BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Web Apps

Chandan Ravandur N

Website

What & Why?

- A group of interlinked web pages having a single domain name
  - Hosted on a webserver
  - Accessible over the web with internet connection
  - Easily accessible through browsers
  - Can be developed and maintained by individuals / teams for personal or businesses usage

- For example,
  - BITS Pilani website , Any Newpaper website

- Why?
  - Easy to share the information for individual, product or services

- Shortcoming
  - Same interface / information shown to all – no personalization

Web Application

What?

- Application software that runs on a (usually) on remote computer
  - Hosted on a webserver
  - Accessible over the web with internet connection or over intranet
  - Easily accessible through browsers
  - Usually based on client –server architecture
  - Can be personalized , customized as per the users need

- For example,
  - BITS Elearn portal
  - Your company's Payroll app
  - Gmail

- Why?
  - Easy to develop, maintain and access!

Comparison

Website vs Web Apps

	Website	Web Apps
Meant for	Rendering static content like text, images etc.	Rendering customized / dynamic contents
Interaction	One way – from website to all users Same content for all users	Two way – between portal and users Content changes based on interaction
Authentication	Usually not required	Both authentication and authorization required
Complexity of development	Easier to develop HTML, CSS	Will vary per requirement of applications HTML, CSS, JS Many frameworks

Evolution of the web

- Lets Visit!

Web Apps Architecture

Why?

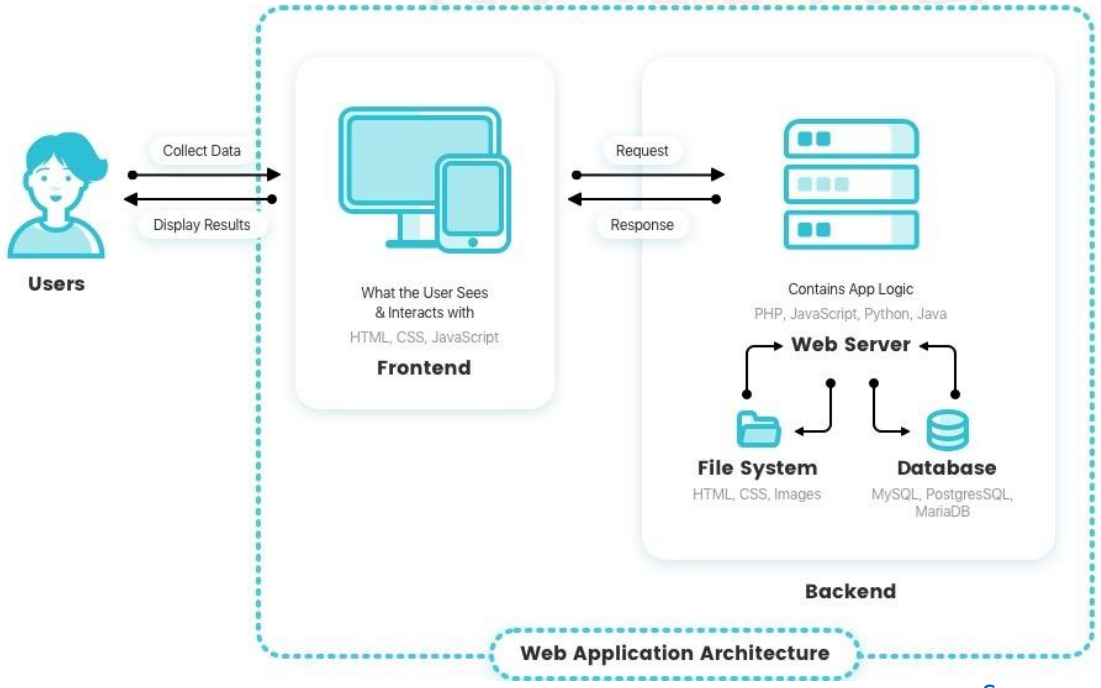
- For years, businesses are maintaining their website
  - Without knowing anything about web apps architecture
  - Possible because of players like GoDaddy etc. which provides web hosting capabilities

- Web apps are required when
  - content needs to be dynamic
  - Complexity is involved – databases , persistence storage comes in
  - more control required on the content

- Then start feeling need of way to organise the things In better manner
  - Then Comes in Web Apps architecture!
  - Conventionally two tier – client server
  - But can be easily extended to – n tier

Web Apps Architecture(2)

- Consists of many components
  - user interfaces
  - server side
  - databases
- Web application architecture is used to logically define the relationships and manner of interactions between all of these components for a Web app



Source

Web App Components

Frontend , Backend and Databases

- Typically Web application consists of a front-end, a back-end and databases

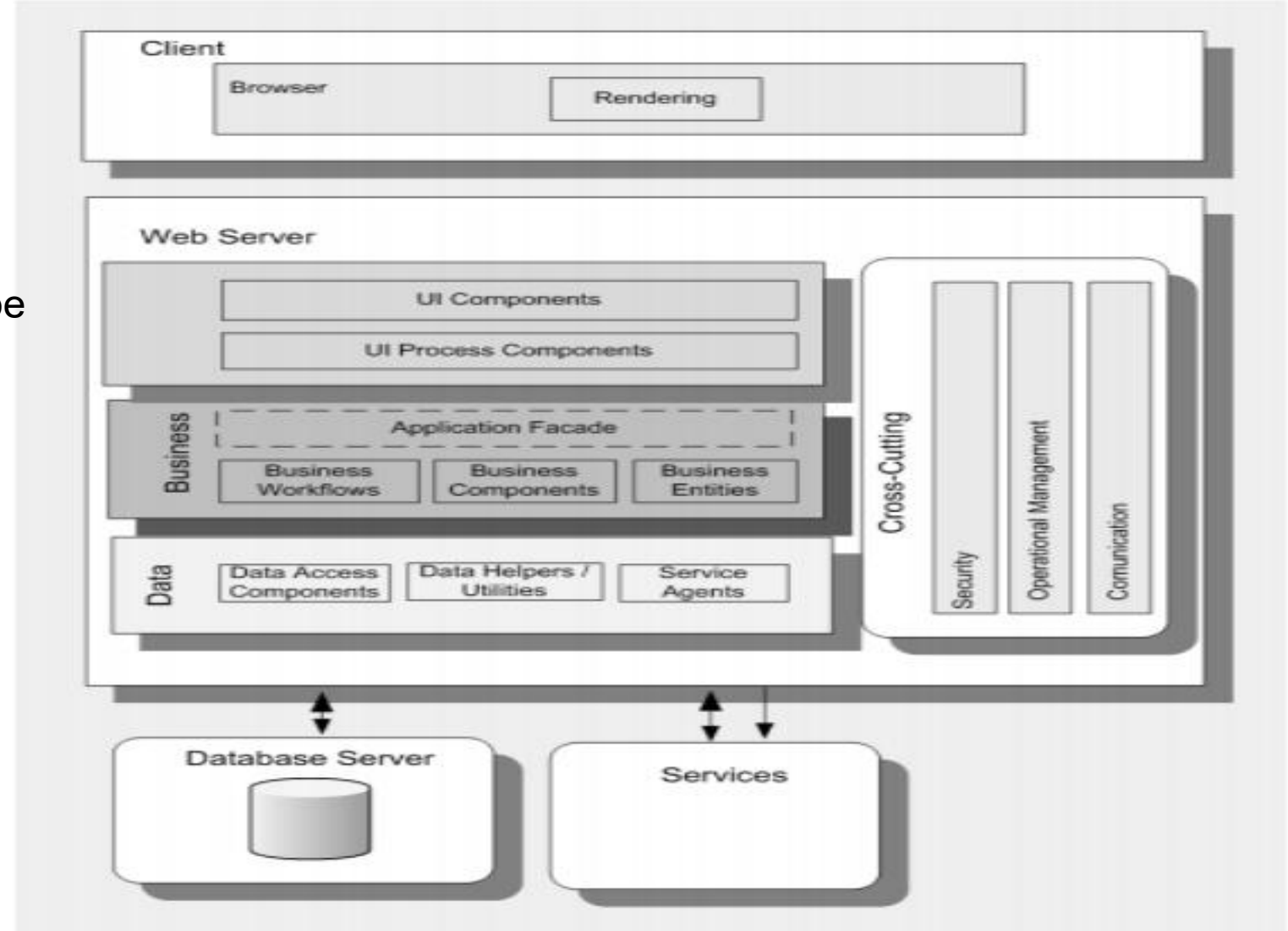
- The front-end (the client-side)
  - Whatever the user sees and interacts within inside their browser
  - The main purpose of the client-side is to interact with users
  - HTML, CSS, and JavaScript

- Back-end (the server-side)
  - Not visible to the users - stores and manipulates data
  - Accepts and fulfills the HTTP requests which essentially “fetch” the data (text, images, files, etc.) called for by the user
  - PHP, Java, Python, JavaScript

- Databases
  - Usually Relational Database Management Systems (RDBMS) are used to store the data in structured format
  - Backend interacts with Databases to fetch the required data

Common Web App Architecture

- The core of a Web application is
  - its server-side logic
- The Web application layer itself can be comprised of many distinct layers
  - Typically a three-layered architecture
  - comprised of
    - ❖ Presentation
    - ❖ Business
    - ❖ data layers

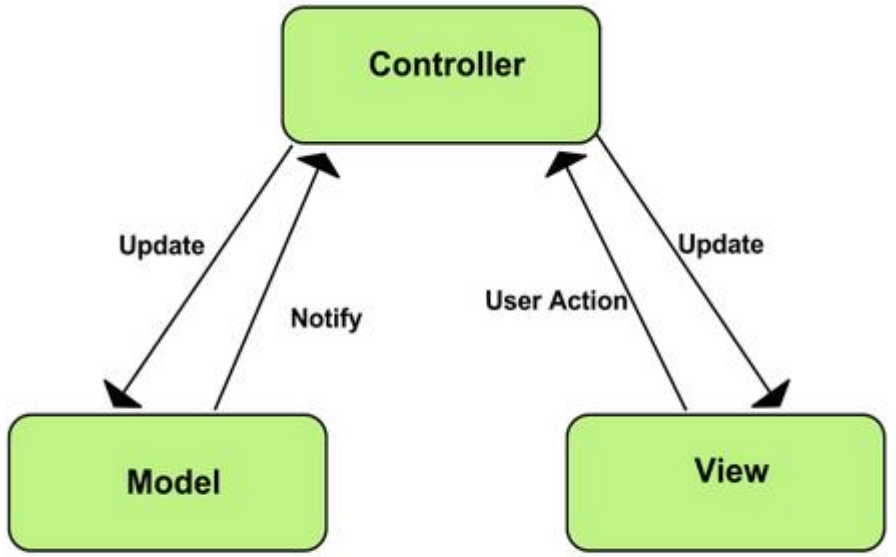


Source : MS Web Application Architecture Guide

Model – View – Controller Architecture

MVC

- Software design pattern commonly used for developing user interfaces that divides the related program logic into three interconnected elements
  - Model
  - View
  - Controller
- Supported well in JavaScript, Python, Ruby, PHP, Java, C#



Model

Source : Google Chrome

- Model
  - responsible for managing the data of the application
  - Accepts user input from the controller
- View
  - means presentation of the model in a particular format to the user
- Controller
  - responds to the user input and performs interactions on the data model objects
  - Receives the input, optionally validates it and then passes the input to the model

Trends in Web Application Architecture

SSR , CSR

- Server-Side Rendering (SSR):
  - Conventional approach
  - Separate request - response cycle for each activity carried out on by user on browser
  - When clicking a URL
    - ❖ a request is sent to the server
    - ❖ server processes request
    - ❖ the browser receives the files (HTML, CSS, and JavaScript) and the content of the page and then renders it
    - ❖ Repeated for every request

- Client-Side Rendering (CSR):
  - Only a single request will be made to the server to load the main skeleton of the app
  - Content is then dynamically generated using JavaScript
  - Aka Single Page Applications

Comparison

SSR Vs CSR

	Server Side Rendering	Client Side Rendering
Suitable for	Rendering apps with more static content	More dynamism is involved
Initial load time	Initial loads are faster	Initial loads are slow
Request – Response behavior	Full request – response cycle for each action	After initial load, response comes in fast or computed locally
Technologies	Conventional like JSP, Java, PHP etc.	Modern Web App Frameworks like Angular

Web App Framework

Defined

- A framework is a library that offers opinions about how software gets built.

- Web application framework (WAF)
  - software framework that is designed to support the development of web applications including
    - ❖ web services
    - ❖ web resources
    - ❖ web APIs
  - Provide a standard way to build and deploy web applications on the World Wide Web including support for
    - libraries for database access
    - templating frameworks
    - session management etc.

- Two types
  - Client side
  - Server Side

Client-Side Programming

Frontend / User Interface

- Involves everything users see on their screens.

- Major frontend technology stack components:
  - HTML, CSS and JS

- Hypertext Markup Language (HTML) and Cascading Style Sheets (CSS)
  - HTML tells a browser how to display the content of web pages
  - CSS styles that content
  - Bootstrap is a helpful framework for managing HTML and CSS

- JavaScript (JS)
  - Makes web pages interactive
  - Many JavaScript libraries (such as jQuery, React.js)
  - frameworks (such as Angular, Vue, Backbone, and Ember)

Server-Side Programming

Backend

- Not visible to users, but it powers the client side

- Major server side technology stack components:
  - Programming language, Framework , web server and databases

- Server-side programming languages used to create the logic of applications
- Frameworks offer lots of tools for simpler and faster development of applications

- Options
  - Ruby (Ruby on Rails)
  - Python (Django, Flask, Pylons)
  - PHP (Laravel)
  - Java (Spring)
  - Scala (Play)
  - Javascript (Node.js)

Server-Side Programming

Other Components

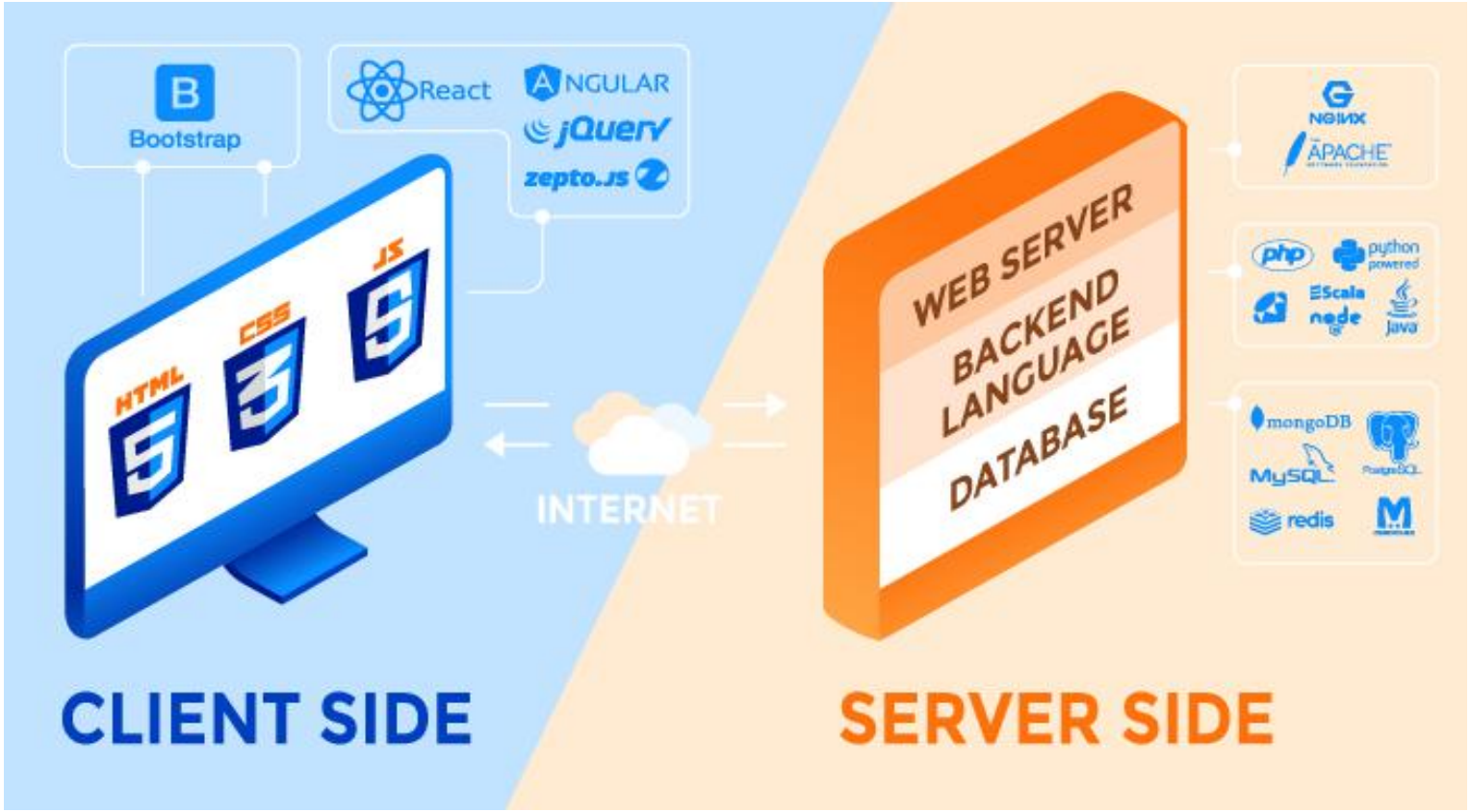
- Storage
  - Apps needs a place to store its data
  - Two types of databases:
    - relational and non-relational
  - Most common databases for web development:
    - MySQL (relational)
    - PostgreSQL (relational)
    - MongoDB (non-relational, document)

- Caching system
  - Used to reduce the load on the database and to handle large amounts of traffic
  - Memcached and Redis are the most widespread

- Web servers
  - Needs a server to handle requests from clients' computers
  - Two major players:
    - Apache
    - Nginx

Web App Tech Stack

Modern Tech Stack



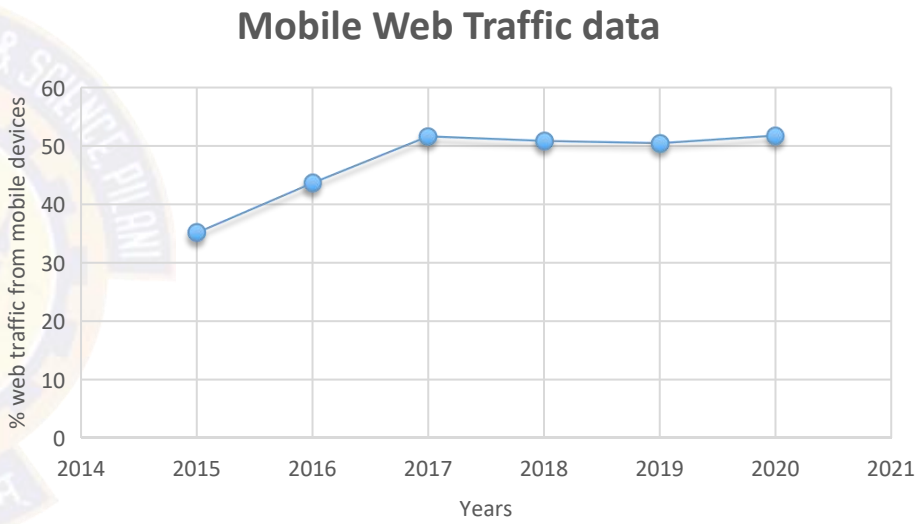
Source : RubyGarage

Mobile device website traffic

Increasing!

- Now mobile causes half of worldwide web traffic!
- Continuously hovering over 50% for last years!

- Causes
  - acceleration to digital initiatives
  - moving to digital models of business exclusively
  - the rollout of 4G, plans for 5G
  - increasing IoT devices
  - Lot of mobile only population in developing countries



Mobile Apps vs Mobile Websites

- No doubt businesses can ignore Mobiles!
- Which way to go ?
  - Mobile websites
  - Mobile Apps
- Looks similar but are different mediums!
- Depends also upon
  - Target audience and intent
  - Budget

Image source : Wikipedia

Mobile Apps

Aka Native Apps

- Are meant for specific platforms
  - Apple iOS
  - Google Android
- Needs to be downloaded and installed on mobile devices

- Advantages
  - Offers a faster and more responsive experience
  - More Interactive Ways For The User To Engage
  - Ability To Work Offline
  - Leverage Device Capabilities

Source : Wikipedia

Mobile Websites

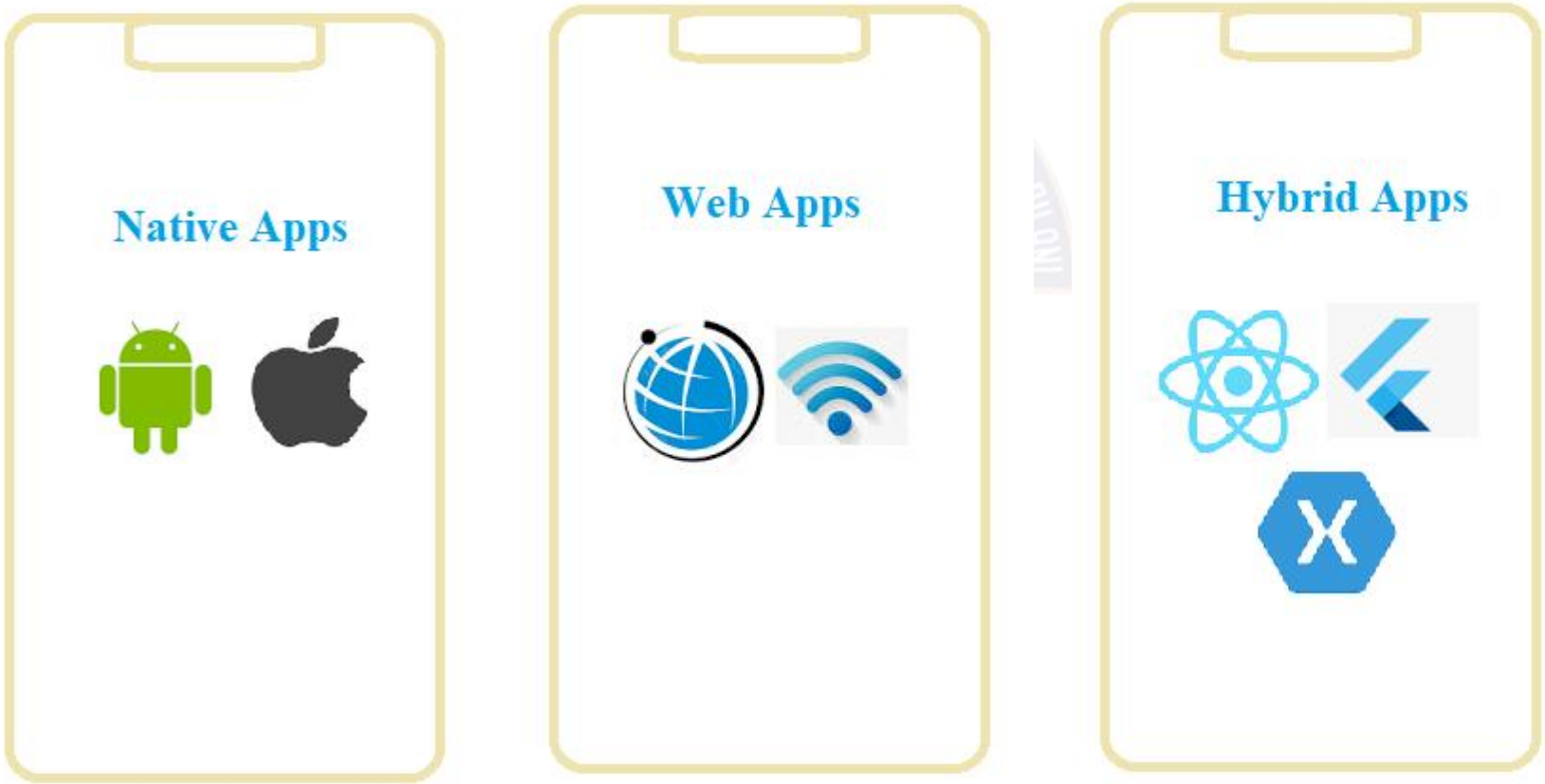
Aka Responsive mobile websites

- Websites that can accommodate different screen sizes
- Customized version of a regular website that is used correctly for mobile
- Accessed through Mobile browsers

- Advantages
  - Available For All Users
  - Users Don’t Have To Update
  - Cost-Effective

Mobile Apps - Types

Three!



Native Apps

- Developed specifically for a particular mobile device
- Installed directly onto the device itself
- Needs to be downloaded via app stores such as
  - Apple App Store
  - Google Play store, etc.

- Built for specific mobile operating system such as
  - Apple iOS
  - Android OS
- An app made for Apple iOS will not work on Android OS or Windows OS

- Need to target all major mobile operating systems
  - require more money and more effort

Native Apps

Pros and Cons

- Pros
  - Can be Used offline - faster to open and access anytime
  - Allow direct access to device hardware that is either more difficult or impossible with a web apps
  - Allow the user to use device-specific hand gestures
  - Gets the approval of the app store they are intended for
  - User can be assured of improved safety and security of the app

- Cons
  - More expensive to develop - separate app for each target platform
  - Cost of app maintenance is higher - especially if this app supports more than one mobile platform
  - Getting the app approved for the various app stores can prove to be long and tedious process
  - Needs to download and install the updates to the apps onto users mobile device

Web Apps

- Basically internet-enabled applications
  - Accessible via the mobile device's Web browser
- Don't need to downloaded and installed onto mobile device

- Written as web pages in HTML and CSS
  - with the interactive parts in Jquery, JavaScript etc.
- Single web app can be used on most devices capable of surfing the web
  - irrespective of the operating system

Web Apps

Pros and Cons

- Pros
  - Instantly accessible to users via a browser
  - Easier to update or maintain
  - Easily discoverable through search engines
  - Development is considerably more time and cost-effective than development of a native app
  - common programming languages and technologies
  - Much larger developer base.

- Cons
  - Only have limited scope as far as accessing a mobile device's features is concerned
    - device-specific hand gestures, sensors, etc.
  - Many variations between web browsers and browser versions and phones
  - Challenging to develop a stable web-app that runs on all devices without any issues
  - Not listed in 'App Stores'
  - Unavailable when offline, even as a basic version

Hybrid Apps

- Part native apps, part web apps
- Like native apps,
  - available in an app store
  - can take advantage of some device features available
- Like web apps,
  - Rely on HTML, CSS , JS for browser rendering

- The heart of a hybrid-mobile application is application that is written with HTML, CSS, and JavaScript!
- Run from within a native application and its own embedded browser, which is essentially invisible to the user
  - iOS application would use the WKWebView to display application
  - Android app would use the WebView element to do the same function

Hybrid Apps

Pros and Cons

- Pros
  - Don’t need a web browser like web apps
  - Can access to a device’s internal APIs and device hardware
  - Only one codebase is needed for hybrid apps

- Cons
  - Much slower than native apps
  - With hybrid app development, dependent on a third-party platform to deploy the app’s wrapper
  - Customization support is limited

Compared!

Key Features: Native, Web, & Hybrid

Feature	Native	Web-only	Hybrid
Device Access	Full	Limited	Full (with plugins)
Performance	High	Medium to High	Medium to High
Development Language	Platform Specific	HTML, CSS, Javascript	HTML, CSS, Javascript
Cross-Platform Support	No	Yes	Yes
User Experience	High	Medium to High	Medium to High
Code Reuse	No	Yes	Yes

Source : Ionic

Computing Platform

- Is the environment in which a software code is executed
- A computing platform is the stage on which computer programs can run

- May be
  - the hardware or the operating system (OS)
    - ❖ x86 or ARM architecture
    - ❖ Windows, Linux, Mac, iOS, Android

  - a web browser
    - ❖ Chrome, Edge, Firefox etc.

  - Or other underlying software as long as the program code is executed with it
    - ❖ JVM

Cross Platform Software

- Computer software that is implemented to run on multiple computing platforms
- May run on as many as all existing platforms, or on as few as two platforms

- Two types:
- First one requires individual building or compilation for each platform that it supports
  - For example,
  - Installers of Software products for different OS like Windows, Mac or Linux
  - Mobile apps meant for platforms like Android and iOS

- Other one can be directly run on any platform without special preparation
  - Software written in an interpreted language or pre-compiled portable bytecode
  - Java App meant to be executed of different OS

Cross Platform Apps

Four Types

- Binary Software's / Installers
  - Application software distributed to end-users as binary file, especially executable files
  - Executables only support the operating system and computer architecture that they were built for
  - For example, Firefox, an open-source web browser, is available on Windows, macOS, Linux
    - ❖ The four platforms are separate executable distributions, although they come from the same source code
- Web applications
  - Typically described as cross-platform because, ideally, they are accessible from any of various web browsers within different operating systems
  - Generally employ a client–server system architecture, and vary widely in complexity and functionality
- Scripted / Interpreted Languages
  - Interpreter is available on multiple platforms and the script only uses the facilities provided by the language
  - Same script can be used on all computers that have software to interpret the script
  - script is generally stored in plain text in a text file
  - Script written in Python for a Unix-like system will likely run with little or no modification on Windows
- Video Games
  - Video games released on a range of video game consoles, specialized computers dedicated to the task of playing games
  - Wii, PlayStation 3, Xbox 360, personal computers (PCs), and mobile devices







Cross-platform programming

Two Conventional Approaches

- The practice of actively writing software that will work on more than one platform

- Approaches to cross-platform programming
- Using separate code bases for each platform
  - Simply to create multiple versions of the same program in different source trees
  - Microsoft Windows version of a program might have one set of source code files and the Macintosh version might have another
  - Straightforward approach to the problem
  - considerably more expensive in development cost, development time
    - more problems with bug tracking and fixing
    - different programmers, and thus different defects in each version

- Using abstractions
  - Depend on pre-existing software that hides the differences between the platforms
  - called abstraction of the platform—such that the program itself is unaware of the platform it is running on
  - Programs are platform agnostic
  - Programs that run on the Java Virtual Machine (JVM) are built in this fashion

Modern Cross Platform Development

Using only one code base / framework

- Cross-platform app development is process of creating apps that can be
  - deployed or published on multiple platforms
  - using a single codebase
  - instead of having to develop the app multiple times
  - using the respective native technologies for each platform

- The term is commonly used in the context of Mobile apps but quite appropriate for applications targeted for both all three categories mobile, web and desktop!

- Frameworks available for each of the category i.e.
  - Mobile only cross platform app development
  - Targeted for all sorts of apps development

Cross-Platform Development Pros

Advantages

- Code reusability
  - Tools allow to write code once then export app to many operating systems and platforms without having to create a dedicated app for every single platform
- Convenience while developing
  - Tools saves from the hassle of having to learn multiple programming languages and instead offers one substitute for all of these different technologies
- Easier Code Maintenance
  - With every change, only one codebase needs to updated and can pushed to all the apps on different platforms
- Cost Efficiency
  - Saves the cost of having multiple teams working on different versions of app and substituting them with one team
  - Tools are also free to use, with some offering paid subscriptions for additional features
- Market Outreach
  - Reaching to wider audience is easier

Cross-Platform Development Cons

Disadvantages

- Performance
  - Performance similar to native apps but never quite as good as native apps
  - Shouldn’t be using cross-platform development tools if performance is a high priority
- User look and Feel
  - Tools aren’t known for delivering the best graphics and user experiences and can lack access to core OS libraries like graphics
  - Might not be the best option if app relies heavily on graphics
- Single Platform App
  - App to be published on a single platform (e.g. iOS or Android), then should develop a native app
- Platform-Specific Features
  - Tools offer many of the basic features shared between different platforms, they can lack some of the specific features offered by platforms
  - Need to survive with whatever is common across all platforms

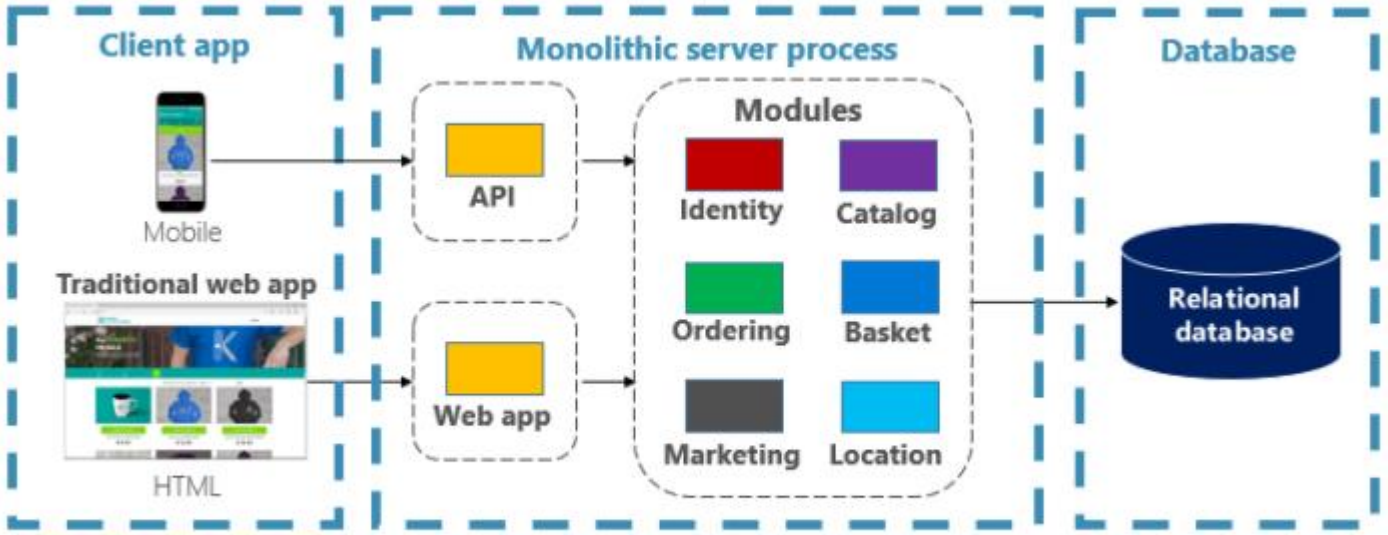
Design “Modern” Web Application

eCommerce App

- Required for start-up
- Should be cutting edge

- You may design
  - A large core application containing all of domain logic
  - And modules such as
    - Identity
    - Catalog
    - Ordering
    - and more

- The core app
  - communicates with a large relational database
  - exposes functionality via an HTML interface



Traditional monolithic design

Source : Microsoft

A monolithic application

Conventional Layered Apps

- Distinct advantages:
  - straightforward to...
    - build
    - test
    - deploy
    - troubleshoot
    - scale

- Many successful apps that exist today were created as monoliths!

- The app is a hit and continues to evolve
  - iteration after iteration
  - adding more and more functionality

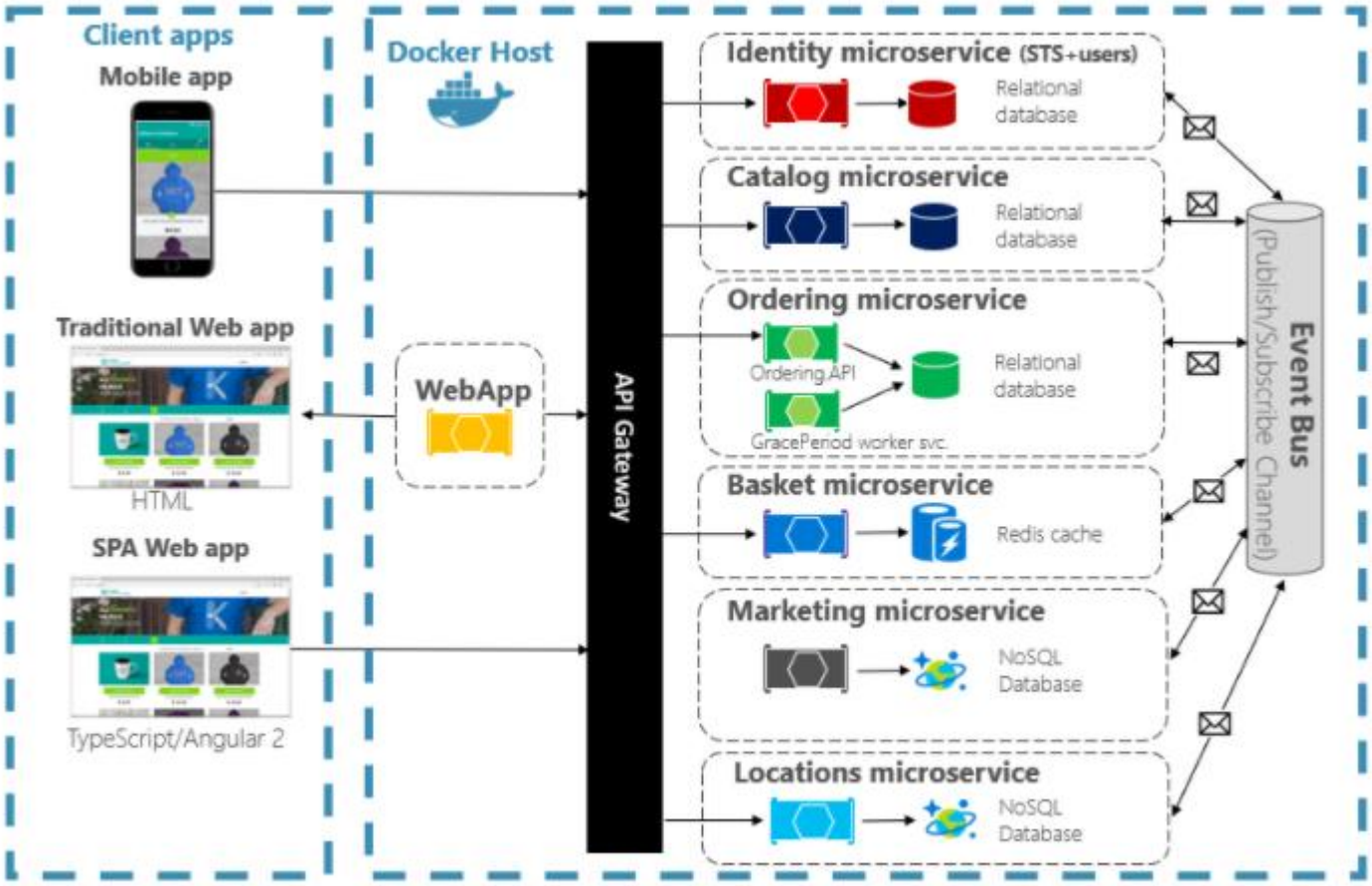
Monolithic Fear Cycle

- At the same time
  - App become overwhelmingly complicated
  - You started losing control of the application
  - Team begin to feel uncomfortable about change in application!

- Concerns:
  - no single person understands it
  - fear making changes - each change has unintended and costly side effects
  - new features/fixes become tricky, time-consuming, and expensive to implement
  - each release requires a full deployment of the entire application
  - one unstable component can crash the entire system

Cloud-native applications

Solution



Cloud-native design

Source : Microsoft

Cloud-native design

Solution explained

- Application is decomposed across a set of small isolated microservices

- Each service is
  - self-contained
  - encapsulates its own code, data, and dependencies
  - deployed in a software container
  - managed by a container orchestrator
  - owns it own data store - relational, no-sql or cache

- API Gateway service
  - All traffic routes through an API Gateway service that is responsible for
  - directing traffic to the core back-end services
  - enforcing many cross-cutting concerns

- Application takes full advantage of the
  - scalability
  - availability
  - resiliency

  features found in modern cloud platforms

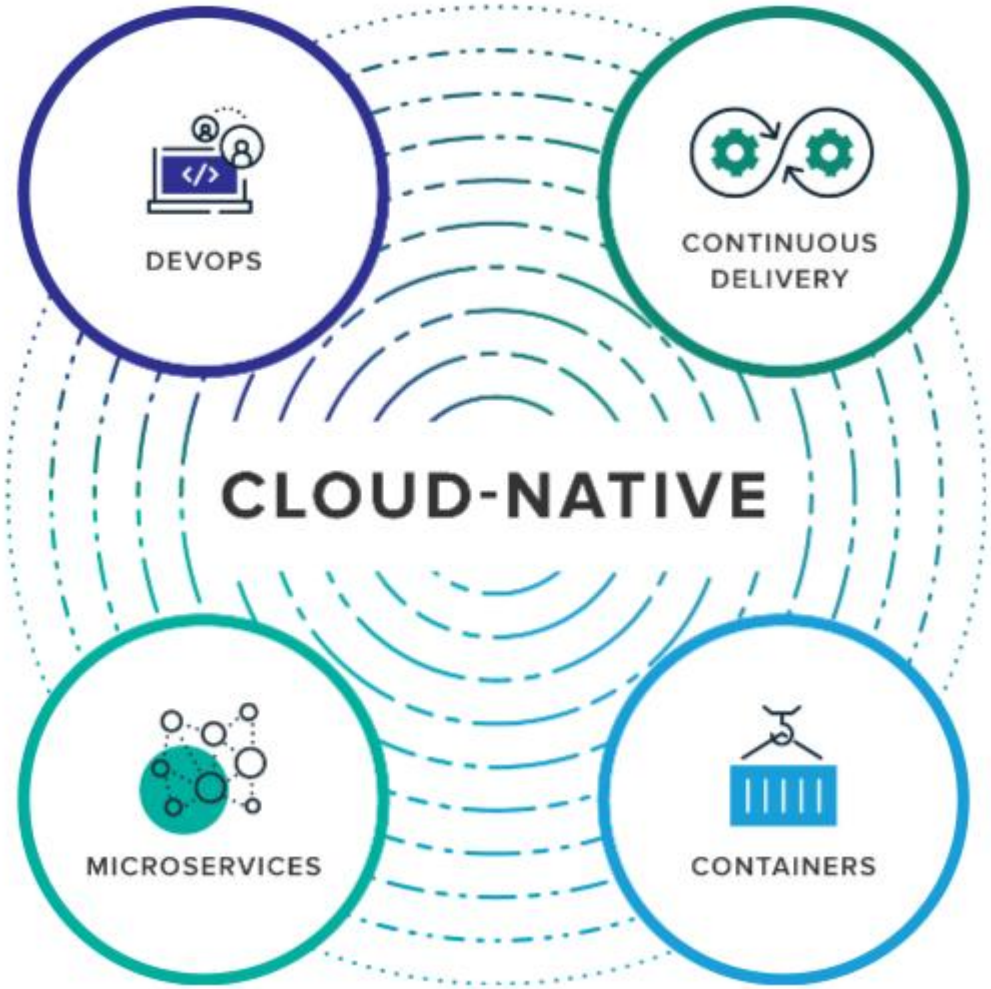
Cloud native

- Is all about changing the way you think about constructing critical business systems
  - embracing rapid change, large scale, and resilience
- An approach to building and running applications that exploits the advantages of the cloud computing delivery model
- Appropriate for both public and private clouds
- Is the ability to offer nearly limitless computing power, on-demand, along with modern data and application services
- Is about how applications are created and deployed, not where!

- The Cloud Native Computing Foundation provides an official definition:

  Cloud-native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.

Cloud native Building Blocks



Source : VMWare Blog

Cloud native Building Blocks (2)

DevOps, Continuous Delivery, Microservices & Containers

- Microservices
  - is an architectural approach to developing an application as a collection of small services
  - each service implements business capabilities, runs in its own process and communicates via HTTP APIs or messaging

- Containers
  - offer both efficiency and speed compared with standard virtual machines (VMs)
  - Using operating system (OS)-level virtualization, a single OS instance is dynamically divided among one or more isolated containers, each with a unique writable file system and resource quota
  - Low overhead of creating and destroying containers combined with the high packing density in a single VM makes containers an ideal compute vehicle for deploying individual microservices

- DevOps
  - Collaboration between software developers and IT operations with the goal of constantly delivering high-quality software that solves customer challenges
  - Creates a culture and an environment where building, testing and releasing software happens rapidly, frequently, and more consistently

- Continuous Delivery
  - is about shipping small batches of software to production constantly, through automation
  - makes the act of releasing reliable, so organizations can deliver frequently, at less risk, and get feedback faster from end users

Source : VMWare Blog

Containers

- "Containers are a great enabler of cloud-native software." - Cornelia Davis

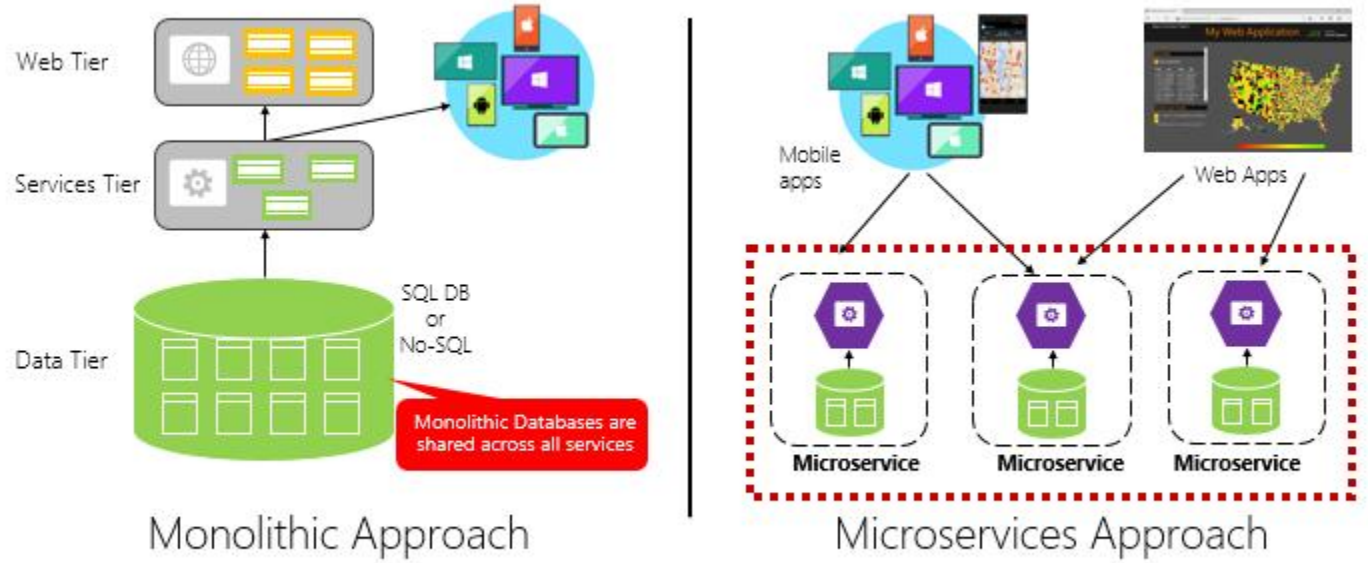
- The Cloud Native Computing Foundation places microservice containerization as the first step in their Cloud-Native Trail Map - guidance for enterprises beginning their cloud-native journey.

- Cloud Native Trail Map

Microservices Architecture

Characteristics

- Each implements a specific business capability within a larger domain context
- Each is developed autonomously and can be deployed independently
- Each is self-contained encapsulating its own data storage technology (SQL, NoSQL) and programming platform
- Each runs in its own process and communicates with others
- Compose together to form app.



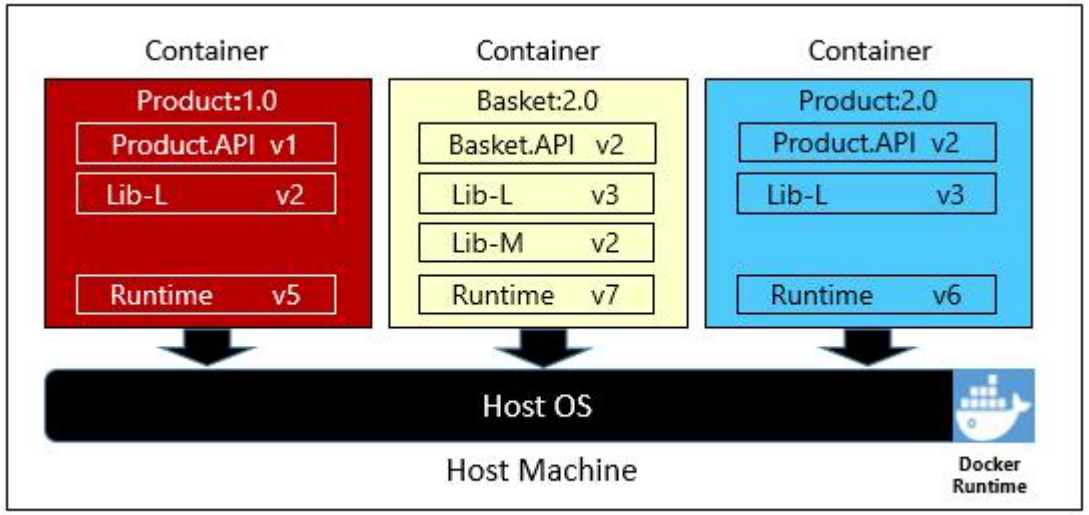
Monolithic deployment versus microservices

Source : Microsoft

Containerizing a Microservices

Simple and straightforward

- The code, its dependencies, and runtime are packaged into a binary called a container image
- Images are stored in a container registry, which acts as a repository or library for images
- A registry can be located on your development computer, in your data center, or in a public cloud
- Docker itself maintains a public registry via Docker Hub
- When needed, transform the image into a running container instance
- The instance runs on any computer that has a container runtime engine installed
- Can have as many instances of the containerized service as needed



Multiple containers running on a container host

Source : Microsoft

Advantages

- Can be easier to manage
  - as iterative improvements occur using Agile and DevOps processes
- Can be improved incrementally and automatically to continuously add new and improved application features
  - as microservices are used
- Can be improved in non-intrusive manner
  - causing no downtime or disruption of the end-user experience
- Scaling up or down proves easier
  - Because of elastic infrastructure that underpins cloud native apps

Source: IBM Blog

Disadvantages

- Create the necessity of managing more elements
  - Rather than one large application, it becomes necessary to manage far more small, discrete services
- Demand additional toolsets
  - to manage the DevOps pipeline, replace traditional monitoring structures, and control microservices architecture
- Allow for rapid development and deployment
  - also demand a business culture that can cope with the pace of that innovation

Cloud native vs. traditional applications

Cloud native vs. Cloud enabled

- A cloud enabled application is an application that was developed for deployment in a traditional data center but was later changed so that it also could run in a cloud environment
- Cloud native applications, however, are built to operate only in the cloud
- Developers design cloud native applications to be
  - Scalable
  - platform agnostic
  - and comprised of microservices

Cloud native vs. traditional applications (2)

Cloud native vs. Cloud ready

- In the history of cloud computing, the meaning of "cloud ready" has shifted several times
  - Initially, the term applied to services or software designed to work over the internet
  - Today, the term is used more often to describe
    - ❖ an application that works in a cloud environment
    - ❖ a traditional app that has been reconfigured for a cloud environment

- The term "cloud native" has a much shorter history
  - Refers to an application developed from
    - ❖ the outset to work only in the cloud and takes advantage of the characteristics of cloud architecture
    - ❖ an existing app that has been refactored and reconfigured with cloud native principles

Cloud native vs. traditional applications (3)

Cloud native vs. Cloud based

- Cloud based
  - A general term applied liberally to any number of cloud offerings
  - A cloud based service or application is delivered over the internet

- Cloud native is a more specific term
  - Cloud native describes applications designed to work in cloud environments
  - The term denotes applications that rely on
    - microservices
    - Containers
    - continuous integration and continuous delivery (CI/CD)
  - can be used via any cloud platform

Cloud native vs. traditional applications (4)

Cloud native vs. Cloud first

- Cloud first
  - describes a business strategy in which organizations commit to using cloud resources first when
    - launching new IT services
    - refreshing existing services
    - replacing legacy technology
  - Cost savings and operational efficiencies drive this strategy

- Cloud native applications pair well with a cloud-first strategy because
  - they use only cloud resources
  - are designed to take advantage of the beneficial characteristics of cloud architecture

Conventional Web Apps

Issues

- Dev Teams build and deploy applications onto the server
- Application runs on that server and dev are responsible for provisioning and managing the resources for it!

- A few issues:
  - Server needs to be up and running even when not serving out any requests
  - Dev teams are responsible for
    - uptime and maintenance of the server and all its resources
    - applying the appropriate security updates to the server
    - managing scaling up server as well

- For smaller companies and individual developers this can be a lot to handle
- At larger organizations this is handled by the infrastructure team
  - However, the processes necessary to support this can end up slowing down development times.

Serverless

Defined

- Serverless architectures are application designs
  - that incorporate third-party “Backend as a Service” (BaaS) services, and/or
  - that include custom code run in managed, ephemeral containers on a “Functions as a Service” (FaaS) platform

- Serverless computing enables developers to build applications faster by eliminating the need for them to manage infrastructure
  - Cloud service provider automatically provisions, scales and manages the infrastructure required to run the code.
- The Serverless name comes from the fact that the tasks associated with infrastructure provisioning and management are invisible to the developer
  - Enables developers to increase their focus on the business logic and deliver more value to the core of the business

- Servers are still running the code!

Source : Serverless

Serverless Apps

Two types

- **BaaS**
  - First used to describe applications that significantly or fully incorporate third-party, cloud-hosted applications and services, to manage server-side logic and state
  - Typically “rich client” applications—think single-page web apps, or mobile apps—that use the vast ecosystem of cloud-accessible databases (e.g., Parse, Firebase), authentication services (e.g., Auth0, AWS Cognito) etc
  - Described as “(Mobile) Backend as a Service" (mBaaS)

- **FaaS**
  - Can also mean applications where server-side logic is still written by the application developer
  - but, unlike traditional architectures, it’s run in stateless compute containers that are
    - event-triggered
    - ephemeral (may only last for one invocation)
    - fully managed by a third party
  - Think it like “Functions as a Service” or "FaaS"
  - Examples:
    - AWS: AWS Lambda
    - Microsoft Azure: Azure Functions
    - Google Cloud: Cloud Functions

Serverless – Changes Required

- Microservices and Functions
  - The biggest change faced with while transitioning to a Serverless world is that application needs to be architectured in the form of small functions
  - functions are typically run inside secure (almost) stateless containers
  - Typically required to adopt a more microservices based architecture

- Workaround
  - Can get around this by running entire application inside a single function as a monolith and handling the routing yourself
    - But this isn't recommended since it is better to reduce the size of functions

- Cold Starts
  - Functions are run inside a container that is brought up on demand to respond to an event, there is some latency associated with it - Cold Start
  - Improved over period of time!

Serverless - Compared

- Benefits
  - Reduced operational cost
  - BaaS: reduced development cost
  - FaaS: scaling costs
    - ❖ occasional requests
    - ❖ inconsistent traffic
  - Easier operational management
  - Reduced packaging and deployment complexity
  - Helps with "Greener" computing

- Drawbacks
  - Vendor control
  - Vendor lock in
  - Multitenancy problems
  - Security concerns

AWS

Serverless Offering

- AWS provides a set of fully managed services that can be used to build and run serverless applications
  - Serverless applications don't require provisioning, maintaining, and administering servers for backend components such as compute, databases, storage, stream processing, message queueing, and more
  - No longer need to worry about ensuring application fault tolerance and availability
  - AWS handles all of these capabilities for applications!

- AWS Lambda
  - Allows to run code without provisioning or managing servers

- AWS Fargate
  - Purpose-built serverless compute engine for containers

- Amazon API Gateway
  - Fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale

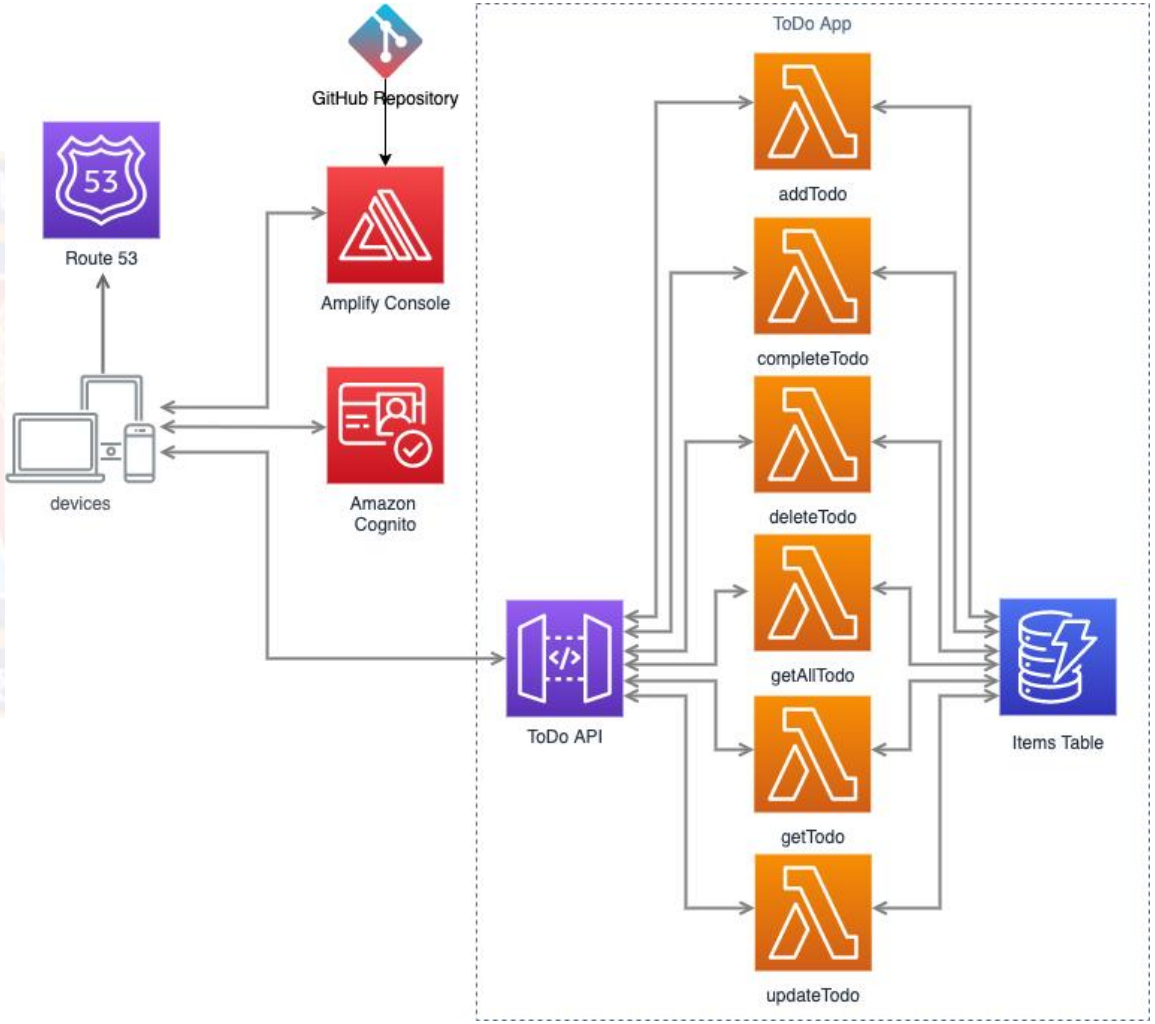
Source : AWS Serverless

AWS Serverless Architecture

Reference Architecture – Web Application

- General-purpose, event-driven, web application back-end that uses
  - AWS Lambda, Amazon API Gateway for its business logic
- Uses Amazon DynamoDB
  - as its database
- Uses Amazon Cognito
  - for user management
- All static content is hosted using AWS Amplify Console

- This application implements a simple To Do app, in which a registered user can
  - create, update, view the existing items, and eventually, delete them



Source : AWS

Google Cloud Platform

Serverless computing

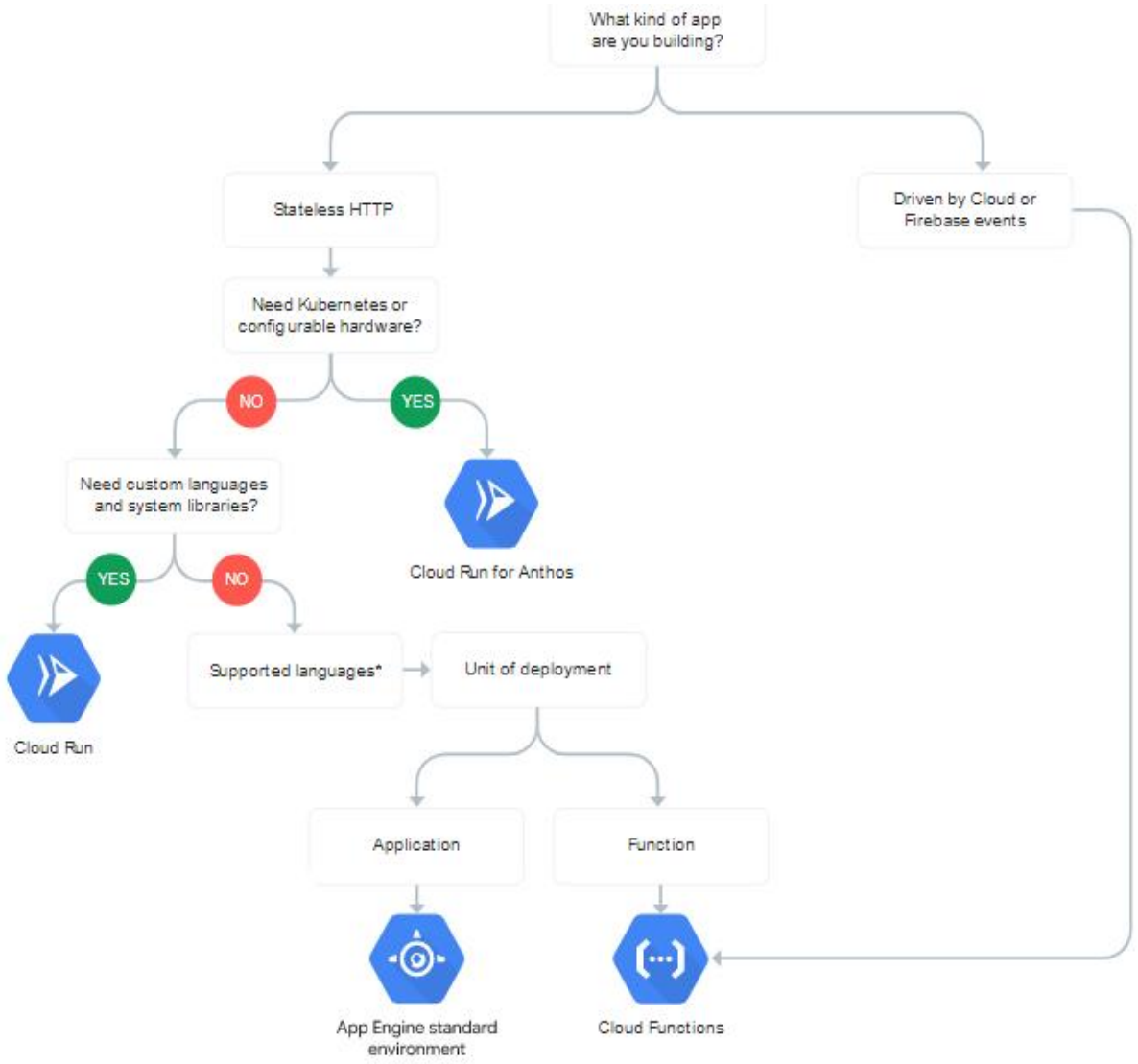
- Google Cloud's serverless platform lets you write code your way without worrying about the underlying infrastructure
  - Deploy functions or apps as source code or as containers
  - Build full stack serverless applications with Google Cloud's storage, databases, machine learning, and more
  - Easily extend applications with event-driven computing from Google or third-party service integrations
  - You can even choose to move your serverless workloads to on-premises environments or to the cloud

Source : Google Product Page

Google Cloud Platform (2)

Services Offered

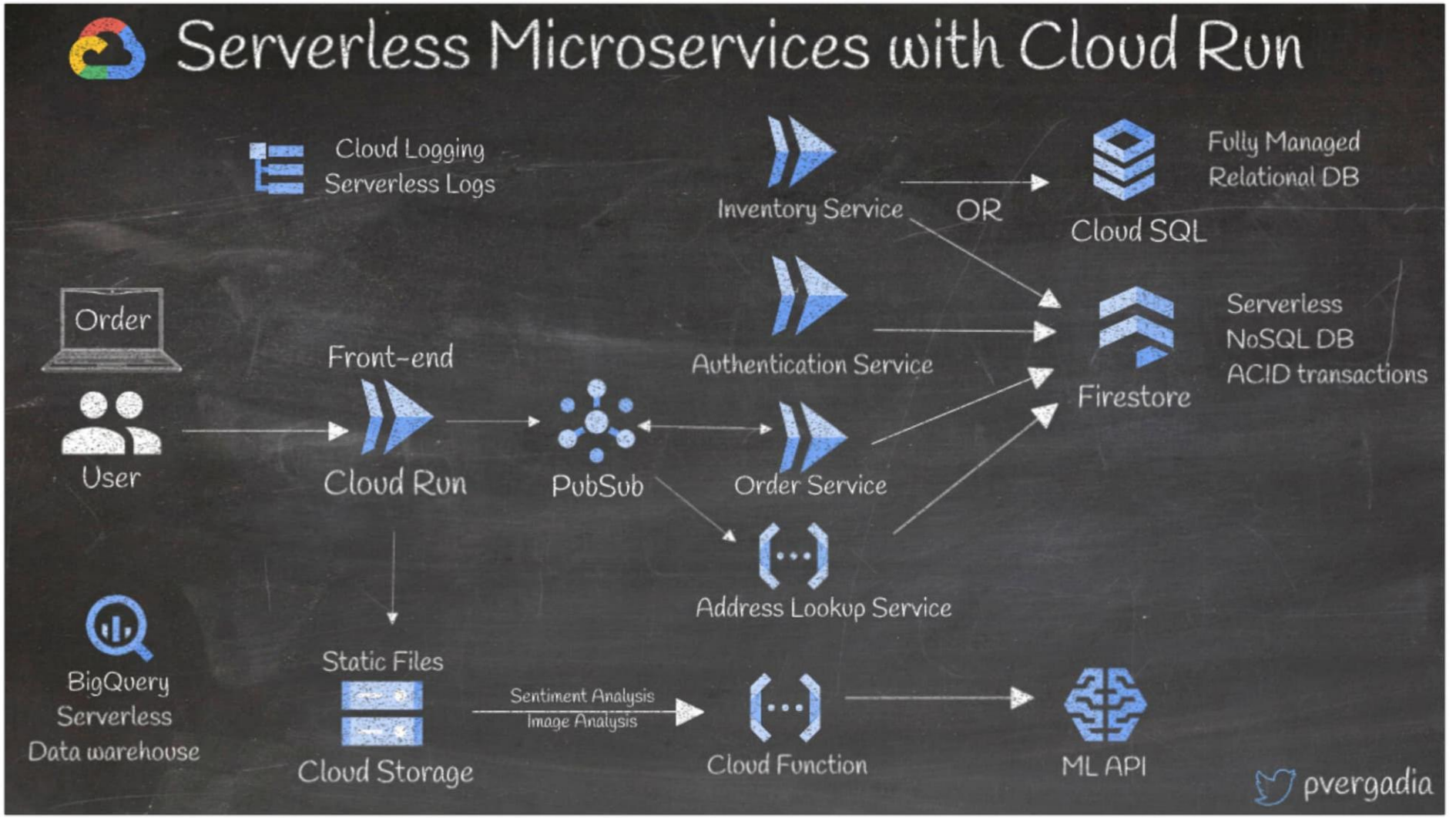
- Cloud Functions
  - An event-driven compute platform to easily connect and extend Google and third-party cloud services and build applications that scale from zero to planet scale.
- App Engine standard environment
  - A fully managed Serverless application platform for web and API backends. Use popular development languages without worrying about infrastructure management.
- Cloud Run
  - A Serverless compute platform that enables you to run stateless containers invocable via HTTP requests
  - Cloud Run is available as a fully managed, pay-only-for-what-you-use platform and also as part of Anthos.



Source : Google Product Page

GCP Serverless Architecture

Example – Order Processing



Source : Google

GCP Serverless Architecture (2)

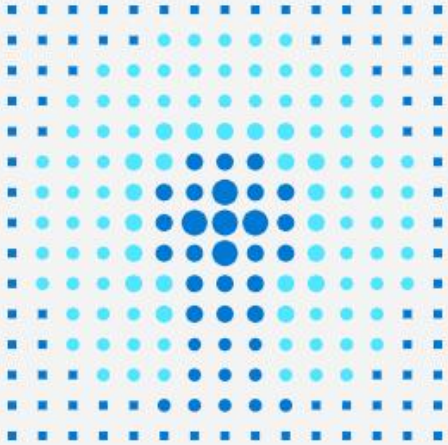
Example - Flow

- A good way to build a serverless microservice architecture on Google Cloud is to use Cloud Run.

- Example of an e-commerce app:
  - When a user places an order, a frontend on Cloud Run receives the request and sends it to Pub/Sub, an asynchronous messaging service.
  - The subsequent microservices, also deployed on Cloud Run, subscribe to the Pub/Sub events.
  - Let's say the authentication service makes a call to Firestore, a serverless NoSQL document database.
  - The inventory service queries the DB either in a CloudSQL fully managed relational database or in Firestore
  - Then the order service receives an event from Pub/Sub to process the order.
  - The static files are stored in Cloud Storage, which can then trigger a cloud function for data analysis by calling the ML APIs.
  - There could be other microservices like address lookup deployed on Cloud Functions.
  - All logs are stored in Cloud Logging.
  - BigQuery stores all the data for serverless warehousing.

Microsoft Azure

Serverless application patterns



Serverless functions

Serverless functions accelerate development by using an event-driven model, with triggers that automatically execute code to respond to events and bindings to seamlessly integrate additional services. A pay-per-execution model with sub-second billing charges only for the time and resources it takes to execute the code.

Serverless Kubernetes

Developers bring their own containers to fully managed, Kubernetes-orchestrated clusters that can automatically scale up and down with sudden changes in traffic on spiky workloads.

Serverless workflows

Serverless workflows take a low-code/no-code approach to simplify orchestration of combined tasks. Developers can integrate different services (either cloud or on-premises) without coding those interactions, having to maintain glue code or learning new APIs or specifications.

Serverless application environments

With a serverless application environment, both the back end and front end are hosted on fully managed services that handle scaling, security and compliance requirements.

Serverless API gateway

A serverless API gateway is a centralised, fully managed entry point for serverless backend services. It enables developers to publish, manage, secure and analyse APIs at global scale.

Source : MS Serverless

Microsoft Azure Serverless Architecture

Example – Architect scalable e-commerce web app

1 User accesses the web app in browser and signs in.

2 Browser pulls static resources such as images from Azure Content Delivery Network.

3 User searches for products and queries SQL database.

4 Web site pulls product catalog from database.

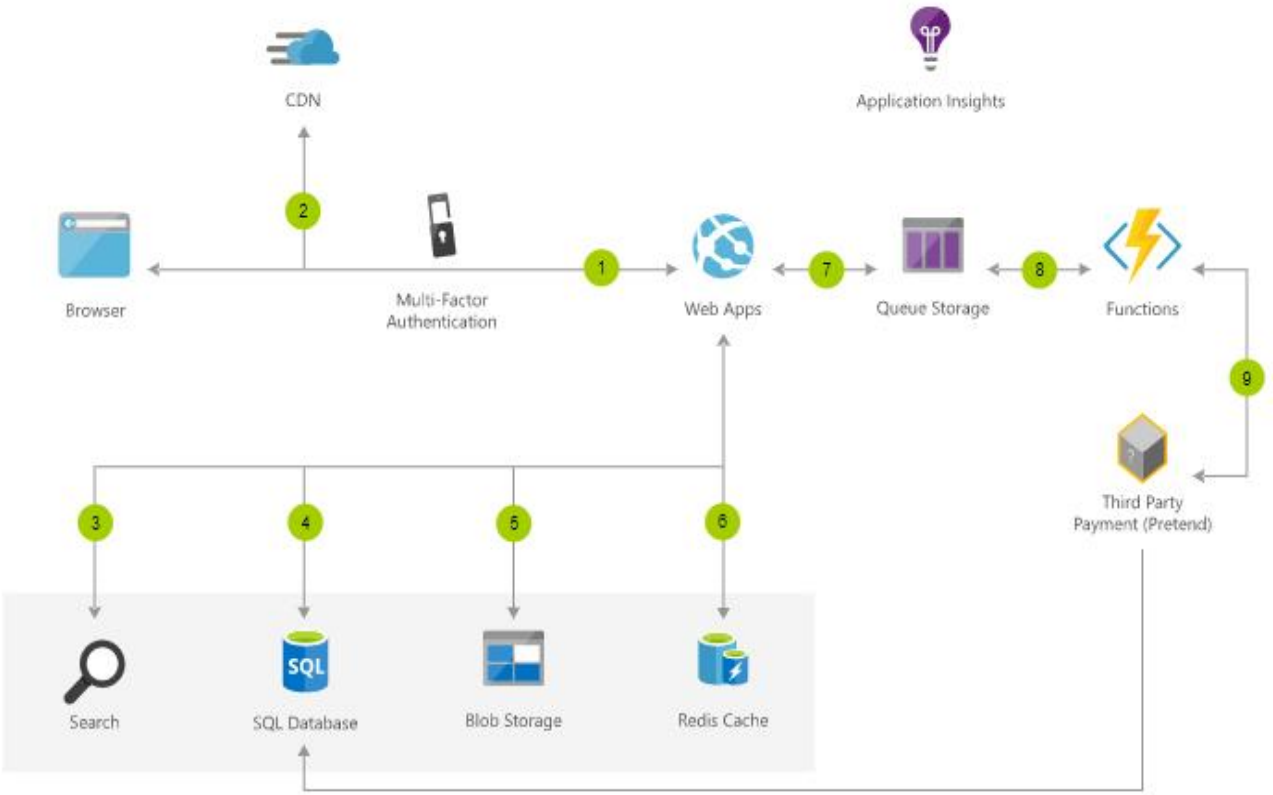
5 Web app pulls product images from Blob Storage.

6 Page output is cached in Azure Cache for Redis for better performance.

7 User submits order and order is placed in the queue.

8 Azure Functions processes order payment.

9 Azure Functions makes payment to third party and records payment in SQL database.

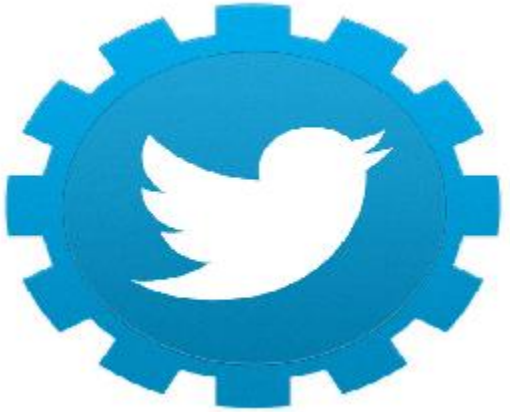


Source : MS Serverless

API

API stands for ‘Application Programming Interface'

- Technically,
  - ✓ an API describes how to connect a dataset or business process with some sort of consumer application or another business process



- We are probably familiar with a lot of the big names that use APIs all the time

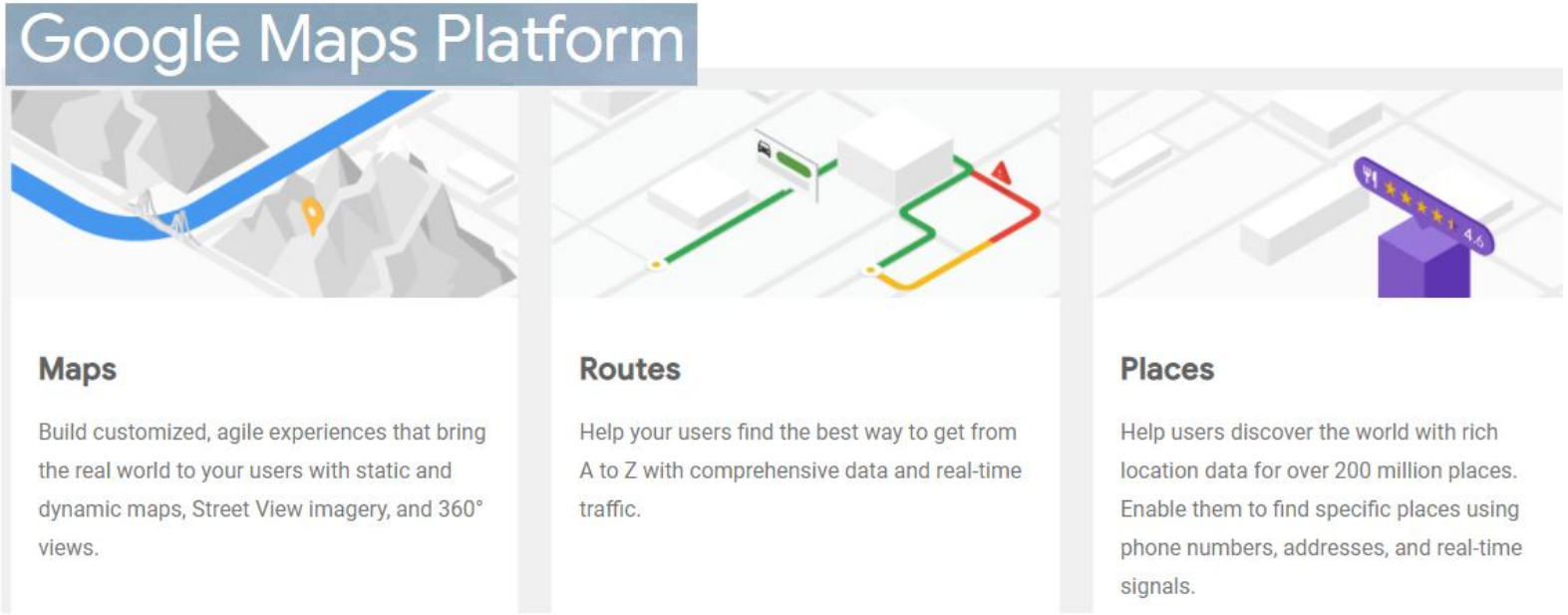
- For example,
- whenever use Facebook account to join another site, login request is being routed via an API
- whenever use the Share functions of an application on your mobile device, those apps are using APIs to connect you to Twitter, Instagram, etc.

Mapping API

Google Maps

- When you search for an address, an API helps interact with a map database to identify the latitude and longitude, and other related data, for that address

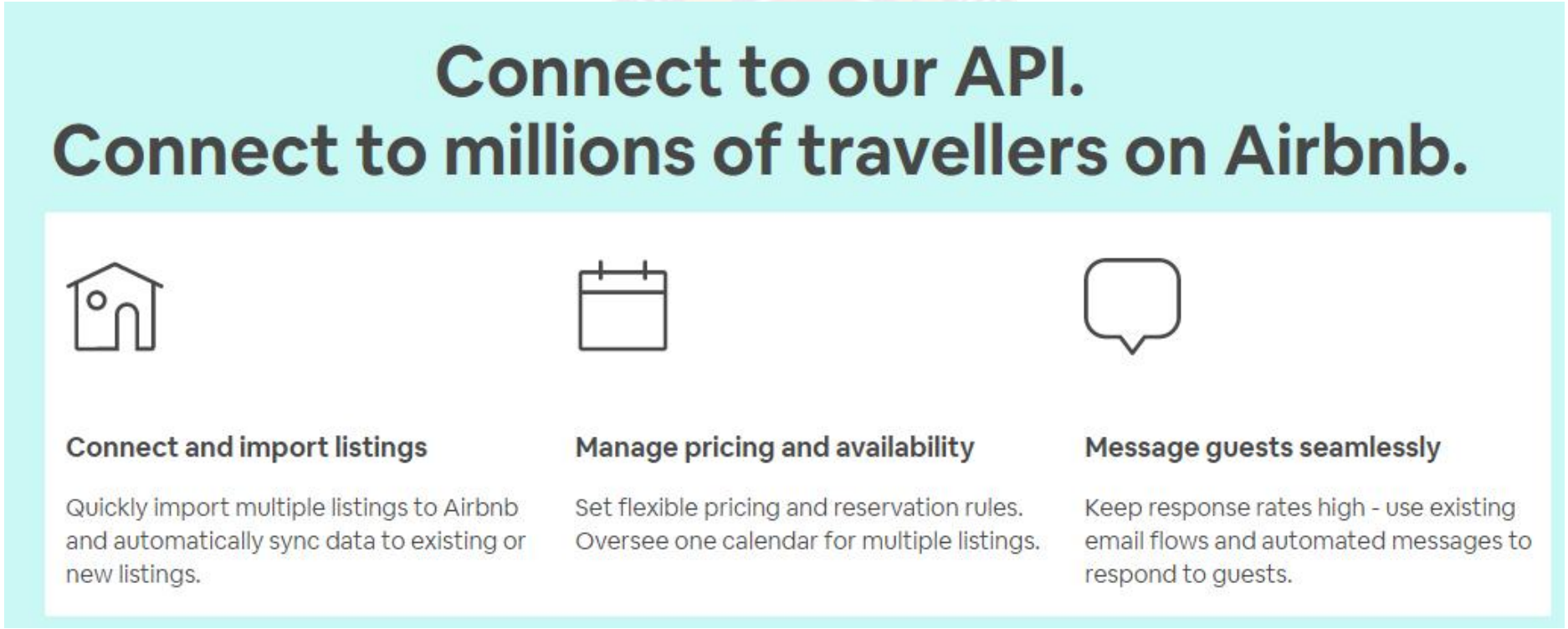
- The API also makes it possible for a mapping interface to then display
  - ✓ the address on the map
  - ✓ any additional information such as the directions to that destination



Business APIs

Airbnb

- When you search for hotel online, API helps you
  - ✓ to interact with hotels database
  - ✓ filter out the entries based upon your specification
  - ✓ Do the booking



Payment APIs

- Payment APIs are APIs (Application Programming Interfaces) designed for managing payments.

- They enable eCommerce sites to process:
- credit cards,
- track orders,
- and maintain customers lists.

- In many instances, they can help protect merchants from fraud and information breaches.

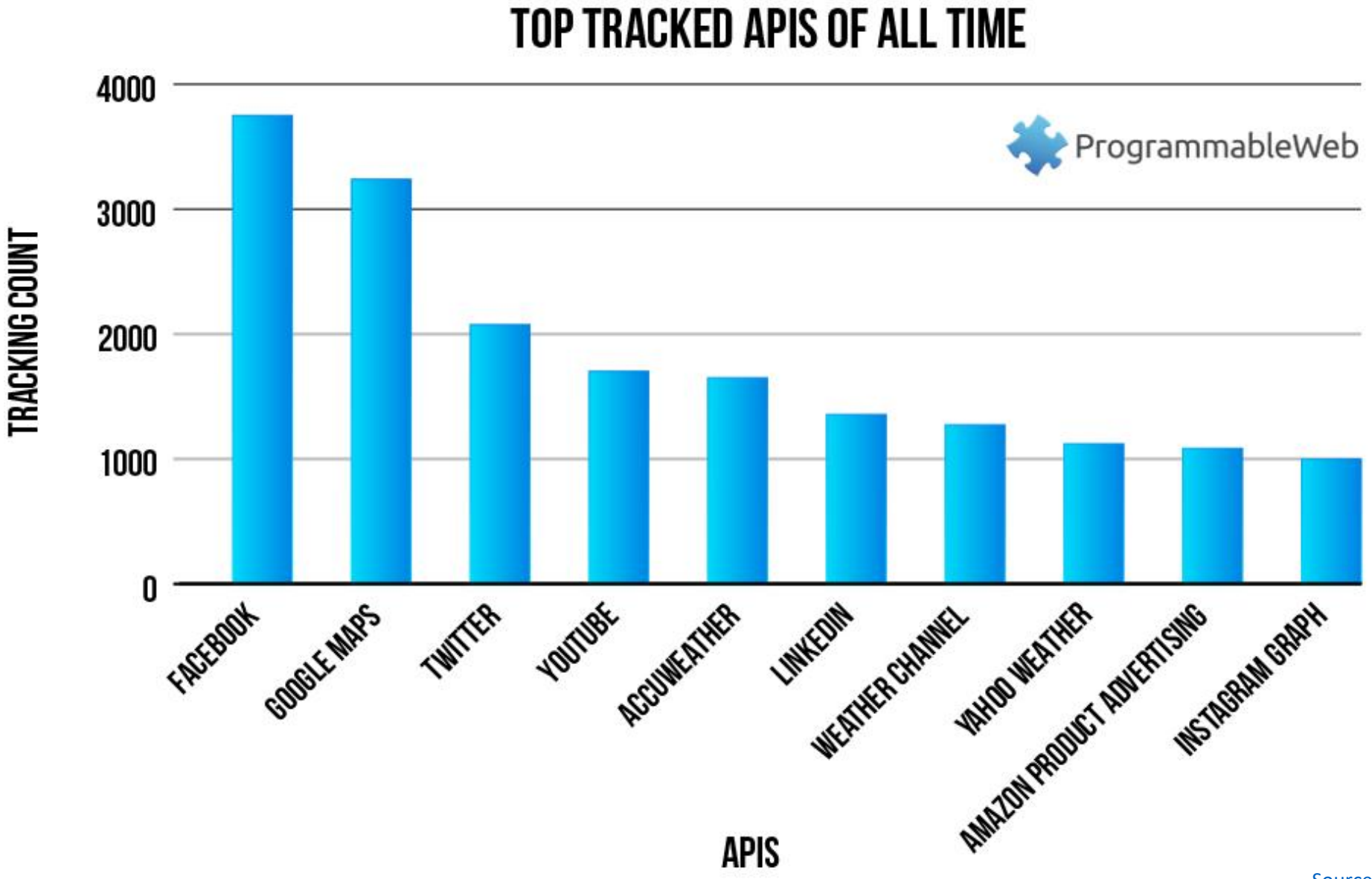
Source : rapidapi

Marketing API

- Marketing APIs are a collection of API endpoints that can be used to help you advertise on social media platform like Facebook, Twitter etc.
- Amazon offers vendors and professional sellers the opportunity to advertise their products on Amazon

Source : pinterest

Popular APIs



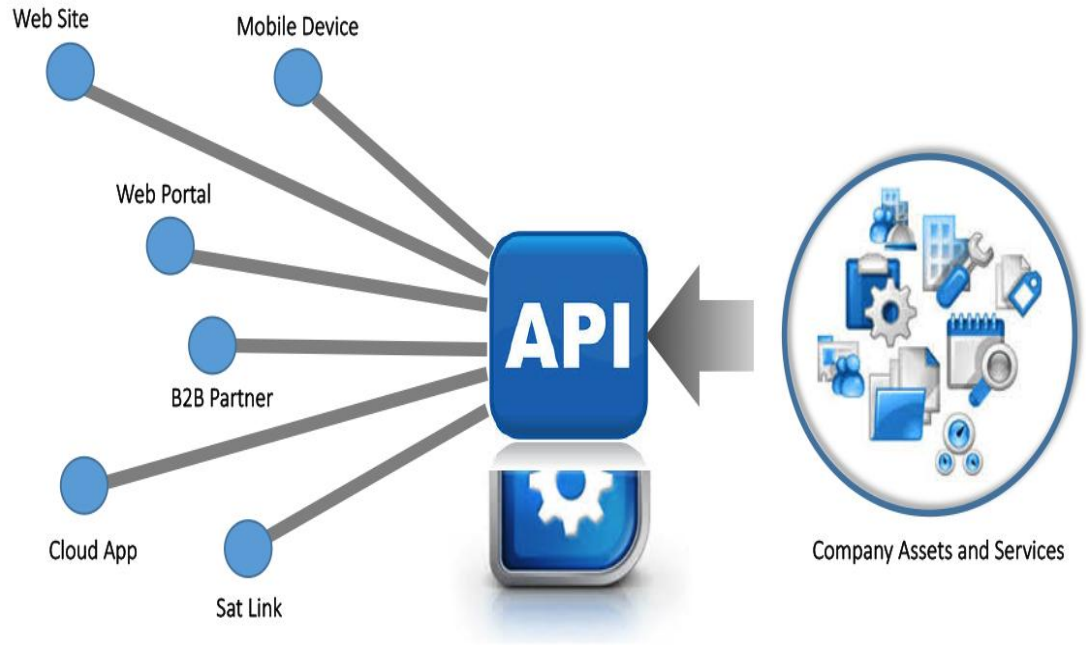
Source: programmable web

APIs

- API services are much more than just a description of how to access a database or how to help a machine read the data!

- Enable a business to become a platform.
  - ✓ APIs help you break down your business products and services into compassable functions you can share with other businesses for direct insertion into their processes.

- APIs provide a way for businesses to leverage new markets
  - ✓ allow partners and third-party developers to access a business' database assets, or create a seamless workflow that accesses a business' services.



Source : forum systems

What are APIs?

Application Programming Interface

- Application:
  - App provides a function, but it's not doing this all by itself—it needs to communicate both with the user, and with the backend it's accessing
  - App deals in both inputs and outputs
  - May be a customer-facing app like a travel booking site, or a back-end app like server software that funnels requests to a database
  - Think of an application like an ATM - provides you services by interacting with Bank

- Programming:
  - Engineering part of the app's software that translates input into output
  - Accepts, validates and processes user request
  - For example, in ATM case
    - translates request for cash to the bank's database
    - verifies there's enough cash in account to withdraw the requested amount
    - the bank grants permission
    - then the ATM communicates back to the bank how much money to withdraw
    - bank can update account balance

- Interface:
  - Interfaces are how we communicate with a machine
  - With APIs, it's much the same, only we're replacing users with software
  - In the case of the ATM, it's the screen, keypad, and cash slot—where the input and output occurs

- API: an interface that software uses to access whatever resource it needs: data, server software, or other applications

The Components of APIs

What resources are shared and with whom?

- Shared assets / Resources
  - Are the currency of an API
  - Can be anything a company wants to share - data points, pieces of code, software, or services that a company owns and sees value in sharing
- API
  - Acts as a gateway to the server
  - Provides a point of entry for developers to use those assets to build their own software
  - Can acts like a filter for those assets - only reveal what you want them to reveal
- Audience
  - Immediate audience of an API is rarely an end user of an app
  - Typically developers creating software or an app around those assets
- Apps
  - Results in apps that are connected to data and services, allowing these apps to provide richer, more intelligent experiences for users
  - API-powered apps are also compatible with more devices and operating systems
  - Enable end users tremendous flexibility to access multiple apps seamlessly between devices, use social profiles to interact with third-party apps etc.

API

Benefits

- Acts as a doorway that people with the right key can get through
  - a gateway to the server and database that those with an API key can use to access whatever assets you choose to reveal
- Lets applications (and devices) seamlessly connect and communicate
  - With API one can create a seamless flow of data between apps and devices in real time
  - Enables developers to create apps for any format—a mobile app, a wearable, or a website
  - Allows apps to “talk to” one another - heart of how APIs create rich user experiences
- Let you build one app off another app
  - Allows you to write applications that use other applications as part of their core functionality
  - Developers get access to reusable code and technology
  - Other technology gets automatically leverages for your own apps
- Acts like a “universal plug”
  - Apps in Different languages does not matter
  - Everyone, no matter what machine, operating system, or mobile device they’re using—gets the same access
  - Standardizes access to app and its resources
- Acts as a filter
  - Security is a big concern with APIs
  - Gives controlled access to assets, with permissions and other measures that keep too much traffic

Types

Public APIs vs. Private APIs - Very Different Value Chains

- Public API
  - Twitter API, Facebook API, Google Maps API, and more
  - Granting Outside Access to Your Assets
  - Provide a set of instructions and standards for accessing the information and services being shared
  - Making it possible for external developers to build an application around those assets
  - Much more restricted in the assets they share, given they're sharing them publicly with developers around the web

- Private API
  - Self-Service Developer & Partner Portal API
  - Far more common (and possibly even more beneficial, from a business standpoint)
  - Give developers an easy way to plug right into back-end systems, data, and software
  - Letting engineering teams do their jobs in less time, with fewer resources
  - All about productivity, partnerships, and facilitating service-oriented architectures

REST

Representation State Transfer

- Most commonly known item in API space
- has become very common amongst web APIs
- First defined by Roy Fielding in his doctoral dissertation in the year 2000
- Architectural system defined by a set of constraints for web services based on
  - stateless design ethos
  - standardized approach to building web APIs
- Operations are usually defined using GET, POST, PUT, and other HTTP methodologies
- One of the chief properties of REST is the fact that it is hypermedia rich
- Supports a layered architecture, efficient caching, and high scalability
- All told, REST is a very efficient, effective, and powerful solution for the modern micro service API industry

gRPC

Backed by Google

- Actually a new take on an old approach known as RPC, or Remote Procedure Call
- RPC is a method for executing a procedure on a remote server
- RPC functions upon an idea of contracts, in which the negotiation is defined and constricted by the client-server relationship rather than the architecture itself
  - RPC gives much of the power (and responsibility) to the client for execution
  - offloading much of the handling and computation to the remote server hosting the resource
- RPC is very popular for IoT devices and other solutions requiring custom contracted communications for low-power devices
  - gRPC is a further evolution on the RPC concept, and adds a wide range of features
- The biggest feature added by gRPC is the concept of protobufs
  - Protobufs are language and platform neutral systems used to serialize data, meaning that these communications can be efficiently serialized and communicated in an effective manner
- gRPC has a very effective and powerful authentication system that utilizes SSL/TLS through Google's token-based system
- Open source, meaning that the system can be audited, iterated, forked, and more

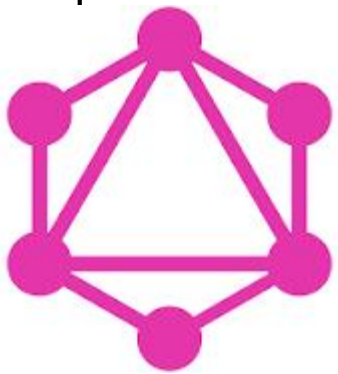
GraphQL

Backed by Facebook

- Approach to the idea of client-server relationships is unique amongst all other options
- GraphQL is a query language for APIs and a runtime for fulfilling those queries with existing data
- Client determines what data they want, how they want it, and in what format they want it in
  - Reversal of the classic dictation from the server to the client
  - Allows for a lot of extended functionality

- A huge benefit of GraphQL is - it typically delivers the smallest possible request
- More useful in specific use cases where
  - a needed data type is well-defined
  - a low data package is preferred

- Defines a “new relationship between client and data”



REST vs gRPC vs GraphQL

When to use?

- REST
  - A stateless architecture for data transfer that is dependent on hypermedia
  - Tie together a wide range of resources that might be requested in a variety of formats for different purposes
  - Systems requiring rapid iteration and standardized HTTP verbiage will find REST best suited for their purposes

- gRPC
  - A nimble and lightweight system for requesting data
  - Best used when a system requires a set amount of data or processing routinely and requester is either low power or resource-jealous
  - IoT is a great example

- GraphQL
  - An approach wherein the user defines the expected data and format of that data
  - Useful in situations in which the requester needs the data in a specific format for a specific use

Thank You!

In our next session:

Slides contribution from prof Pravin Y Pawar