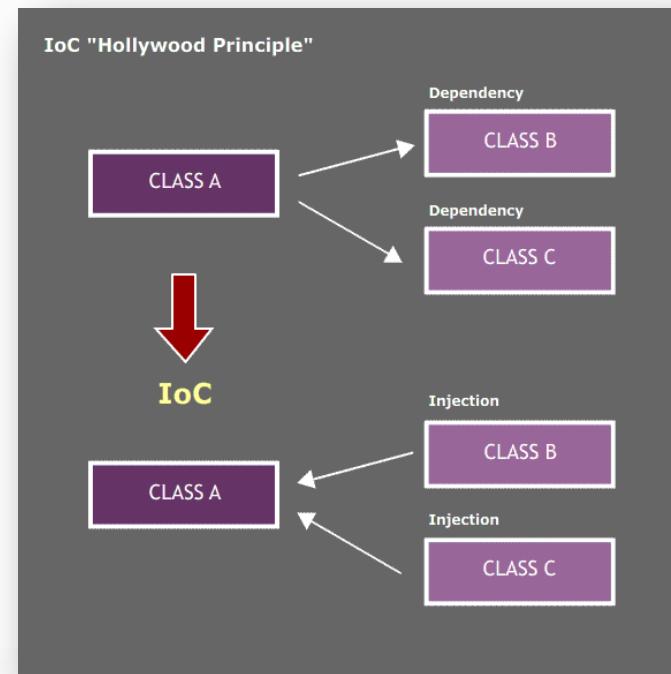


JS Async Programming



IoC Programming Style

- ➊ Inversion of Control (IoC) is a design principle (some people refer to it as a pattern).
- ➋ Behavioral pattern that Reverses the “traditional” flow of control
- ➌ IoC helps in designing loosely coupled classes which make them testable, maintainable and extensible.
- ➍ Implementation examples :
 - ➎ Callbacks in general
 - ➎ Events
 - ➎ Observer/Mediator design patterns
 - ➎ DI Technique
 - ➎ Promises



Using Event Emitter

- emit is used to trigger an event
- on is used to add a callback function that's going to be executed when the event is triggered

```
const EventEmitter = require('events')
const eventEmitter = new EventEmitter()
```

```
eventEmitter.on('start', number => {
  console.log(`started ${number}`)
})
```

```
eventEmitter.emit('start', 42)
```

Using Event Emitter

```
const express = require('express')
const other = require('./otherModule')
const app = express()
const port = 3000
other.setTheCb((seconds) => {
  console.log(`the session ended
after ${seconds} sec.`)
});

app.get('/', (req, res) => {
  res.send('Hello World!')
  other.start(3000)
})
app.listen(port, () => {
  console.log(`listening at
http://localhost:${port}`)
})
```

otherModule.js

```
const EventEmitter = require('events')
const eventEmitter = new EventEmitter()

var setTheCb=function(cb)
{
  console.log("set cb")
  eventEmitter.on('endTimer',cb)
}

function start(seconds)
{
  console.log("counting")
  setTimeout(() =>
{eventEmitter.emit('endTimer',seconds)},seconds);
}

module.exports={start:start,
  setTheCb:setTheCb
};
```

Promises

- A *promise* is an object which represents the result of an asynchronous operation which is either resolved or rejected
- Promises are used to handle asynchronous operations. They are easy to manage when dealing with multiple asynchronous operations where callbacks can create ***callback hell*** leading to unmanageable code
- The function passed to new Promise is called the executor. When new Promise is created, the executor runs automatically. It contains the producing code which should eventually produce the result.

```
let promise = new Promise(function(resolve, reject){  
  //do something });
```

Promises

- states:

- Fulfilled: resolve() was called)
- Rejected: reject() was called)
- Pending: not yet fulfilled or rejected
- settled: Promise has fulfilled or rejected

```
// the function is executed automatically when the promise is constructed
// after 1 second signal that the job is done with the result "done"
let promise = new Promise(function(resolve, reject){
  setTimeout(()=> resolve("done"), 1000);
});
```



```
// after 1 second signal that the job is finished with an error
let promise = new Promise(function(resolve, reject){
  setTimeout(()=> reject(new Error("Whoops!")), 1000);});
```



Promises

```
let promise = new
Promise(function(resolve, reject) {
setTimeout(() => resolve("done!"), 1000);});
```

```
promise.then(
result => console.log(result),
error => console.log(error)
);
```

```
const myPromise = new Promise((resolve,
reject) => {
if (Math.random() > 0) {
    resolve('Hello, I am positive number!');
}
reject(new Error('I failed some times'));
})
```

<https://javascript.info/promise-basics>

async functions

- a function declared with the `async` keyword, and the `await` keyword is permitted within them.
- The `async` and `await` keywords enable asynchronous, promise-based behavior to be written in a cleaner style, avoiding the need to explicitly configure promise chains.

```
function resolveAfter2Seconds() {
  return new Promise(resolve => {
    setTimeout(() => {
      resolve('resolved');
    }, 2000);
  });
}

async function asyncCall() {
  console.log('calling');
  const result = await resolveAfter2Seconds();
  console.log(result);
  // expected output: "resolved"
}

asyncCall();
```

async functions

- a function declared with the `async` keyword, and the `await` keyword is permitted within them.
- The `async` and `await` keywords enable asynchronous, promise-based behavior to be written in a cleaner style, avoiding the need to explicitly configure promise chains.

```
function resolveAfter2Seconds() {
  return new Promise(resolve => {
    setTimeout(() => {
      resolve('resolved');
    }, 2000);
  });
}

async function asyncCall() {
  console.log('calling');
  const result = await resolveAfter2Seconds();
  console.log(result);
  // expected output: "resolved"
}

asyncCall();
```



Namespace & IIFE Patterns



IIFE Pattern

- IIFE (Immediately Invoked Function Expression)
- Limit the number of global variables

```
(function ()  
{ statements})();
```

```
(function ()  
{ let firstVariable;  
  let secondVariable;  
 })();
```

Namespacing

- ➊ Namespaces can be found in almost any serious JavaScript application Unless we're working with a simple code.

- ➋ Common Patterns:
 - ➌ Single global variables
 - ➌ Prefix namespacing
 - ➌ Object literal notation
 - ➌ Nested namespacing
 - ➌ Immediately-invoked Function
 - ➌ Expressions
 - ➌ Namespace injection

Revealing Module pattern

```
var namespace = (function () {  
  
    // defined within the local scope  
    var privateMethod1 = function () { console.log("function 1") },  
        privateMethod2 = function () { console.log("function 2") }  
        privateProperty1 = "foobar";  
  
    return {  
  
        // the object literal returned here can have as many nested depths as we wish, however  
        // this way of doing things works best for smaller limited-scope  
  
        publicMethod1: privateMethod1,  
  
        // nested namespace with public properties  
        properties:{  
            publicProperty1: privateProperty1  
        },  
  
        // another tested namespace  
        utils:{  
            publicMethod2: privateMethod2  
        }  
    }  
})(());  
  
console.log(namespace.properties.publicProperty1)
```

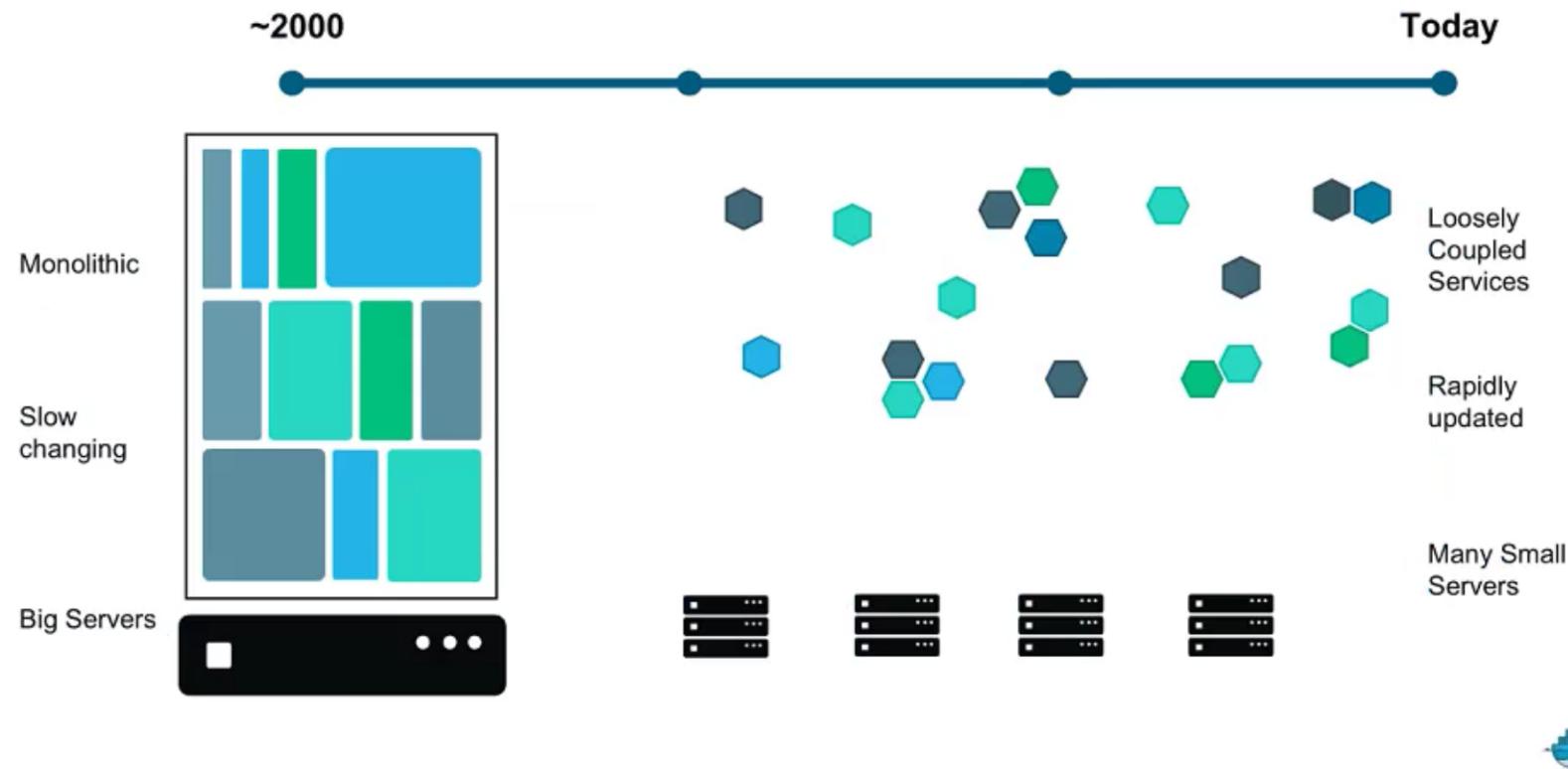


Service Orientation & Microservices Pattern



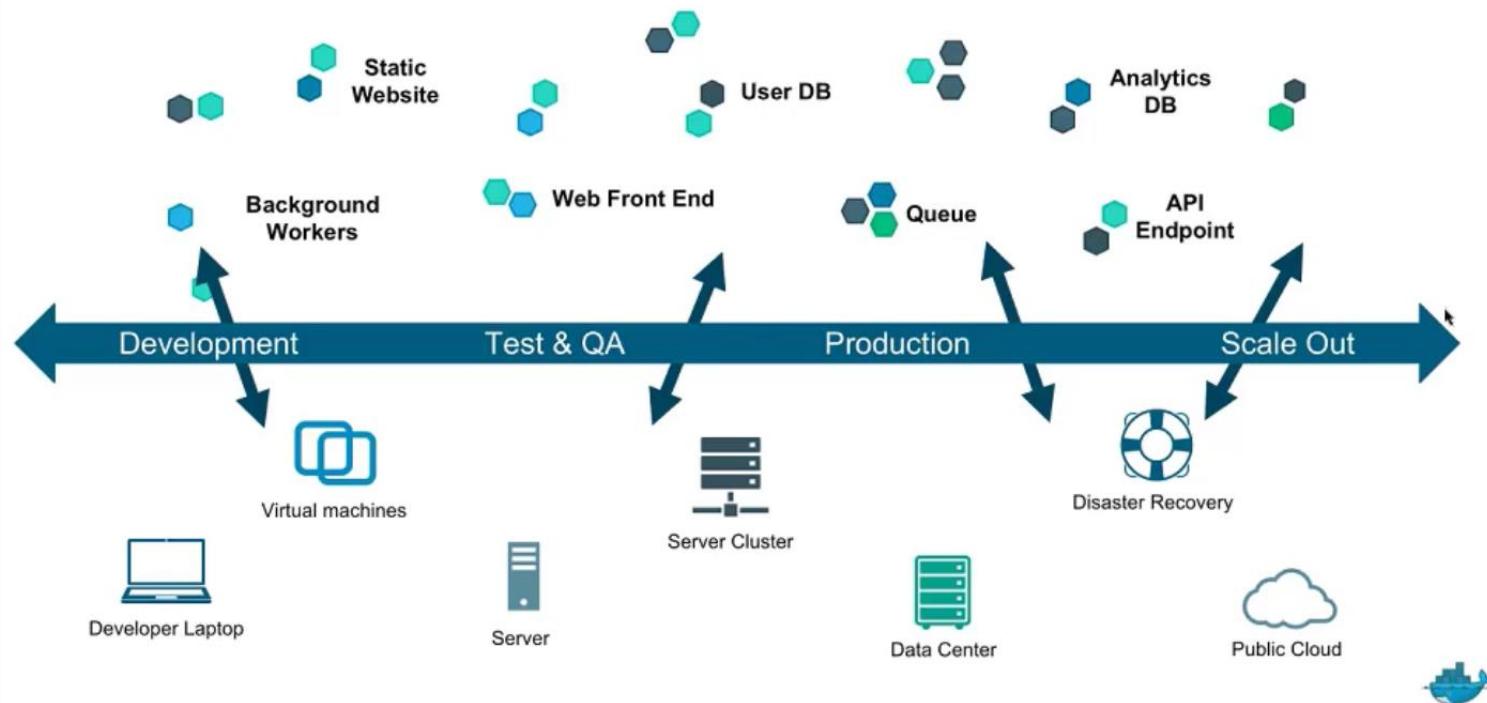
Development Trends

Transforming the Application Landscape



Development Trends

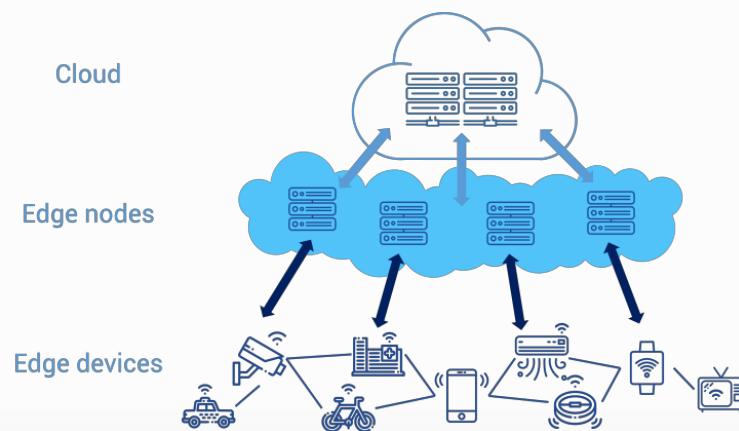
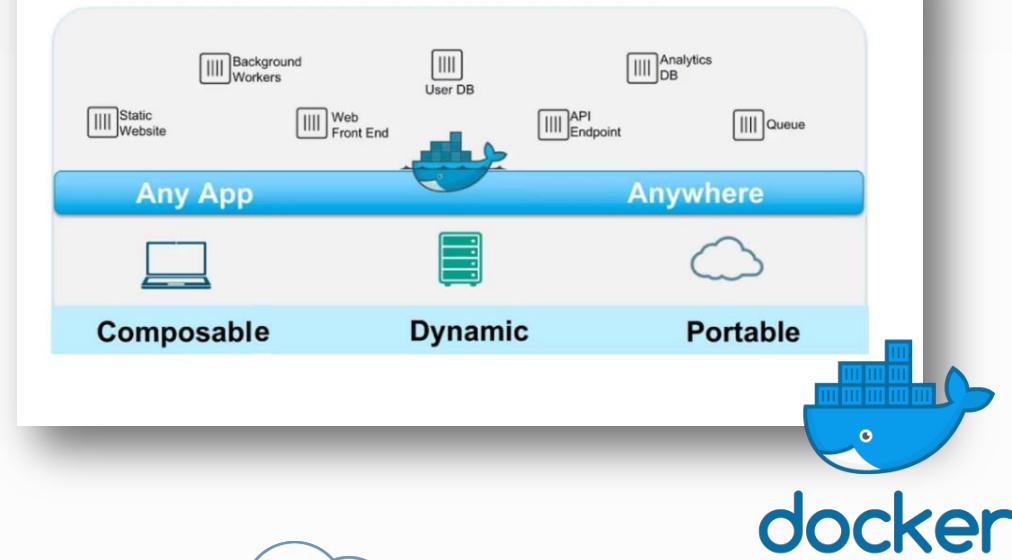
The New Challenge of Distributed Apps



Where Services Live ?



Distributed Application Solution



Software Services

- ➊ Several Granularity References (SaaS, PaaS, IaaS etc.)
- ➋ Our Focus: Functionality Units supplying Two types of Services
 - ➌ Compute
 - ➌ Storage
- ➌ The Successor of Software Component
- ➌ Loose Coupling between services
- ➌ Cross Platform
- ➌ Implementation Alternatives:
 - ➌ RESTful API
 - ➌ gRPC
 - ➌ GraphQL
- ➌ Services can be implemented as Microservices

<https://www.imaginarycloud.com/blog/grpc-vs-rest/>

Microservices Defined

- ➊ There is no formal definition of the term microservices
- ➋ Yet most microservice systems share a few notable characteristics:
 - 1) Software broken down into multiple component services
 - 2) Organized around business capabilities and priorities
 - 3) Simple Routing - microservices have smart endpoints that process info and apply logic, and dumb pipes through which the info flows.
 - 4) Decentralized & involve a variety of technologies and platforms
 - 5) Failure Resistant - microservices are designed to cope with failure
 - 6) Evolutionary by Design

Microservices - Pros

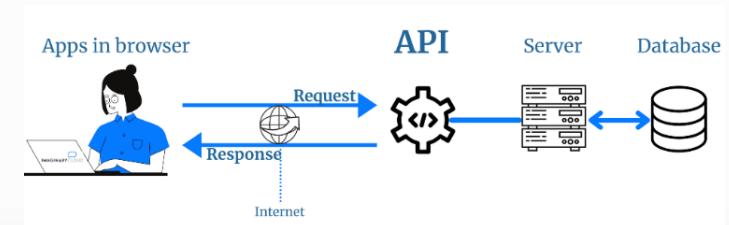
- Microservice architecture gives developers the freedom to independently develop and deploy services
- A microservice can be developed by a fairly small team
- Code for different services can be written in different languages (though many practitioners discourage it).
- Easy integration and automatic deployment (using open-source continuous integration tools such as Jenkins, Hudson, etc.)
- Easy to understand & modify for developers, can help a new team member become productive quickly.
- The code is organized around business capabilities.
- When change is required in a certain part of the application, only the related service can be modified and redeployed—no need to modify and redeploy the entire application.
- Better fault isolation: if one microservice fails, the other will continue to work (although one problematic area of a monolith application can jeopardize the entire system).
- Easy to scale and integrate with third-party services.
- No long-term commitment to technology stack, Can adopt new one's easily.

Microservices - Cons

- Due to distributed deployment, testing can become complicated and tedious.
- Increasing number of services can result in information barriers.
- The architecture brings additional complexity as the developers have to mitigate fault tolerance, network latency, and deal with a variety of message formats as well as load balancing.
- Being a distributed system, it can result in duplication of effort.
- When number of services increases, integration and managing whole products can become complicated
- In addition to several complexities of monolithic architecture, the developers have to deal with the additional complexity of a distributed system.
- Developers have to put additional effort implementing mechanism of communication between the services
- Handling use cases that span more than one service without using distributed transactions is not only tough but also requires communication and cooperation between different teams.
- Partitioning the application into microservices is very much an art.

API's

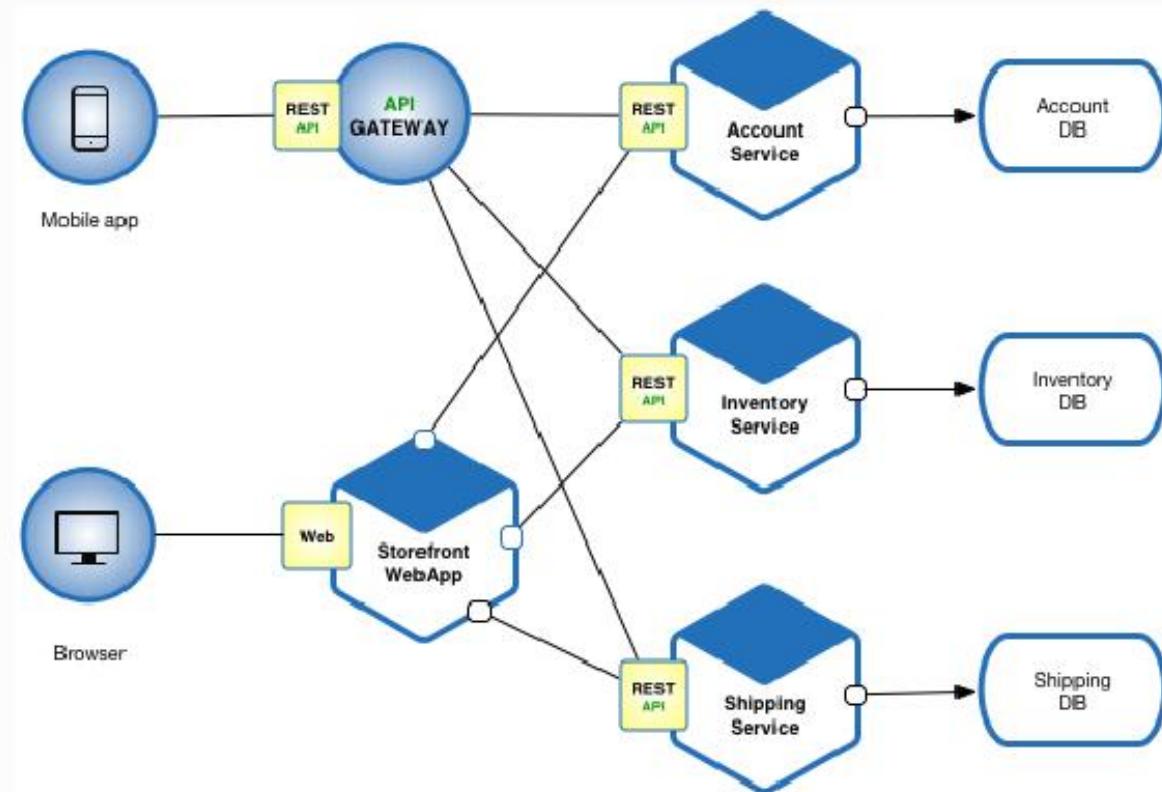
- **Monolithic application:** all the project's functionalities are included in a single unit, more precisely, in a single codebase.
- **Microservice architecture:** comprised of several smaller services that communicate with each other using protocols like HTTP.
- The component services that are part of the microservices architecture communicate and interact with each other through APIs.
- APIs allow all the services that are integrated into a microserviceapplication to connect and communicate.
- Most used architectural style is the **REST API**. Yet there are three main models when building an API:
 - **RPC** (Remote Procedure Call)
 - **REST** (Representational State Transfer)
 - **GraphQL**



API Gateway

API gateway is an API management tool that sits between a client and a collection of backend services.

An API gateway acts as a reverse proxy to accept all application programming interface (API) calls, aggregate the various services required to fulfill them, and return the appropriate result.



<http://microservices.io/patterns/microservices.html>

Swagger & OpenAPI

- Swagger is a software tool used for designing, building, documenting, and using RESTful APIs. It follows the OpenAPI specification
- OpenAPI specification is a specification used for creating interfaces used in describing, producing, consuming, and visualizing RESTful APIs.

```
module.exports = {
  // method of operation
  get: {
    tags: ["Todo CRUD operations"], // operation's tag.
    description: "Get todos", // operation's desc.
    operationId: "getTodos", // unique operation id.
    parameters: [], // expected params.
    // expected responses
    responses: {
      // response code
      200: {
        description: "Todos were obtained", // response desc.
        content: {
          // content-type
          "application/json": {
            schema: {
              $ref: "#/components/schemas/Todo", // Todo model
            },
          },
        },
      },
    },
  };
};
```

<https://www.section.io/engineering-education/documenting-node-js-rest-api-using-swagger/>



Containers & Docker



Containers

Who is Malcom McLean ?



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

Docker Terminology

Let's start with some basic concepts



Docker Image

The basis of a Docker container



Docker Container

The standard unit in which the application service resides



Docker Engine

Creates, ships and runs Docker containers deployable on physical or virtual host locally, in a datacenter or cloud service provider



Docker Trusted Registry

Dedicated image store and distribution service deployed in your firewall



Docker

We can build our software *and the environment it runs on* as code and deploy easily.

```
docker run -it ubuntu
```

After a bit of spinning, you'll see a prompt like this:

```
root@719059da250d:/#
```

Try out a few commands and then exit the container:

```
root@719059da250d:/# lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 14.04.4 LTS
Release:        14.04
Codename:       trusty
root@719059da250d:/# exit
```

1. We issue a docker command:

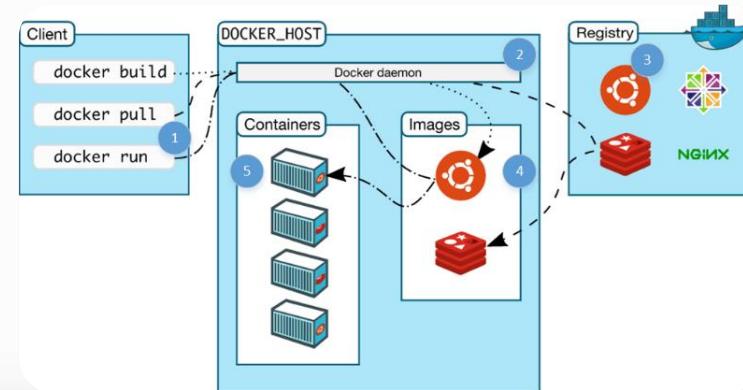
- `docker`: run the docker client
- `run`: the command to run a new container
- `-it`: option to give the container an interactive terminal
- `ubuntu`: the image to base the container on

2. The docker service running on the host (our machine) checks to see if we have a copy of the requested image locally- which there isn't.

3. The docker service checks the public registry (the docker hub) to see if there's an image named `ubuntu` available- which there is.

4. The docker service downloads the image and stores it in its local cache of images (ready for next time).

5. The docker service creates a new container, based on the `ubuntu` image.



Docker Images

 nginx	official	3.5K STARS	10M+ PULLS	 DETAILS
 busybox	official	737 STARS	10M+ PULLS	 DETAILS
 ubuntu	official	4.3K STARS	10M+ PULLS	 DETAILS
 redis	official	2.4K STARS	10M+ PULLS	 DETAILS
 docker	registry	950 STARS	10M+ PULLS	 DETAILS
 swarm	official	410 STARS	10M+ PULLS	 DETAILS
 mongo	official	2.1K STARS	10M+ PULLS	 DETAILS

<https://hub.docker.com/explore/>

Docker Version

Command Prompt

```
D:\>docker version
Client:
  Version: 1.13.1
  API version: 1.26
  Go version: go1.7.5
  Git commit: 092cba3
  Built: Wed Feb 8 08:47:51 2017
  OS/Arch: windows/amd64

Server:
  Version: 1.13.1
  API version: 1.26 (minimum version 1.12)
  Go version: go1.7.5
  Git commit: 092cba3
  Built: Wed Feb 8 08:47:51 2017
  OS/Arch: linux/amd64
  Experimental: true

D:\>
```

Docker Info



```
OS/Arch: linux/amd64
Experimental: true

D:\>docker info
Containers: 1
Running: 0
Paused: 0
Stopped: 1
Images: 3
Server Version: 1.13.1
Storage Driver: aufs
Root Dir: /var/lib/docker/aufs
Backing Filesystem: extfs
Dirs: 53
Dirperm1 Supported: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
Volume: local
Network: bridge host ipulang macvlan null overlay
Swarm: inactive
Runtimes: runc
Default Runtime: runc
Init Binary: docker-init
containerd version: aa8187dbd3b7ad67d8e5e3a15115d3eef43a7ed1
runc version: 9df8b306d01f59d3a8029be411de015b7304dd8f
init version: 949e6fa
Security Options:
seccomp
Profile: default
Kernel Version: 4.9.8-moby
Operating System: Alpine Linux v3.5
OSType: linux
Architecture: x86_64
CPUs: 2
Total Memory: 1.934 GiB
Name: moby
ID: GHYL:N3UT:B3PR:H7OM:LIJF:FFSA:7X22:6R2L:DDL3:WH3H:25WK:025M
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): true
File Descriptors: 13
Goroutines: 21
System Time: 2017-03-01T21:40:10.3012494Z
EventsListeners: 0
Registry: https://index.docker.io/v1/
Experimental: true
Insecure Registries:
127.0.0.0/8
Live Restore Enabled: false

D:\>
```

Docker Hello World

```
docker run docker/whalesay cowsay Hello world
```

```
D:\>docker run docker/whalesay cowsay Hello world
Unable to find image 'docker/whalesay:latest' locally
latest: Pulling from docker/whalesay
e190868d63f8: Pull complete
909cd34c6fd7: Pull complete
0b9bfabab7c1: Pull complete
a3ed95caeb02: Pull complete
00bf65475aba: Pull complete
c57b6bcc83e3: Pull complete
8978f6879e2f: Pull complete
8eed3712d2cf: Pull complete
Digest: sha256:178598e51a26abbc958b8a2e48825c90bc22e641de3d31e18aaef55f3258ba93b
Status: Downloaded newer image for docker/whalesay:latest

-----
< Hello world >
-----
          ##          .
          ## ## ##      ==
          ## ## ## ##    ===
          {~~ ~~~~ ~~~ ~~~~ ~~ /  ===- ~~~
             \     o      /-
              \_ \_/ /- /-
```



Serverless Development



What is Serverless

- ➊ Serverless is a cloud development model that allows developers to build and run applications without having to manage servers
- ➋ Servers are abstracted away from application developers
- ➌ Cloud provider deals with non-functional aspects like scaling or provisioning
- ➍ Developers can simply package their code in Containers for deployment.
- ➎ Serverless apps respond to demand and Automatically scale up and down as needed.
- ➏ Usually using event driven execution model
- ➐ Two Types:
 - ➑ Backend-as-a-Service (BaaS)
 - ➒ Function-as-a-Service (FaaS)

Serverless

Simulates a physical machine

Provides a local file system

Can be accessed over a network



shutterstock.com · 647778754

<https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-getting-started-hello-world.html>

<https://www.serverless.com/blog/serverless-express-rest-api/>