



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

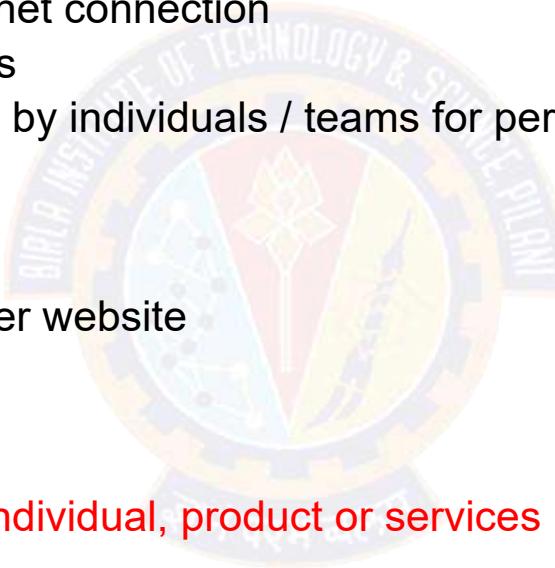
# Web Apps

Chandan Ravandur N

# Website

## What & Why?

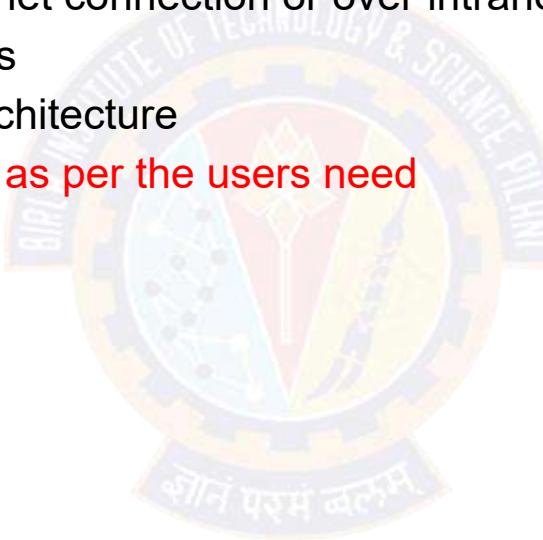
- A group of interlinked web pages having a single domain name
  - Hosted on a webserver
  - Accessible over the web with internet connection
  - Easily accessible through browsers
  - Can be developed and maintained by individuals / teams for personal or businesses usage
- For example,
  - BITS Pilani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

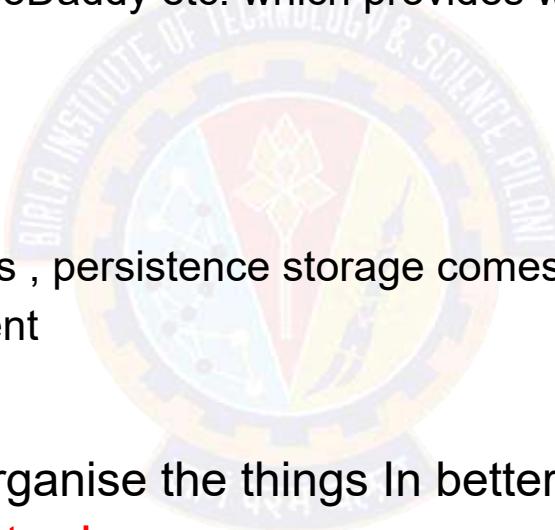
- Lets Visit!



# Web Apps Architecture

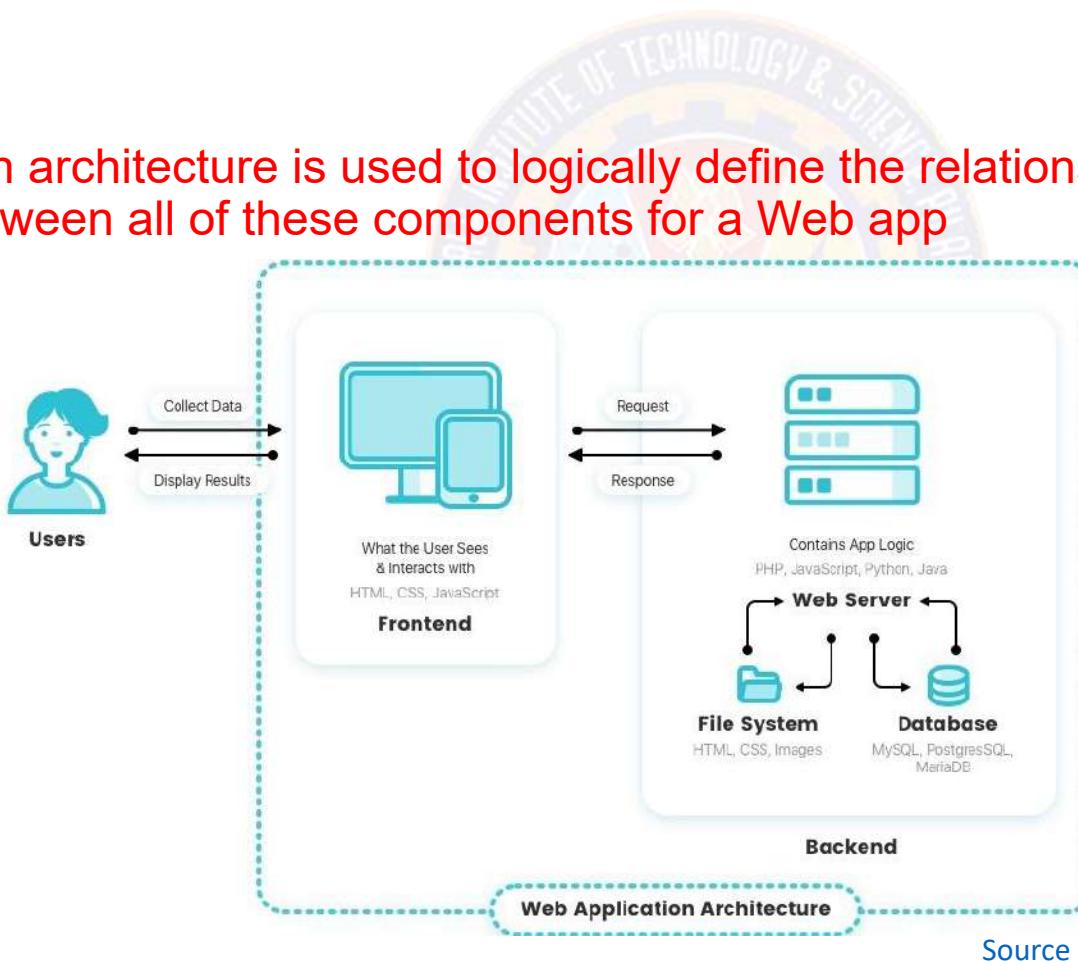
## Why?

- 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

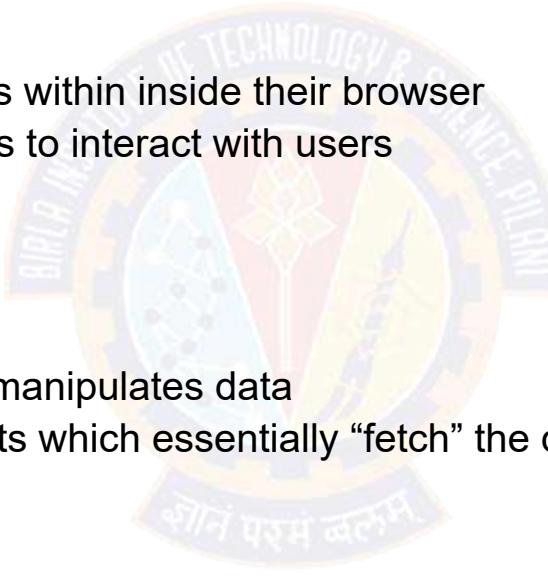


Source

# Web App Components

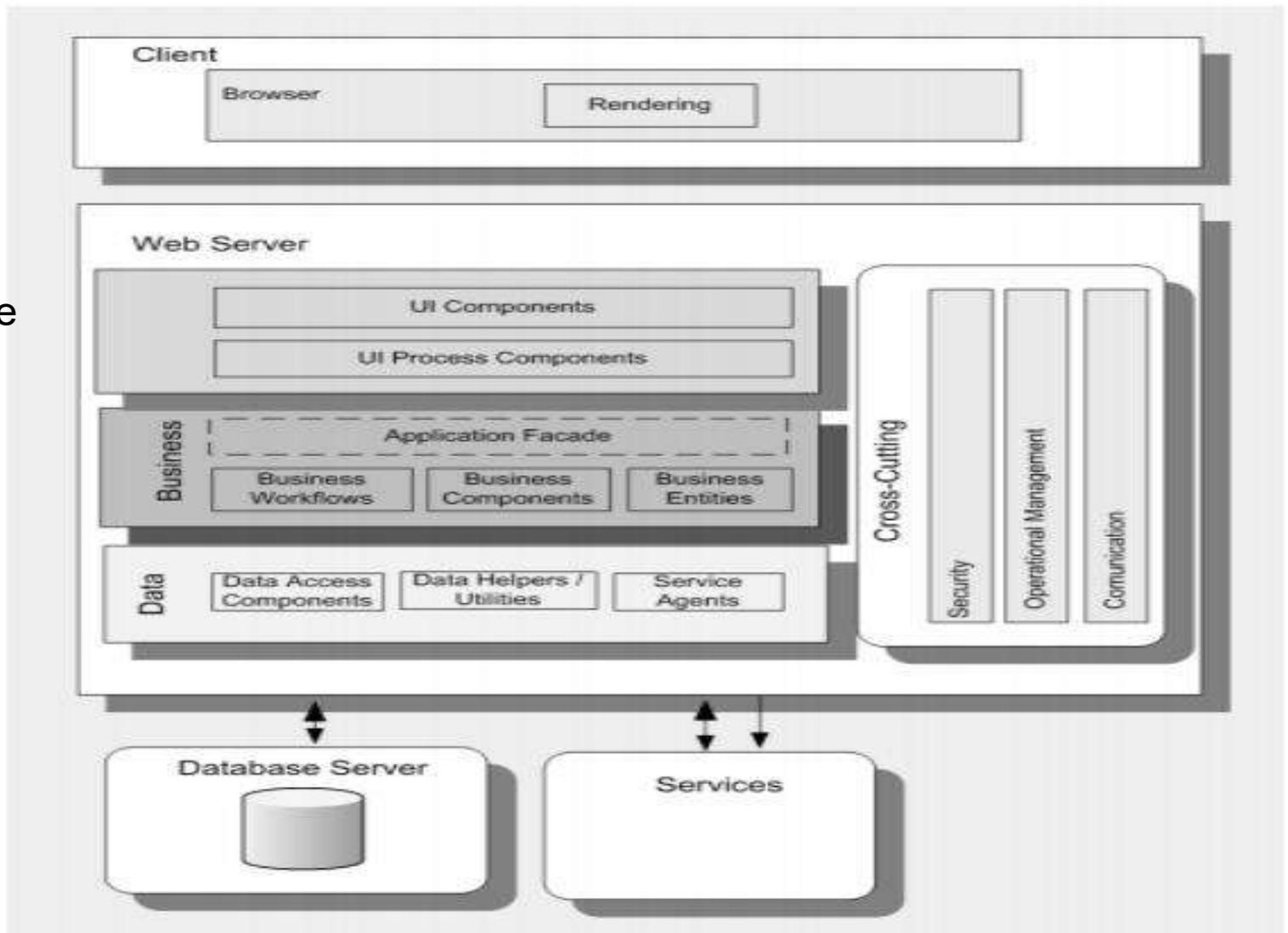
## Frontend , Backend and Databases

- 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 with 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

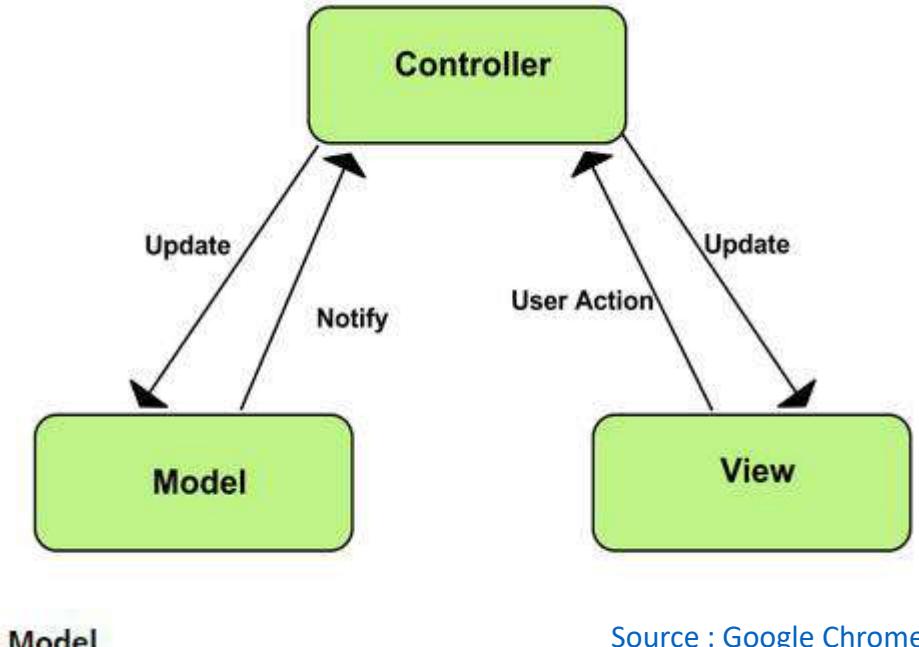
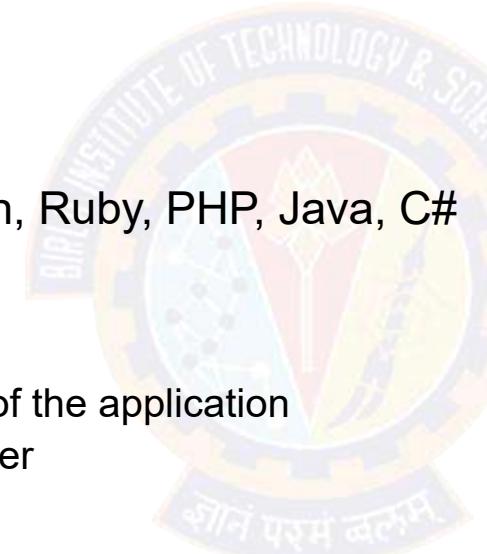
- The core of a Web application is
  - its server-side logic
- The Web application layer itself can be comprised of many distinct layers
  - Typically a three-layered architecture
  - comprised of
    - ❖ Presentation
    - ❖ Business
    - ❖ data layers



# 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
  - 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



Model

Source : Google Chrome

# 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

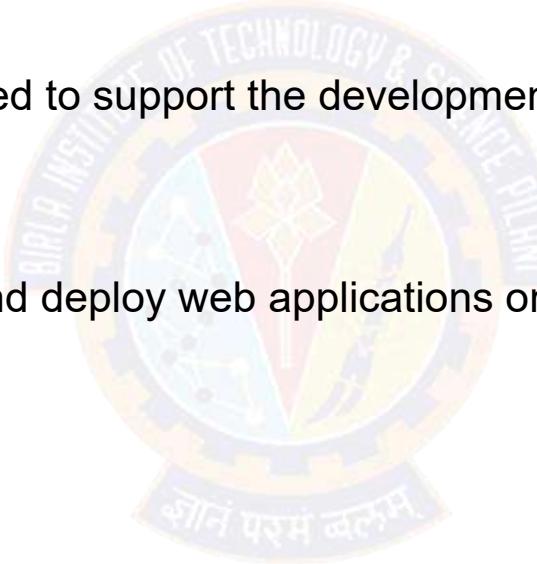
## SSR Vs CSR

	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

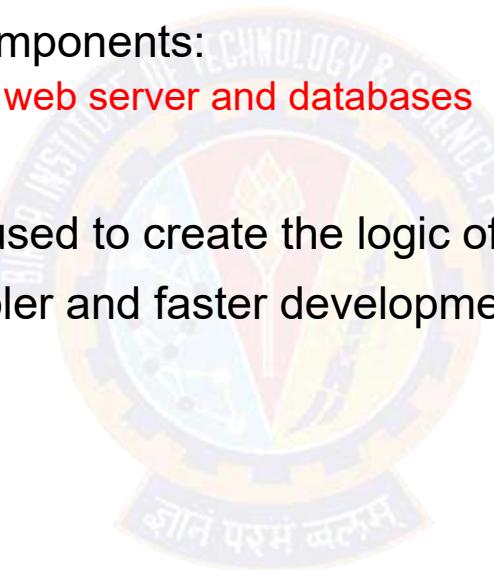
## Frontend / User Interface

- 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

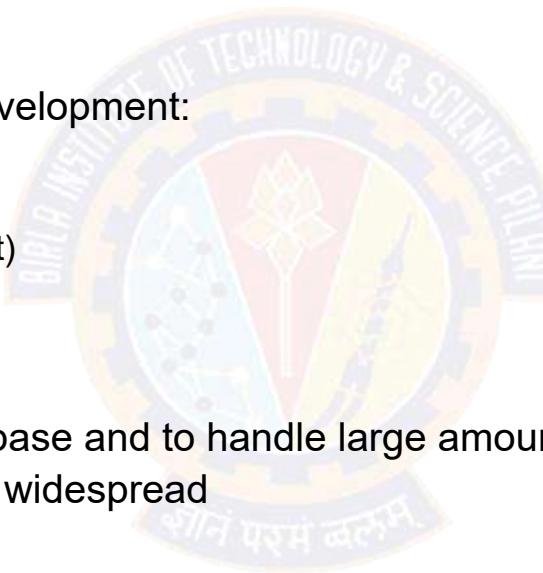
- Not visible to users, but it powers the client side
- Major server side technology stack components:
  - Programming language, Framework , web server and databases
- Server-side programming languages used to create the logic of applications
- Frameworks offer lots of tools for simpler and faster development of applications
- Options
  - Ruby (Ruby on Rails)
  - Python (Django, Flask, Pylons)
  - PHP (Laravel)
  - Java (Spring)
  - Scala (Play)
  - Javascript (Node.js)



# Server-Side Programming

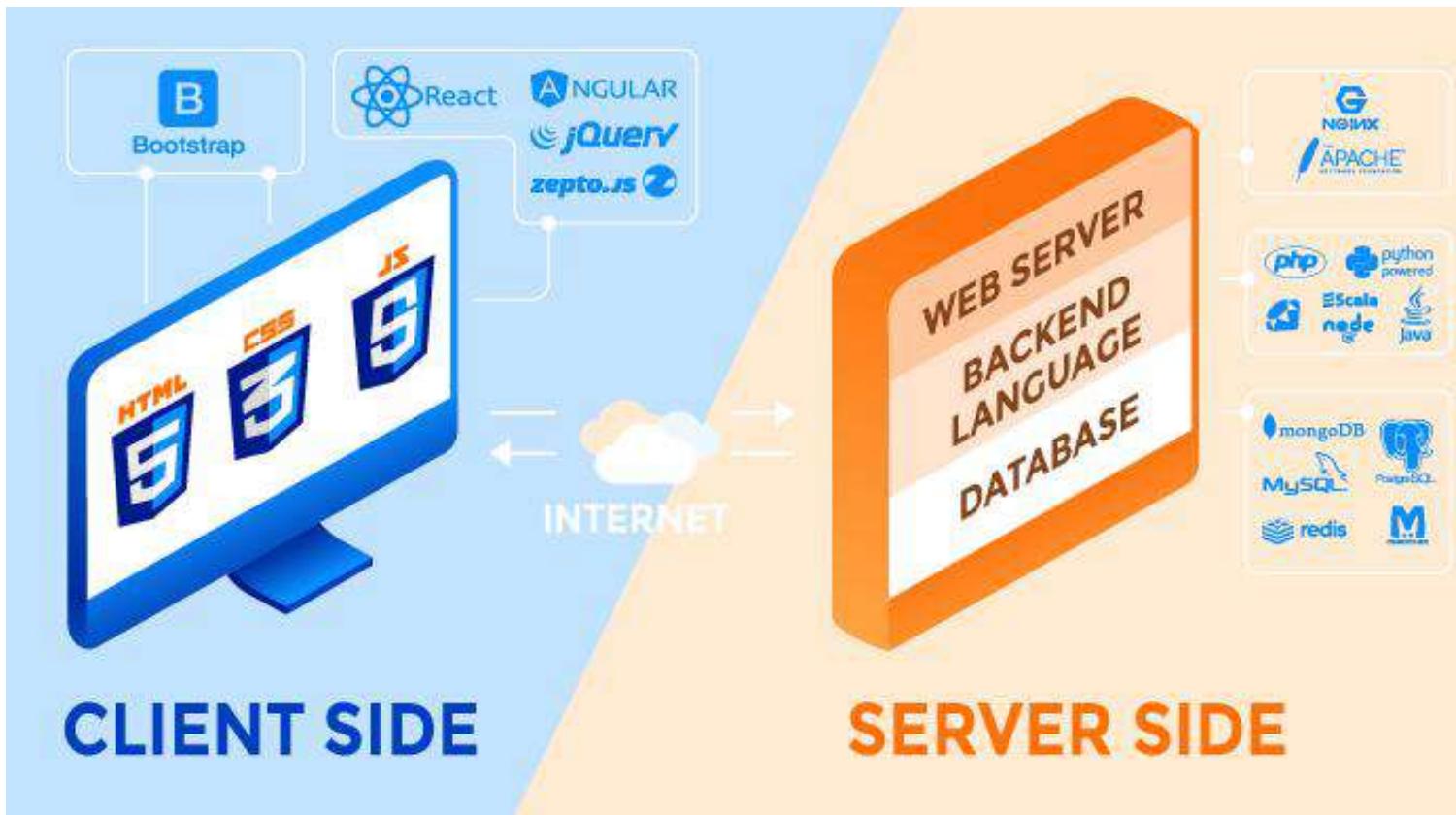
## Other Components

- 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

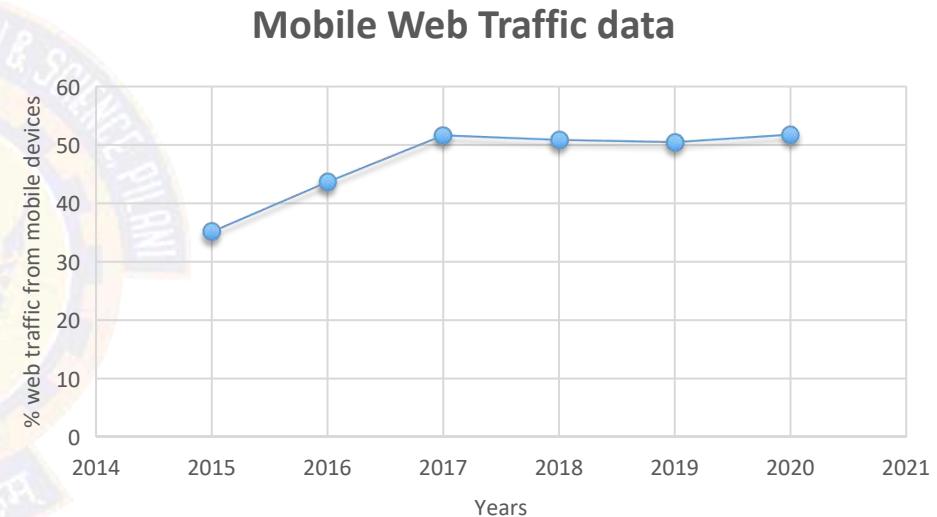


Source : [RubyGarage](#)

# 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

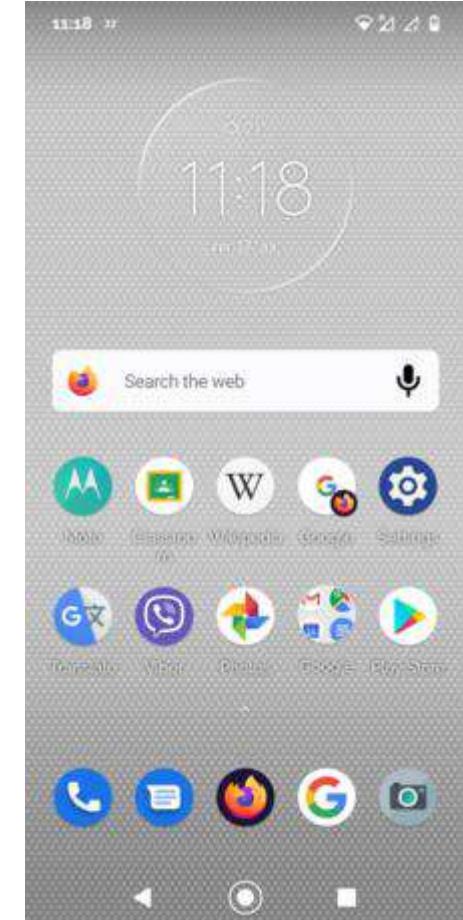


Image source : Wikipedia

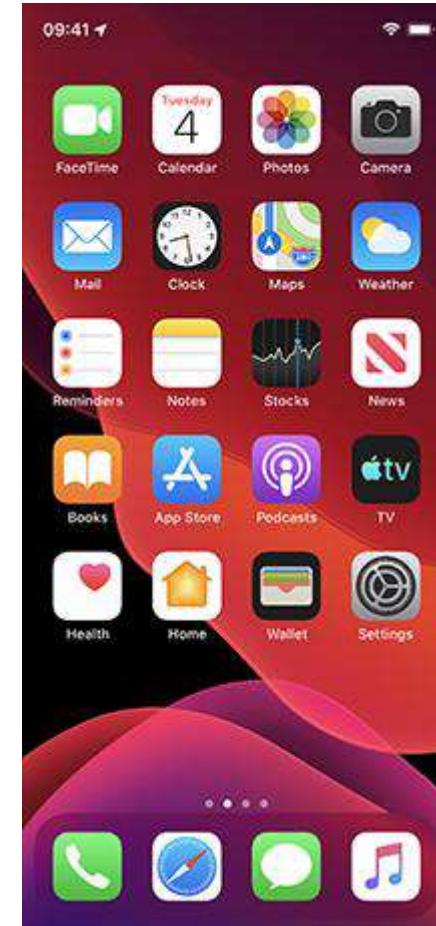
# Mobile Apps

## 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

android

iOS



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



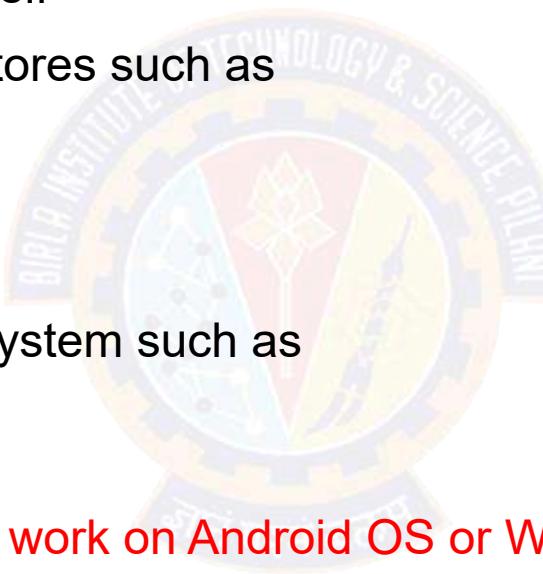
# Mobile Apps - Types

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



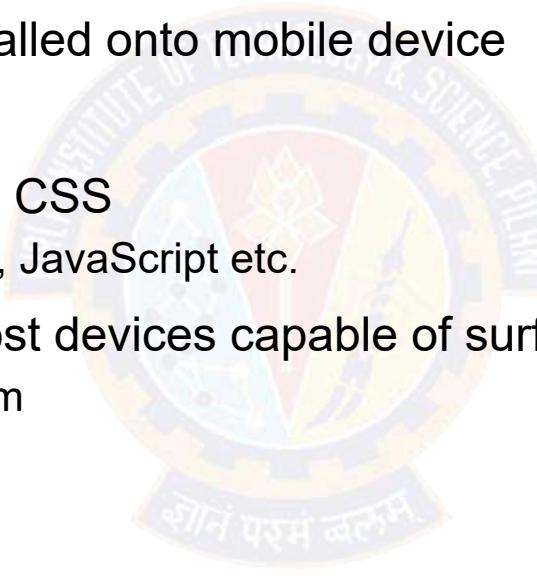
# Native Apps

## 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 download 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



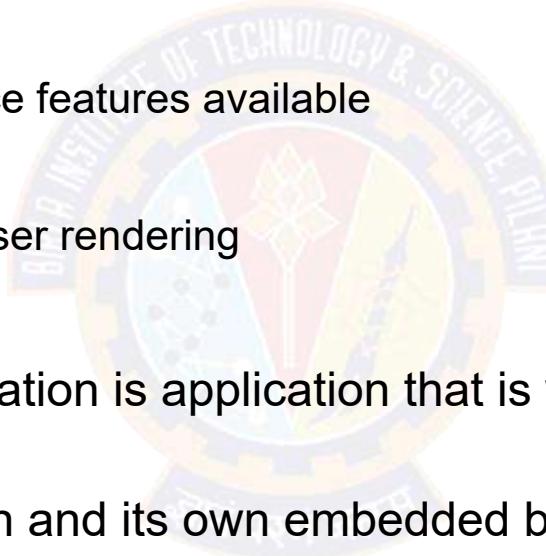
# Web Apps

## 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

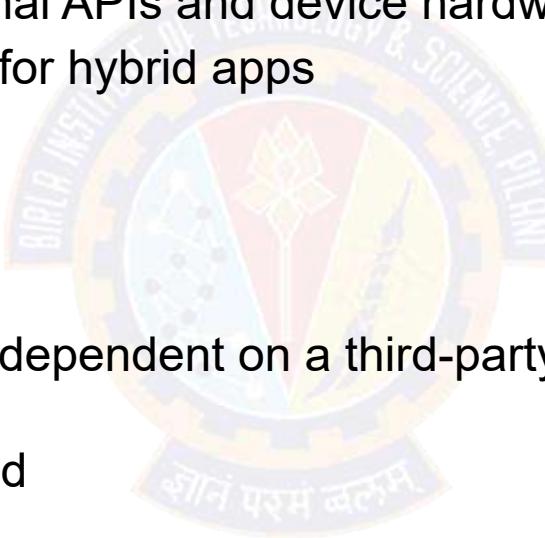
- Part native apps, part web apps
- Like native apps,
  - available in an app store
  - can take advantage of some device features available
- Like web apps,
  - Rely on HTML, CSS , JS for browser rendering
- The heart of a hybrid-mobile application is application that is written with HTML, CSS, and 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](#)

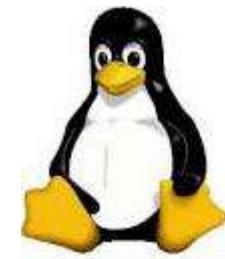
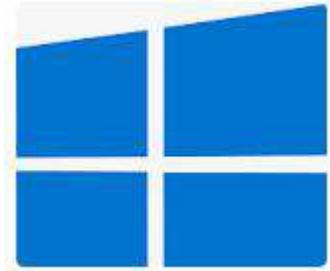
# Computing Platform

- 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:
  - 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

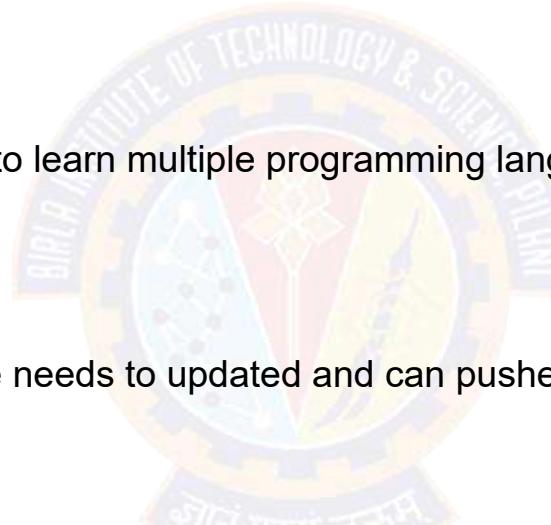
## Using only one code base / framework

- 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

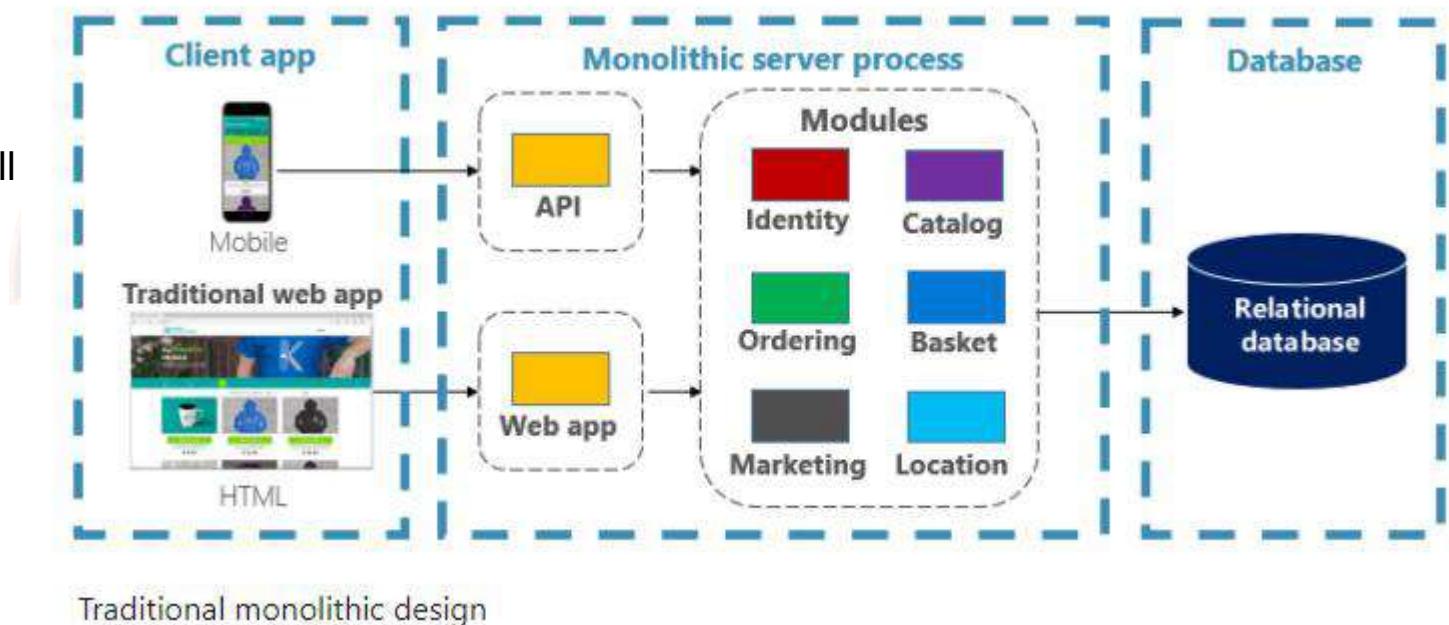
## Disadvantages

- 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



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

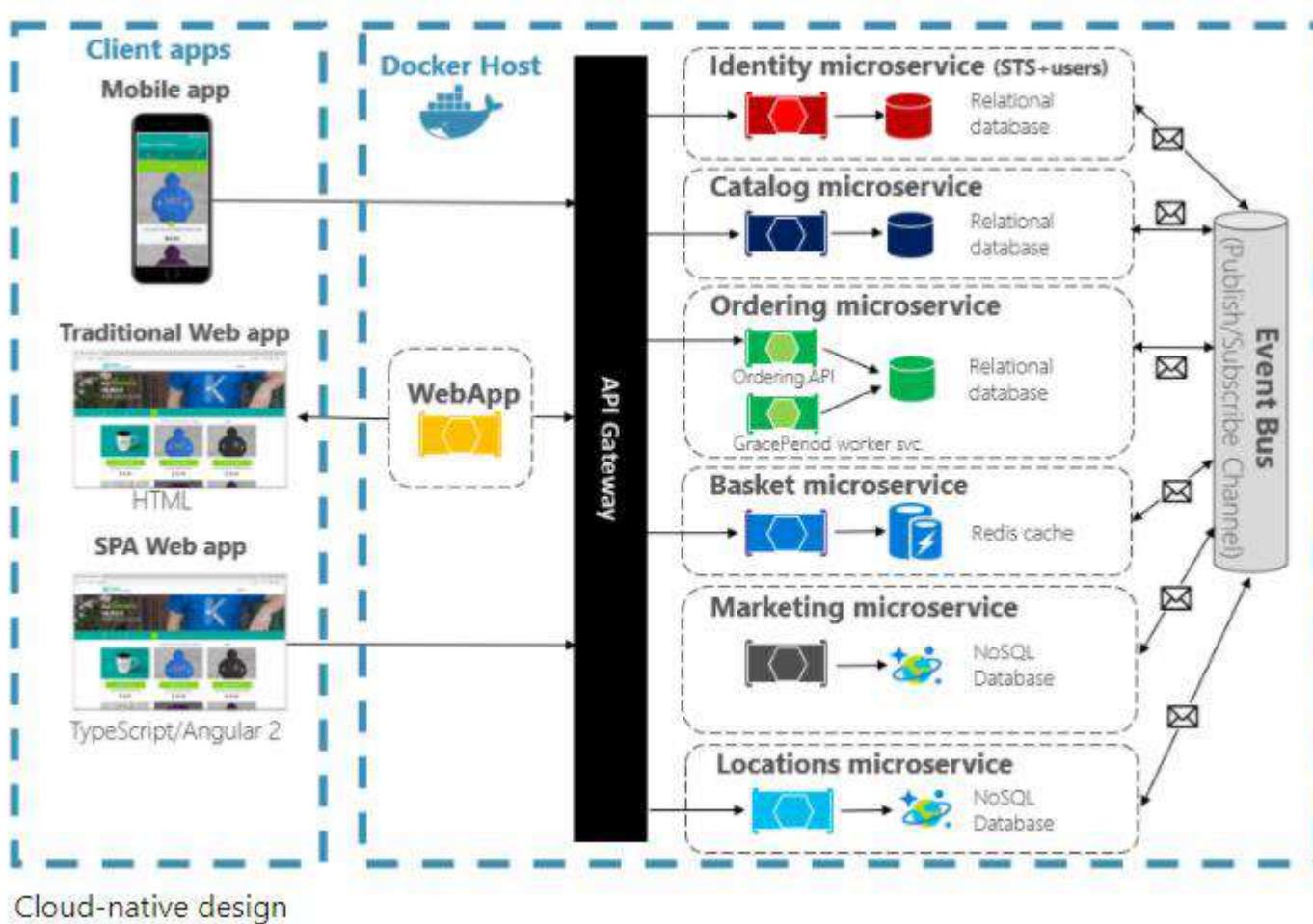


# Monolithic Fear Cycle

- 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

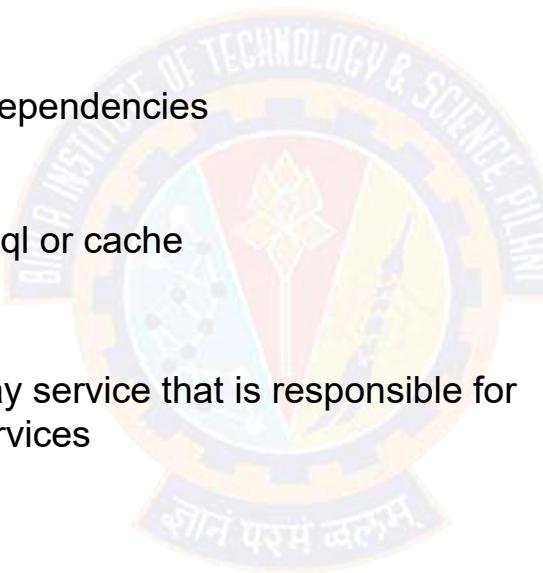


Source : Microsoft

# 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 its 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

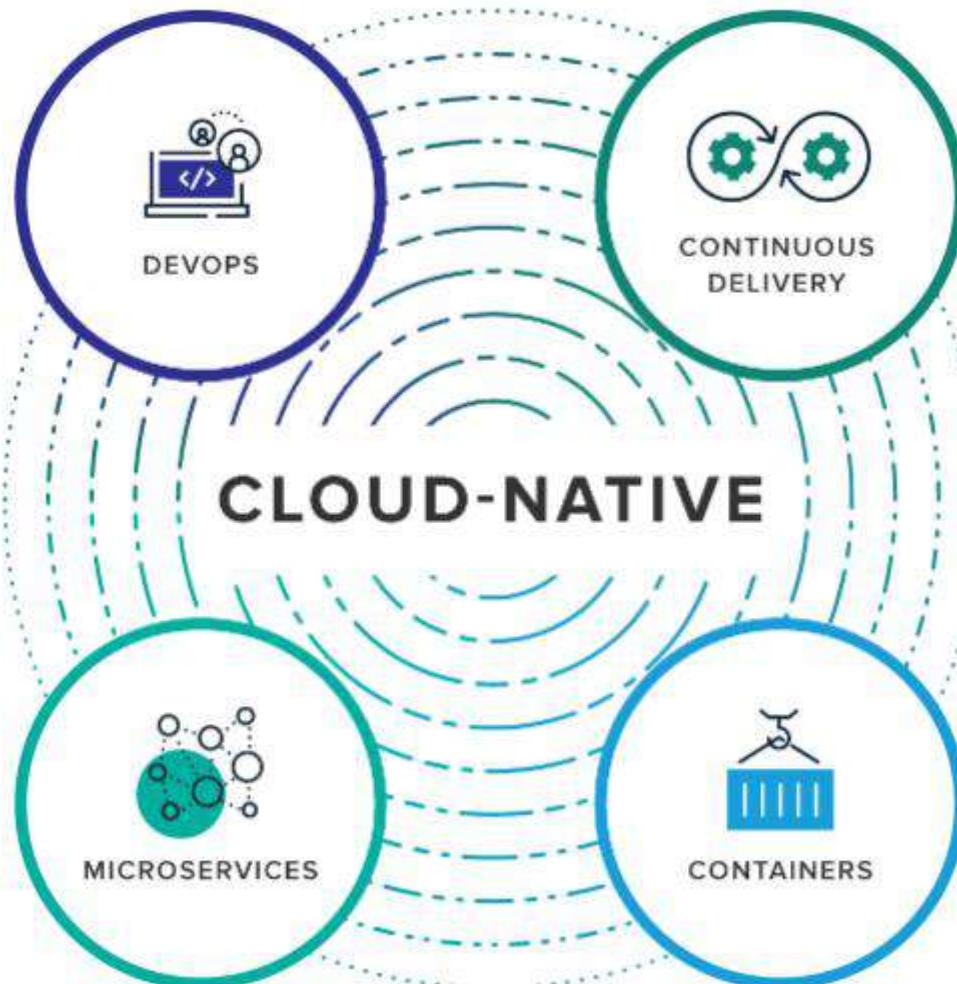


# Cloud native

- 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.*

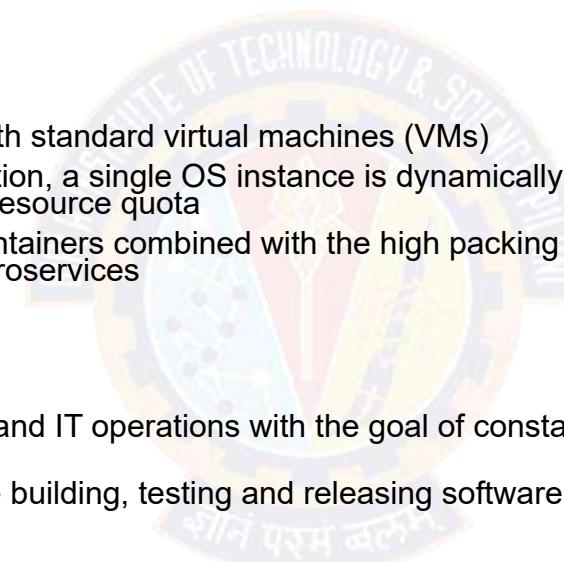
# Cloud native Building Blocks



# 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



# Containers

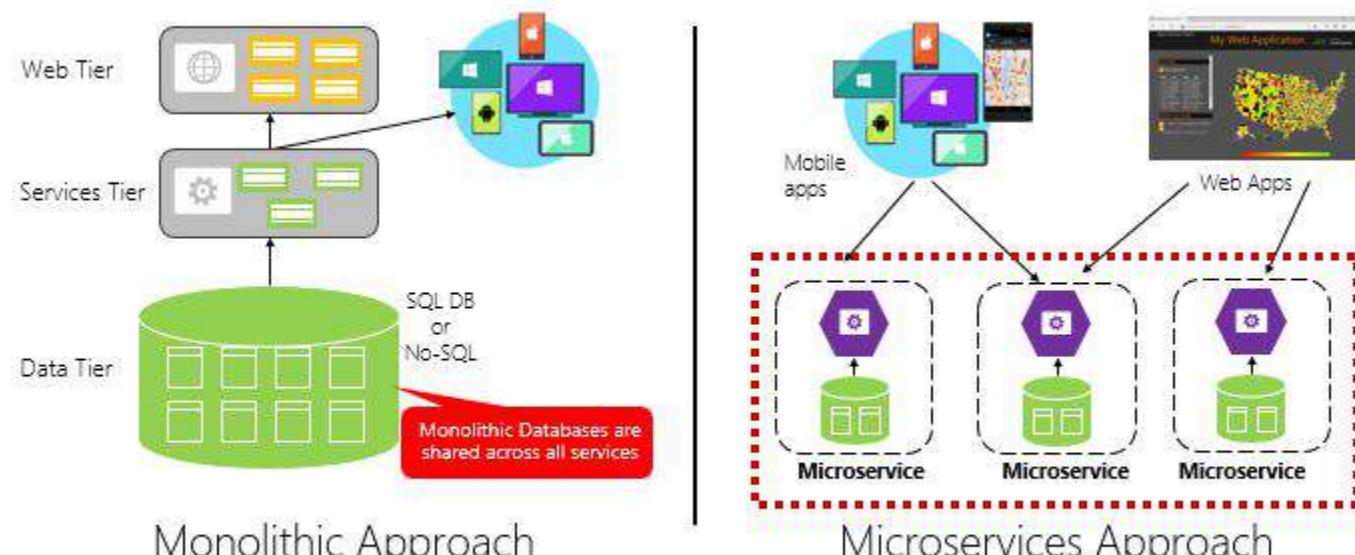
- "Containers are a great enabler of cloud-native software." - Cornelia Davis
- The Cloud Native Computing Foundation places microservice containerization as the first step in their Cloud-Native Trail Map - guidance for enterprises beginning their cloud-native journey.
- [Cloud Native Trail Map](#)



# 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.



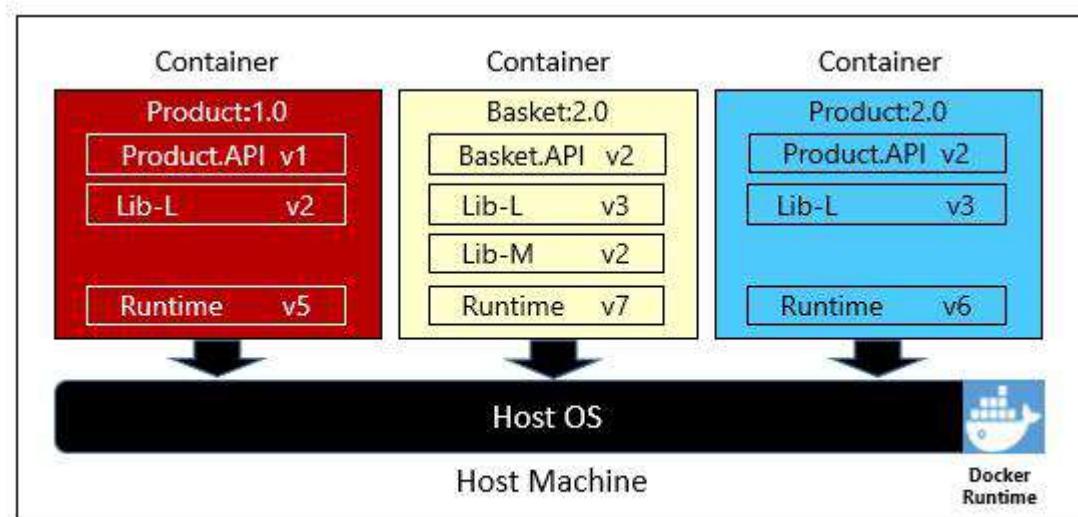
Monolithic deployment versus microservices

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

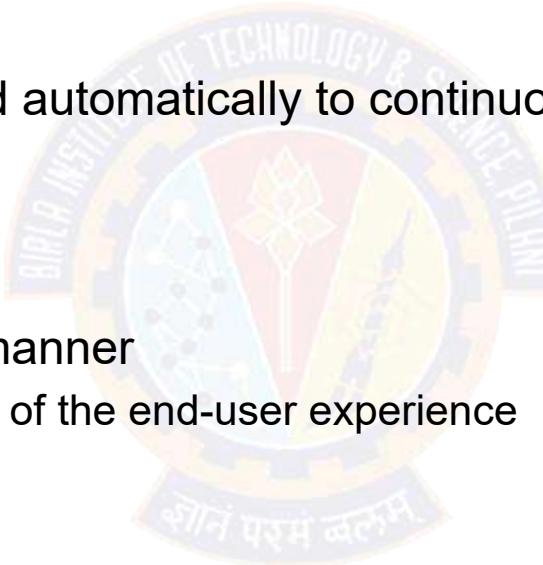


Multiple containers running on a container host

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



# Disadvantages

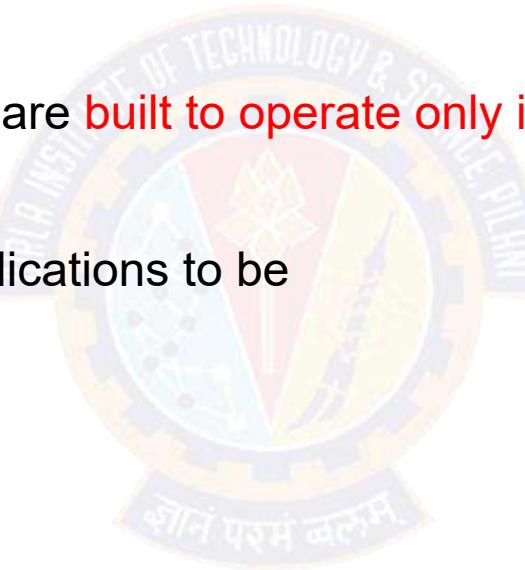
- Create the necessity of managing more elements
  - Rather than one large application, it becomes necessary to manage far more small, discrete services
- Demand additional toolsets
  - to manage the DevOps pipeline, replace traditional monitoring structures, and control microservices architecture
- 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)

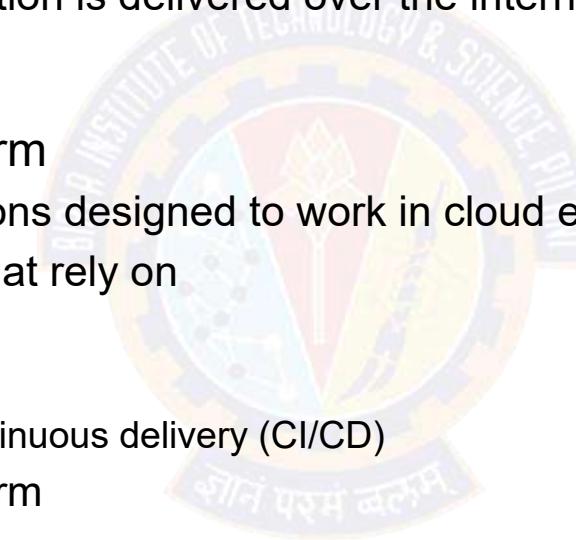
## Cloud native vs. Cloud ready

- 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)

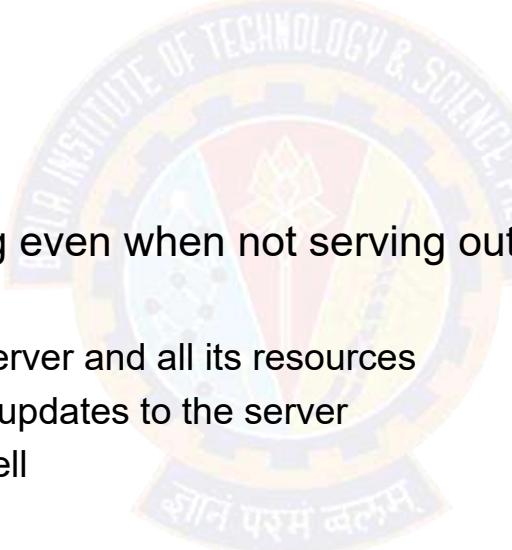
## Cloud native vs. Cloud first

- 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 architected 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

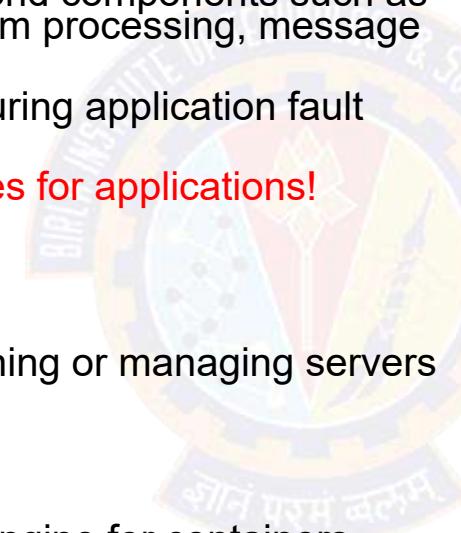
- Benefits
  - Reduced operational cost
  - BaaS: reduced development cost
  - FaaS: scaling costs
    - ❖ occasional requests
    - ❖ inconsistent traffic
  - Easier operational management
  - Reduced packaging and deployment complexity
  - Helps with "Greener" computing



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

## Serverless Offering

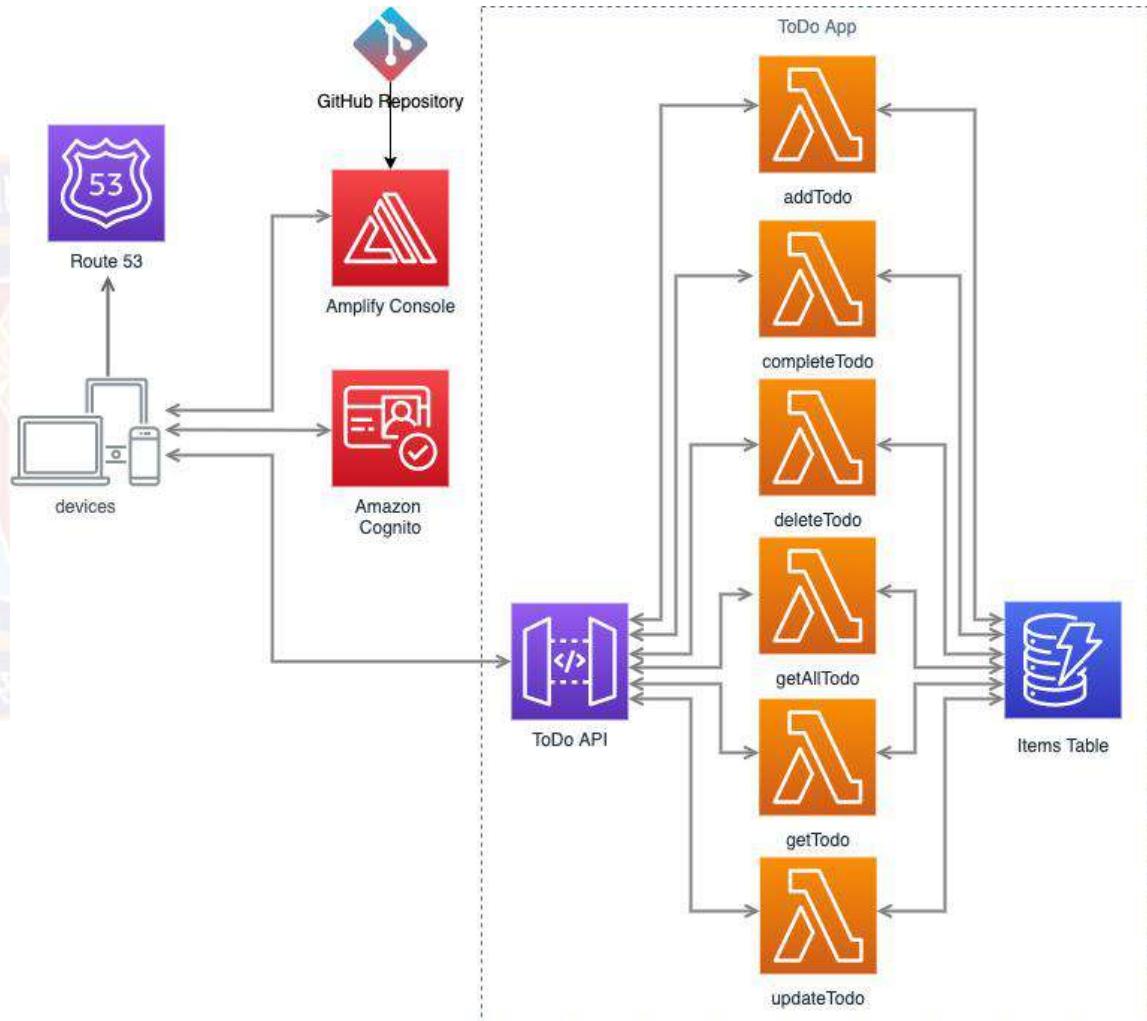
- AWS provides a set of fully managed services that can be used to build and run serverless applications
  - Serverless applications don't require provisioning, maintaining, and administering servers for backend components such as compute, databases, storage, stream processing, message queueing, and 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



# 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

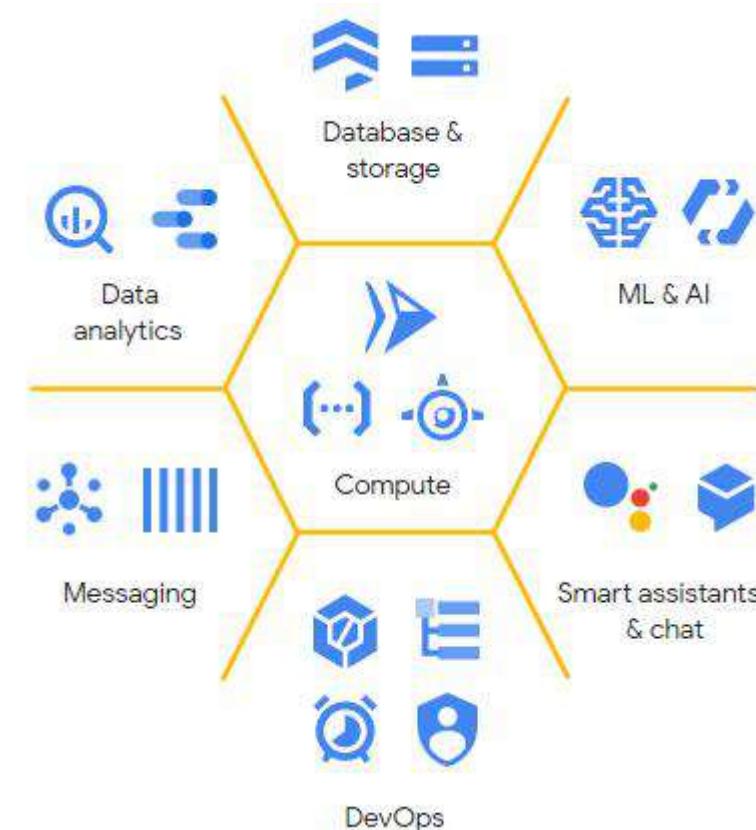


Source : AWS

# 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

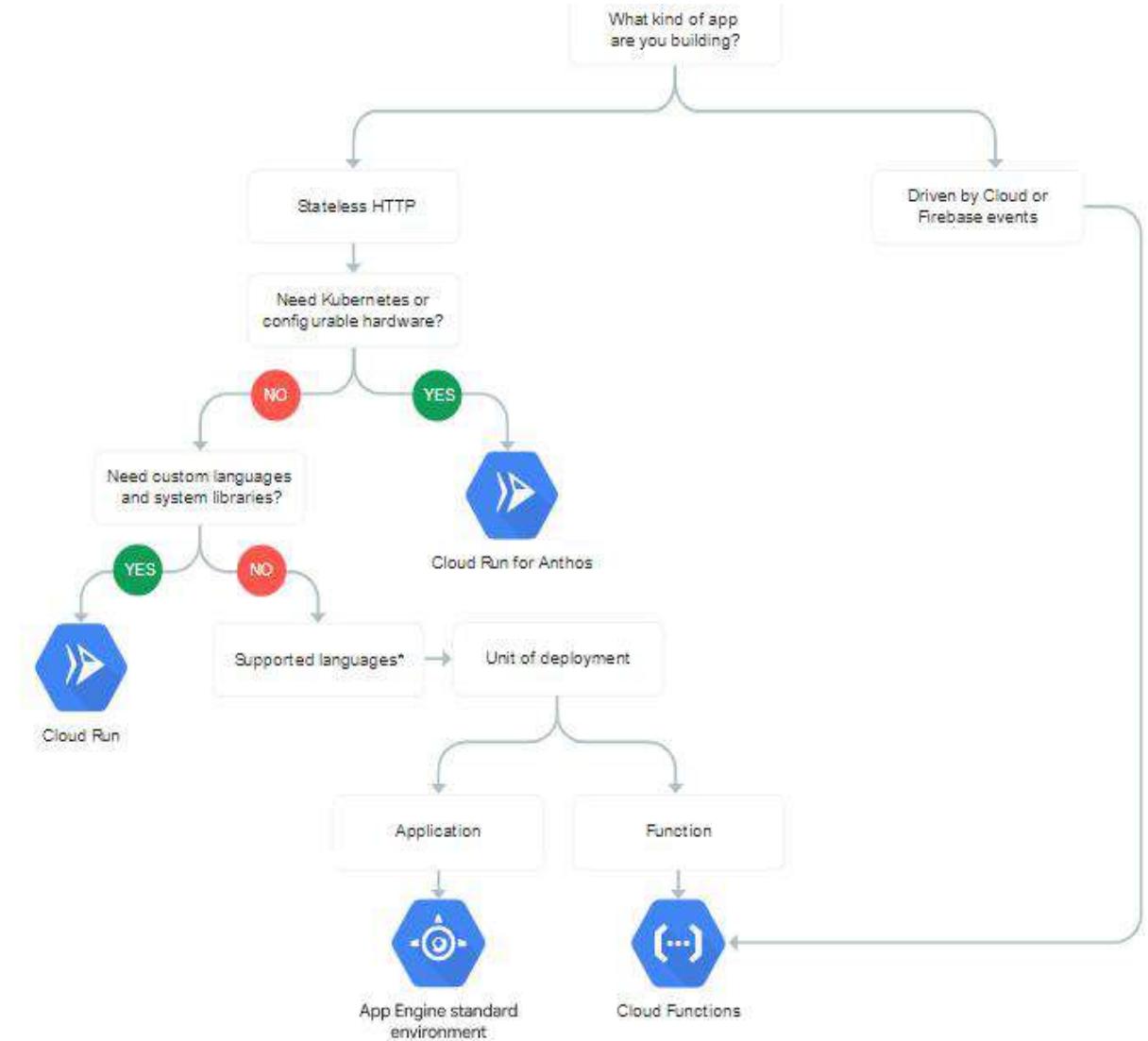
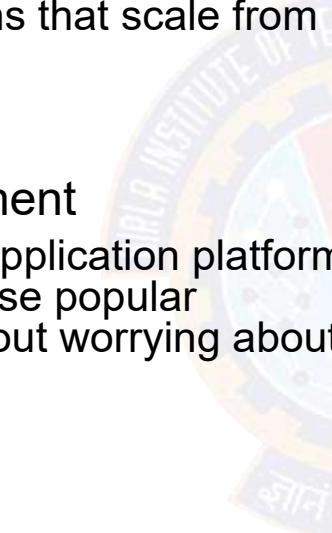


[Source : Google Product Page](#)

# Google Cloud Platform (2)

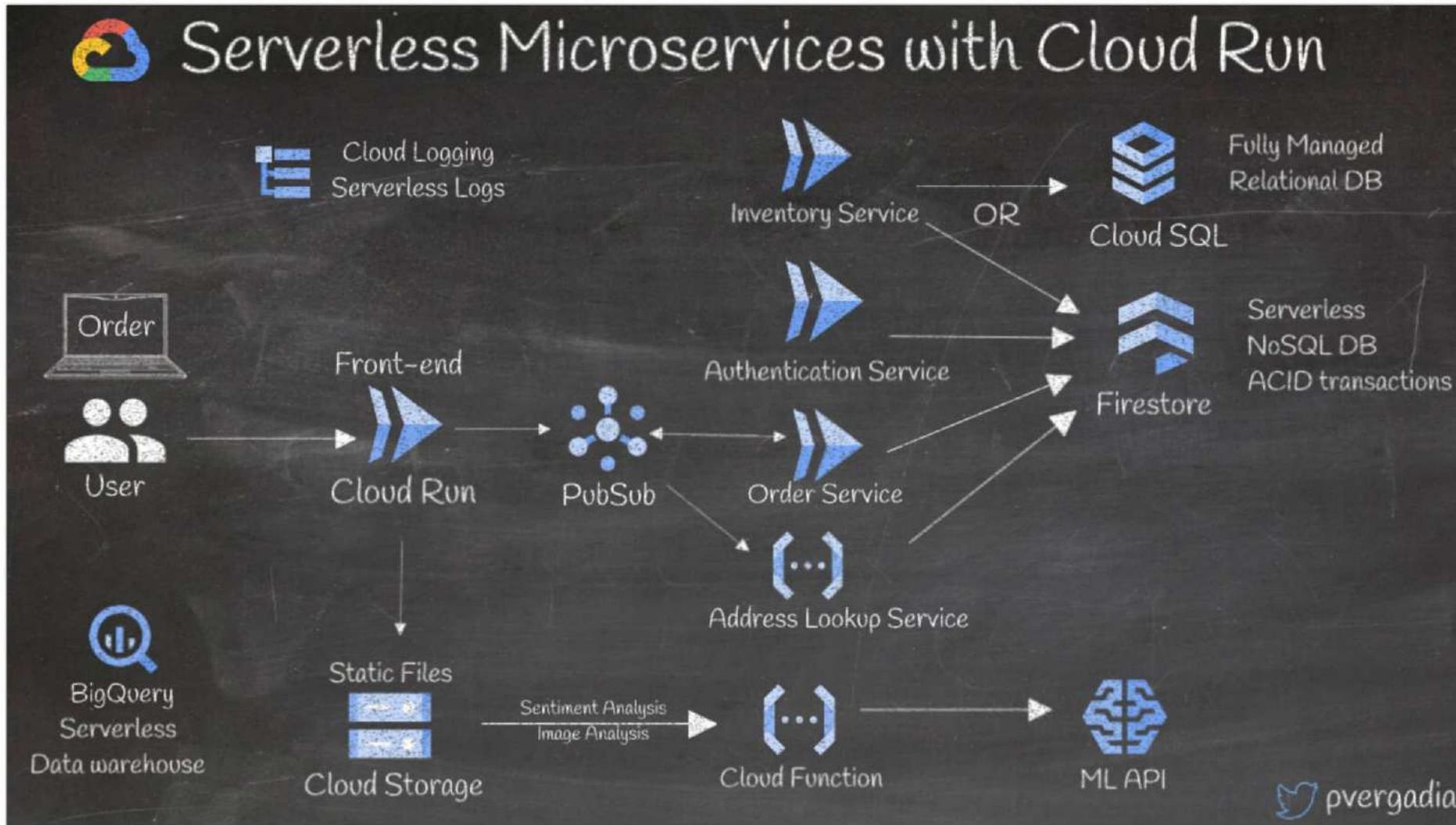
## Services Offered

- 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.



# GCP Serverless Architecture

## Example – Order Processing



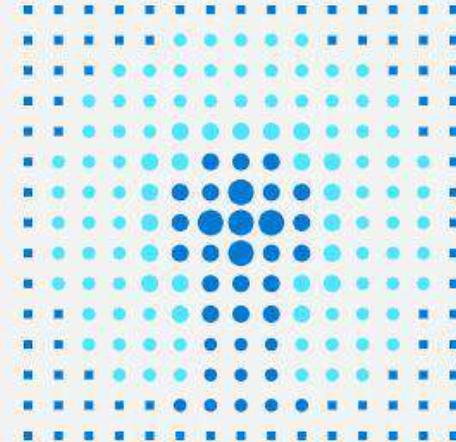
# GCP Serverless Architecture (2)

## 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 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 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 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 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 API gateway

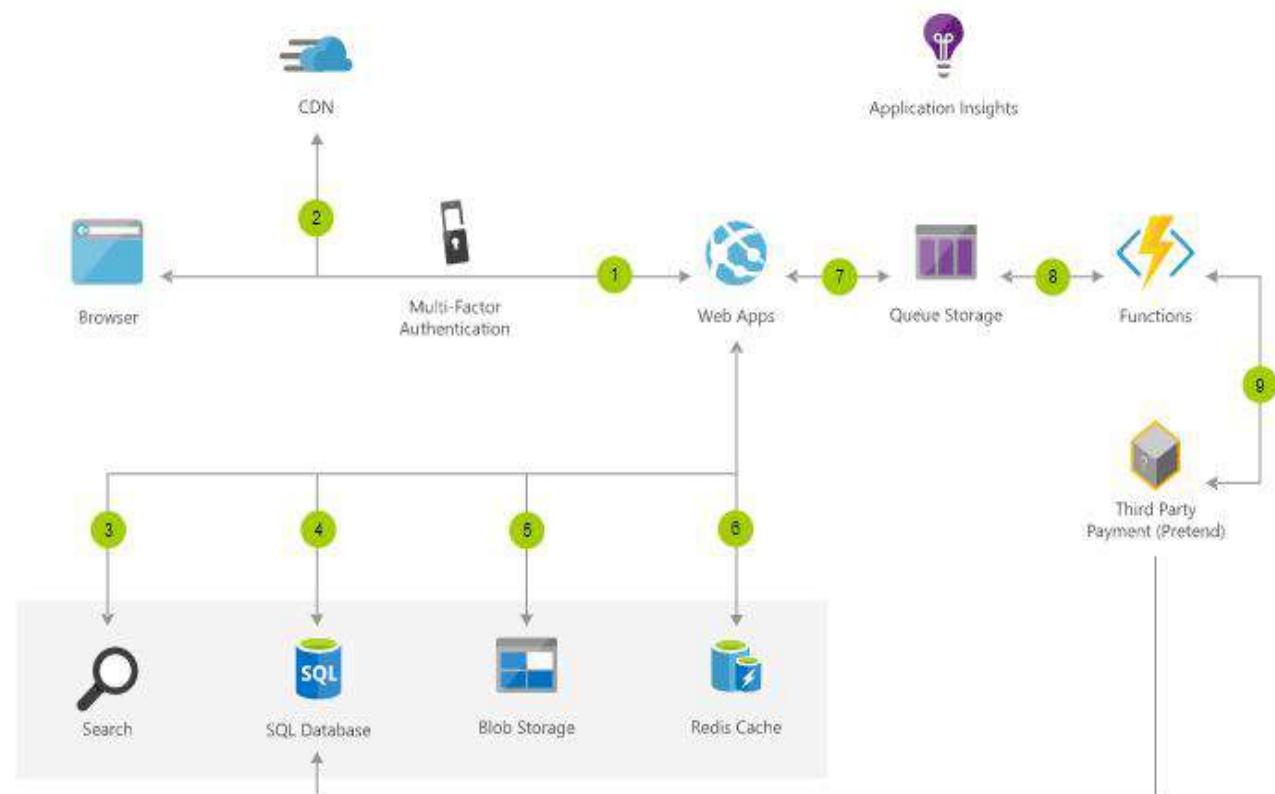
A serverless API gateway is a centralised, fully managed entry point for serverless backend services. It enables developers to publish, manage, secure and analyse APIs at global scale.

[Source : MS Serverless](#)

# Microsoft Azure Serverless Architecture

## Example – Architect scalable e-commerce web app

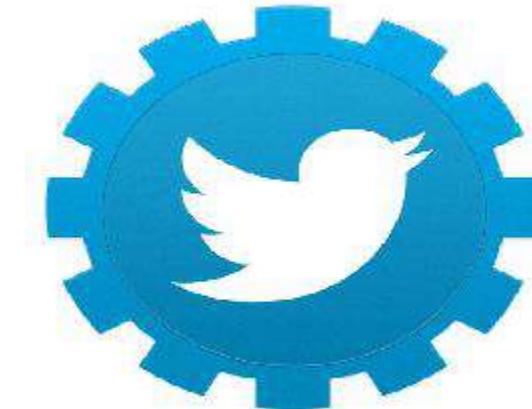
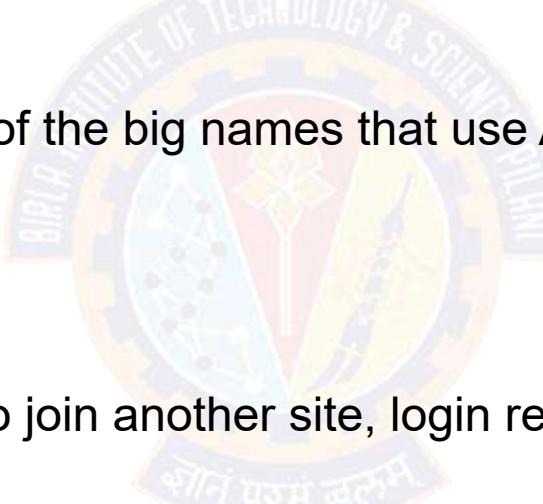
- 1 User accesses the web app in browser and signs in.
- 2 Browser pulls static resources such as images from Azure Content Delivery Network.
- 3 User searches for products and queries SQL database.
- 4 Web site pulls product catalog from database.
- 5 Web app pulls product images from Blob Storage.
- 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.



# 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

The image shows the Google Maps Platform interface. At the top, there's a circular logo with text around the border. Below the logo, the text "Google Maps Platform" is displayed. The interface is divided into three main sections:

- Maps**: Shows a 3D map of a coastal area with a blue route line and a yellow location pin.
- Routes**: Shows a 3D map of a city street with a green route line and a red turn arrow.
- Places**: Shows a 3D building with a purple ribbon banner displaying a 4.6 rating and five yellow stars.

Under each section, there is a brief description:

- Maps**: Build customized, agile experiences that bring the real world to your users with static and dynamic maps, Street View imagery, and 360° views.
- Routes**: Help your users find the best way to get from A to Z with comprehensive data and real-time traffic.
- Places**: Help users discover the world with rich location data for over 200 million places. Enable them to find specific places using phone numbers, addresses, and real-time signals.

# 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



**Connect to our API.**  
**Connect to millions of travellers on Airbnb.**



### Connect and import listings

Quickly import multiple listings to Airbnb and automatically sync data to existing or new listings.



### Manage pricing and availability

Set flexible pricing and reservation rules. Oversee one calendar for multiple listings.



### Message guests seamlessly

Keep response rates high - use existing email flows and automated messages to respond to guests.

# 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

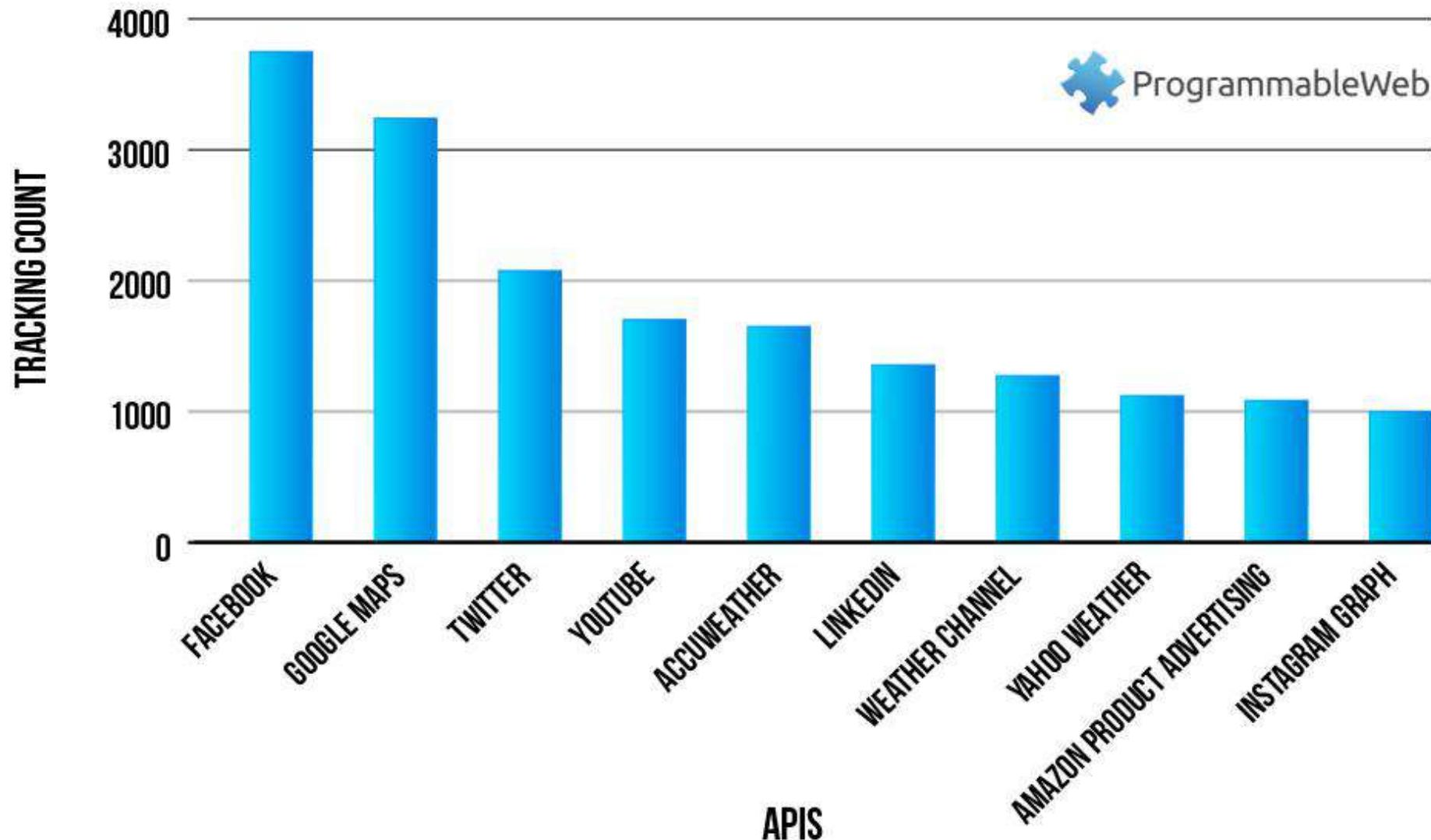
- Marketing APIs are a collection of API endpoints that can be used to help you advertise on social media platform like Facebook, Twitter etc.
- Amazon offers vendors and professional sellers the opportunity to advertise their products on Amazon



Source : pinterest

# Popular APIs

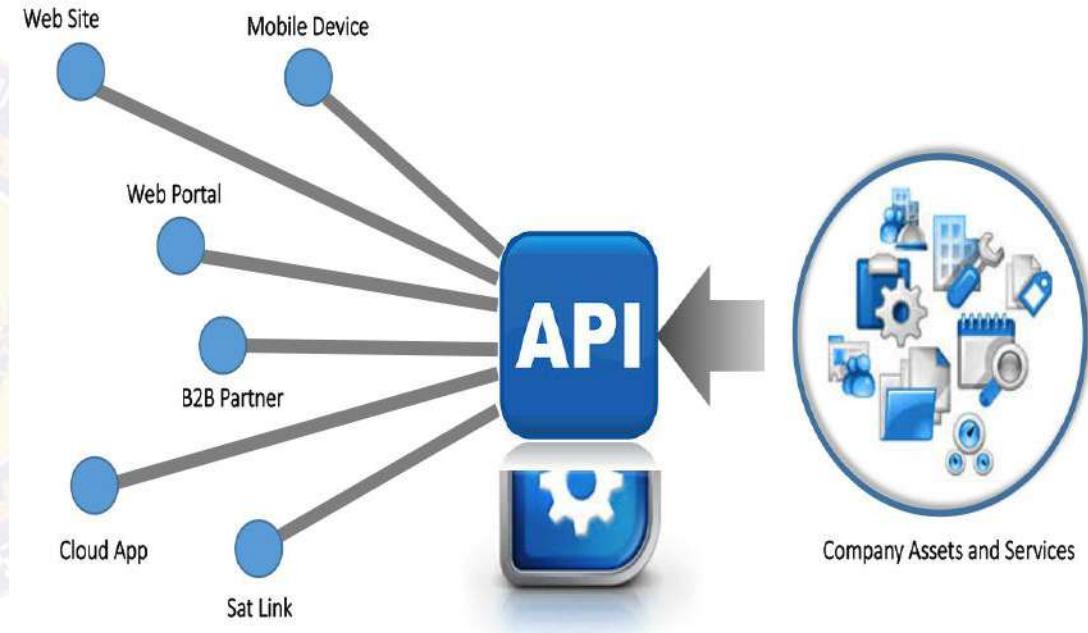
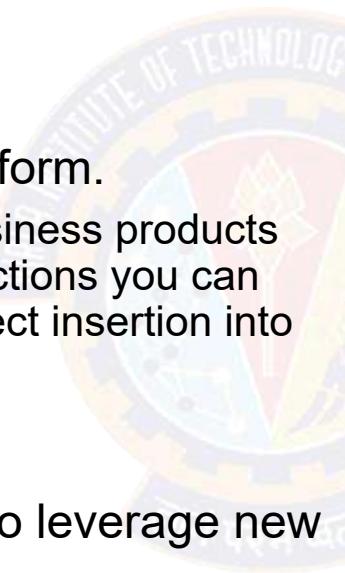
## TOP TRACKED APIs OF ALL TIME



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 consumable 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 act 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

{ REST }

# gRPC

## Backed by Google

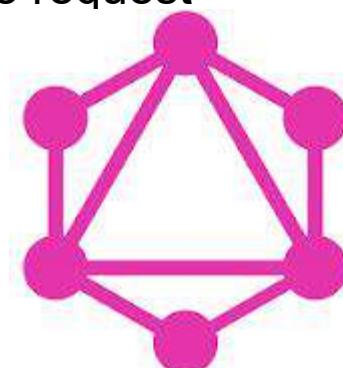
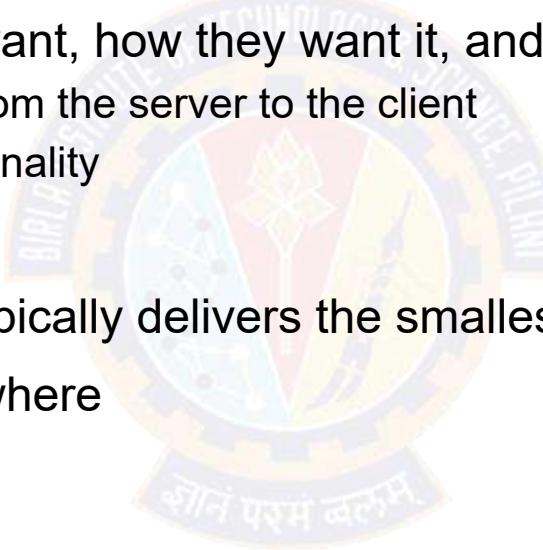


- Actually a new take on an old approach known as RPC, or Remote Procedure Call
- RPC is a method for executing a procedure on a remote server
- RPC functions upon an idea of contracts, in which the negotiation is defined and constricted by the client-server relationship rather than the architecture 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**





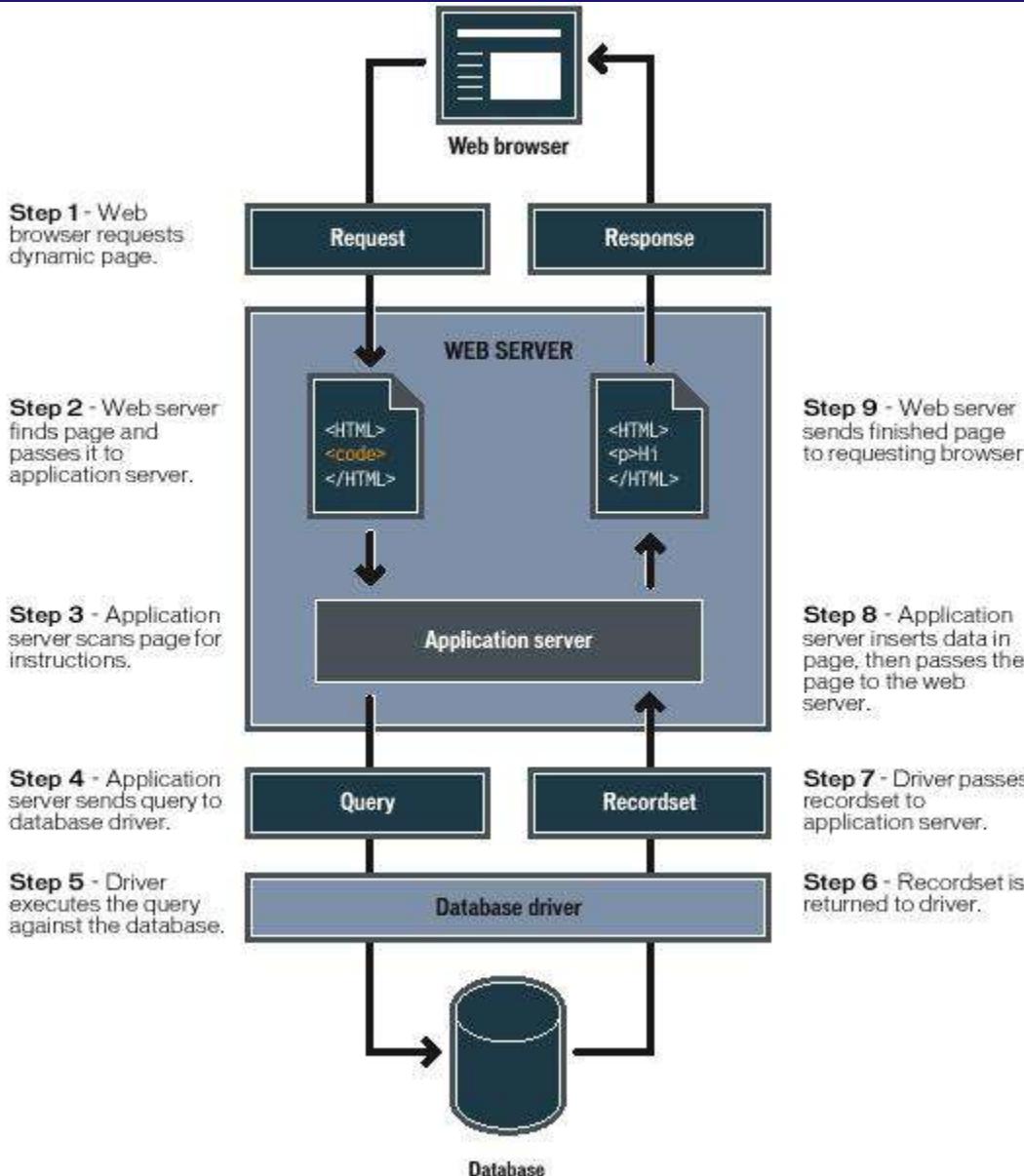
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

**FrontEnds**

**Chandan Ravandur N**

# WebApps

## Working

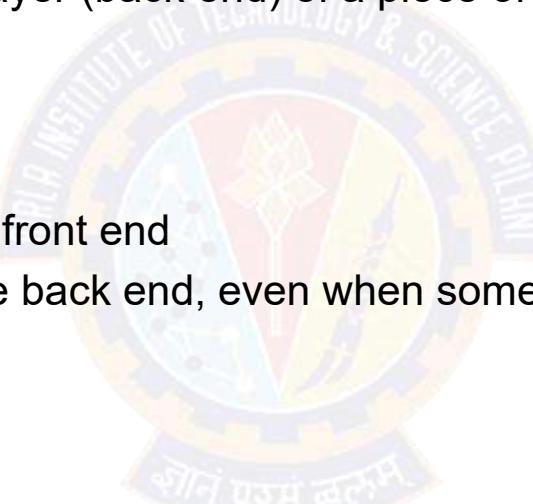


[Image source](#)

# Front end and back end

## the separation of concerns

- In software engineering,
  - Front end refer to the presentation layer (front end),
  - Backend refer to the data access layer (back end) of a piece of software, or the physical infrastructure or hardware
- In the client–server model,
  - the client is usually considered the front end
  - the server is usually considered the back end, even when some presentation work is actually done on the server itself.
- A rule of thumb is that
  - the client-side (or "front end") is any component manipulated by the user
  - the server-side (or "back end") code usually resides on the server, often far remote from user



# Front-end

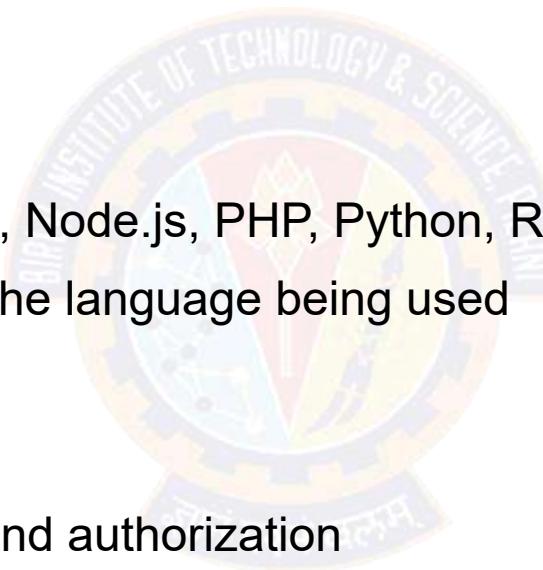
## Focus is on

- User Interface elements
- Mark-up and web languages such as HTML, CSS, JavaScript and supporting libraries
- Asynchronous request handling and AJAX
- Single-page applications (with frameworks like React, AngularJS or Vue.js)
- Web performance
- Responsive web design
- Cross-browser compatibility issues and workarounds
- End-to-end testing with a headless browser
- Build automation to transform and bundle JavaScript files, reduce images size
- Search engine optimization
- Accessibility concerns

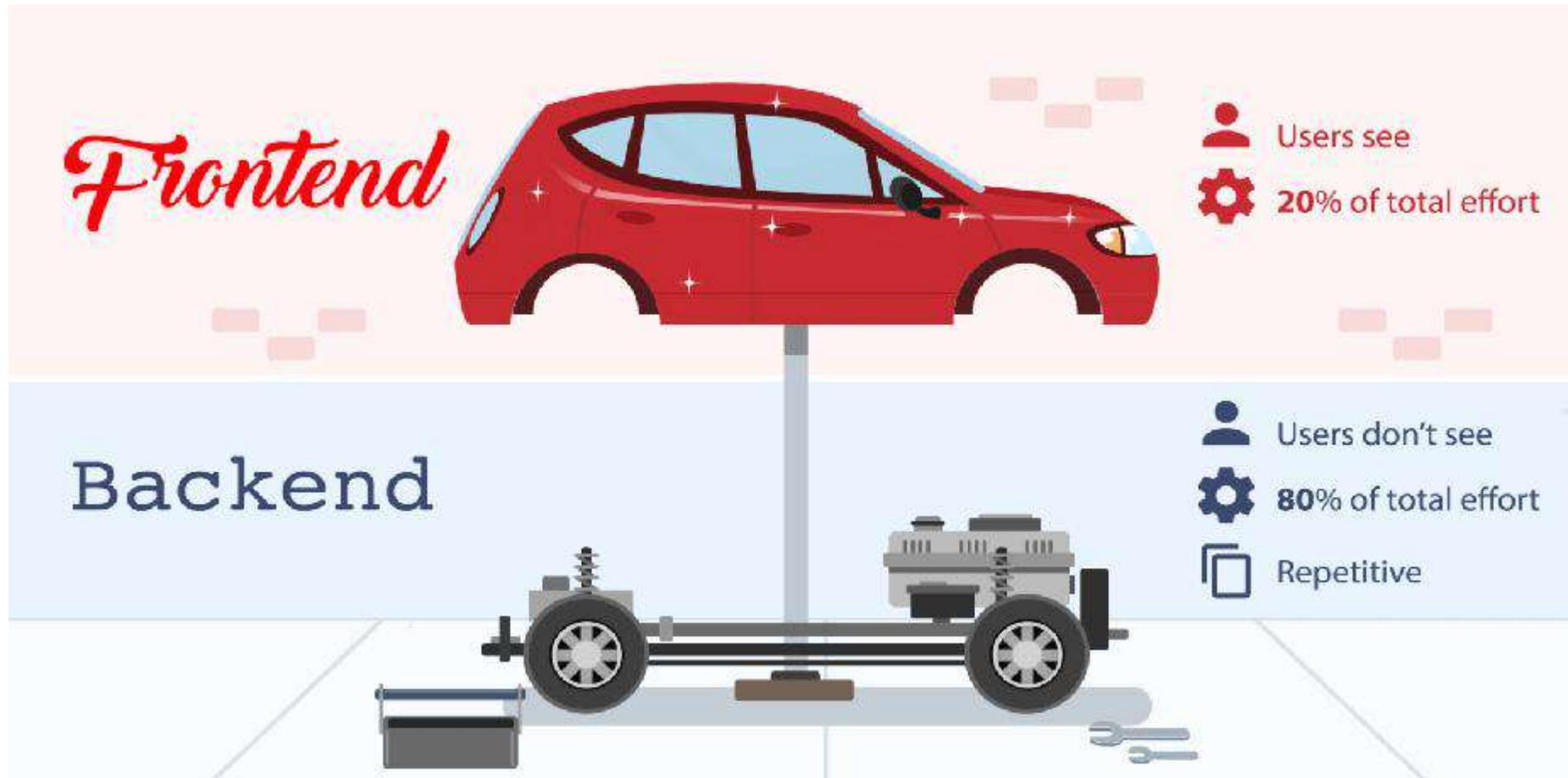
# Back-end

## Focus is on

- Software Architecture
- Application Business Logic
- Application Data Access
- Database management
- Scripting languages like JavaScript, Node.js, PHP, Python, Ruby, Java etc.
- Automated testing frameworks for the language being used
- Scalability
- High availability
- Security concerns, authentication and authorization



# Front end – Back end



[Image Source : Back4App](#)

# Frontend

## Elements

- Not just about the beautification of the web / mobile interfaces
- Involves elements for
  - Content Structure
  - Styling
  - Interaction
- Deals with
  - Rendering of the content on the browsers / within apps
  - Beautification of content rendered
  - Interaction carried out by user on web pages / mobile apps
- Building blocks – **HTML + CSS + JS!**



# Content Structure

## Markup

- Structure of the page is
  - the foundation of websites
  - essential for search engine optimization
  - vital to provide the style and the interaction that the reader will ultimately use
- Hyper Text Meta Language (HTML)
  - will be at the very center of it all regardless of how complex the site is
  - uses tags, as opposed to a programming language, in order to identify all the various types of content out there on every single webpage
  - For example, in a newspaper article, a header, sub header, text body and other things are present
  - HTML works in the exact same way to label all the stuff on the webpage, except it uses HTML tags
  - even a JavaScript webpage, is comprised of HTML tags corresponding to each element on the webpage and every single content type is bundled into HTML tags as well.

# Styling

## Cascading Style Sheets

- A core functionality of front-end development
- lays out the page and give it both its unique visual flair and a clear, user-friendly view to allow readers
- An important aspect of styling is checking across several browsers and to write concise, terse code that is
  - specific yet generic at the same time
  - displays well in as many renderers as possible
- CSS determines how all the HTML elements must appear on the frontend
  - HTML gives all of the raw tools necessary to structure webpage
  - CSS allows to style everything so it will appear to the user exactly the way you would like it to

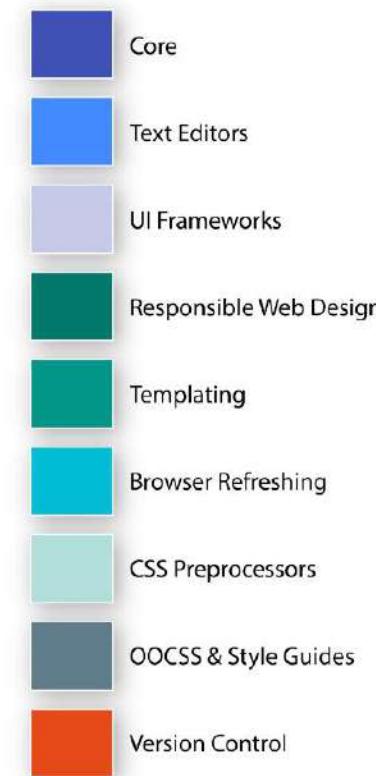
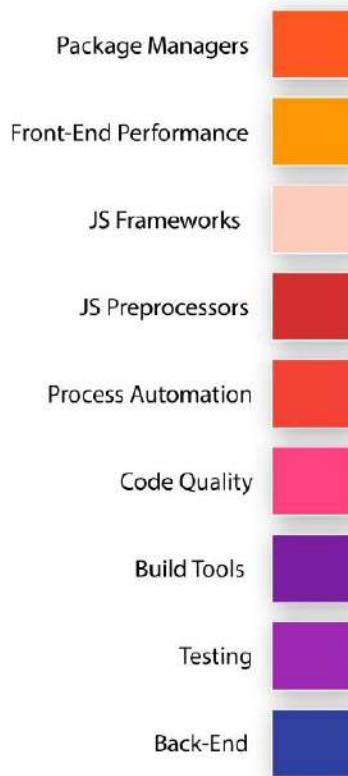
# Interaction

## User Interaction and interaction with backend

- JavaScript
- User Interaction on page
  - Supported by all modern browsers
  - employed by pretty much every website in order to gain increased functionality and power
  - Used mainly for website content adjustment and to make the website itself act certain ways depending on the user's actions
  - Used for creating call-to-action buttons, confirmation boxes and adding new details to current information
- Dynamic Behavior
  - drastically improves a browser's default actions and controls
  - helps in placing asynchronous requests to server side and render the response received

# Frontend Tools Spectrum

## THE FRONT-END SPECTRUM



Source : Medium



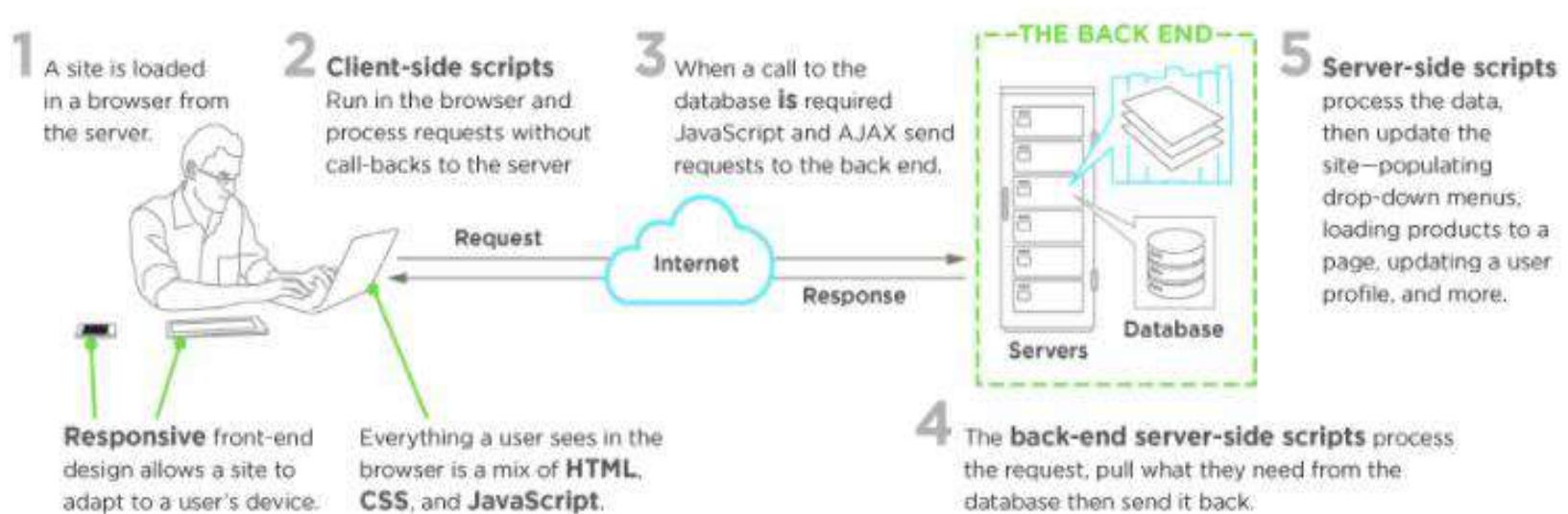
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# FrontEnd - Development

**Chandan Ravandur N**

# Typical Development scenario

- A front-end developer
  - architects and develops websites and applications using web technologies (i.e., HTML, CSS, DOM, and JavaScript)
  - which run on the web platform or act as compilation input for non-web platform environments (i.e., NativeScript)



[Image Source : upwork](#)

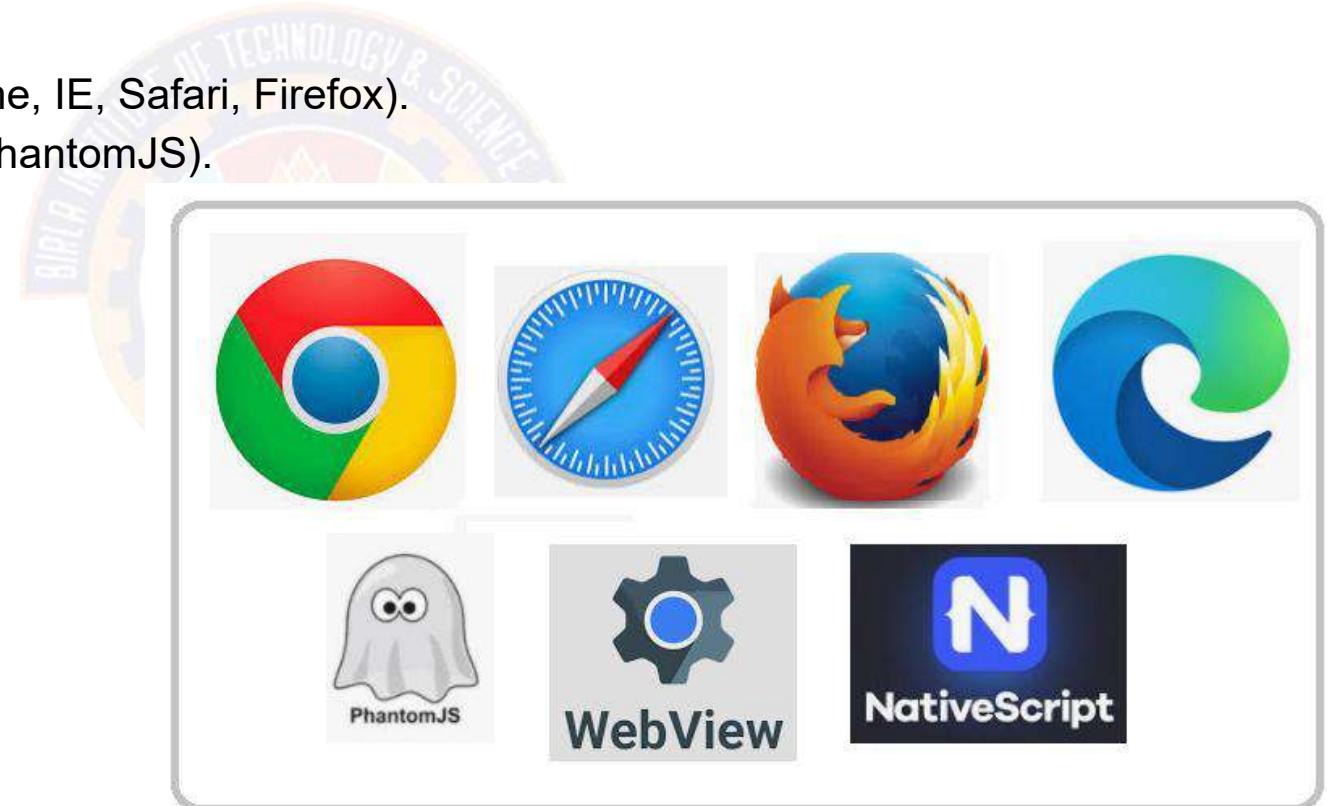
# Role of a front end developer

- A front-end developer crafts HTML, CSS, and JS that typically runs on the web platform (e.g. a web browser) delivered from one of the following operating systems (aka OSs):
  - Android
  - Chromium
  - iOS
  - Ubuntu (or some flavor of Linux)
  - Windows
- These operating systems typically run on one or more of the devices:
  - Desktop computer
  - Laptop / netbook computer
  - Mobile phone
  - Tablet
  - TV
  - Watch
  - Things (i.e., anything you can imagine, car, refrigerator, lights, thermostat, etc.)



# Web Platform

- Front-end technologies can run on the aforementioned operating systems and devices using the following run time web platform scenarios:
  - A web browser (examples: Chrome, IE, Safari, Firefox).
  - A headless browser (examples: phantomJS).
  - A WebView/browser tab
  - A native application



# Web Platform

## Web Browsers

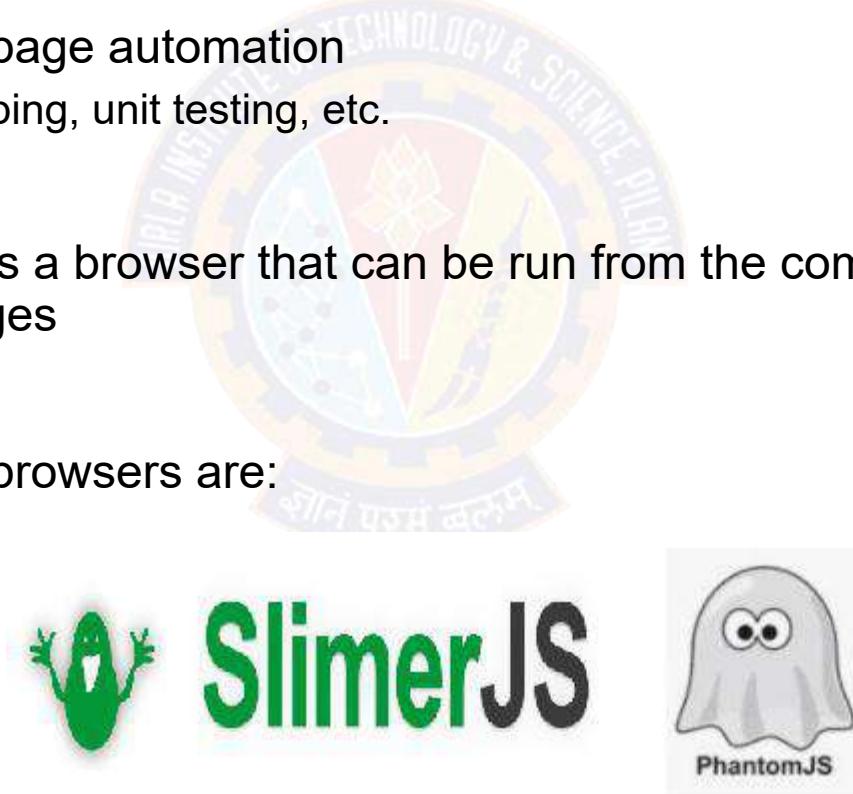
- Is software used to retrieve, present, and traverse information on the WWW
- Typically run on a desktop or laptop computer, tablet, or phone,
  - but as of late a browser can be found on just about anything (i.e., on a fridge, in cars, etc.).
- The most common web browsers are (shown in order of most used first):
  - Chrome
  - Internet Explorer
  - Firefox
  - Safari



# Web Platform

## Headless Browsers

- Are a web browser without a graphical user interface
- that can be controlled from a command line interface programmatically
- Used for the purpose of web page automation
  - e.g., functional testing, scraping, unit testing, etc.
- Think of headless browsers as a browser that can be run from the command line that can retrieve and traverse web pages
- The most common headless browsers are:
  - PhantomJS
  - slimerjs
  - trifleJS



# Web Platform

## Webviews

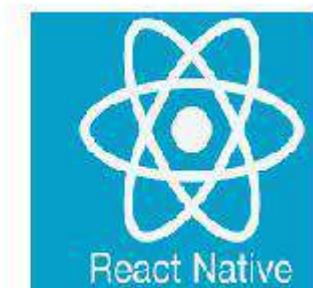
- Are used by a native OS, in a native application, to run web pages
- Think of a webview like an iframe or a single tab from a web browser that is embedded in a native application running on a device
  - e.g., webviews iOS, android, windows
- The most common solutions for webview development are:
  - Cordova (typically for native phone/tablet apps)
  - NW.js (typically used for desktop apps)
  - Electron (typically used for desktop apps)



# Web Platform

## Native from Web Tech

- Web browser development can be used by front-end developers to craft code for environments that are not fueled by a browser engine
- Development environments are being dreamed up that use web technologies (e.g., CSS and JavaScript), without web engines, to create native applications
- Some examples of these environments are:
  - NativeScript
  - React Native





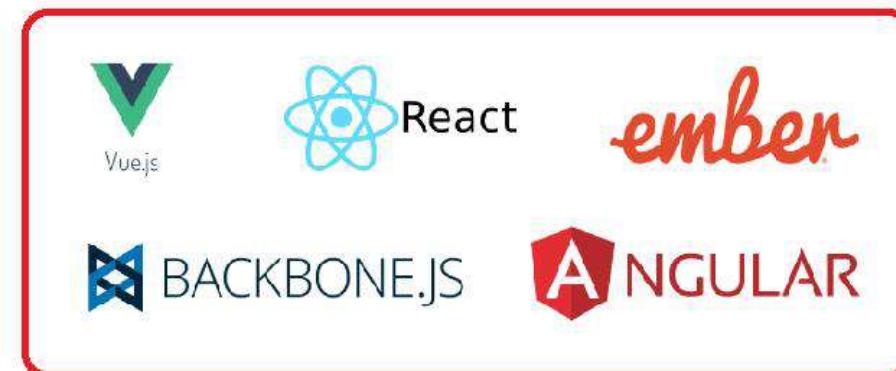
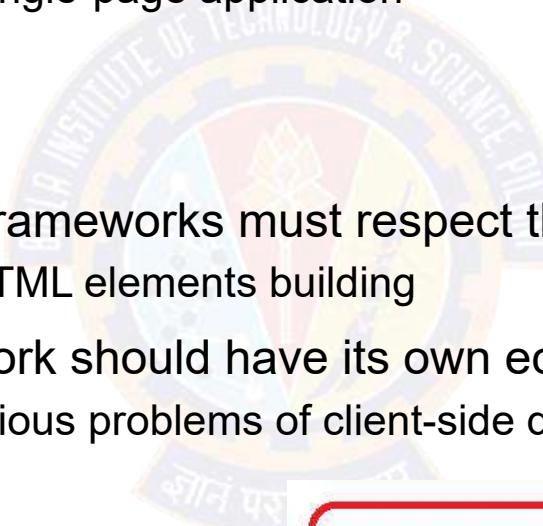
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# FrontEnd - Frameworks

Chandan Ravandur N

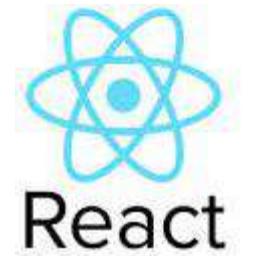
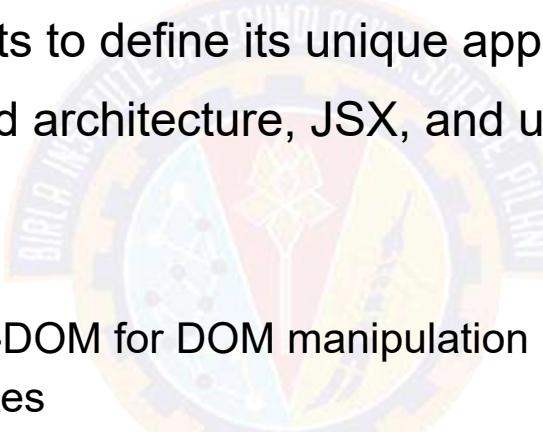
# Why frameworks?

- For frontend developers, it's increasingly challenging to make up their minds about which JavaScript application framework to choose
  - especially when need to build a single-page application
- Requirements
- First, modern frontend JavaScript frameworks must respect the Web Components specification.
  - should have support for custom HTML elements building
- Second, a solid JavaScript framework should have its own ecosystem
  - Ready solutions aim at solving various problems of client-side development such as
    - ❖ Routing
    - ❖ managing app state
    - ❖ communicating with the backend



# React

- Stormed the JS world several years ago to become its definite leader
- Encourages to use a reactive approach and a functional programming paradigm
- Introduced many of its own concepts to define its unique approach to frontend web development
- Need to master a component-based architecture, JSX, and unidirectional data flow
- Ecosystem:
  - The React library itself plus React-DOM for DOM manipulation
  - React-router for implementing routes
  - JSX, a special markup language that mixes HTML into JavaScript
  - React Create App, a command line interface that allows you to rapidly set up a React project
  - Axios and Redux-based libraries, JS libraries that let you organize communication with the backend.
  - React Developer Tools for Chrome and Firefox browsers.
  - React Native for development of native mobile applications for iOS and Android.



# Angular 2

- Marks a turning point in the history of the Angular framework
  - Has substantially changed its architecture to come to terms with React
  - Is from a Model-View-Whatever architecture to a component-based architecture
- 
- Ecosystem:
    - A series of modules that can be selectively installed for Angular projects: `@angular/common`, `@angular/compiler`, `@angular/core`, `@angular/forms`, `@angular/http`, `@angular/platform-browser`, `@angular/platform-browser-dynamic`, `@angular/router`, and `@angular/upgrade`
    - TypeScript and CoffeeScript, supersets of JavaScript that can be used with Angular
    - Angular command line interface for quick project setup
    - Zone.js, a JS library used to implement zones, otherwise called execution context, in Angular apps
    - RxJS and the Observable pattern for asynchronous communication with server-side apps
    - Angular Augury, a special Chrome extension used for debugging Angular apps
    - Angular Universal, a project aimed at creating server-side apps with Angular
    - NativeScript, a mobile framework for developing native mobile applications for iOS and Android platforms



# Vue

- At first sight, it may look like the Vue library is just a mix of Angular and React
- Borrowed concepts from Angular and React
- For example,
  - Vue wants you to store component logic and layouts along with stylesheets in one file. That's how React works without stylesheets
  - To let Vue components talk to each other, Vue uses the props and state objects - existed in React before Vue adopted it
  - Similarly to Angular, Vue wants you to mix HTML layouts with JavaScript
- The VueJS ecosystem consists of:
  - Vue as a view library.
  - Vue-loader for components.
  - Vuex, a dedicated library for managing application state with Vue; Vuex is close in concept to Flux.
  - Vue.js devtools for Chrome and Firefox.
  - Axios and vue-resource for communication between Vue and the backend.
  - Nuxt.js, a project tailored to creating server-side applications with Vue; Nuxt.js is basically a competitor to Angular Universal.
  - Weex, a JS library that supports Vue syntax and is used for mobile development.



# Ember

- Like Backbone and AngularJS, is an “ancient” JavaScript framework
- But the fact that Ember is comparatively old doesn’t mean that it’s out of date
- Allows implement component-based applications just like Angular, React, and Vue do
- One of the most difficult JavaScript frameworks for frontend web development
- Realizes a typical MVC JavaScript framework, and Ember’s architecture comprises the following parts:
  - adapters, components, controllers, helpers, models, routes, services, templates, utils, and addons.
- The EmberJS ecosystem includes:
  - The actual Ember.js library and Ember Data, a data management library.
  - Ember server for development environments, built into the framework.
  - Handlebars, a template engine designed specifically for Ember applications.
  - QUnit, a testing JavaScript framework used with Ember.
  - Ember CLI, a highly advanced command line interface tool for quick prototyping and managing Ember dependencies.
  - Ember Inspector, a development tool for Chrome and Firefox.
  - Ember Observer, public storage where various addons are stored. Ember addons are used for Ember apps to implement generic functionalities.



# Backbone/Marionette

- Is an MV\* framework. Backbone partly implements an MVC architecture, as Backbone's View part carries out the responsibilities of the Controller
- Has only one strong dependency – the Underscore library that gives us many helper functions for convenient cross-browser work with JavaScript
- Attempts to reduce complexity to avoid performance issues
- The BackboneJS ecosystem contains:
  - The Backbone library, which consists of events, models, collections, views, and router
  - Underscore.js, a JavaScript library with several dozen helper functions that you can use to write cross-browser JavaScript
  - Underscore's microtemplating engine for Backbone templates
  - BackPlug, an online repository with a lot of ready solutions (Backbone plugins) tailored for Backbone-based apps
  - Backbone generator, a CLI for creating Backbone apps
  - Marionette, Thorax, and Chaplin – JS libraries that allow you to develop a more advanced architecture for Backbone apps



# Compared

Name	Type	Shadow DOM EcmaScript 6+	Relative Popularity	Difficulty of Learning
React	Library	Supported	*****	*****
Angular	Framework	Supported	***	*****
Ember	Framework	Supported	*	*****
Vue	Library	Supported	**	***
Backbone	Framework	Supported	*	***

Source : [RubyGarage](#)

Reference : RubyGarage Blog



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

**Backend**

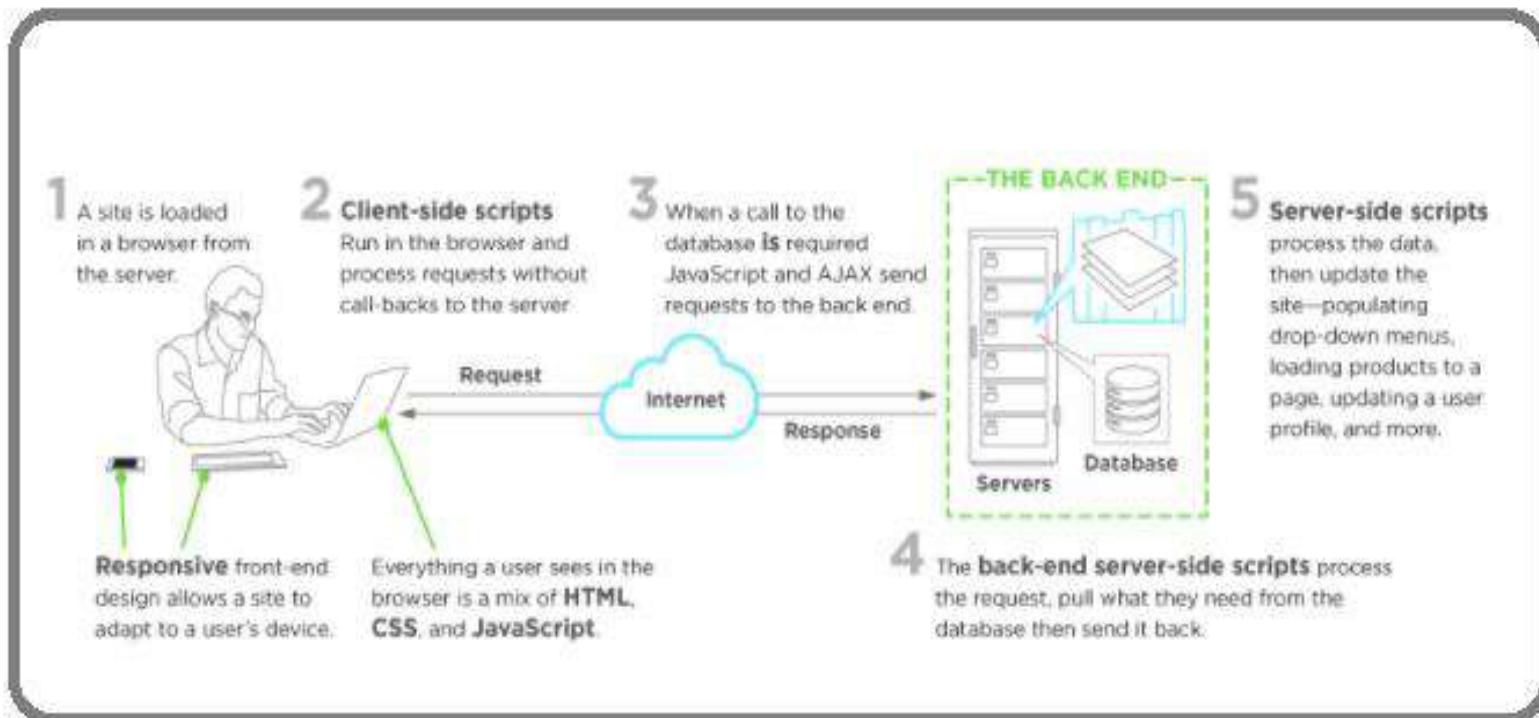
**Chandndan Ravandu N**

---

# Server side

## Interaction between front-end and back-end

- To understand server side, one need to know the front end and how the two interact



[Image : Upwork](#)

# Server side

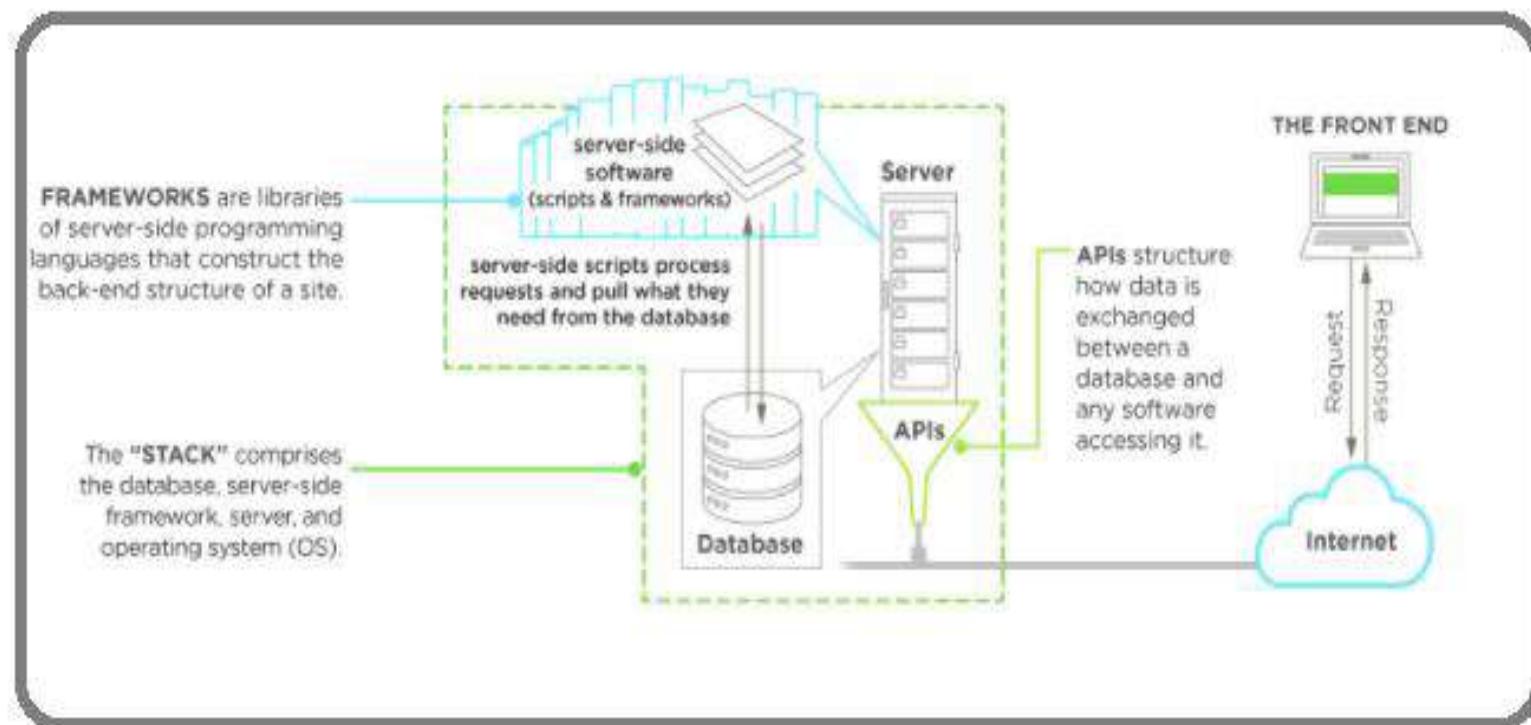
## Interaction Explained

- The front end is what happens in the browser—everything the end users see and interact with
- The back end, on the other hand, happens on the server
  - on site, or in the cloud
  - databases
- Machinery that works behind the scenes—everything the end user doesn't see or directly interact with, but **that powers what's happening!**
- Server-side manages all those requests that come from users' clicks
  - Front-end scripts sends those requests over to the server side to be processed, returning the appropriate data to the front end
  - Often happens in a constant loop of requests and responses to the server

# Server side architecture

## Components

- Server (web / application server)
- Database
- Operating System
- API



[Image : Upwork](#)

# Server side architecture

## Components

- The “traditional” back end was simple
  - Mix of the server, databases, APIs, and operating systems that power an app’s front end
- The back end of applications can look very different from application to application based upon
  - frameworks
  - cloud-based servers
  - containerization with a service like Docker
  - Backend-as-a-Service (BaaS) providers
  - APIs to replace more complex processing





# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Backend - Components

Chandan Ravandur N



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

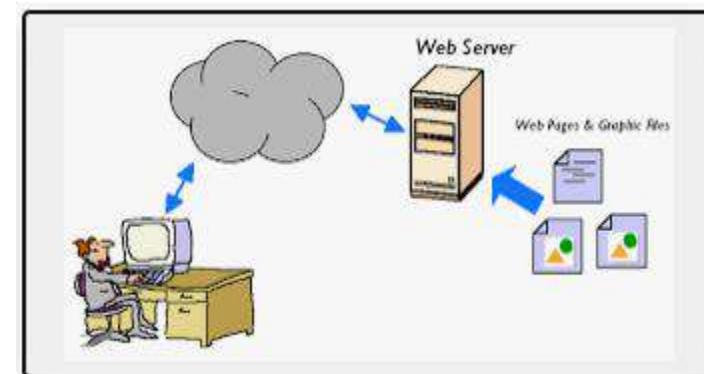
# Backend - WebServers

Chandan Ravandur N

# Web Server

## Defined

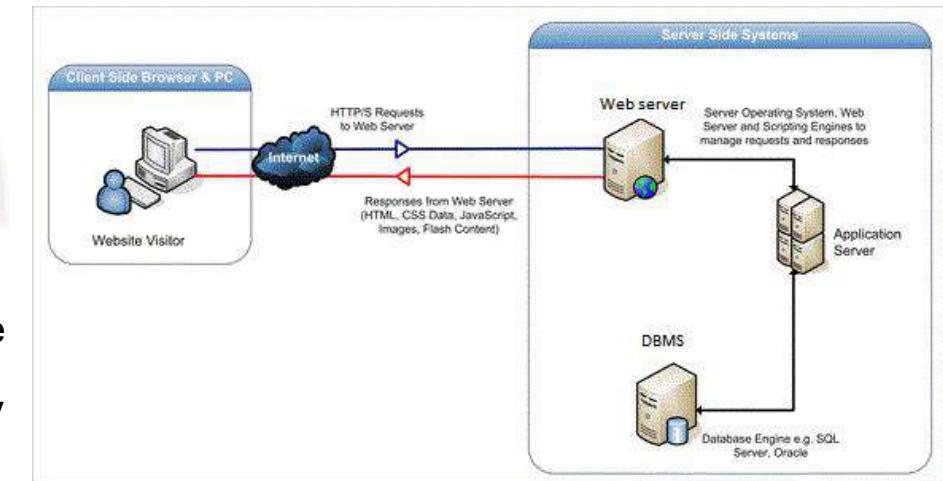
- All computers that host websites must have web server software
- Software and hardware that uses HTTP (Hypertext Transfer Protocol) to respond to client requests made over the World Wide Web
  - Main job of a web server is to display website content through storing, processing and delivering webpages to users
  - Also support SMTP (Simple Mail Transfer Protocol) and FTP (File Transfer Protocol), used for email, file transfer and storage
  - Web server hardware is connected to the internet and allows data to be exchanged with other connected devices
  - software controls how a user accesses hosted files
- Used for
  - web hosting
  - hosting of data for websites
  - web applications



# Web Server

## Working

- Accessed through the domain names of websites and ensures the delivery of the site's content to the requesting user
- The software side at least an HTTP server - understand HTTP and URLs
- The hardware side, a computer that stores web server software and other files related to a website, such as HTML documents, images and JavaScript files
- A user needs a file hosted on web server
  - A person will specify a URL in a web browser's address bar
  - The web browser will then obtain the IP address of the domain name -- translating the URL through DNS (Domain Name System)
  - The browser will then request the specific file from the web server by an HTTP request
  - When the request is received by the web server,
    - the HTTP server will accept the request
    - find the content
    - send it back to the browser through HTTP
  - If the requested page does not exist or if something goes wrong, the web server will respond with an error message. The browser will then be able to display the webpage.



[Image source : stackoverflow](#)

# Web Server - Types

## Dynamic vs. static web servers

- A web server can be used to serve either static or dynamic content
  - Static refers to the content being shown as is
  - Dynamic content can be updated and changed
- A static web server
  - Consist of a computer and HTTP software
  - Considered static because the sever will send hosted files as is to a browser
- Dynamic web server
  - Consist of a web server and other software such as an application server and database
  - Considered dynamic because the application server can be used to update any hosted files before they are sent to a browser
  - Can generate content when it is requested from the database
  - Process is more flexible, it is also more complicated

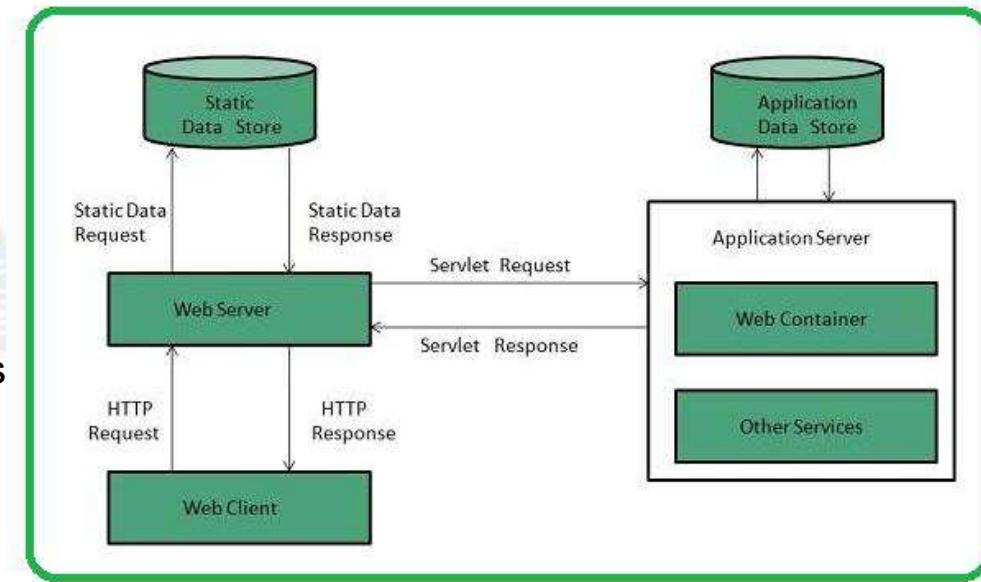
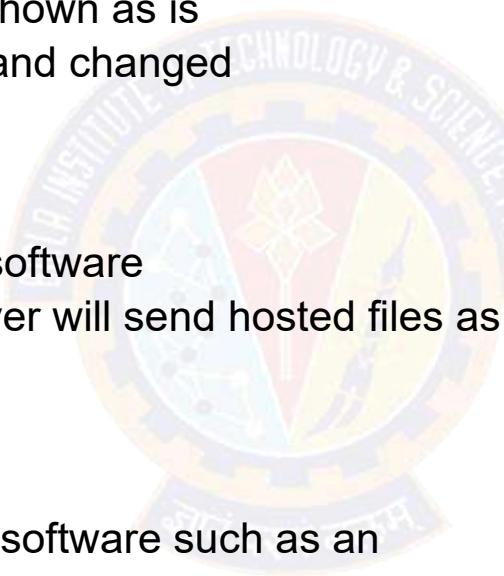


Image Source : [TutorialsPoint](#)

# Common Web servers

- Apache HTTP Server
  - Developed by Apache Software Foundation
  - Free and open source web server
  - For Windows, Mac OS X, Unix, Linux, Solaris and other operating systems
- Microsoft Internet Information Services (IIS)
  - Developed by Microsoft for Microsoft platforms
  - Not open sourced, but widely used
- Nginx
  - A popular open source web server for administrators
  - Has very light resource utilization and scalability
  - Can handle many concurrent sessions due to its event-driven architecture
  - Can be used as a proxy server and load balancer
- Sun Java System Web Server
  - A free web server from Sun Microsystems that can run on Windows, Linux and Unix
  - Well-equipped to handle medium to large websites





# Thank You!

In our next session:

# Server side Components

## Servers: The machinery

- Server acts as the lifeblood of the network
  - Can be on-site or in the cloud
- High-powered computers provide shared resources that need to run for application
  - file storage
  - security and encryption
  - databases
  - email
  - web services

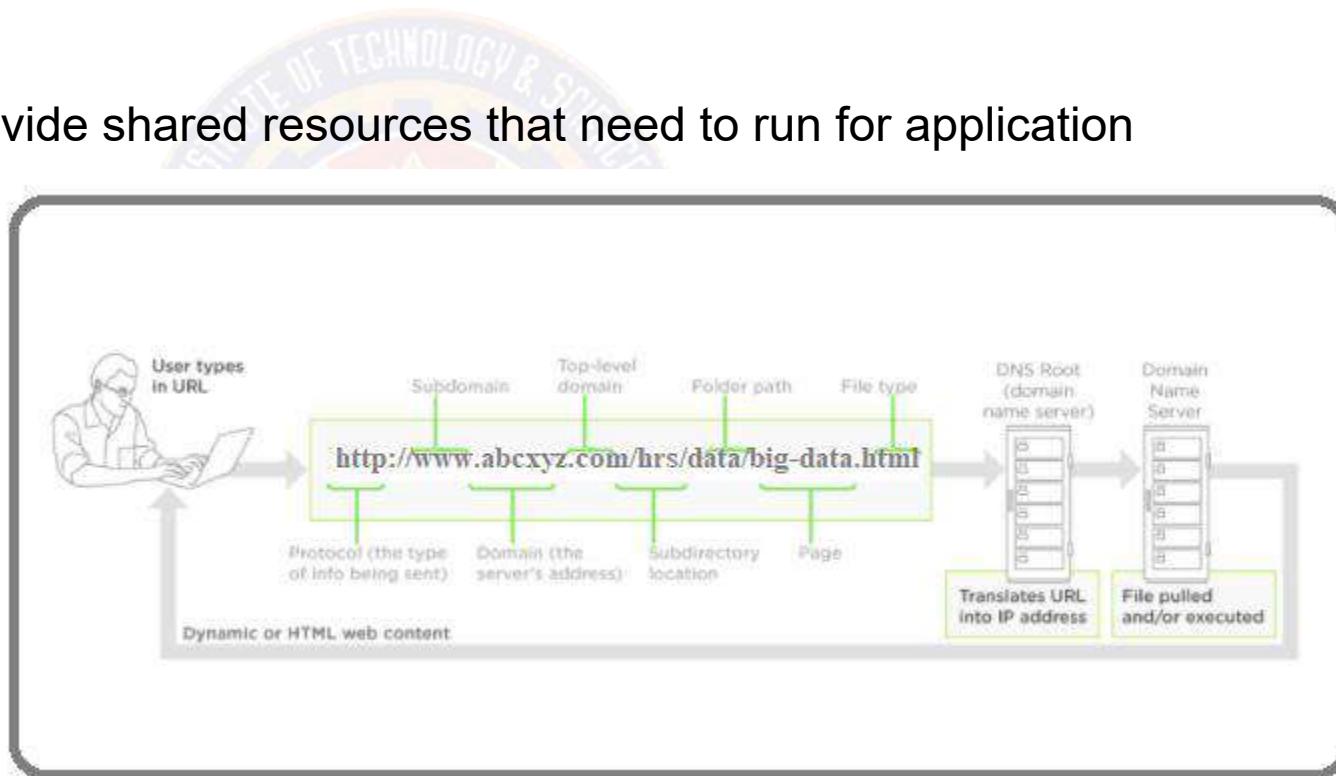
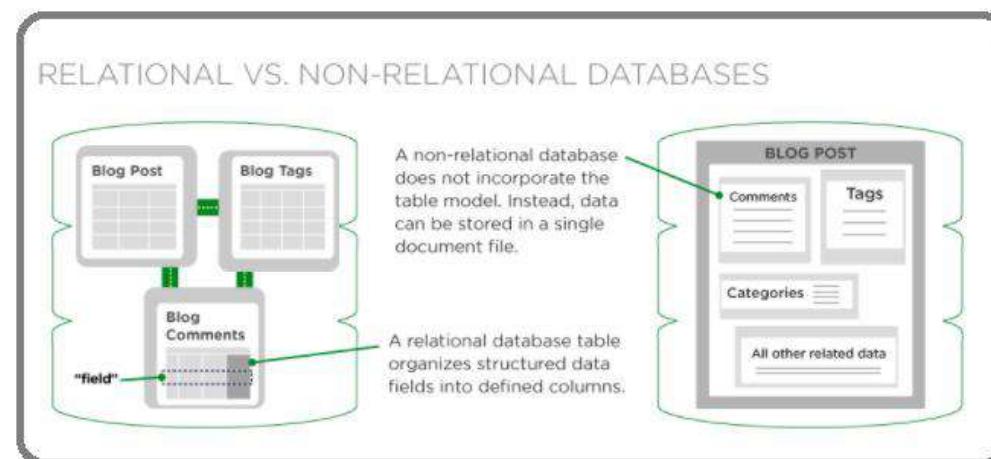


Image : Adapted from Upwork

# Server side Components

## Databases: The brains

- Are the brains that make websites dynamic
- Is responsible for accepting that query, fetching the data, and returning it to the website
  - Searching for a product in an online store
  - Searching for hotel locations within a specific state
- Accepts new and edited data when users of a website or application interact with them
- The client can change information in a database from the browser such as
  - posting articles to a CMS
  - uploading photos to a social media profile
  - updating their customer information

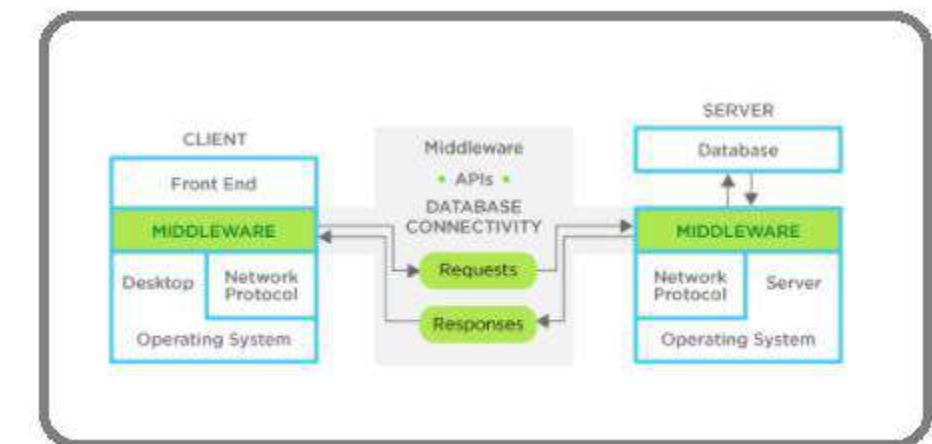
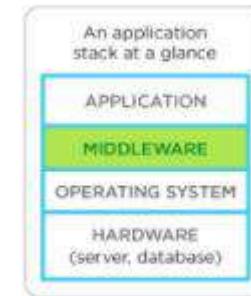


[Image : Upwork](#)

# Server side Components

## Middleware: The plumbing

- Any software on the server that connects an application's front end to its back end
- Think of it as plumbing for your site
  - pipes any communication, like requests and responses, back and forth between your application and server/database
  - you don't see middleware, but it's there and it has to be reliable and always do what's expected of it
- Middleware (server-side software) facilitates client-server connectivity, forming a middle layer between the app(s) and the network
- Can be multi-layered, organized into different layers of a site, whether it's the presentation layer or the business layer
- Web APIs can play into the stack, providing a bridge between the business layer and presentation layer

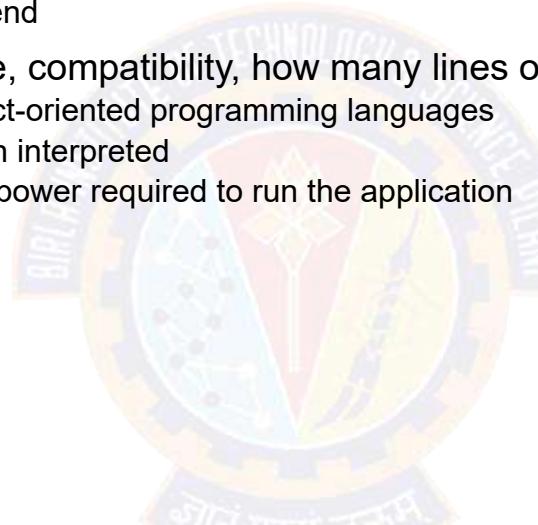


[Image : Upwork](#)

# Server side Components

## Programming languages & frameworks: The nuts & bolts

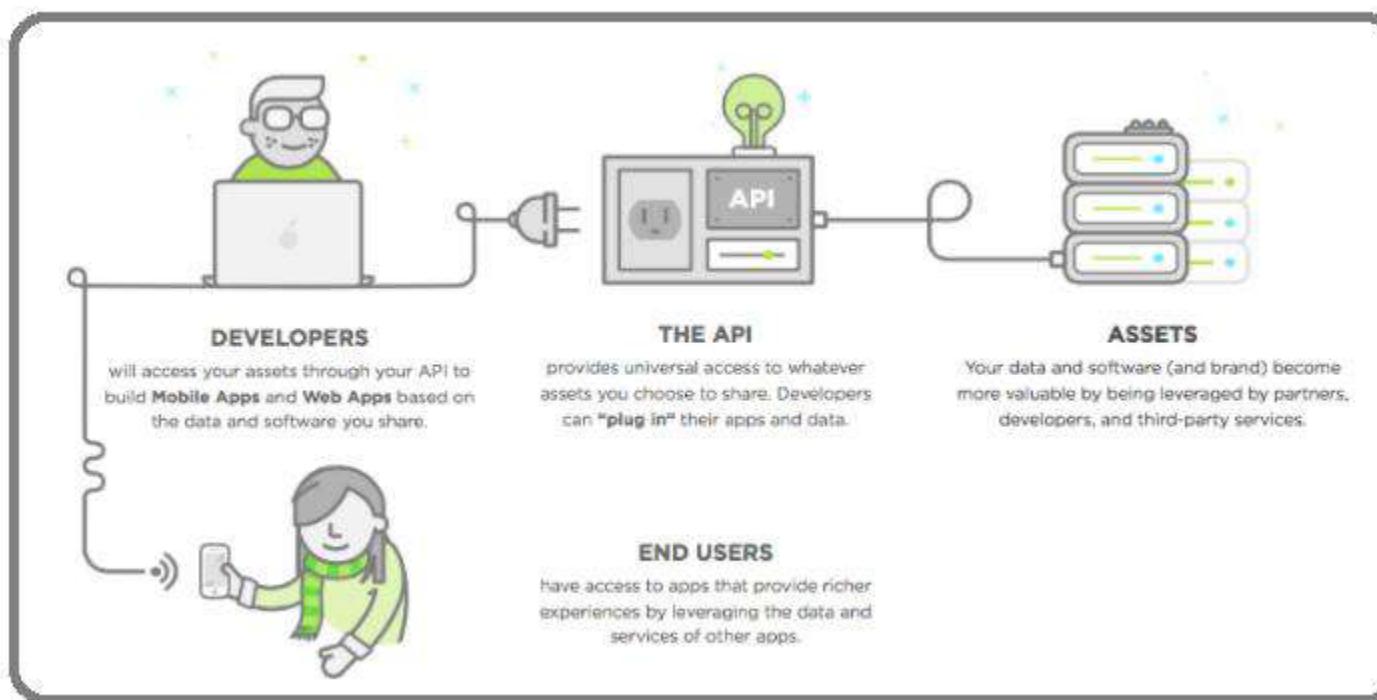
- Can choose from a variety of languages and frameworks depending on
  - the type of application being built
  - the specific processing requirements
  - other components already exist on the back end
- Languages will differ in file size, performance, compatibility, how many lines of code required, and the style of programming
  - Some back-end scripting languages are object-oriented programming languages
  - Other languages may be compiled rather than interpreted
  - affects load time, readability, and processing power required to run the application
- Big hitters in back-end programming, like:
  - Java
  - C# and C++
  - .NET
  - Perl
  - Scala
  - Node.js
- Frameworks that developers use as scaffolding to build server-side applications
  - Django (for Python)
  - Spring framework (for Java)
  - Node.js including MeteorJS and ExpressJS (for JavaScript with Node.js)
  - Ruby on Rails
  - Symfony (for PHP)



# Server side Components

## APIs: Crucial tech in Back-End programming

- Can't talk about the back-end portion of an application these days without touching on APIs (application programming interfaces)
  - Connect software, applications, databases, and services together seamlessly
- Plays an integral role in how most server-side software architectures are built
  - Replacing more complicated programming to allow software to communicate and data to be transferred



[Image : Upwork](#)



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Backend - Server Side Scripting

Chandan Ravandur N

# Server Side Scripting

- A technique used in web development which involves employing scripts on a web server
  - which produce a response customized for each user's (client's) request to the website
- Scripts can be written in server-side scripting languages
  - PHP
  - Python
  - JSP
  - ASP .net
  - Java
- Is often used to provide a customized interface for the user
- Scripts may assemble client characteristics for use in customizing the response based on those characteristics, the user's requirements, access rights, etc.
- Enables the website owner to hide the source code that generates the interface, whereas with client-side scripting, the user has access to all the code received by the client
- A down-side to the use of server-side scripting is that the client needs to make further requests over the network to the server in order to show new information to the user via the web browser



# Server-side script basics

- Runs on a server, embedded in the site's code
- Facilitates the transfer of data from server to browser, bringing pages to life in the browser
- Designed to interact with back-end permanent storage, like databases,
- Runs on-call. When a webpage is “called up,” or when parts of pages are “posted back” to the server with AJAX, server-side scripts process and return data
- Powers functions in dynamic web applications
  - user validation
  - saving and retrieving data
  - navigating between other pages
- Build application programming interfaces (APIs)
  - which control what data and software a site shares with other apps

# Popular server-side languages

- PHP
  - The most popular server-side language on the web, PHP is designed to pull and edit information in the database
  - Most commonly bundled with databases written in the SQL language
  - PHP-powered sites: WordPress, Wikipedia, Facebook
- Python
  - With fewer lines of code, is fast, making it ideal for getting things to market quickly
  - Emphasis is on readability and simplicity, which makes it great for beginners
  - The oldest of the scripting languages, is powerful, and works well in object-oriented designs
  - Python-powered sites: YouTube, Google, The Washington Post
- Ruby
  - Handles complicated logic on the database side of site vrey well
  - Great for startups, easy maintenance, and high-traffic demands
  - Requires developers to use the Ruby on Rails framework, which has vast libraries of code to streamline back-end development
  - Ruby-powered sites: Hulu, Twitter (originally), Living Social, Basecamp

# Popular server-side languages

- C#
  - Language of Microsoft's .NET Framework—the most popular framework on the web
  - Combines productivity and versatility by blending the best aspects of the C and C++ languages
  - Excellent for developing Windows applications, and can be used to build iOS, Android mobile apps with the help of a cross-platform technology like Xamarin
- Java
  - Comes with a huge ecosystem of add-on software components
  - “Compile once, run anywhere” is its motto
  - Excellent for enterprise-level applications, high-traffic sites, and Android apps
  - Java sites: Twitter, Verizon, AT&T, Salesforce



# Thank You!

In our next session:



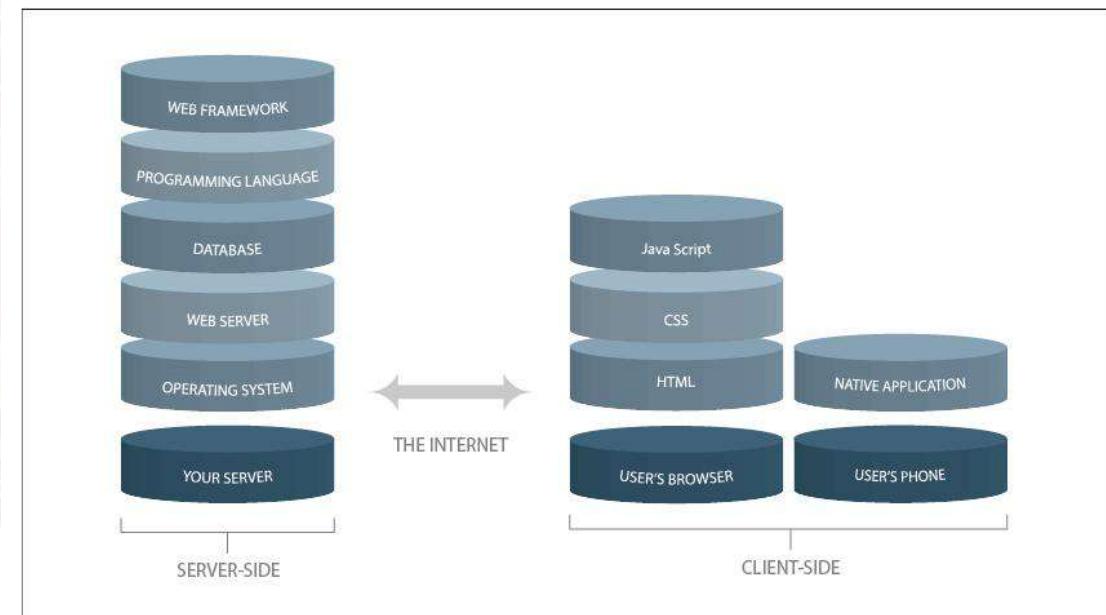
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Backend - TechStacks

Chandan Ravandur N

# Tech stack

- Is the combination of programming languages, tools and frameworks that the developers use to create web and mobile applications
  - Two main components to any application, known as client side and server side, also popular as front end and back end
- A stack is created when one layer of application is built atop the other
  - with the help of codes and hardware modules ranging from generic to specific
- A stack contains different layers of components/servers that developers use to build software applications and solutions



[Image source : hackernoon](#)

# Tech stack

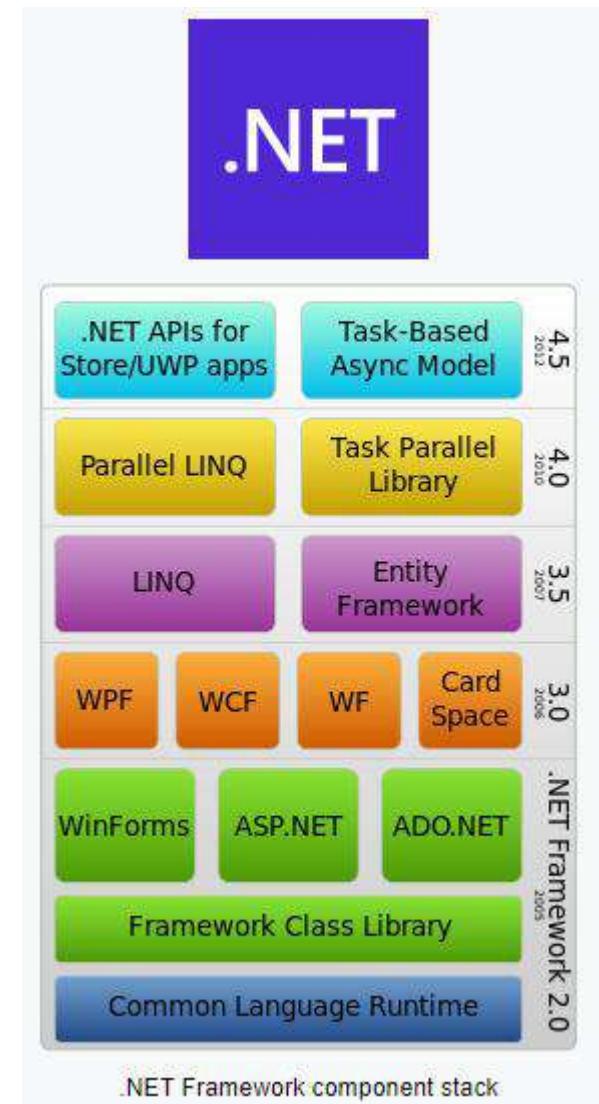
## Deciphered

- Client-side
  - Is where the real interaction with the user happens
  - The user will interact with the website, the web app or a mobile app depending on what he is using
  - Three main technologies in front end: HTML, CSS and JavaScript
- Server-side
  - The back end consists of a server, an application (OS, Web server, Programming language, Web framework), and a database
  - Umbrella term used for websites where developers perform the following functions like
    - programming the business logic
    - server side hosting
    - app deployments
    - integrating with databases
- Common Stacks
  - .net
  - LAMP
  - MEAN / MERN / MEVN / MEEN
  - RoR



# .net stack

- Developed by Microsoft
- Is a feature rich, thoroughly battle tested framework that lets to build dynamic and interactive web apps
- Subset of Overflow Stack, a comprehensive tech stack that fulfils all the requirements of web front-end, database and .NET developers
- C# language and .NET framework form the centerpiece
- Capable of supporting a wide variety of applications
  - right from small scale server side web applications
  - to huge enterprise-level, transaction processing systems
- .NET Stack has about 60 frameworks, platforms, SDKs, IDEs, SOA, libraries, etc. spread over 13 layers

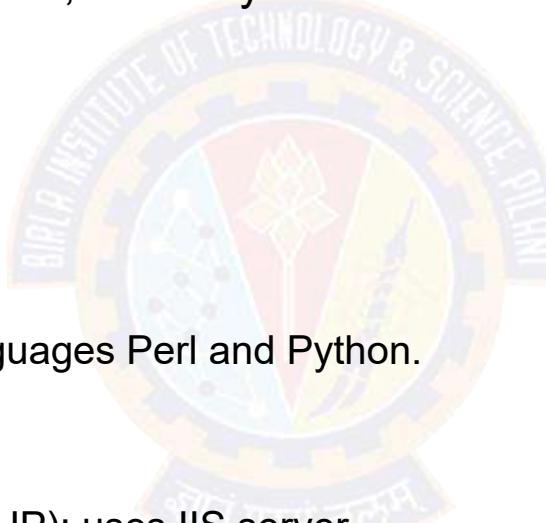


[Image source : wikipedia](#)

# LAMP

## Linux, Apache, MySQL, Python / Perl/ Php

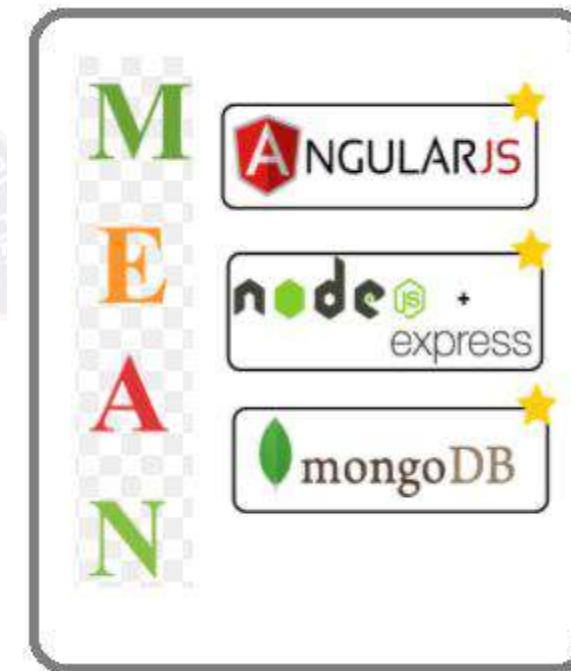
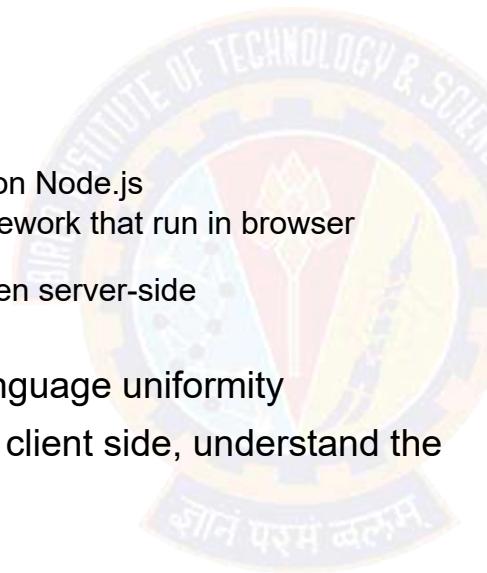
- Made of all open source software components is the one of the earliest stacks to get popular
- The most traditional stack model around, and very solid
- The main components are
  - Linux operating system,
  - Apache HTTP Server
  - MySQL RDBMS
  - PHP programming language
  - PHP is interchangeable with the languages Perl and Python.
- Variations of LAMP include:
  - WAMP (Windows/Apache/MySQL/PHP): uses IIS server.
  - LAPP (Linux/Apache/PostgreSQL/PHP): uses PostgreSQL database variation that's configured to work with enterprise-level projects
  - MAMP (Mac OS X/Apache/MySQL/PHP): Mac OS X operating system variation
  - XAMP (Linux, Mac OS X, Windows/Apache/MySQL/PHP, Perl): More complete bundle, and runs on Linux, Windows, and Mac operating systems



# ME(A/R/V)N Stack

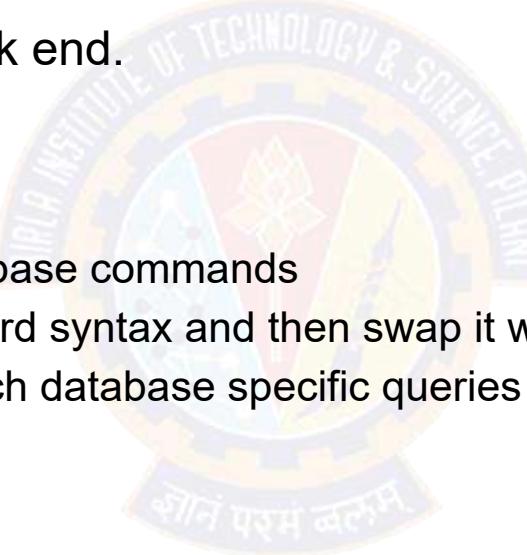
## MongoDB, Express, Angular, Node

- Open source, free Javascript software stack developers use for building dynamic web apps and websites
- Supports MVC pattern
- Components are
  - MongoDB—a NoSQL database
  - Express.js—a web app framework that runs on Node.js
  - Angular JS (or Angular) Javascript MVC framework that run in browser JavaScript engines
  - Node.js—an execution domain for event-driven server-side
- Programs are coded in JavaScript - helps language uniformity
- Makes it easier for developer working on the client side, understand the codes of the server side
- Uses JSON for data transfer
- The variation of MEAN include:
  - MERN: MongoDB, Express.js, React.js and Node.js
  - MEVN: MongoDB, Express.js, Vue.js and Node.js.
  - MEEN: MongoDB, Express.js, Ember.js and Node.js



# Ruby on Rails

- Facilitates quick app development thanks to its rich repository of gems—library integrations
- Highly scalable and it follows the ActiveRecord pattern
- Compatible with MySQL on the back end.
- Has its own built in database
  - Can abstract away the lower data base commands
  - Can write the codes in Active Record syntax and then swap it with lower level database specific queries
  - Useful when you don't use too much database specific queries





# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Backend - API design styles

Chandan Ravandur N

# 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

{ REST }

# gRPC

## Backed by Google

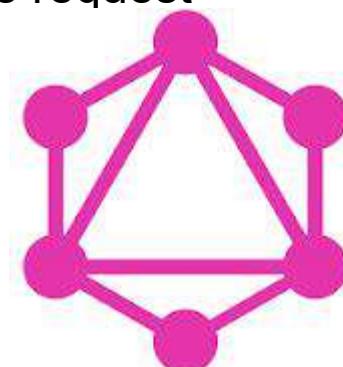
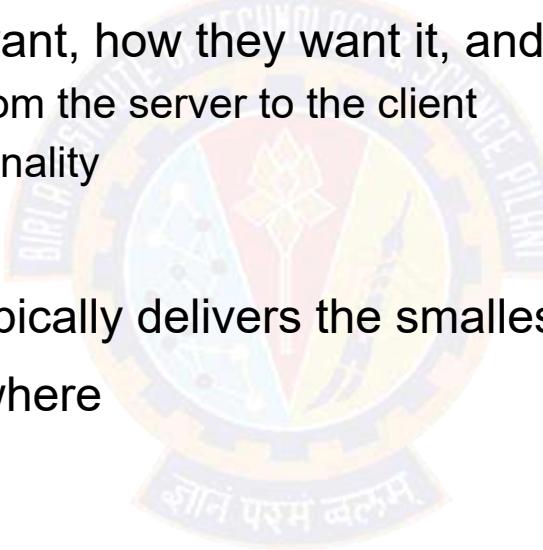


- Actually a new take on an old approach known as RPC, or Remote Procedure Call
- RPC is a method for executing a procedure on a remote server
- RPC functions upon an idea of contracts, in which the negotiation is defined and constricted by the client-server relationship rather than the architecture 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:



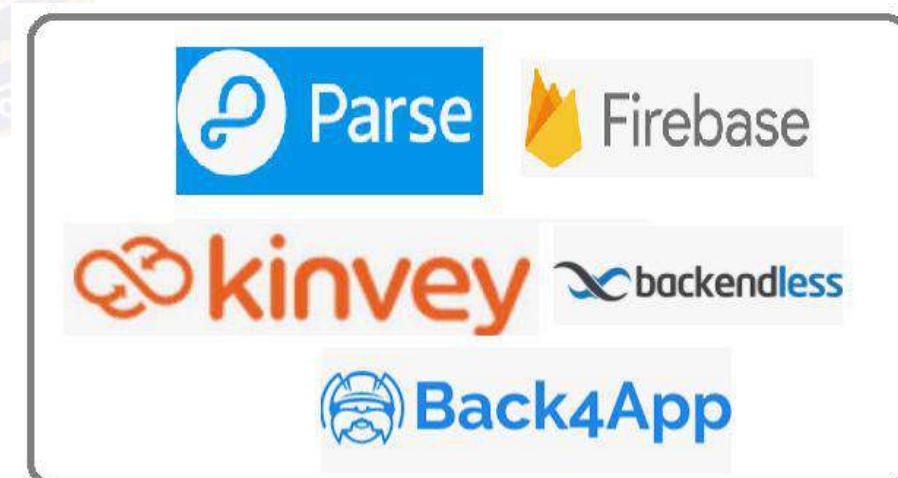
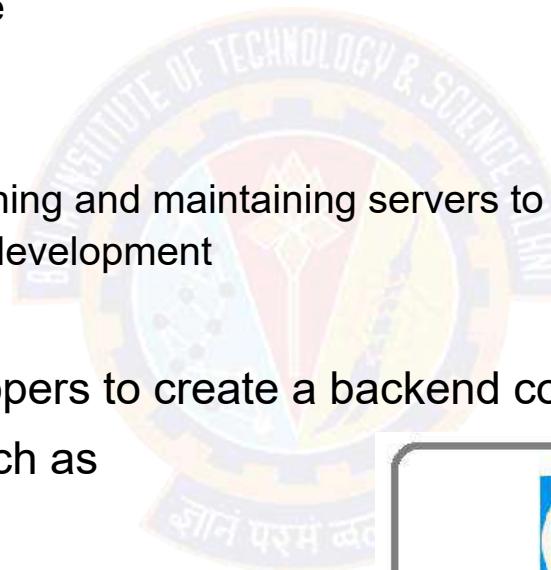
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Backend as a Service (BaaS)

Chandan Ravandur N

# BaaS

- is a platform that
  - automates backend side development
  - takes care of the cloud infrastructure
- App teams
  - outsource the responsibilities of running and maintaining servers to a third party
  - focus on the frontend or client-side development
- Provides a set of tools to help developers to create a backend code speedily
- with help of ready to use features such as
  - scalable databases
  - APIs
  - cloud code functions
  - social media integrations
  - file storage
  - push notifications



# Apps suitable for BaaS

- Social media apps
  - alike Facebook, Instagram
- Real-time chat applications
  - alike WhatsApp
- Taxi apps
  - alike Uber, Ola
- Video and music streaming apps
  - similar to Netflix
- Mobile games
- Ecommerce apps



# Why Backend as a service?

- A BaaS platform solves two problems:
  - Manage and scale cloud infrastructure
  - Speed up backend development
- Business reasons to use BaaS:
  - Reduce time to market
  - Save money and decrease the cost of development
  - Assign fewer backend developers to a project
  - Outsource cloud infrastructure management

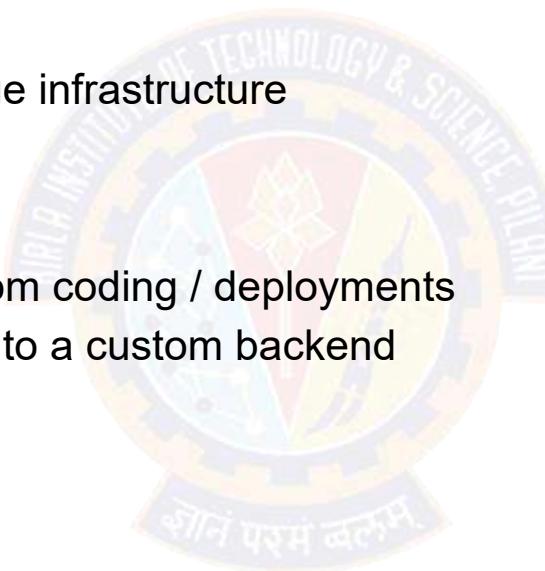
Technical reasons to use BaaS:

- Focus on frontend development
- Excludes redundant stack setup
- No need to program boilerplate code
- Standardize the coding environment
- Let backend developers program high-value lines of code
- Provides ready to use features like authentication, data storage, and search



# Pros-Cons

- Advantages
  - Speedy Development
  - Reduced Development price
  - Serverless, and no need to manage infrastructure
- Disadvantages
  - Less flexible as compared to custom coding / deployments
  - Less customization in comparison to a custom backend
  - Vendor lock-in possible





# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Database Management Systems

Chandan Ravandur N

# Traditional file systems for storing the data

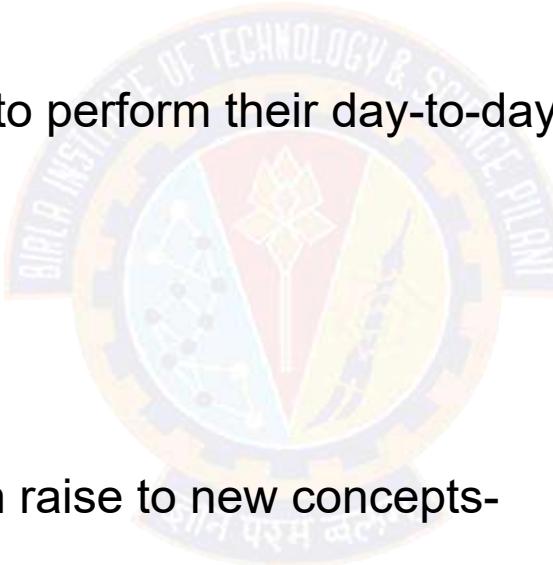
## Issues

- In any business application, information about customers and savings accounts etc. need to be stored
- One way to keep the information on computers is to store in files provided by operating systems (OS).
- Disadvantages of the above System
  - Difficulty in accessing data (possible operations need to be hardcoded in programs)
  - Redundancy leading to inconsistency
  - Inconsistent changes made by concurrent users
  - No recovery on crash
  - The security provided by OS in the form of password is not sufficient
  - Data Integrity is not maintained

# DBMS

## Solution to Data Storage and Retrieval issues

- Databases and Systems to manage them have become significant components of any present day business of any nature
- These databases help businesses to perform their day-to-day activities in an efficient and effective manner
  - Banking
  - Travel ticket reservation
  - Library catalog search
- Advances in technology have given raise to new concepts-
  - Multimedia databases
  - GIS
  - Web data
  - Data warehousing and mining



# Data and Databases

## Related

- Data
  - Known fact that can be recorded and that has implicit meaning
  - Example – Students data
    - ❖ ID
    - ❖ Name
    - ❖ Telephone number
    - ❖ Email id
    - ❖ Programme
  - The data can be stored in a file on a computer
- Database
  - A collection of logically related data
  - Example – University database
    - ❖ collection of all students data
    - ❖ courses data
    - ❖ faculties data
  - A database is designed, built and populated with data for a specific purpose



# DBMS

## Defined

- A collection of programs that enables users to create and maintain databases in a convenient and effective manner
- DBMS is a software system that facilitates the following:

### 1. Defining the database

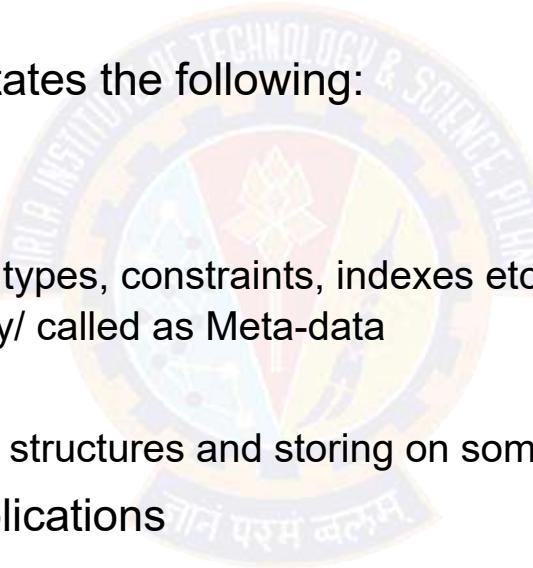
- includes defining the structures, data types, constraints, indexes etc.
- Like Database catalog/Data dictionary/ called as Meta-data

### 2. Constructing the database

- means storing data into the database structures and storing on some storage medium

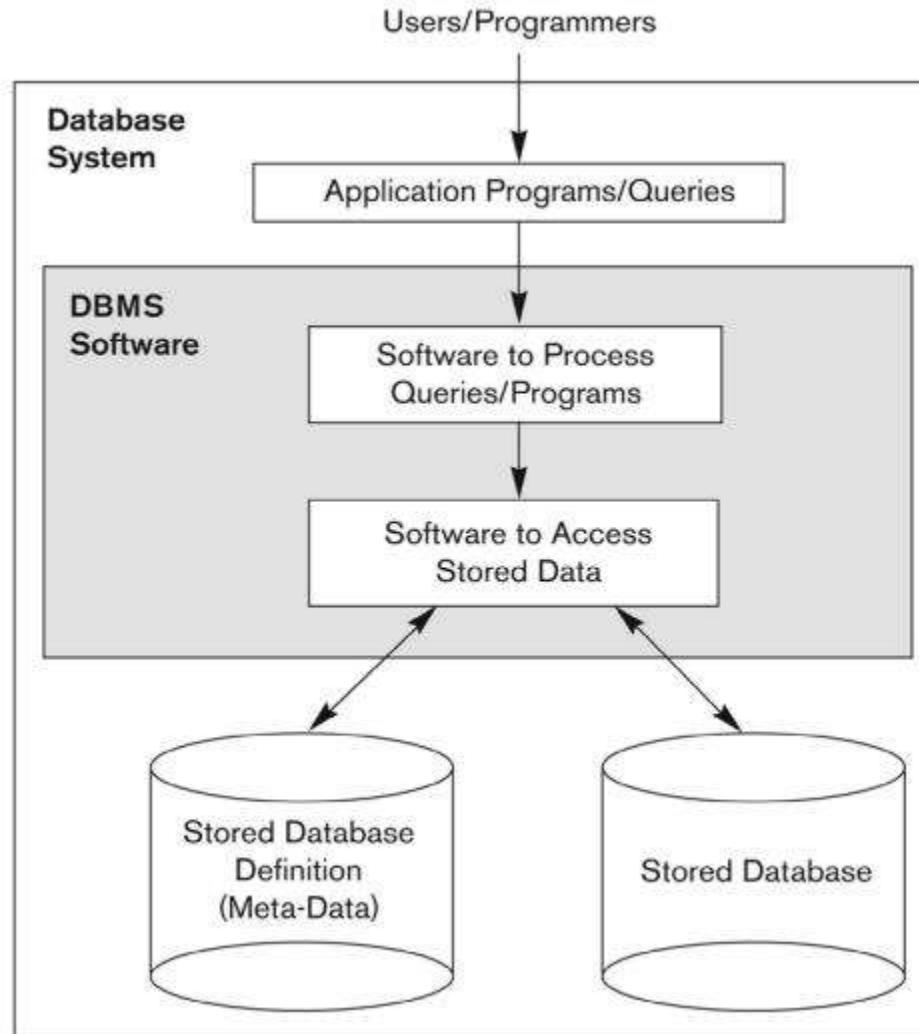
### 3. Manipulating database for various applications

- encompasses activities like –
  - querying the database
  - inserting new records into the database
  - updating some data items
  - deleting certain items from the database



# DBMS

## A simplified database system environment



Adapted from Fundamentals of Database Systems  
by Elmasri, Navathe



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

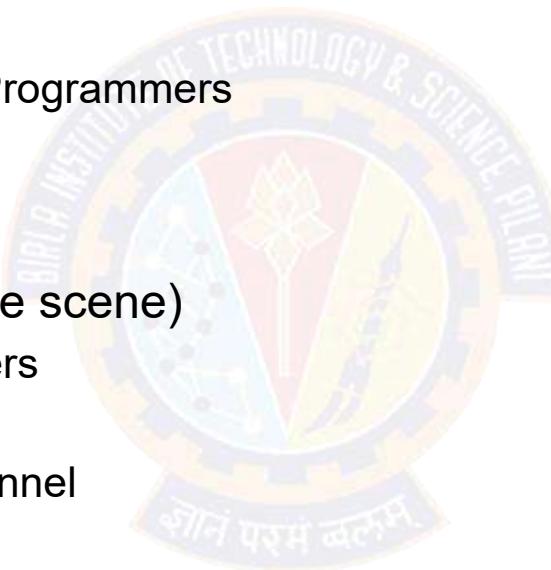
## DBMS - Actors

Chandan Ravandur N

# Actors

## Users of DBMS

- Day Today using Databases (on the scene)
  - Database Admins
  - Database Designers
  - System Analysts and Application Programmers
  - End users
- Maintains the databases (behind the scene)
  - System designers and implementers
  - Tool developers
  - Operators and maintenance personnel



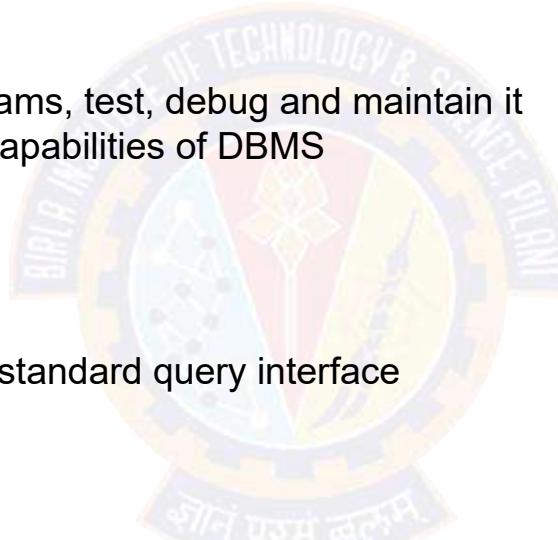
# On the scene actors

## Deals directly with data

- DBA
  - Oversees and manages the database resources
  - Responsible for access management, usage monitoring
  - Needs to look into security aspects, and performance issues
  - In large organizations, pool of DBAs can be deployed for these purpose
  
- Database Designers
  - Defines the schema and actual data which needs to be stored in database
  - Identifies the data to be stored in the database and select proper structures for the same
  - Needs to interact with all stakeholders to understand their data requirements and then design the database matching to those requirements
  - Needs to design views of database based on users requirements

# On the scene actors

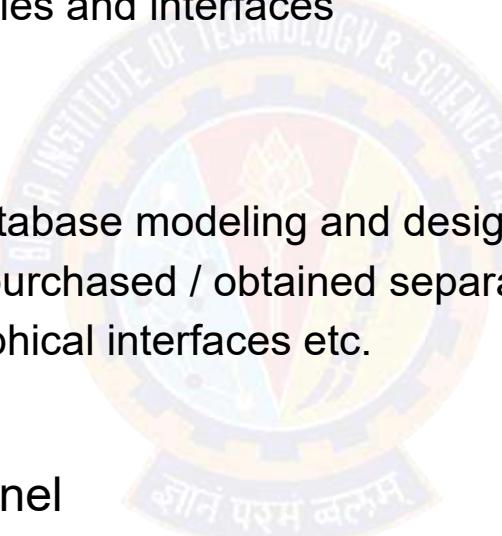
- System Analysts
  - Determine the requirements of end users and prepares the standard canned transactions
- Application Programmers
  - Implements the specifications as programs, test, debug and maintain it
  - Needs to be familiar with full range of capabilities of DBMS
- End Users
- Casual
  - occasionally access database through standard query interface
  - Middle or high level managers
- Naïve (Parametric)
  - sizeable portion of users
  - uses standard types of queries (canned transactions) to deal with database
- Sophisticated
  - knows thoroughly about capabilities of DBMS and implements query / programs to fulfil their data requirements
- Standalone
  - uses menu based user interface to interact with database



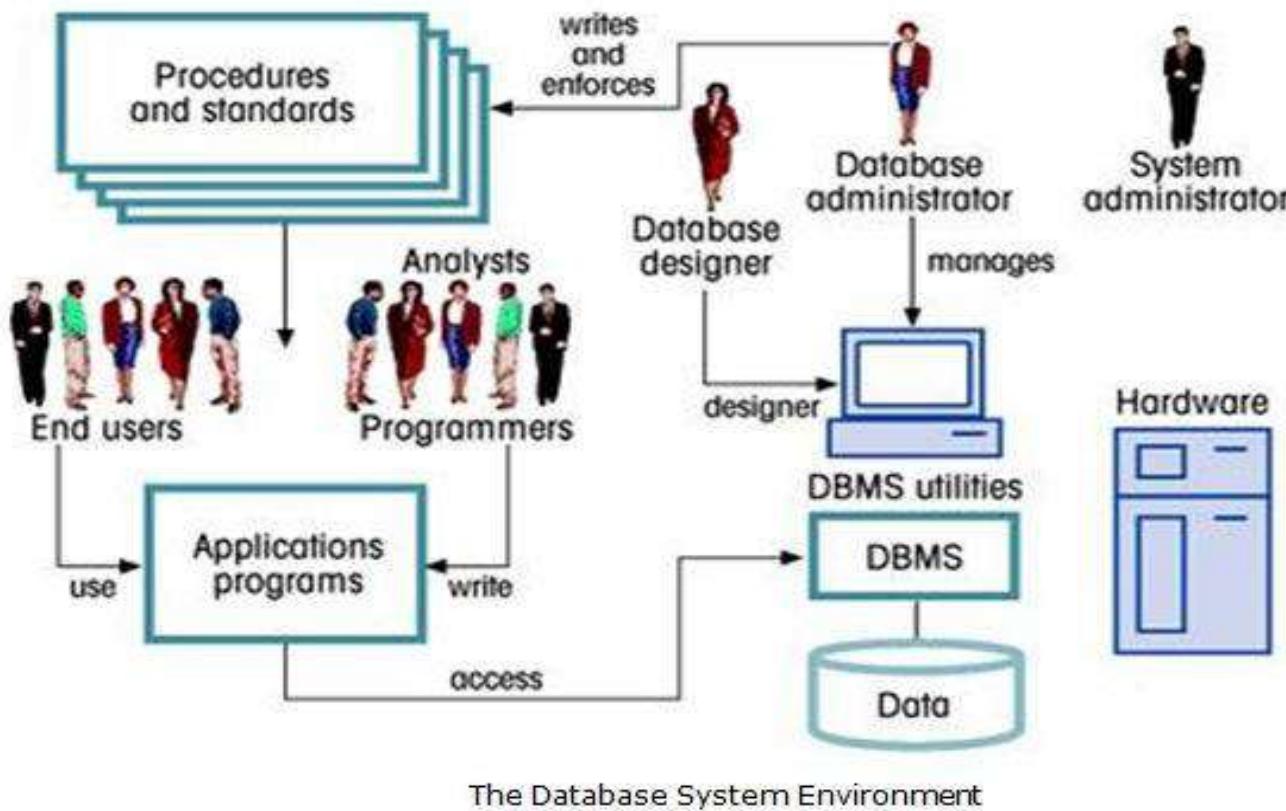
# Behind the Scene

## Not interested into the content of database

- System designers and implementers
  - DBMS is complex software – backup , security, catalog, query processing , interface etc.
  - Design and implement these modules and interfaces
- Tool developers
  - Implements tools that facilitates database modeling and design, database system design etc.
  - Many times optional, needs to be purchased / obtained separately need basis
  - Performance monitoring tools, graphical interfaces etc.
- Operators and Maintenance personnel
  - Responsible for actual running and maintenance of hardware and software environment for DBMS



# In Summary



[Source : MyReadingRoom](#)

Reference :  
Fundamentals of Database Systems, by Elmasri, Navathe



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Advantages Disadvantages of DBMS

Chandan Ravandur N

# DBMS

## Advantages

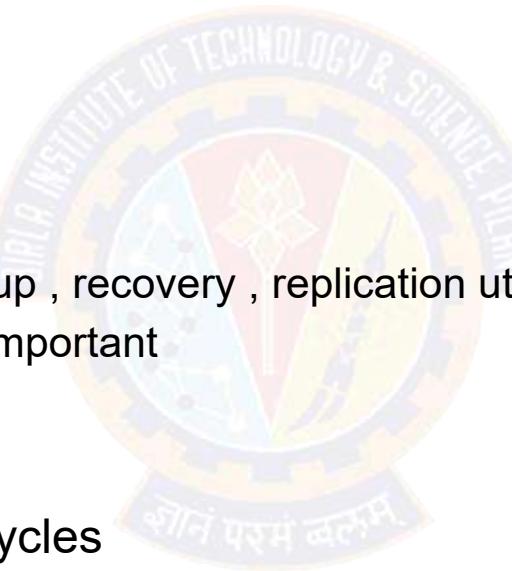
- Data independence
- Efficient data access
- Data integrity and security
- Easier Data Administration
- Concurrent access and Crash recovery
- Reduced application development time



# DBMS

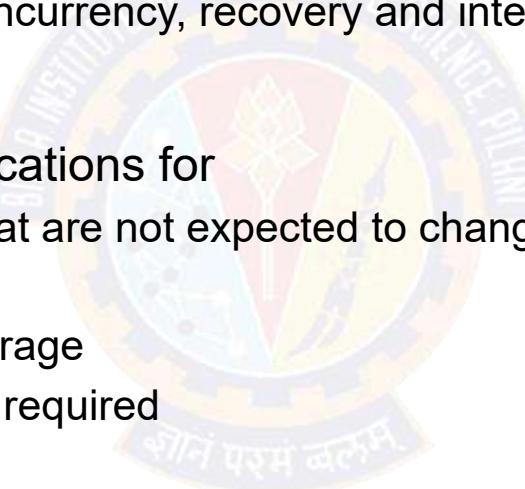
## Disadvantages

- Increased cost
  - Hardware / software cost
  - Training cost
  - People cost
- Increased complexities
  - Lot of components involved – backup , recovery , replication utilities
  - Integration among sub systems is important
  - Specialized skillsets required
- Frequent upgrades / maintenance cycles
  - Patches needs to be applied
  - Versions need to be upgraded
  - Needs to insure that nothing is broken



# When not to use DBMS?

- Overhead costs associated with DBMS
  - High initial investment in hardware, software and training
  - The generality that DBMS provides for defining and processing data
  - Overhead for providing security, concurrency, recovery and integrity function
- Develop customized database applications for
  - Simple, well defined feature sets that are not expected to change at all
  - Stringent, real time requirements
  - Embedded systems with limited storage
  - No multiple users access to data is required





# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Classification of DBMS

Chandan Ravandur N

# Classification of DBMS

## Criteria's

- Data Model
  - On which DBMS is based
- Number of Users supported
  - Accessing the database
- Number of sites
  - Over which database is distributed
- Cost
- Usage Purpose



# Classification of DBMS

## Data Model

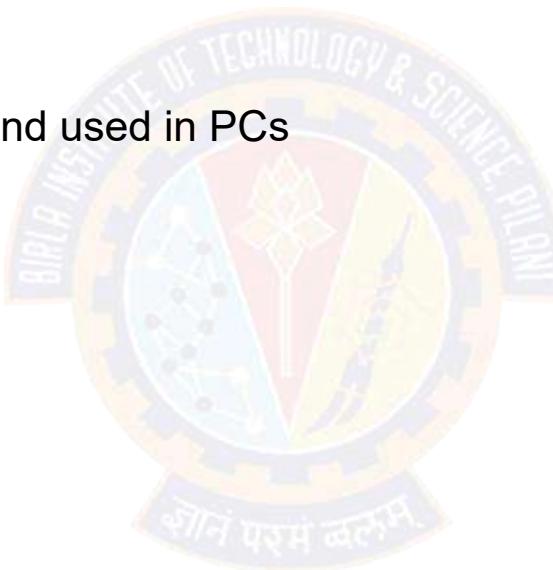
- Relational Data model
  - Quite common in commercial DBMS products
  - Aka SQL systems
  - Oracle , SQLServer, MySQL
- Object Data model
  - Not much usage - Tornado
- NoSQL Data model
  - Aka Big Data systems
  - Further classified as
    - ❖ Document based – MongoDB
    - ❖ Graph based – Neo4J
    - ❖ Column based – HBase, Cassandra
    - ❖ Key-value based – Redis, DynamoDB
- Tree structured model
  - Conventional XML based systems



# Classification of DBMS

## Number of Users supported

- Based upon concurrent number of users supported
- Single User Systems
  - Supports only one user at a time and used in PCs
- Multi User Systems
  - Includes majority of DBMS
  - Support concurrent multiple users



# Classification of DBMS

## Number of sites

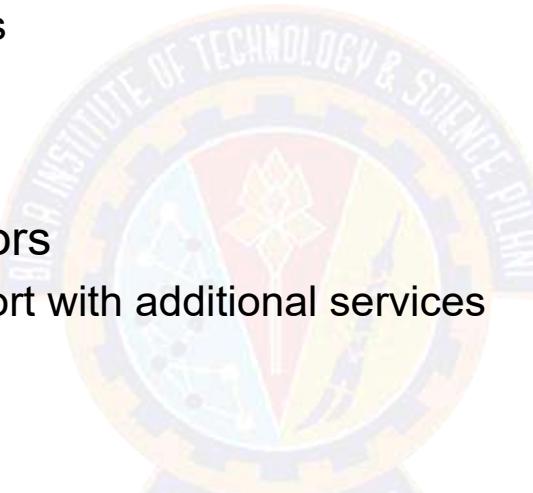
- Number of places where database is distributed
- Centralized
  - Data is stored at a single computer site
  - Can support multiple users
  - Prone to failure
- Distributed
  - Actual database and DBMS software distributed over many sites connected by network
  - Big Data systems are common with massive distributed architecture
  - Data often replicated on multiple sites – making it failure proof



# Classification of DBMS

## Cost

- Open Source Products
  - Source code available for anybody
  - Driven by community of developers
  - MySQL, PostgreSQL
- Open Source with Third party vendors
  - Third party vendors provides support with additional services
- Licenced versions
  - Most commercial DBMS products offers various licences
  - Personnel usage – less cost, may not all features supported / required
  - Based on sites count – allows unlimited use of DBMS with any number of copies running on customer site
  - Based on users count – number of concurrent users supported



# Classification of DBMS

## Usage Purpose

- Special Purpose
  - When performance is primary concern , such special purpose systems are used
  - Optimized
  - Can't be used for other types of data
  - DBMS for embedded devices
- General Purpose
  - Flexible enough to be used various purposes



Reference :  
Fundamentals of Database Systems, by Elmasri, Navathe



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

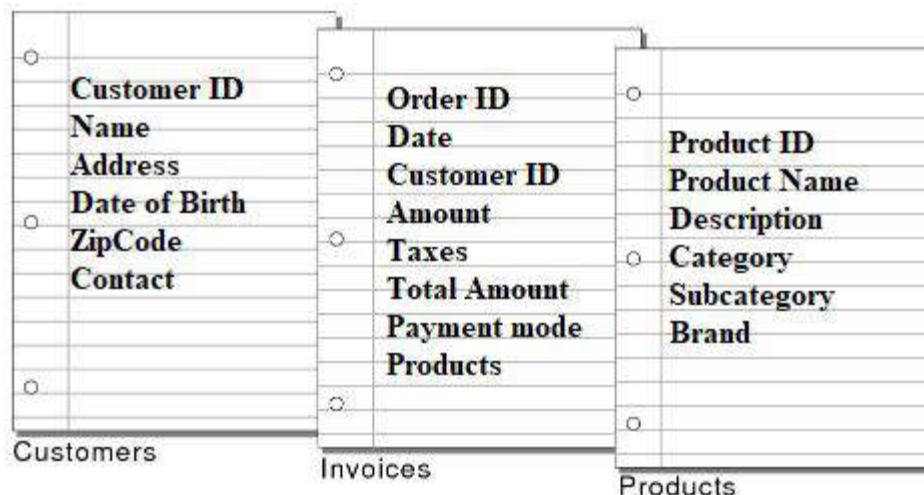
# Relational DBMS

Chandan Ravandur N

# Table

- Table
  - An arrangement of words, numbers or signs , or combinations of them
  - as in parallel columns,
  - To exhibit a set of facts or relations in a definite, compact and comprehensive form

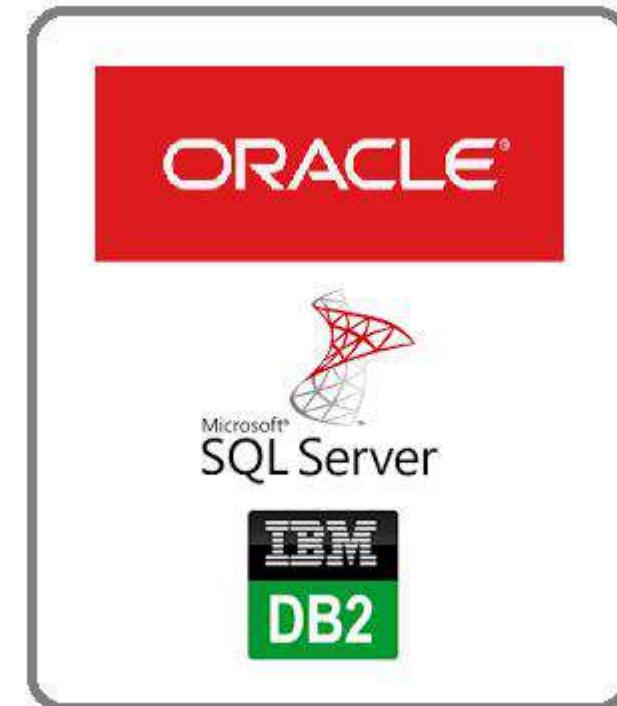
- Webster's Dictionary of English Language



# Relational DBMS

## Table based

- Relational model proposed by Codd in 1970
- Revolutionized the database field and replaced earlier models
  - Hierarchical and network data models
- Several vendors offering relational DBMS software
  - Oracle, IBM DB2, Informix, Sybase, MS Access, MS SQL Server etc
- **Ubiquitous in marketplace and multibillion dollar industry!**
- Bases on very simple and elegant data model
- Provides easy to understand use query language

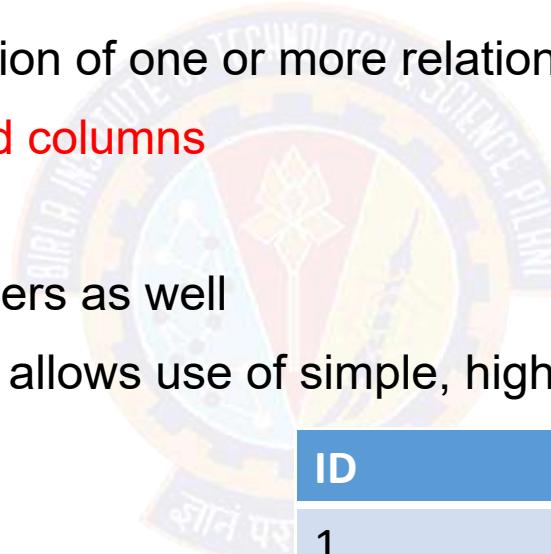


# Relational model

## Relation

- Based on concept of **relation** or table
- Database is represented as collection of one or more relations
- Each relation is **table with rows and columns**
- Simple to understand for novice users as well
- Structured Query Language (SQL) allows use of simple, high level language to query data

- Advantage
  - Simple data representation
  - Ease with which complex queries can be written

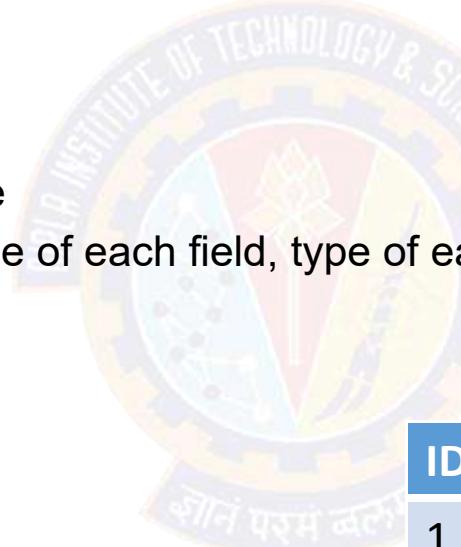


ID	Name	Age	Marks
1	Suresh	21	34
2	Dinesh	22	21
3	Mahesh	21	45

# Relation

## Explained

- Relation is main construct of RDBMS
- Consists of relation schema and relation instance
- Relation schema
  - describes the columns of the table
  - Specifies the relations name, name of each field, type of each field
- Relation instance
  - table containing the records
  - Set of tuples
  - Each tuple has same number of fields as the schema



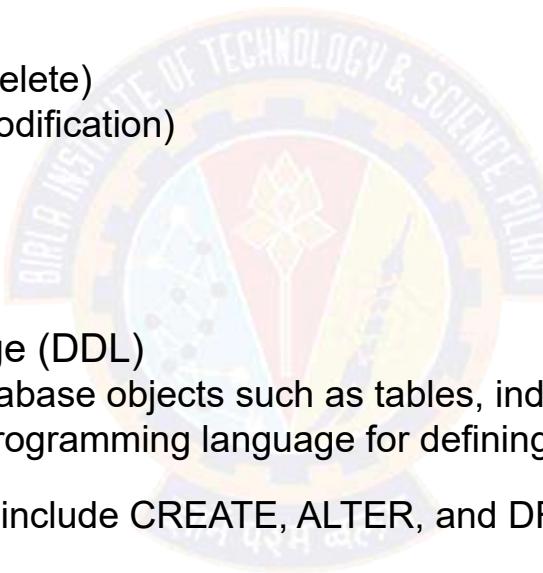
STUDENT		
STUDENT_ID (PK)	NUMBER(8,0)	NOT NULL
SALUTATION	VARCHAR2(5)	NULL
FIRST_NAME	VARCHAR2(25)	NULL
LAST_NAME	VARCHAR2(25)	NOT NULL
STREET_ADDRESS	VARCHAR2(50)	NULL
ZIP (PK)	VARCHAR2(5)	NOT NULL
PHONE	VARCHAR2(15)	NULL
EMPLOYER	VARCHAR2(50)	NULL
REGISTRATION_DATE	DATE	NOT NULL
CREATED_BY	VARCHAR2(30)	NOT NULL
CREATED_DATE	DATE	NOT NULL
MODIFIED_BY	VARCHAR2(30)	NOT NULL
MODIFIED_DATE	DATE	NOT NULL

ID	Name	Age	Marks
1	Suresh	21	34
2	Dinesh	22	21
3	Mahesh	21	45

# SQL

## Language of RDBMS

- SQL is a domain-specific language used in programming and designed for managing data held in a relational database management system (RDBMS)
- The scope of SQL includes
  - data query
  - data manipulation (insert, update and delete)
  - data definition (schema creation and modification)
  - data access control
- SQL Consists of sublanguages
- Data definition or data description language (DDL)
  - A syntax for creating and modifying database objects such as tables, indexes, and users
  - Statements are similar to a computer programming language for defining data structures, especially database schemas
  - Common examples of DDL statements include CREATE, ALTER, and DROP
- Data manipulation language (DML)
  - A syntax for adding (inserting), deleting, and modifying (updating) data in a database
  - Statements consists of different clauses like where , group , having etc.



Reference :  
Database Management Systems, by Ramakrishna



# Thank You!

In our next session:

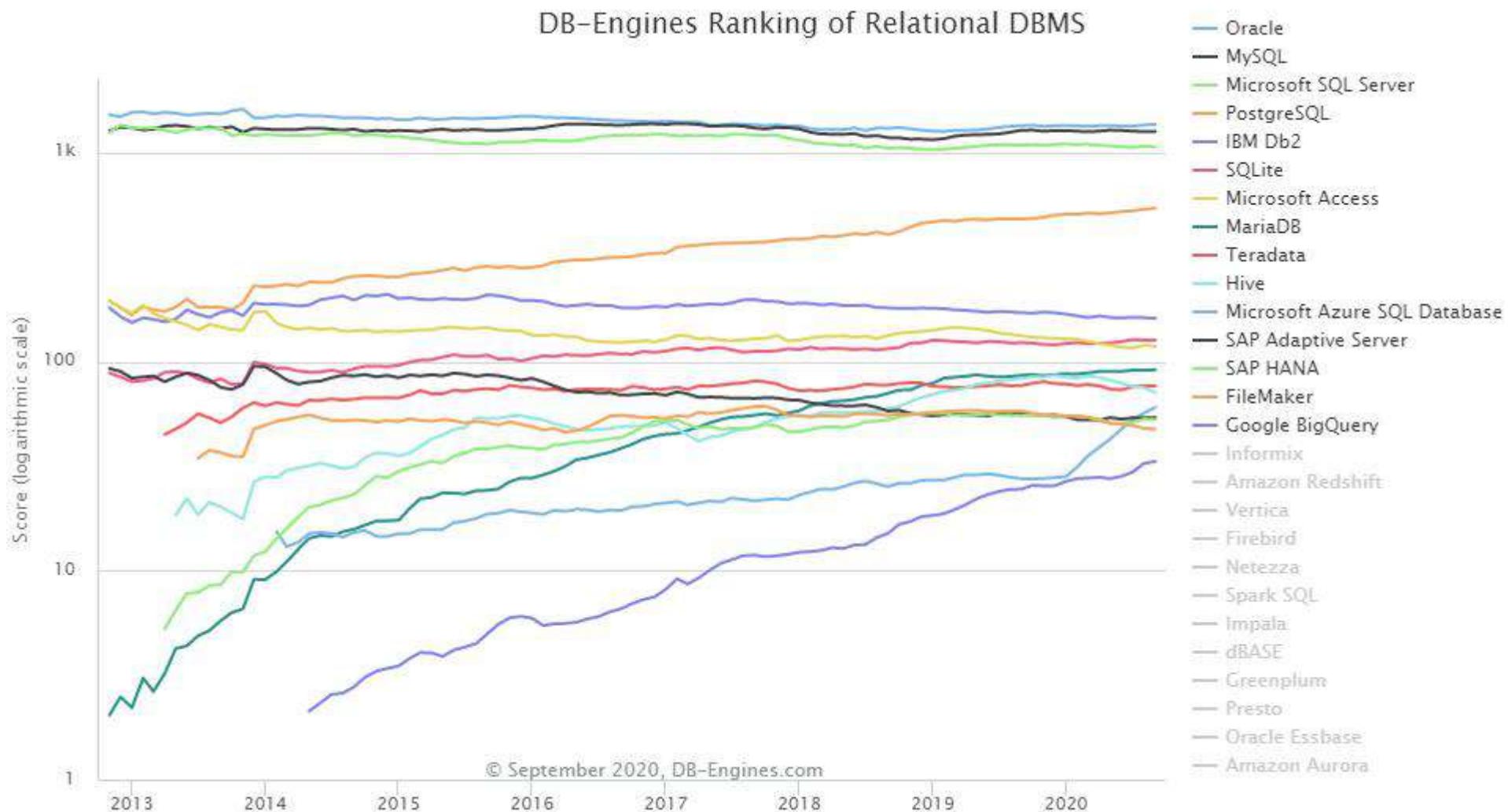


**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# RDBMS - Landscape

Chandan Ravandur N

# RDBMS Landscape



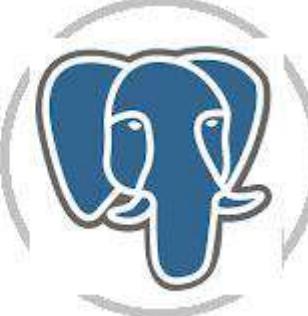
Source : [db-engines](https://db-engines.com)

# Dominant Players

- Commercial Vendors



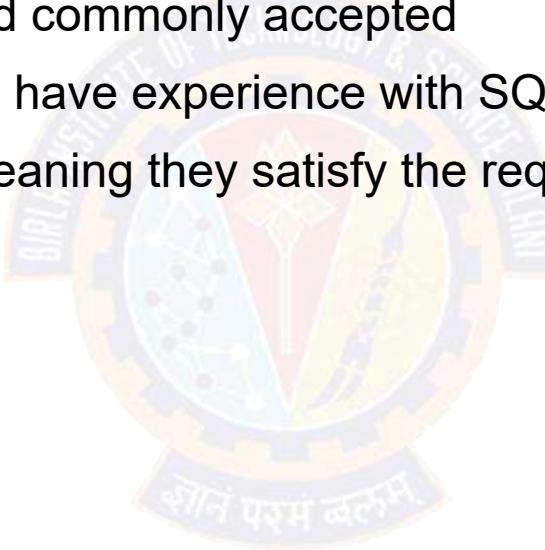
- Open Source Challengers



# RDBMS

## Advantages

- Well-documented and mature technologies, and RDBMSs are sold and maintained by a number of established corporations
- SQL standards are well-defined and commonly accepted
- A large pool of qualified developers have experience with SQL and RDBMS
- All RDBMS are ACID-compliant, meaning they satisfy the requirements of
  - Atomicity
  - Consistency
  - Isolation
  - Durability



# RDBMS

## Disadvantages

- RDBMSs don't work well — or at all — with unstructured or semi-structured data due to schema and type constraints
  - Makes them ill-suited for big data analytics
- The tables in relational database will not necessarily map one-to-one with an object or class representing the same data
  - Resulting into mismatch between tables and data structures used in programme
- When migrating one RDBMS to another, schemas and types must generally be identical between source and destination tables for migration to work (schema constraint)
- Extremely complex datasets or those containing variable-length records are generally difficult to handle with an RDBMS schema



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# NoSQL Databases - Categories

Chandan Ravandur N

- Not Only SQL
- Meant to convey that many applications need systems other than traditional relational SQL systems to manage their data management needs
- Most of them are distributed databases or distributed storage systems with focus on
  - ✓ Semi structured data storage
  - ✓ High performance
  - ✓ Availability
  - ✓ Data replication
  - ✓ Scalability



Image Source : DataVersity

# Emergence of NoSQL systems

## Use cases

- Free email application like Gmail
  - Have millions of users, each user with thousands of emails – need for storage
  - SQL may not be appropriate because
    - ✓ It has too many services which are not required here
    - ✓ Structured data model is restrictive
- Facebook – social media platforms
  - Have millions of users, each user submits posts – images, videos, text , displayed on other users pages
  - User profiles, relationships , posts requires – storage
  - Needs to show the posts to other users – processing
  - SQL many not be appropriate because
    - ✓ Can not handle some of this data well
    - ✓ Can not handle when storage is huge and distributed across nodes



# Emergence of NoSQL systems (2)

## Solutions

- BigTable
  - ✓ Googles proprietary NoSQL system used in many of Googles applications
  - ✓ dealing with vast amounts of data storage like Gmail, Maps, Web site indexing etc.
  - ✓ Apache Hbase is open source version of BigTable
  - ✓ Column-based or wide column stores



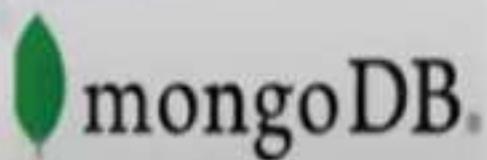
- DynamoDB
  - ✓ Amazons NoSQL system available on AWS
  - ✓ Innovation in key-value type data stores



- Cassandra
  - ✓ Facebooks NoSQL system
  - ✓ Uses concepts from both key-value stores and column-based systems



- MongoDB
  - ✓ Document based NoSQL system



# Characteristics of NoSQL systems

- Related to Distributed databases and distributed systems
  - ✓ Scalability
  - ✓ Availability
  - ✓ Replication models
  - ✓ Sharding
  - ✓ High performance
- Related to data models and query languages
  - ✓ No fixed schema
  - ✓ Less powerful query languages
  - ✓ Versioning

# Characteristics of NoSQL systems (2)

## Related to Distributed databases and distributed systems

- Scalability
  - ✓ Horizontal or Vertical
  - ✓ In NoSQL systems, horizontal scalability is used by adding more nodes for data storage and processing as the volume of data grows
  - ✓ Used when system is operational, so techniques for distributing the existing data among new nodes without interrupting the system operation is of upmost importance
- Availability
  - ✓ NoSQL systems requires continuous system availability
  - ✓ Data is replicated over two or more nodes in transparent manner so that if one node fails, data is still available on the other nodes
  - ✓ Replication improves the data availability and performance as the reads can be answered from replicas as well
  - ✓ Write performance become cumbersome as update must be applied to every copy

# Characteristics of NoSQL systems (3)

## Related to Distributed databases and distributed systems

- Replication models
- Master-slave or Master – master
- Master – slave configuration
  - ✓ One copy as master, others as slaves
  - ✓ Changes are first applied to master and then propagated to slaves, using eventual consistency
  - ✓ Reads can be done in two ways
  - ✓ Read from master – always latest data returned
  - ✓ Read from slaves – no guarantee of latest data as its based on eventual consistency
- Master-Master configuration
  - ✓ Allows read and write at any of replicas but may not guarantee that reads at nodes that store different copies see the same values

# Characteristics of NoSQL systems (4)

## Related to Distributed databases and distributed systems

- Sharding of files
  - ✓ Files (collections of data objects) have millions of records which are accessed simultaneously
  - ✓ Not practical to store the file at one node
  - ✓ Sharding – horizontal partitioning is used to distribute loads of accessing the files records to multiple nodes
  - ✓ Combination of sharding and replication improve load balancing and data availability
- High Performance Data Access
  - ✓ Finding a data record is important in many NoSQL systems
  - ✓ Either uses hashing or range partitioning of keys
  - ✓ In hashing, hash function  $h(K)$  is applied to key  $K$  and location of object with key  $K$  is determined by value of  $h(K)$
  - ✓ In range partitioning, location is determined by range of key values, location  $i$  would hold objects whose key values  $K$  are in range  $K_{\min} \leq K \leq K_{\max}$ .

# Characteristics of NoSQL systems

## Related to Data models and query language

- Not requiring Schema
  - ✓ Allowed by semi structured, self describing data
  - ✓ Not required to have a schema in most of NoSQL systems
  - ✓ There may not be a schema to specify constraints, any constraints on the data would have to be programmed in applications
  - ✓ Languages like JSON, XML are used while defining models
- Low powerful query languages
  - ✓ Many applications not require powerful query languages as SQL as read queries in these systems often locate single objects in a single file based on their object keys
  - ✓ NoSQL systems provide a set of functions and APIs
  - ✓ Supports SCRUD operations – Search , CRUD
  - ✓ Many systems do not provide join operations as part of language, hence needs to be implemented in application
- Versioning

# NoSQL

- Not Only SQL
  - Coined by Carlo Strozzi in 1998 to name lightweight, open source, relational database that did not expose the standard SQL interface
  - Johan Oskarsson , in 2009 reintroduced the term to discuss open-source distributed network
- 
- Features
    - ✓ Open source
    - ✓ Non-relational
    - ✓ Distributed
    - ✓ Schema-less
    - ✓ Cluster friendly
    - ✓ Born out of 21<sup>st</sup> century web applications



# Categories (1)

- Document based
- Key-Value stores
- Column based or wide column
- Graph based



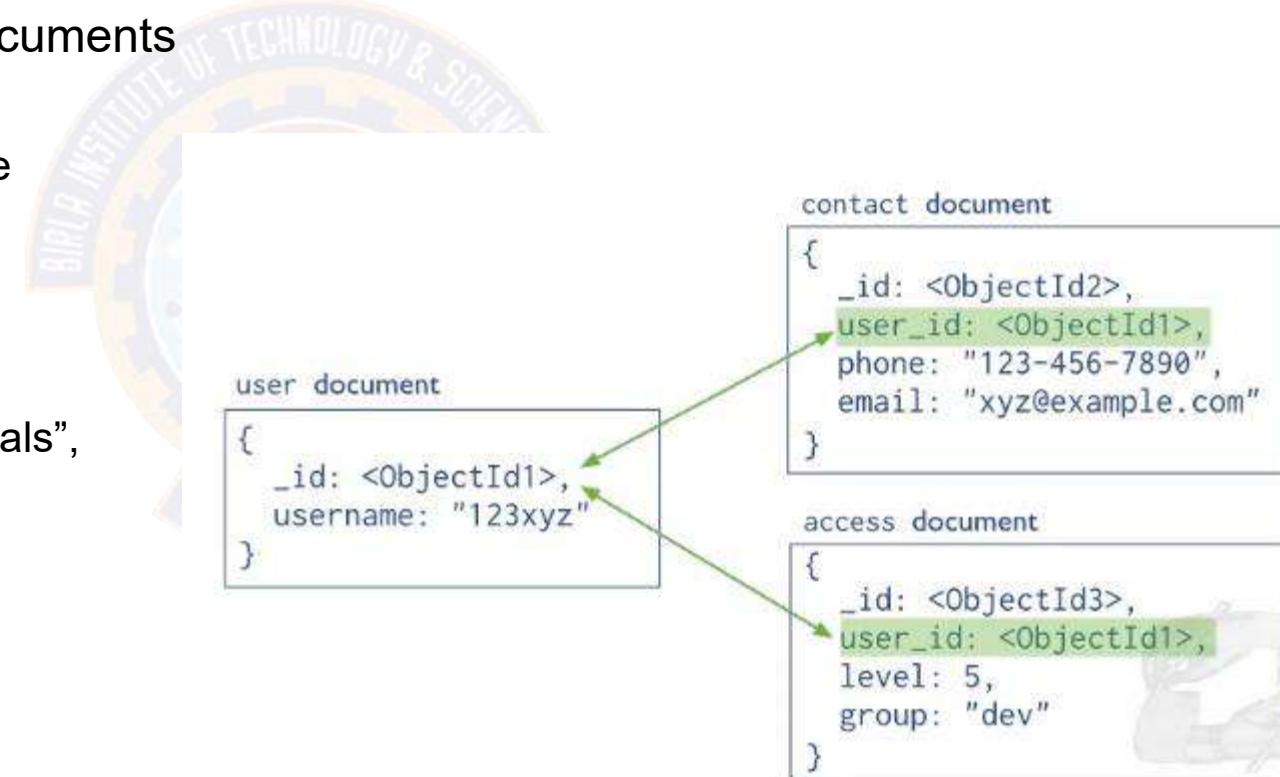
Key-value stores	Column-Oriented	Document based	Graph based
Riak	Cassandra	MongoDB	Neo4J
Redis	Hbase	CouchDB	InfiniteGraph
Membase	HyperTable	RavenDB	AllegroGraph

# Categories (2)

## Document based

- Store data in form of documents using well known formats like JSON
- Documents accessible via their id, but can be accessed through other index as well
- Maintains data in collections of documents
- Example,
  - MongoDB, CouchDB, CouchBase
- Book document :

```
{  "Book Title" : "Database Fundamentals",  "Publisher" : "My Publisher",  "Year of Publication" : "2020"}
```

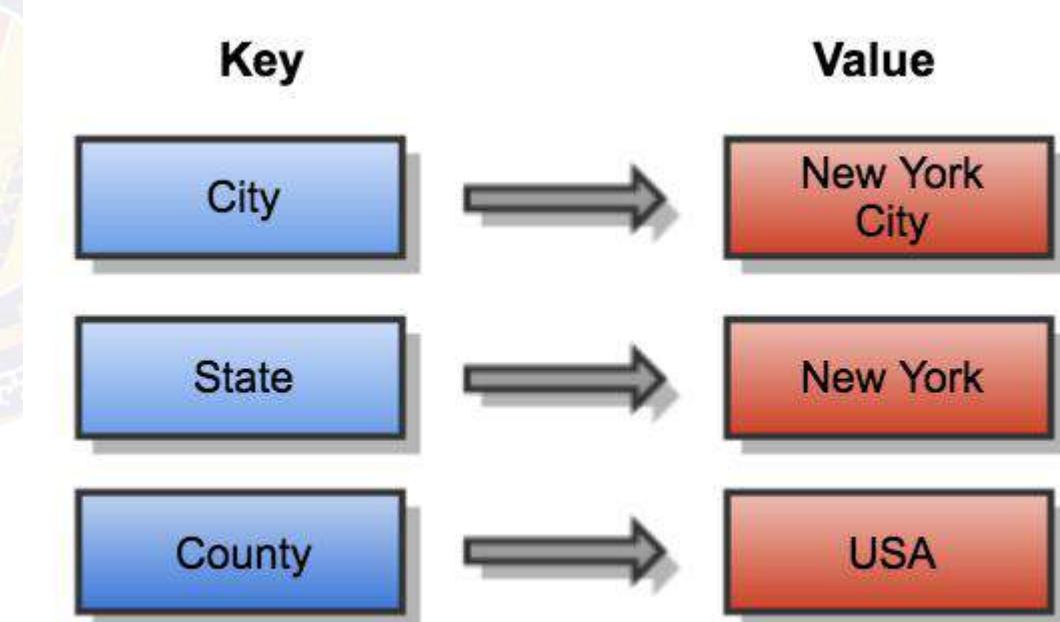


# Categories (3)

## Key-value stores

- Simple data model based on fast access by the key to the value associated with the key
- Value can be a record or object or document or even complex data structure
- Maintains a big hash table of keys and values
- For example,
  - ✓ Dynamo, Redis, Riak

Key	Value
2014HW112220	{ Santosh,Sharma,Pilani}
2018HW123123	{Eshwar,Pillai,Hyd}



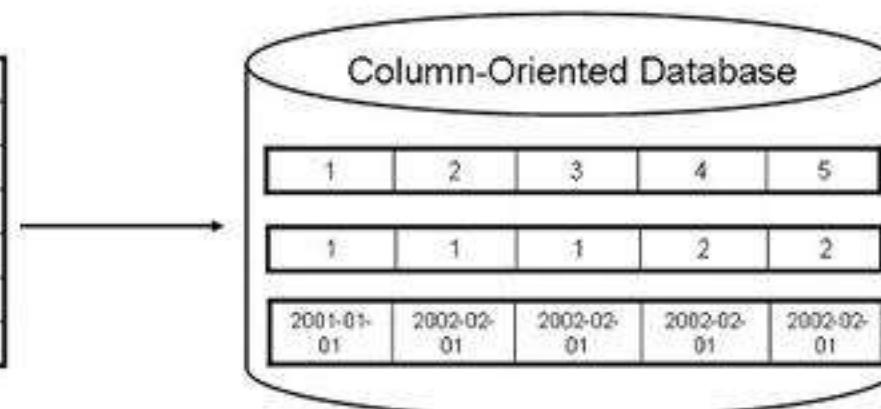
# Categories (4)

## Column based

- Partition a table by column into column families
- A part of vertical partitioning where each column family is stored in its own files
- Allows versioning of data values
- Each storage block has data from only one column
- Example,
  - ✓ Cassandra, Hbase



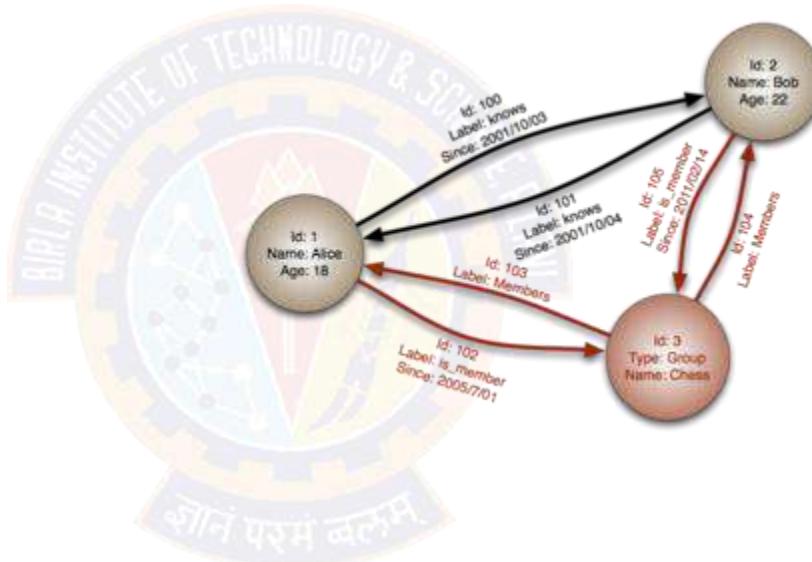
Emp_no	Dept_id	Hire_date	Emp_fn	Emp_ln
1	1	2001-01-01	Smith	Bob
2	1	2002-02-01	Jones	Jim
3	1	2002-05-01	Young	Sue
4	2	2003-02-01	Sternle	Bill
5	2	1999-06-15	Aurora	Jack
6	3	2000-08-15	Jung	Laura



# Categories (5)

## Graph Based

- Data is represented as graphs and related nodes can be found by traversing the edges using the path expression
- aka network database
- Example
  - ✓ Neo4J, HyperGraphDB





# Thank You!

In our next session: Document based NoSQL Databases



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# NoSQL Databases - Characteristics

Chandan Ravandur N

# NoSQL

- Not Only SQL
- Meant to convey that many applications need systems other than traditional relational SQL systems to manage their data management needs
- Most of them are distributed databases or distributed storage systems with focus on
  - ✓ Semi structured data storage
  - ✓ High performance
  - ✓ Availability
  - ✓ Data replication
  - ✓ Scalability

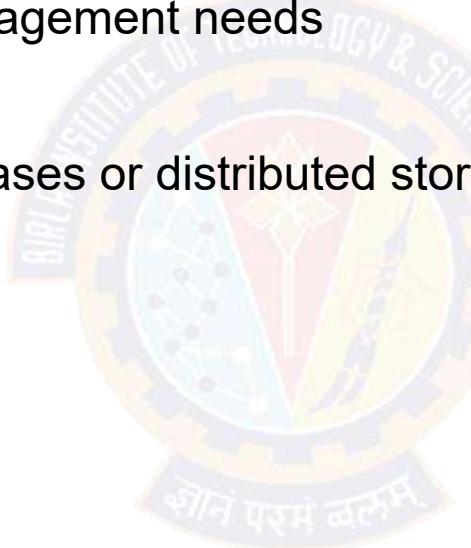
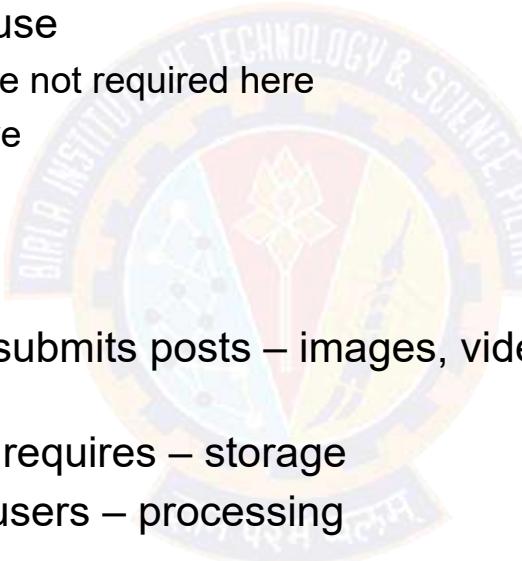


Image Source : DataVarsity

# Emergence of NoSQL systems

## Use cases

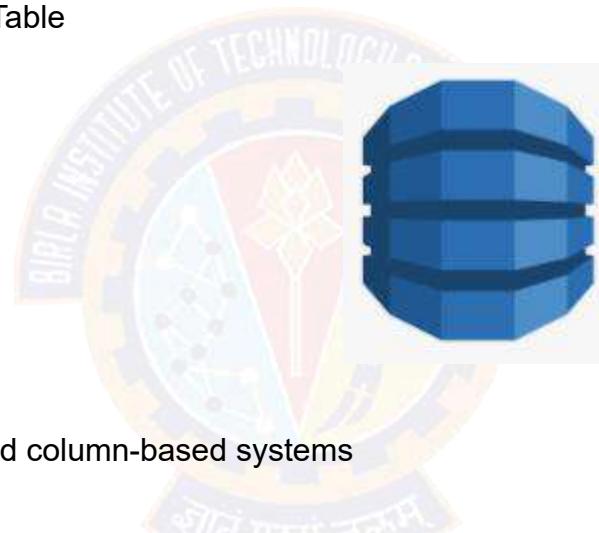
- Free email application like Gmail
  - Have millions of users, each user with thousands of emails – need for storage
  - SQL may not be appropriate because
    - ✓ It has too many services which are not required here
    - ✓ Structured data model is restrictive
- Facebook – social media platforms
  - Have millions of users, each user submits posts – images, videos, text , displayed on other users pages
  - User profiles, relationships , posts requires – storage
  - Needs to show the posts to other users – processing
  - SQL many not be appropriate because
    - ✓ Can not handle some of this data well
    - ✓ Can not handle when storage is huge and distributed across nodes



# Emergence of NoSQL systems (2)

## Soultions

- BigTable
  - ✓ Googles proprietary NoSQL systems used in many of Googles applications
  - ✓ dealing with vast amounts of data storage like Gmail, Maps, Web site indexing etc.
  - ✓ Apache Hbase is open source version of BigTable
  - ✓ Column-based or wide column stores
- DynamoDB
  - ✓ Amazons NoSQL system available on AWS
  - ✓ Innovation in key-value type data stores
- Cassandra
  - ✓ Facebooks NoSQL system
  - ✓ Uses concepts from both key-value stores and column-based systems
- MongoDB
  - ✓ Document based NoSQL system
- Neo4J
  - ✓ Graph based NoSQL systems



# Characteristics of NoSQL systems

- Related to Distributed databases and distributed systems
  - ✓ Scalability
  - ✓ Availability
  - ✓ Replication models
  - ✓ Sharding
  - ✓ High performance
- Related to data models and query languages
  - ✓ No fixed schema
  - ✓ Less powerful query languages
  - ✓ Versioning



# Characteristics of NoSQL systems (2)

## Related to Distributed databases and distributed systems

- Scalability
  - ✓ Horizontal or Vertical
  - ✓ In NoSQL systems, horizontal scalability is used by adding more nodes for data storage and processing as the volume of data grows
  - ✓ Used when system is operational, so techniques for distributing the existing data among new nodes without interrupting the system operation is of upmost importance
- Availability
  - ✓ NoSQL systems requires continuous system availability
  - ✓ Data is replicated over two or more nodes in transparent manner so that if one node fails, data is still available on the other nodes
  - ✓ Replication improves the data availability and performance as the reads can be answered from replicas as well
  - ✓ Write performance become cumbersome as updated must be applied to every copy

# Characteristics of NoSQL systems (3)

## Related to Distributed databases and distributed systems

- Replication models
- Master-slave or Master – master
- Master – slave configuration
  - ✓ One copy as master, others as slaves
  - ✓ Changes are first applied to master and then propagated to slaves, using eventual consistency
  - ✓ Reads can be done in two ways
  - ✓ Read from master – always latest data returned
  - ✓ Read from slaves – no guarantee of latest data as its based on eventual consistency
- Master-Master configuration
  - ✓ Allows read and write at any of replicas but may not guarantee that reads at nodes that store different copies see the same values
  - ✓ Different users may write same data item concurrently at different nodes of system, so values of item will be temporarily inconsistent

# Characteristics of NoSQL systems (4)

## Related to Distributed databases and distributed systems

- Sharding of files
  - ✓ Files (collections of data objects) have millions of records which are accessed simultaneously
  - ✓ Not practical to store the file at one node
  - ✓ Sharding – horizontal partitioning is used to distribute loads of accessing the files records to multiple nodes
  - ✓ Combination of sharding and replication improve load balancing and data availability
- High Performance Data Access
  - ✓ Finding a data record is important in many NoSQL systems
  - ✓ Either uses hashing or range partitioning of keys
  - ✓ In hashing, hash function  $h(K)$  is applied to key  $K$  and location of object with key  $K$  is determined by value of  $h(K)$
  - ✓ In range partitioning, location is determined by range of key values, location  $i$  would hold objects whose key values  $K$  are in range  $K_{\min} \leq K \leq K_{\max}$ .

# Characteristics of NoSQL systems

## Related to Data models and query language

- Not requiring Schema
  - ✓ Allowed by semi structured, self describing data
  - ✓ Not required to have a schema in most of NoSQL systems
  - ✓ There may not be a schema to specify constraints, any constraints on the data would have to be programmed in applications
  - ✓ Languages like JSON, XML are used while defining models
- Low powerful query languages
  - ✓ May applications not require powerful query languages as SQL as read queries in these systems often locate single objects in a single file based on their object keys
  - ✓ NoSQL systems provide a set of functions and APIs
  - ✓ Supports SCRUD operations – Search , CRUD
  - ✓ Many systems do not provide join operations as part of language, hence needs to be implemented in application
- Versioning
  - ✓ Some NoSQL systems provides storage of multiple versions of data items with timestamps

Reference :  
Fundamentals of Database Systems, by Elmasri, Navathe



# Thank You!

In our next session: NoSQL Databases - Categories



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Document Oriented Databases

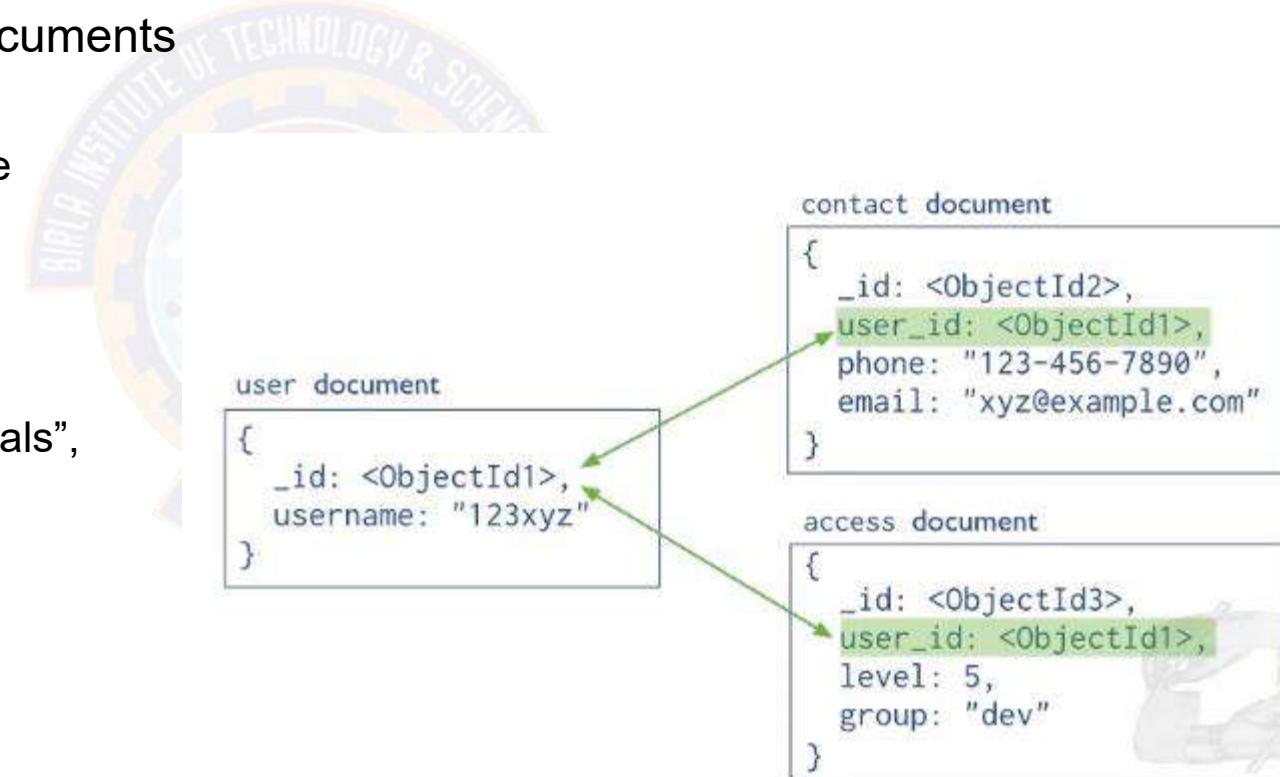
Chandan Ravandur N

# Document based Databases

## Document based

- Store data in form of documents using well known formats like JSON
- Documents accessible via their id, but can be accessed through other index as well
- Maintains data in collections of documents
- Example,
  - MongoDB, CouchDB, CouchBase
- Book document :

```
{  "Book Title" : "Database Fundamentals",  "Publisher" : "My Publisher",  "Year of Publication" : "2020"}
```



# What?

## MongoDB is

- NoSQL
- Cross-platform
- Distributed
- Document-oriented
- Open source

Datastore



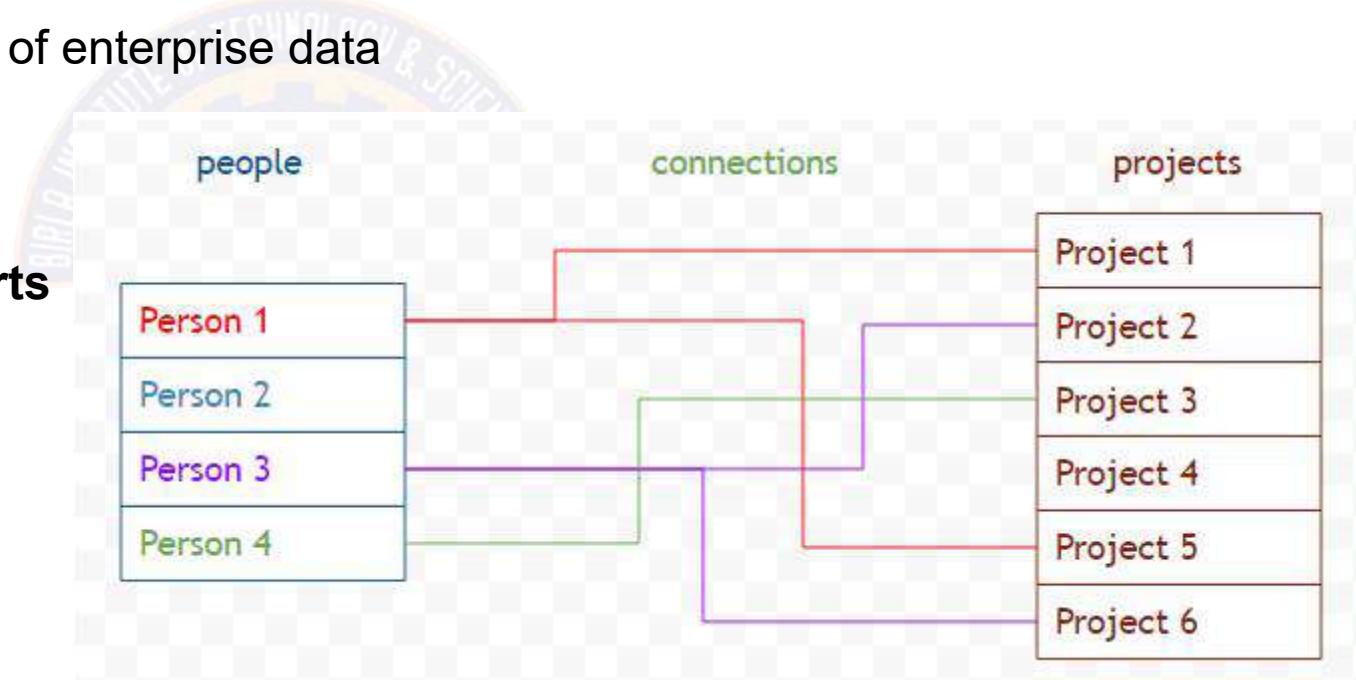
[Image Source : MongoDB](#)

# Why?

## Challenges with Relational Databases

- Not able to cope with high data volume
- Limited capabilities to deal with unstructured data
- Restrictions on the scalable needs of enterprise data

- **Require a Data store that supports**
  - ✓ Horizontal scaling
  - ✓ Flexible schema
  - ✓ Partitioning



[Image source : GoogleSite](#)

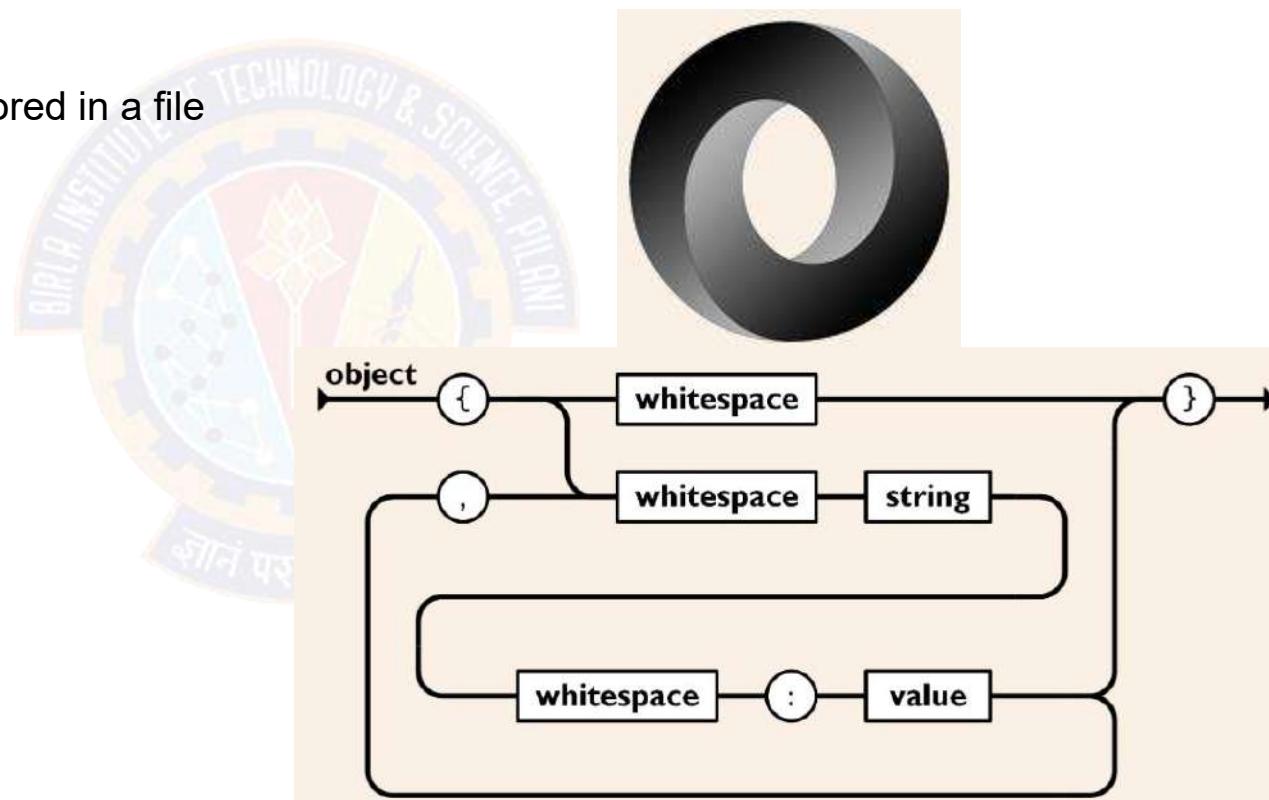
# Why? (2)



# MongoDB and JSON

- MongoDB uses BSON (Binary JSON) – Open standard to store complex data structures
- Example
- Assume following Employee Record is stored in a file
  - 123, Rohit Kumar , 8123561290
  - 124, Naresh Pande , 8904561239
- Corresponding JSON representation

```
{  
    Empld : 123,  
    EmpName : Rohit Kumar,  
    ContactNo : 8123561290  
}  
  
{  
    Empld : 124,  
    EmpName : Naresh Pande ,  
    ContactNo : 8904561239  
}
```

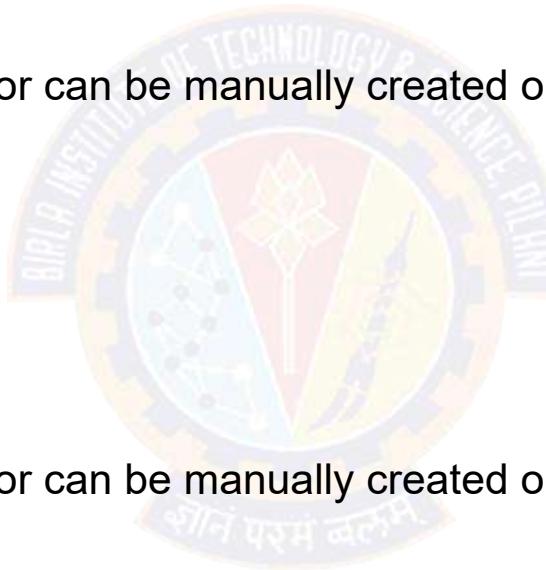


[Image Source : Json.org](http://Json.org)

# MongoDB concepts

## Database, Collection and Document

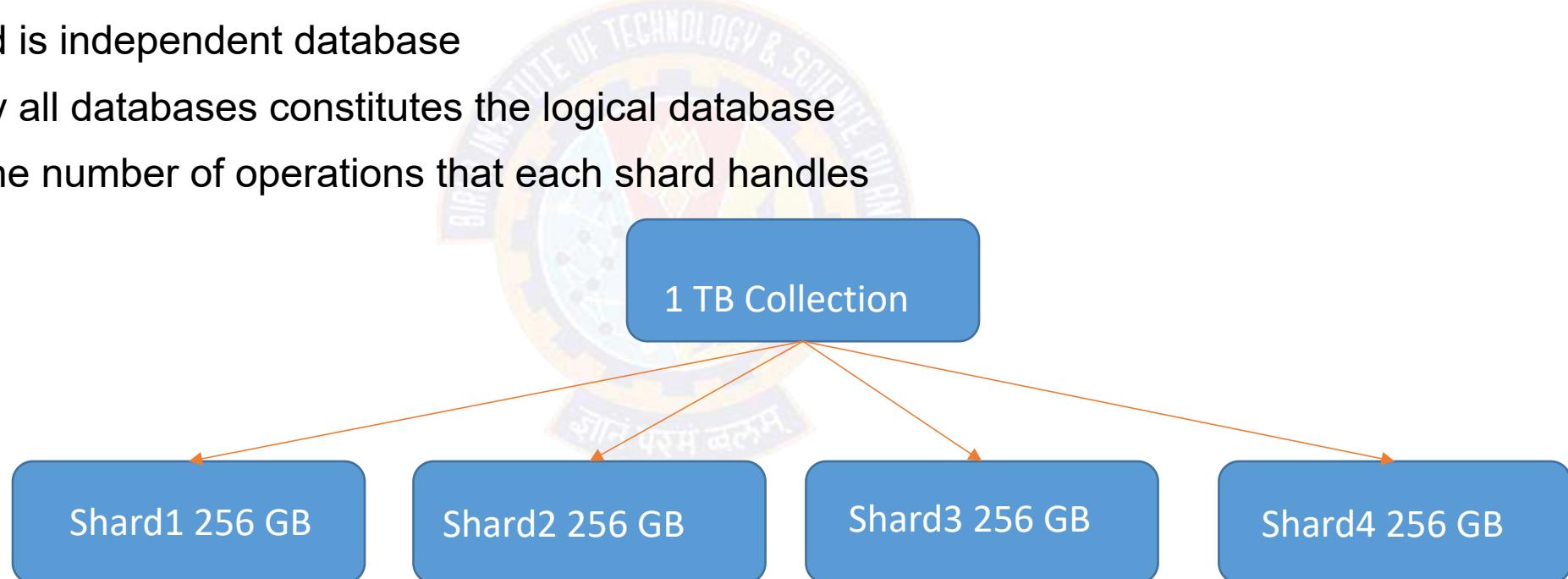
- Database
  - ✓ Collection of collections
  - ✓ Container for collections
  - ✓ Gets created when its referenced or can be manually created on demand
- Collection
  - ✓ Similar to table in RDBMS
  - ✓ Holds multiple documents within it
  - ✓ No enforcement of Schema
  - ✓ Gets created when its referenced or can be manually created on demand
- Document
  - ✓ Similar to row in RDBMS table
  - ✓ Has a unique identifier
  - ✓ Supports dynamic schema



# MongoDB Features

## Auto Sharding

- Similar to Horizontal scaling
- Dataset is divided and distributed over multiple nodes or shards
- Each shard is independent database
- Collectively all databases constitutes the logical database
- Reduces the number of operations that each shard handles



# MongoDB Features(2)

## Rich Query Language

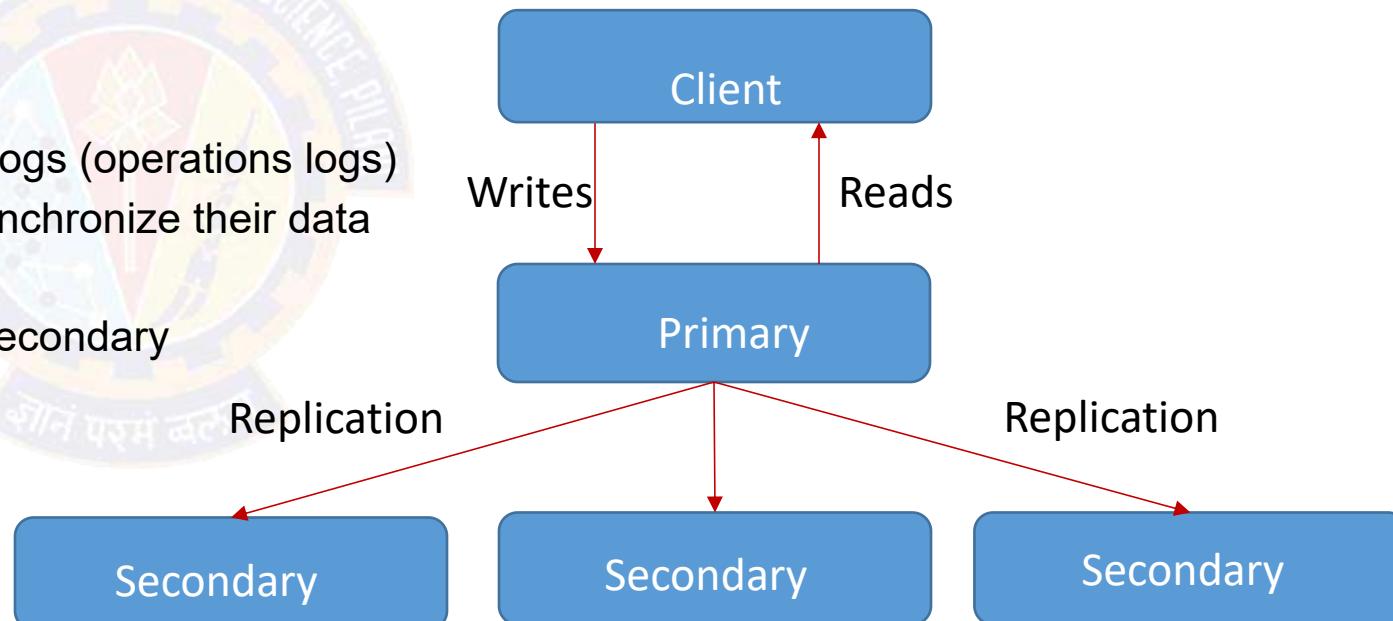
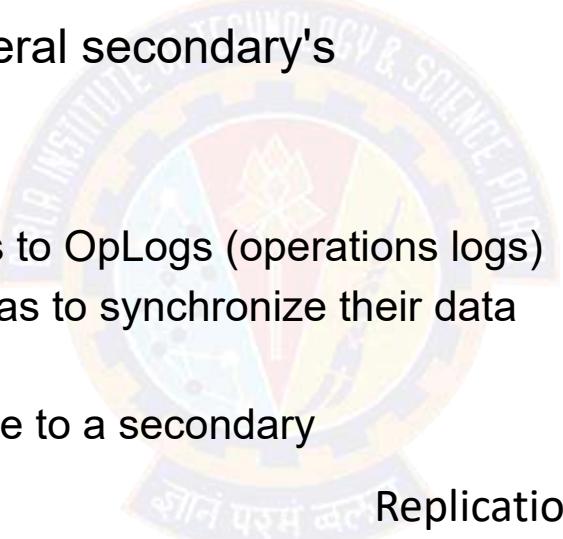
- For read and write operations (CRUD)
- Data Aggregation
- Text Search and Geospatial Queries



# MongoDB Features(3)

## Replication

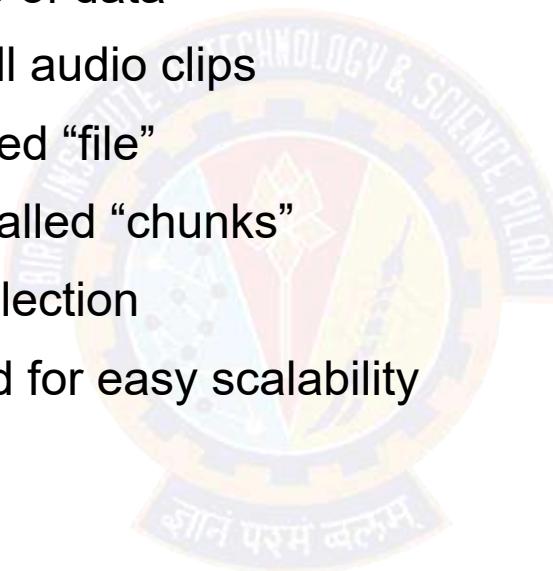
- Supports data redundancy and high availability
- Helps to recover from hardware failure and service interruptions
- Replica has single primary and several secondary's
- Working
  - ✓ Write is directed to primary
  - ✓ Primary then logs all write requests to OpLogs (operations logs)
  - ✓ OpLog is used by secondary replicas to synchronize their data
  - ✓ Clients read from primary
  - ✓ Client can specify a read preference to a secondary



# MongoDB Features(4)

## Scalability

- Stores data in binary format (BSON)
- Provides GridFS to support storage of data
- Can easily store photographs, small audio clips
- Stores metadata in a collection called “file”
- Breaks the data into small pieces called “chunks”
- “Chunks” are stored in “chunks” collection
- This process takes care about need for easy scalability



# MongoDB Features(5)

## In-place data updates

- Updates data in-place
- Means updates the data wherever it is available
- Does it by lazy writes
- Writes to disk once every second
- Fewer reads and writes to disk hence performance is better
- No guarantee that data will be stored safely on the disk





# Thank You!

In our next session: Columnar Databases



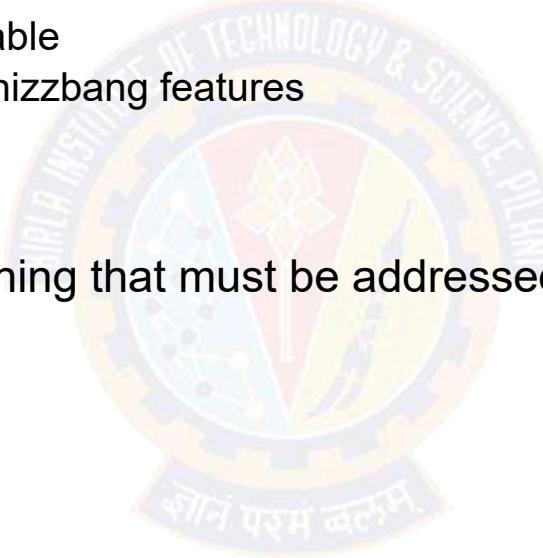
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Today's application requirements

Chandan Ravandur N

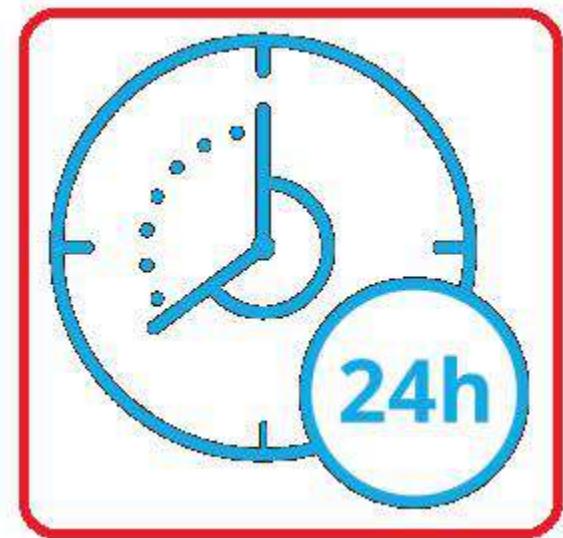
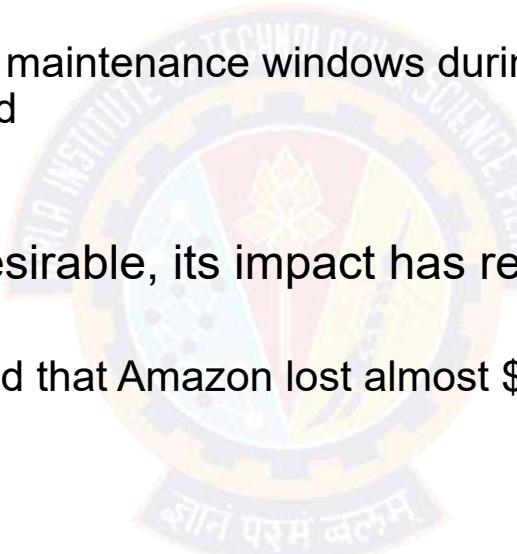
# Today's application requirements

- Digital experiences play a major part in many or most of the activities that we engage in on a daily basis
  - ✓ pushed the boundaries of expectations from the software :
  - ✓ want applications to be always available
  - ✓ be perpetually upgraded with new whizzbang features
  - ✓ provide personalized experiences
- Fulfilling these expectations is something that must be addressed right from the beginning of the idea-to-production lifecycle
- Key requirements:
  - ✓ Zero downtime
  - ✓ Shortened feedback cycles
  - ✓ Mobile and multi-device support
  - ✓ Connected devices—also known as the Internet of Things
  - ✓ Data-driven



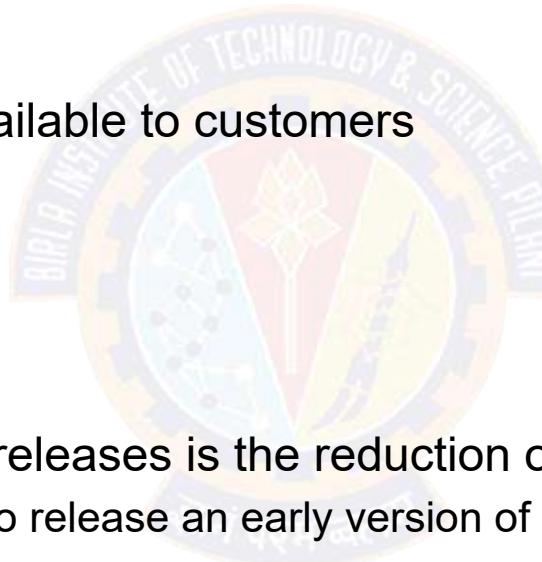
# Zero downtime

- One of the key requirements of the modern application: it must always be available
  - ✓ The world is working in "now mode"
  - ✓ Forgotten are the days when even short maintenance windows during which applications are unavailable are tolerated
- Unplanned downtime has never been desirable, its impact has reached astounding levels
  - ✓ For example, long back Forbes estimated that Amazon lost almost \$2 million during a 13-minute unplanned outage
- Downtime, planned or not, results in significant revenue loss and customer dissatisfaction
  - ✓ no more just a problem only for the operations team
  - ✓ Software developers or architects are responsible for developing zero downtime software's!



# Shortened feedback cycles

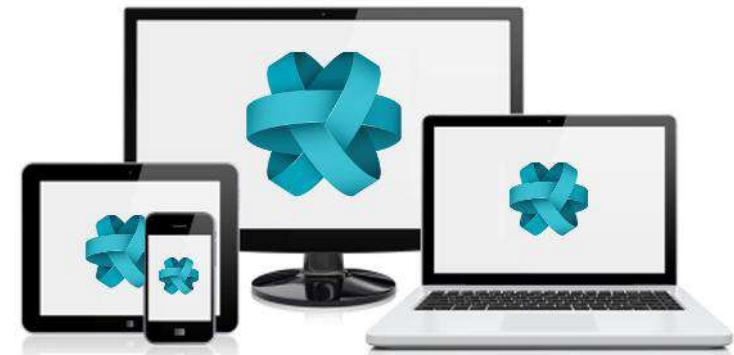
- Another critical ability is to release code frequently
  - ✓ Required by significant competition and ever-increasing consumer expectations
- Application releases are being made available to customers
  - ✓ several times a month
  - ✓ numerous times a week
  - ✓ even several times a day
- The biggest driver for these continuous releases is the reduction of risk!
  - ✓ The best way to get mitigate the risk is to release an early version of a feature and get feedback
  - ✓ Using feedback, can make adjustments or even change direction entirely
  - ✓ Frequent software releases shorten feedback loops and reduce risk!



Source : facebook

# Mobile and multi-device support

- Internet usage via mobile devices has already eclipsed that of desktop / laptop computers
- Today's applications need to support at least
  - ✓ two mobile device platforms, iOS and Android
  - ✓ as well as the desktop
- Users increasingly expect to seamlessly move from one device to another as they navigate through the day with same app experience
- For example,
  - ✓ Users may be watching a cricket match on an Android TV and then transition to viewing the program on a mobile device when they're travelling
- Designing applications the right way is essential to meeting these needs!



Source : [savoirfairelinux](#)

# Connected devices

## Internet of Things

- The internet is no longer only for connecting humans to systems
- Today, billions of devices are connected to the internet
  - ✓ Allowing devices to be monitored and even controlled by other connected entities
- The home-automation market alone is estimated to be a \$53 billion market by 2022
- The connected home has sensors and remotely controlled devices such as
  - ✓ motion detectors
  - ✓ cameras
  - ✓ smart thermostats
  - ✓ lighting systems etc.
- Other connected devices are automobiles, home appliances, farming equipment, jet engines, and the smartphone
- Internet-connected devices change the nature of the software in two fundamental ways
  - ✓ the volume of data flowing over the internet is dramatically increased
  - ✓ the computing infra must be significantly different from those of the past, to capture and process this data
- Difference in data volume and infrastructure architecture necessitates new software designs and practices.



Source : ThreatPost

# Data-driven

- Volumes of data are increasing
- Sources of data are becoming more widely distributed
  - ✓ These factors are affecting the large, centralized, shared database making them unusable
- Instead of the single, shared database, application requirements
  - ✓ call for a network of smaller, localized databases,
  - ✓ software that manages data relationships across that collection of data management systems
- New approaches drive the need for software development and management agility all the way through to the data tier.
- All of the newly available data is wasted if it goes unused
  - ✓ Today's applications must increasingly use data to provide greater value to the customer through smarter applications



Source : Dataversity

Reference:  
Cloud Native Patterns by Cornelia Davis



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Modern Architectures

Chandan Ravandur N

# Approach

- The cloud is changing how applications are designed and secured
  - Instead of monoliths, applications are decomposed into smaller, decentralized services
  - These services communicate through APIs or by using asynchronous messaging or eventing
  - Applications scale horizontally, adding new instances as demand requires
- Requires a structured approach for designing applications that are
  - Scalable
  - Secure
  - Resilient
  - and highly available
- These trends bring new challenges!



source : softeng

# New challenges

- Application state is distributed
- Operations are done in parallel and asynchronously
- Applications must be resilient when failures occur
- Malicious actors continuously target applications
- Deployments must be automated and predictable
- Monitoring and telemetry are critical for gaining insight into the system



Source : prismatic

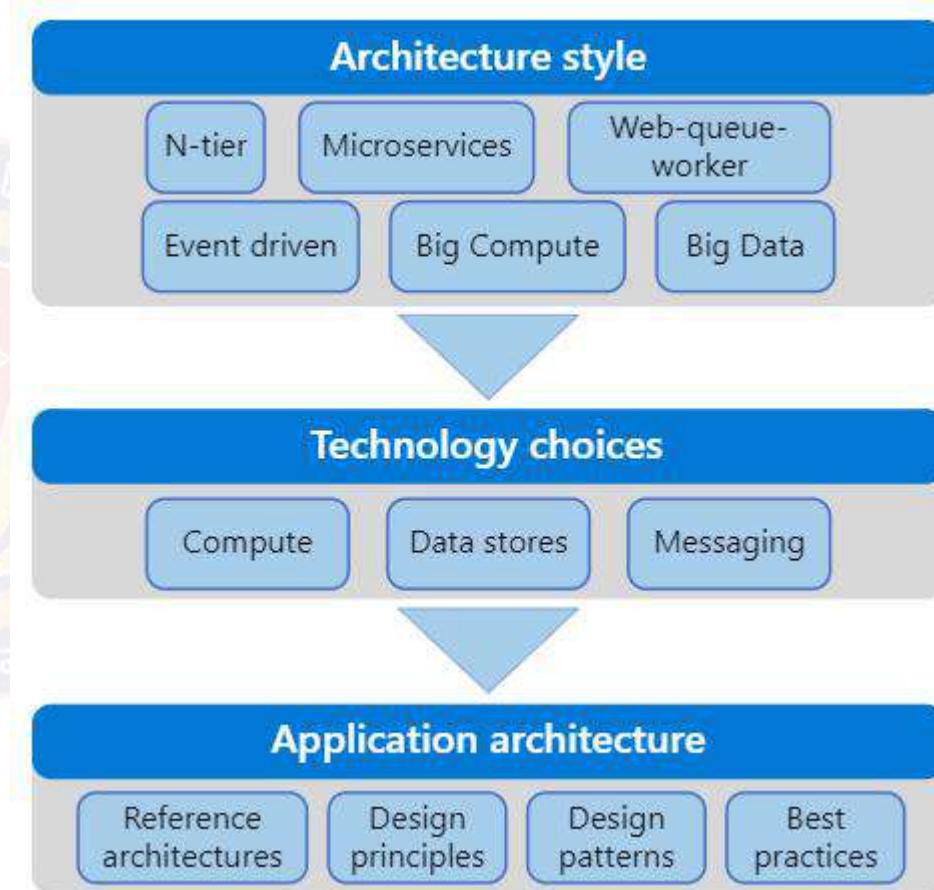
# Compared

<b>Traditional on-premises</b>	<b>Modern cloud</b>
Monolithic	Decomposed
Designed for predictable scalability	Designed for elastic scale
Relational database	Polyglot persistence (mix of storage technologies)
Synchronized processing	Asynchronous processing
Design to avoid failures (MTBF)	Design for failure (MTTR)
Occasional large updates	Frequent small updates
Manual management	Automated self-management
Snowflake servers	Immutable infrastructure

source : Microsoft

# Structured Approach Needed

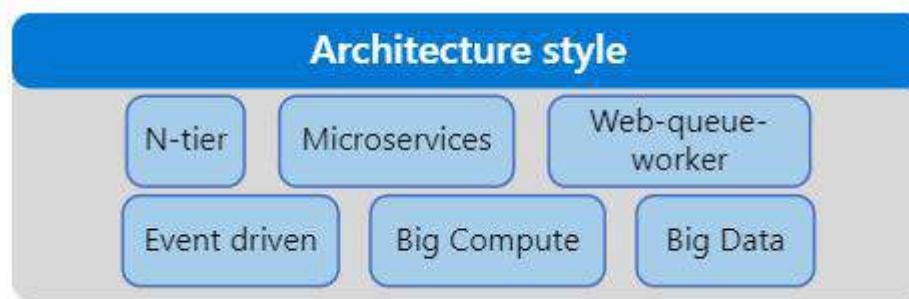
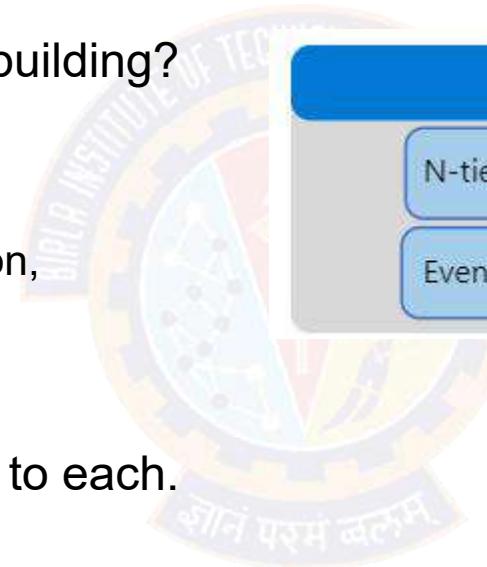
- Series of steps, from the architecture and design to implementation are involved
- For each step, there is supporting guidance needed to design application architecture



source : Microsoft

# Architecture styles

- The first decision point is the most fundamental
- What kind of architecture are you building?
- It might be
  - ✓ a microservices architecture,
  - ✓ a more traditional N-tier application,
  - ✓ or a big data solution
- There are benefits and challenges to each.

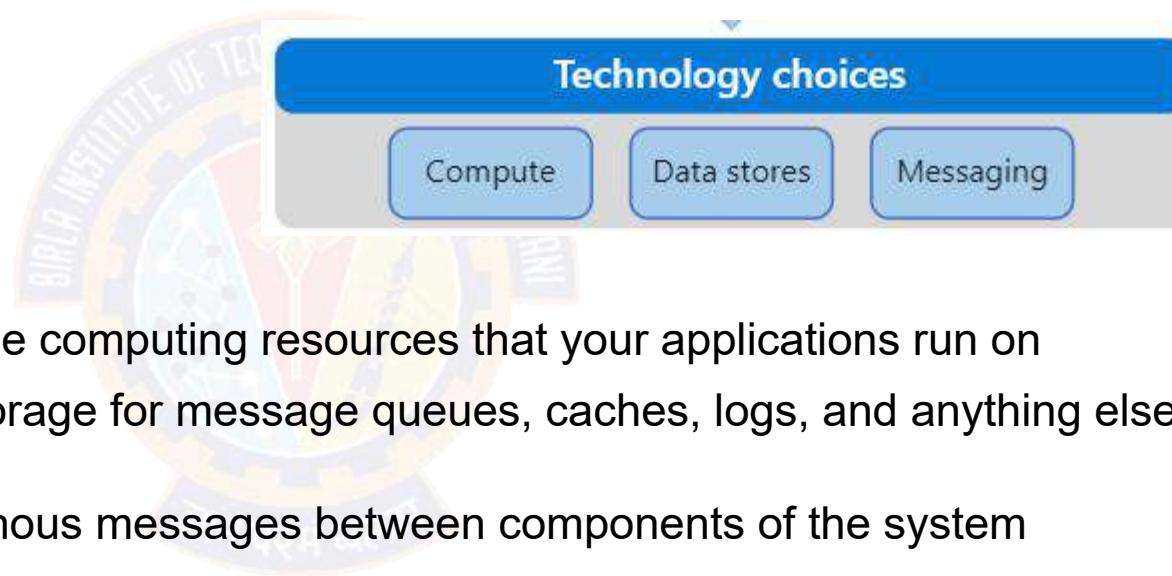


# Technology choices

- Now you can start to choose the main technology pieces for the architecture

- Critical Technology choices are:

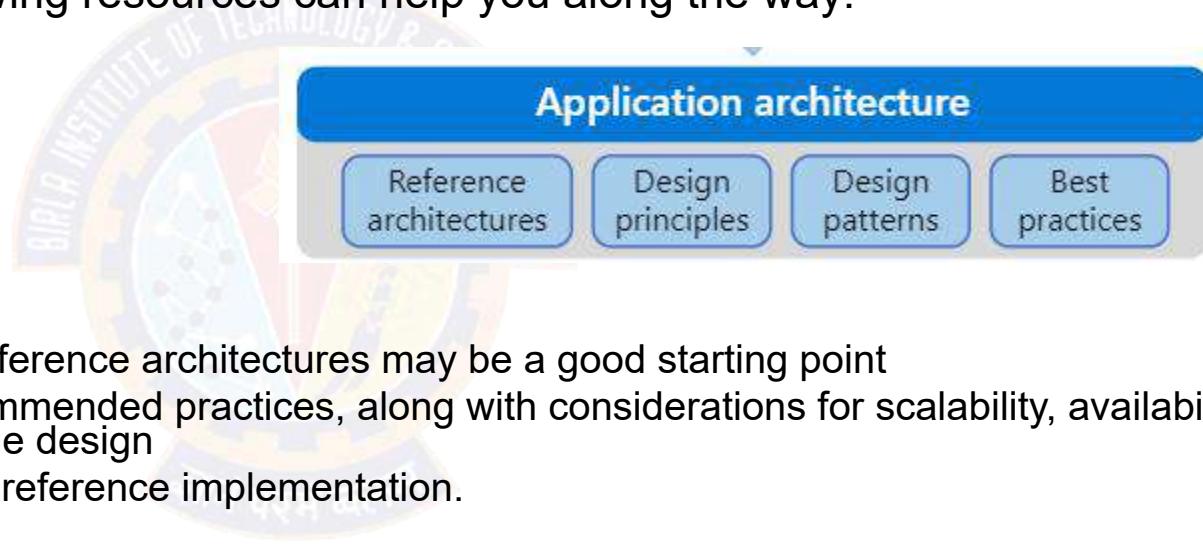
- ✓ Compute
- ✓ Data stores
- ✓ Messaging



- Compute refers to the hosting model for the computing resources that your applications run on
- Data stores include databases but also storage for message queues, caches, logs, and anything else that an application might persist to storage
- Messaging technologies enable asynchronous messages between components of the system
- You will probably have to make additional technology choices along the way, but these three elements (compute, data, and messaging) are central to most cloud applications and will determine many aspects of your design.

# Application Architecture

- Ready to tackle the specific design of your application
- Every application is different, but the following resources can help you along the way:
  - ✓ Reference architectures
  - ✓ Design principles
  - ✓ Design patterns
  - ✓ Best practices
- Reference architectures
  - ✓ Depending on your scenario, one of our reference architectures may be a good starting point
  - ✓ Each reference architecture includes recommended practices, along with considerations for scalability, availability, security, resilience, and other aspects of the design
  - ✓ Most also include a deployable solution or reference implementation.
- Design patterns
  - ✓ Software design patterns are repeatable patterns that are proven to solve specific problems
  - ✓ They address aspects such as availability, high availability, operational excellence, resiliency, performance, and security



Reference:  
Azure Application Architecture Guide



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Architectural Styles Overview

Chandan Ravandur N

# An architecture style

- A family of architectures that share certain characteristics
- For example,
  - ✓ N-tier is a common architecture style.
  - ✓ More recently, microservice architectures have started to gain favor
- Architecture styles don't require the use of particular technologies, but some technologies are well-suited for certain architectures
  - ✓ For example, containers are a natural fit for microservices



source : pintrest

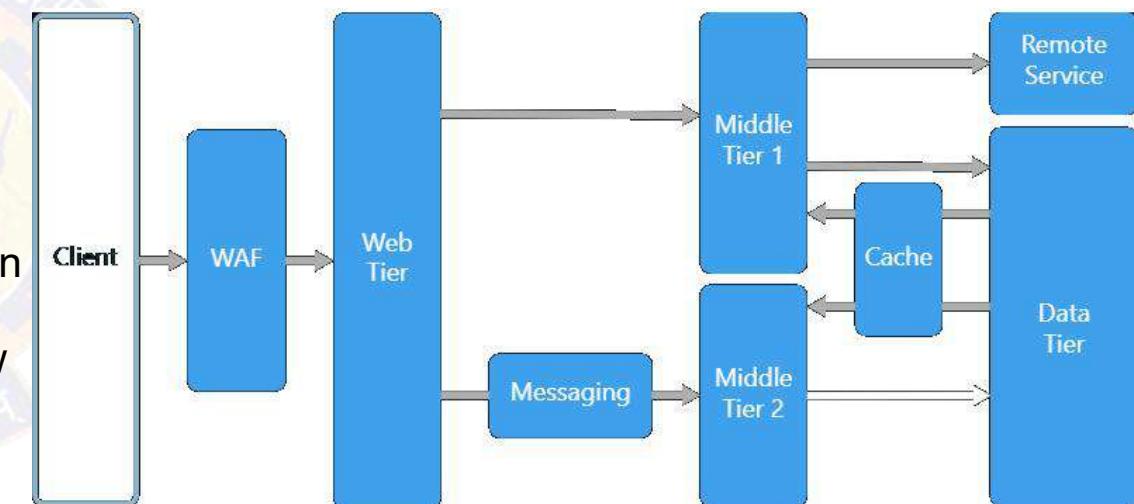
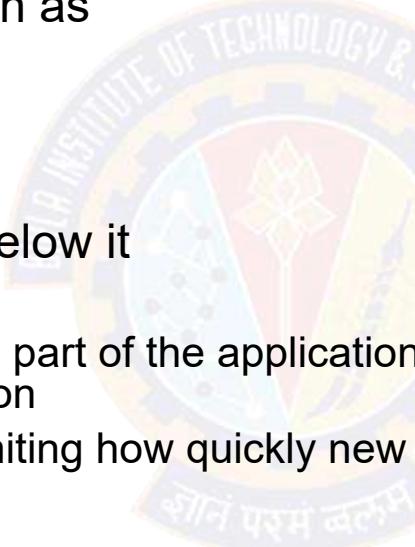
# A quick tour of the styles

- A set of architecture styles that are commonly found in cloud applications
- N-tier
- Web-Queue-Worker
- Microservices
- Event-driven architecture
- Big Data, Big Compute



# N-tier

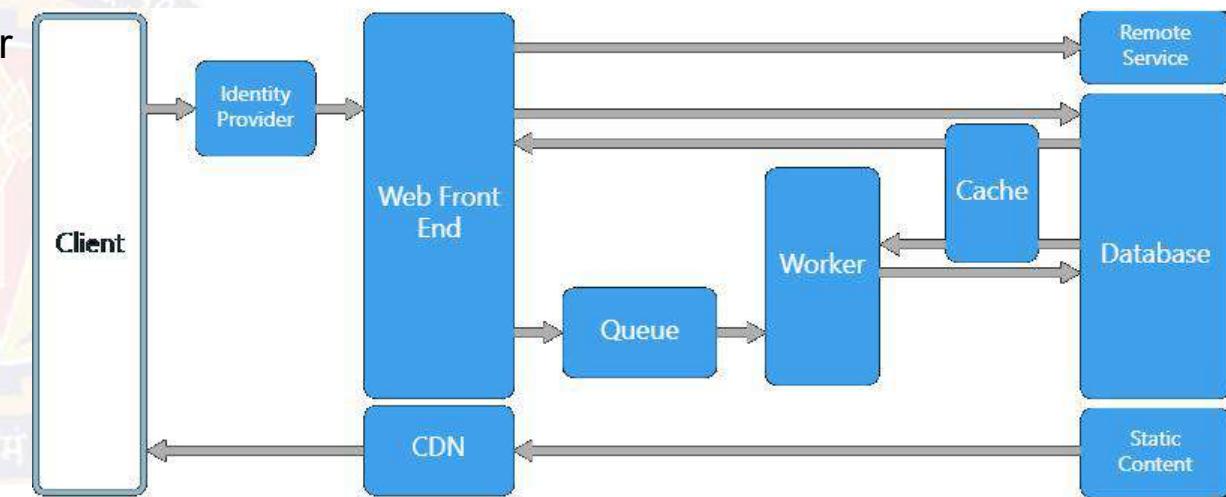
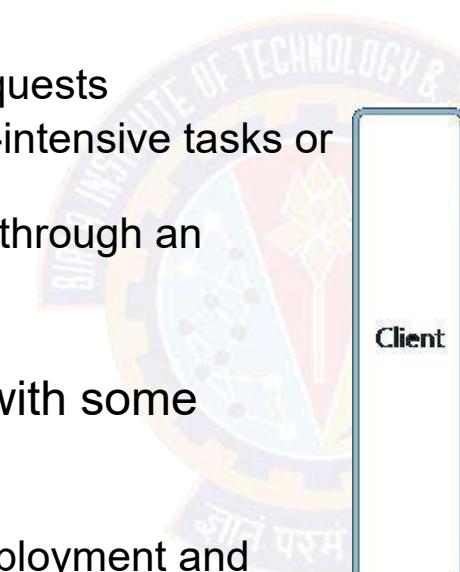
- A traditional architecture for enterprise applications
- Dependencies are managed by dividing the application into layers that perform logical functions, such as
  - ✓ presentation,
  - ✓ business logic,
  - ✓ and data access
- A layer can only call into layers that sit below it
  - ✓ this horizontal layering can be a liability
  - ✓ can be hard to introduce changes in one part of the application without touching the rest of the application
  - ✓ makes frequent updates a challenge, limiting how quickly new features can be added
- Is a natural fit for migrating existing applications that already use a layered architecture
  - ✓ For that reason, often seen in
  - ✓ infrastructure as a service (IaaS) solutions,
  - ✓ or application that use a mix of IaaS and managed services



source : Microsoft

# Web-Queue-Worker

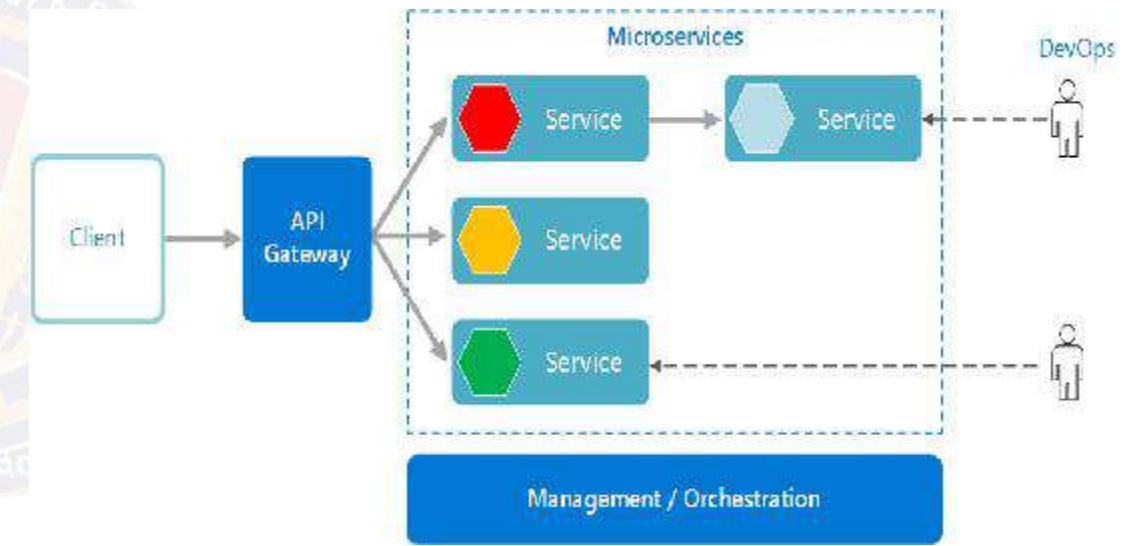
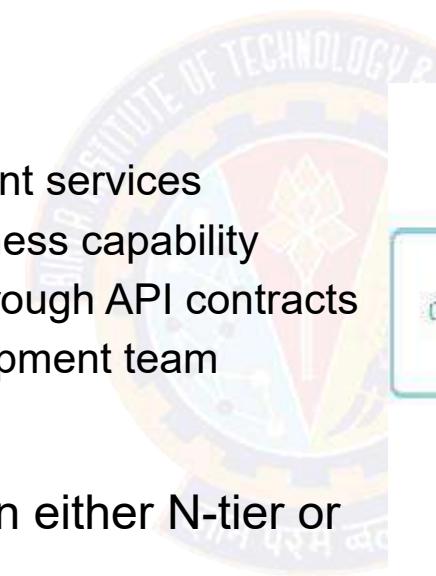
- For a purely PaaS solution, consider a Web-Queue-Worker architecture
- Application has
  - ✓ a web front end that handles HTTP requests
  - ✓ a back-end worker that performs CPU-intensive tasks or long-running operations
  - ✓ front end communicates to the worker through an asynchronous message queue
- Suitable for relatively simple domains with some resource-intensive tasks
  - ✓ easy to understand
  - ✓ use of managed services simplifies deployment and operations
- But with complex domains, it can be hard to manage dependencies
  - ✓ front end and the worker can easily become large, monolithic components that are hard to maintain and update



source : Microsoft

# Microservices

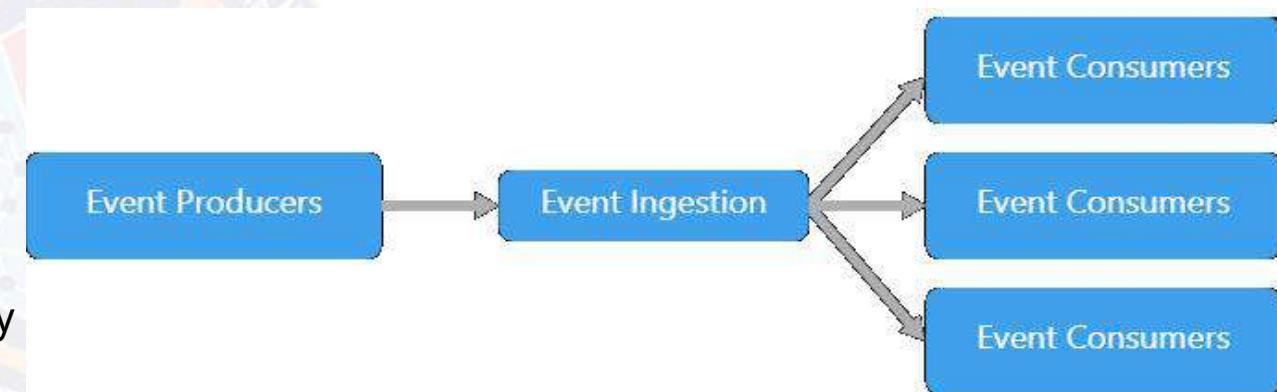
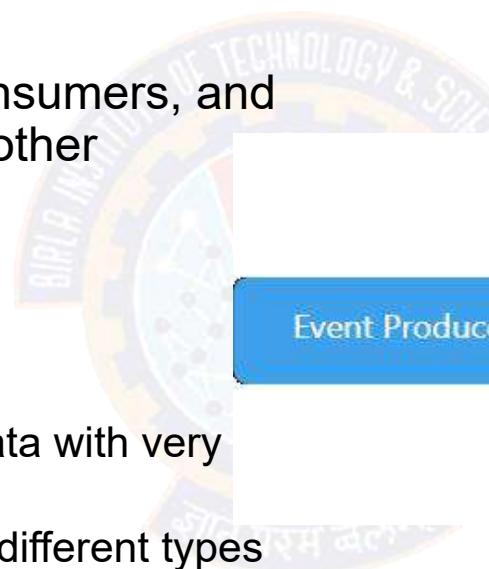
- If your application has a more complex domain, consider moving to a Microservices architecture
- Microservice Application
  - ✓ is composed of many small, independent services
  - ✓ each service implements a single business capability
  - ✓ are loosely coupled, communicating through API contracts
  - ✓ can be built by a small, focused development team
- More complex to build and manage than either N-tier or web-queue-worker
  - ✓ requires a mature development and DevOps culture
  - ✓ can lead to higher release velocity, faster innovation, and a more resilient architecture



source : Microsoft

# Event-driven architecture

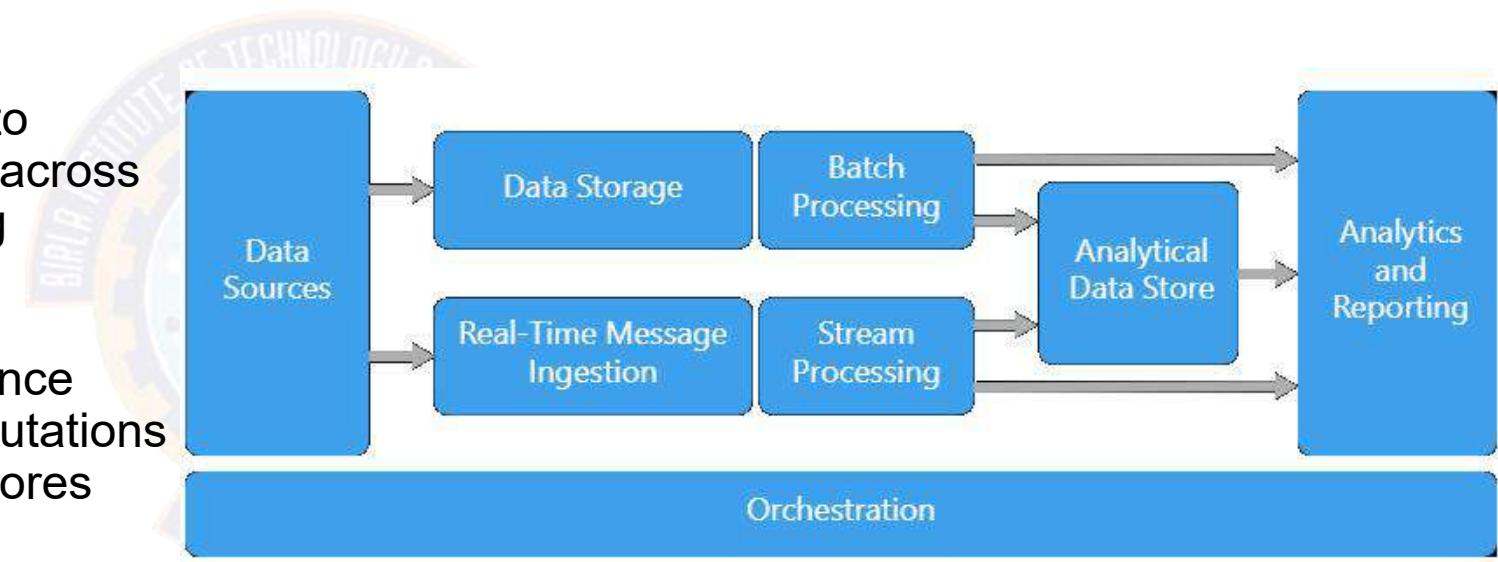
- Use a publish-subscribe (pub-sub) model, where producers publish events, and consumers subscribe to them
- Producers are independent from the consumers, and consumers are independent from each other
- Useful for applications that
  - ✓ ingest and process a large volume of data with very low latency, such as IoT solutions
  - ✓ has different subsystems must perform different types of processing on the same event data



source : Microsoft

# Big Data, Big Compute

- Specialized architecture styles for workloads that fit certain specific profiles
- Big data divides a very large dataset into chunks, performing parallel processing across the entire set, for analysis and reporting
- Big compute, also called high-performance computing (HPC), makes parallel computations across a large number (thousands) of cores
- Domains include simulations, modeling, and 3-D rendering.



source : Microsoft

Reference:  
Azure Application Architecture Guide



# Thank You!

In our next session:

**Slides contribution from prof  
Pravin Y Pawar**

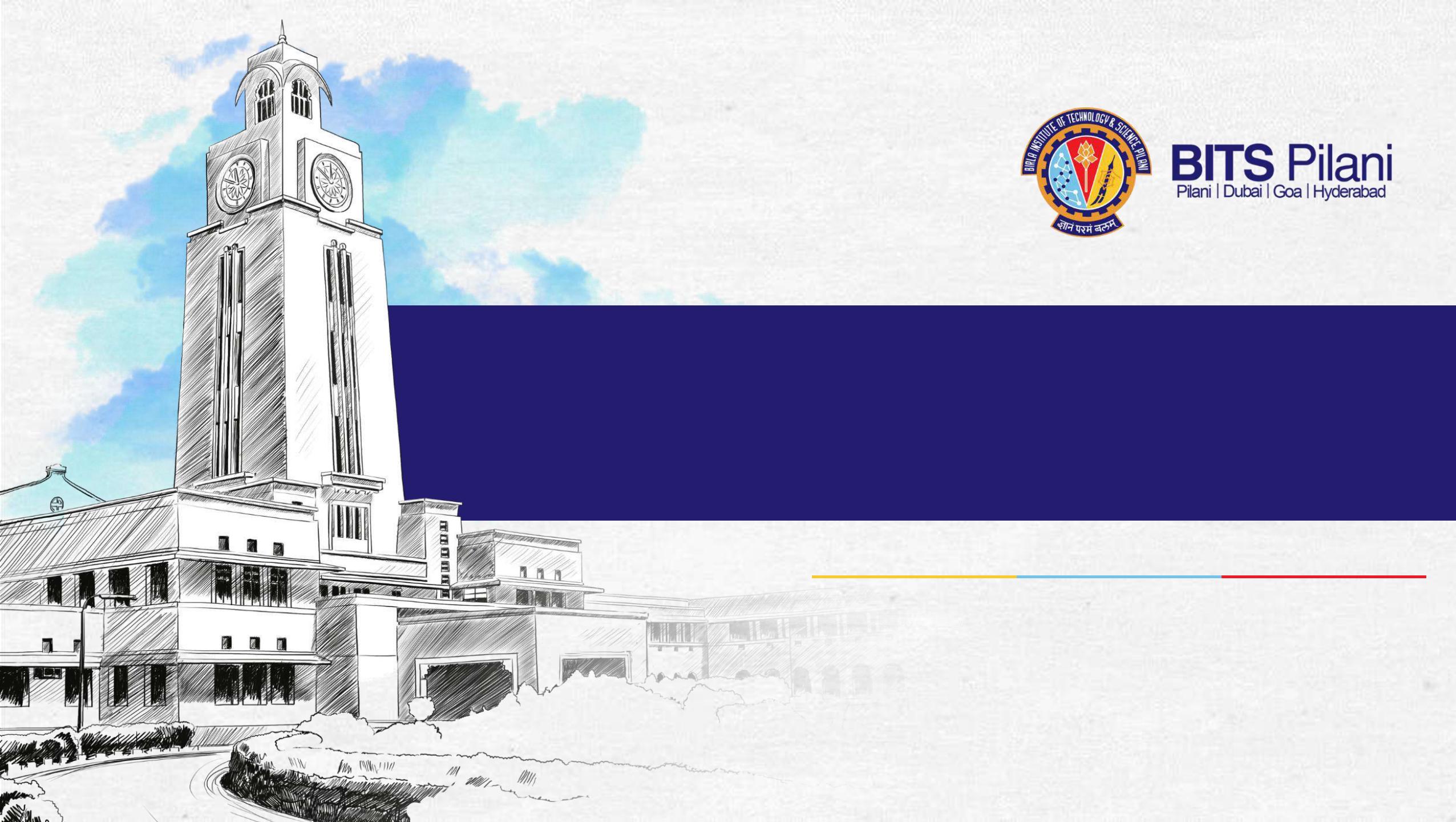




**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Monolithic Architecture

Chandan Ravandur N



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Scenario

- Lets say you are developing a server-side enterprise application
  - ✓ Must support a variety of different clients including desktop browsers, mobile browsers and native mobile applications
  - ✓ Might also expose an API for 3rd parties to consume
  - ✓ Might also integrate with other applications via either web services or a message broker
- The application handles
  - ✓ requests (HTTP requests and messages) by executing business logic
  - ✓ accessing a database
  - ✓ exchanging messages with other systems
  - ✓ and returning a HTML/JSON/XML response
- There are logical components corresponding to different functional areas of the application.

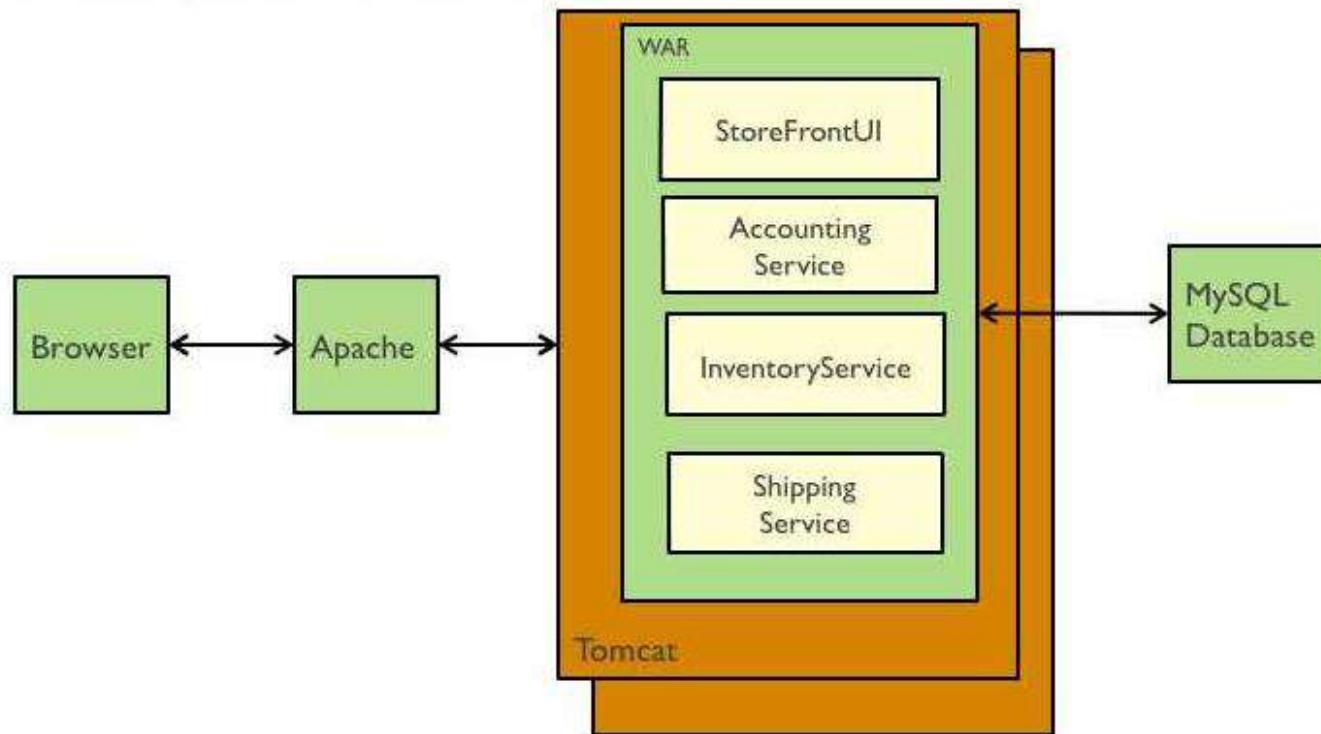
# Problem (Forces) and Solution

What's the application's deployment architecture?

- Forces
  - ✓ A team of developers working on the application
  - ✓ New team members must quickly become productive
  - ✓ Must be easy to understand and modify
  - ✓ Want to practice continuous deployment of the application
  - ✓ Must run multiple instances of the application on multiple machines in order to satisfy scalability and availability requirements
  - ✓ Want to take advantage of emerging technologies (frameworks, programming languages, etc)
- Solution
- Build an application with a monolithic architecture.
- For example:
  - ✓ a single Java WAR file.
  - ✓ a single directory hierarchy of Rails
  - ✓ NodeJS code

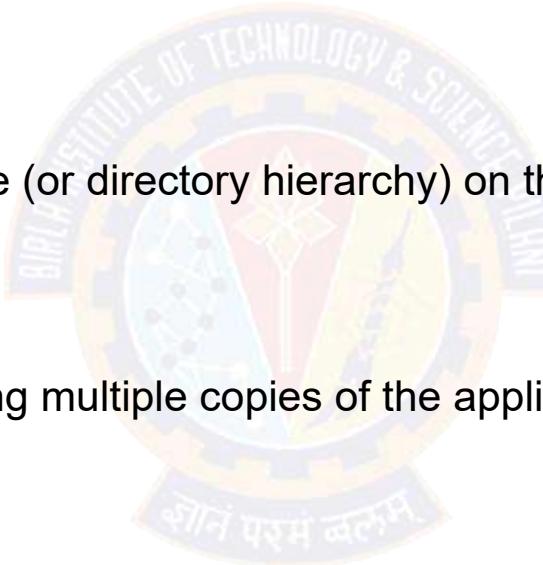
# Example

- Building an e-commerce application that takes orders from customers, verifies inventory and available credit, and ships them
- The application consists of several components including
  - ✓ the StoreFrontUI, which implements the user interface
  - ✓ backend services for checking credit, maintaining inventory and shipping orders



# Resulting context

- Simple to develop
  - ✓ the goal of current development tools and IDEs is to support the development of monolithic applications
- Simple to deploy
  - ✓ simply need to deploy the WAR file (or directory hierarchy) on the appropriate runtime
- Simple to scale
  - ✓ can scale the application by running multiple copies of the application behind a load balancer



References:  
[microservices.io](https://microservices.io)



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Types of Monoliths

Chandan Ravandur N

# Monolith

## The unit of deployment

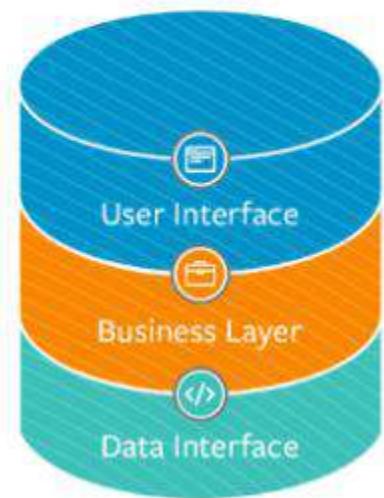
- When all the functionality in a system had to be deployed together
- Three types
  - ✓ Single process monolith
  - ✓ Distributed monolith
  - ✓ Third party black-box systems



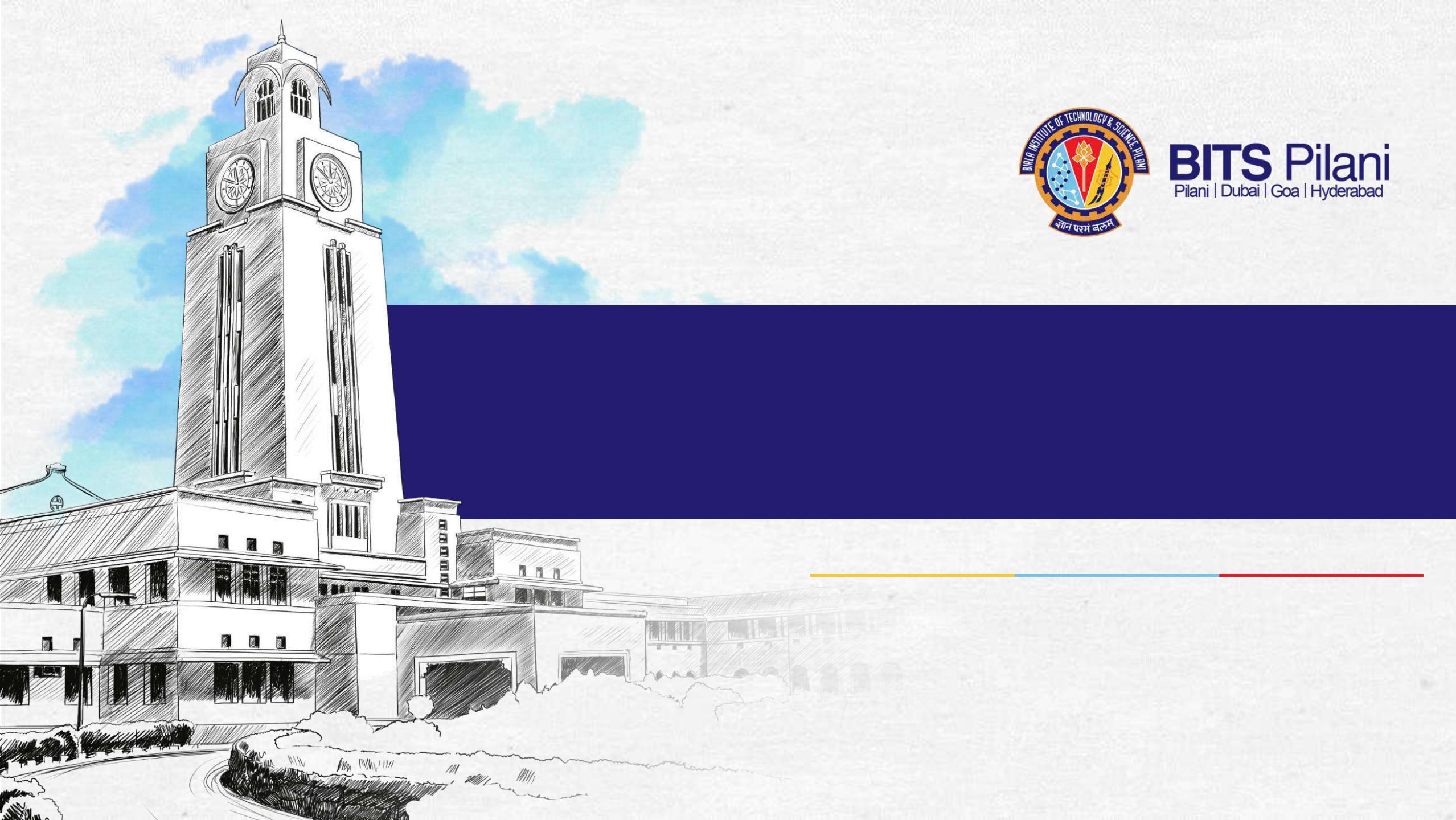
# Single Process Monolith

- Most common example is system where all of the code is deployed as a single process
  - ✓ May have multiple instances of this process for robustness or scaling
- Fundamentally all code is wrapped into one process
- In reality, these are simple distributed system
  - ✓ They nearly end up reading data from or storing data into database

Monolithic Architecture



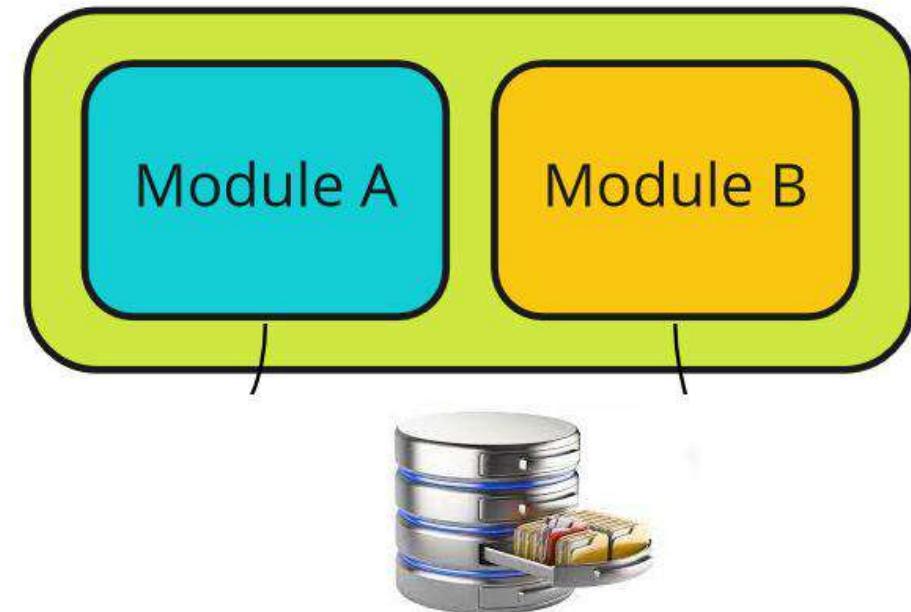
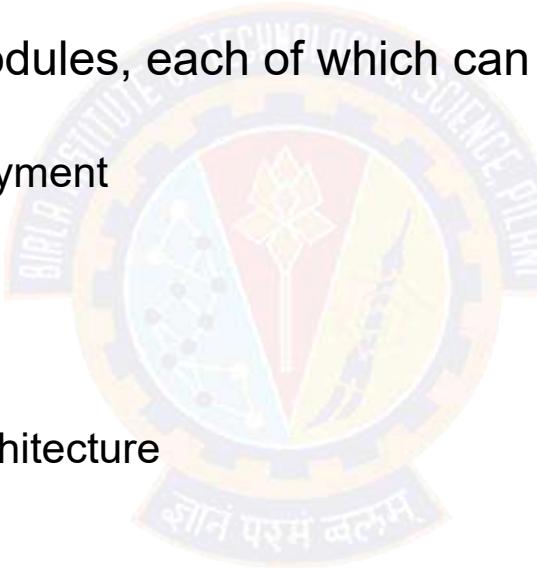
[source](#)



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

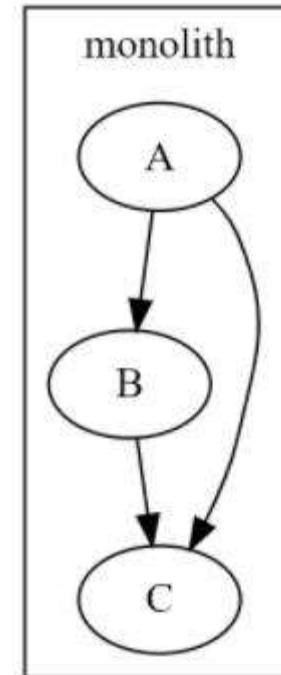
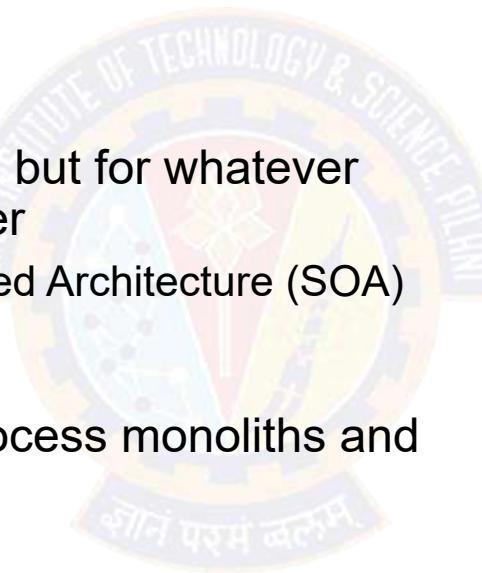
# Modular monolith

- Subset of single process monolith
- Single process consists of separate modules, each of which can be worked on independently
  - ✓ But still need to be combined for deployment
- If module boundaries are well defined
  - ✓ allows high degree of parallel working
  - ✓ Sidesteps challenges of distributed architecture
  - ✓ Provides much simpler deployments



# Distributed monolith

- A distributed system is one in which failure of computer you didn't even know existed can render your own computer unusable. – Leslie Lamport
- System consisting of multiple services but for whatever reasons needs to be deployed together
  - ✓ May meet definition of Service Oriented Architecture (SOA)
- Have disadvantages of both single process monoliths and distributed systems



# Third party black-box systems

- Third party systems like payroll systems, CRM systems, HR systems
- Common factors
  - ✓ Software developed by other people
  - ✓ Don't have control on the change of code or deployment
- Examples
  - ✓ Can be off-the-shelf software deployed on own infrastructure
  - ✓ Can be Software-as-a-Service (SaaS) product being used
- **Integration is the key here!**



[source](#)

Source :  
Monolith to Microservices by Sam Newman



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Monolith to MicroServices

Chandan Ravandur N

# Changing World

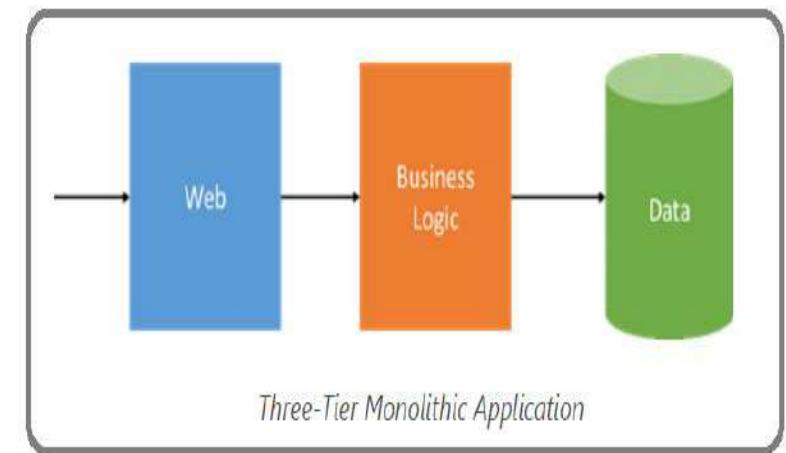
## App development and deployment

- Application development and IT system management revolution is driven by the cloud
- Operational efficiency and faster-time-to-market is being enabled by
  - Fast, agile, inexpensive, and massively scalable infrastructure
  - fully self-service applications
  - pay-as-you-go billing model
  - emergence of containers etc.
- Companies are finding it difficult to make their applications highly available, scalable and agile
- Competitive business pressures demand that applications continuously
  - evolve
  - add new features
  - remain available 24x7
- For example, no longer acceptable for a bank website to have a maintenance window!

# Monolithic application models

## Three tier

- For decades, application development is influenced by the cost, time, and complexity of provisioning new hardware
  - More pronounced when those applications are mission-critical
- IT infrastructure is static, applications were written to be statically sized and designed for specific hardware
- Applications were decomposed to minimize overall hardware requirements and offer some level of agility and independent scaling
  - classic three-tier model, with web, business logic and data tiers
  - each tier was still its own monolith, implementing diverse functions
  - combined into a single package deployed onto hardware pre-scaled for peak loads
- When load caused an application to outgrow its hardware, the answer was typically to “scale up”



[Source : Microsoft](#)

# Monolithic application

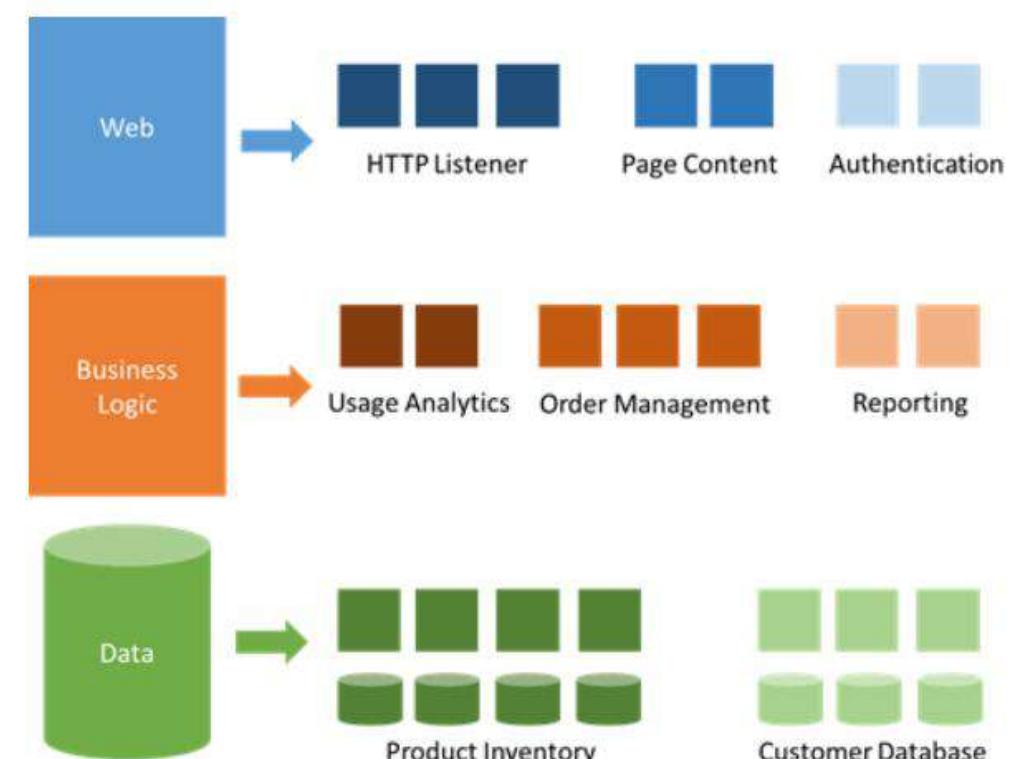
## Challenges

- Natural result of the limitations of infrastructure agility
  - resulted in inefficiencies
  - little advantage to decomposing applications beyond a few tiers
- Challenges
  - A tight coupling between unrelated application services within tiers
  - A change to any application service, even small ones, required its entire tier to be retested and redeployed
  - A simple update could have unforeseen effects on the rest of the tier - changes are risky
  - Lengthening development cycles to allow for more rigorous testing
  - An outright hardware failure could send the entire application into a tailspin

# Solution

## Microservices

- Microservices are a different approach to application development and deployment
  - perfectly suited to the agility, scale and reliability requirements of many modern cloud applications
- A microservices application is decomposed into independent components called “microservices,”
  - work in concert to deliver the application’s overall functionality
- Emphasizes the fact that applications should
  - be composed of services small enough to truly reflect independent concerns such that each microservice implements a single function
  - have well-defined contracts (API contracts) – typically RESTful - for other microservices to communicate and share data with it
  - be able to version and update independently of each other
- Loose coupling is what supports the rapid and reliable evolution of an application



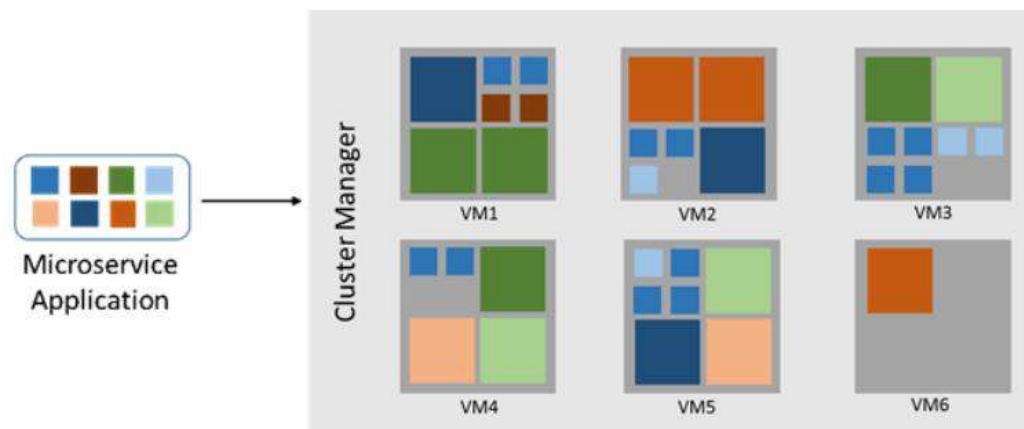
*Breaking the Monolith into Microservices*

[Source : Microsoft](#)

# Micorservices

## Fast Deployments

- For Monolithic applications developers declare resource requirements to IT
- Microservices
  - enable the separation of the application from the underlying infrastructure on which it runs
  - declare their resource requirements to a distributed software system known as a “cluster manager”
  - “schedules,” or places, them onto machines assigned to the cluster
  - Commonly packaged as containers and many usually fit within a single server or virtual machine
    - ❖ deployment is fast and they can be densely packed to minimize the cluster’s scale requirements



*Cluster of Servers with Deployed Microservices*

[Source : Microsoft](#)

# Micorservices

## Easier Deployments

- Microservices-based applications are independent and distributed in nature
- Enables rolling updates
  - only a subset of the instances of a single microservice will update at any given time
  - If a problem is detected, a buggy update can be “rolled back,” or undone
- If the update system is automated and integrated with Continuous Integration (CI) and Continuous Delivery (CD) pipelines
  - allow developers to safely and frequently evolve the application without fear of impacting availability

Reference :

Microservices: An application revolution powered by the cloud  
Mark Russinovich Chief Technology Officer, Microsoft Azure



# Thank You!

In our next session:



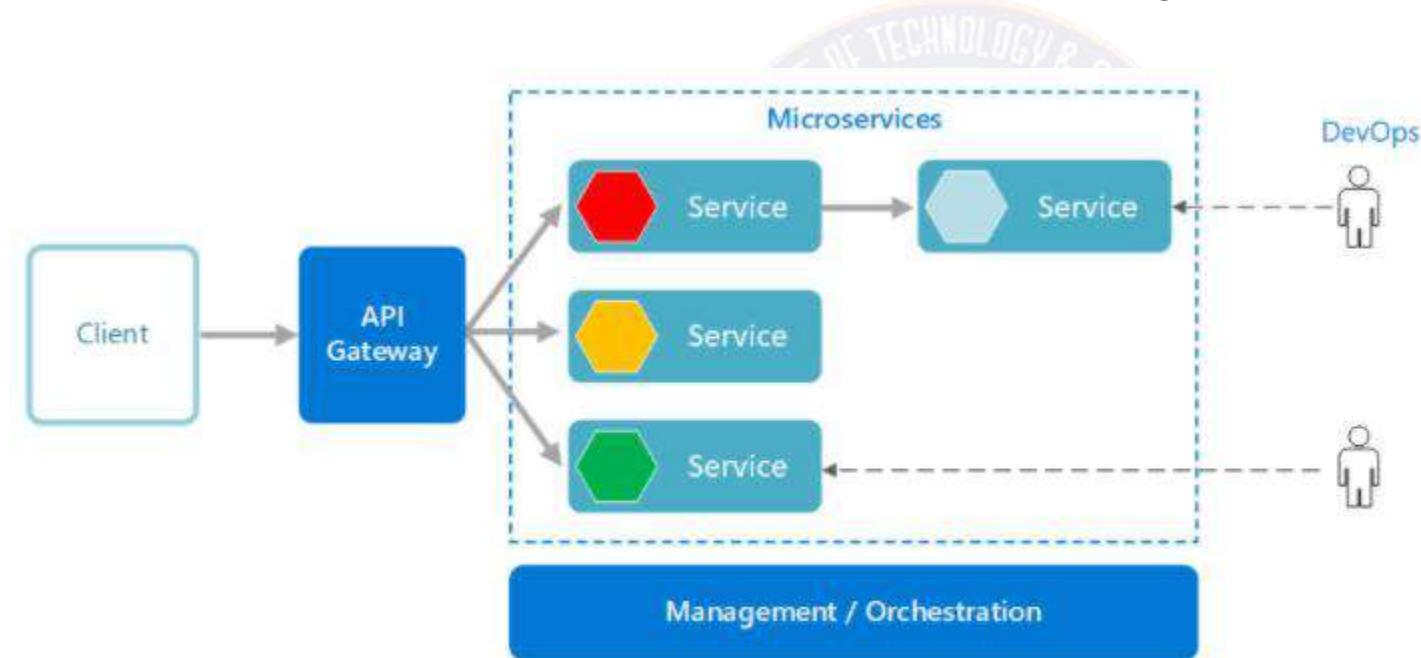
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Microservices

Chandan Ravandur N

# Microservices architecture style

- A microservices architecture consists of a collection of small, autonomous services
- Each service is self-contained and should implement a single business capability



[Source Image : Microsoft](#)

# What are microservices?

- Small, independent, and loosely coupled
  - A single small team of developers can write and maintain a service.
- Each service is a separate codebase
  - can be managed by a small development team.
- Services can be deployed independently
  - A team can update an existing service without rebuilding and redeploying the entire application
- Services are responsible for persisting their own data or external state
- Services communicate with each other by using well-defined APIs
  - Internal implementation details of each service are hidden from other services
- Services don't need to share the same technology stack, libraries, or frameworks

# Characteristics

## Multiple Components

- Software built as microservices can, by definition, be broken down into multiple component services
- Reason
  - ✓ Each of these services can be deployed, tweaked, and then redeployed independently without compromising the integrity of an application
  - ✓ As a result, one might only need to change one or more distinct services instead of having to redeploy entire applications
- Downsides,
  - ✓ Expensive remote calls (instead of in-process calls)
  - ✓ coarser-grained remote APIs
  - ✓ increased complexity when redistributing responsibilities between components

# Characteristics (2)

## Built For Business

- The microservices style is usually organized around business capabilities and priorities
- Unlike a traditional monolithic development approach
  - ✓ where different teams each have a specific focus on,
  - ✓ UIs, databases, technology layers, or server-side logic — microservices architecture utilizes cross-functional teams
  - ✓ The responsibilities of each team are to make specific products based on one or more individual services communicating via message bus
- In microservices, a team owns the product for its lifetime
  - ✓ as in Amazon's oft-quoted maxim “**You build it, you run it**”.

# Characteristics (3)

## Decentralized

- Since microservices involve a variety of technologies and platforms
  - ✓ old-school methods of centralized governance aren't optimal
- Decentralized governance is favored by the microservices community because
  - ✓ its developers strive to produce useful tools that can then be used by others to solve the same problems
- Just like decentralized governance
  - ✓ microservice architecture also favors decentralized data management
- Monolithic systems use a single logical database across different applications.
- In a microservice application, each service usually manages its unique database.

# Characteristics (4)

## Failure Resistant

- Microservices are designed to cope with failure.
- As several unique and diverse services are communicating together
  - ✓ quite possible that a service could fail, for one reason or another
  - ✓ e.g., when the supplier isn't available
- Here, client should allow its neighboring services to function while it bows out in as graceful a manner as possible
  - ✓ Monitoring microservices can help prevent the risk of a failure
  - ✓ For obvious reasons, this requirement adds more complexity to microservices as compared to monolithic systems architecture.

References:

- Microservices architecture style
  - ✓ Microsoft Architecture Guide
- [Microservices by SmartBear](#)



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Microservices – Benefits & Challenges

Chandan Ravandur N

# Benefits

- Agility
  - Easier to manage bug fixes and feature releases
  - Update a service without redeploying the entire application, and roll back an update if something goes wrong
- Small, focused teams
  - Should be small enough that a single feature team can build, test, and deploy it
  - Small team sizes promote greater agility - less communication, less management
- Small code base
  - In a monolithic application, there is a tendency over time for code dependencies to become tangled. Adding a new feature requires touching code in a lot of places. By not sharing code or data stores, a microservices architecture minimizes dependencies, and that makes it easier to add new features.

# Benefits(2)

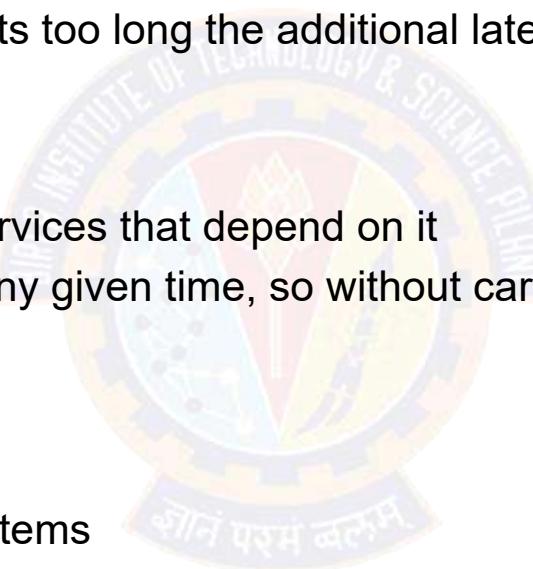
- Mix of technologies
  - Teams can pick the technology that best fits their service, using a mix of technology stacks as appropriate
- Fault isolation
  - If an individual microservice becomes unavailable, it won't disrupt the entire application, as long as any upstream microservices are designed to handle faults correctly
- Scalability
  - Services can be scaled independently, letting you scale out subsystems that require more resources, without scaling out the entire application
  - Using an orchestrator such as Kubernetes or Service Fabric
- Data isolation
  - It is much easier to perform schema updates, because only a single microservice is affected

# Challenges

- Complexity
  - Has more moving parts than the equivalent monolithic application
  - Each service is simpler, but the entire system as a whole is more complex
- Development and testing
  - Existing tools are not always designed to work with service dependencies
  - Refactoring across service boundaries can be difficult
  - Challenging to test service dependencies, especially when the application is evolving quickly
- Lack of governance
  - Many different languages and frameworks that the application becomes hard to maintain
  - It may be useful to put some project-wide standards in place

# Challenges (2)

- Network congestion and latency
  - The use of many small, granular services can result in more interservice communication
  - If the chain of service dependencies gets too long the additional latency can become a problem
- Versioning
  - Updates to a service must not break services that depend on it
  - Multiple services could be updated at any given time, so without careful design, you might have problems with backward or forward compatibility.
- Skillset
  - Microservices are highly distributed systems
  - Carefully evaluate whether the team has the skills and experience to be successful



Reference:

Microservices architecture style  
Microsoft Architecture Guide



# Thank You!

In our next session:

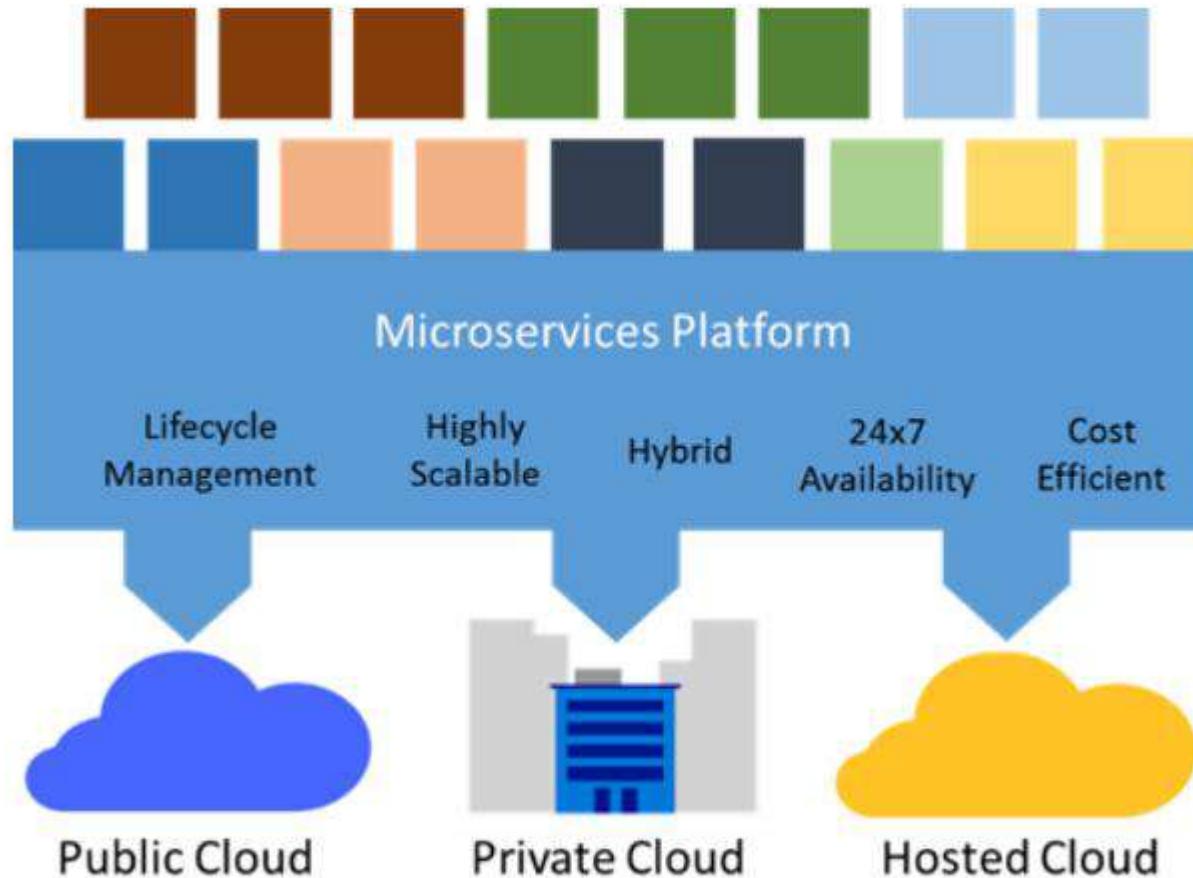


**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Microservice Application Platforms

Chandan Ravandur N

# Microservice application platforms



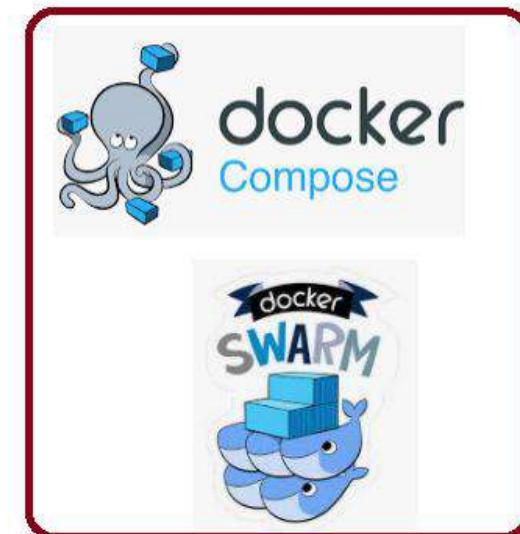
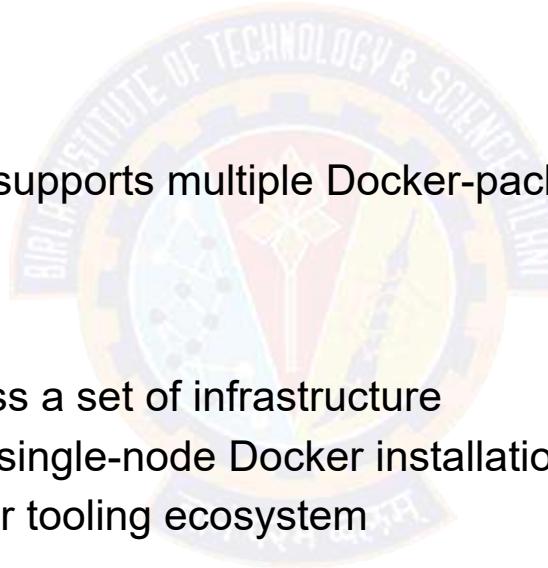
*Microservices Platform*

[Source : Microsoft](#)

# Microservice application platforms

## Docker Swarm and Docker Compose

- Natural fit with microservices architectures
  - Standard packaging format and resource isolation of Docker containers
- Docker Compose
  - defines an application model that supports multiple Docker-packaged microservices
- Docker Swarm
  - serves as a cluster manager across a set of infrastructure
  - exposes the same protocol that a single-node Docker installation does
  - easily works with the broad Docker tooling ecosystem



# Microservice application platforms

## Kubernetes

- Developed by Google - based on experience in Search and Gmail
- Open-source system for
  - automating deployment
  - operations
  - scaling of containerized applications
- Groups containers that make up an application into logical units for easy management and discovery
  - Even some traditional PaaS solutions are merging with Kubernetes, like Apprenda



**kubernetes**

# Microservice application platforms

## Mesosphere DCOS, with Apache Mesos and Marathon

- Mesosphere Datacenter Operating System (DCOS), powered by Mesos
  - is a scalable cluster manager that includes Mesosphere's Marathon, a production-grade container orchestration tool
- Provides microservices platform capabilities including
  - service discovery
  - load-balancing
  - health-checks
  - placement constraints
  - metrics aggregation
- Offers a library of certified services can all be installed with a single command
  - Kafka
  - Chronos
  - Cassandra
  - Spark
- Microsoft and Mesosphere have partnered to bring open source components of the Mesosphere Datacenter Operating System (DCOS)
  - including Apache Mesos and Marathon, to Azure



# Microservice application platforms

## OpenShift

- Backed by Red Hat
- Platform-as-a-service offering that leverages Docker container-based packaging for
  - deploying container orchestration and compute management capabilities for Kubernetes
  - enabling users to run containerized JBoss Middleware
  - multiple programming languages, databases and other application runtimes
- OpenShift Enterprise offers a devops experience
  - enables developers to automate the application build and deployment process within a secure, enterprise-grade application infrastructure





# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Distributed Systems and Fallacies

Chandan Ravandur N

# Distributed Systems

- The hurdles todays developers face when they are building applications for the first time are
  - ✓ Needs to deal with services that are not on the same machine
  - ✓ Need to deal with patterns that consider network between machines
- Without knowing they have entered into world of distributed systems!
- Distributed system is system in which individual computers are connected through a network and appear as single computer to the outside world.
  - ✓ Provides power across machines to accomplish scalability, reliability and economics
- For example,
  - ✓ Most cloud providers are using cheaper commodity hardware for solving common problems of high availability and reliability through software

# Fallacies of Distributed System

## Couple of incorrect or unfounded assumptions

- Peter Deutsch, in 1994, identified them, still have validity today



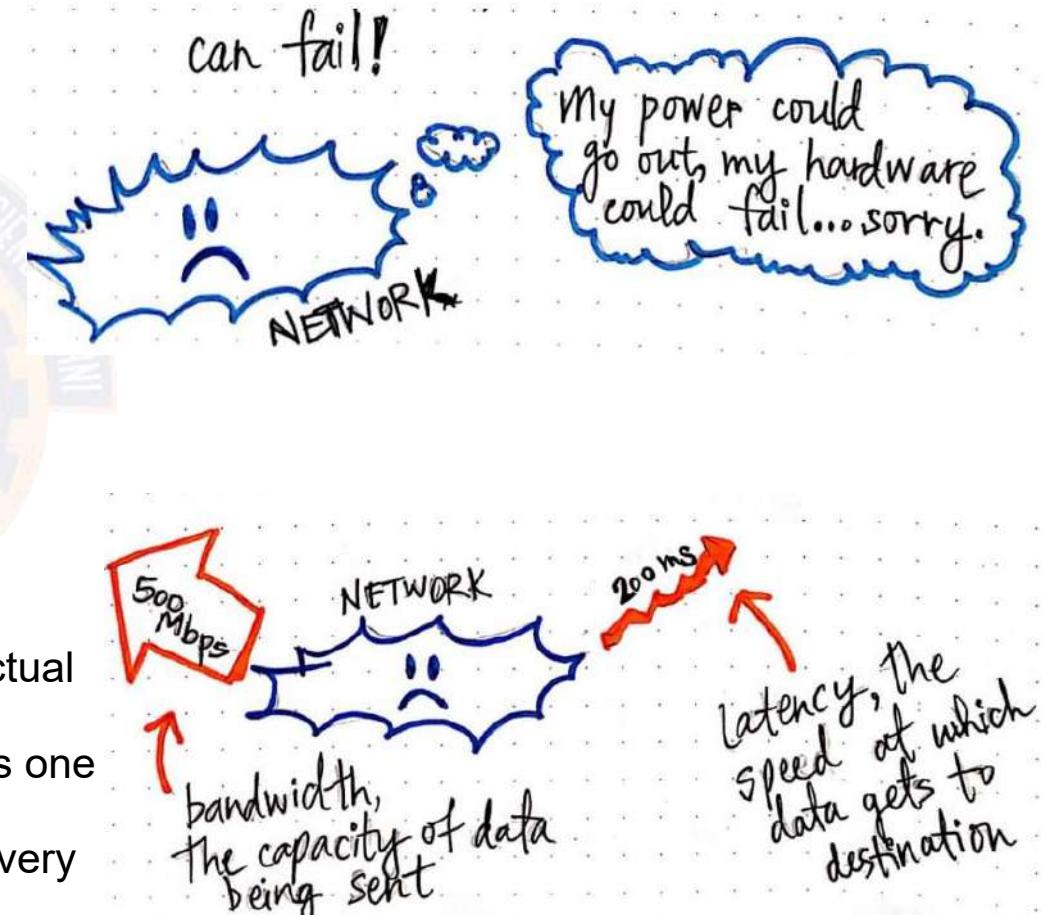
source

# Fallacies

- Fallacy #1: The network is reliable
  - An easy way to set up for failure
  - A few possible problems are:
    - ✓ power failure
    - ✓ old network equipment
    - ✓ network congestion
    - ✓ weak wireless signal
    - ✓ software network stack issues
    - ✓ rodents
    - ✓ DDOS attacks... etc.

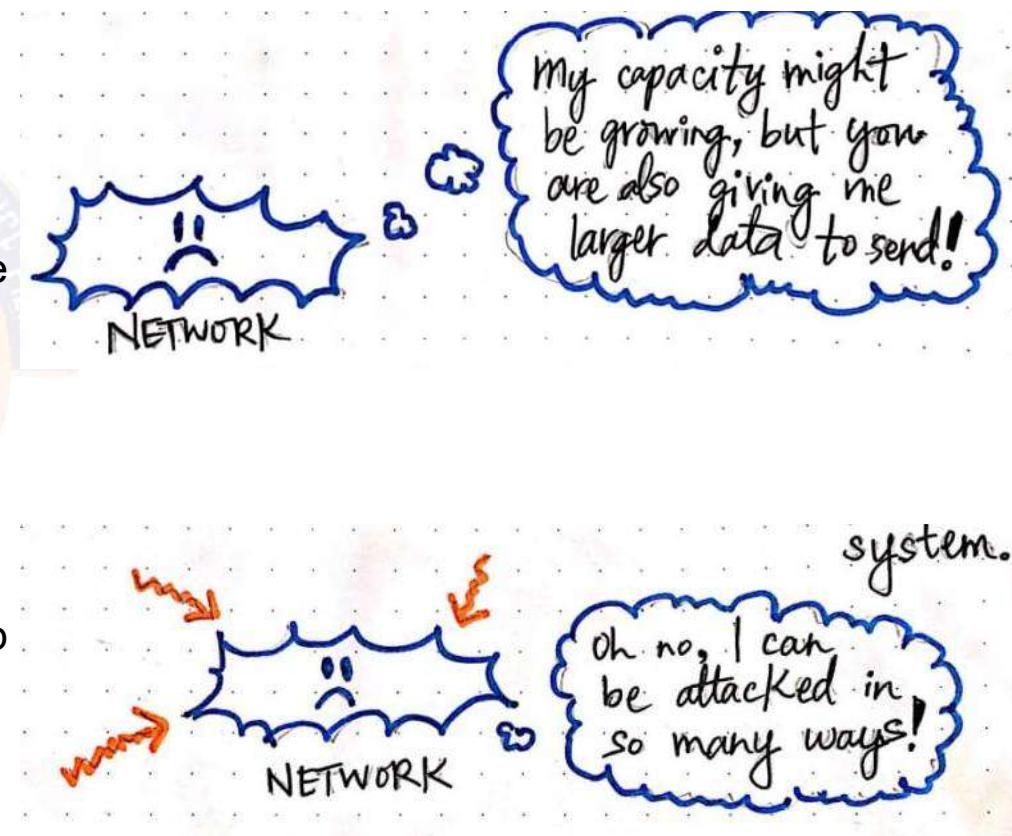
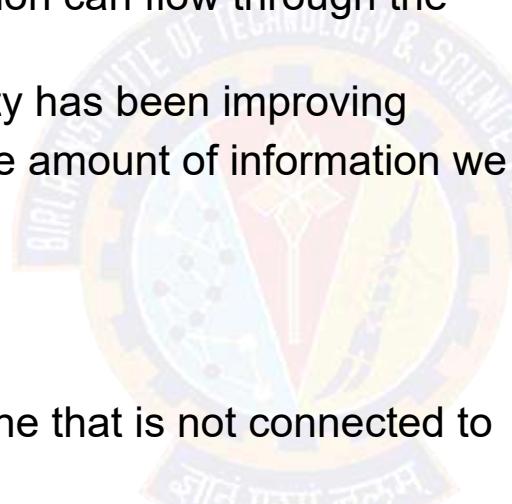


- Fallacy #2: Latency is zero
  - ✓ Latency is the time it takes between a request and the start of actual response data
  - ✓ Latency within development platforms networks can be as low as one millisecond
  - ✓ In production, when traffic is going over the internet, the story is very different
  - ✓ At this phase, latency is not a constant rate but changes very often.



# Fallacies(2)

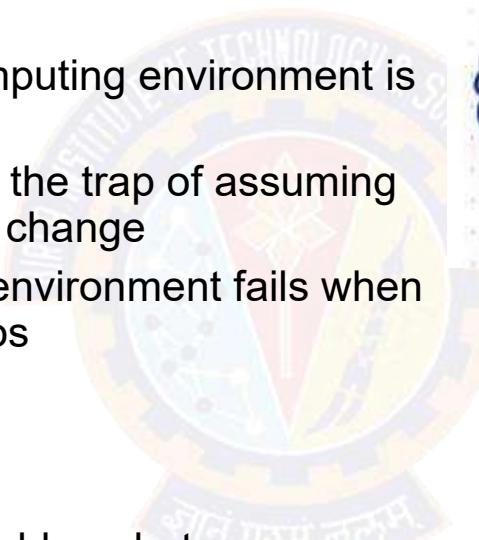
- Fallacy #3: Bandwidth is infinite
  - ✓ Bandwidth is the capacity of a network to transfer data
  - ✓ Higher bandwidth means more information can flow through the network
  - ✓ Even though network bandwidth capacity has been improving
  - ✓ at the same time we tend to increase the amount of information we want to transfer
- Fallacy #4: The network is secure
  - ✓ The only completely secure system is one that is not connected to any network
  - ✓ Keeping the network secure is a complex task and is a full-time job for many professionals
  - ✓ There is a wide variety of both passive and active network attacks that can render network traffic unsafe from malicious users



# Fallacies(3)

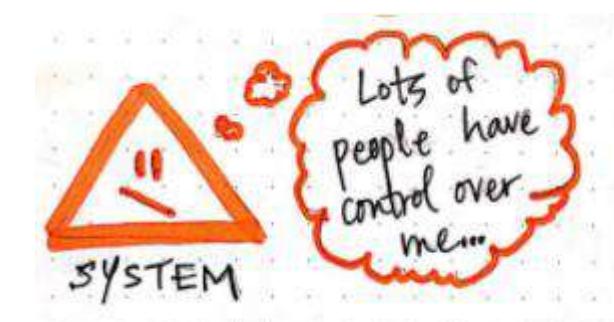
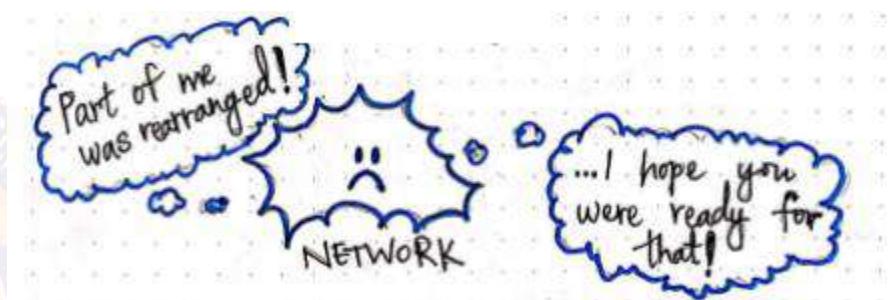
- Fallacy #5: Topology doesn't change

- ✓ A network topology is the arrangement of the various network elements
- ✓ The network landscape in a modern computing environment is always evolving
- ✓ For new developers, it is easy to fall into the trap of assuming that the operating environment does not change
- ✓ Software that works in the dev and test environment fails when deployed to production or cloud scenarios



- Fallacy #6: There is one administrator

- ✓ For small systems this might not be a problem, but once you deploy to an enterprise-wide or Internet scenario
- ✓ multiple networks are being touched
- ✓ one can be sure they are managed by different people, different security policies, and requirements



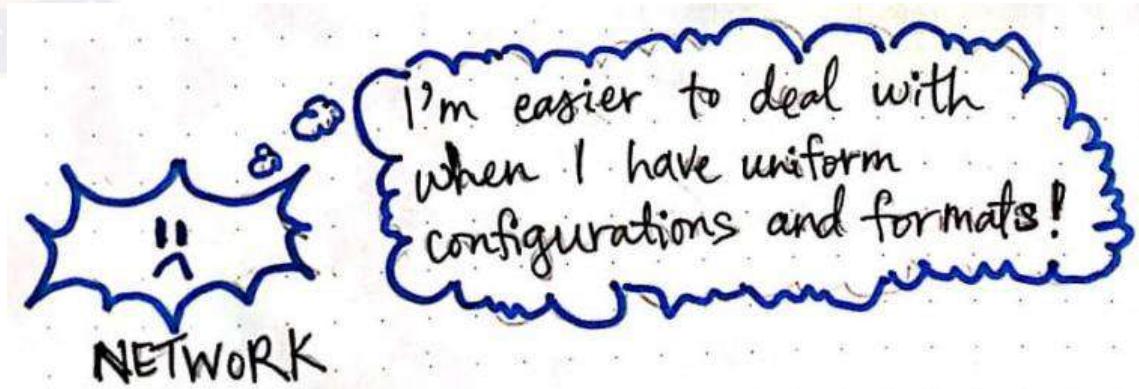
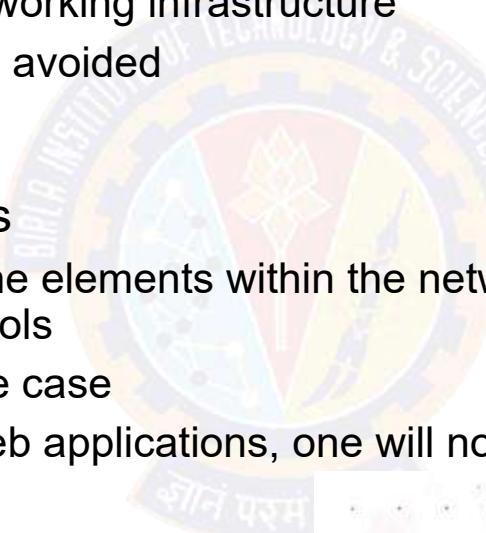
# Fallacies(4)

- Fallacy #7: Transport cost is zero

- ✓ Need to do with the overhead of serializing data into the network
- ✓ Other has to do with the costs of the networking infrastructure
- ✓ Both of these are realities that cannot be avoided

- Fallacy #8: The network is homogeneous

- ✓ A homogeneous network is one where the elements within the network are using a uniform set of configuration and protocols
- ✓ For very small networks this might be the case
- ✓ For large distributed systems such as web applications, one will not be able to
  - ❖ the devices that will connect
  - ❖ the protocols used to connect
  - ❖ the operating systems, browsers, etc



The cost of transporting **data** to/from the system takes **resources**, which **costs money**.

The cost of sending **data** to/from the system **costs time and effort** to serialize and deserialize that data.

# Solutions

Fallacy	Solutions
The network is reliable	Automatic Retries, Message Queues
Latency is zero	Caching Strategy, Bulk Requests, Deploy in AZs near client
Bandwidth is infinite	Throttling Policy, Small payloads with Microservices
The network is secure	Network Firewalls, Encryption, Certificates, Authentication
Topology does not change	No hardcoding IP, Service Discovery Tools
There is one administrator	DevOps Culture eliminates Bus Factor
Transport cost is zero	Standardized protocols like JSON, Cost Calculation
The network is homogenous	Circuit Breaker, Retry and Timeout Design Pattern

References :

- 1) [Fallacies of distributed Systems](#)
- 2) [Images](#)



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

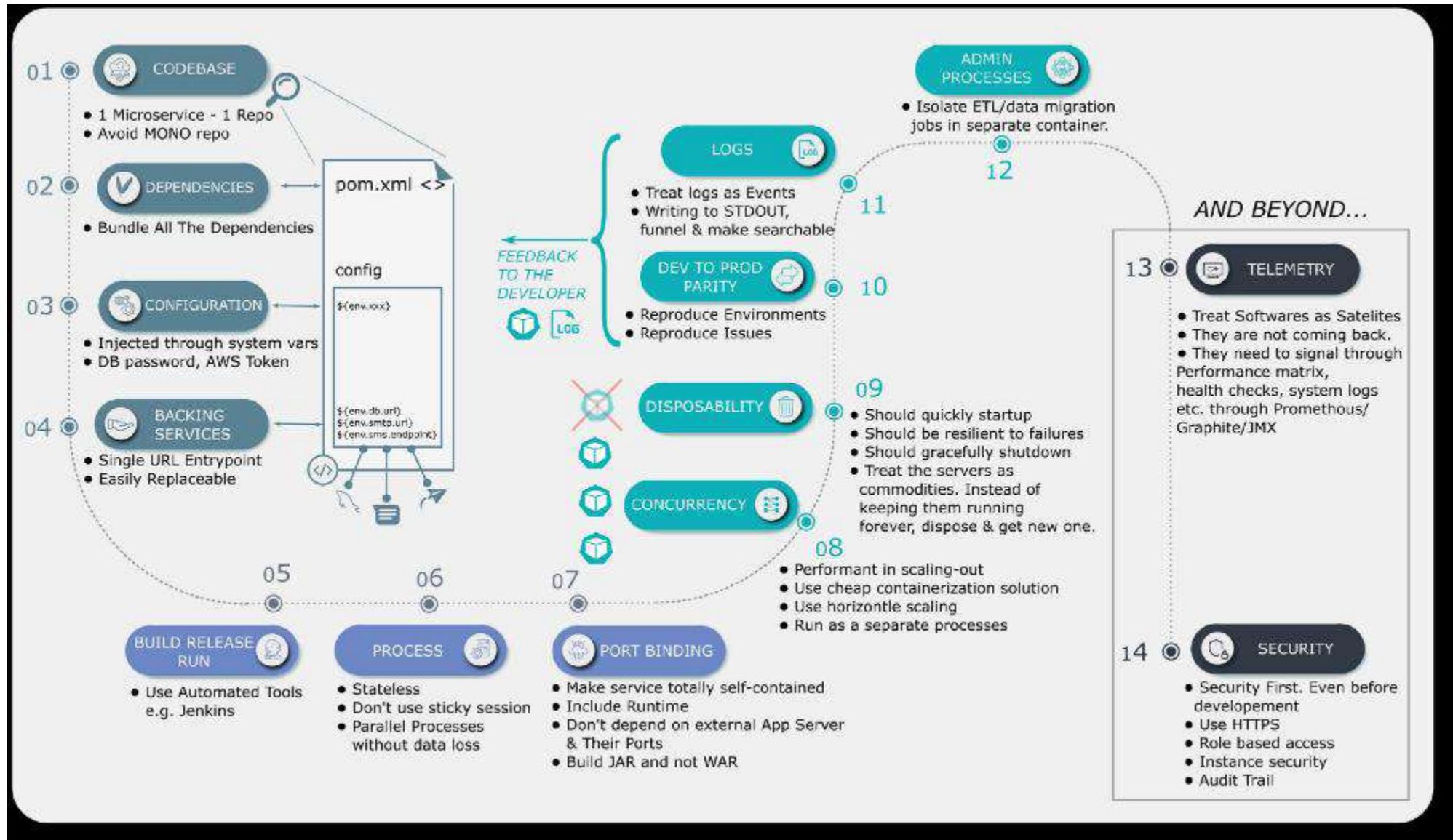
# 12 Factor App

Chandan Ravandur N

# Need

- Developers are moving apps to the cloud, and in doing so, they become more experienced at designing and deploying cloud-native apps!
- From that experience, a set of best practices, commonly known as the twelve factors, has emerged
- Designing an app with these factors in mind
  - ✓ lets you deploy apps to the cloud that are more portable and resilient
  - ✓ when compared to apps deployed to on-premises environments where it takes longer to provision new resources
- Designing modern, cloud-native apps requires a change in how you think about
  - ✓ software engineering
  - ✓ configuration
  - ✓ and deployment
- when compared to designing apps for on-premises environments

# 12 factors and beyond



source



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

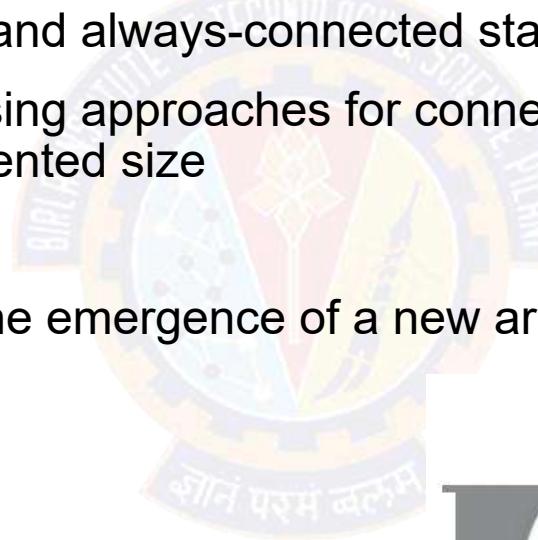
# Cloud Native Architecture

Chandan Ravandur N

# Introducing cloud-native software

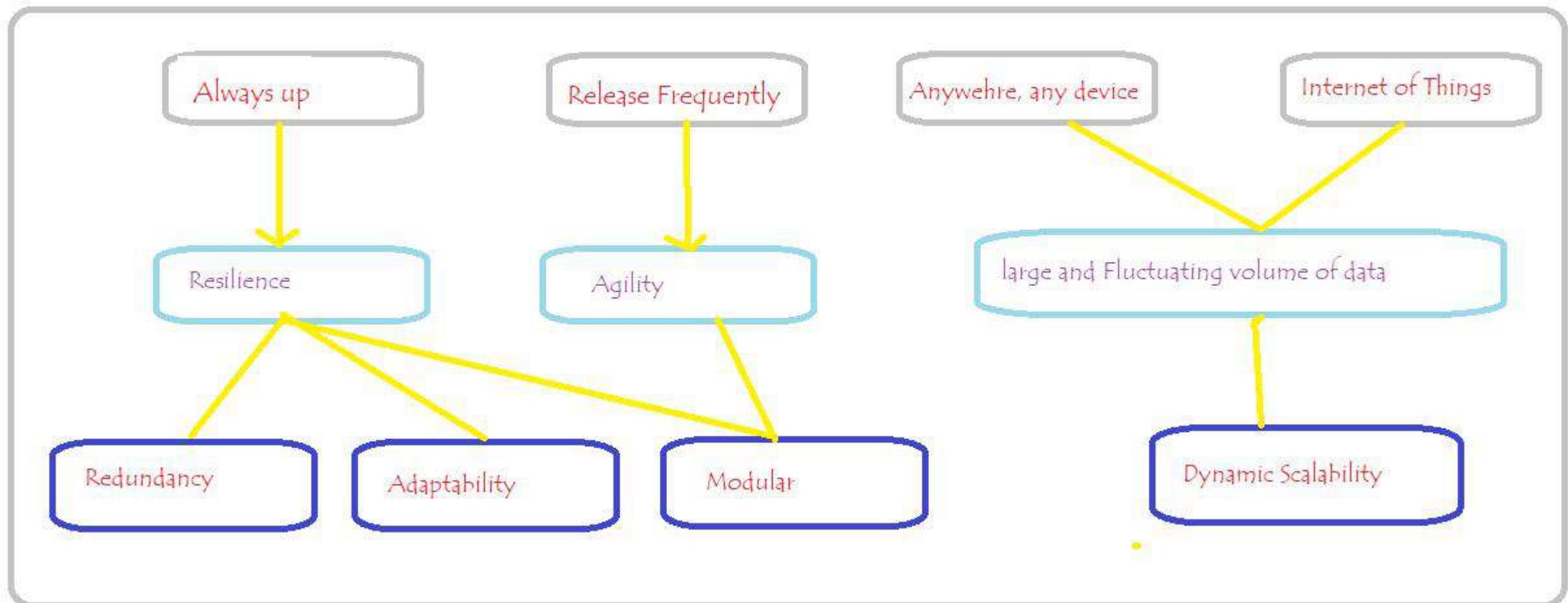
## New Edge Applications Demands

- Needs to be up, software available 24/7 for 365 days
- Need to be able to release frequently to give users the instant gratification
- Needs to take care of the mobility and always-connected state of users drives
- Requires new storage and processing approaches for connected devices (“things”) that form a distributed data fabric of unprecedented size
- These needs have led directly to the emergence of a new architectural style for software:
  - **cloud-native software!**



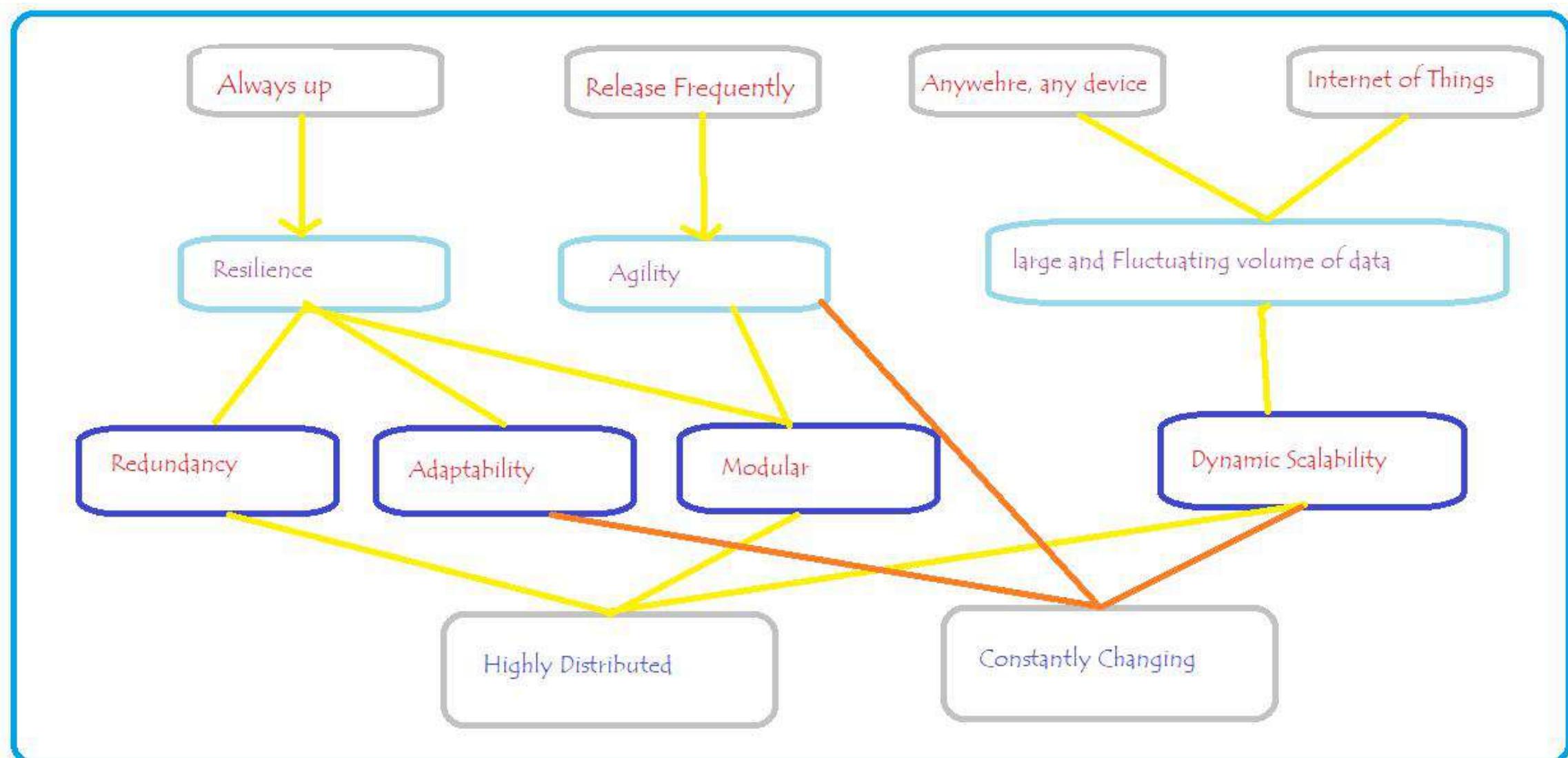
# User requirements for software

## What characterizes cloud-native software?



# Architectural and management tenets

## What defines “cloud native” software?



# Defining “Cloud Native”

## Two important Characteristics

- Deployment
  - ✓ Software that's constructed as a set of independent components, redundantly deployed, implies distribution
  - ✓ Redundant copies were all deployed close to one another, be at greater risk of local failures
  - ✓ Make efficient use of the infrastructure resources while deploying additional instances of an app
  - ✓ from cloud services such as AWS, Google Cloud Platform (GCP), and Microsoft Azure
  - ✓ As a result, you deploy software modules in a **highly distributed manner**
- Adaptable software
  - ✓ “able to adjust to new conditions,” and the conditions are those of the infrastructure and the set of interrelated software modules
  - ✓ intrinsically tied together: as the infrastructure changes, the software changes, and vice versa
  - ✓ Frequent releases mean frequent change
  - ✓ adapting to fluctuating request volumes through scaling operations represents a constant adjustment
  - ✓ Clear that software and the environment it runs in are **constantly changing**
- **Cloud-native software is highly distributed, must operate in a constantly changing environment, and is itself constantly changing.**

Reference:  
Cloud Native Patterns by Cornelia Davis



# Thank You!

In our next session:



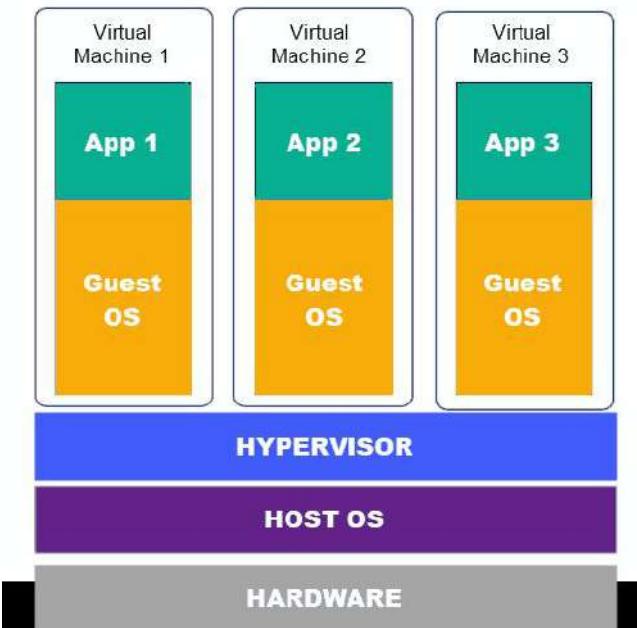
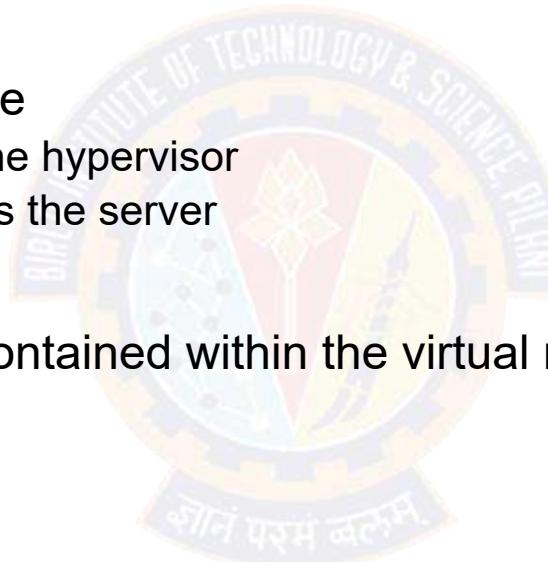
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Containers

Chandan Ravandur N

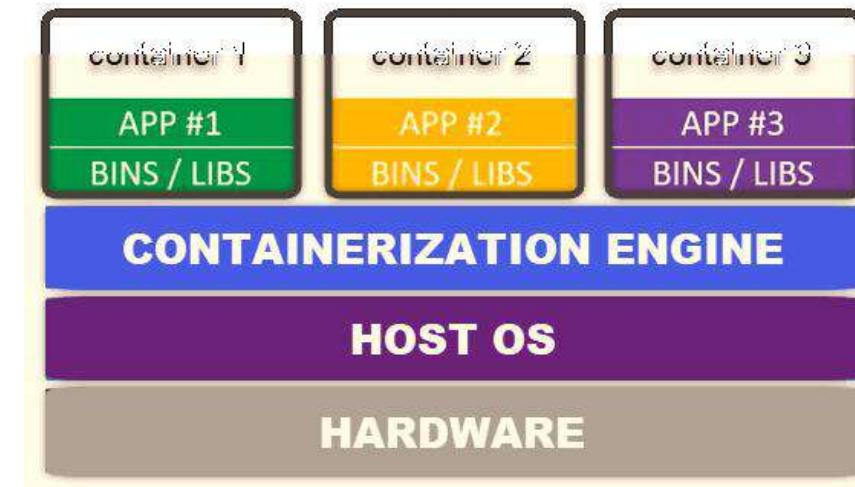
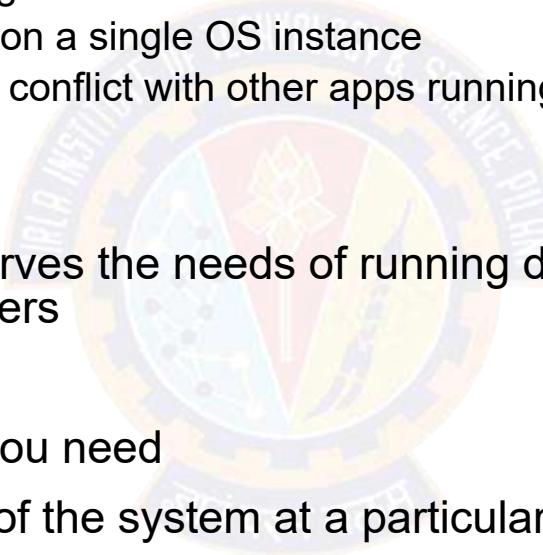
# Virtual Machines

- A virtual machine (VM) is system that shares the physical resources of one server
  - ✓ is a guest on the host's hardware, which is why it is also called a guest machine
- Several layers make up a virtual machine
  - ✓ The layer that enables virtualization is the hypervisor
  - ✓ A hypervisor is a software that virtualizes the server
- Everything necessary to run an app is contained within the virtual machine –
  - ✓ the virtualized hardware
  - ✓ an OS
  - ✓ any required binaries and libraries
- Virtual machines have their own infrastructure and are self-contained
- Disadvantages
  - ✓ Virtual machines may take up a lot of system resources of the host machine
  - ✓ The process of relocating an app running on a virtual machine can also be complicated



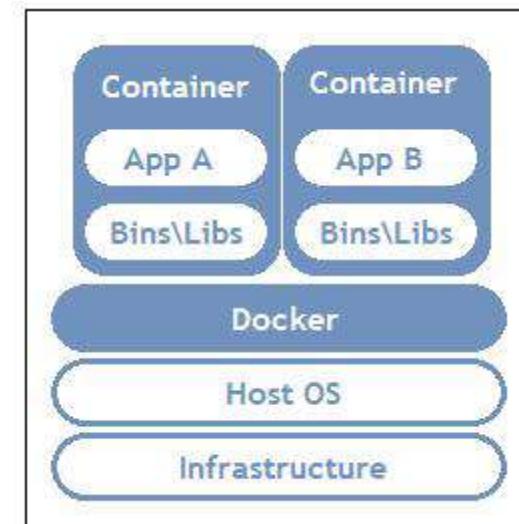
# Containers

- A container is an environment that runs an application that is not dependent on the operating system
  - ✓ isolates the app from the host by virtualizing it
  - ✓ allows users to created multiple workloads on a single OS instance
  - ✓ cannot harm the host machine nor come in conflict with other apps running in separate containers
- The kernel of the host operating system serves the needs of running different functions of an app, separated into containers
- Can create a template of an environment you need
- The container essentially runs a snapshot of the system at a particular time
  - ✓ providing consistency in the behavior of an app
- The container shares the host's kernel to run all the individual apps within the container
  - ✓ The only elements that each container requires are bins, libraries and other runtime components



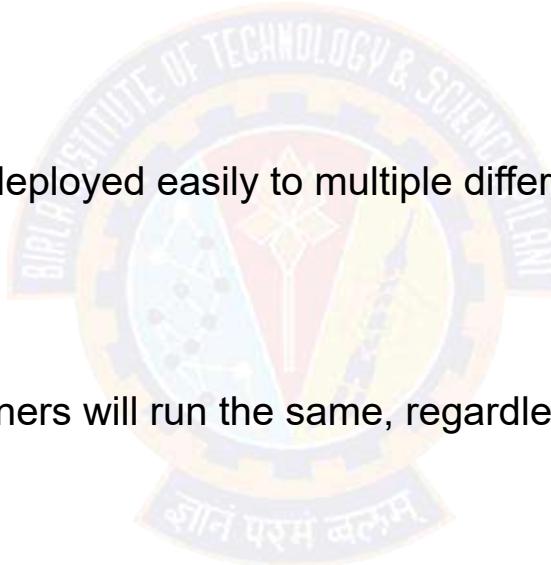
# Containers(2)

- Containers were brought into spotlight by start-ups and born-in-cloud companies
- Over couple of years, they have become synonymous with app modernization
- Mainly people talk about Docker containers
- It has really made container popular
- But other container runtimes are also available such as
  - ✓ Containerd
  - ✓ CoreOS rkt
  - ✓ Hyper-V Containers
  - ✓ LXC Linux Containers
  - ✓ OpenVZ
  - ✓ RunC
  - ✓ Vagrant



# Benefits of containers

- Less overhead
  - ✓ Containers require less system resources than traditional or hardware virtual machine environments because they don't include operating system images.
- Increased portability
  - ✓ Applications running in containers can be deployed easily to multiple different operating systems and hardware platforms.
- More consistent operation
  - ✓ DevOps teams know applications in containers will run the same, regardless of where they are deployed.
- Greater efficiency
  - ✓ Containers allow applications to be more rapidly deployed, patched, or scaled.
- Better application development
  - ✓ Containers support agile and DevOps efforts to accelerate development, test, and production cycles.

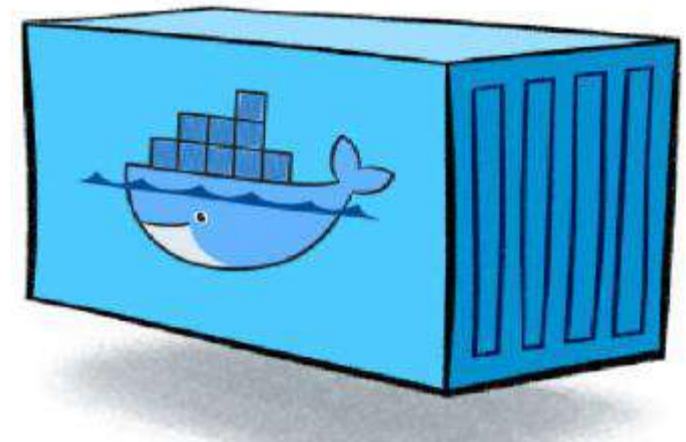
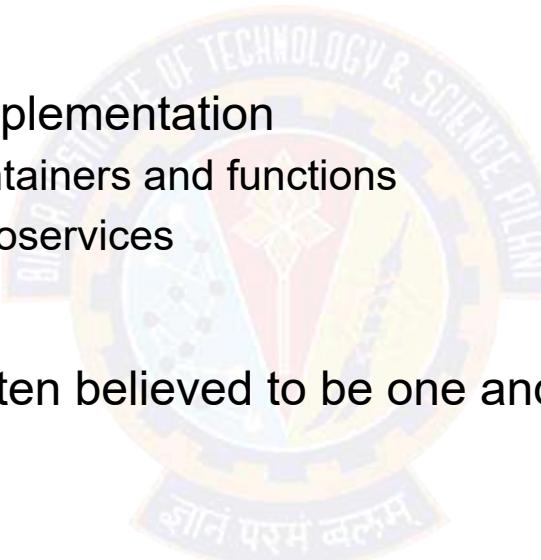


# Container use cases

- “Lift and shift” existing applications into modern cloud architectures
  - ✓ Some organizations use containers to migrate existing applications into more modern environments
  - ✓ does not offer the full benefits of a modular, container-based application architecture
- Refactor existing applications for containers
  - ✓ Although refactoring is much more intensive than lift-and-shift migration, it enables the full benefits of a container environment.
- Develop new container-native applications
  - ✓ Much like refactoring, this approach unlocks the full benefits of containers.
- Provide better support for microservices architectures
  - ✓ Distributed applications and microservices can be more easily isolated, deployed, and scaled using individual container building blocks.
- Provide DevOps support for continuous integration and deployment (CI/CD)
  - ✓ Container technology supports streamlined build, test, and deployment from the same container images.
- Provide easier deployment of repetitive jobs and tasks
  - ✓ Containers are being deployed to support one or more similar processes, which often run in the background, such as ETL functions or batch jobs.

# Containers in Cloud native

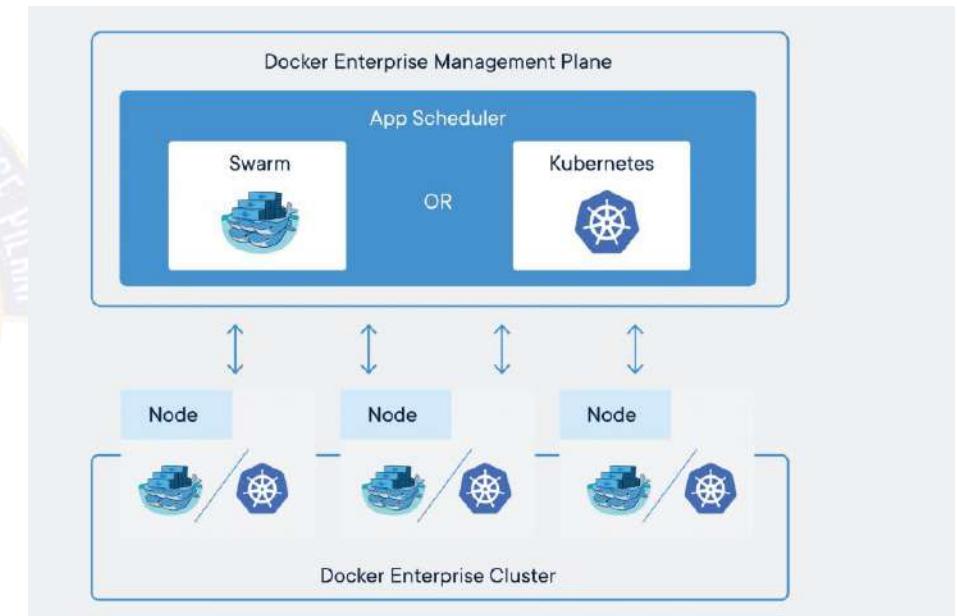
- Cloud native apps are distributed in nature and utilize cloud infrastructure
- Many technologies and tools used for implementation
  - ✓ From compute perspective – its only containers and functions
  - ✓ From architectural perspective – its microservices
- These terms are mistakenly used and often believed to be one and the same
- Understanding how to best use functions and containers, along with eventing or messaging technologies, allows developers to design, develop and operate new generation of cloud-native microservices based applications



[source](#)

# Containers in Production

- Containers are a form of operating system virtualization
  - ✓ A single container might be used to run anything from a small microservices or software process to a larger application
  - ✓ Inside a container are all the necessary executables, binary code, libraries, and configuration files
- Containers do not contain operating system images
  - ✓ makes them more lightweight and portable, with significantly less overhead
- In larger application deployments, multiple containers may be deployed as one or more container clusters
  - ✓ Such clusters might be managed by a container orchestrator such as Kubernetes

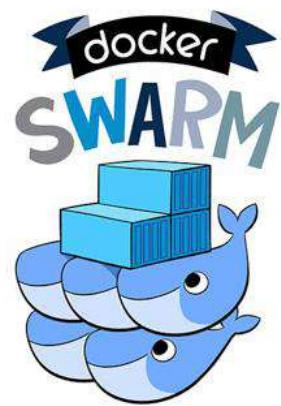
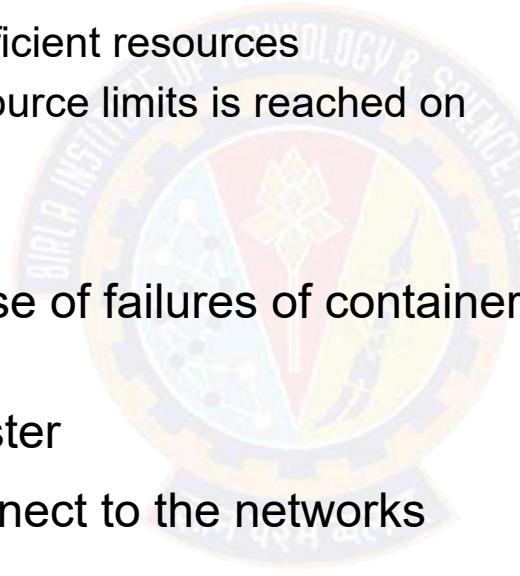


Source : medium

# Container Orchestration

## Tasks of container orchestrator

- Provisioning and deployment of containers onto cluster nodes
- Resource management of containers
  - ✓ Placing containers on nodes having sufficient resources
  - ✓ Moving containers to other nodes if resource limits is reached on node
- Health monitoring of containers
- Container restarting, rescheduling in case of failures of container or node
- Scaling in or out containers within a cluster
- Providing mapping for containers to connect to the networks
- Initial load balancing between containers



[source](#)



# Thank You!

In our next session:



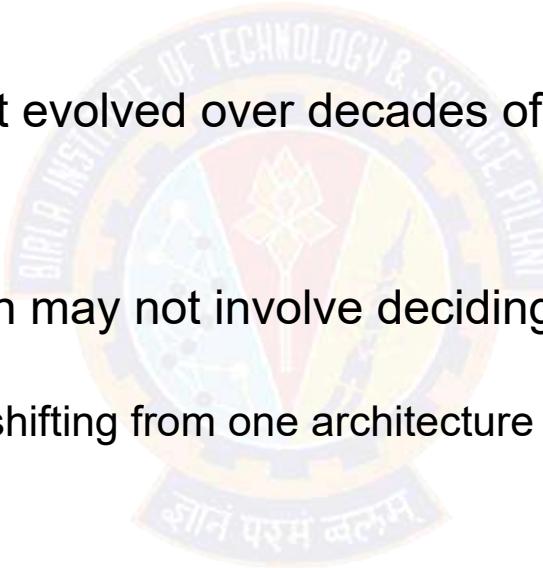
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Architecture deployment approaches - I

Chandan Ravandur N

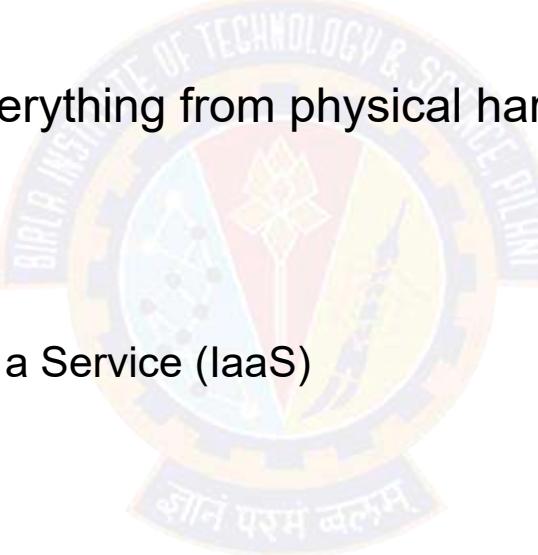
# Architecture approaches

- Understanding existing approaches to architecting enterprise apps helps clarify the role played by Serverless
- Many approaches and patterns that evolved over decades of software development
  - ✓ all have their own pros and cons
- In many cases, the ultimate solution may not involve deciding on a single approach but may integrate several approaches
  - ✓ Migration scenarios often involve shifting from one architecture approach to another through a hybrid approach
- Common approaches
  - ✓ Monoliths
  - ✓ N-layer
  - ✓ Microservices etc.



# Architecture deployment approaches

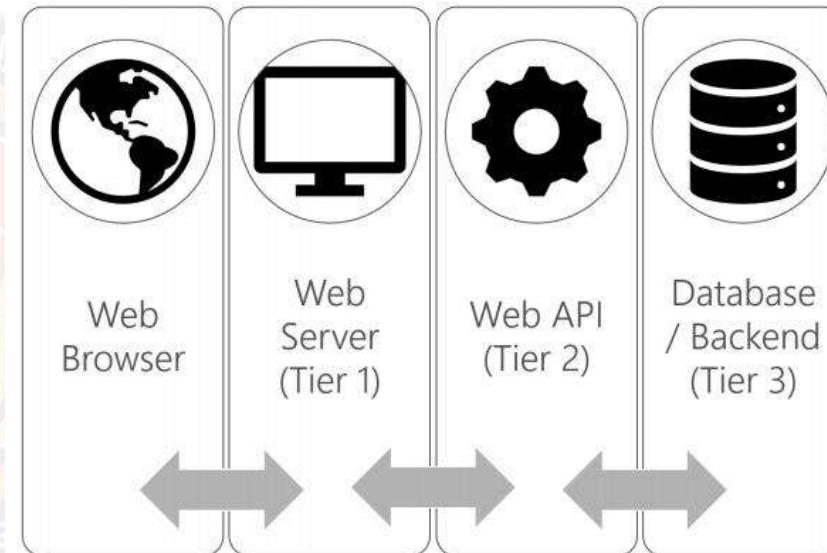
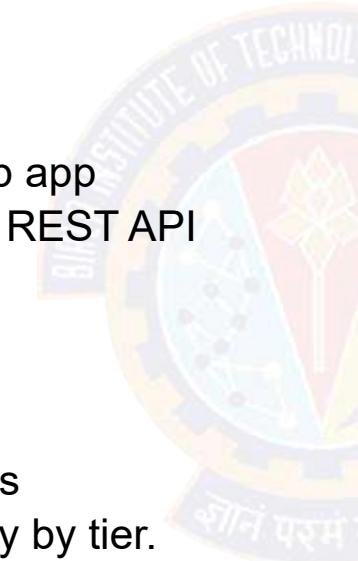
- Regardless of the architecture approach used to design a business application,
- the implementation, or deployment of those applications may vary
- Businesses host applications on everything from physical hardware to Serverless functions
- Conventional deployment patterns
  - ✓ N-Tier applications
  - ✓ On-premises and Infrastructure as a Service (IaaS)
  - ✓ Platform as a Service (PaaS)
  - ✓ Software as a Service (SaaS)
- Modern deployment patterns
  - ✓ Containers and Functions as a Service (Caas and FaaS)
  - ✓ Serverless



# N-Tier applications

## Mature architecture

- Refers to applications that separate various logical layers into separate physical tiers
- A physical implementation of N-Layer architecture
- The most common implementations:
  - ✓ A presentation tier, for example a web app
  - ✓ An API or data access tier, such as a REST API
  - ✓ A data tier, such as a SQL database
- Characteristics:
  - ✓ Projects are typically aligned with tiers
  - ✓ Testing may be approached differently by tier.
  - ✓ Tiers provide layers of abstraction
  - ✓ Typically, layers only interact with adjacent layers.
  - ✓ Releases are often managed at the project, and therefore tier, level.
  - ✓ A simple API change may require a new release of an entire middle tier

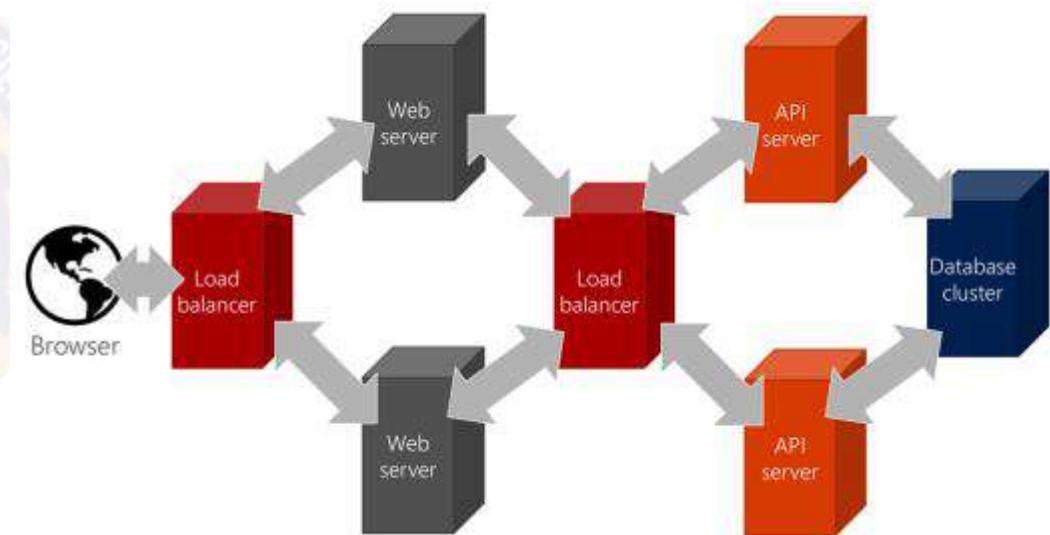
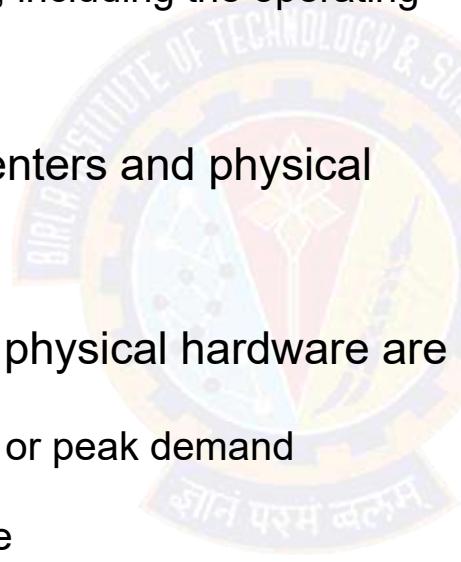


source : Microsoft

# On-premises

## On-premise hosting

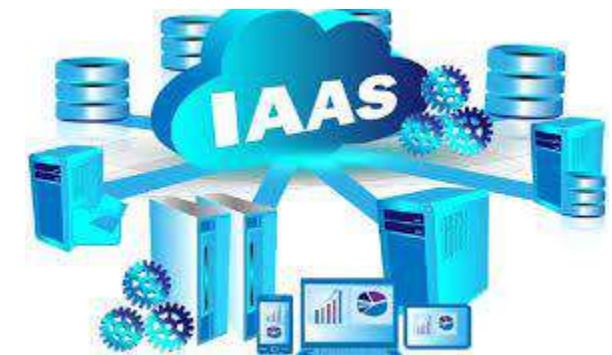
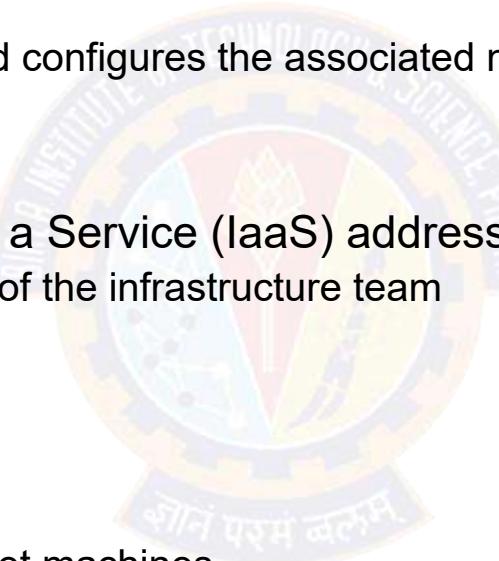
- The traditional approach to hosting applications requires
  - ✓ buying hardware
  - ✓ managing all of the software installations, including the operating system
- Originally this involved expensive data centers and physical hardware
- The challenges that come with operating physical hardware are many, including:
  - ✓ The need to buy excess for “just in case” or peak demand scenarios
  - ✓ Securing physical access to the hardware
  - ✓ Responsibility for hardware failure (such as disk failure)
  - ✓ Cooling
  - ✓ Configuring routers and load balancers
  - ✓ Power redundancy
  - ✓ Securing software access



source : Microsoft

# Infrastructure as a Service (IaaS)

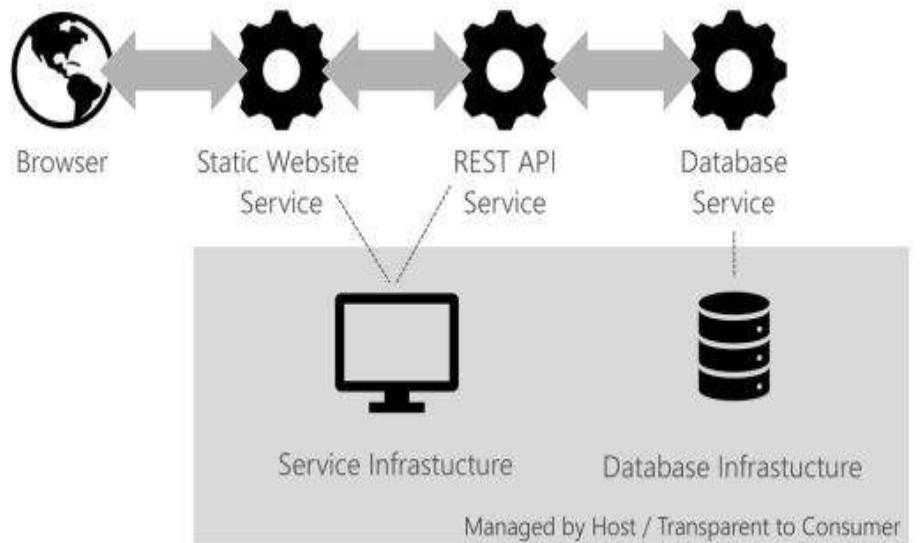
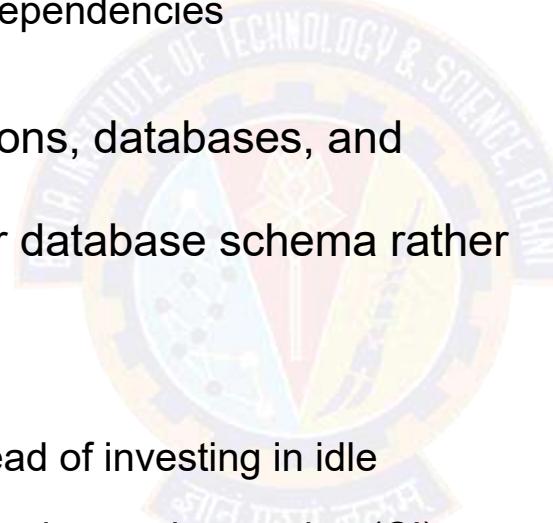
- Virtualization of hardware, via “virtual machines” enables Infrastructure as a Service (IaaS)
  - ✓ Host machines are effectively partitioned to provide resources to instances with allocations for their own memory, CPU, and storage
  - ✓ The team provisions the necessary VMs and configures the associated networks and access to storage.
- Although virtualization and Infrastructure as a Service (IaaS) address many concerns
  - ✓ still leaves much responsibility in the hands of the infrastructure team
- The team maintains
  - ✓ operating system versions
  - ✓ applies security patches
  - ✓ installs third-party dependencies on the target machines
- Although many organizations deploy N-Tier applications to these targets, many companies benefit from deploying to a more cloud native model such as Platform as a Service.



source:FileCloud

# Platform as a Service (PaaS)

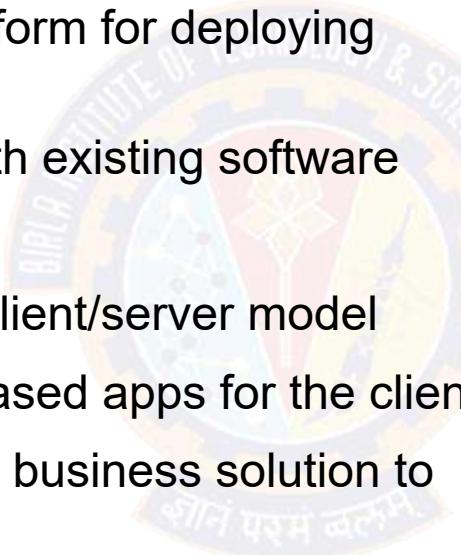
- Offers configured solutions that developers can plug into directly
  - ✓ Another term for managed hosting
  - ✓ Eliminates the need to manage the base operating system, security patches and in many cases any third-party dependencies
- Examples of platforms include web applications, databases, and mobile back ends
- Allows the developer to focus on the code or database schema rather than how it gets deployed
- Benefits of PaaS include:
  - ✓ Pay for use models that eliminate the overhead of investing in idle machines
  - ✓ Direct deployment and improved DevOps, continuous integration (CI), and continuous delivery (CD) pipelines
  - ✓ Automatic upgrades, updates, and security patches
  - ✓ Push-button scale out and scale up (elastic scale)
- The main disadvantage of PaaS traditionally has been vendor lock-in



source : Microsoft

# Software as a Service (SaaS)

- Is centrally hosted and available without local installation or provisioning
- Often is hosted on top of PaaS as a platform for deploying software
- Provides services to run and connect with existing software
- Often industry and vertical specific
- Often licensed and typically provides a client/server model
- Most modern SaaS offerings use web-based apps for the client
- Companies typically consider SaaS as a business solution to license offerings
- Most SaaS solutions are built on IaaS, PaaS, and/or Serverless back ends



Source:brainwire

Reference:

Serverless apps Architecture patterns and Azure implementation  
By Microsoft



# Thank You!

In our next session:



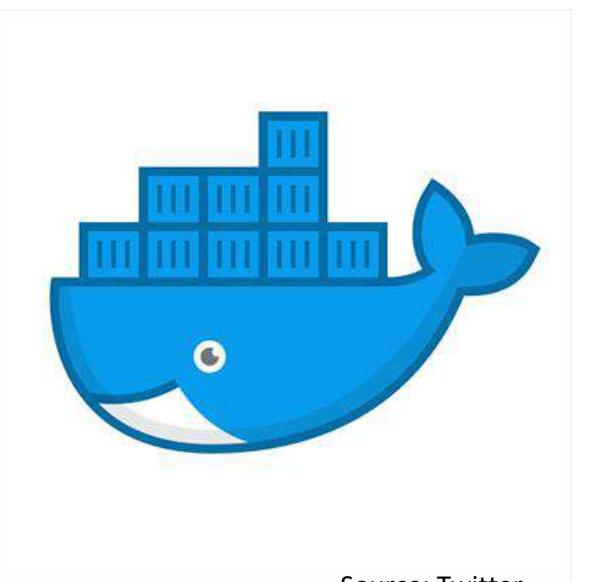
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Architecture deployment approaches - II

Chandan Ravandur N

# Containers

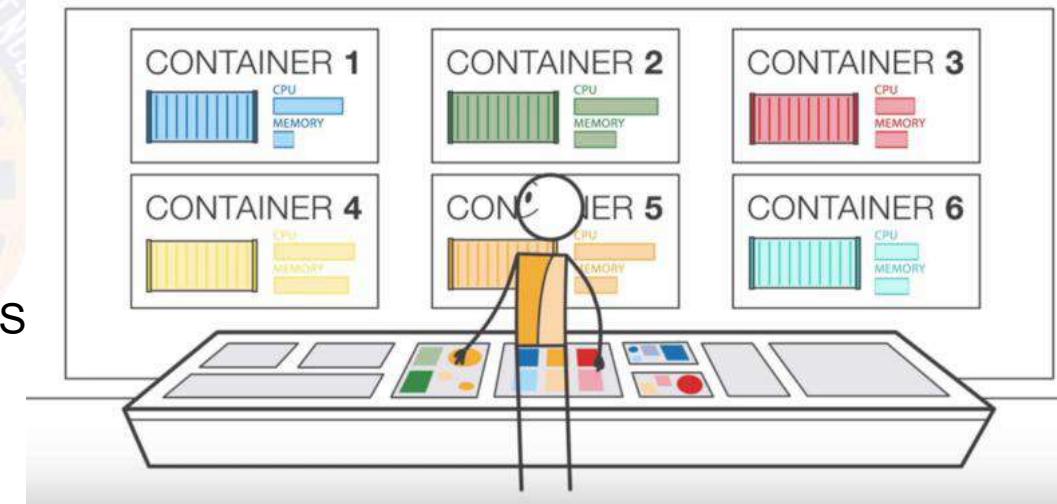
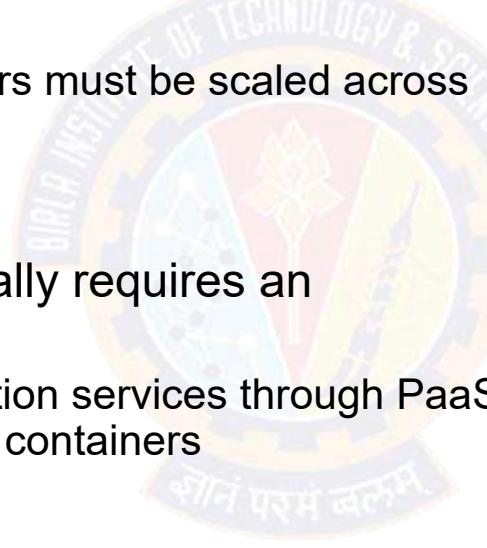
- Containers are an interesting solution that enables PaaS-like benefits without the IaaS overhead
- A container is essentially a runtime that contains the bare essentials needed to run a unique application
- The kernel or core part of the host operating system and services such as storage are shared across a host
- The shared kernel enables containers to be lightweight
  - ✓ some are mere megabytes in size, compared to the gigabyte size of typical virtual machines
- With hosts already running, containers can be started quickly, facilitating high availability
- The ability to spin up containers quickly also provides extra layers of resiliency
- Docker is one of the more popular implementations of containers.
- Benefits of containers include:
  - ✓ Lightweight and portable
  - ✓ Self-contained so no need to install dependencies
  - ✓ Provide a consistent environment regardless of the host (runs exactly same on a laptop as on a cloud server)
  - ✓ Can be provisioned quickly for scale-out
  - ✓ Can be restarted quickly to recover from failure



Source: Twitter

# Container as a Service

- A container runs on a container host (that in turn may run on a bare metal machine or a virtual machine)
  - ✓ Multiple containers or instances of the same containers may run on a single host
  - ✓ For true failover and resiliency, containers must be scaled across hosts
- Managing containers across hosts typically requires an orchestration tool such as Kubernetes
  - ✓ Many cloud providers provide orchestration services through PaaS solutions to simplify the management of containers
- Containers as a service (CaaS) is a cloud service model that allows users to upload, organize, start, stop, scale and otherwise manage containers, applications and clusters
  - ✓ enables these processes by using either a container-based virtualization, an application programming interface (API) or a web portal interface



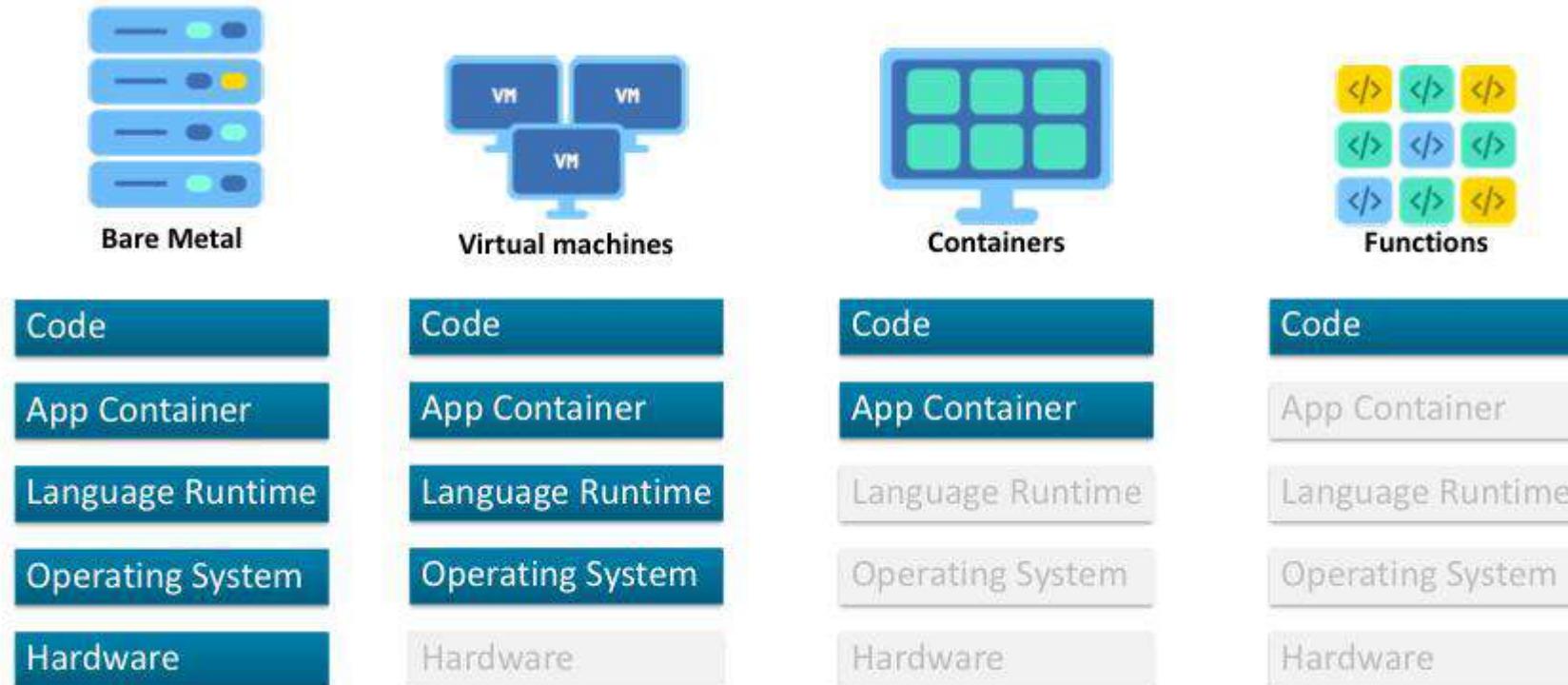
source:freecodecamp

# Functions as a Service (FaaS)

- A category of cloud computing services that provides a platform allowing customers to develop, run, and manage application functionalities
  - ✓ without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app
- Building an application following this model is one way of achieving a "serverless" architecture
  - ✓ typically used when building microservices applications
- Functions as a Service (FaaS) is a specialized container service that is similar to serverless
  - ✓ A specific implementation of FaaS, called OpenFaaS, sits on top of containers to provide serverless capabilities
  - ✓ OpenFaaS provides templates that package all of the container dependencies necessary to run a piece of code
  - ✓ Using templates simplifies the process of deploying code as a functional unit
  - ✓ Although it provides serverless functionality, it specifically requires you to use Docker and an orchestrator



# Evaluation of Cloud Services



source : Oracle blog

Reference:

Serverless apps Architecture patterns and Azure implementation  
By Microsoft



# Thank You!

In our next session:



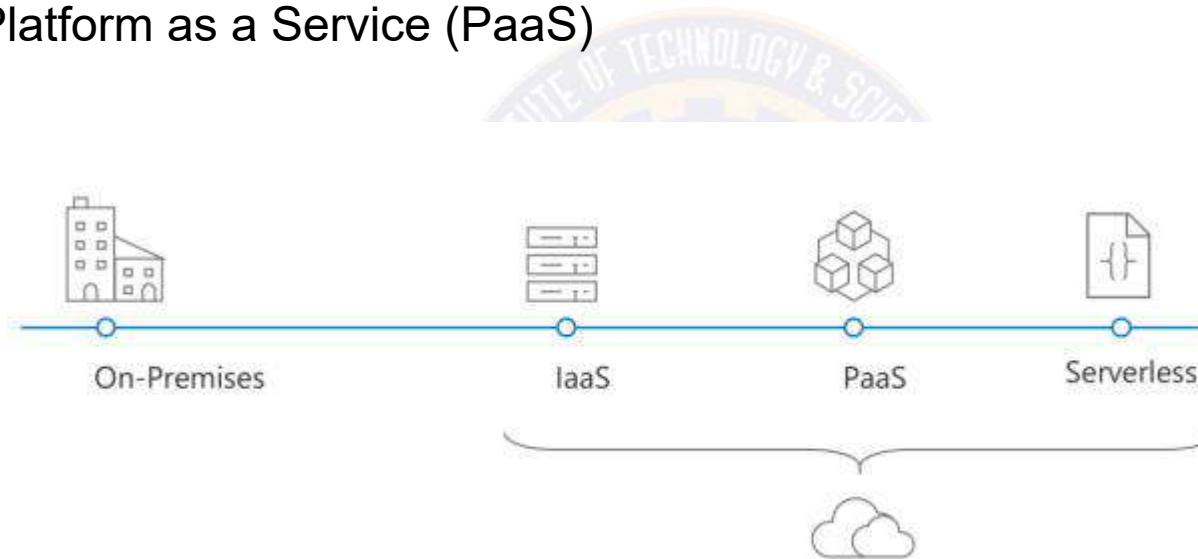
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Serverless Architecture

Chandan Ravandur N

# Evolution of cloud platforms

- Serverless is the culmination of several iterations of cloud platforms
- The evolution began with physical metal in the data center and progressed through Infrastructure as a Service (IaaS) and Platform as a Service (PaaS)



# Going Serverless

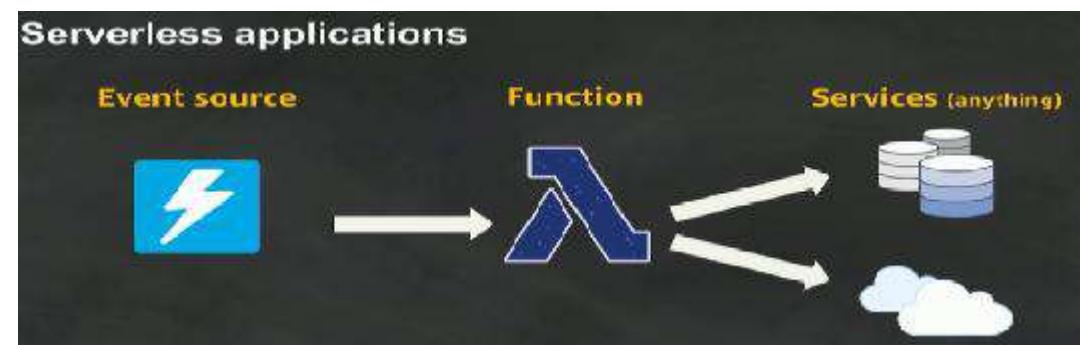
## Serverless

- Is the evolution of cloud platforms in the direction of pure cloud native code
- Brings developers closer to business logic while insulating them from infrastructure concerns
- A pattern that doesn't imply "no server" but rather, "less server"
- Serverless code is event-driven
  - ✓ Code may be triggered by anything from a traditional HTTP web request to a timer or the result of uploading a file
- The infrastructure behind Serverless allows for instant scale to meet elastic demands
  - ✓ offers micro-billing to truly "pay for what you use"
- Serverless requires a new way of thinking and approach to building applications and isn't the right solution for every problem!

# Serverless

- A Serverless architecture provides a clear separation between the code and its hosting environment
  - ✓ You implement code in a function that is invoked by a trigger
  - ✓ After that function exits, all its needed resources may be freed
  - ✓ The trigger might be manual, a timed process, an HTTP request, or a file upload
  - ✓ The result of the trigger is the execution of code
- Although Serverless platforms vary, most provide access to pre-defined APIs and bindings to streamline tasks such as writing to a database or queueing results
  - ✓ Serverless is an architecture that relies heavily on abstracting away the host environment to focus on code
  - ✓ Container solutions provide developers existing build scripts to publish code to Serverless-ready images

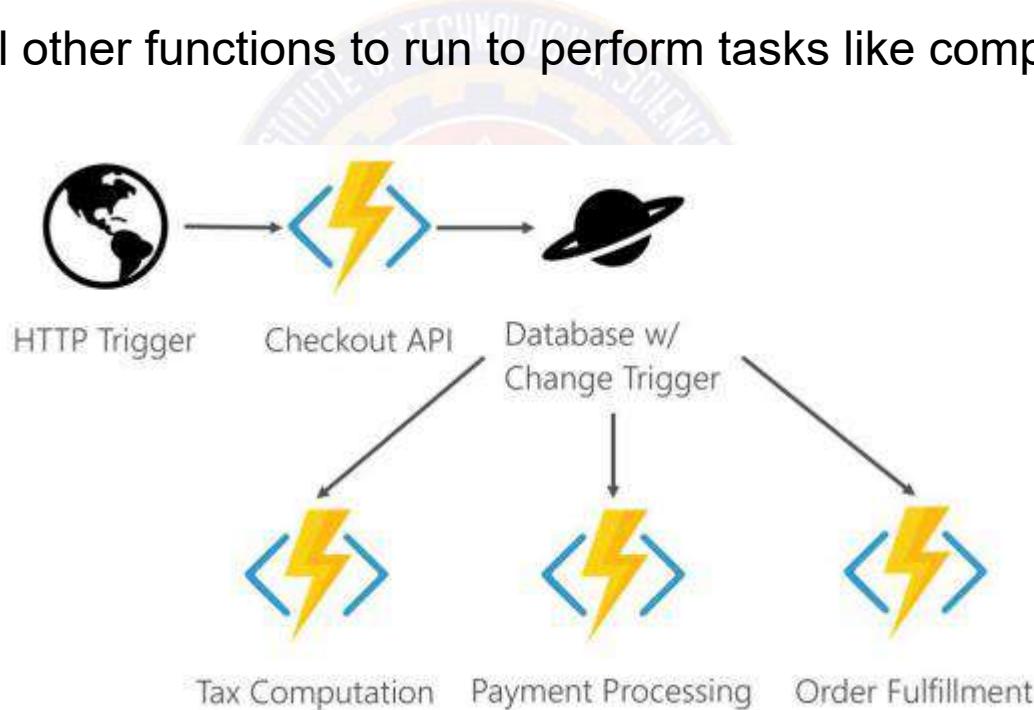
source : AWS



- It can be thought of as less server!

# Serverless components

- An HTTP request causes the Checkout API code to run
- The Checkout API inserts code into a database
- The insert triggers several other functions to run to perform tasks like computing tasks and fulfilling the order.



source : Microsoft

# Advantages of Serverless

- High density
  - ✓ Many instances of the same serverless code can run on the same host compared to containers or virtual machines
  - ✓ The instances scale across multiple hosts scale out and resiliency
- Micro-billing
  - ✓ Most serverless providers bill based on serverless executions, enabling massive cost savings in certain scenarios
- Instant scale
  - ✓ Serverless can scale to match workloads automatically and quickly
- Faster time to market
  - ✓ Developers focus on code and deploy directly to the serverless platform
  - ✓ Components can be released independently of each other



source : DZone

# Summarized

	IaaS	PaaS	Container	Serverless
Scale	VM	Instance	App	Function
Abstracts	Hardware	Platform	OS Host	Runtime
Unit	VM	Project	Image	Code
Lifetime	Months	Days to Months	Minutes to Days	Milliseconds to Minutes
Responsibility	Applications, dependencies, runtime, and operating system	Applications and dependencies	Applications, dependencies, and runtime	Function

- **Scale** refers to the unit that is used to scale the application
- **Abstracts** refers to the layer that is abstracted by the implementation
- **Unit** refers to the scope of what is deployed
- **Lifetime** refers to the typical runtime of a specific instance
- **Responsibility** refers to the overhead to build, deploy, and maintain the application

source : Microsoft

Reference:

Serverless apps Architecture patterns and Azure implementation  
By Microsoft



# Thank You!

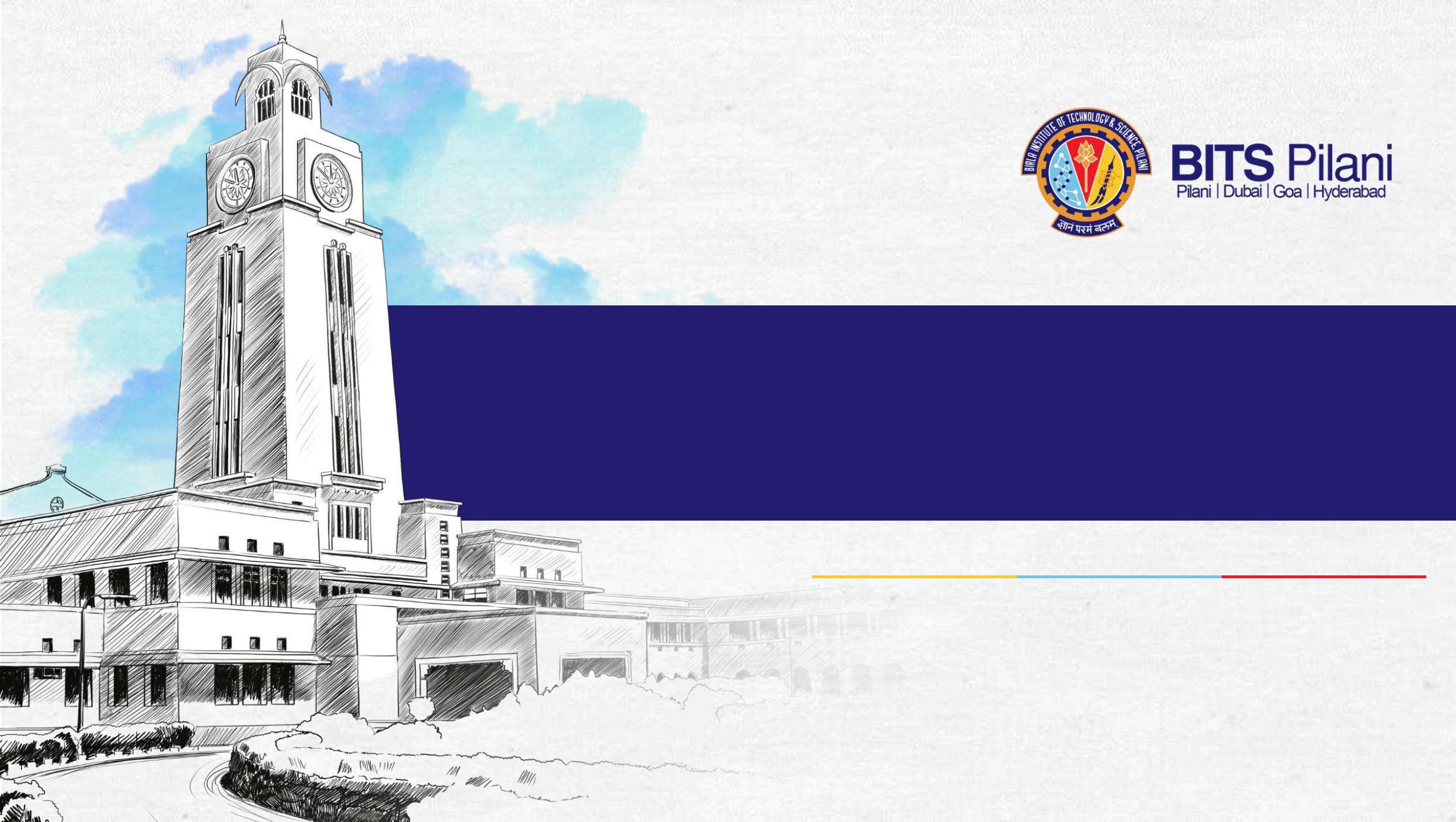
In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

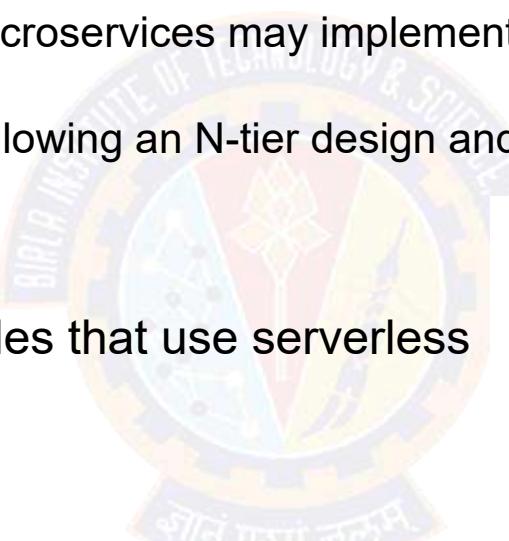
# Serveless App Examples

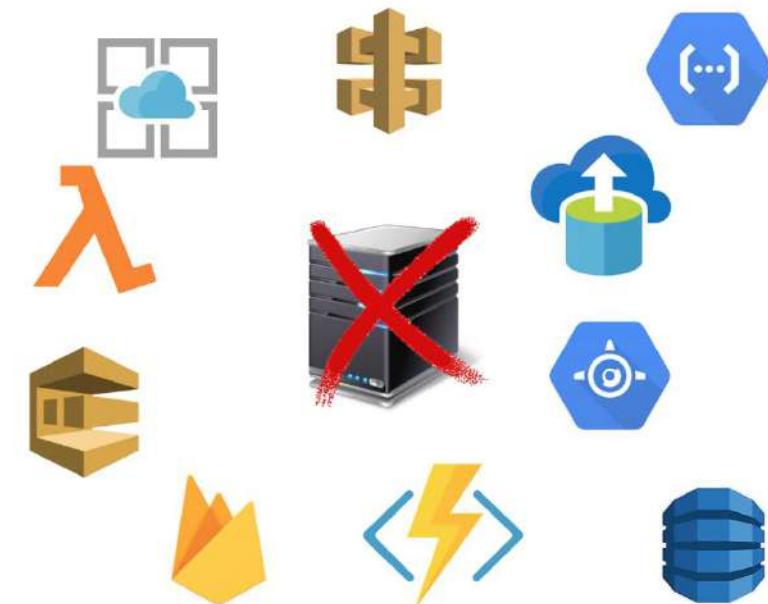
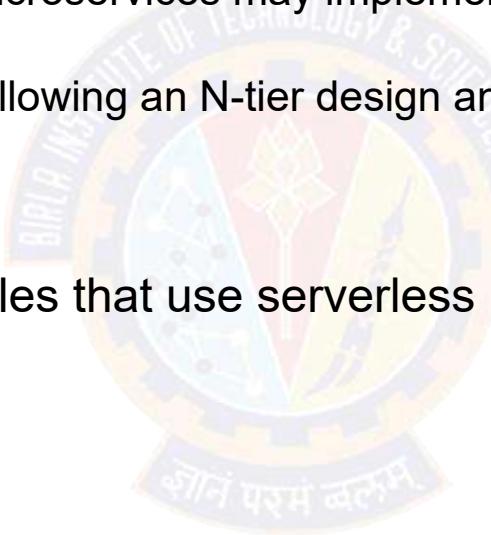
Chandan Ravandur N



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Common architecture examples for Serverless

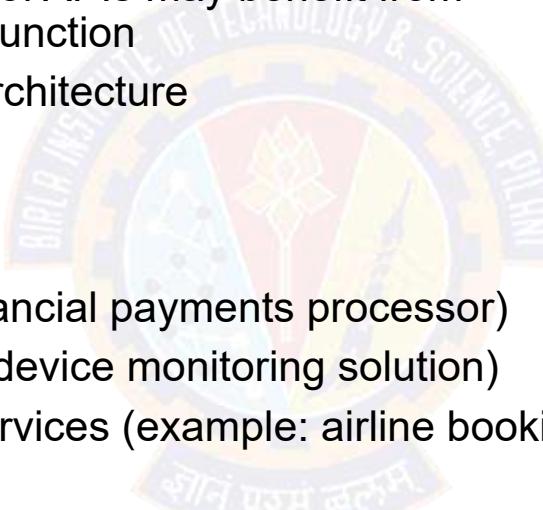
- Many approaches to using serverless architectures
    - ✓ Some projects may benefit from taking an “all-in” approach to serverless
    - ✓ Applications that rely heavily on microservices may implement all microservices using serverless technology
    - ✓ The majority of apps are hybrid, following an N-tier design and using serverless for the components that make sense
  - Some common architecture examples that use serverless
    - ✓ Full serverless back end
    - ✓ Monoliths and “starving the beast”
    - ✓ Web apps
    - ✓ Mobile back ends
    - ✓ Internet of Things (IoT)



source : medium

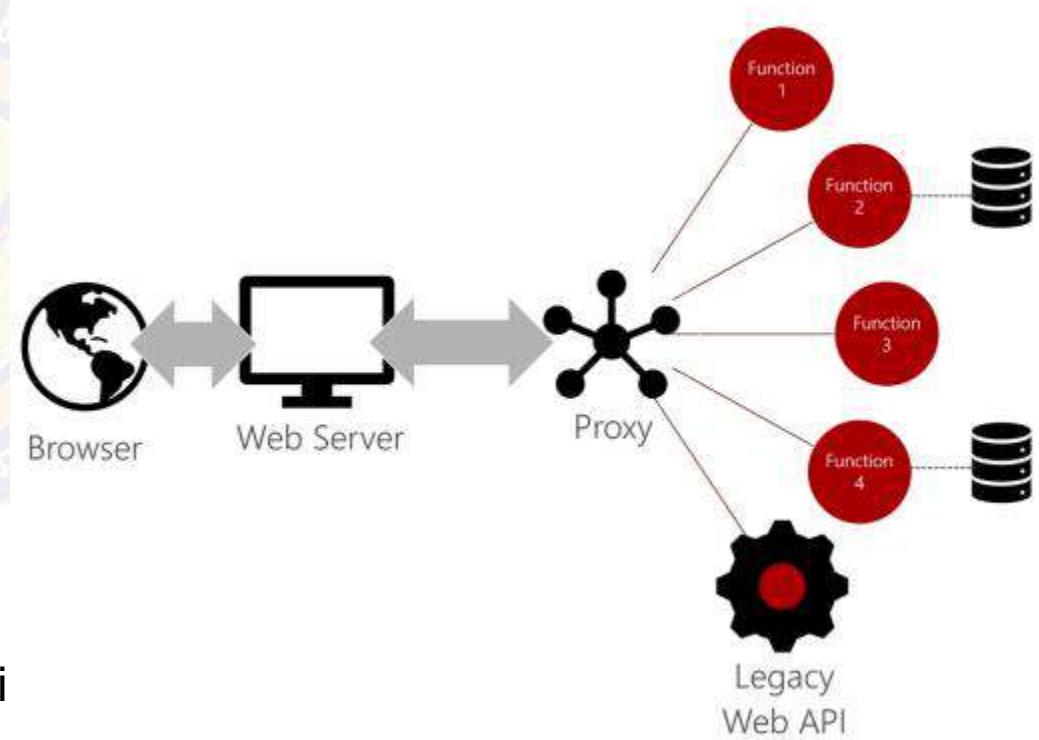
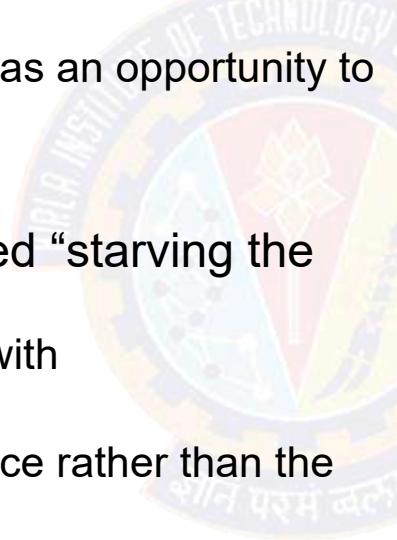
# Full Serverless back end

- Ideal for several types of scenarios, especially when building new or “green field” applications
  - ✓ An application with a large surface area of APIs may benefit from implementing each API as a serverless function
  - ✓ Apps that are based on microservices architecture
- Specific scenarios include:
  - ✓ API-based SaaS products (example: financial payments processor)
  - ✓ Message-driven applications (example: device monitoring solution)
  - ✓ Apps focused on integration between services (example: airline booking application)
  - ✓ Processes that run periodically (example: timer-based database clean-up)
  - ✓ Apps focused on data transformation (example: import triggered by file upload)
  - ✓ Extract Transform and Load (ETL) processes



# Monoliths and “starving the beast”

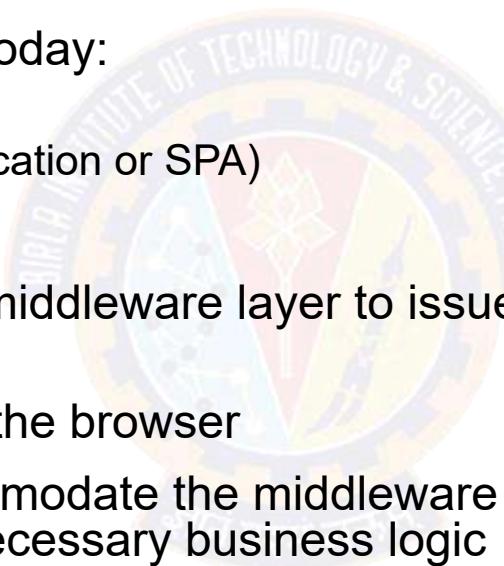
- A common challenge is migrating an existing monolithic application to the cloud
  - ✓ The least risky approach is to “lift and shift” entirely onto virtual machines
  - ✓ Many shops prefer to use the migration as an opportunity to modernize their code base
- A practical approach to migration is called “starving the beast”
  - ✓ the monolith is migrated “as is” to start with
  - ✓ then, selected services are modernized
  - ✓ clients are updated to use the new service rather than the monolith endpoint
  - ✓ eventually, all clients are migrated onto the new services
- The monolith is “starved” (its services no longer called) until all functionality has been replaced
  - ✓ The combination of serverless and proxies can facilitate much of this migration



source : Microsoft

# Web apps

- Web apps are great candidates for serverless applications
- Two common approaches to web apps today:
  - ✓ server-driven
  - ✓ client-driven (such as Single Page Application or SPA)
- Server-driven web apps typically use a middleware layer to issue API calls to render the web UI
- SPA make REST API calls directly from the browser
- In both scenarios, serverless can accommodate the middleware or REST API request by providing the necessary business logic
- A common architecture is to stand up a lightweight static web server
  - ✓ The Single Page Application (SPA) serves HTML, CSS, JavaScript, and other browser assets
  - ✓ The web app then connects to a microservices back end



source : peerbits

# Mobile back ends

- The event-driven paradigm of serverless apps makes them ideal as mobile back ends
- The mobile device triggers the events and the serverless code executes to satisfy requests
- Taking advantage of a serverless model enables developers to enhance business logic without having to deploy a full application update
- The serverless approach also enables teams to share endpoints and work in parallel
- Mobile developers can build business logic without becoming experts on the server side
- Serverless abstracts the server-side dependencies and enables the developer to focus on business logic
- For example,
  - ✓ A mobile developer who builds apps using a JavaScript framework can build serverless functions with JavaScript as well
  - ✓ The serverless host manages the operating system, a Node.js instance to host the code, package dependencies, and more
  - ✓ The developer is provided a simple set of inputs and a standard template for outputs
  - ✓ They then can focus on building and testing the business logic.

# Internet of Things (IoT)

- IoT refers to physical objects that are networked together aka “connected devices” or “smart devices.”
- Everything from cars and vending machines may be connected and send information
  - ✓ ranging from inventory to sensor data such as temperature and humidity
- For example,
  - ✓ In the enterprise, IoT provides business process improvements through monitoring and automation
  - ✓ IoT data may be used to regulate the climate in a large warehouse or track inventory through the supply chain
  - ✓ IoT can sense chemical spills and call the fire department when smoke is detected.
- Serverless is an ideal solution for several reasons:
  - ✓ Enables scale as the volume of devices and data increases
  - ✓ Accommodates adding new endpoints to support new devices and sensors
  - ✓ Facilitates independent versioning so developers can update the business logic for a specific device without having to deploy the entire system
  - ✓ Resiliency and less downtime
  - ✓ Automates tasks such as device registration, policy enforcement, tracking, and even deployment of code to devices at the edge



source : medium

Reference:

Serverless apps Architecture patterns and Azure implementation  
By Microsoft



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Serverless Design Patterns

Chandan Ravandur N

# Serverless Patterns

- Many design patterns exists for Serverless
- Commonality is the fundamental combination of an event trigger and business logic
- Patterns
  - ✓ Scheduling
  - ✓ Command and Query Responsibility Segregation (CQRS)
  - ✓ Event-based processing
  - ✓ File triggers and transformations
  - ✓ Web apps and APIs
  - ✓ Data pipeline
  - ✓ Stream processing
  - ✓ API gateway



source : Serverlesslife

# Scheduling

- Scheduling tasks is a common function
- Diagram shows a legacy database that doesn't have appropriate integrity checks
- The database must be scrubbed periodically
- The serverless function finds invalid data and cleans it
- The trigger is a timer that runs the code on a schedule

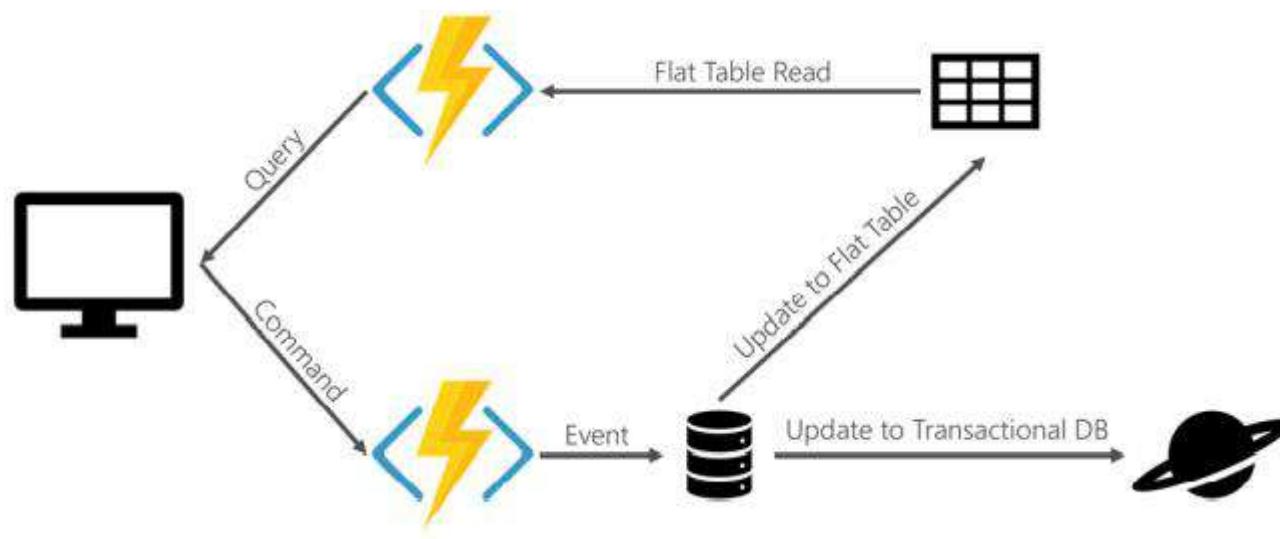


Source : Microsoft

# Command and Query Responsibility Segregation (CQRS)

- A pattern that provides different interfaces for reading (or querying) data and operations that modify data
- Problems in Older system
  - ✓ In traditional Create Read Update Delete (CRUD) based systems, conflicts can arise from high volume of both reads and writes to the same data store
  - ✓ Locking may frequently occur and dramatically slow down reads
  - ✓ Often, data is presented as a composite of several domain objects and read operations must combine data from different entities
- Using CQRS
  - ✓ Read might involve a special “flattened” entity that models data the way it’s consumed
  - ✓ Read is handled differently than how it’s stored
- For example,
  - ✓ Database may store a contact as a header record with a child address record
  - ✓ Read could involve an entity with both header and address properties
  - ✓ It might be materialized from views
  - ✓ Update operations could be encapsulated as isolated events that then trigger updates to two different models
  - ✓ Separate models exist for reading and writing

# Command and Query Responsibility Segregation (CQRS) -2



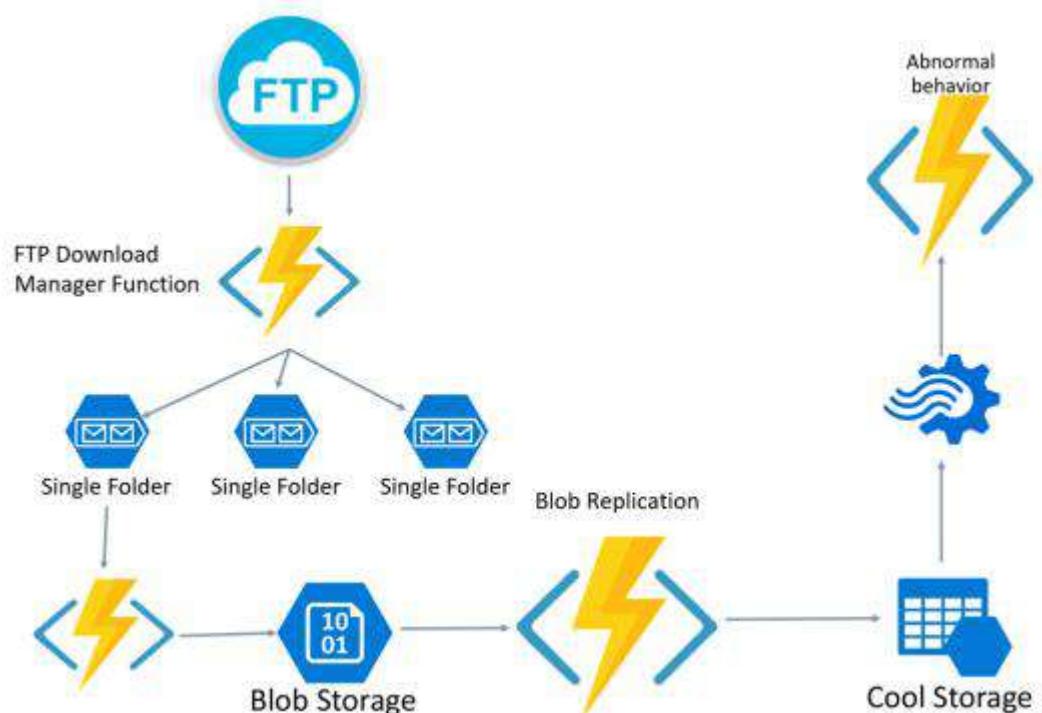
Source : Microsoft

# Event-based processing

- Events are informational messages
  - In message-based systems, events are often collected in queues or publisher/subscriber topics to be acted upon
  - These events can trigger Serverless functions to execute a piece of business logic
- 
- Example of event-based processing is event-sourced systems
  - An “event” is raised to mark a task as complete
  - A Serverless function triggered by the event updates the appropriate database document
  - A second Serverless function may use the event to update the read model for the system

# File triggers and transformations

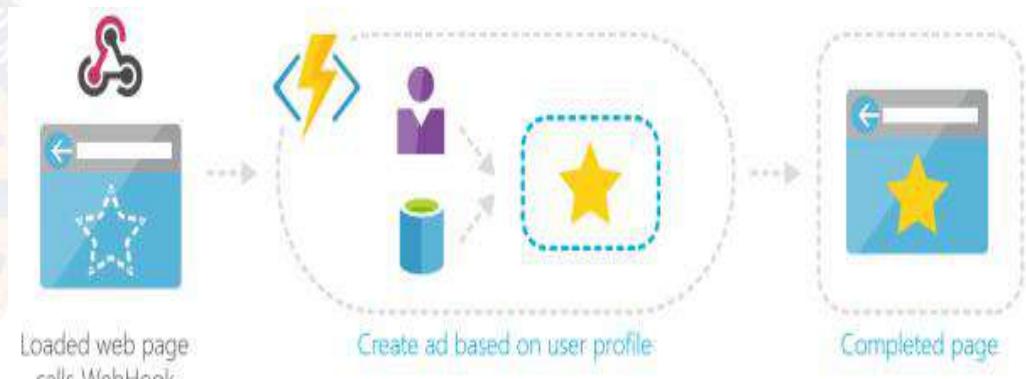
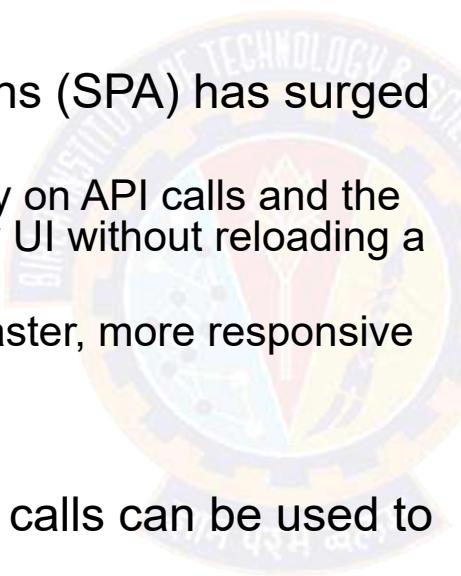
- Extract, Transform, and Load (ETL) is a common business function
- Serverless is a great solution for ETL because it allows code to be triggered as part of a pipeline
- Individual code components can address various aspects
  - ✓ One Serverless function may download the file
  - ✓ Another applies the transformation
  - ✓ Another loads the data
- The code can be tested and deployed independently, making it easier to maintain and scale where needed.



Source : Microsoft

# Web apps and APIs

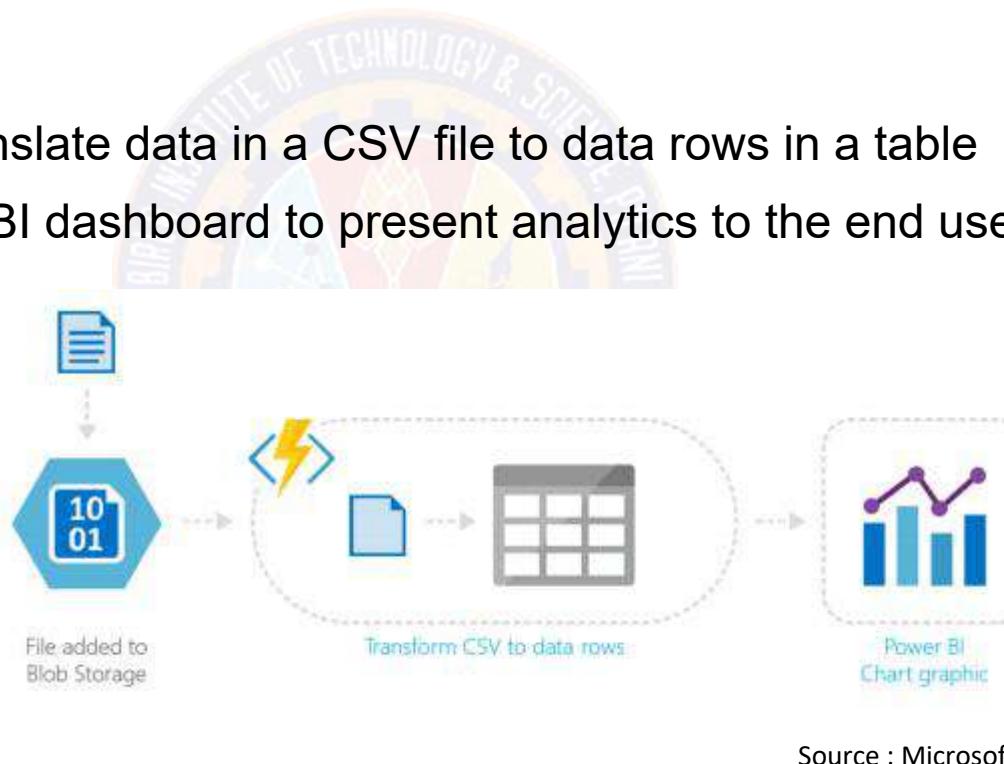
- A popular scenario for serverless is N-tier applications
  - ✓ most commonly ones where the UI layer is a web app
- The popularity of Single Page Applications (SPA) has surged recently
  - ✓ SPA apps render a single page, then rely on API calls and the returned data to dynamically render new UI without reloading a full page
  - ✓ Client-side rendering provides a much faster, more responsive application to the end user
- Serverless endpoints triggered by HTTP calls can be used to handle the API requests
- For example,
  - ✓ Ad services company may call a serverless function with user profile information to request custom advertising
  - ✓ The serverless function returns the custom ad and the web page renders it



Source : Microsoft

# Data pipeline

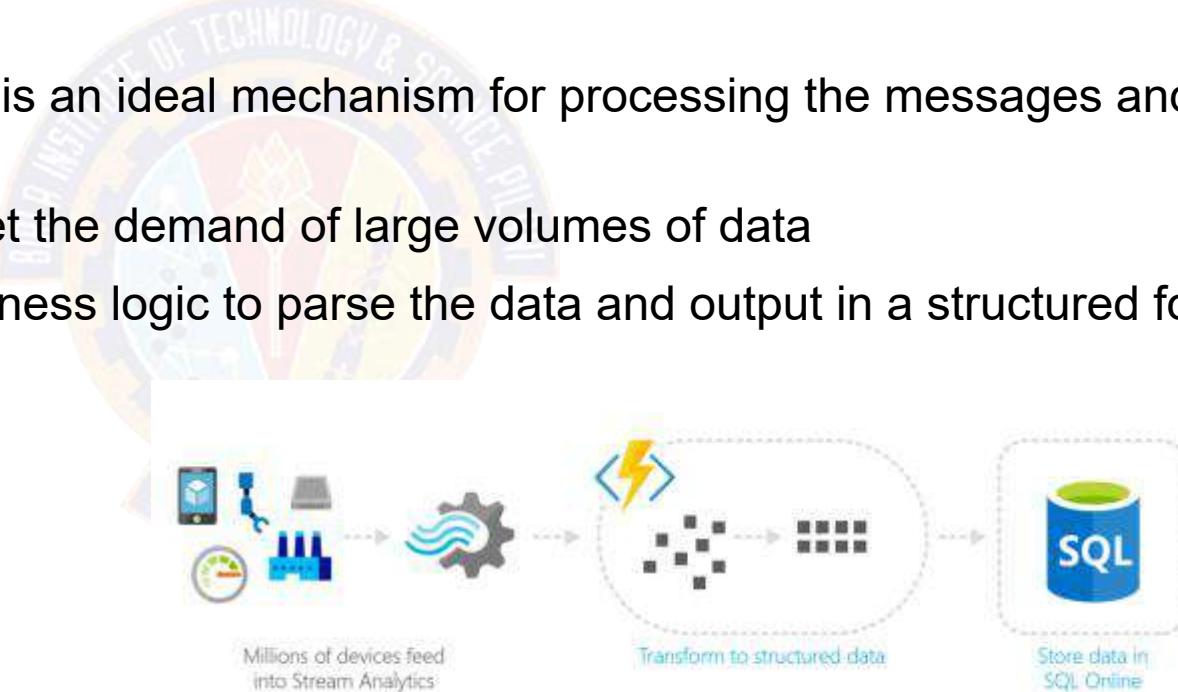
- Serverless functions can be used to facilitate a data pipeline
- For example,
- A file triggers a function to translate data in a CSV file to data rows in a table
- The organized table allows a BI dashboard to present analytics to the end user



Source : Microsoft

# Stream processing

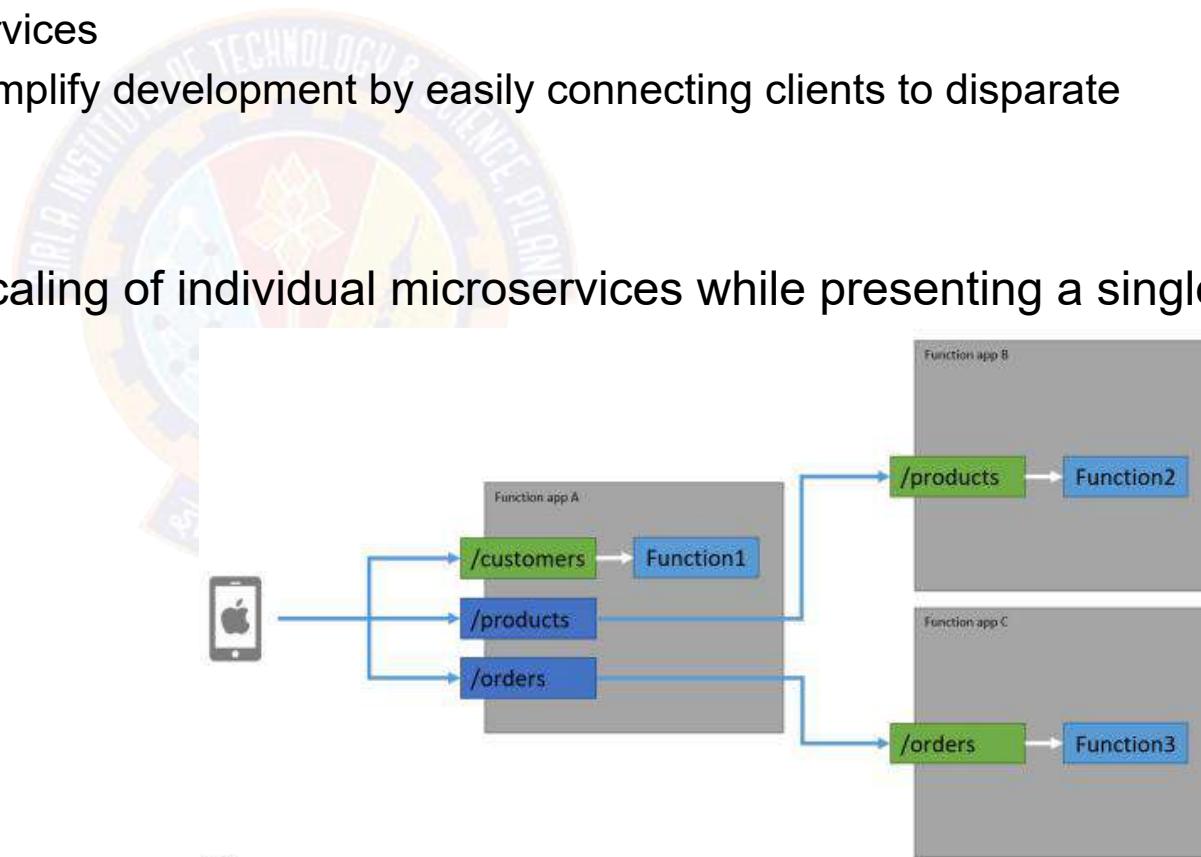
- Devices and sensors often generate streams of data that must be processed in real time
- There are a number of technologies that can capture messages and streams from Event Hubs and IoT Hub to Service Bus
- Regardless of transport, serverless is an ideal mechanism for processing the messages and streams of data as they come in
- Serverless can scale quickly to meet the demand of large volumes of data
- The serverless code can apply business logic to parse the data and output in a structured format for action and analytics



Source : Microsoft

# API gateway

- An API gateway provides a single point of entry for clients
  - ✓ then intelligently routes requests to back-end services
  - ✓ useful to manage large sets of services
  - ✓ can also handle versioning and simplify development by easily connecting clients to disparate environments
- Serverless can handle back-end scaling of individual microservices while presenting a single front end via an API gateway



Source : Microsoft

Reference:

Serverless apps Architecture patterns and Azure implementation  
By Microsoft



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Low code development

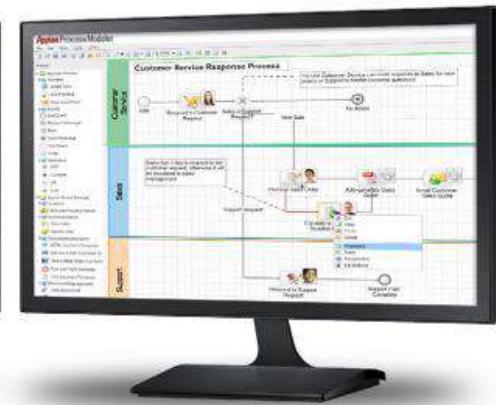
Chandan Ravandur N

# What is low code?

- Low-code development is a way to build apps more quickly by reducing the need to code
- According to Forrester Research, low-code development platforms:

“...enable rapid application delivery with minimal hand-coding, and quick setup and deployment.”

- Low-code development has evolved to take advantage of visual design tools like
  - ✓ drag-and-drop modelers
  - ✓ point-and-click interface creation
- —to enable the rapid creation, launch, use and change of powerful business apps



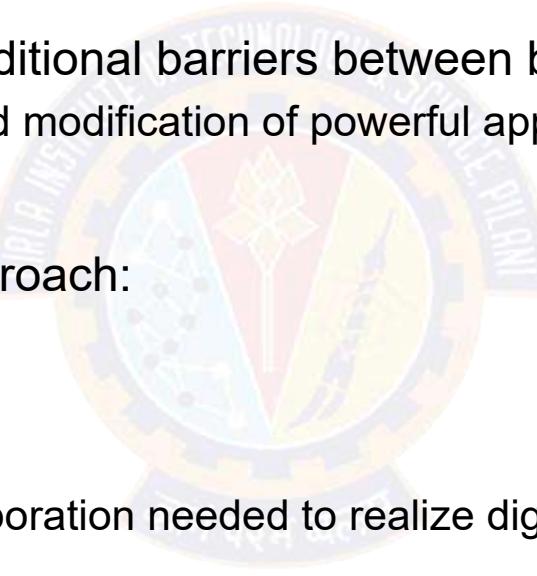
# But why did low-code come about?

- In a word, mobile...
  - ✓ The explosion of mobile, and the resulting change in consumer and (even) employee expectations
  - ✓ The sheer demand for digital services is growing faster than ever, and it does not appear to be slowing anytime soon
- And considering the hundreds, if not thousands of disjointed processes, systems, apps, solutions, etc. at any given organization
  - ✓ not surprising that even brands considered to be leaders struggle to keep pace.
- So, how do you get ahead of the new digital expectations?
  - ✓ And how can you possibly stay there, with the incredible pace of change?
  - ✓ What if your IT organization could deliver solutions FASTER with FEWER RESOURCES?

With a low-code development platform it's possible. Low-code development platforms offer speedy, iterative delivery of new business applications. So you can build great apps. Innovate faster. And run smarter.

# LOW-CODE: TRANSFORM IDEAS TO INNOVATION

- The fastest way to transform ideas into innovation
- These platforms break down the traditional barriers between business and IT
  - ✓ allowing the rapid build, launch, and modification of powerful apps
- This model-driven development approach:
  - ✓ Speeds app creation
  - ✓ Unites legacy systems
  - ✓ Gets ahead of Shadow IT
  - ✓ Fosters the agile Business/IT collaboration needed to realize digital transformation's massive potential



The ability to quickly build, deploy, and evolve business applications is what separates digital leaders from those left-behind. This is low-code development...jet fuel for your digital transformation efforts.

# DO I NEED LOW-CODE?

## Signs where you could benefit from using a low-code development platform:

- Keeping up with demands from the business is difficult
  - ✓ IT organization is constantly slammed with demands from the larger organization
  - ✓ The IT backlog is large... and perpetually growing. IT is falling behind.
- Reliance on legacy apps
  - ✓ Legacy applications drain efficiency... and your IT resources
  - ✓ They keep talented IT resources in a continual state of updates and fixes
- More time spent on maintenance than innovation
  - ✓ most IT teams spend nearly 80% of their time on maintenance, and only 20% on new innovation
  - ✓ Too little time focused on innovative solutions leads to ...
- Shadow IT
  - ✓ Employees don't wait for IT
  - ✓ They're creating their own solutions—that are not a part of your architecture—in a world of Shadow IT that adds even more complexity to your business
- Scarce development resources
  - ✓ You urgently need top-notch software developers
  - ✓ But it's getting increasingly harder to find and retain them

# Key features of low-code



## Visual Modeling

Application development is expedited with visual representations of processes. These visual models are easier to understand than traditional displays. Which allows citizen developers to grasp application design easily.



## Drag-and-drop interfaces

Typing out long strands of code to produce is not only difficult, but also extremely time consuming. Low-code allows simple drag-and-drop so developers can create applications visually, resulting in faster time-to-launch.



## No-code options

No-code means just that...zero code required. Empower citizen developers to quickly transform ideas into business apps...with no-code app-building functionality.

# Key features of low-code (2)



## Agile development

Accelerate time to value by rapidly creating and launching applications...then enhance and expand them over time. Low-code development means you can iterate apps, and release them as soon as functionality is built. Since change is so fast with low-code development, agile transformation is made easier.



## Instant mobility

Build once, deploy everywhere. With the explosion of mobile devices like smart phones and tablets, applications must have cross-platform functionality standard in their design. With true low-code development, it should all happen behind the scenes automatically, with no extra effort, coding, or resources.



## Declarative Tools

With low-code software, declarative tools are implemented through visual models and business rules. Removing the need to write custom-coding for these mitigates the difficulty of future changes or additions. And speeds development times.

Reference:  
Appian Low code Guide



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Developing Low Code Apps

Chandan Ravandur N

# Turn your business ideas into apps

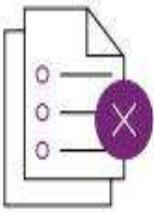
## no coding experience required

Have you ever thought, "I wish we had an app for that?" You're not alone.

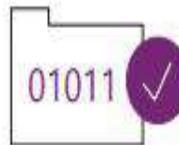
There's rapidly growing demand around the world for apps that can:



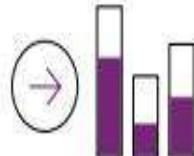
Support better customer  
and employee experiences.



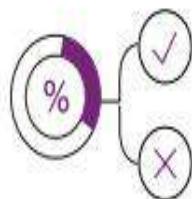
Eliminate paper-based  
and manual processes.



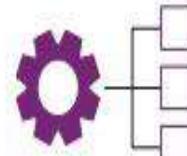
Keep databases and  
files up to date.



Democratize dashboards  
and analytics.



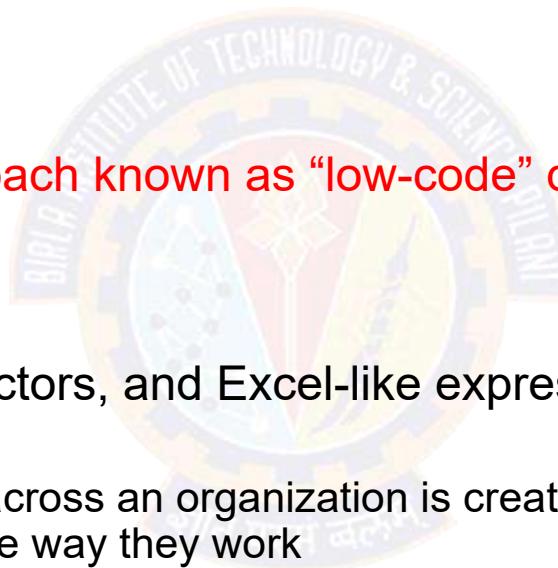
Enable data-based  
decision-making.



Automate repetitive tasks to  
free up more time to focus  
on strategic work.

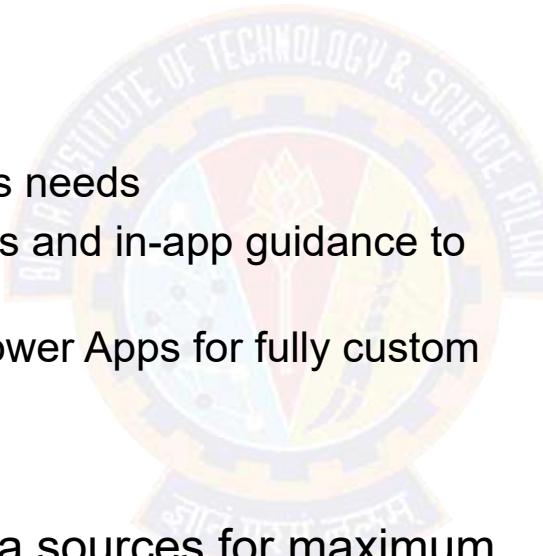
# Developing Low Code Apps

- You might think that building apps like that would require teams of developers to write the code line by line
  - ✓ In the past, it would have!
- But today, with an innovative approach known as “low-code” development, you can build them yourself!
- Using existing data, prebuilt connectors, and Excel-like expressions, nontechnical workers are becoming app developers
  - ✓ Democratizing app development across an organization is creating opportunities for individuals and organizations alike to transform the way they work
  - ✓ These new apps are becoming part of the fabric of the workday across organizations of all kinds.



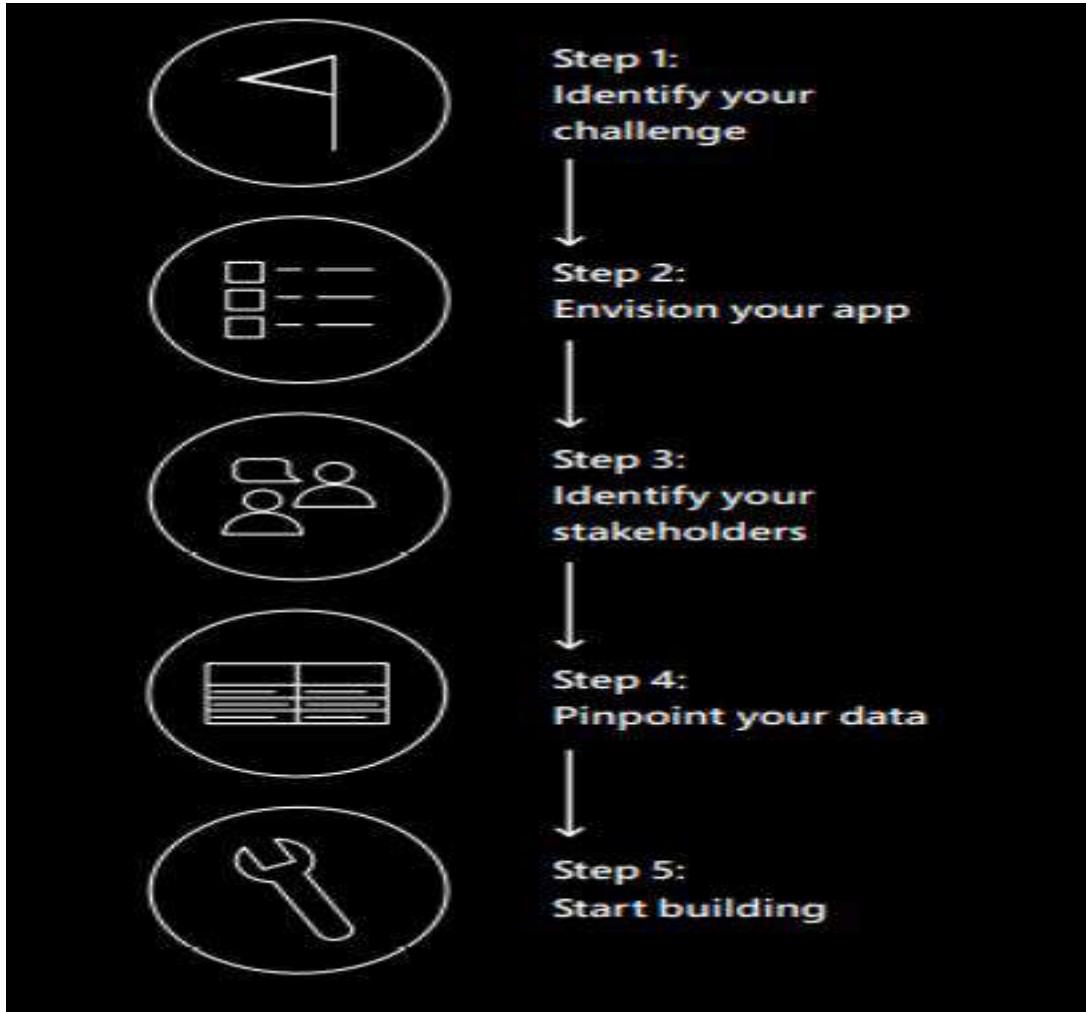
# Citizen developer

- Ready to make your business ideas a reality?
- With LCAP, you can
  - ✓ rapidly automate processes
  - ✓ build custom apps to meet your business needs
  - ✓ easy to start small with prebuilt templates and in-app guidance to deliver quick wins
  - ✓ if needed, pro developers can extend Power Apps for fully custom solutions
- Solutions connect to a wide range of data sources for maximum flexibility
- Apps you build work seamlessly across the web, iOS, and Android devices



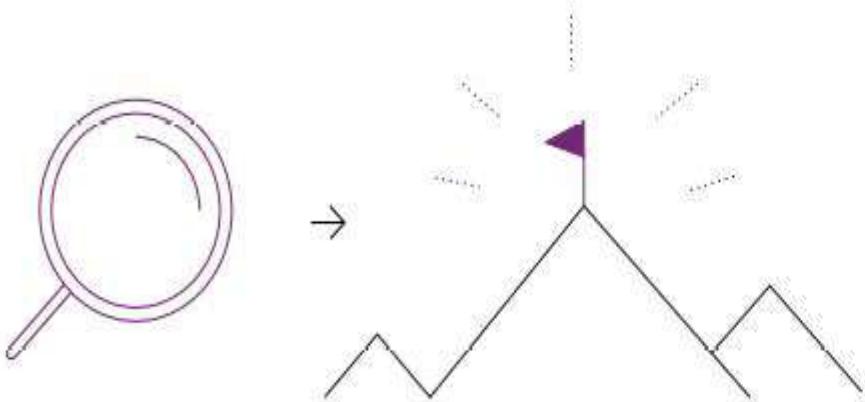
Source : youtube

# Get started with low-code apps in 5 simple steps



# Step 1: Identify your challenge

To be successful out of the gate, it helps to clearly define the business challenge you're trying to solve—and to keep it simple. Choose a process you know well: how it works, who's involved, and what is impacted downstream. It also helps to have a business outcome in mind. Faster process? Better collaboration? Real-time visibility? Knowing the end goal will help make the app you create effective.

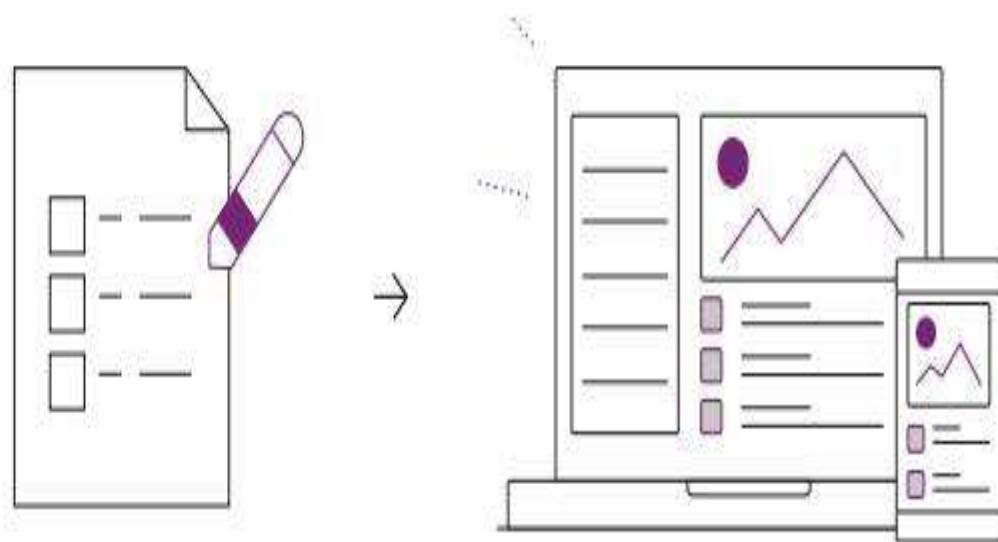


## Common uses for Power Apps

- ✓ Maintenance and repair processes
- ✓ Project management
- ✓ Proposal creation and workflows
- ✓ Incident reporting
- ✓ Training management
- ✓ Resource scheduling
- ✓ Asset tracking
- ✓ Quality control
- ✓ Appointment scheduling
- ✓ Customer experience management
- ✓ Interactive dashboards

## Step 2: Envision your app

Take time to define your vision for the app. It doesn't have to be elaborate—just a few simple bullets about what you want it to do, who will use it, and what the experience will be like, along with a mockup of how the app could be structured. Getting a clear vision in your mind before you start will help you identify the functionality required to bring your app to life. You can update this document as you go through the process to further refine your ideas.



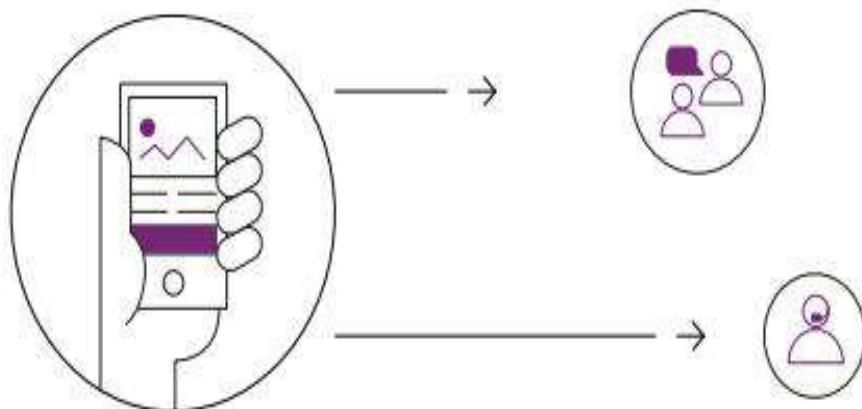
# Step 3: Identify your stakeholders

The great thing about low-code apps is that they can be built by the same people who will use them. That means you and your team are likely key stakeholders, but who else will need to be involved? How will the app affect their daily work? What devices are they likely to use? Considering their needs early avoids surprises down the road and helps you deliver the greatest value.

Here are some starting points:

- ✓ **App user:** Who will interact with the application directly?
- ✓ **Data user:** Who controls or uses data related to the app?

- ✓ **Customer:** How will the application affect the customer experience?



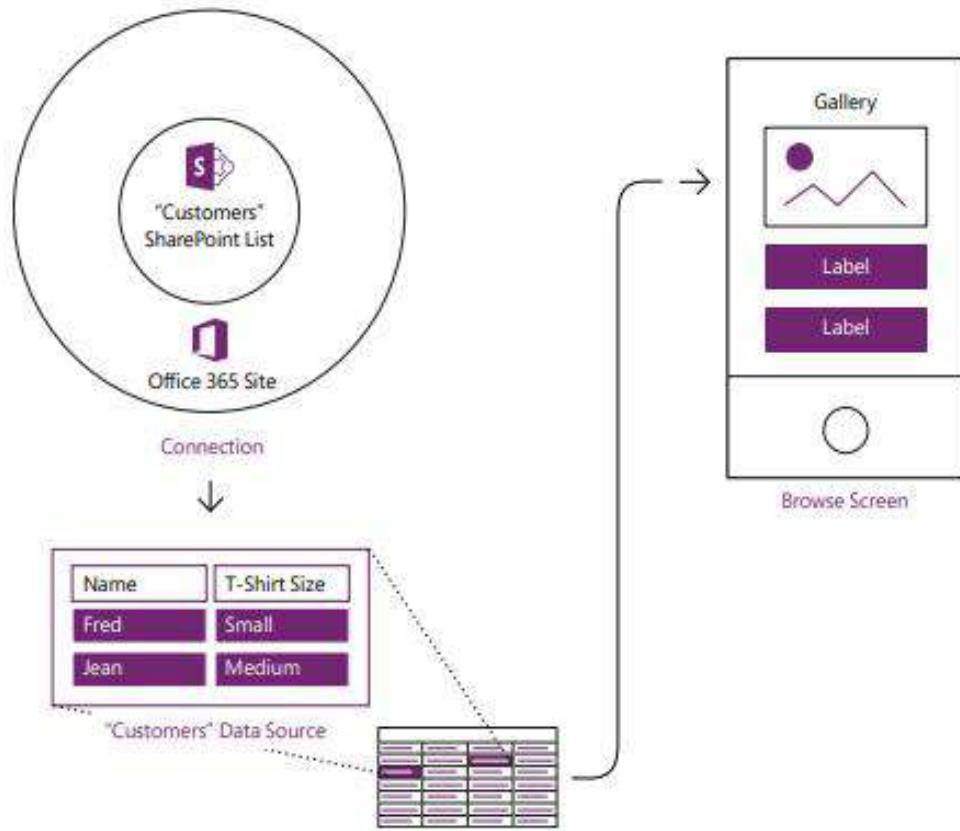
You don't have to address the needs of every stakeholder all at once. In fact, you'll want to start small. However, it's best to understand early on who could benefit from your app or provide valuable input along the way.

# Step 4: Pinpoint your data

Apps rely on data to do their work. They can access existing data in a location such as SharePoint or a database. They can also be used to capture data, ranging from text to GPS location to video. If your app relies on external data, where will you source it? If you're collecting data, where will you store it? Knowing where data will live and how you'll manage it will help make your app a success.

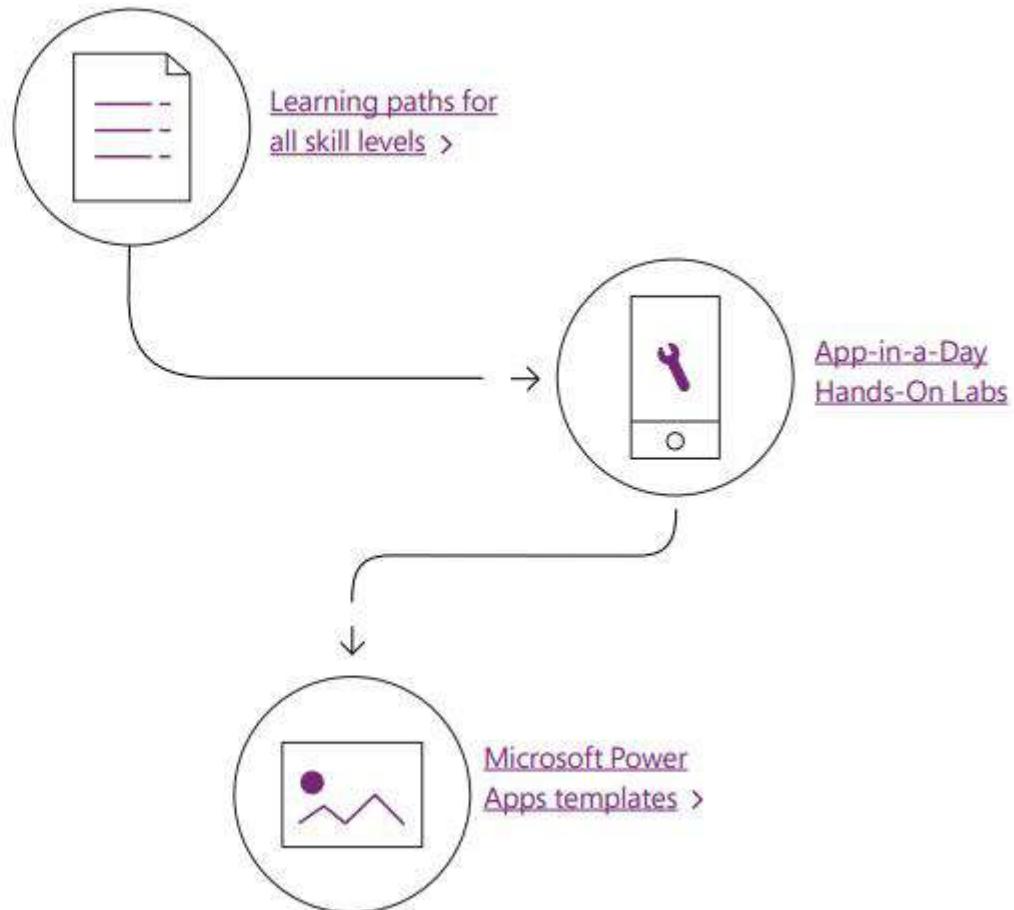
It's also important to consider whether the data might be sensitive or proprietary and work with the appropriate people in your organization to keep it secure and compliant.

How a Canvas App pulls data from a source, in this case, a SharePoint list.



# Step 5: Start building

With a clear vision of what you want to build, it's time to get started. Building with Power Apps is easy and intuitive, with a drag and drop interface. If you've ever made a PowerPoint presentation, the design experience will feel familiar.



Reference:

The DIY Guide to Building Your First Business App

Turn bright ideas into brilliant apps

by Microsoft



# Thank You!

In our next session:



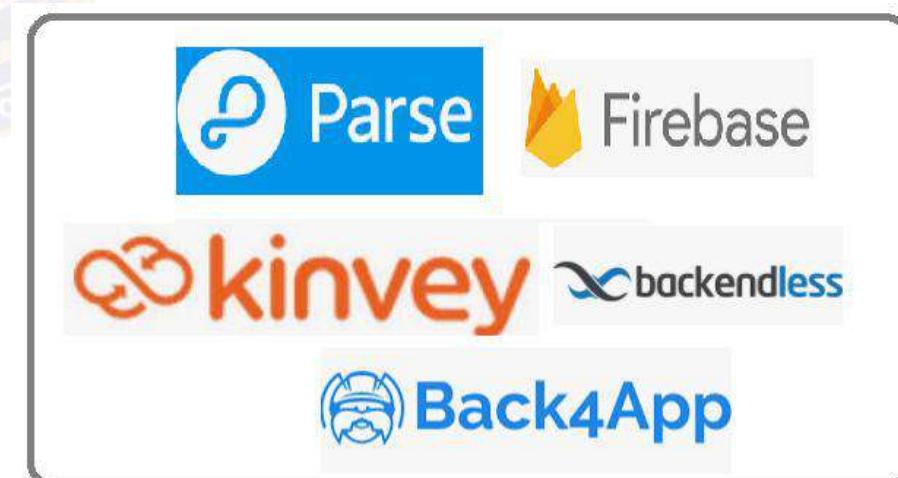
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Backend as a Service (BaaS)

Chandan ravandur N

# BaaS

- is a platform that
  - automates backend side development
  - takes care of the cloud infrastructure
- App teams
  - outsource the responsibilities of running and maintaining servers to a third party
  - focus on the frontend or client-side development
- Provides a set of tools to help developers to create a backend code speedily
- with help of ready to use features such as
  - scalable databases
  - APIs
  - cloud code functions
  - social media integrations
  - file storage
  - push notifications



# Apps suitable for BaaS

- Social media apps
  - alike Facebook, Instagram
- Real-time chat applications
  - alike WhatsApp
- Taxi apps
  - alike Uber, Ola
- Video and music streaming apps
  - similar to Netflix
- Mobile games
- Ecommerce apps



# Why Backend as a service?

- A BaaS platform solves two problems:
  - Manage and scale cloud infrastructure
  - Speed up backend development
- Business reasons to use BaaS:
  - Reduce time to market
  - Save money and decrease the cost of development
  - Assign fewer backend developers to a project
  - Outsource cloud infrastructure management

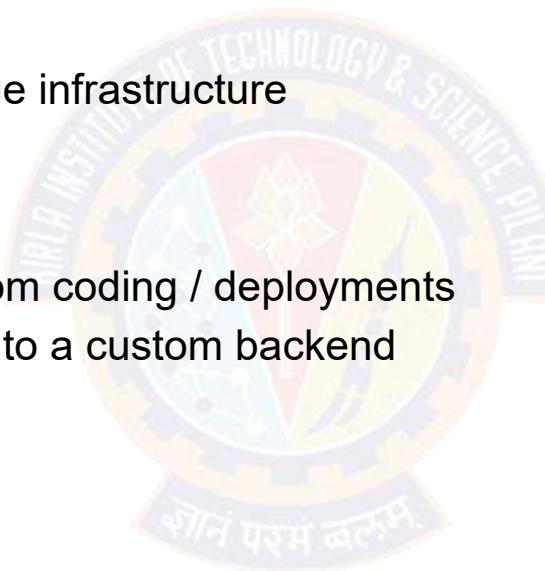
Technical reasons to use BaaS:

- Focus on frontend development
- Excludes redundant stack setup
- No need to program boilerplate code
- Standardize the coding environment
- Let backend developers program high-value lines of code
- Provides ready to use features like authentication, data storage, and search



# Pros-Cons

- Advantages
  - Speedy Development
  - Reduced Development price
  - Serverless, and no need to manage infrastructure
- Disadvantages
  - Less flexible as compared to custom coding / deployments
  - Less customization in comparison to a custom backend
  - Vendor lock-in possible





# Thank You!

In our next session:



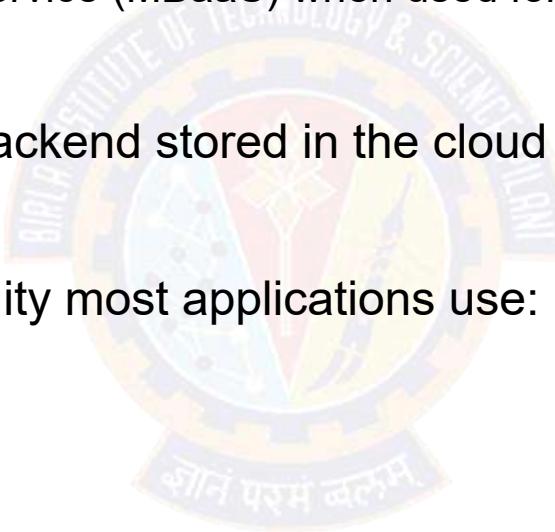
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Mobile Backend as a Service MBaaS

Chandan Ravandu N

# MBaaS

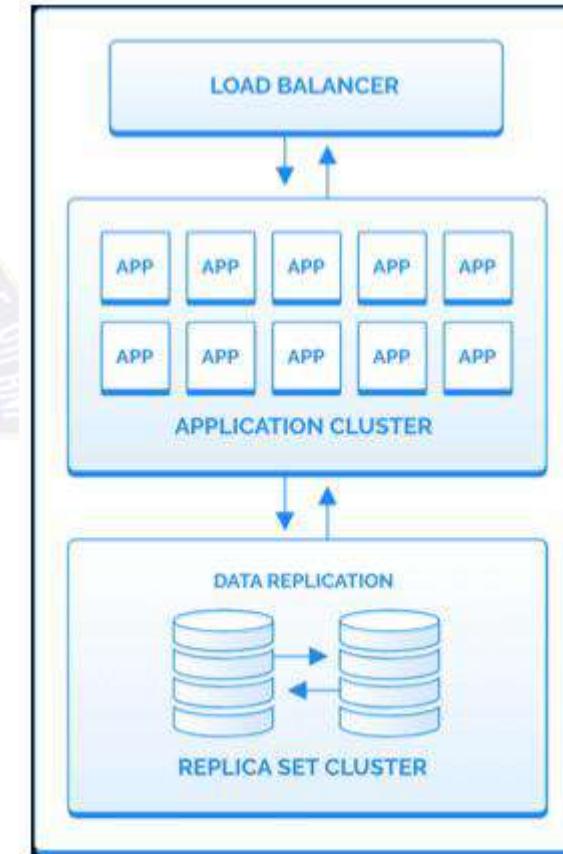
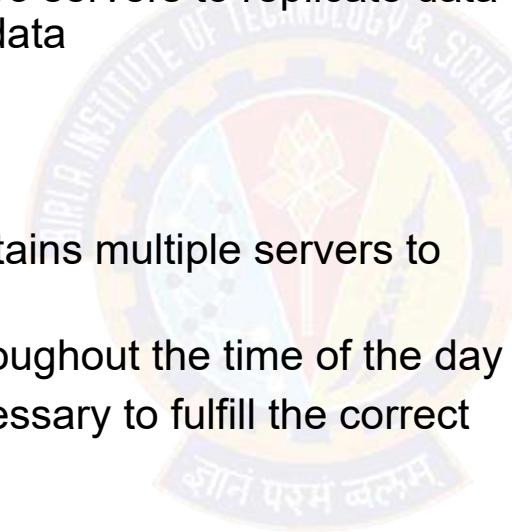
- Pretty much similar to BaaS!
  - BaaS can be used for web projects or mobile projects
  - Termed as Mobile backend as a service (MBaaS) when used for mobile development
- Allows you to use pre-developed backend stored in the cloud
- Backend includes unified functionality most applications use:
  - push notifications
  - social networks integrations
  - cloud storage
  - messaging
  - analytics etc.



# MBaaS / BaaS Architecture

## Three Tier

- The first layer - Database
  - Is the foundation and contains the database servers
  - A database cluster has at least two servers to replicate data and a backup routine to retrieve data
- The second layer - Application
  - Is the application cluster and contains multiple servers to process requests
  - Quantity of servers fluctuates throughout the time of the day
  - Auto-scaling procedures are necessary to fulfill the correct number of servers
- The third layer - Gateway
  - Connects the application servers to the Internet
  - Composed of load balancers and CDNs

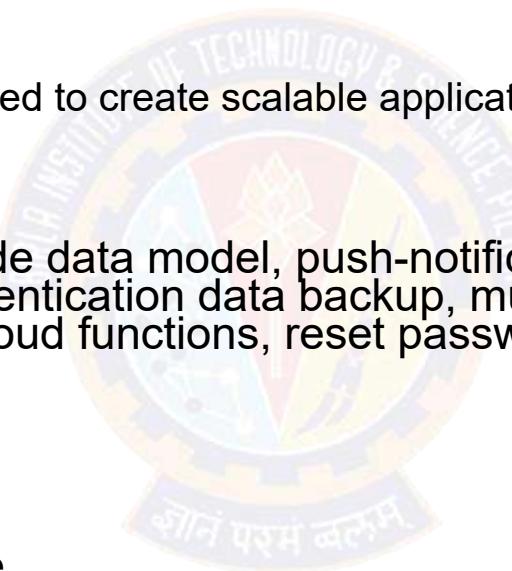


[Image source : Back4App](#)

# MBaaS Providers

## Back4App

- Offers a comprehensive product that uses several open-source technologies
  - NodeJS, Parse Server, and MongoDB
- Uses a simple approach
  - Help developers with resources needed to create scalable applications without the hassles of reinventing the wheel
- Features: The platform features include data model, push-notifications, REST and GraphQL APIs, login, authentication data backup, multiple SDK, non-technical administration panel, cloud functions, reset passwords, and CLI,
- Open Source
- Cloud-Hosting or self hosting possible
- Private Cloud and on-premises deployment possible
- Professional Services: Solutions Architect, Consulting, and enterprise plans



# MBaaS Providers

## Parse

- Excellent framework for expediting mobile application development
- Facebook converted the project to open source in 2016
- Features: JSON-like data management console, social-login (Facebook, Apple, Twitter, WeChat, GitHub, Google, etc.), password rest, integration with AWS file storage, push-notifications, SDKs (GraphQL, Rest, Javascript, iOS, Android, etc.)
- Open Source
- Cloud-Hosting Not possible, but Self-Hosting possible
- Remarks: This framework has over 16k stars and 4k forks on Github



# MBaaS Providers

## Firebase

- Versatile platform for mobile and web application development from Google
  - Developers can create serverless apps and send push notifications to connected mobile devices
  - Highly secure and efficient platform for scalability
  - Unique feature of Firebase is its realtime database - makes it an excellent choice for developing apps with live chat
- 
- Features: The main features include realtime database, phone authentication, storage, cloud functions, hosting, ML kit, and many other excellent features
  - Cloud-Hosting possible
  - Pay as you go pricing model
  - Support Services: The platform supports case submission.
  - Remarks: Firebase does neither offer private clouds or enterprise plans.





# Thank You!

In our next session:



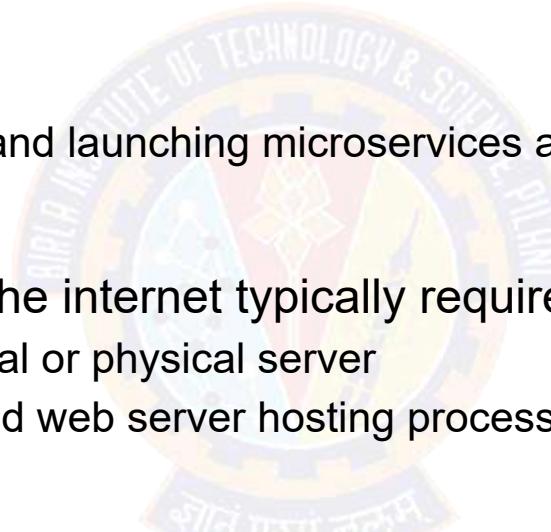
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Function as a Service

Chandan Ravandur n

# What is FaaS (Function-as-a-Service)?

- A type of cloud-computing service that
  - ✓ allows to execute code
  - ✓ in response to events
  - ✓ without the complex infrastructure
  - ✓ typically associated with building and launching microservices applications
- Hosting a software application on the internet typically requires
  - ✓ provisioning and managing a virtual or physical server
  - ✓ managing an operating system and web server hosting processes
- With FaaS, the physical hardware, virtual machine operating system, and web server software management are all handled automatically by cloud service provider
  - ✓ allowing developers to focus solely on individual functions in application code

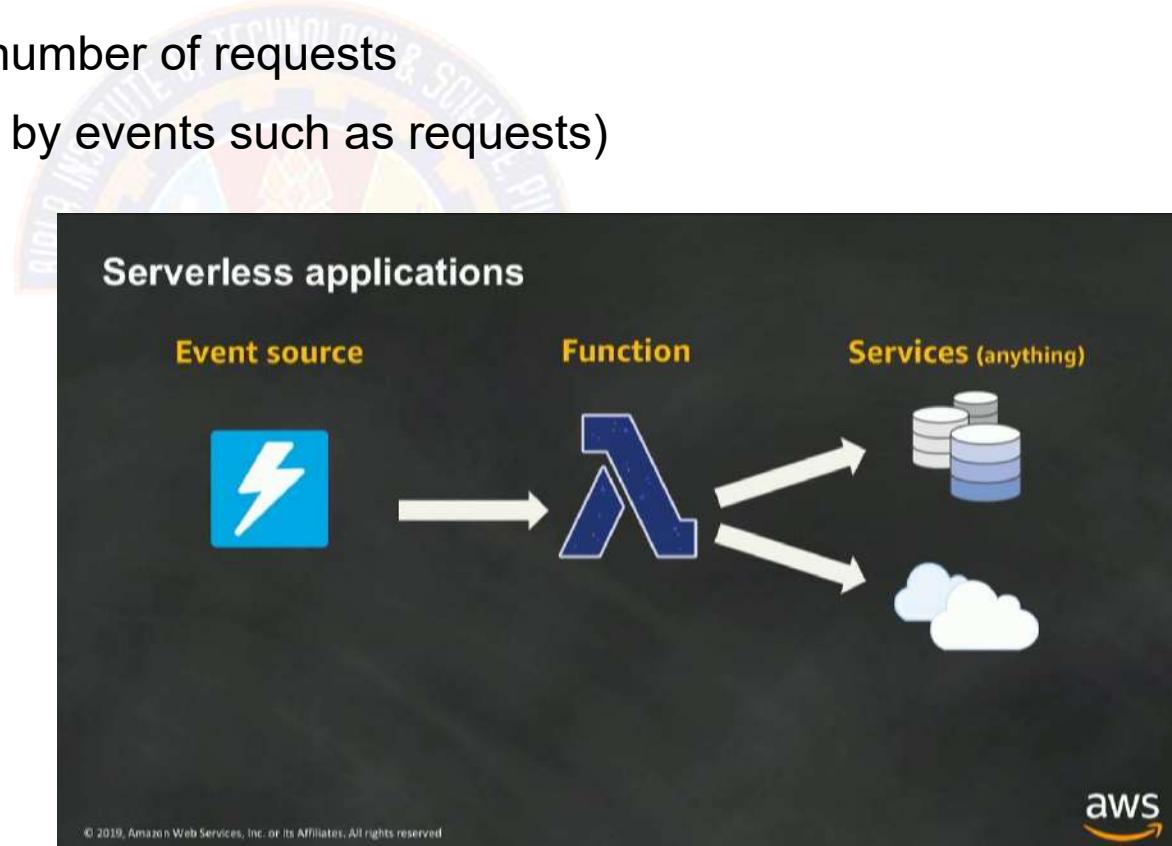


FaaS



# Characteristics of FaaS

- No server management or maintenance needed
- Stateless
- Automatic scaling, fine grained to number of requests
- Runs only when needed (triggered by events such as requests)

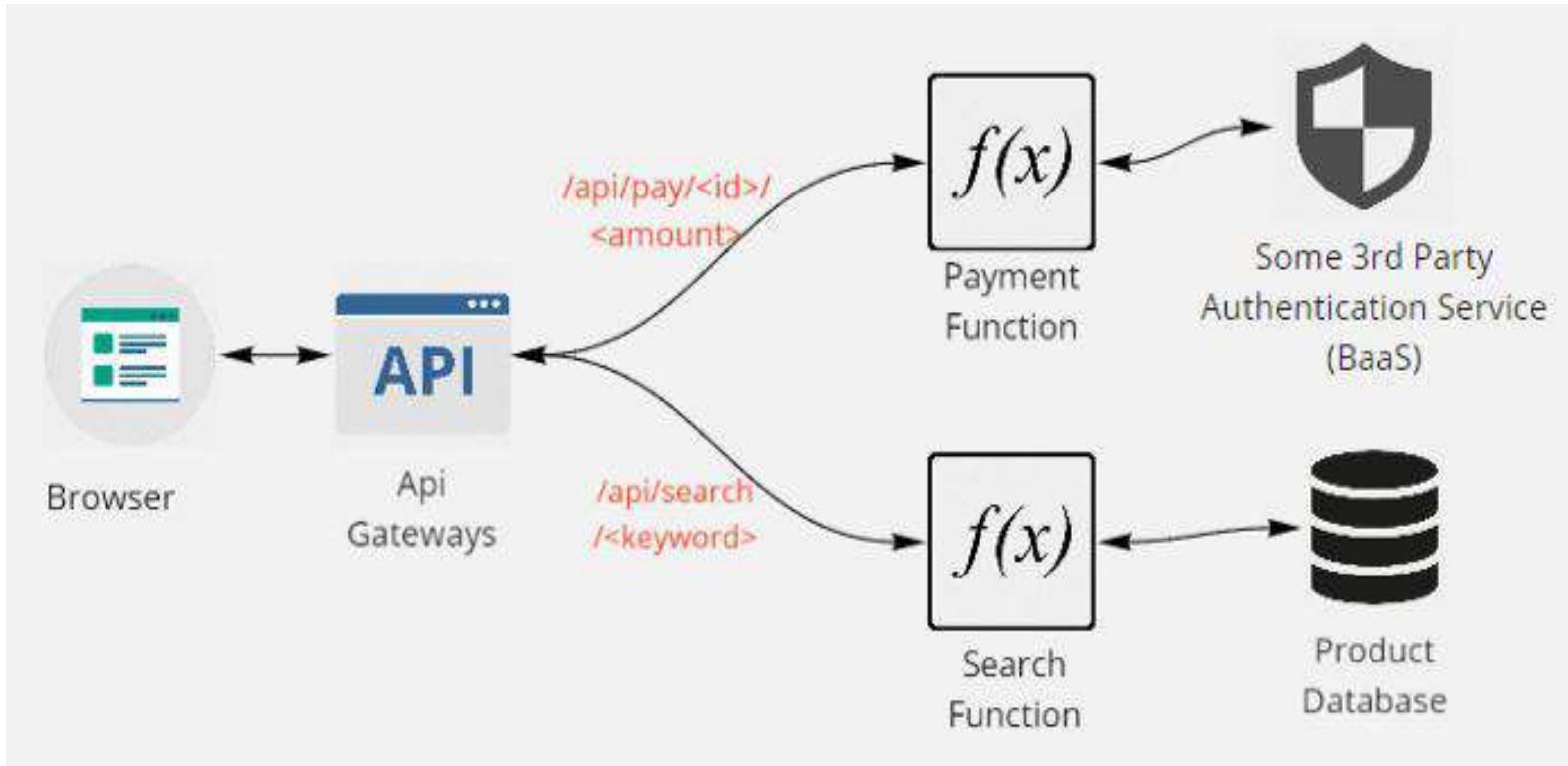


# Benefits of FaaS

**FaaS is a valuable tool if you're looking to efficiently and cost-effectively migrate applications to the cloud.**

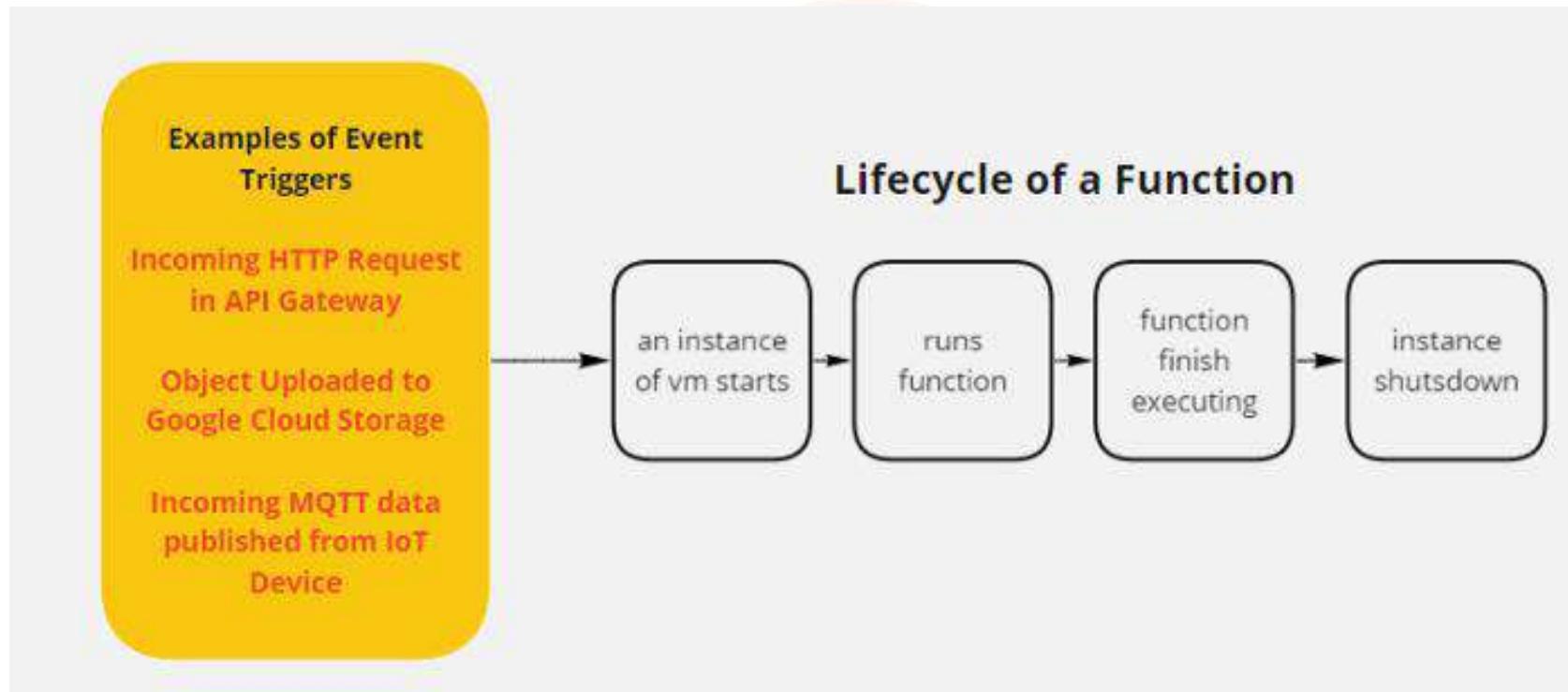
- Focus more on code, not infrastructure:
  - ✓ With FaaS, can divide the server into functions that can be scaled automatically and independently so don't have to manage infrastructure
  - ✓ This allows to focus on the app code and can dramatically reduce time-to-market
- Pay only for the resources you use, when you use them:
  - ✓ With FaaS, you pay only when an action occurs
  - ✓ When the action is done, everything stops—no code runs, no server idles, no costs are incurred
  - ✓ FaaS is, therefore, cost-effective, especially for dynamic workloads or scheduled tasks
  - ✓ FaaS also offers a superior total-cost-of-ownership for high-load scenarios
- Scale up or down automatically:
  - ✓ With FaaS, functions are scaled automatically, independently, and instantaneously, as needed
  - ✓ When demand drops, FaaS automatically scales back down
- Get all the benefits of robust cloud infrastructure:
  - ✓ FaaS offers inherent high availability because it
  - ✓ is spread across multiple availability zones per geographic region
  - ✓ can be deployed across any number of regions without incremental costs

# Serverless Architecture



[Source : medium](#)

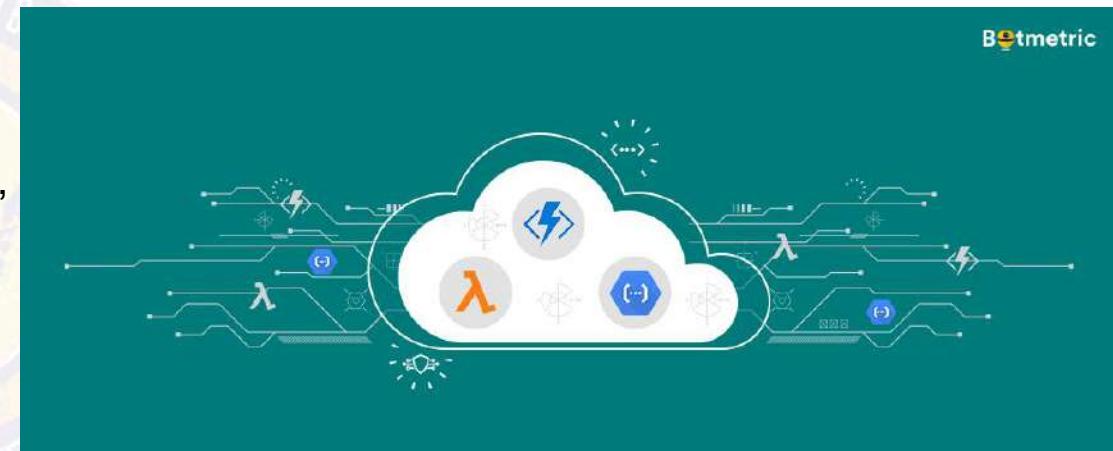
# How it works?



Source : [medium](#)

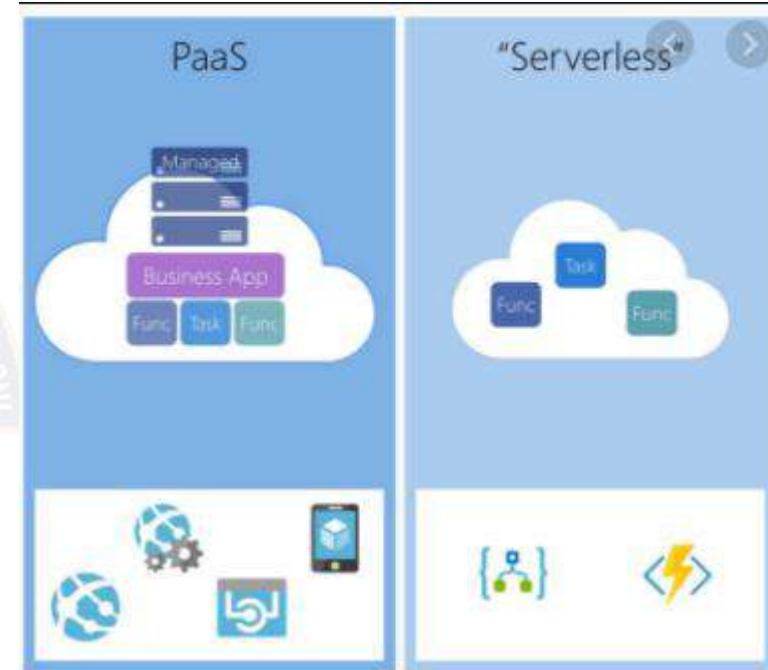
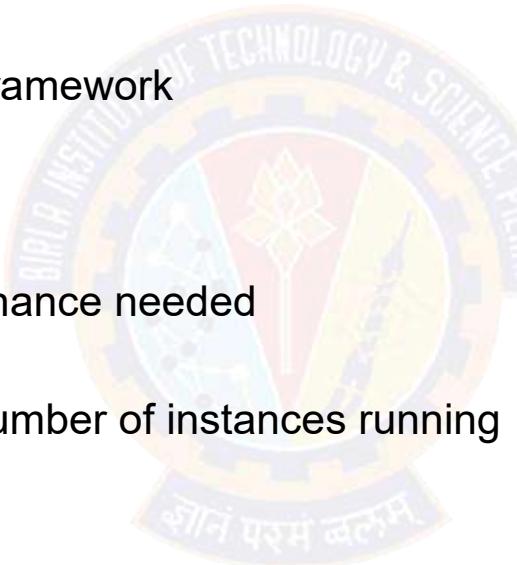
# FaaS vs Serverless

- Serverless and Functions-as-a-Service (FaaS) are often conflated with one another
  - ✓ but the truth is that FaaS is actually a subset of serverless
- Serverless is focused on any service category
  - ✓ be it compute, storage, database, messaging, API gateways, etc.
  - ✓ where configuration, management, and billing of servers are invisible to the end user
- FaaS, is focused on the event-driven computing paradigm
  - ✓ while perhaps the most central technology in serverless architectures
  - ✓ wherein application code, or containers, only run in response to events or requests



# PaaS vs FaaS

- PaaS is very similar to FaaS
  - ✓ except that that it does not triggered by event
  - ✓ is typically always on
  - ✓ This does not suit the serverless framework
- Characteristics
  - ✓ No server management or maintenance needed
  - ✓ Stateless
  - ✓ Automatic scaling, adjustable to number of instances running
  - ✓ Typically runs 24/7
- Sounds very similar to FaaS right?
  - ✓ However there is one key operational difference between the both of them:
  - ✓ Scalability, which affects operating cost.



Source : stackify



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

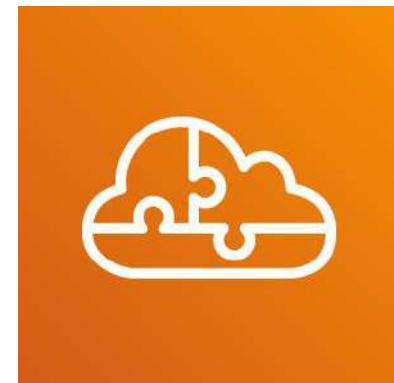
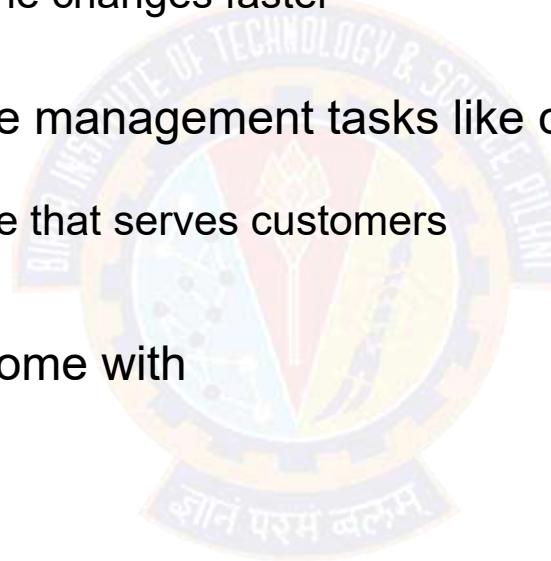
# Serverless on AWS

Chandan Ravandur N

# Serverless on AWS

## Build and run applications without thinking about servers

- Serverless is a way to describe the services, practices, and strategies that enables building more agile applications
  - ✓ to foster innovations and responses to the changes faster
- With serverless computing, infrastructure management tasks like capacity provisioning and patching are handled by AWS
  - ✓ developer can focus on only writing code that serves customers
- Serverless services like AWS Lambda come with
  - ✓ automatic scaling
  - ✓ built-in high availability
  - ✓ and a pay-for-value billing model
- Lambda is an event-driven compute service that enables to run code in response to events from over 150 natively-integrated AWS and SaaS sources
  - ✓ all without managing any servers



# Benefits

## **Move from idea to market, faster**

By eliminating operational overhead, your teams can release quickly, get feedback, and iterate to get to market faster.

## **Adapt at scale**

With technologies that automatically scale from zero to peak demands, you can adapt to customer needs faster than ever.

## **Lower your costs**

With a pay-for-value billing model, you never pay for over-provisioning and your resource utilization is optimized on your behalf.

## **Build better applications, easier**

Serverless applications have built-in service integrations, so you can focus on building your application instead of configuring it.

# Serverless Services on AWS

## Compute



### AWS Lambda

Run code without provisioning or managing servers and pay only for the resources you consume



### Amazon Fargate

Run serverless containers on Amazon Elastic Container Service (ECS) or Amazon Elastic Kubernetes Service (EKS)

# Serverless Services on AWS(2)

## Application Integration



### Amazon EventBridge

Build an event-driven architecture that connects application data from your own apps, SaaS, and AWS services



### AWS Step Functions

Coordinate multiple AWS services into serverless workflows so you can build and update apps quickly



### Amazon SQS

Decouple and scale microservices with message queues that send, store, and receive messages at any volume



### Amazon SNS

Get reliable high throughput pub/sub, SMS, email, and mobile push notifications



### Amazon API Gateway

Create, publish, maintain, monitor, and secure APIs at any scale for serverless workloads and web applications



### AWS AppSync

Create a flexible API to securely access, manipulate, and combine data from one or more data sources

# Serverless Services on AWS(3)

## Data Store



### Amazon S3

Store any amount of data with industry-leading scalability, data availability, security, and performance



### Amazon DynamoDB

Get single-digit millisecond performance at any scale with this key-value and document database



### Amazon Aurora Serverless

Automatically scale capacity based on your application's need with this configuration for Amazon Aurora



### Amazon RDS Proxy

Increase scalability, resiliency, and security with this proxy for Amazon Relational Database Service (RDS)

Reference :  
[AWS Serverless](#)



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# AWS Lambda

Chandan Ravandur n

---

# AWS Lambda

Run code without thinking about servers or clusters. Only pay for what you use.

- AWS Lambda is a serverless compute service that allows to run code
  - ✓ without provisioning or managing servers
  - ✓ creating workload-aware cluster scaling logic
  - ✓ maintaining event integrations
  - ✓ or managing runtimes
- With Lambda, one can run code for virtually any type of application or backend service
  - ✓ all with zero administration
- Just upload code as a ZIP file or container image, and Lambda
  - ✓ automatically and precisely allocates compute execution power
  - ✓ runs code based on the incoming request or event, for any scale of traffic
- Can write Lambda functions in favorite language (Node.js, Python, Go, Java, and more)
  - ✓ use both serverless and container tools, such as AWS SAM or Docker CLI, to build, test, and deploy functions



# Benefits

## No servers to manage

AWS Lambda automatically runs your code without requiring you to provision or manage infrastructure. Just write the code and upload it to Lambda either as a ZIP file or container image.

## Continuous scaling

AWS Lambda automatically scales your application by running code in response to each event. Your code runs in parallel and processes each trigger individually, scaling precisely with the size of the workload, from a few requests per day, to hundreds of thousands per second.

## Cost optimized with millisecond metering

With AWS Lambda, you only pay for the compute time you consume, so you're never paying for over-provisioned infrastructure. You are charged for every millisecond your code executes and the number of times your code is triggered. With Compute Savings Plan, you can additionally save up to 17%.

## Consistent performance at any scale

With AWS Lambda, you can optimize your code execution time by choosing the right memory size for your function. You can also keep your functions initialized and hyper-ready to respond within double digit milliseconds by enabling Provisioned Concurrency.

# How it works?



# Use cases

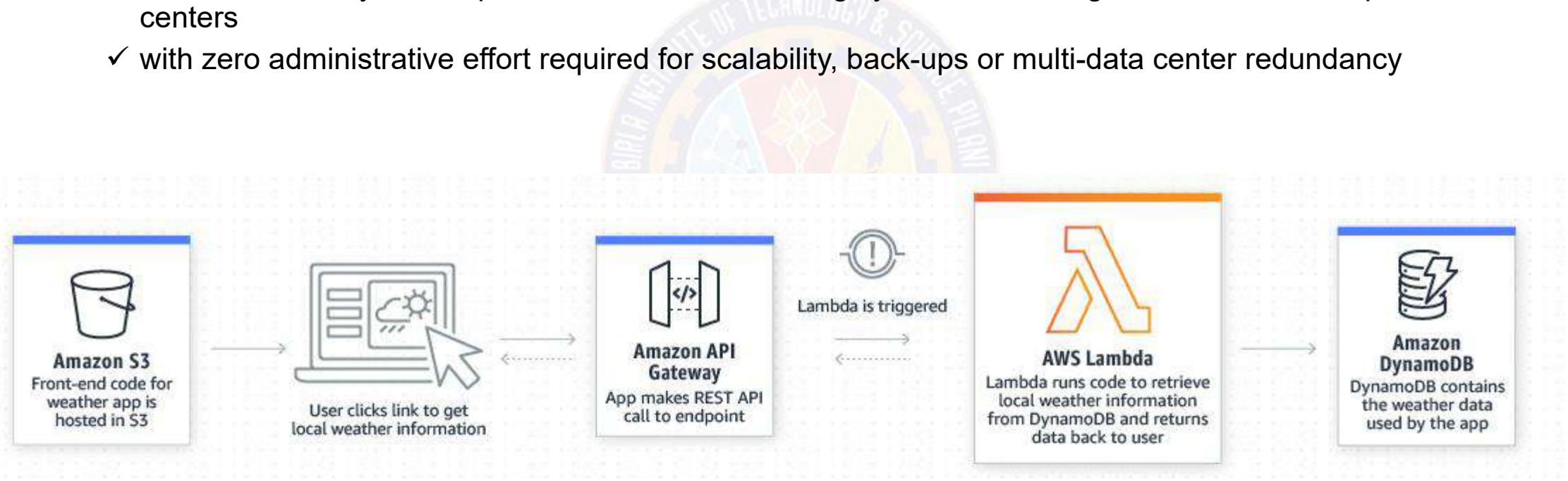
## What can you build with AWS Lambda?

- Backends
  - ✓ Can build serverless backends using AWS Lambda to handle
    - ❖ web
    - ❖ mobile
    - ❖ Internet of Things (IoT)
    - ❖ and 3rd party API requests
  - ✓ Can take advantage of Lambda's consistent performance controls, such as multiple memory configurations and Provisioned Concurrency
    - ❖ for building latency-sensitive applications at any scale
- Data processing
  - ✓ Can use AWS Lambda to execute code in response to triggers such as
    - ❖ changes in data
    - ❖ shifts in system state
    - ❖ or actions by users
  - ✓ Lambda
    - ❖ can be directly triggered by AWS services such as S3, DynamoDB, Kinesis, SNS, and CloudWatch
    - ❖ can connect to existing EFS file systems, or it can be orchestrated into workflows by AWS Step Functions
    - ❖ allows to build a variety of real-time serverless data processing systems

# Use case

## Backends - Web applications

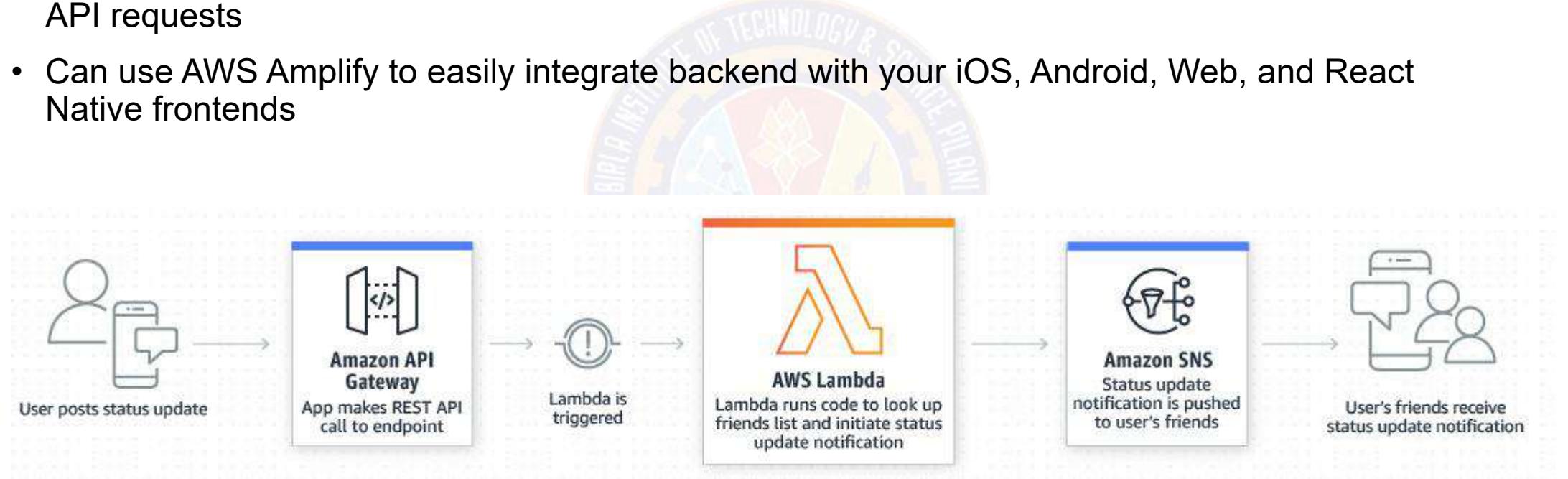
- By combining AWS Lambda with other AWS services, developers can build powerful web applications
  - ✓ that automatically scale up and down and run in a highly available configuration across multiple data centers
  - ✓ with zero administrative effort required for scalability, back-ups or multi-data center redundancy



# Use case (2)

## Backends - Mobile backends

- AWS Lambda makes it easy to create rich, personalized app experiences
- Can build backends using AWS Lambda and Amazon API Gateway to authenticate and process API requests
- Can use AWS Amplify to easily integrate backend with your iOS, Android, Web, and React Native frontends



# Use case (3)

## Backends - IoT backends

- Can build serverless backends using AWS Lambda to handle
  - ✓ web
  - ✓ mobile
  - ✓ Internet of Things (IoT)
  - ✓ and 3rd party API requests



# Use case (4)

## Data processing - Real-time file processing

- Can use Amazon S3 to trigger AWS Lambda to process data immediately after an upload
- Can also connect to an existing Amazon EFS file system directly
- enabling massively parallel shared access for large scale file processing
- Can use Lambda to
  - ✓ thumbnail images
  - ✓ transcode videos
  - ✓ index files
  - ✓ process logs
  - ✓ validate content
  - ✓ and aggregate and filter data in real-time



# Use case (5)

## Data processing - Real-time stream processing

- Can use AWS Lambda and Amazon Kinesis to process real-time streaming data for
  - ✓ application activity tracking
  - ✓ transaction order processing
  - ✓ click stream analysis
  - ✓ data cleansing
  - ✓ metrics generation
  - ✓ log filtering
  - ✓ indexing, social media analysis
  - ✓ and IoT device data telemetry and metering



Reference :  
[AWS Serverless](#)



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

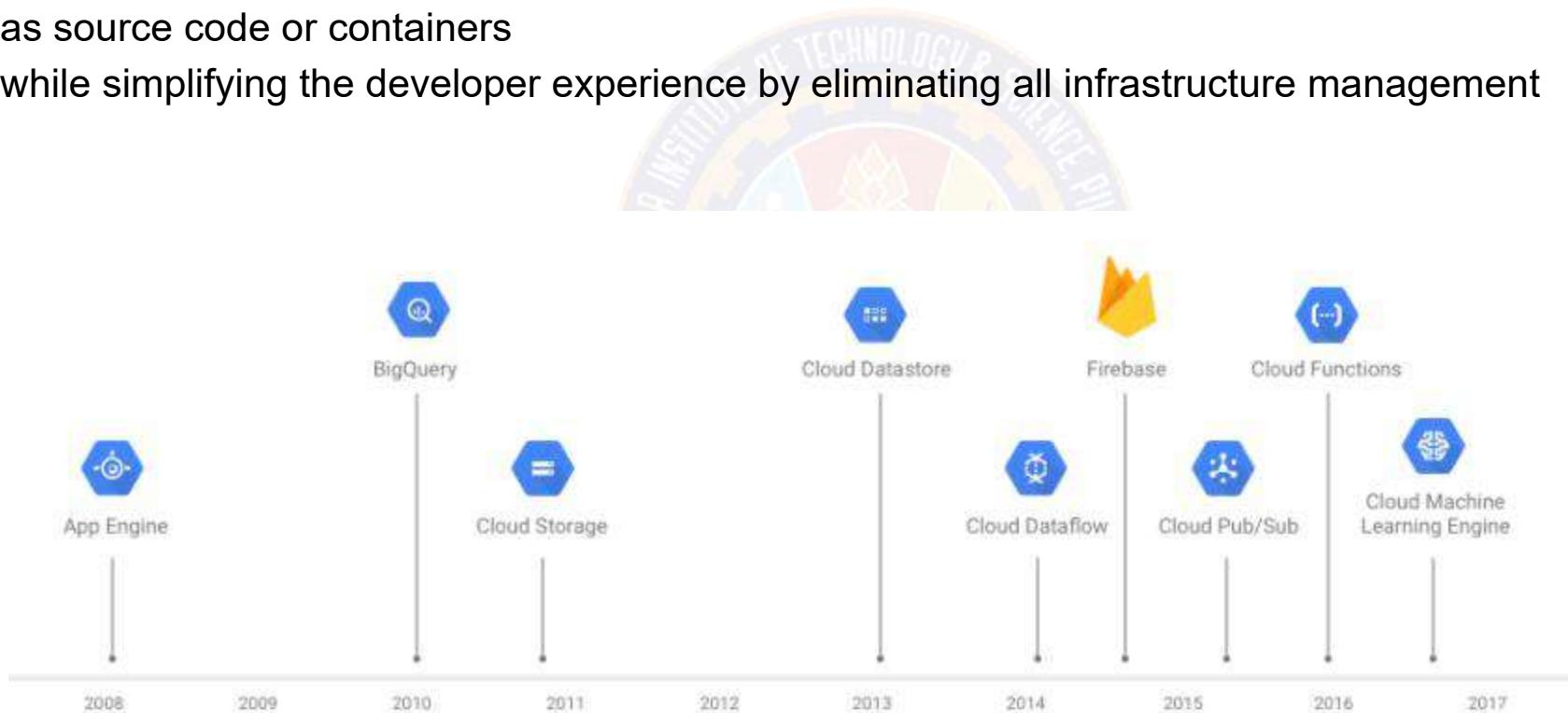
# GCP Serverless computing

Chandan Ravandur N

# Google Cloud Platform

## Serverless

- Google Cloud's serverless platform allows to
  - ✓ build, develop, and deploy functions and applications
  - ✓ as source code or containers
  - ✓ while simplifying the developer experience by eliminating all infrastructure management



# Benefits

## Enable faster and more secure development, deployment, and operations

### Speed to market

Build your apps, deploy them, and run them in production—all within a few seconds. Increase productivity and flexibility by letting your developers write code however they choose.

### Simple developer experience

Free up developers and operators with fully managed infrastructure. No more provisioning, configuration, patching, and managing your servers or clusters.

### Automatic scaling

Our serverless environment automatically scales your workloads up or down, even to zero, depending on traffic.

# Common use cases for serverless compute products

- Build scalable, secure web apps
  - ✓ Code, build, and deploy scalable applications in a fully managed environment
  - ✓ designed to help developers to succeed with
    - ❖ built-in security
    - ❖ auto scaling
    - ❖ ops management for faster deployment
- Develop, deploy, and manage APIs
  - ✓ Build scalable APIs in an environment built for developers to succeed
  - ✓ Can develop REST APIs for web and mobile backends
  - ✓ Manage the connection between different parts of application and internal cloud services



# Common use cases for serverless compute products(2)

- Build apps with data processing in mind
  - ✓ Serverless computing environment manages the infrastructure workloads need, in order to handle
    - ❖ auto scaling
    - ❖ Authorization
    - ❖ and event triggers
  - ✓ The pub/sub model of communication makes it easy to ingest and transform large amounts of data and build complex, scalable data pipelines
    - ✓ while saving time on backend confusion
- Automate event orchestration
  - ✓ Automatically validate policies or configurations and perform other scripted automation using event triggers
  - ✓ Serverless computing products can listen to events from other clouds, handle webhooks, and manage distributing events and workloads to other components
  - ✓ This built-in ability makes it straightforward for application to handle complex event needs

# GCP Serverless products



## Cloud Functions

Scalable pay as you go functions as a service (FaaS) to run your code with zero server management.



## App Engine

Fully managed serverless platform for developing and hosting web applications at scale.



## Cloud Run

Fully managed compute platform for deploying and scaling containerized applications quickly and securely.



## Workflows

Orchestrate and automate Google Cloud and HTTP-based API services. Fully managed service requiring no infrastructure management or capacity planning.

Reference :

GCP Serverless Computing Product pages



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

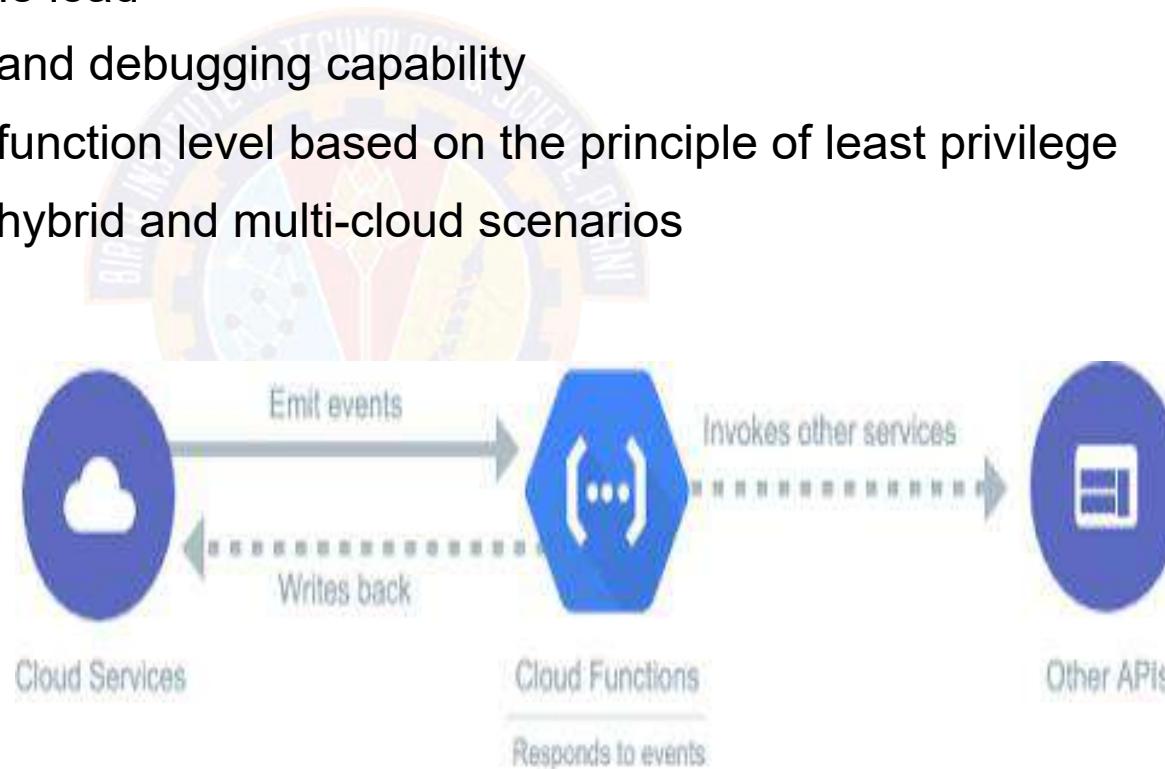
# Google Cloud Functions

Chandan Ravandur N

# Google Cloud Functions

**Scalable pay-as-you-go functions as a service (FaaS) to run your code with zero server management**

- No servers to provision, manage, or upgrade
- Automatically scale based on the load
- Integrated monitoring, logging, and debugging capability
- Built-in security at role and per function level based on the principle of least privilege
- Key networking capabilities for hybrid and multi-cloud scenarios



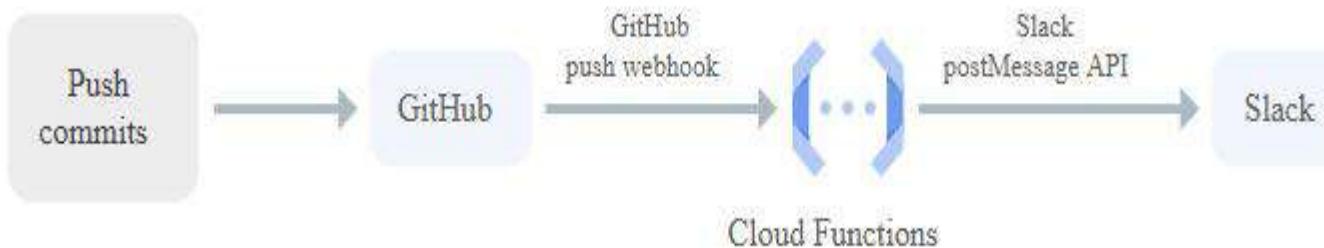
# Key features

- Simplified developer experience and increased developer velocity
  - ✓ Cloud Functions has a simple and intuitive developer experience
  - ✓ Just write code and let Google Cloud handle the operational infrastructure
  - ✓ Develop faster by writing and running small code snippets that respond to events
  - ✓ Connect to Google Cloud or third-party cloud services via triggers to streamline challenging orchestration problems
- Pay only for what you use
  - ✓ Only billed for function's execution time, metered to the nearest 100 milliseconds
  - ✓ Pay nothing when function is idle
  - ✓ Cloud Functions automatically spins up and backs down in response to events
- Avoid lock-in with open technology
  - ✓ Use open source FaaS (function as a service) framework to run functions across multiple environments and prevent lock-in
  - ✓ Supported environments include Cloud Functions, local development environment, on-premises, Cloud Run, and other Knative-based serverless environments

# Use cases

## Integration with third-party services and APIs

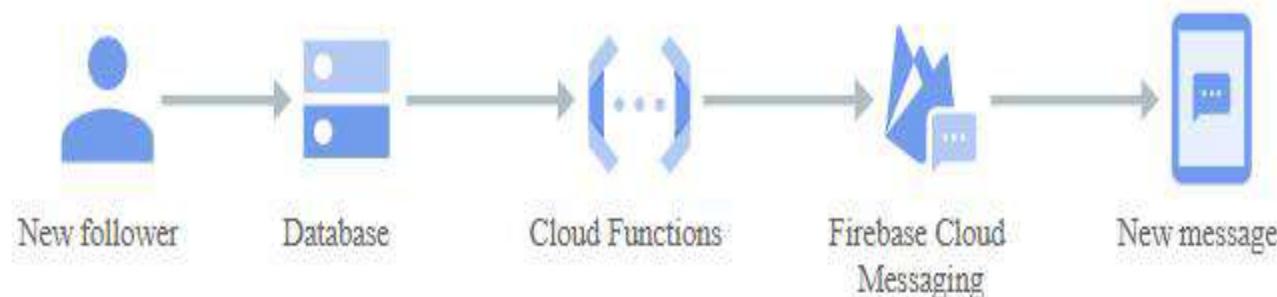
- Use Cloud Functions to
  - ✓ surface out microservices via HTTP APIs
  - ✓ or integrate with third-party services that offer webhook integrations
  - ✓ to quickly extend application with powerful capabilities such as
    - ❖ sending a confirmation email after a successful Stripe payment
    - ❖ or responding to Twilio text message events



# Use cases (2)

## Serverless mobile back ends

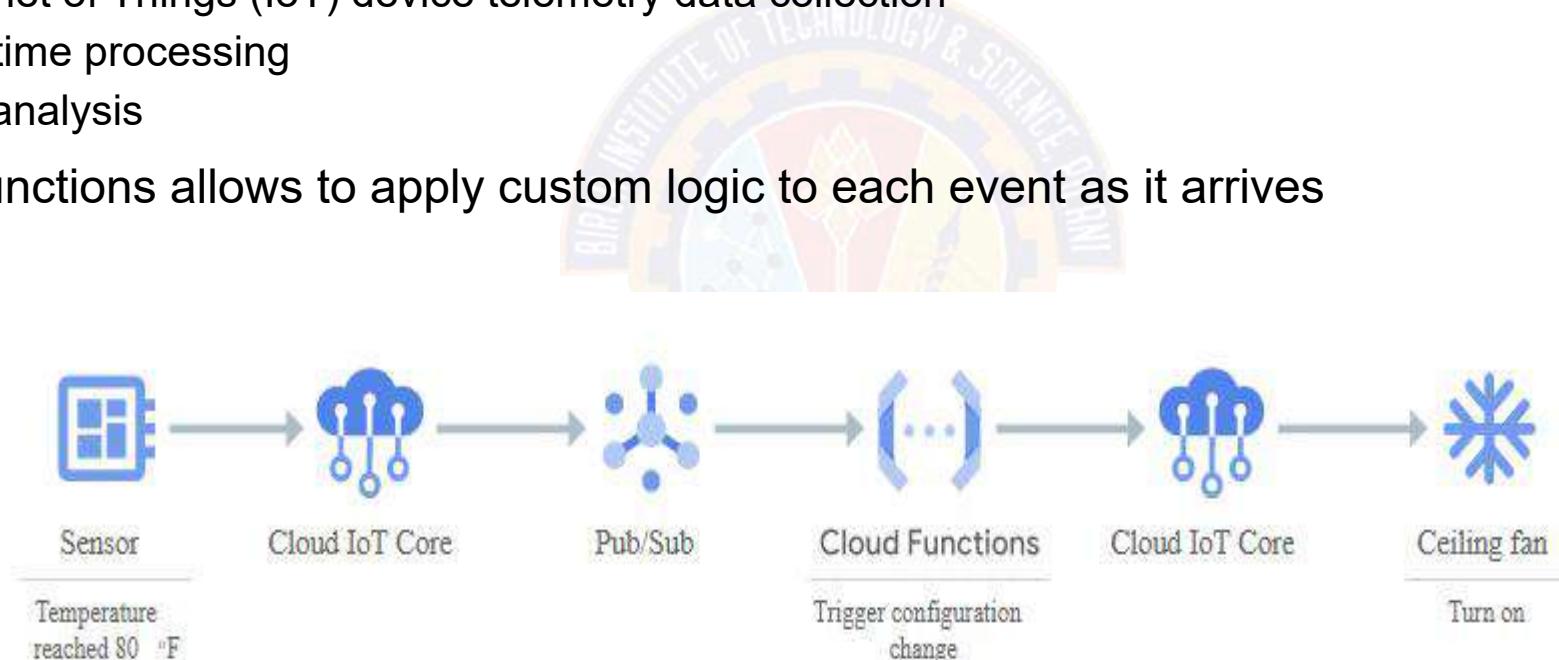
- Use Cloud Functions directly from Firebase to extend application functionality without spinning up a server
- Run code in response to user actions, analytics, and authentication events
  - ✓ to keep users engaged with event-based notifications
  - ✓ to offload CPU- and networking-intensive tasks to Google Cloud



# Use cases (3)

## Serverless IoT back ends

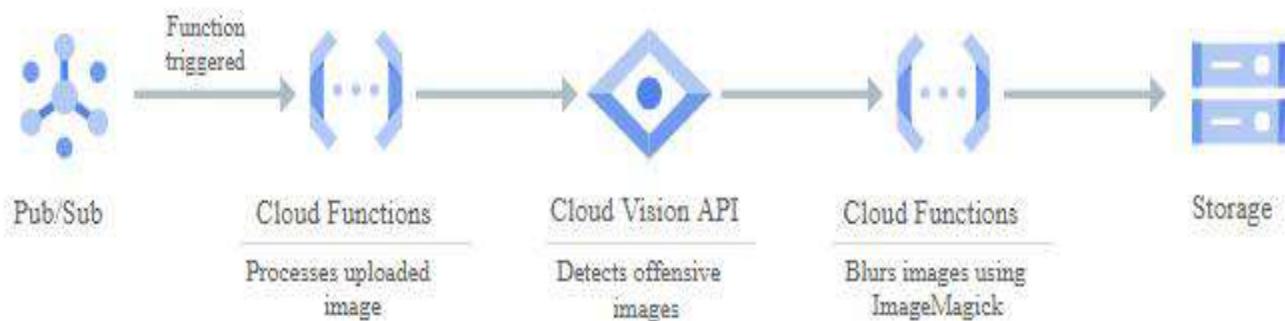
- Use Cloud Functions with Cloud IoT Core and other fully managed services to build back ends for
  - ✓ Internet of Things (IoT) device telemetry data collection
  - ✓ real-time processing
  - ✓ and analysis
- Cloud Functions allows to apply custom logic to each event as it arrives



# Use cases (4)

## Real-time stream processing

- Use Cloud Functions to respond to events from Pub/Sub to process, transform, and enrich streaming data in
  - ✓ transaction processing
  - ✓ click-stream analysis
  - ✓ application activity tracking
  - ✓ IoT device telemetry
  - ✓ social media analysis
  - ✓ and other types of applications



Reference :

GCP Serverless Computing Product pages



# Thank You!

In our next session:



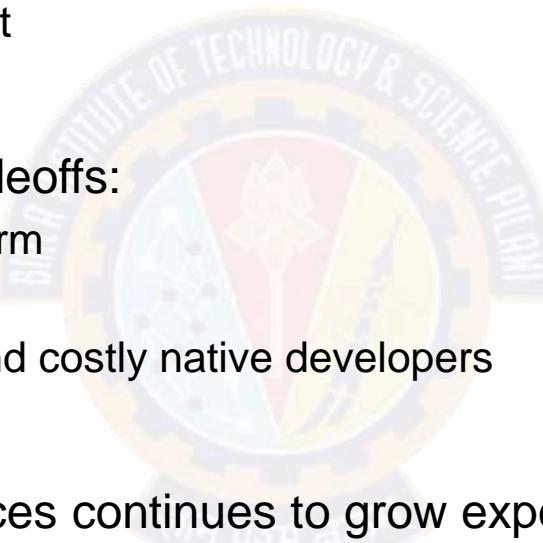
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# The mobile delivery gap

Chandan Ravandur N

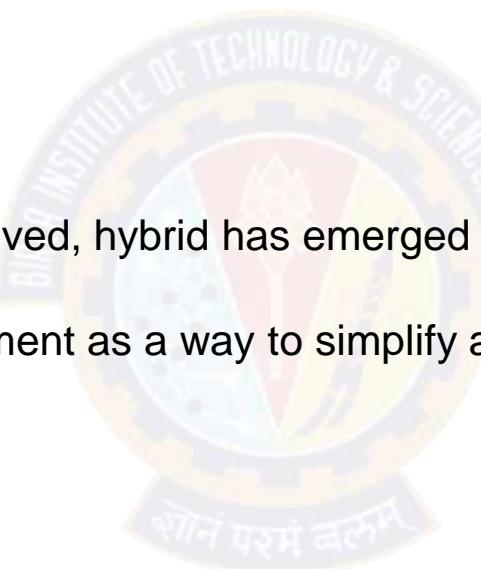
# The mobile delivery gap

- Back in the early days of mobile, there was really only one way to give the users the performance and features they expected
  - ✓ Had to use each platform's native toolset
- That came with real and measurable tradeoffs:
  - ✓ Building in parallel for each mobile platform
  - ✓ Managing multiple codebases
  - ✓ Hiring and retaining highly specialized and costly native developers
- Meanwhile, demand for mobile experiences continues to grow exponentially
  - ✓ By 2023, 80% of all enterprise software interactions are expected to occur on mobile



# Going Hybrid way

- Given the time and cost of traditional native development
  - ✓ it's no surprise that many development teams are struggling to keep up with this demand
- Times have changed
  - ✓ As mobile and web technology have evolved, hybrid has emerged as a viable alternative to native
  - ✓ Many are now looking at hybrid development as a way to simplify and speed up app creation

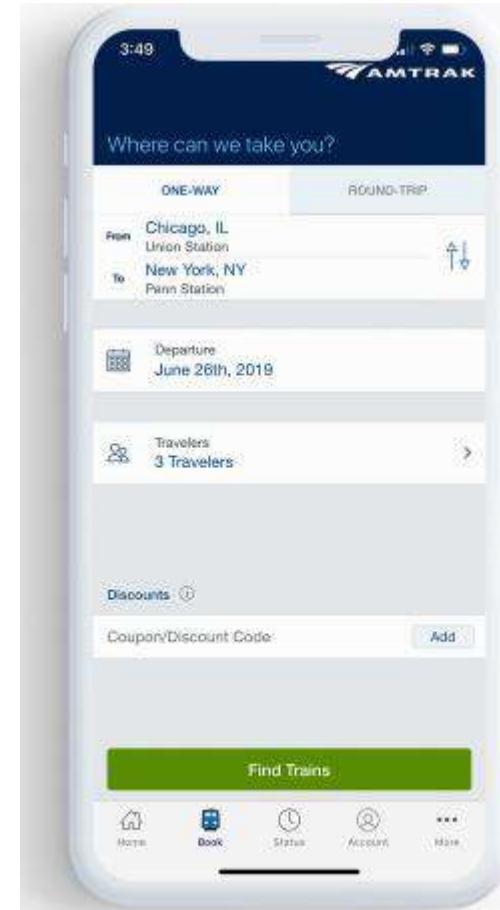
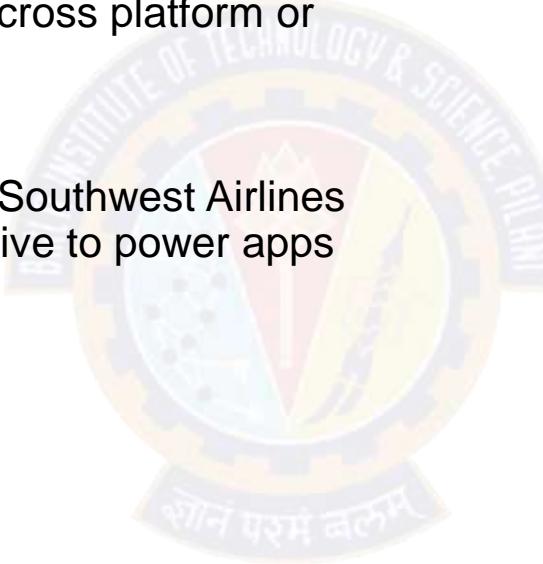


Demand for mobile experiences is growing 5x faster than internal IT teams can deliver.

GARTNER

# The choice to go hybrid

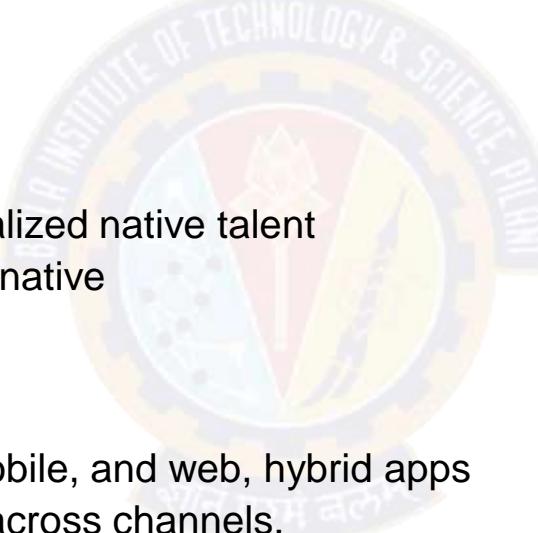
- The growing adoption of hybrid is evident in a Forrester survey
  - ✓ two-thirds of developers are choosing a cross platform or web-based approach over native tools
  - ✓ Top brands like Target, Nationwide, and Southwest Airlines have chosen a hybrid approach over native to power apps for their customers and employees



Source : Ionic whitepaper

# Top reasons for making the switch to hybrid

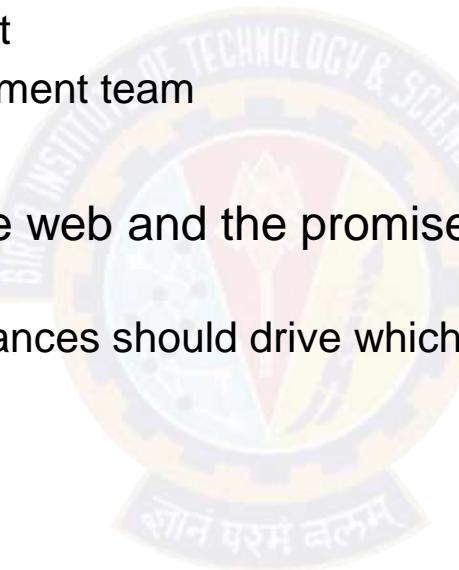
- Speed
  - ✓ Building for multiple platforms from a single codebase
  - ✓ often makes delivering cross-platform apps 2-3x faster than native
- Efficiency
  - ✓ Reduced development times
  - ✓ Avoided costs of hiring and retaining specialized native talent
  - ✓ can save teams 60% or more compared to native
- Design & UX consistency
  - ✓ With one codebase running on desktop, mobile, and web, hybrid apps provide better design and UX consistency across channels.
- Skillset
  - ✓ Hybrid gives web developers and businesses with in-house web teams the tools to build powerful mobile apps using their existing skills and talent



Put together, the advantages of hybrid have helped centralized development teams close the gap and better satisfy the demand for mobile apps for customers and internal employees.

# Comparing hybrid vs. native

- Important to keep in mind that the decision to choose hybrid or native should be based on
  - ✓ the unique goals of your organization,
  - ✓ the circumstances of a given project
  - ✓ the composition of existing development team
- While betting big on the power of the web and the promise and potential of cross platform hybrid development
  - ✓ understand that individual circumstances should drive which route you choose



Reference:  
Hybrid vs. Native  
Ionic Whitepaper



# Thank You!

In our next session:



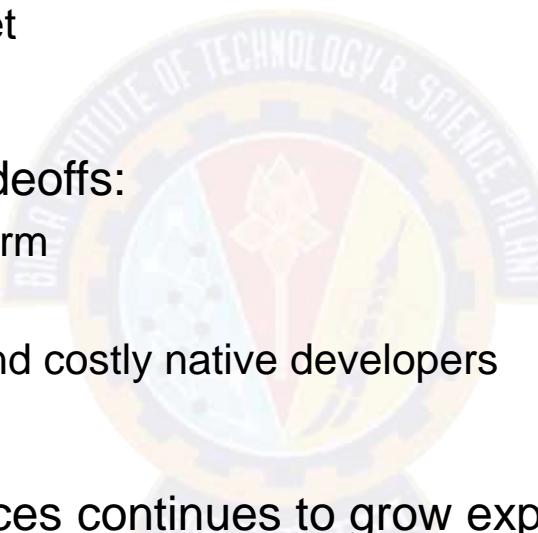
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# The mobile delivery gap

Chandan Ravandur N

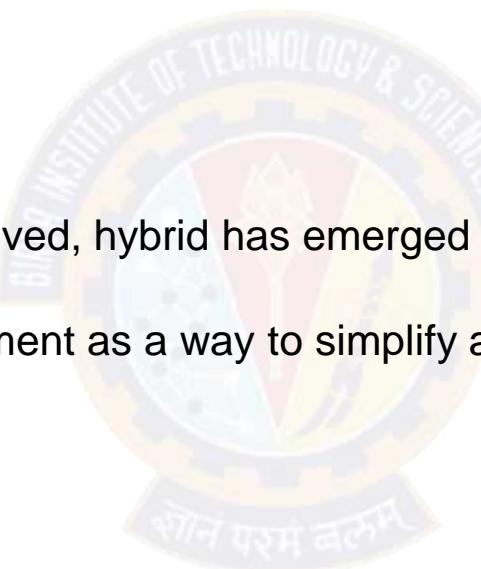
# The mobile delivery gap

- Back in the early days of mobile, there was really only one way to give the users the performance and features they expected
  - ✓ Had to use each platform's native toolset
- That came with real and measurable tradeoffs:
  - ✓ Building in parallel for each mobile platform
  - ✓ Managing multiple codebases
  - ✓ Hiring and retaining highly specialized and costly native developers
- Meanwhile, demand for mobile experiences continues to grow exponentially
  - ✓ By 2023, 80% of all enterprise software interactions are expected to occur on mobile



# Going Hybrid way

- Given the time and cost of traditional native development
  - ✓ it's no surprise that many development teams are struggling to keep up with this demand
- Times have changed
  - ✓ As mobile and web technology have evolved, hybrid has emerged as a viable alternative to native
  - ✓ Many are now looking at hybrid development as a way to simplify and speed up app creation

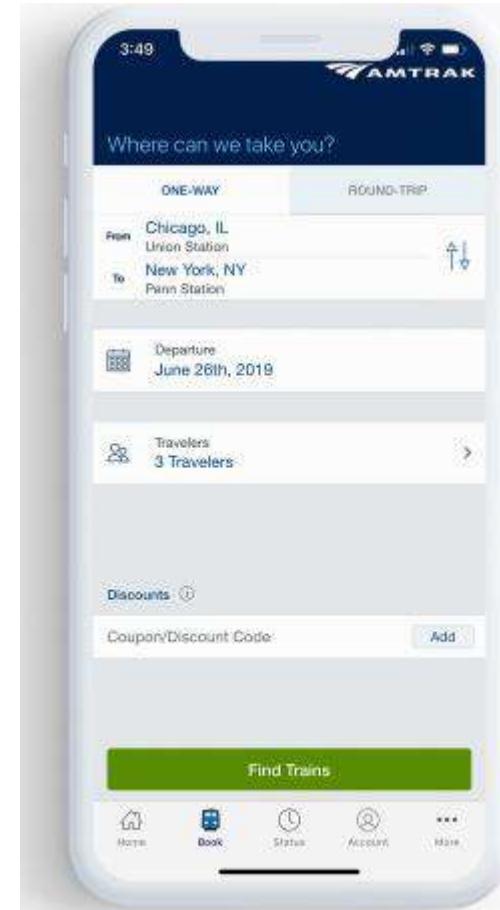
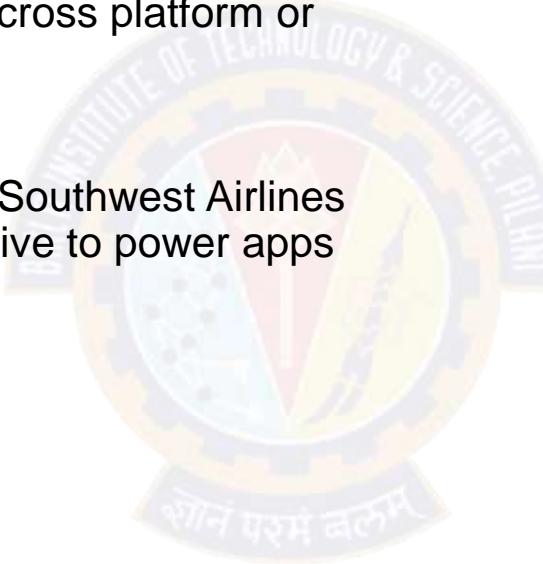


Demand for mobile experiences is growing 5x faster than internal IT teams can deliver.

GARTNER

# The choice to go hybrid

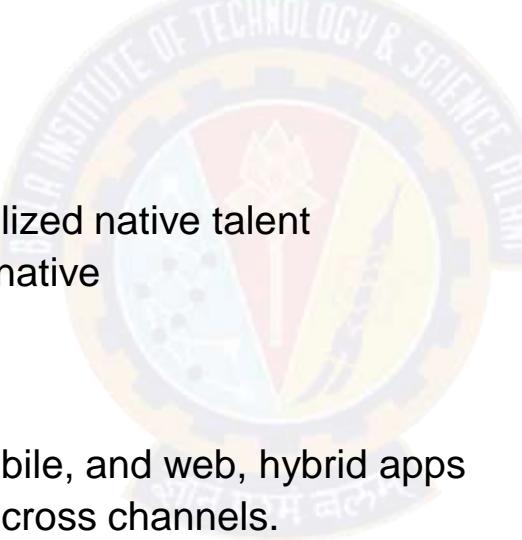
- The growing adoption of hybrid is evident in a Forrester survey
  - ✓ two-thirds of developers are choosing a cross platform or web-based approach over native tools
  - ✓ Top brands like Target, Nationwide, and Southwest Airlines have chosen a hybrid approach over native to power apps for their customers and employees



Source : Ionic whitepaper

# Top reasons for making the switch to hybrid

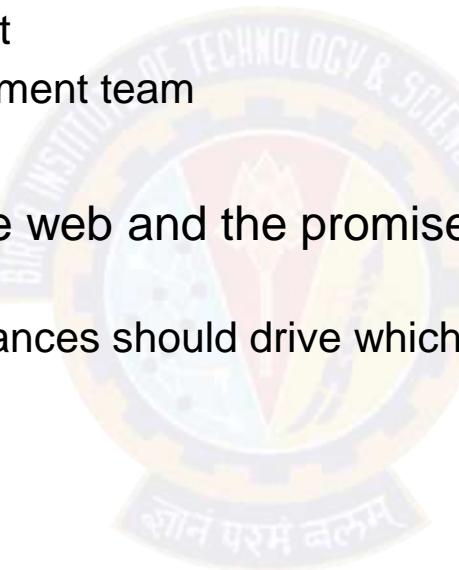
- Speed
  - ✓ Building for multiple platforms from a single codebase
  - ✓ often makes delivering cross-platform apps 2-3x faster than native
- Efficiency
  - ✓ Reduced development times
  - ✓ Avoided costs of hiring and retaining specialized native talent
  - ✓ can save teams 60% or more compared to native
- Design & UX consistency
  - ✓ With one codebase running on desktop, mobile, and web, hybrid apps provide better design and UX consistency across channels.
- Skillset
  - ✓ Hybrid gives web developers and businesses with in-house web teams the tools to build powerful mobile apps using their existing skills and talent



Put together, the advantages of hybrid have helped centralized development teams close the gap and better satisfy the demand for mobile apps for customers and internal employees.

# Comparing hybrid vs. native

- Important to keep in mind that the decision to choose hybrid or native should be based on
  - ✓ the unique goals of your organization,
  - ✓ the circumstances of a given project
  - ✓ the composition of existing development team
- While betting big on the power of the web and the promise and potential of cross platform hybrid development
  - ✓ understand that individual circumstances should drive which route you choose



Reference:  
Hybrid vs. Native  
Ionic Whitepaper



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Why native development?

Chandan Ravandur N

# Common reasons for choosing native



Performance



Rich native library

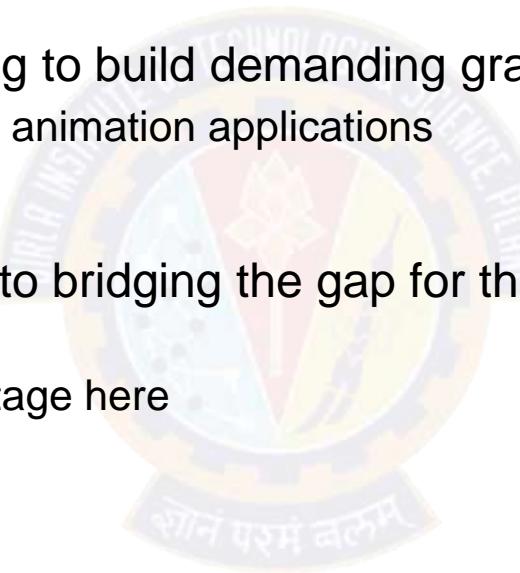


No third-party  
dependencies

# Benefits of native development

## Performance

- Native code is still faster than Javascript and HTML
- Matters when developers are looking to build demanding graphical applications
  - ✓ such as games and other intensive animation applications
- Mobile browsers are coming closer to bridging the gap for these types of intensive applications using WebGL specification
  - ✓ however, native still has the advantage here

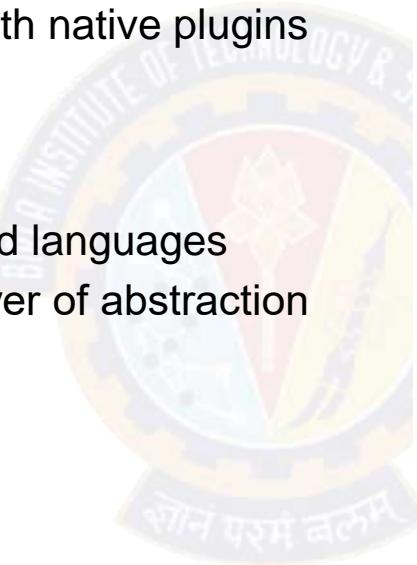


**PERFORMANCE**

# Benefits of native development (2)

## Rich native library

- Using native SDKs allows the developer
  - ✓ to access the latest features specifically designed for those platforms
  - ✓ without the complexity of dealing with native plugins
- A great option if developers
  - ✓ already familiar with native tools and languages
  - ✓ don't want or need an additional layer of abstraction



**RICH NATIVE LIBRARY**

# Benefits of native development(3)

## No third-party dependencies

- By building exclusively with a native SDKs
  - ✓ developers aren't bound to any third-party to keep up with support
- There's not as much of a dependency on open source communities like Cordova
  - ✓ to keep up with the latest features



**NO THIRD-PARTY DEPENDENCIES**

# Challenges of native development

## Longer development cycles

- Native apps usually have longer development cycles
  - ✓ especially when building for multiple platforms
  - ✓ which requires two or three different code bases for iOS, Android and desktop
- Each platform has its own nuances which require specific changes, updates, and maintenance
  - ✓ which bloat the cost of an application and add development time
- Creates a lot of iterations within the development process in order to customize and test for each platform
- Reduces agility of team to launch application or push updates



**LONGER DEVELOPMENT CYCLES**

# Challenges of native development(2)

## High development costs

- Developing mobile applications natively can be expensive and time-consuming
  - ✓ mostly driven by the time it takes to build for each platform,
  - ✓ along with the cost of hiring and retaining highly specialized native talent



# Challenges of native development (3)

## Limited customization

- Creating custom UI components to match company's design system or pattern library
  - ✓ can be more challenging with native components
  - ✓ because limited to the design patterns supported by each platform
- And unlike web components, native components can't be shared outside of their native platform
  - ✓ need to maintain multiple UI libraries



**LIMITED CUSTOMIZATION**

# Challenges of native development (4)

## Native talent hard to find

- Finding and hiring iOS and Android developers is difficult
  - ✓ given that less than 7% of all developers are versed in the necessary programming languages
- Also difficult to repurpose those developers for other projects outside of mobile



**NATIVE TALENT HARD TO FIND**

Reference :  
Native vs Hybrid  
Ionic Whitepaper



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Why Hybrid development?

Chandan Ravandur N

# What is a hybrid app?

- Hybrid apps are native apps
- They
  - ✓ are downloaded from the platform's app store or marketplace and offer the same native features, offline
  - ✓ support, and hardware-based performance acceleration as any app as built with a native SDK
- The key difference is that hybrid apps are built using open web technologies
  - ✓ like HTML, CSS, and JavaScript
  - ✓ rather than the proprietary or specialized languages used by iOS, Android, and others
- Run in a full-screen browser, called a webview, that is invisible to the user
  - ✓ Through customizable native plugins, they can access the native features of specific mobile devices (such as the camera or touch ID)
  - ✓ without the core code being tied to that device
- Cross-platform hybrid applications can run on any platform or device
  - ✓ all from a single codebase
  - ✓ while still delivering native performance



# Why hybrid?



Write once, run anywhere



Use the talent you  
already have



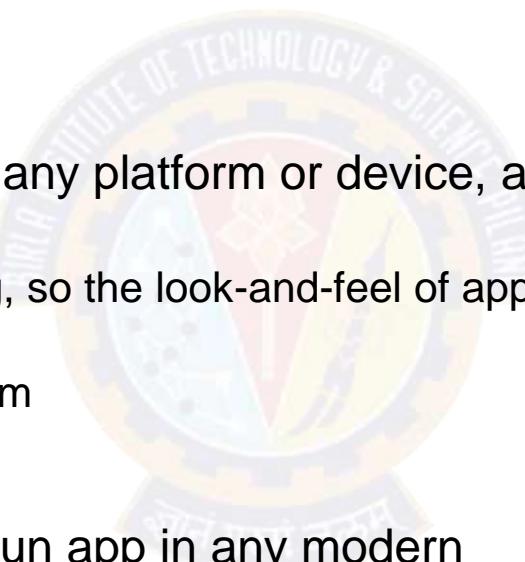
Deliver a great user  
experience across  
platforms



Build for the future

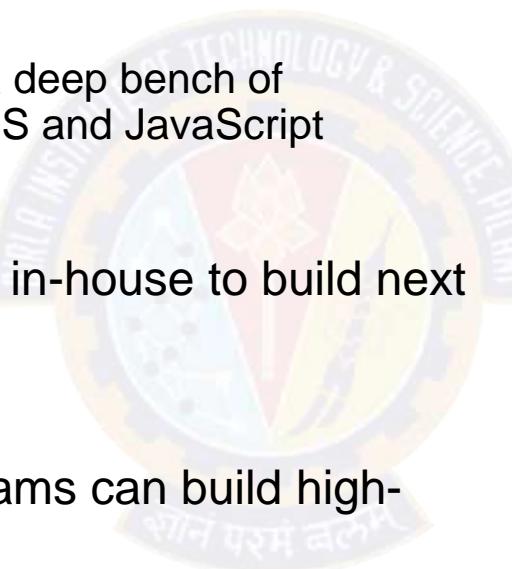
# Write once, run anywhere

- Rarely is a mobile app only designed for a single platform
  - ✓ Consumers, partners, and employees all have a choice of platforms and devices
- With a hybrid framework, can run app on any platform or device, all from a single codebase
  - ✓ Framework also provides adaptive styling, so the look-and-feel of app isn't one-size-fits-all
  - ✓ but it automatically adapts to each platform
- As hybrid technology is web-based, can run app in any modern browser as a Progressive Web App, or PWA
  - ✓ Means users get a great experience across platforms and devices



# Use the talent you already have

- According to the latest Stack Overflow survey
  - ✓ The web developer community is about 10x greater in size than the number of native mobile app developers
  - ✓ Many development teams already have a deep bench of programmers who understand HTML, CSS and JavaScript
- Why not leverage the talent already have in-house to build next mobile apps?
- With a hybrid framework, existing web teams can build high-performance apps
  - ✓ A lot easier than outsourcing development, or recruiting, training, and hiring specialists
  - ✓ Plus, centralizing on a single skill set makes it much easier to reassigned teams when a project is finished
  - ✓ whether that's a desktop web app or another mobile project



Traditional approach



Ionic approach

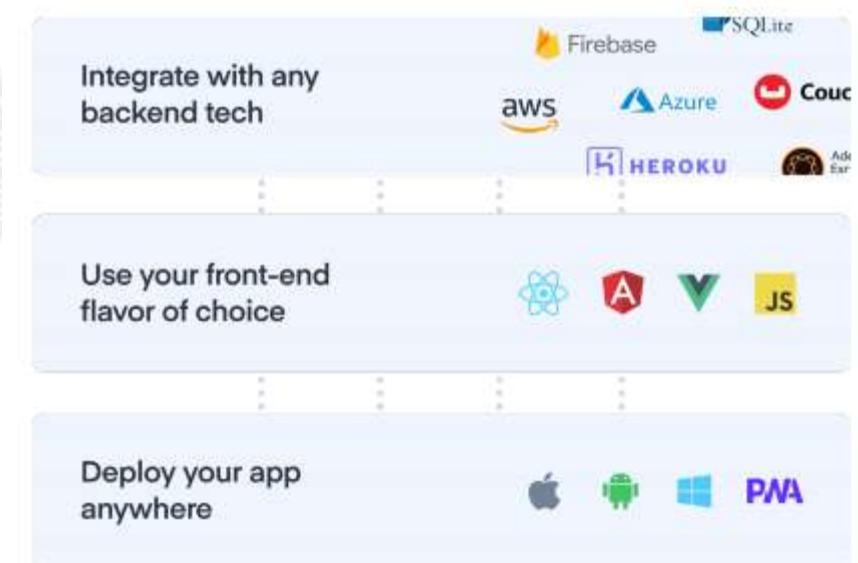
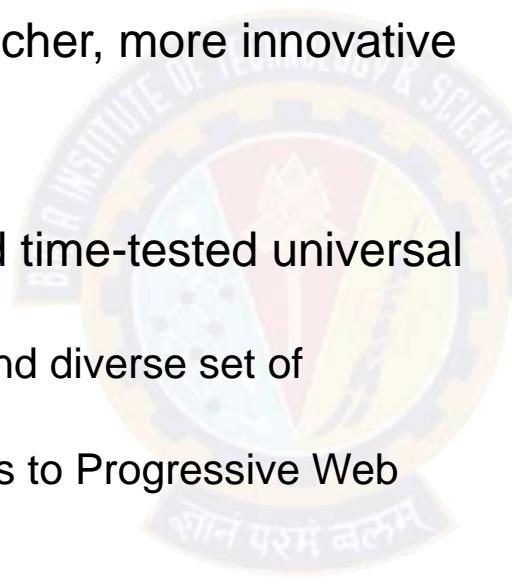


# Deliver the best UX across platforms

- Native advocates will claim that only a native approach can deliver the speed and performance to create a great user experience
  - ✓ Today's solutions offer the same hardware based performance acceleration as native apps
- User experience isn't just about performance
  - ✓ Simplifying the development process and consolidating onto a single codebase means more time to add features, fewer potential defects, and more time to fix bugs that find their way through
- Most importantly, users expect a seamless experience as they move across platforms and devices
  - ✓ If mobile app is completely out of sync with desktop or tablet app, that's not a good experience
  - ✓ By leveraging the web platform for both mobile and desktop solutions, will have a wider shared codebase
  - ✓ that will give that brand consistency and common user experience

# Build for the future

- Development organizations are tasked to build applications for the future
- Moving now to the web platform offers richer, more innovative options moving forward
- The web represents the most stable and time-tested universal runtime in the world
  - ✓ Today, the web is powering a growing and diverse set of applications
  - ✓ from traditional mobile and desktop apps to Progressive Web Apps, wearables, and IoT devices
- By choosing a hybrid framework based on open web standards
  - ✓ future-proof development strategy in some other important ways



# Drawbacks of hybrid



SYSTEM OVERHEAD



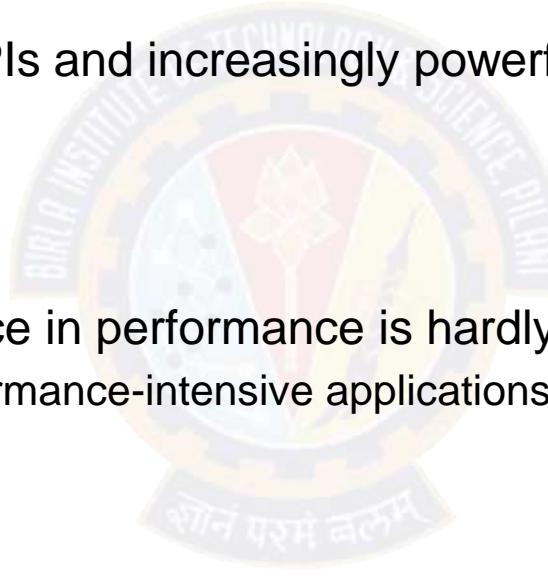
PLUGIN MANAGEMENT



THIRD-PARTY DEPENDENCE

# System overhead

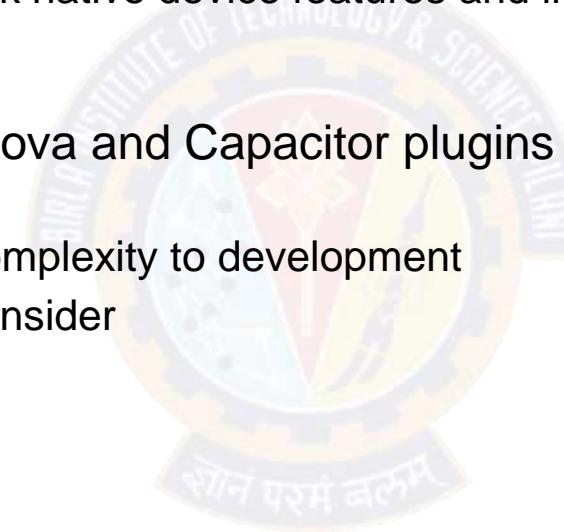
- The use of the webview may introduce a degree of overhead compared to native
- The abundance of performance APIs and increasingly powerful hardware have made this less of a factor in recent years
  - ✓ but it's still something to consider
- For most applications, the difference in performance is hardly noticeable
  - ✓ But for 3D games and other performance-intensive applications, a hybrid solution may not be the best choice.



**SYSTEM OVERHEAD**

# Plugin management

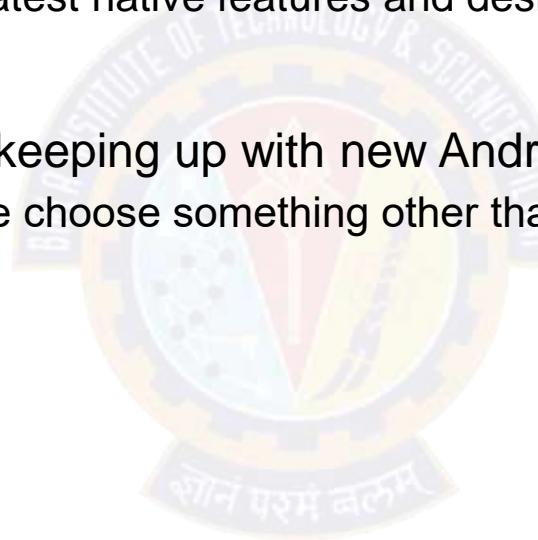
- Cross-platform solutions like Ionic—as well as React Native and others
  - ✓ are able to access nearly every native feature of a device, like the camera or gyroscope
  - ✓ by using native plugins that unlock native device features and integrations using basic JavaScript
- In a hybrid app, open source Cordova and Capacitor plugins are the most popular solution to this problem
  - ✓ Use of these plugins does add complexity to development
  - ✓ Nonetheless, this is a factor to consider



PLUGIN MANAGEMENT

# Third-party dependence

- Choosing a cross-platform approach means
  - ✓ are placing trust in the maker of the framework vendor (whether it's Ionic, React Native, Xamarin, etc.)
  - ✓ to keep up with the latest and greatest native features and design patterns of each mobile platform
- Though vendors are committed to keeping up with new Android and iOS versions
  - ✓ there's still a dependency any time choose something other than the native SDK



**THIRD-PARTY DEPENDENCE**

Reference:  
Native vs Hybrid?  
Ionic Whitepaper



# Thank You!

In our next session:



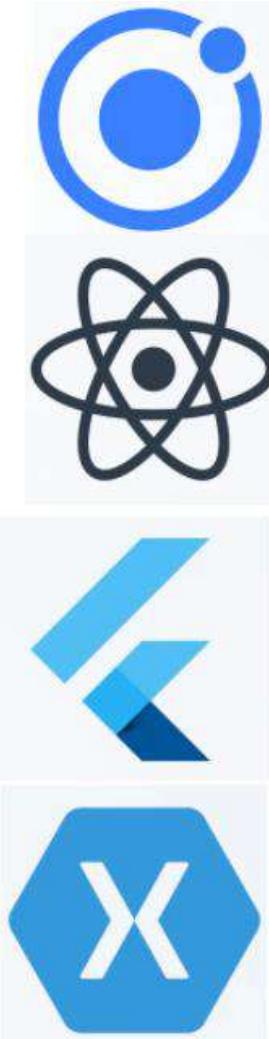
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Comparing Mobile App Approaches

Chandan Ravandur N

# Comparing Cross-Platform Frameworks

- Most people assume that all cross-platform frameworks alike
  - ✓ The decision to choose one over the other should be based on the stack that you're most comfortable with
- For example,
  - ✓ if you're a React shop, choose React Native
  - ✓ if you're an Angular shop, choose Ionic
- The truth of the matter is, this decision isn't nearly as simple as one would expect
  - ✓ Lets explore and understand the different approaches for the same



# Comparing Cross-Platform Frameworks(2)

	Native	Hybrid-Native	Hybrid-Web
Examples	iOS and Android SDKs	React Native, Xamarin, NativeScript, Flutter	Ionic
Languages	Obj-C, Swift, Java	JS + Custom UI Language / Interpreter	HTML + CSS + JS
Code Reuse	Totally Separate Code Bases per Platform	Shared Business Logic with Different UI Codebases	One codebase, UI codebase stays the same
Target Platforms	iOS & Android Native Mobile Apps	iOS & Android Native Mobile Apps	iOS, Android, Electron, Mobile and Desktop Browsers as a Progressive Web App, and anywhere else the web runs

# Comparing Cross-Platform Frameworks(3)

	Native	Hybrid-Native	Hybrid-Web
Investment	Largest investment in staff and time	Medium investment in staff and time	Lowest investment in staff and time
UI Elements	Native UI independent to each platform	A selection of Native UI elements for iOS and Android UI elements are specific to the target platform and not shared  Custom UI elements begin to require split UI code bases	Web UI elements that are shared across any platform, conforming to the native look & feel of wherever they are deployed
API Access / Native Features	Separate Native API & Codebases for each App	Abstracted Single-Codebase Native Access through Plugins (with ability to write custom Plugins)	Abstracted Single-Codebase Native Access through Plugins (with ability to write custom Plugins)
Offline Access	Available	Available	Available
Performance	“Native Performance” with well written code.	Indistinguishable difference on modern devices with well written code.	Indistinguishable difference on modern devices with well written code.

Reference:  
Comparing Cross-Platform Frameworks  
Ionic Blog



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Hybrid-Native vs. Hybrid-Web

Chandan Ravandur N

# Hybrid-Native vs. Hybrid-Web

## Where they are the same?

- The concept of allowing developers
  - ✓ to use technology that they already know
  - ✓ to build apps in the technologies they don't know
- is what all cross-platform frameworks are trying to achieve
- For example:
  - ✓ Ionic allows to use HTML/CSS/JS to build iOS apps, Android apps, PWAs, desktop apps, or apps for any other platform where the web runs
  - ✓ React Native utilizes the JS framework React, but renders native UI elements at runtime, allowing to build iOS and Android apps.
- In that way, all have the same mission:
  - Use technology X to build for platform Y, where X and Y traditionally aren't the same!

# Hybrid-Native

## Native UI, Shared Code

- React Native, along with Xamarin and NativeScript, allows to program user interfaces (UI) in one language
  - ✓ then orchestrates native UI controls at runtime
- It is “cross-platform” in that the React Native framework has mapped calls in JavaScript to calls that manage the native UI for specific mobile platforms
- For example, a React Native component that renders text on a mobile app will be translated into two separate components
  - ✓ [TextView] for Android and [UIView] for iOS
- Not sharing components across platforms, but sharing the code!
- That's why proponents of this approach will tout that building a real native app
  - ✓ JavaScript code runs and orchestrates native UI controls under the hood,
  - ✓ so that app UI is running (almost but not completely) natively
- Means that the underlying native UI component that you would like to use, as well as any customizations of that component
  - ✓ must be supported by React Native in order to use that component or change it
- Also means that, if you dream of a single, shared library of reusable UI components that you can use across all of your digital experiences
  - ✓ the Hybrid-Native model (i.e. React Native, Xamarin, NativeScript, et al) would not be the right approach

# Hybrid-Native (2)

## Shared Codebase, Plus Native Code

- The amount of code reuse in React Native project will depend on how much customizations are needed at the native layer
  - ✓ If sticking with fundamental UI primitives like View, Text, Image, and Touchable
  - ✓ then app code will run on each platform, meaning close to 100% code reuse
- On the other hand, if a lot of native customizations are needed, project will involve three separate codebases:
  - ✓ Two to manage UI for Android and iOS, plus a shared codebase where controller code will live
- Provides the ability to build a real native UI using mostly JavaScript, while enjoying a high degree of code sharing
- If want to mix a cross-platform solution
  - ✓ into an existing native codebase
  - ✓ have existing native skills
  - ✓ prefer native-specific languages over JS
  - ✓ or only plan to target iOS and Android
- Then go for Hybrid-Native frameworks!

# Hybrid-Web

## Web UI, Shared Components

- With Hybrid-Web frameworks like Ionic, instead of JavaScript code acting as a bridge that controls native UI elements
  - ✓ the UI components used in app are actually running across all platforms
  - ✓ In a mobile app, these components execute in a webview container
  - ✓ In a Progressive Web App, they run in the browser
  - ✓ In a native desktop app, they would run in a desktop container like Electron
- Why does this matter?
- Provide Design consistency
  - ✓ Can enforce UX and brand consistency across apps and digital experiences by sharing components across all digital platforms
  - ✓ Can make sure a component built for one app can be reused across any project
- Allows Portability
  - ✓ Based on HTML, CSS, and JavaScript, UI runs on a portable, standardized layer by utilizing open web standards
  - ✓ Will be able to run app on new platforms like Progressive Web Apps (PWAs) with no changes
- Provides Customizability
  - ✓ Able to fully customize every bit of their UI
  - ✓ Have complete and total control over the UI by utilizing the web, which offers a ton of out-of-the-box styling properties
  - ✓ Easier to do so when working in a single codebase with easy-to-use CSS styling on a per-platform basis
- Provides Stability
  - ✓ Biggest benefit of building on the web is the stability of the platform
  - ✓ With open web standards supported in all modern browsers, can be confident that whatever code written today is going to work tomorrow

# Hybrid-Web(2)

## One Codebase, Running Anywhere

- With a Hybrid-Web approach, UI is built with HTML/CSS/JS
  - ✓ with native functionality being accessed through portable APIs (or “plugins”)
  - ✓ that abstract the underlying platform dependencies
- Instead of entire UI depending on the native platform
  - ✓ only certain native device features, like the Camera, are platform-dependent
- Allows app to run anywhere the web runs
  - ✓ iOS, Android, Browsers, Desktop/Electron, PWAs & so on
- Means that UI layer can be shared between all platforms
  - ✓ Even if you customize the look & feel , you'll never need to split your app into multiple codebases
  - ✓ Get the benefit of debugging multiple platforms at the same time by utilizing web-based debugging tools, like the Chrome Developer tools

# Hybrid-Web(2)

## When to use?

- Use (“Hybrid-Web”) approach to cross-platform development:
- If you
  - ✓ are looking to build an application not just for mobile (iOS and Android), but for desktops and browsers as well
  - ✓ don’t want to have to rewrite your UI for each target platform
  - ✓ want to have UI customization and design consistency across all of your target platforms
  - ✓ want to open up the possibilities of your UI to custom components or third-party libraries
  - ✓ want to hire from the huge pool of existing web development talent and build out a JavaScript organization

Reference:  
Comparing Cross-Platform Frameworks  
Ionic Blog



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

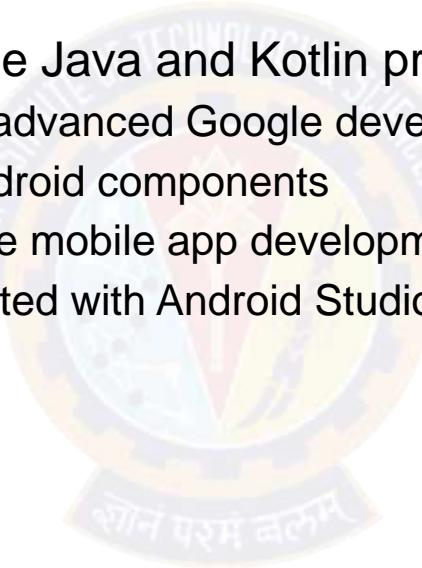
# Android iOS – Pro Cons

Chandan Ravandur N

# Tech Stack

## Developing Android projects

- Android operating system has an open-source code
- To create Android apps, developers use Java and Kotlin programming languages
  - ✓ Android developers use the following advanced Google development tools:
  - ✓ Android Jetpack, a set of pre-build Android components
  - ✓ Firebase is known as a comprehensive mobile app development platform
  - ✓ Android SDK development kit, connected with Android Studio, an integrated development environment

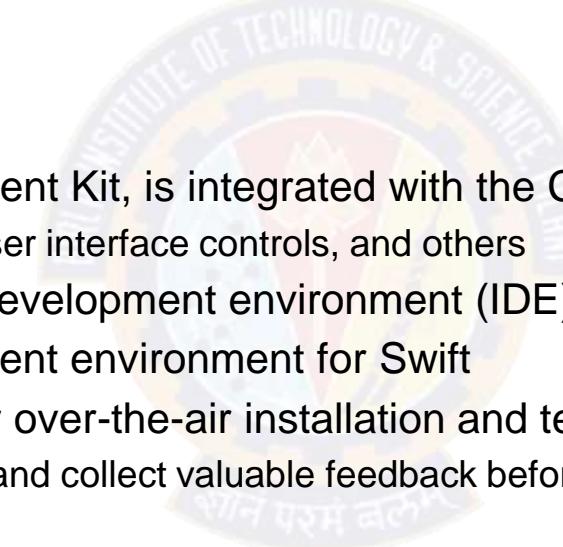


stackoverflow

# Tech Stack (2)

## Developing iOS projects

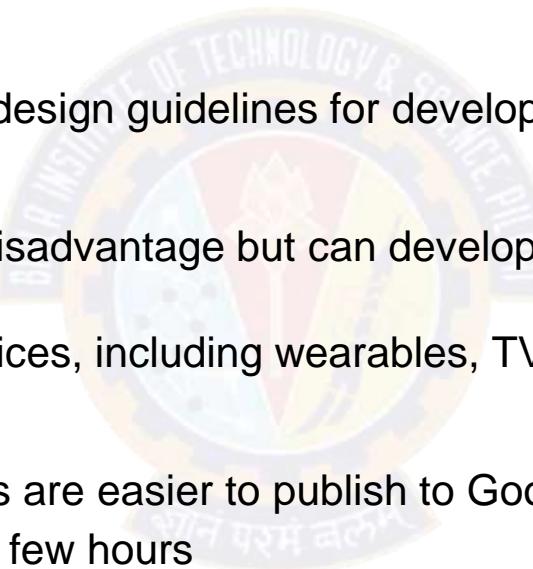
- iOS has closed source code - means that iOS works on Apple devices only
- The development team might use Swift or Objective-C
- iOS developing tools include:
  - ✓ iOS SDK, or Software Development Kit, is integrated with the Cocoa Touch UI framework
    - ❖ provides graphical elements, user interface controls, and others
  - ✓ XCode is the official integrated development environment (IDE) for iOS development
  - ✓ Swift Playgrounds is a development environment for Swift
  - ✓ TestFlight is an online service for over-the-air installation and testing
    - ❖ allows developers to test apps and collect valuable feedback before the app release



# Pros and Cons

## Advantages of developing Android

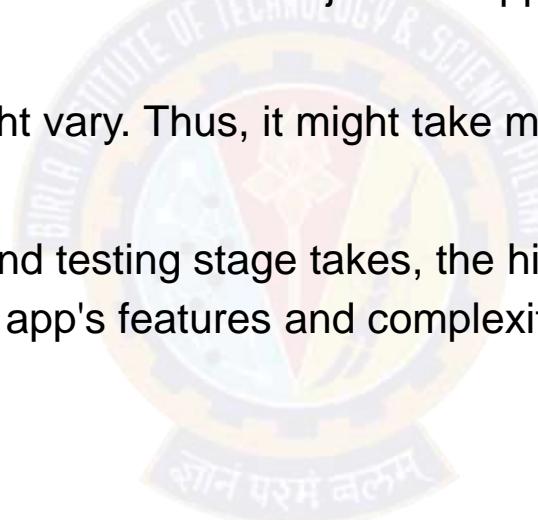
- Open system
  - ✓ Android developers receive access to more features, restricted in iOS applications
- Design
  - ✓ Developers use extensive Google design guidelines for developing an intuitive user interface
- Fragmentation
  - ✓ May consider fragmentation as a disadvantage but can develop apps not only for an Android smartphone
  - ✓ but also for a broader range of devices, including wearables, TVs, in-car systems, and others
- Release
  - ✓ In comparison to iOS, Android apps are easier to publish to Google Play
  - ✓ The whole process may take just a few hours



# Pros and Cons (2)

## Disadvantages of Android

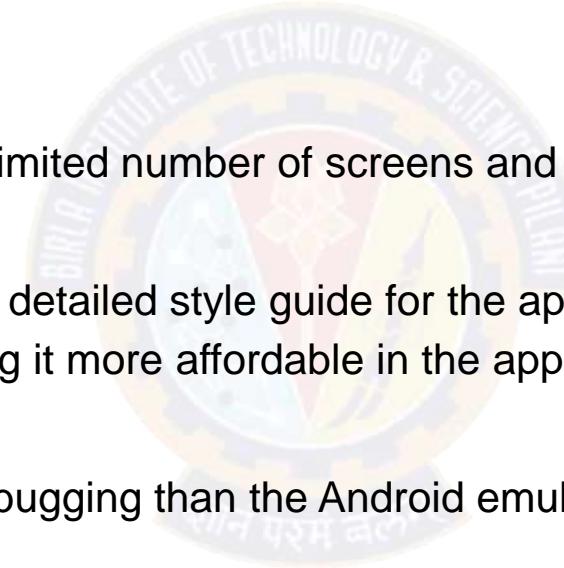
- Fragmentation
  - ✓ May also be an Android drawback - Android devices come in different screen sizes, resolutions, etc.
  - ✓ The development team might need more time to adjust the app's features for particular devices
- Testing
  - ✓ Android versions and devices might vary. Thus, it might take more time for QA specialists to test app
- Costly
  - ✓ The more time the development and testing stage takes, the higher will be the Android app price
  - ✓ But the price also depends on the app's features and complexity



# Pros and Cons (3)

## Advantages of iOS

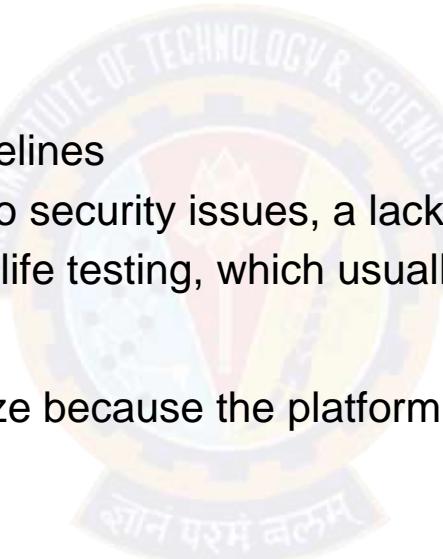
- Revenue
  - ✓ Apple users spend more money on app purchases compared to Android users
- The number of devices
  - ✓ iOS powers only Apple products
  - ✓ For that reason, app should fit a limited number of screens and devices
- UI design
  - ✓ Apple provides developers with a detailed style guide for the app UI
  - ✓ The team needs less time, making it more affordable in the app design stage
- Simulator
  - ✓ The iOS emulator is better for debugging than the Android emulator



# Pros and Cons (4)

## Disadvantages of iOS

- Expensive
  - ✓ The Xcode IDE can only be used on Macs. This significantly increases initial investment before you even start developing
- Stringent App release
  - ✓ App Store has quite strict review guidelines
  - ✓ The marketplace can reject app due to security issues, a lack of valuable content, or poor performance
  - ✓ Developer should submit app for real-life testing, which usually takes a few days
- Flexibility
  - ✓ iOS apps are usually hard to customize because the platform has many restrictions



# Comparing Android vs Apple Development

Key aspects	iOS-based platform	Android-based platform
Development language	Swift	Java, Kotlin
Integrated development environment	Xcode	Android Studio
The target audience	More valuable	Less valuable
Design philosophy	Flexible	Specified requirement
Development complexity	Middle	High
Development time	Depends on complexity	Depends on complexity
Apple Store and Play Store Acceptance and Deployment speed	Long app review by app stores  (7 days on average)	Short app reviews process

Reference:

[ANDROID APPS VS. IOS APPS - WHAT AND WHY IS BETTER IN 2021?](#)



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

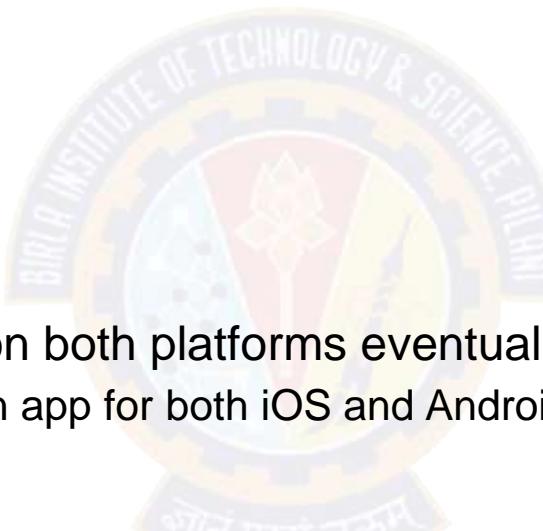
# Android vs iOS

Chandan Ravandur N

# Android vs iOS

## Which Platform to Build Your App for First?

- Deciding whether to build a business app for Android or iOS (Apple) depends on 5 factors
  - ✓ your audience
  - ✓ project timeline
  - ✓ desired app features
  - ✓ app maintenance budget
  - ✓ and revenue goal
- While your goal may be to launch on both platforms eventually
  - ✓ it is risky and expensive to build an app for both iOS and Android
- Instead, most developers choose to build an app for one platform to start
  - ✓ then launch the app on the other platform later
  - ✓ once the first version of the app is established and successful.

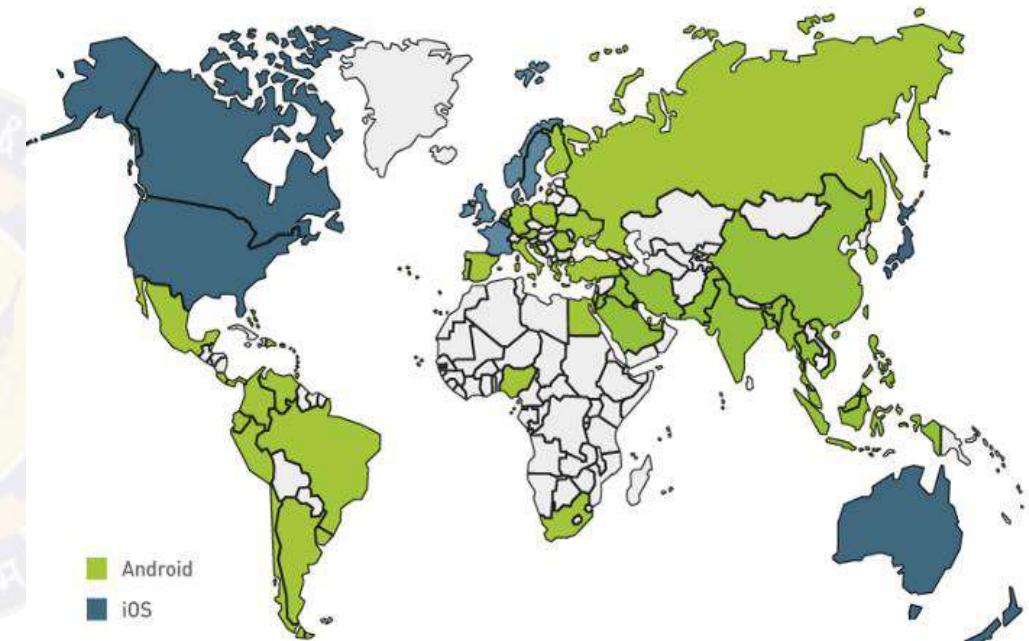
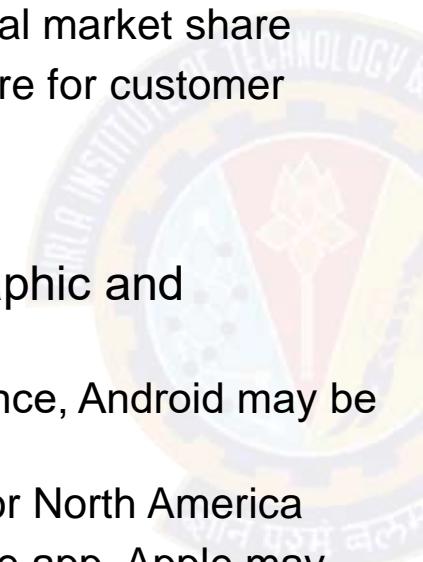


fitzweekly

# Building an App for Android or iOS

## Depends on Your Audience

- There are key differences in the users Android and iOS attract
  - ✓ Android devices have the greatest global market share
  - ✓ However, Apple dominates the App Store for customer spending
- Consider your target audience's geographic and demographic characteristics
  - ✓ If you're targeting a broad global audience, Android may be your best bet
  - ✓ If your audience is in Western Europe or North America
  - ✓ or if you're an e-commerce or enterprise app, Apple may be a better choice



App Annie

# Making an App

## for iOS is Faster and Less Expensive

- It's faster, easier, and cheaper to develop for iOS
  - ✓ some estimates put development time at 30–40% longer for Android
- One reason why iOS is easier to develop for is the code
  - ✓ Android apps are generally written in Java, a language that involves writing more code than Swift, Apple's official programming language
- Another reason is that Android is an open source platform
  - ✓ A lack of standardization means more devices, components, and software fragmentation to account for
  - ✓ Apple's closed ecosystem means you're developing for a few standardized devices and operating systems
- Apple App Store has stricter rules and quality expectations and a longer review process, so it may take longer for apps to be approved
  - ✓ Your app may even be rejected if it's not up to Apple's standards

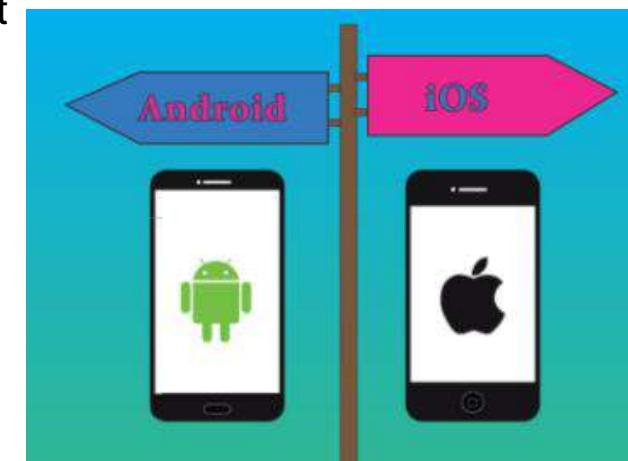
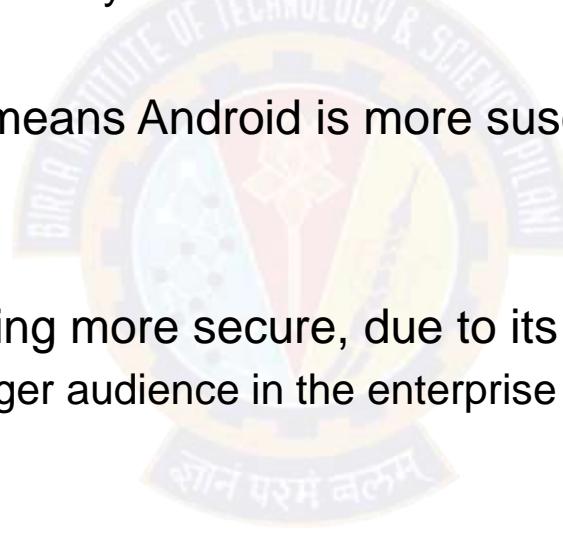


medium

# Developing an App for Android

## Allows More Flexibility With Features

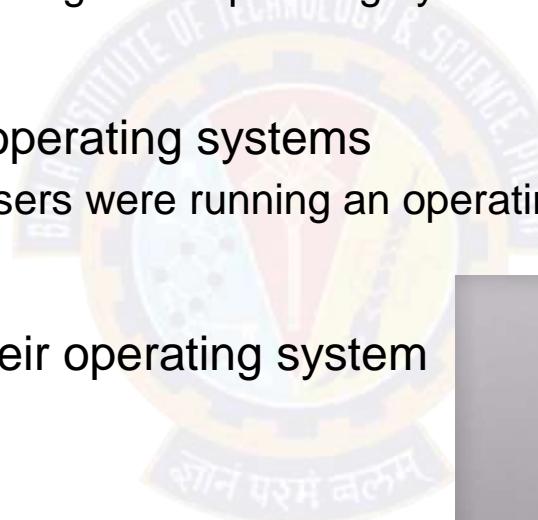
- What features will you offer through your business app?
  - ✓ Because Android is open source, there's more flexibility to customize your app
  - ✓ building the features and functions that your audience wants
- Of course, this open environment means Android is more susceptible to pirate apps and malware
- Apple is generally perceived as being more secure, due to its closed nature
  - ✓ which is largely why iOS has a larger audience in the enterprise market



# Maintaining App on Android or iOS

## Is Easier If Users Update OS

- Developing for Android may mean
  - ✓ spending more time ensuring that your app remains compatible with the platform
  - ✓ avoiding bugs and crashes for users running older operating systems
- Android users are slower to adopt new operating systems
  - ✓ One study found over 50% of Android users were running an operating system more than 2 years old
- Apple users are more likely to update their operating system



# Monetizing Your App

## Depends on Likelihood of Making Purchases on iOS or Android

- Android users tend to be less willing to pay for apps than iOS users, so free apps with in-app ads are more common.
  - ✓ The Apple App Store generates twice as much revenue as Google Play, despite having half as many downloads
- Apple users are more likely to make in-app purchases and spend more on them.
- Apple users are more likely to pay for apps, except in one category — utility apps. Android outperforms when it comes to utility apps.
- Shopping apps generate the most revenue, and North America leads in sales.
- Asian users spend 40% more on in-app purchases, and China is the biggest driver of iOS revenue.
- Both Google and Apple are pushing subscriptions and offering developers a larger cut of the revenue.
- The takeaway
  - ✓ If you want to monetize your app without ads, whether through subscriptions, freemium models, or in-app purchases,
  - ✓ then Apple may be your best bet. The same applies to e-commerce apps.
- However, revenue from Android apps is growing
  - ✓ Though Apple is expected to remain the dominant App Store, Google Play and third-party Android stores combined
  - ✓ are expected to overtake the App Store in terms of revenue

# Android vs Apple

## Choose One Platform to Develop for First

- Whether you should build a business app for Android or iOS first depends on
  - ✓ Where your audience lives
  - ✓ who they are
  - ✓ the features they want
  - ✓ development timeline
  - ✓ and budget
- If you need to build a minimum viable product quickly and cheaply, then iOS may be the way to go
- If you have want to make money with your app or build an e-commerce app, look to iOS
- If you're targeting global or emerging markets, especially in Asia and Latin America, or
- If your app involves features that Apple doesn't support, then Android is your best bet
- Whichever platform you choose
  - ✓ once you've worked out the kinks and built a user base on your initial platform
  - ✓ you may find that users on the other platform begin clamoring for a version of your app
- When this happens, it's a sign that it's time to build another version of your app for the other platform



computerworld

Reference:

[Android vs iOS: Which Platform to Build Your App for First?](#)



# Thank You!

In our next session:



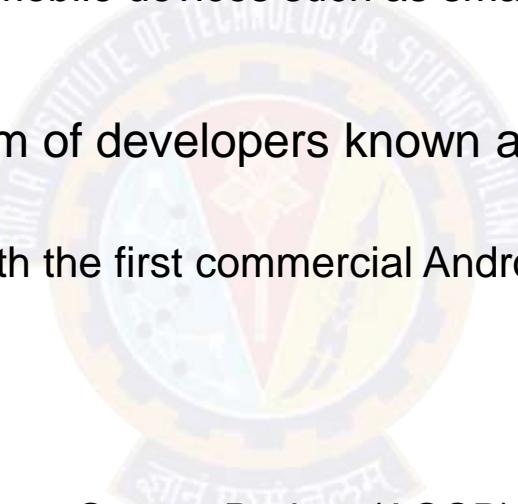
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Android Platform

Chandan Ravandur N

# Android (operating system)

- Android is a mobile operating system based on a modified version of the Linux kernel and other open source software
  - ✓ designed primarily for touchscreen mobile devices such as smartphones and tablets
- Android is developed by a consortium of developers known as the Open Handset Alliance
  - ✓ commercially sponsored by Google
  - ✓ was unveiled in November 2007, with the first commercial Android device launched in September 2008
- Free and open source software
  - ✓ source code is known as Android Open Source Project (AOSP), which is Apache License



# Android platform

- Google's Andy Rubin describes Android as follows:

"The first truly open and comprehensive platform for mobile devices. It includes an operating system, user-interface and applications — all of the software to run a mobile phone but without the proprietary obstacles that have hindered mobile innovation."

— Where's My Gphone?

(<http://googleblog.blogspot.com/2007/11/wheres-my-gphone.html>)



New York times

# Android ecosystem

- Free, open source operating system
- Open source development platform for applications
- Android devices
- Operating system for mobile devices
- Application development platform
- Developed & managed by OHA



# Open Source Alliance



# Android Timeline



# Why Android ?

- No certification for developer
- Google Play for application distribution
- No approval process for distribution
- Developers controls brand
- Android devices
  - ✓ Smartphones
  - ✓ Tablets
  - ✓ E-reader devices
  - ✓ Netbooks
  - ✓ MP4 players
  - ✓ Internet TVs



BrowserStack

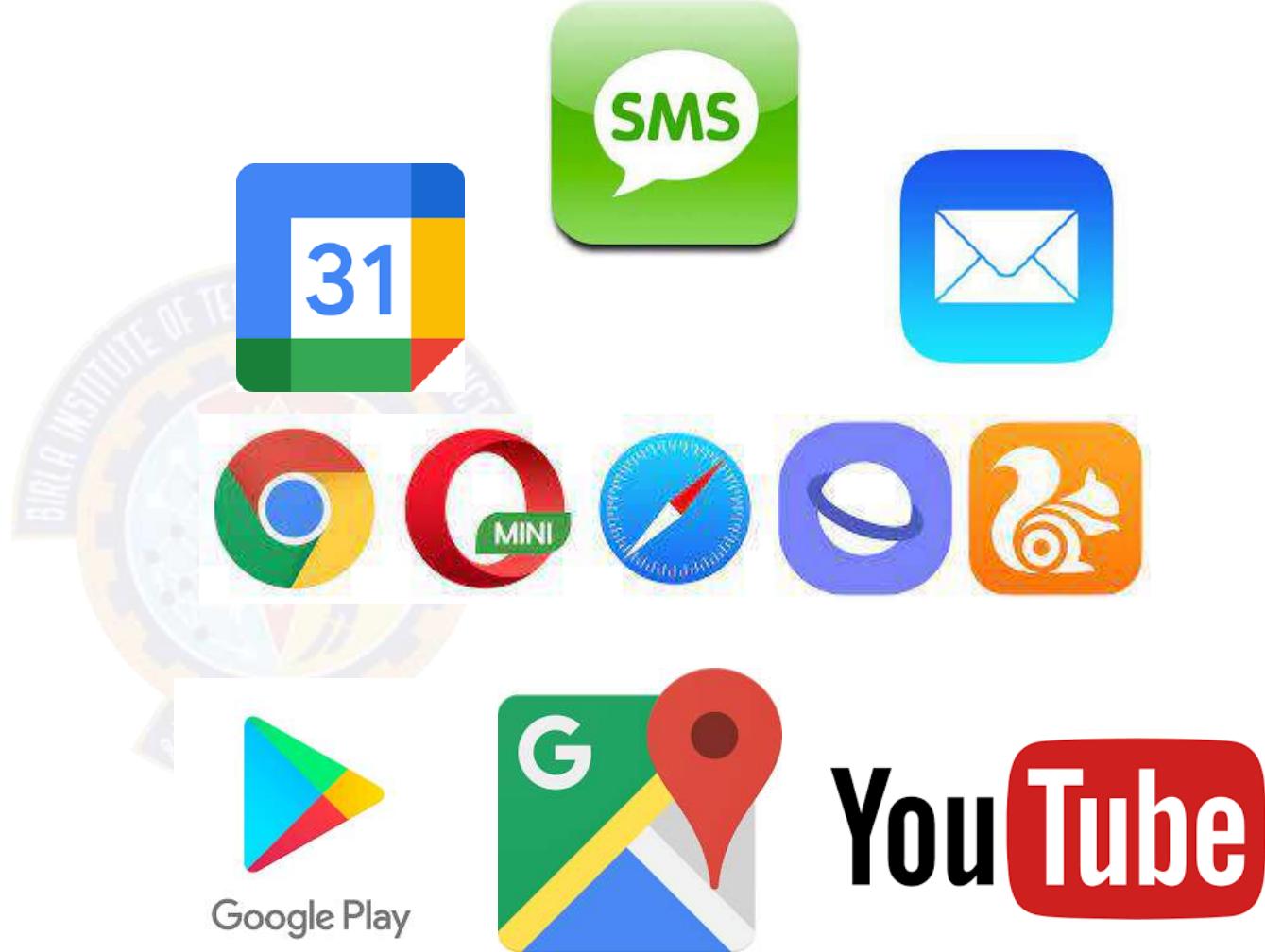
# Android features

- Connectivity – GSM , Bluetooth, Wi-Fi etc
- Storage – SQLLite , light weight relational database for data storage
- Messaging – supports SMS, MMS
- Web browser – open source web kit based
- Media support – MPEG, MP4, JPEG, PNG, GIF, BMP
- Hardware support – Camera, GPS
- Multi touch screens
- Flash support
- Multi tasking



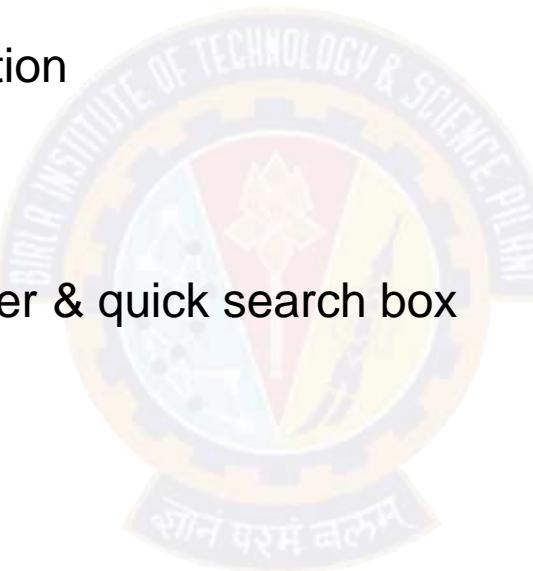
# Applications

- Pre-installed applications :
  - ✓ SMS Management application
  - ✓ E-Mail client
  - ✓ Web browser
  - ✓ Calendar & Contacts
  - ✓ Music player
  - ✓ Picture gallery
  - ✓ Camera, video recording application
  - ✓ Calculator
  - ✓ Home screen
  - ✓ Alarm clock
- Google applications :
  - ✓ Google play store
  - ✓ Google Maps
  - ✓ Gmail
  - ✓ Google talk
  - ✓ YouTube video player



# What's different ?

- Google Maps application
- Background services & applications
- Shared & inter process communication
- All applications are equal
- Wi-Fi Direct & Android Beam
- Home screen widgets, Live wallpaper & quick search box





# Thank You!

In our next session:

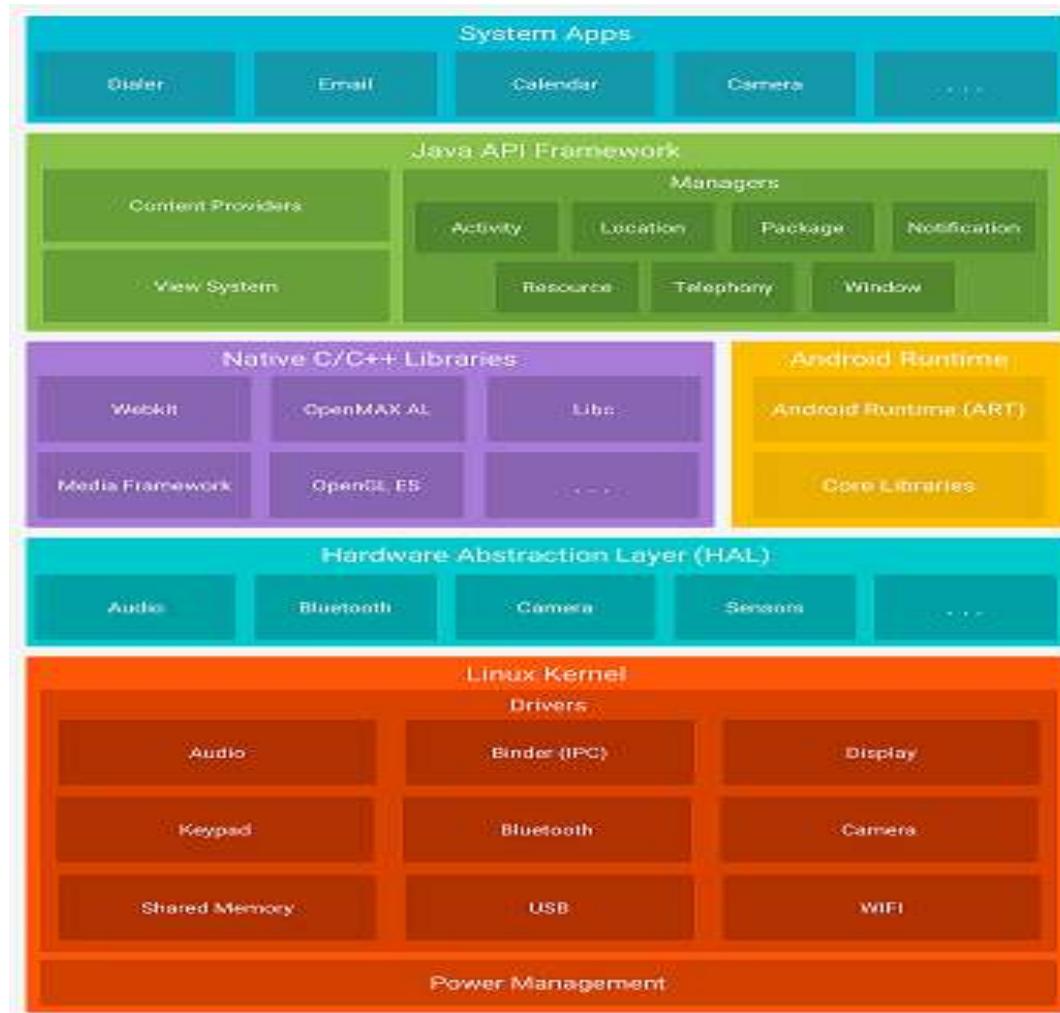


**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Android Architecture

Chandan Ravandur N

# Android architecture



# Android architecture (2)

## Application layer

- Core applications
  - ✓ Email client
  - ✓ Web browser
  - ✓ Calendar , contacts
- Third party applications
- Google applications
  - ✓ Maps
  - ✓ Play store
- Uses JAVA language



# Android architecture (3)

## Application framework

- Abstract classes used to create applications
- Generic abstraction for hardware access
- Classes to manage user interface
- Classes to deal with application resources
- Enables and simplifies component reuse
  - ✓ Framework APIs
  - ✓ Component reuse



# Android architecture (4)

## Set of services

Feature	Role
View System	Used to build an application, including lists, grids, text boxes, buttons, and embedded web browser
Content Provider	Enabling applications to access data from other applications or to share their own data
Resource Manager	Providing access to non-code resources such as localized strings, graphics, and layout files
Notification Manager	Enabling all applications to display custom alerts in the status bar
Activity Manager	Manages the lifecycle of applications and provides a common navigation backstack

# Android architecture (5)

## Android native libraries

- Code to access features of Android OS
- Set of C/C++ libraries
- Compiled for particular hardware architecture used by phone
- Preinstalled by phone vendor
- SQLite library for database support
- WebKit library for web browsing
- Accessible to developers through application framework



# Android architecture (6)

## Android runtime

- Core java libraries
  - ✓ Enables developers to write application using Java
- APIs
  - ✓ - data structures
  - ✓ - utilities
  - ✓ - file access
  - ✓ - network access
  - ✓ - graphics



# Android architecture (7)

## Dalvik JVM

- Dalvik JVM
  - ✓ Every application run in its own process, with its own instance of Dalvik Virtual machine
  - ✓ Specialized VM for Android
  - ✓ Optimized for battery powered mobile devices with limited memory & CPU
- executes files in dalvik executable format (.dex)
- Runs classes compiled by Java compiler transformed into .dex format
- Relies on linux kernel for low level functions like memory management, threading



# Android architecture (8)

## Linux kernel

- Android relies on Linux kernel for system services like

- ✓ Memory management
- ✓ Process management
- ✓ Network stack
- ✓ Security



- Acts as abstraction layer between hardware
- and rest of software stack



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

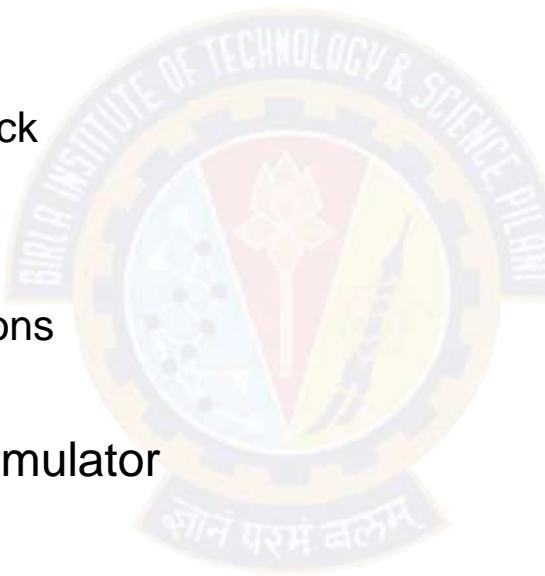
# Android development tools

Chandan Ravandur N

# Android development tools

## Android SDK

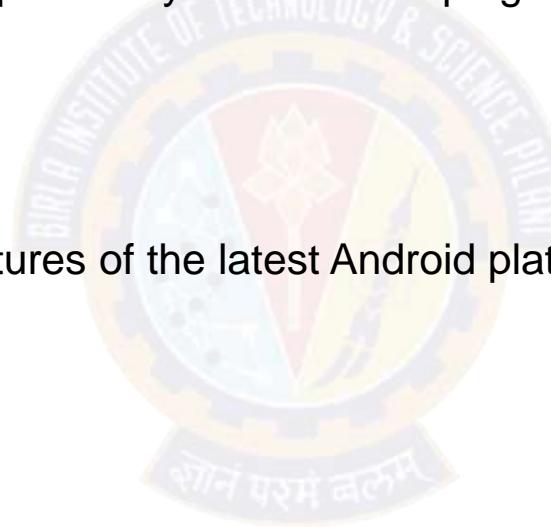
- Tools & utilities to create, test & debug applications
- Android APIs
  - ✓ API libraries to access android stack
- Development tools
  - ✓ tools to compile & debug applications
- Android virtual device manager & emulator
- Full documentation
- Sample code
- Online support



# Android development tools(2)

## Types of tools

- SDK tools
  - ✓ Platform independent
  - ✓ required no matter which Android platform you are developing on
- Platform tools
  - ✓ Android Platform dependent
  - ✓ are customized to support the features of the latest Android platform.



# Android development tools(3)

## SDK tools

- installed with the SDK starter package
- periodically updated
- required if you are developing Android applications
- commonly used tools -
  - ✓ Android SDK Manager (android sdk)
  - ✓ the AVD Manager (android avd)
  - ✓ the emulator (emulator)
  - ✓ the Dalvik Debug Monitor Server (ddms)



# Android development tools(4)

- Android SDK manager
  - ✓ used to see SDK versions installed
  - ✓ to install new SDKs when released
  - ✓ each platform release is displayed
    - ❖ Platform tools
    - ❖ Number of additional support packages
- AVD manager
  - ✓ to create and manage virtual devices that hosts emulators
  - ✓ AVDs are used to simulate software builds and hardware configurations available on different physical devices
  - ✓ Each virtual device has
    - ❖ Name
    - ❖ Target build of android
    - ❖ SD card capacity
    - ❖ Screen resolution



# Android development tools(5)

- Emulator
  - ✓ Testing & debugging applications
  - ✓ Implementation of Dalvik VM
  - ✓ Platform for running android applications as android phone
  - ✓ Decoupled from any hardware
  - ✓ Full network connectivity
  - ✓ Simulating placing & receiving voice calls & SMS messages
  - ✓ Need to create virtual device
  - ✓ Emulator launches virtual device & run Dalvik instance within it.
- DDMS
  - ✓ debugging tool
  - ✓ Helps to see what's happening under the surface
  - ✓ View processes
  - ✓ View the stack & heap
  - ✓ Watch & pause active threads

# Android development tools(7)

- Logcat
  - ✓ To view & filter output of logging system
- SQLite3
  - ✓ Database tool to access SQLite database files
- Traceview
  - ✓ Graphical analysis tools for viewing trace logs from application
- Dx
  - ✓ Converts .class Java bytecode into Android .dex bytecode



# Android development tools(8)

## Platform tools

- typically updated every time you install a new SDK platform
- Each update of the platform tools is backward compatible with older platforms
- only one of the platform tools—the Android Debug Bridge (adb)
- Android debug bridge
  - ✓ Client service application to connect to AVD
  - ✓ Consists of
    - ❖ A daemon running on device or Emulator
    - ❖ A service running on development computer
    - ❖ Client applications (DDMS) that communicates with daemon through service
  - ✓ to install application on device
  - ✓ Push/pull files to/from device
  - ✓ Run shell commands on target device
  - ✓ Query or modifies SQLite database on the device



# Thank You!

In our next session:



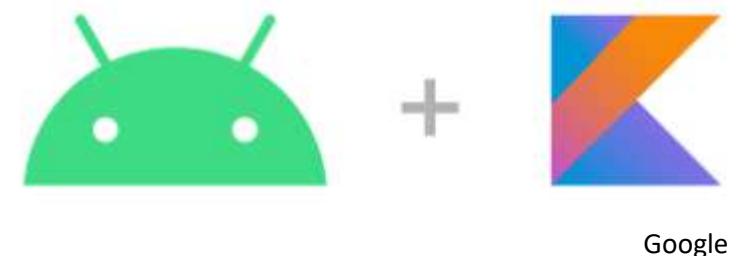
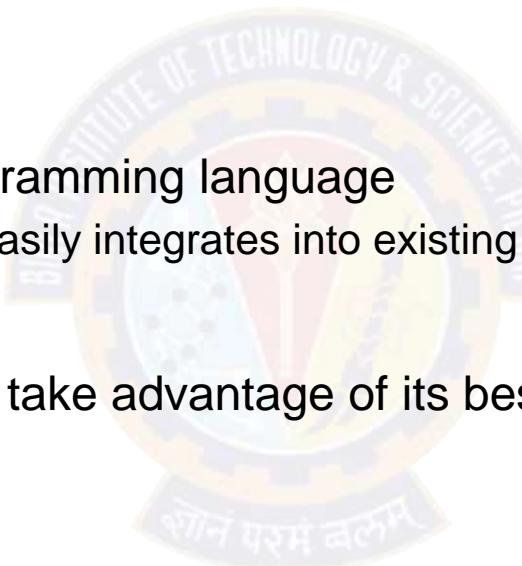
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Android's Kotlin-first approach

Chandan Ravandur N

# Android's Kotlin-first approach

- At Google I/O 2019, Google announced that Android development will be increasingly Kotlin-first
  - ✓ they have stood by that commitment!
- Kotlin is an expressive and concise programming language
  - ✓ that reduces common code errors and easily integrates into existing apps
- Recommended for starting with Kotlin to take advantage of its best-in-class features for Android app development
- In an effort to support Android development using Kotlin
  - ✓ co-founded the Kotlin Foundation and have ongoing investments in improving compiler performance and build speed



Google

# Why is Android development Kotlin-first?

- Google Reviewed feedback that came directly from developers at conferences, Customer Advisory Board (CAB), Google Developers Experts (GDE), developer research
  - ✓ Many developers already enjoy using Kotlin
  - ✓ The request for more Kotlin support was clear
- Here's what developers appreciate about writing in Kotlin:
- Expressive and concise
  - ✓ Can do more with less
  - ✓ Express ideas and reduce the amount of boilerplate code
  - ✓ 67% of professional developers who use Kotlin say Kotlin has increased their productivity
- Safer code
  - ✓ Kotlin has many language features to help you avoid common programming mistakes such as null pointer exceptions
  - ✓ Android apps that contain Kotlin code are 20% less likely to crash
- Interoperable
  - ✓ Call Java-based code from Kotlin, or call Kotlin from Java-based code
  - ✓ Kotlin is 100% interoperable with the Java programming language
- Structured Concurrency
  - ✓ Kotlin coroutines make asynchronous code as easy to work with as blocking code
  - ✓ Coroutines dramatically simplify background task management for everything from network calls to accessing local data



Google

# What does Kotlin-first mean?

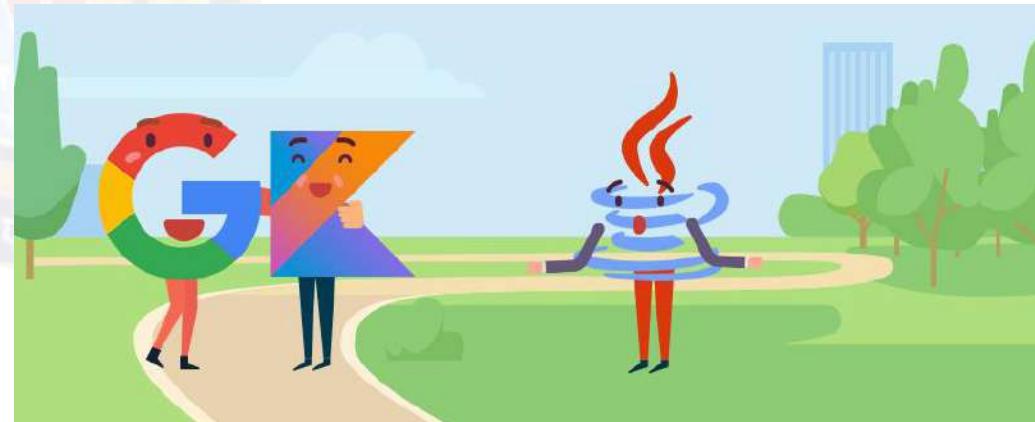
- When building new Android development tools and content
  - ✓ such as Jetpack libraries, samples, documentation, and training content
  - ✓ Google will design them with Kotlin users in mind while continuing to provide support for using APIs from the Java programming language



	Java language	Kotlin
Platform SDK support	Yes	Yes
Android Studio support	Yes	Yes
Lint	Yes	Yes
Guided docs support	Yes	Yes
API docs support	Yes	Yes
AndroidX support	Yes	Yes
AndroidX Kotlin-specific APIs (KTX, coroutines, and so on)	N/A	Yes
Online training	Best effort	Yes
Samples	Best effort	Yes
Multi-platform projects	No	Yes
Jetpack Compose	No	Yes

# Google use Kotlin, too!

- Google engineers enjoy the language features Kotlin offers
  - ✓ Today over 60 of Google's apps are built using Kotlin
  - ✓ includes apps like Maps, Home, Play, Drive, and Messages
- One example of success comes from the Google Home team
  - ✓ where migrating new feature development to Kotlin resulted in
  - ✓ a 33% reduction in codebase size
  - ✓ a 30% reduction in the number of NPE crashes



TechYourChance



# Thank You!

In our next session:



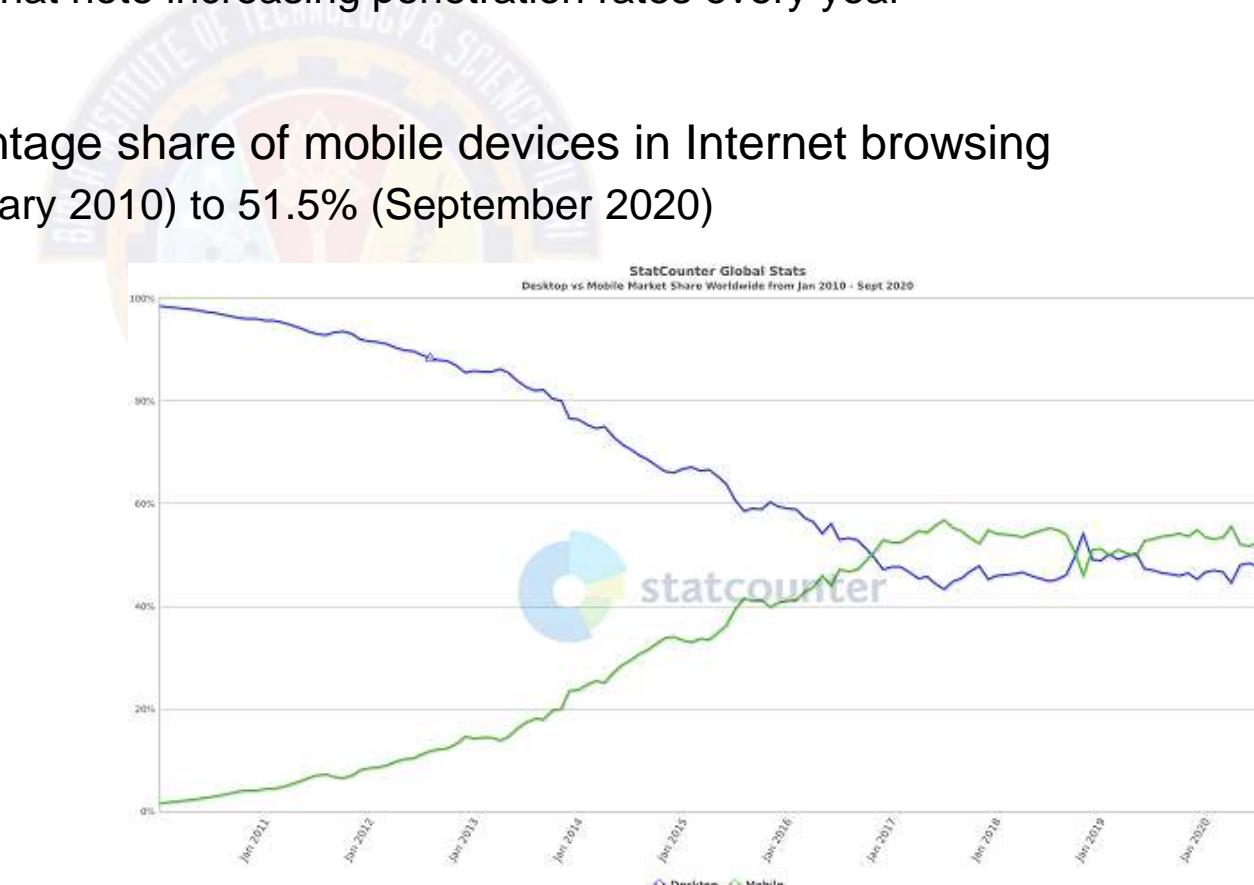
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# React Native

Chandan Ravandur N

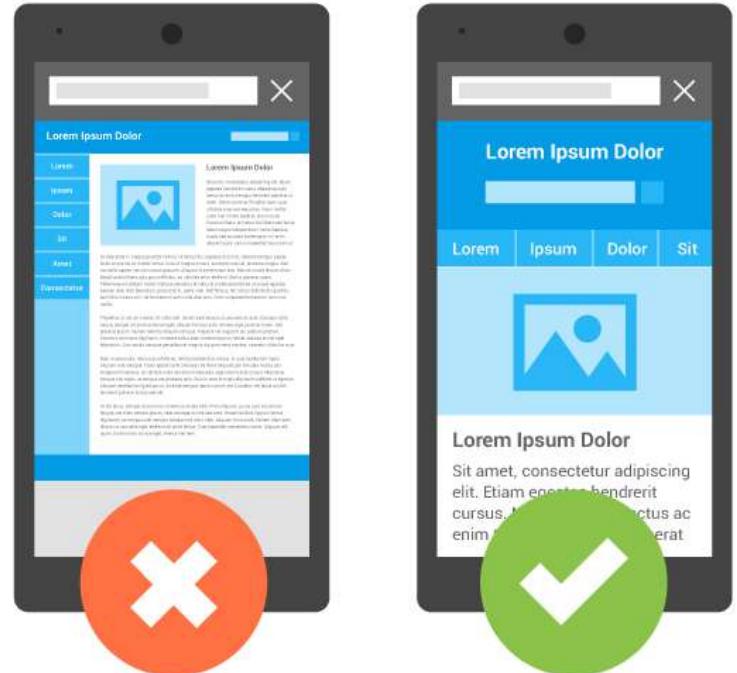
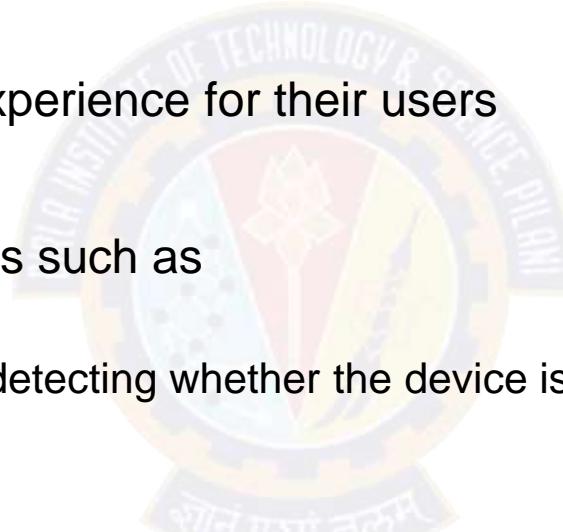
# Rise of Mobile Apps

- More and more people simply can't imagine their lives without mobile devices
  - ✓ such as smartphones, tablets, and wearables anymore
  - ✓ especially true for smartphones that note increasing penetration rates every year
- Over the last 10 years, the percentage share of mobile devices in Internet browsing
  - ✓ has increased from 1.56% (January 2010) to 51.5% (September 2020)



# Web for Mobile

- Many website owners have decided to create two versions of their sites –
  - ✓ one for wide-screen personal computers
  - ✓ other for narrow-screen mobile devices
- doing this to ensure the best possible experience for their users
- Turned out to be problematic, with issues such as
  - ✓ maintaining two applications
  - ✓ problems with SEO to the challenge of detecting whether the device is a smartphone or a PC
- A better solution: a responsive web design that adapts to the device on which it's displayed
  - ✓ only need one code to operate both platforms for a website to look good on any screen



Google Developers

# Issue with mobile app development process

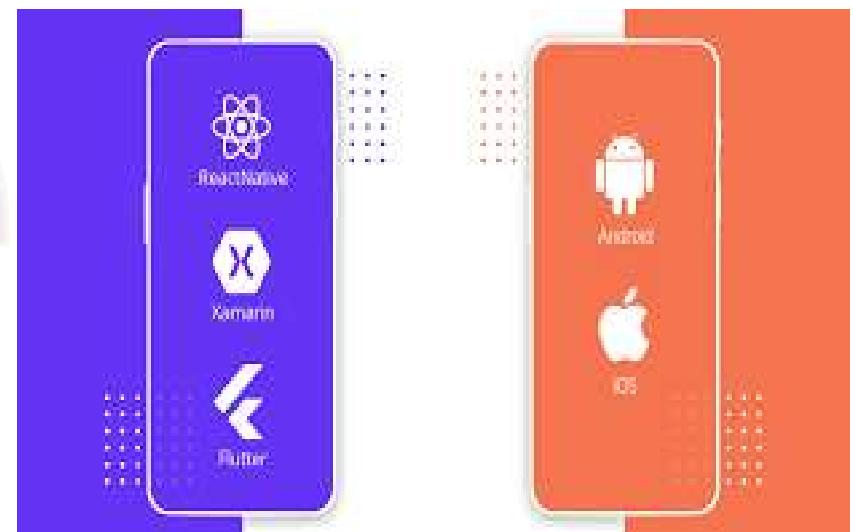
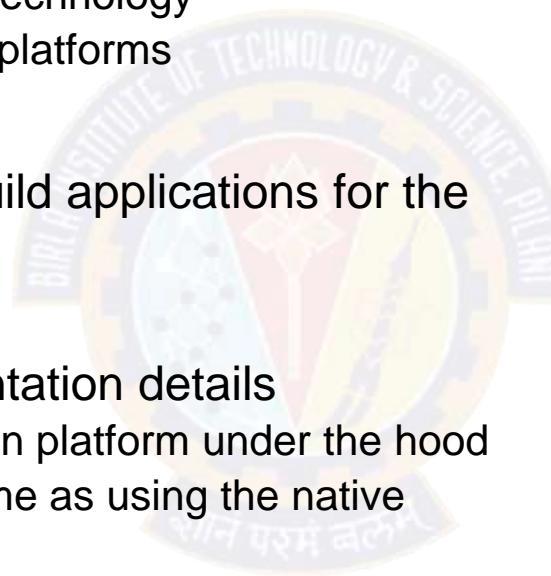
- A similar problem occurs in the mobile app development process
  - ✓ Many apps are created based on a non-standard look
  - ✓ compatible with the company's brand book and other customer requirements
  - ✓ while maintaining the same design across platforms such as iOS and Android
- Developers of individual platforms duplicate their work by writing the same functionalities
  - ✓ login, registration, adding a product to the basket, etc. in the language native to a given platform
  - ✓ has a significant impact on the cost of the developed application
- Many companies and developers have tried to solve this problem over the years
  - ✓ One of the solutions is writing cross-platform apps



medium

# What is a cross-platform native application?

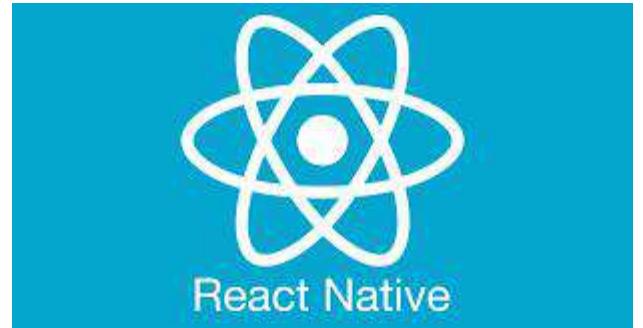
- The latest solution is cross-platform technologies that allow code to be written in one language
  - ✓ using components provided by a given technology
  - ✓ then building an application for multiple platforms
- Write one code which can be used to build applications for the platforms interested in
- Particular frameworks differ in implementation details
  - ✓ but all of them use code native to a given platform under the hood
  - ✓ The UX they deliver is similar or the same as using the native language for a given platform
- The most popular technologies
  - ✓ React Native
  - ✓ Flutter
  - ✓ Xamarin



TechAhead

# What is React Native?

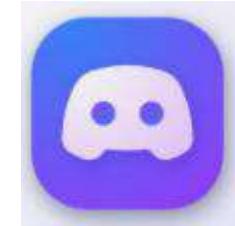
- React Native is a framework built by Facebook engineers in 2015
  - ✓ based on ReactJS
  - ✓ that helps to create cross-platform applications for Android and iOS platforms
- Allowing
  - ✓ to write the code just once using the popular JavaScript (TypeScript) language
  - ✓ to create a mobile application for both Android and iOS
  - ✓ to accelerate the process of creating mobile applications and reduce the costs of their production
- Facebook uses React Native in many of its applications
  - ✓ Facebook mobile application (for example, the Marketplace functionality is made using React Native)
  - ✓ Instagram
- By writing one code in Javascript (according to the StackOverflow survey, currently the most popular programming language)
  - ✓ one can create a native application for Android and iOS systems
  - ✓ get the same file as if we have used the native language of the platform



# Powered by React Native

- In addition to Instagram and the Marketplace functionality in the Facebook mobile application, React Native is also used in applications such as:

- ✓ Uber Eats
- ✓ Discord (iOS version),
- ✓ Walmart
- ✓ Wix
- ✓ Pinaster
- ✓ and Bloomberg

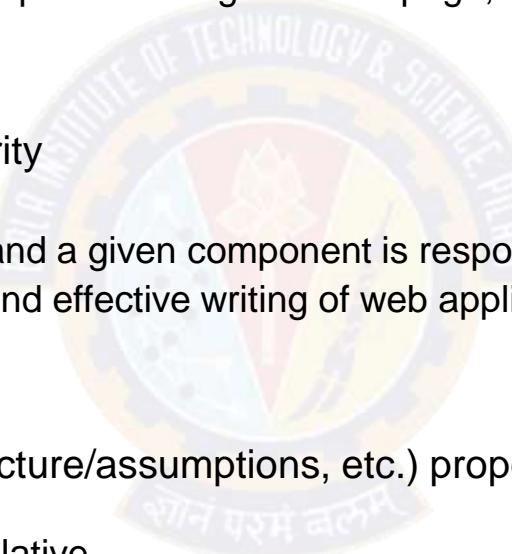


- Recently, Microsoft has also shown interest in this technology
  - ✓ its developers have released the React Native XP library
  - ✓ allows to create applications for iOS, Android, Web, and Windows 10 – UWP



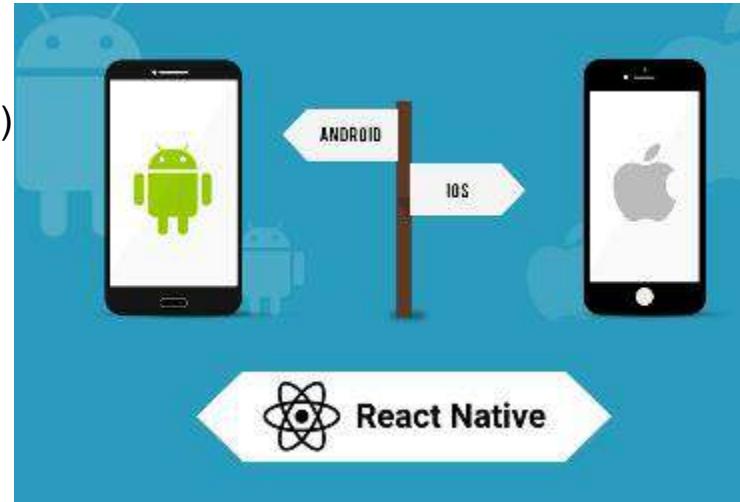
# What's the difference between React (ReactJS) and React Native?

- Published by Facebook in 2013, React is a library designed for web applications
  - ✓ main framework used to create Facebook web applications
  - ✓ main role of ReactJS is to display individual components/widgets of the page, depending on the current configuration (provided data)
- ReactJS has gained both recognition and popularity
  - ✓ due its simplicity and declarative nature
  - ✓ developer declares what they want to achieve and a given component is responsible for this effect
  - ✓ one of the 3 most popular libraries for the fast and effective writing of web applications, next to Angular and Vue
- The way of thinking about the application (architecture/assumptions, etc.) proposed by ReactJS is so universal
  - ✓ it was also used in its younger sibling – React Native
- React Native uses the mechanisms of ReactJS in line with the “Learn once, write everywhere” principle
  - ✓ main difference between the siblings is the type of application under construction
  - ✓ ReactJS focuses on web applications, so uses HTML, CSS, and, of course, JavaScript there
  - ✓ React Native helps in building an application for multiple platforms at the same time



# The greatest advantages of React Native

- The framework is based on one of the most popular languages at the moment – JavaScript (source)
  - ✓ developers have access to many external libraries that work in React Native applications
- Can be integrated with an already existing mobile application
- At the stage of designing a mobile application, can distinguish between
  - ✓ views/functionalities that are worth writing in React Native (e.g. user profile, login)
  - ✓ ones for which speed is critical, should therefore be implemented using native code.
- “Learn once, write everywhere” makes it easy for anyone who knows ReactJS to get into a React Native project very quickly
  - ✓ allows to write a code fragment for just a given platform
- The ability to create own native code that can communicate with JavaScript code
  - ✓ helpful for communication with the API of a platform that is not supported or when we want to integrate with a ready-made solution
- Architecture is focused on adding new platforms
  - ✓ supports platforms such as Windows, macOS and offers the possibility to write web applications
- The Expo framework is worth a mention as well
  - ✓ significantly improves work with React Native by providing many useful components for creating a typical mobile application
  - ✓ for example, access to text messages, contacts, cameras, or login via social media



Blue Whale Apps

Reference

Adapted from “What is React Native and When to Use it? Introduction for App Owners”



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# How React Native works?

Chandan Ravnadur N

# How does React Native work?

- React Native's main task is to provide components
  - ✓ which are then translated (mapped) to their native equivalents on a given platform
  - ✓ This is how it maintains the UX of a given platform
- For this reason, the framework has its own special components (e.g., Button, Modal, TextInput) from which we build the UI
  - ✓ React Native knows how to build an application for a specific platform.
- For example, when creating an Android application,
  - ✓ React Native translates its Button component
  - ✓ into a native button available in Android

```
1 <Button
2   onPress={() => console.log('Hello Reader')}
3   title="Button with native styling"
4 />
```



macOS (React Native for Windows + macOS)

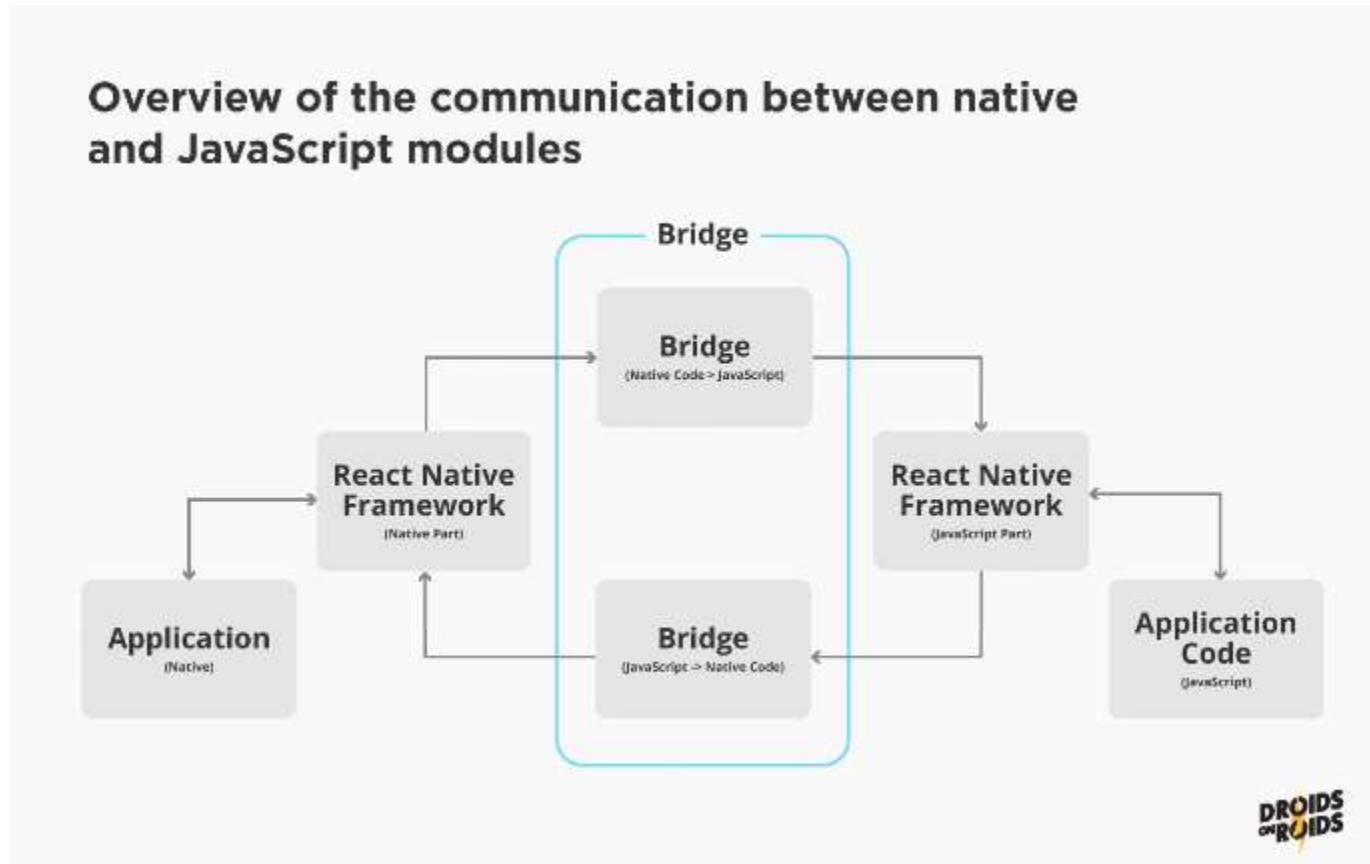


Android (React Native)



iOS (React Native)

# How does React Native work?(2)



# How does React Native work?(3)

- Important thing is communication between
  - ✓ the JavaScript world (where application code exists)
  - ✓ the native environment of the platform in which the application is displayed
- For example,
- the code in React Native needs to know when the user has pressed a button, typed text into a text field, or switched to another view to react to it accordingly
  - ✓ task of the built-in mechanism in React Native Bridge is to ensure communication understandable for both environments
- the downside of this solution is the additional time associated with communication between platforms during the application's operation
  - ✓ must be taken into account – for instance, when creating efficient animations in React Native

# React Native – on Android and iOS

## differences in development for Android and iOS

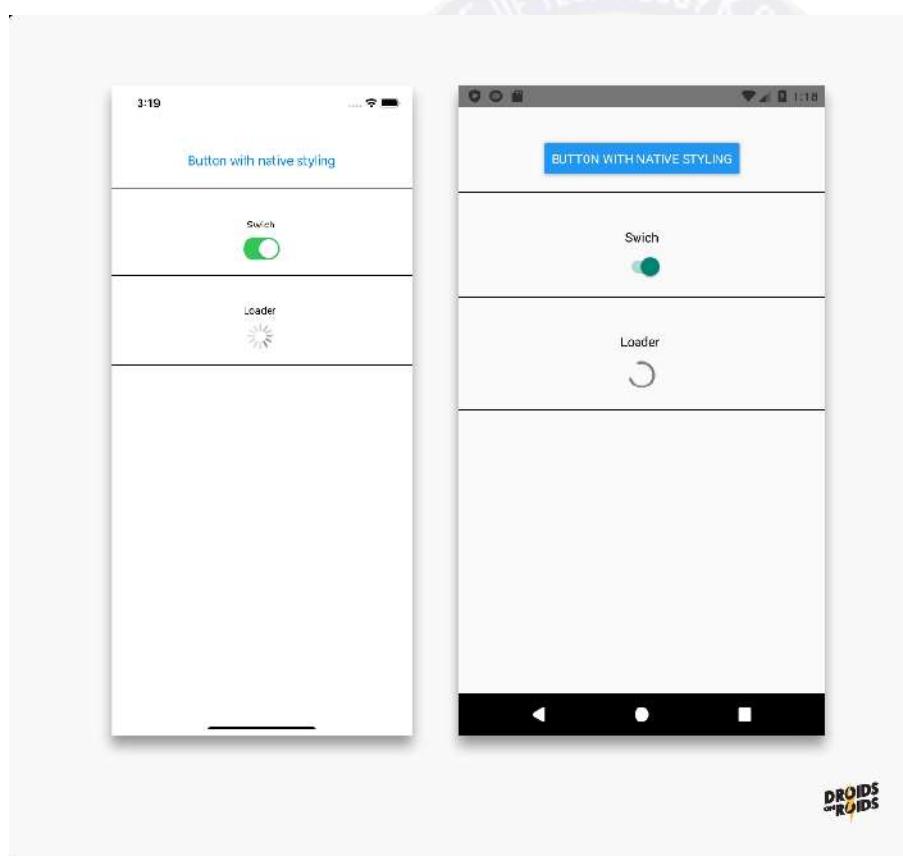
- Will React Native make app look and run the same way on iOS and Android?
- Need to bother about
  - ✓ Look of application
  - ✓ Developer's experience
  - ✓ Size of application



# React Native – on Android and iOS(2)

## Look of application

- By default, when run the React Native application on different platforms
  - ✓ will see a different-looking application
  - ✓ because it works the way it works, React Native will use native styling for each platform



# React Native – on Android and iOS(2)

## Look of application

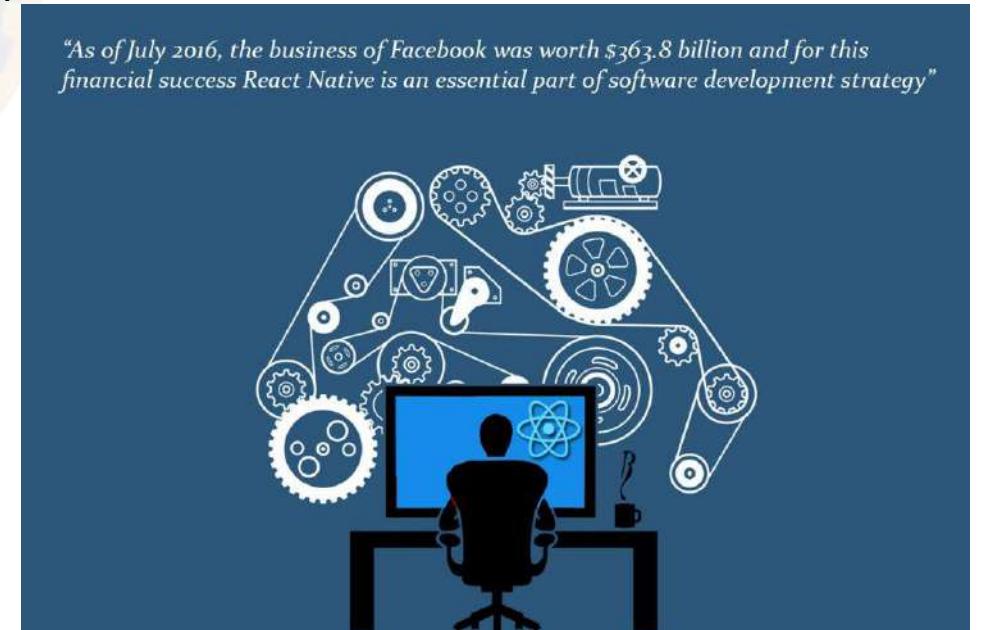
- Nothing stands in the way of making the appearance of individual components consistent so that they look the same, regardless of the platform
- The React Native community also provides solutions in the form of UI frameworks
  - ✓ ready-made components styled in accordance with a given design (for example, material design)
  - ✓ also have completely new components ready to be used in application (like a floating action button, chip, and avatar)
  - ✓ use of ready-made UI frameworks significantly accelerates the development process
- The most popular of these are:
  - ✓ React Native Elements
  - ✓ NativeBase
  - ✓ UI Kitten
  - ✓ Shoutem UI toolkit
  - ✓ React Native Paper
- React Native gives complete freedom in how application looks
  - ✓ can resemble the native look and also be consistent on individual platforms

# React Native – on Android and iOS(3)

## Developer's experience

- From the programmer's perspective
  - ✓ Due to the introduction of abstractions for the application code, it's often irrelevant on what platform it's executed
- React Native provides tools that allow to judge what platform it's running on
  - ✓ so that one can create a styling/behavior that resembles the native one more closely
- In extreme cases, when a given component looks completely different
  - ✓ a developer can easily create a component for a specific platform

*"As of July 2016, the business of Facebook was worth \$363.8 billion and for this financial success React Native is an essential part of software development strategy"*



# React Native – on Android and iOS(4)

## Size of application

- Since the Android system doesn't provide the mechanism needed to run JavaScript code (JavaScript Virtual Machine) by default
  - ✓ React Native must add it when building an application for this platform
  - ✓ makes the application size larger than the same application built for iOS



# Does React Native mean we don't need native developers?

- In typical use cases for mobile applications, a native developer isn't needed in the development process
  - ✓ A React Native developer works with JavaScript code in their everyday work
- The stages of creating an application such as configuration for release, configuring CI/CD, and the release process are just like in the case of native applications
  - ✓ After building a cross-platform application, we get the same file as in native development
  - ✓ React Native developers need to use tools and services that support mobile applications
  - ✓ Some of these tools might be a complete novelty, especially for those who previously worked only with web applications
- Due to the high popularity of React Native, more and more tools support this framework (such as Bitrise or App Center)
  - ✓ so the presence of a native developer for these steps is also not necessary
  - ✓ although it can certainly speed them up
- Experience in native application development will also be useful
  - ✓ when integrating with native modules - create code on the native side, which can communicate with JavaScript code
  - ✓ when use React Native only in specific areas of the mobile application

Reference

Adapted from “What is React Native and When to Use it? Introduction for App Owners”



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Pros and Cons of React Native Development

Chandan Ravandur Ns

# Pros of React Native

## App development process and cost

- Speeds up the app development process
  - ✓ ability to create applications for two platforms using one code
  - ✓ for a product released on both platforms, this can save you time and money by up to 30%
- Due to the community support and React Native's extensibility, the list of supported platforms is growing
  - ✓ just need to install additional libraries
- Lower cost of app development
  - ✓ The common source code allows to build the app with one development team familiar with Javascript.
  - ✓ significantly contributes to the reduction of app development cost
  - ✓ can involve a smaller number of people in the project - no need to “duplicate” teams per platform

# Pros of React Native (2)

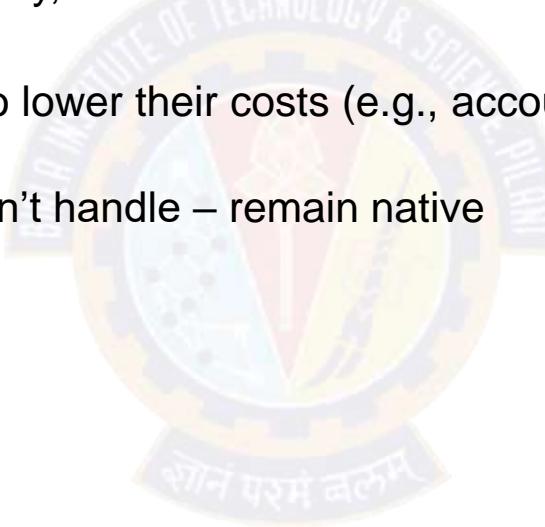
## Ecosystem and Growth

- A rich ecosystem
  - ✓ has access to a wide range of libraries and tools that facilitate developers' work and accelerate the software development process
    - ❖ the library repository for Javascript npm returns 29,352 results after typing the phrase "react-native,"
    - ❖ the corresponding package search engine for Flutter returns only 12,900 results for libraries
  - ✓ means that the choice of 3rd party libraries for Flutter is 44% smaller than React Native
  - ✓ shows the community's commitment to developing tools for the React Native framework
  - ✓ ReactJS is one of the most popular web development frameworks
  - ✓ Thanks to a common mechanism, many libraries for ReactJs also work for React Native
- Ensures stable growth of the app
  - ✓ RN shares its set of components that "know" how to display the app on a given platform
  - ✓ this abstraction, React Native allows the developer to focus on developing the application without going into the details of the platform
  - ✓ the process of building app features is more stable and resistant to discrepancies between platforms than in separate native teams
  - ✓ allows for better planning of application releases with new features/upgrades

# Pros of React Native (3)

## Integration

- Can integrate with the native application
  - ✓ React Native has mechanisms that allow it to integrate with an already existing native application
  - ✓ can find both views that are written natively, as well as views written with React Native in a mobile application
  - ✓ Developers can write some views with React Native in order to lower their costs (e.g., account management, purchase summary, etc.)
  - ✓ the rest views – which the framework can't handle – remain native



# Pros of React Native (4)

## Supported by external tools

- The application development process isn't only writing code but also repetitive routine tasks
  - ✓ Building (compiling a new version of the application)
  - ✓ testing the application using automated tests
  - ✓ releasing the application to the store
  - ✓ as well as investigating and fixing bugs
- Such repetitive tasks are usually delegated to external tools or services (3rd party software)
  - ✓ which, once configured, perform them for developers who can instead focus on adding new functionalities
- The most popular tools supporting the process of developing mobile applications that also support React Native are, for example:
  - ✓ CircleCI, CodeMagic, Bitrise – a platform supporting the process of automatic testing, building, and releasing a new version of the application
  - ✓ AppCenter – a platform that comprehensively supports the development process – allows you to test, build, and release a new application
  - ✓ Sentry – a platform dedicated to collecting and displaying bug reports that provide the necessary information for its repair
  - ✓ Bugsnag – a platform that collects and displays bug reports in order to provide the necessary information for its repair
  - ✓ Amplify – a framework allowing for easier integration with AWS services, incl. authentication, data retrieval from API, saving/reading files, and notification service

# Cons of React Native

## It's not a native solution

- React Native is a cross-platform technology based on JavaScript
  - ✓ the authors of the framework had to create communication mechanisms between the native world and Javascript
  - ✓ an application written in React Native might be slower compared to native applications and take up more space than its native counterpart.
- The performance problem can only arise in applications with high dynamism
  - ✓ in which there are continuous/large changes on the screen, such as, for example, action games
- Facebook engineers have made every effort to ensure that applications written in React Native render at 60 frames per second
  - ✓ in accordance with the applicable standard, providing the user with a native experience

# Cons of React Native(2)

## Testing is more challenging

- Doesn't speed up the testing process
  - ✓ While the working time of development teams decreases
  - ✓ the same cannot be said about the work of the Quality Assurance team
  - ✓ time spent on testing is similar to testing in native development
- Use of cross-platform technology introduces two new potential error groups
- The first is related to how the application communicates with native components and bad code that causes errors on each platform
  - ✓ React Native translates JavaScript components into their native equivalents and provides the ability to freely communicate between them
  - ✓ should test a React Native app much more thoroughly
  - ✓ there may be bugs and edge cases in the components provided by the framework
- The second group of errors that occur only in cross-platform solutions is related to the common source code
  - ✓ If there's an error there, you can be sure that it will occur on every platform
  - ✓ For this reason, when finding an error, the tester should check whether that error occurs on both platforms
- In the case of native development, such propagation of errors between platforms doesn't occur

# Cons of React Native(3)

## It's harder to build a cross-platform team

- React Native is a cross-platform technology
- The team should also have knowledge of both the world of web technologies
  - ✓ React, JavaScript, and its ecosystem
  - ✓ native technologies (project configuration, CI, UX guidelines for the platform)
- Much more difficult to build a team able to cope with all the challenges that may arise in the development of a cross-platform application



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Mobile App Development: React Native vs Native (iOS, Android)

Chandan Ravandur N

# Going React Native Way

- React Native is a framework created by Facebook
  - ✓ that allows to develop native mobile apps for iOS and Android with a single JavaScript codebase
- With the rise of React Native popularity and the growing number of popular mobile apps
  - ✓ such as Facebook, Instagram, Pinterest, Uber, Discord, SoundCloud, Skype...
  - ✓ being partially or completely rewritten in React Native
- The question arises:
  - ✓ Should mobile developers use React Native for mobile development instead of going full native with Java or Swift?
- Lets compare based on
  - ✓ Codebase
  - ✓ The programming language
  - ✓ Hot reloading and Live reloading
  - ✓ User Interface
  - ✓ Native modules



# Codebase

- React Native enables a single JavaScript codebase for 2 different platforms
  - ✓ easier to maintain the app by having the same development process for both platforms and reusing the same code
  - ✓ requires less resources, because there is no need for separate iOS and Android teams
- Benefit comes with a cost
  - ✓ Well-known that Android and iOS have different design guidelines
  - ✓ Human Interface Guideline for iOS and Material Design for Android have a big share of differences
- If the project requirements dictate that these specific OS requirements should be followed for each native platform
  - ✓ React Native developer will need to write platform-specific code
  - ✓ which defeats the purpose of the single codebase
- Might be a bigger issue when it comes to iOS, because Apple often updates and deprecates their technologies
  - ✓ Android apps generally have more control of the system and are allowed more freedom
- Even with the factor of being allowed to write an app that looks the same on both platforms
  - ✓ still impossible to write an app without writing any platform-specific code
  - ✓ some components will just look great on iOS and bad on Android and vice versa
  - ✓ which raises the need for fine-tuning those specific cases

# The programming language

- React Native is based on React.js, which is written in JavaScript
  - ✓ major advantage is fact that JavaScript is an extensive and popular language
  - ✓ very hard to find a programmer who hasn't used it at some point in their career
- What is the disadvantage ... it's JavaScript
  - ✓ Java and Swift/Objective-C are strongly-typed, compiled languages
  - ✓ JavaScript is interpreted and often called an untyped language
- Means that variables can be anything at any time and a compiler is not going to help
  - ✓ if not extra careful while writing apps, the floor is paved for Javascript horrors to ensue
- When true native apps have more control of the variables and the logic of the app is more predictable,
  - ✓ JavaScript relies on the programmer's experience, Lint tools and automatic tests
- Still, it is a fact that JavaScript remains one of the most widely-used programming languages in the world
  - ✓ maturity of the JavaScript community implies that finding a React Native developer for a project should typically be trivial

# Hot reloading and Live reloading

- Developers worked with XCode or Eclipse / Android Studio knows how long and tedious build time can be
  - ✓ can especially be frustrating if are working on a feature which is multiple screens away from the launch screen
  - ✓ makes even trivial tasks such as changing a view color or changing a label text very time consuming
- React Native's solution for this problem is called hot reloading
  - ✓ keeps the app running and saves the state (the data and the screen you're on)
  - ✓ only injecting the changes that are being made in the code
  - ✓ saving a lot of precious development time
- App build time is significantly faster using React Native than on native iOS and Android
  - ✓ live reloading mechanism can be used to automatically reload the app every time the code is changed
  - ✓ without having to build it manually every single time

# User Interface

- React Native's approach to structuring UI is Flexbox
  - ✓ already very popular in web development
  - ✓ enables the developer to create a responsive web or mobile UI very easily
- A technology which rivals Android's XML / Constraint layout approach, or iOS's Storyboard / XIB / Coding with UI libraries such as Neon approach
- One considerable advantage that native mobile has though is that have access to all the native APIs
  - ✓ such as Camera, Touch ID and GPS, as well as tools for creating animations and complex user interface
  - ✓ no middle layer and are free to take advantage of everything the mobile platform has to offer
- React Native does not excel at creating complex UI and animations
- Also, some complex mobile features are simply best done in native
  - ✓ One example of this is Apple's ARKit for creating augmented reality experiences
  - ✓ Another example are apps which are very platform specific, such as apps for watchOS or tvOS

# Native modules

- If the team uses a cloud service for Continuous Integration such as Azure, even more native linking problems might arise
  - ✓ XCode and Android Studio on the cloud might have a different version and configuration than local one
  - ✓ so even if everything works well on local machine, it might not build remotely
  - ✓ the developer will need to do more fine-tuning in order for the linking to work on both sides
- The same goes for publishing the app
- If the team doesn't have developers that are experienced in native iOS and Android, they may struggle to release the app
  - ✓ More of an iOS problem, with Android release workflow being more straightforward
- In order to release an iOS app, React Native developer will still need to use XCode
  - ✓ in order to configure release-related data such as provisioning profiles and certificates, which can be overwhelming at first
- React Native is much more effective if used by developers who already have experience with native mobile development
  - ✓ because it doesn't completely remove the need of going native

# Conclusion

## In the end, is React native worth it?

- The answer is
  - ✓ It depends on your project.
- Go native when
  - ✓ need to make an iOS-only or Android-only app?
  - ✓ need to make a highly complex app which utilizes a large portion of platform-specific code
  - ✓ plan to maintain the app over a long period of time, without fear of Facebook quitting React Native
  - ✓ app need to support new mobile OS features as soon as they are released
- Go React native when
  - ✓ have a small team with limited time and resources, and need to make an app for both platforms
  - ✓ want to take advantage of fast build time, and features such as hot reloading and live reloading
  - ✓ developers have a strong React / Web development background
  - ✓ app going to look and behave the same on both platforms

Reference:

**Mobile App Development: React Native vs Native (iOS, Android)**

Dino Trnka



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

**Flutter**

**Chandan Ravandur N**

# Flutter

- Flutter is an open-source UI software development kit created by Google
  - ✓ used to develop applications for Android, iOS, Linux, Mac, Windows, Google Fuchsia and the web from a single codebase
- The first version of Flutter was known as codename "Sky"
  - ✓ ran on the Android operating system
  - ✓ was unveiled at the 2015 Dart developer summit, with the stated intent of being able to render consistently at 120 frames per second
- On December 4, 2018, Flutter 1.0 was released at the Flutter Live event, denoting the first "stable" version of the Framework
  - ✓ On December 11, 2019, Flutter 1.12 was released at the Flutter Interactive event
  - ✓ On May 6, 2020, the Dart SDK in version 2.8 and the Flutter in version 1.17.0 were released
    - ❖ where support was added to the Metal API, improving performance on iOS devices (approximately 50%), new Material widgets, and new network tracking
- On March 3, 2021, Google released Flutter 2 during an online Flutter Engage event
  - ✓ major update brought official support for web-based applications as well as early-access desktop application support for Windows, MacOS, and Linux



# What is Flutter?

- “Flutter is Google’s UI toolkit for building beautiful, natively compiled applications for mobile, web, and desktop from a single codebase.”
  - Google, [flutter.dev](https://flutter.dev)
- Flutter is a free and open-source mobile UI framework created by Google
  - ✓ allows to create a native mobile application with only one codebase
  - ✓ can use one programming language and one codebase to create two different apps (for iOS and Android)
- Flutter consists of two important parts:
- An SDK (Software Development Kit)
  - ✓ A collection of tools that are going to help develop applications
  - ✓ includes tools to compile code into native machine code (code for iOS and Android)
- A Framework (UI Library based on widgets)
  - ✓ A collection of reusable UI elements (buttons, text inputs, sliders, and so on) that can personalize for app needs
  - ✓ use a programming language called Dart - created by Google in October 2011, but it has improved a lot over these past years
- Dart focuses on front-end development, and can be used to create mobile and web applications



Google



# Who's using Flutter?

Google

GROUPON

Alibaba Group  
阿里巴巴集团

CapitalOne

Tencent 腾讯

Square

eBay



DREAM11

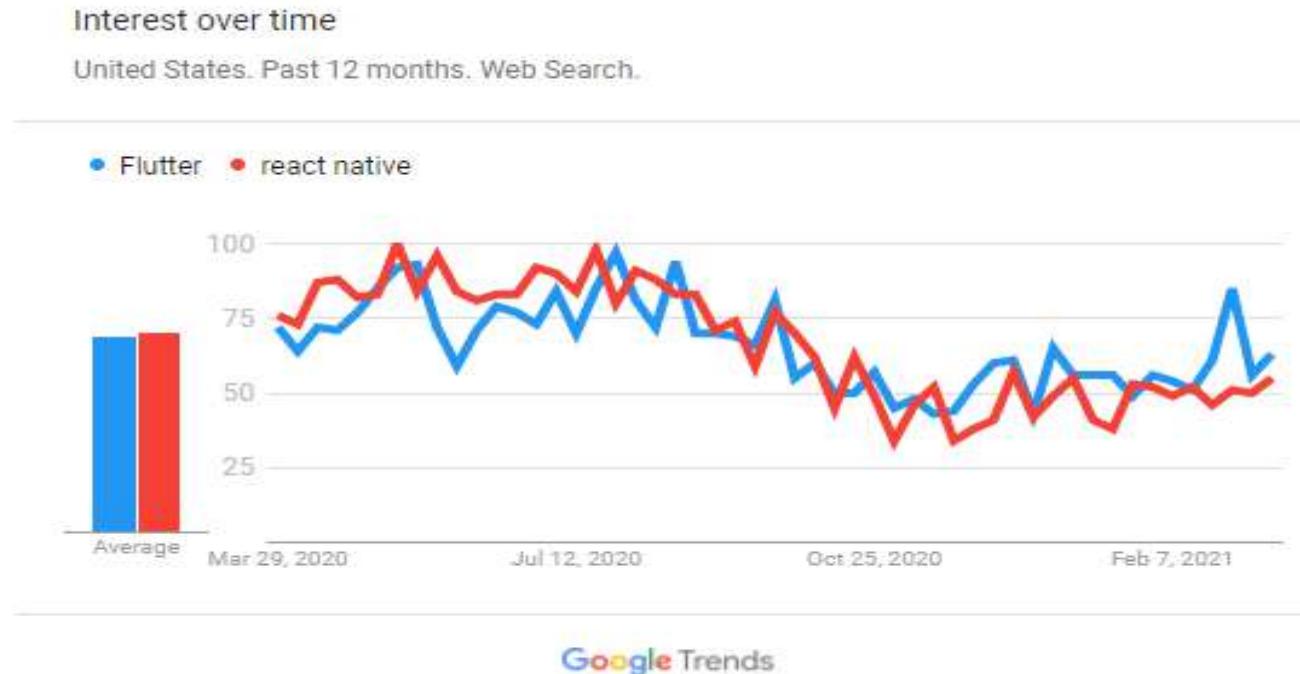
SONOS

BY bank

EMAAR

Google

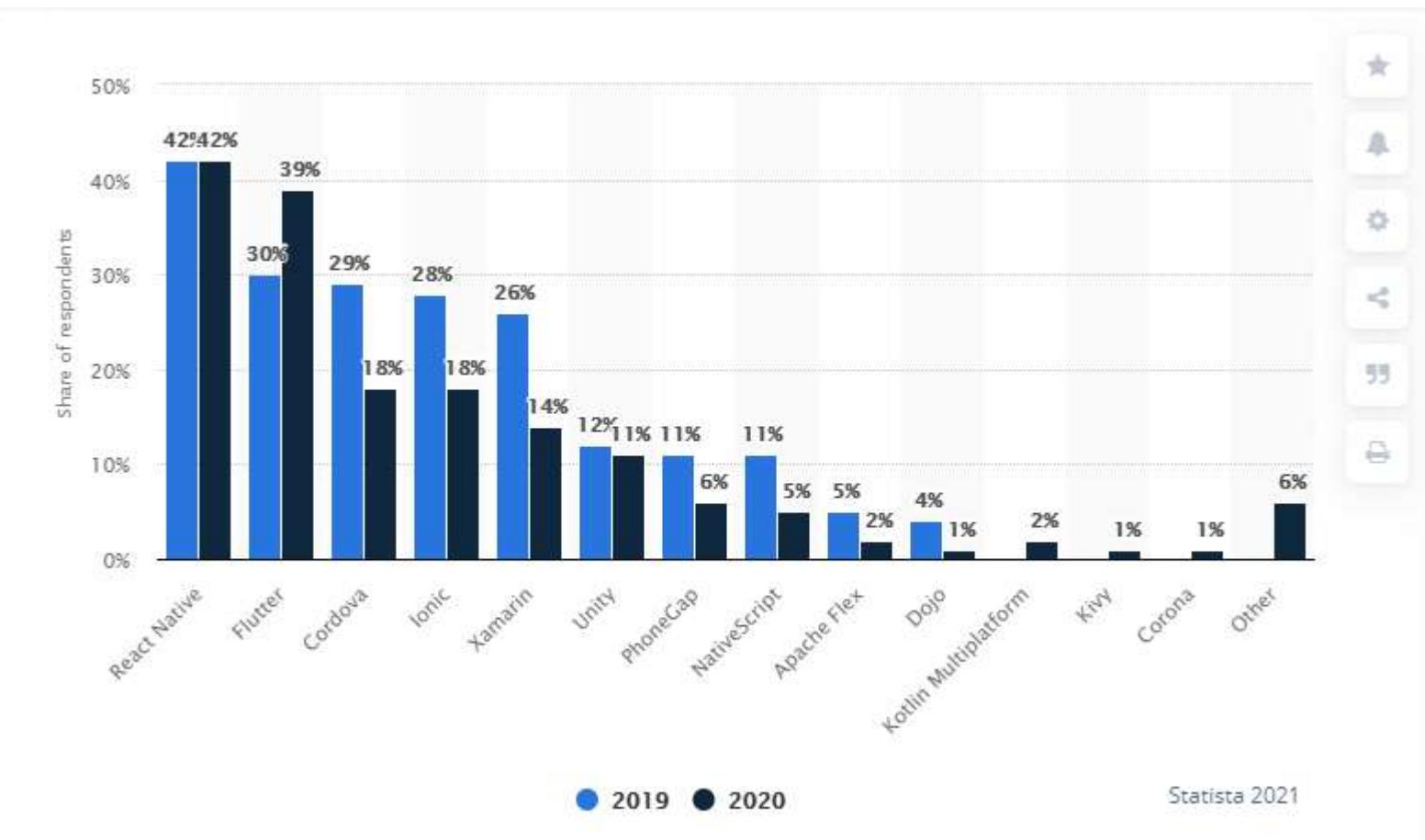
# Flutter – Interest over time



*Interest in Flutter vs React Native in 2020. Source: Google Trends*

# Flutter – Developer Trend

Cross-platform mobile frameworks used by developers worldwide 2019 and 2020





# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

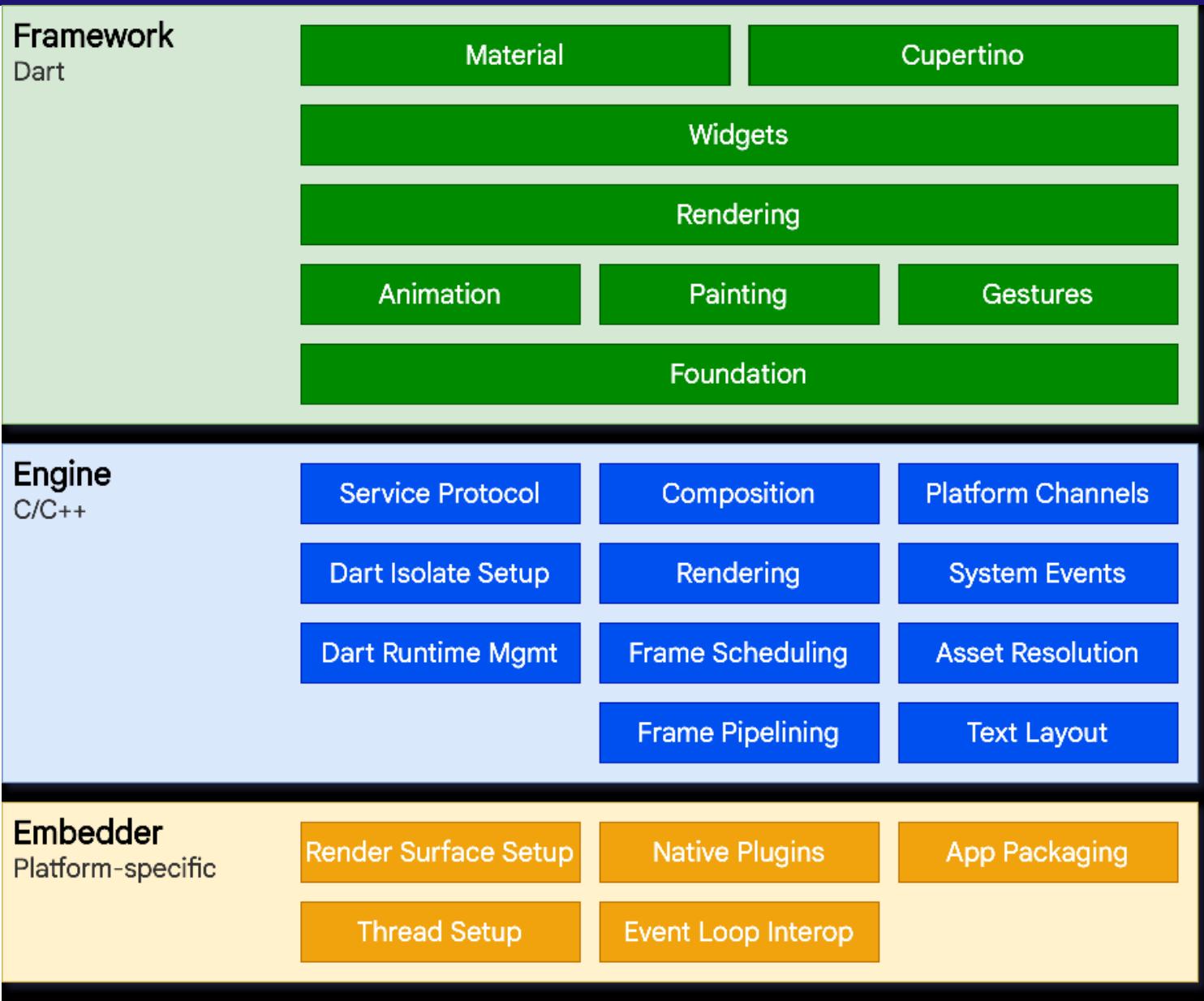
# How Flutter Works?

Chandan Ravandru N

# Flutter architectural overview

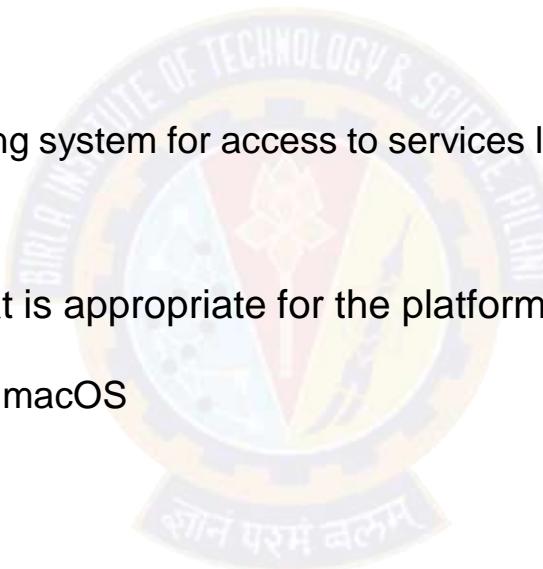
- The goal is to enable developers to deliver high-performance apps that feel natural on different platforms
  - ✓ embracing differences where they exist while sharing as much code as possible
- During development, Flutter apps run in a VM that offers stateful hot reload of changes without needing a full recompile
- For release, Flutter apps are compiled directly to machine code, whether Intel x64 or ARM instructions, or to JavaScript if targeting the web
- The framework is open source, with a permissive BSD license
  - ✓ has a thriving ecosystem of third-party packages that supplement the core library functionality

# Architectural layers



# Embedder

- To the underlying operating system, Flutter applications are packaged in the same way as any other native application
- A platform-specific embedder
  - ✓ provides an entrypoint
  - ✓ coordinates with the underlying operating system for access to services like rendering surfaces, accessibility, and input
  - ✓ manages the message event loop
- The embedder is written in a language that is appropriate for the platform
  - ✓ currently Java and C++ for Android
  - ✓ Objective-C/Objective-C++ for iOS and macOS
  - ✓ C++ for Windows and Linux
- Using the embedder, Flutter
  - ✓ code can be integrated into an existing application as a module,
  - ✓ the code may be the entire content of the application
- Flutter includes a number of embedders for common target platforms, but other embedders also exist



# Flutter engine

- At the core of Flutter is the Flutter engine
  - ✓ mostly written in C++
  - ✓ supports the primitives necessary to support all Flutter applications
- The engine
  - ✓ is responsible for rasterizing composited scenes whenever a new frame needs to be painted
  - ✓ provides the low-level implementation of Flutter's core API
  - ✓ including graphics (through Skia), text layout, file and network I/O, accessibility support, plugin architecture, and a Dart runtime and compile toolchain
- The engine is exposed to the Flutter framework through `dart:ui`
  - ✓ which wraps the underlying C++ code in Dart classes
  - ✓ exposes the lowest-level primitives, such as classes for driving input, graphics, and text rendering subsystems

# Flutter framework

- Typically, developers interact with Flutter through the Flutter framework
  - ✓ provides a modern, reactive framework written in the Dart language
  - ✓ includes a rich set of platform, layout, and foundational libraries, composed of a series of layers
- Working from the bottom to the top:
- Basic foundational classes,
  - ✓ Basic foundational classes, and building block services such as animation, painting, and gestures that offer commonly used abstractions over the underlying foundation
- The rendering layer provides an abstraction for dealing with layout
  - ✓ can build a tree of renderable objects
  - ✓ can manipulate these objects dynamically, with the tree automatically updating the layout to reflect changes
- The widgets layer
  - ✓ is a composition abstraction
  - ✓ Each render object in the rendering layer has a corresponding class in the widgets layer
  - ✓ allows to define combinations of classes that you can reuse
  - ✓ is the layer at which the reactive programming model is introduced
- The Material and Cupertino libraries
  - ✓ offer comprehensive sets of controls that use the widget layer's composition primitives to implement the Material or iOS design languages

# Reactive user interfaces

- In most traditional UI frameworks, the user interface's initial state is described once
  - ✓ then separately updated by user code at runtime, in response to events
  - ✓ Challenge of this approach is that, as the application grows in complexity
  - ✓ the developer needs to be aware of how state changes cascade throughout the entire UI
- Flutter takes an alternative approach to this problem
  - ✓ by explicitly decoupling the user interface from its underlying state
  - ✓ With React-style APIs, only create the UI description
  - ✓ The framework takes care of using that one configuration to both create and/or update the user interface as appropriate
- Flutter is a reactive, pseudo-declarative UI framework
  - ✓ the developer provides a mapping from application state to interface state
  - ✓ the framework takes on the task of updating the interface at runtime when the application state changes
  - ✓ is inspired by work that came from Facebook for their own React framework

# Widgets

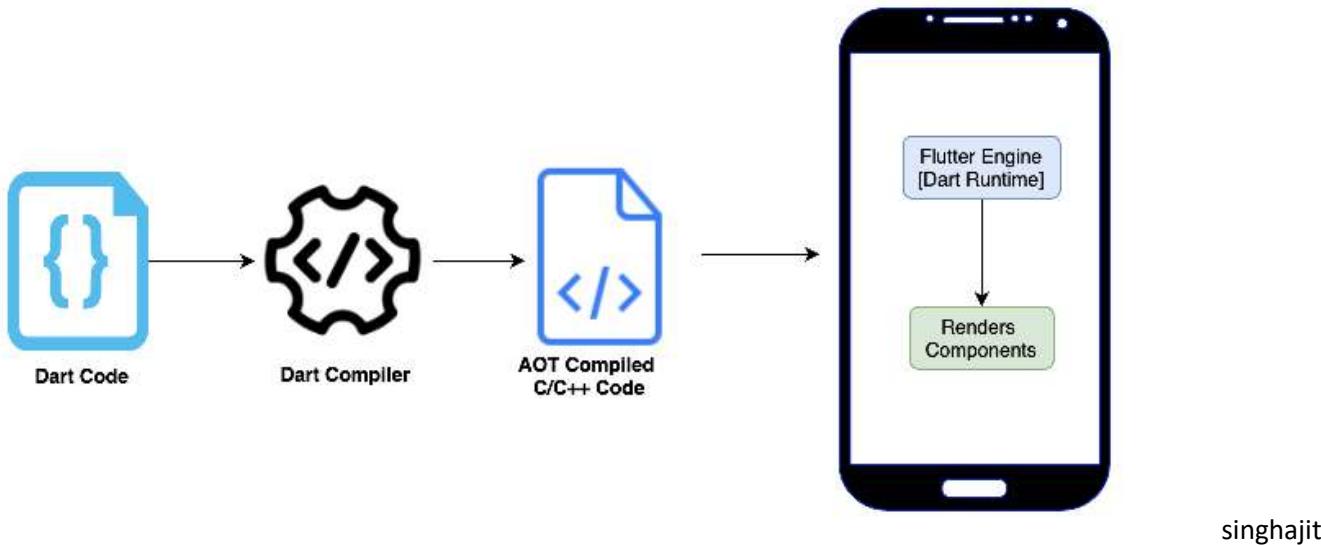
- Flutter emphasizes widgets as a unit of composition
  - ✓ Widgets are the building blocks of a Flutter app's user interface
  - ✓ Each widget is an immutable declaration of part of the user interface
  - ✓ Flutter has its own implementations of each UI control, rather than deferring to those provided by the system
- Widgets form a hierarchy based on composition
  - ✓ Each widget nests inside its parent and can receive context from the parent
  - ✓ This structure carries all the way up to the root widget
- Apps update their user interface in response to events by telling the framework to replace a widget in the hierarchy with another widget
  - ✓ The framework then compares the new and old widgets, and efficiently updates the user interface.

# How it works?

- Flutter doesn't use OEM widgets, but provides developers with its own ready-made widgets that look native to Android or iOS apps
  - ✓ following Material Design or Cupertino
  - ✓ Naturally, developers can create their own widgets as well
- Flutter also provides developers with reactive-style views
- Flutter uses Dart
  - ✓ to avoid performance issues deriving from using a compiled programming language to serve as the JavaScript bridge
  - ✓ compiles Dart ahead of time (AOT) into the native code for multiple platforms
- That way, Flutter can easily communicate with the platform without needing a JavaScript bridge
  - ✓ that involves a context switch between the JavaScript realm and the native realm
- Compiling to native code also boosts the app startup time
  - ✓ Today, Flutter is the only mobile SDK that offers reactive views without the need for a JavaScript bridge

# How Flutter code runs on device

A simplistic representation of how the dart code runs on device.



singhajit

- Steps:
  - ✓ First of all the Dart code is compiled into AOT (Ahead of time) native, ARM library code.
  - ✓ The compiled native code is embedded into apk or ipa file
  - ✓ When user opens the app, Flutter Embedder code kicks in
  - ✓ Flutter Embedder initializes the Flutter app
  - ✓ Dart runtime takes the native code and starts executing it

Reference:  
Flutter.dev



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Pros & Cons of Flutter Mobile Development

Pravin Y Pawar

# Benefits of Flutter

- Open-source
  - ✓ Flutter is an open-source technology surrounded by an active community of developers
  - ✓ who provide support, contribute to the tool's extensive documentation, and develop helpful resources
  - ✓ Both Dart and Flutter are free to use
- It saves time and money
  - ✓ Flutter is a cross-platform development tool
  - ✓ means software developers can use the same code base for building an iOS and Android app
  - ✓ best method for saving time and resources throughout the development process
- Excellent performance
  - ✓ Flutter offers outstanding performance for two reasons
  - ✓ First, it uses Dart, which compiles into native code
  - ✓ Second, Flutter has its own widgets, so there's no need to access OEM ones
  - ✓ As a result, there's less communication between the app and the platform
  - ✓ ensures fast app startup times and fewer performance issues in general

# Benefits of Flutter (2)

- Quick development thanks to hot reload
  - ✓ Flutter is gaining a lot of traction among mobile developers because of hot reload
  - ✓ Hot reload allows to instantly view the changes applied to the code on emulators, simulators, and hardware
  - ✓ The changed code is reloaded in less than a second\
  - ✓ All the while, the app is running and developers don't need to waste time on restarting it
  - ✓ That makes building UIs, adding new features, and fixing bugs easier
  - ✓ Even if forced to do a full app reload, can be sure that it's completed in no time, accelerating the development process
- Compatibility
  - ✓ Comes with its own widgets that result in fewer compatibility issues
  - ✓ Developers will see fewer problems on different OS versions and can spend less time on testing the app on older OS versions
    - ❖ can be confident that app will work on future OS versions.
- Once a new version of Android or iOS comes out, Flutter widgets will have to be updated
  - ✓ Google is a massive internal user of Flutter
  - ✓ the Flutter team is strongly motivated to keep their widget sets as current and close to the platform widgets as possible
  - ✓ Flutter widgets are customizable and can be updated by anyone

# Benefits of Flutter (3)

- A growing community
  - ✓ Flutter has a robust community, and it's only the beginning!
  - ✓ Flutter Awesome
    - ❖ An awesome list that curates the best Flutter libraries and tools
    - ❖ Publishes daily content with lots of examples, application templates, advice, and so on.
  - ✓ Awesome Flutter
    - ❖ A GitHub repository (linked to Flutter Awesome) with a list of articles, videos, components, utilities, and so on.
  - ✓ It's all widgets!
    - ❖ An open list of apps built with Flutter
  - ✓ Flutter Community
    - ❖ A Medium publication where you can find articles, tutorials, and much more
- Supported by Android Studio and VS Code
  - ✓ Flutter is available on different IDEs
  - ✓ The two main code editors for developing are Android Studio (IntelliJ) and VS Code.
  - ✓ Android Studio is a complete software with everything already integrated
    - ❖ have to download Flutter and Dart plugins to start
  - ✓ VS Code is a lightweight tool, and everything is configurable through plugins from the marketplace

# Cons of Flutter Mobile Development

- Large File Sizes
  - ✓ One big loophole that cannot be ignored is the large file size of apps developed in Flutter
  - ✓ File sizes could be a significant issue and cause a developer to choose an alternative tool for the development.
  - ✓ Many older devices are unable to store additional apps without users being forced to pick and choose between an app or photos/music on their device
  - ✓ it's not easy to understand the audience you are appealing to
- Lack of Third-party Libraries
  - ✓ Third-party libraries and packages have a significant impact on software development as it enables some features for developers
  - ✓ These third-party libraries are normally free, open-source, pre-tested, and easily available
  - ✓ Since Flutter is new for mobile app development, it's not easy to find such free packages and libraries
  - ✓ The tool itself is still in the growing phase and improving

# Cons of Flutter Mobile Development

- Look and Feel
  - ✓ The look & feel is not 100% the same as with native solutions as Flutter doesn't create native components
  - ✓ Replicates Android's Material Design and iOS-specific components with its Cupertino library, but it's not exactly the same
- Dart
  - ✓ Flutter is using a Dart programming language - has both benefits and drawbacks
  - ✓ This object-oriented programming language is not as great as other languages like C#, Java, Objective C, and JavaScript
  - ✓ Not many fresher's will be able to develop an app using this language
  - ✓ essential factor to keep in mind while developing a cross platform application
- Development guidelines
  - ✓ There are no single "guidelines" when it comes to developing Flutter apps
  - ✓ can be problematic when building more complex software.

Reference:

**Pros & Cons of Flutter Mobile Development by Wojciech Rozwadowski**

**What is Flutter and Why You Should Learn it in 2020 by Gaël Thomas**



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

**Xamarin**

**Chandan Ravandur N**

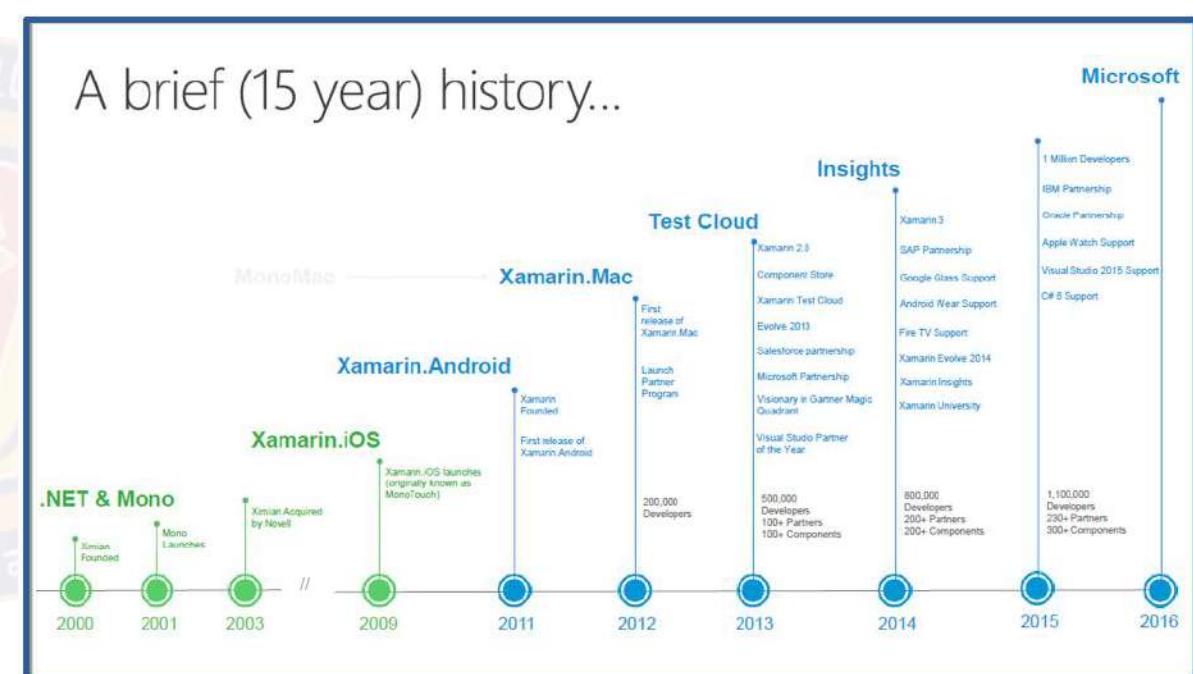
# Xamarin

- Xamarin is an open-source platform for building modern and performant applications for iOS, Android, and Windows with .NET
  - ✓ abstraction layer that manages communication of shared code with underlying platform code
  - ✓ runs in a managed environment that provides conveniences such as memory allocation and garbage collection
- Xamarin enables developers to share an average of 90% of their application across platforms
- allows developers to write all of their business logic in a single language (or reuse existing application code)
  - ✓ but achieve native performance, look, and feel on each platform
- Xamarin applications can be written on PC or Mac
  - ✓ compile into native application packages
  - ✓ such as an .apk file on Android, or an .ipa file on iOS



# History

- The platform was built by the developers behind Mono, an open source development platform based on the .NET Framework
  - ✓ led by Miguel de Icaza and first introduced in 2001
  - ✓ The Xamarin company was founded on May 16, 2011
- However, unlike its predecessor, Xamarin was created as a commercial project until the company was acquired by Microsoft in 2016
  - ✓ Xamarin became a popular cross-platform product for developing mobile apps within the Microsoft ecosystem
  - ✓ This acquisition broke the financial barrier for using Xamarin
  - ✓ As Microsoft made Xamarin SDK open-source, it became part of Xamarin Visual Studio Integrated Development Environment



# Who Xamarin is for

- Xamarin is for developers with the following goals:
- Share code, test and business logic across platforms
- Write cross-platform applications in C# with Visual Studio



# Features

- Complete binding for the underlying SDKs
  - ✓ Xamarin contains bindings for nearly the entire underlying platform SDKs in both iOS and Android
  - ✓ Additionally, these bindings are strongly-typed
  - ✓ which means that they're easy to navigate and use, and provide robust compile-time type checking during development
  - ✓ Strongly-typed bindings lead to fewer runtime errors and higher-quality applications
- Objective-C, Java, C, and C++ Interop
  - ✓ Xamarin provides facilities for directly invoking Objective-C, Java, C, and C++ libraries
  - ✓ giving the power to use a wide array of third party code
  - ✓ lets you use existing iOS and Android libraries written in Objective-C, Java, or C/C++
  - ✓ offers binding projects that allow to bind native Objective-C and Java libraries using a declarative syntax

# Features(2)

- Modern language constructs
  - ✓ Xamarin applications are written in C#, a modern language that includes significant improvements over Objective-C and Java
  - ✓ such as dynamic language features, functional constructs such as lambdas, LINQ, parallel programming, generics etc.
- Robust Base Class Library (BCL)
  - ✓ Xamarin applications use the .NET BCL, a large collection of classes that have comprehensive and streamlined features
  - ✓ such as powerful XML, Database, Serialization, IO, String, and Networking support etc.
  - ✓ Existing C# code can be compiled for use in an app
  - ✓ which provides access to thousands of libraries that add functionality beyond the BCL

# Features(3)

- Modern Integrated Development Environment (IDE)
  - ✓ Xamarin uses Visual Studio, a modern IDE that includes features such as
    - ❖ code auto completion
    - ❖ a sophisticated project and solution management system,
    - ❖ a comprehensive project template library
    - ❖ integrated source control etc.
- Mobile cross-platform support
  - ✓ Xamarin offers sophisticated cross-platform support for the three major platforms of iOS, Android, and Windows
  - ✓ Applications can be written to share up to 90% of their code
  - ✓ Xamarin.Essentials offers a unified API to access common resources across all three platforms
  - ✓ Shared code can significantly reduce both development costs and time to market for mobile developers



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# How Xamarin Works?

Chandan Ravandur N

# How Xamarin Works

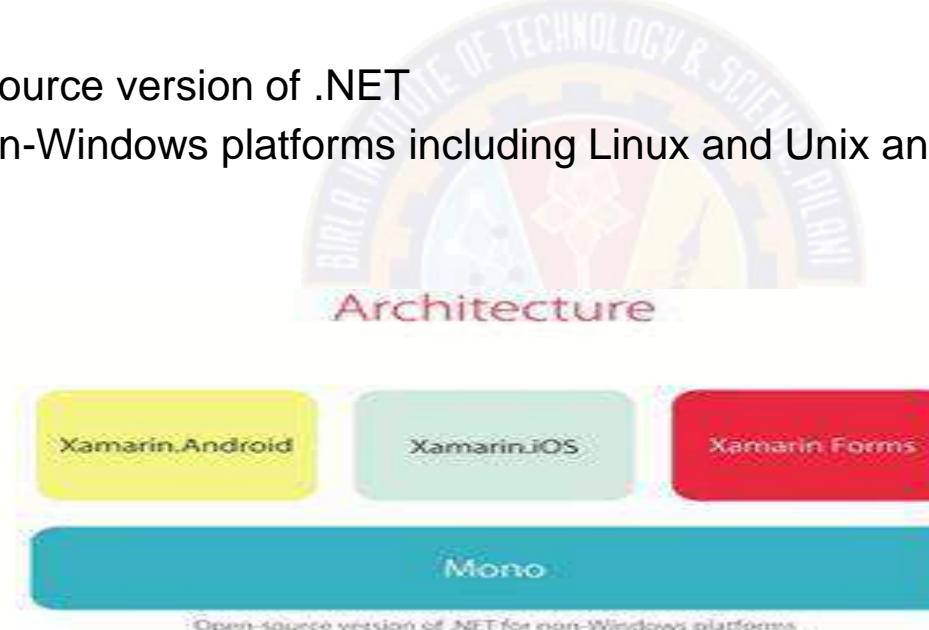


microsoft

- The diagram shows the overall architecture of a cross-platform Xamarin application
- Xamarin allows to create native UI on each platform and write business logic in C# that is shared across platforms
- In most cases, 90% of application code is sharable using Xamarin
- Xamarin is built on top of .NET, which automatically handles tasks such as
  - ✓ memory allocation
  - ✓ garbage collection
  - ✓ interoperability with underlying platforms

# Architecture of Xamarin

- The way Xamarin works is quite interesting
- All Xamarin programs, like Xamarin.Android, Xamarin.iOS, Xamarin.Forms are built on the top of MONO
  - ✓ which is an open source version of .NET
  - ✓ runs on various non-Windows platforms including Linux and Unix and their versions



# MONO

- Mono is a project that has been around for a while now but it wasn't very popular until recently
  - ✓ Initially, Mono had two products, Mono for Andriod and Mono Touch
  - ✓ Later, were rebranded as Xamarin.Android and Xamarin.iOS
  - ✓ Xamarin.Android is used for Android platform and Xamarin.iOS is used for iOS platform

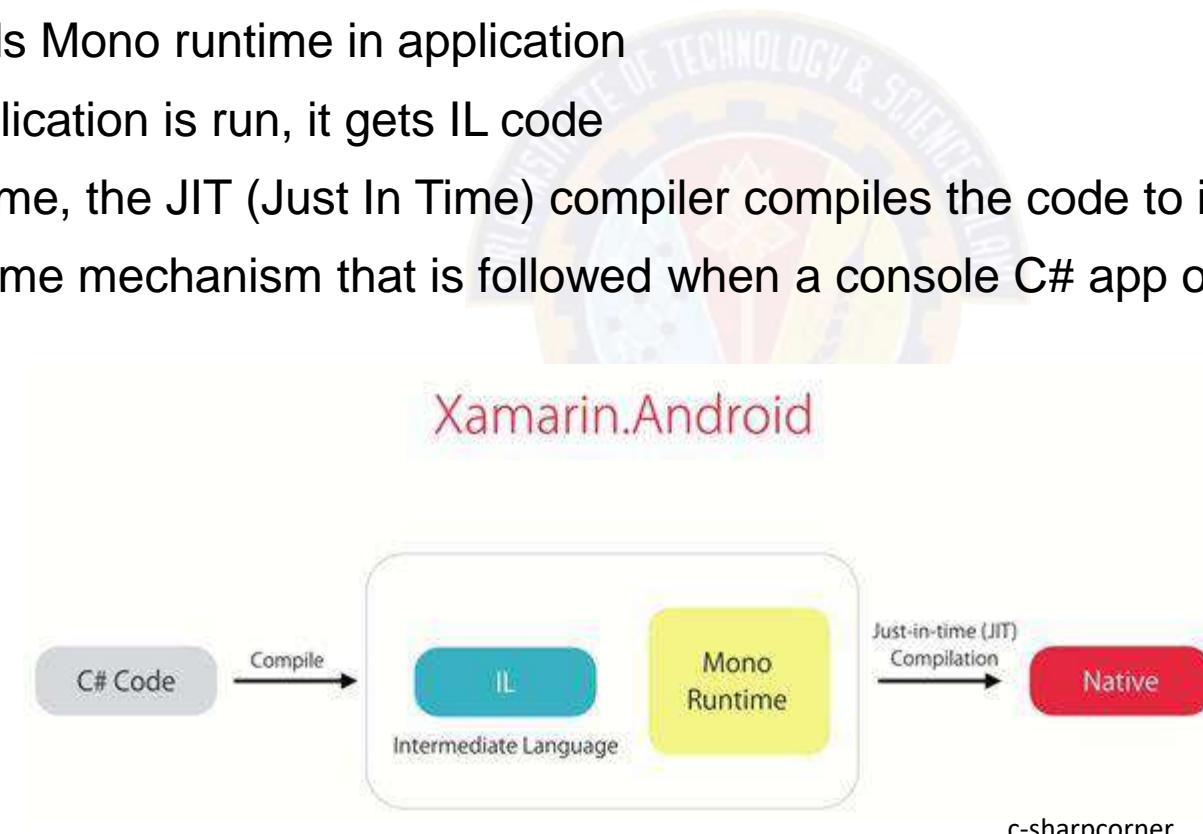


- Both of these products give access to the .NET base class library
  - ✓ also provide access to native APIs that allow dealing with platform-specific hardware and features



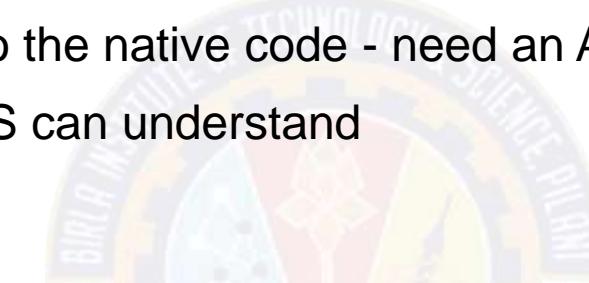
# Xamarin.Android

- When the code is run, the Xamairn C# compiler compiles the code into intermediate language (IL)
- It also embeds Mono runtime in application
- When an application is run, it gets IL code
- In the meantime, the JIT (Just In Time) compiler compiles the code to its native form
- This is the same mechanism that is followed when a console C# app or a Web app is built



# Xamarin.iOS

- iOS works in a different way
- The Xamarin C# compiler compiles code into the intermediate language(IL)
- The Apple Compiler compiles it to the native code - need an Apple machine
- The output is native code that iOS can understand
- There is no Mono runtime



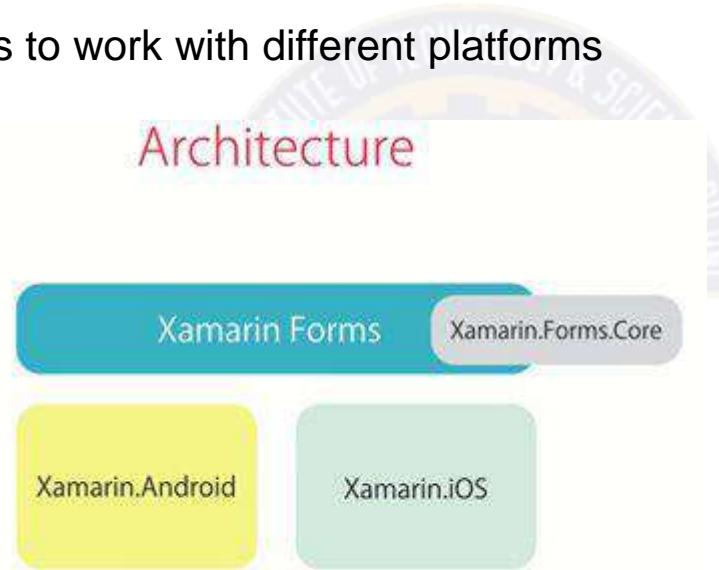
Xamarin.iOS



c-sharpcorner

# Xamarin.Forms

- Xamarin.Forms is built upon these two libraries
- The assembly is called `Xamarin.Forms.Core`
  - ✓ has common unified APIs to work with different platforms



c-sharpcorner

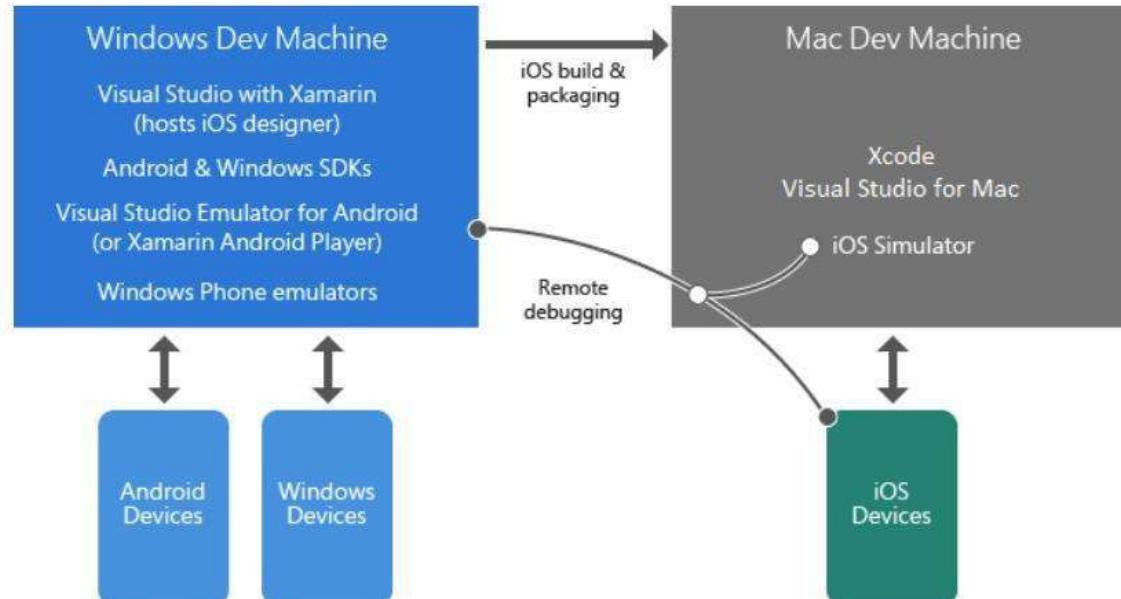
# Xamarin.Essentials

- Xamarin.Essentials is a library that provides cross-platform APIs for native device features
- abstraction that simplifies the process of accessing native functionality
- Some examples of functionality provided by Xamarin.Essentials include:
  - ✓ Device info
  - ✓ File system
  - ✓ Accelerometer
  - ✓ Phone dialer
  - ✓ Text-to-speech
  - ✓ Screen lock



# Xamarin project structure

- Most of the Xamarin-related work is expected to be run via a Windows development computer
  - ✓ with Visual Studio and Xamarin installed
- The apps can be debugged straight from the desktop or on devices and emulators
  - ✓ If plan to develop iOS apps on Windows, it's also possible as Visual Studio connects to the iOS storyboard designer and iOS simulator
  - ✓ There's also Visual Studio for Mac which allows for running a simulator on the Mac or directly on a tethered iPhone.





# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Xamarin Compared

Chandan Ravandur N



# Xamarin Compared

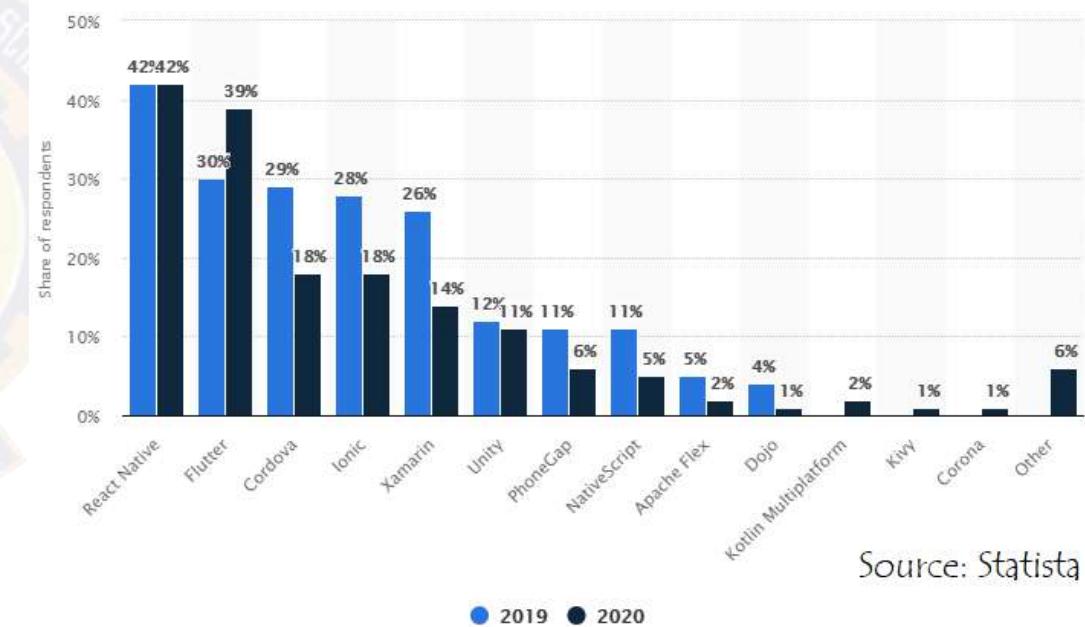
- Recently, many developers tend to agree that Xamarin can be considered “native” development tool
- There is an opinion that
  - ✓ “anything that can be done in an iOS application using Objective-C or Swift”
  - ✓ and “anything that can be done in an Android app using Java”
  - ✓ can be done in C# using Xamarin.
- Yet, there are many pitfalls in the native vs Xamarin debate
  - ✓ Let’s see how Xamarin compares to the native development tools and hybrid development platforms (Ionic, PhoneGap/Cordova)

# Xamarin Compared(2)

	Xamarin	Native	Hybrid
<b>Tech stack</b>	One tech stack, single codebase (C#, .Net framework + native libraries)	Different tech stacks for each platform.	One tech stack, single codebase (Javascript, HTML5, CSS)
<b>Code sharing</b>	Yes (up to 96% with Xamarin.Forms)	No, different code bases	Yes, 100%
<b>UI/UX</b>	Complete UI customization is possible for each platform (with Xamarin.iOS and Xamarin.Android)	Completely platform-specific UI	Common UI for all platforms (limited customization capabilities)
<b>Performance</b>	Good, close to native	Excellent	Medium - Poor
<b>Hardware capabilities</b>	High - Xamarin uses platform-specific APIs, and supports linking with native libraries	High - Native tools have complete support for system capabilities out of the box.	Medium - The capabilities can be accessed through third-party APIs and plugins, although there are some risks due to the poor quality and unreliability of most of these tools.
<b>Time to market</b>	With Xamarin.Forms the time to market is fast due to the limited customization and extensive code sharing. Xamarin.iOS and Xamarin.Android require slightly more time as the amount of custom code increases.	The time to market for iOS or Android native app might equal that of Xamarin.Forms or Hybrid tools. Yet, building apps for multiple platforms, will require you to either prolong time to market or increase the number of developers involved.	Hybrid solutions offer the fastest time to market thanks to the single code base and minimum customization. These tools are even used for prototyping and proof of concept projects.

# Xamarin Compared(3)

- Although hybrid mobile development tools are evolving quickly
  - ✓ still lack the performance and native capabilities that Xamarin offers at roughly the same cost
- As for choosing between Xamarin or native iOS/Android
  - ✓ have to consider the available time and budget (native development is usually more expensive and takes longer) and the type of app
  - ✓ If need high-end performance and a perfectly adjusted UI, it's worth going for native apps
- As more cross-platform frameworks emerge and develop
  - ✓ becomes harder for Xamarin to keep its position in the market
- Today, a number of frameworks have already outpaced Xamarin in terms of popularity and performance
  - ✓ React Native was voted the top choice of 2020, closely followed by Flutter



Source: Statista



# Thank You!

In our next session:



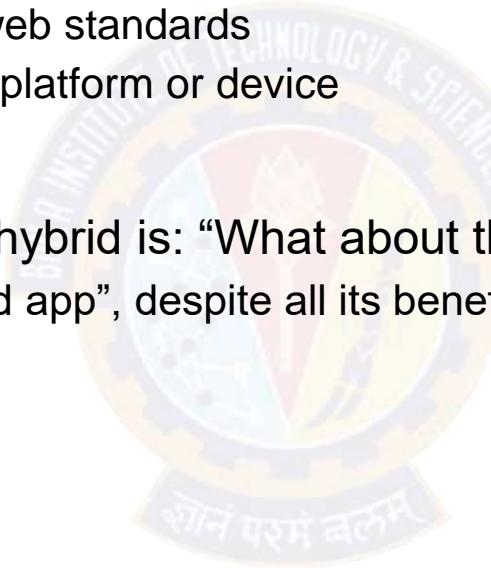
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Hybrid App Development

Chandan Ravandur N

# Benefits of Hybrid Apps

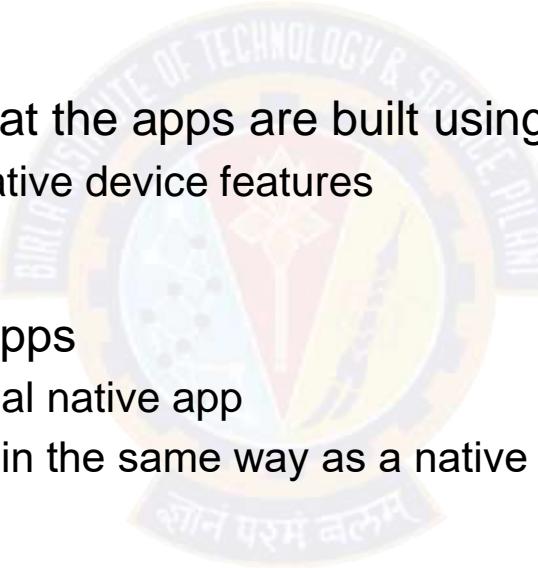
- A ton of benefits to choosing a hybrid framework like Ionic to build next mobile app
  - ✓ Use of familiar web languages
  - ✓ A stable platform based on open web standards
  - ✓ One shared codebase across any platform or device
- But the most common objection to hybrid is: “What about the user experience?!”
  - ✓ The implication being that a “hybrid app”, despite all its benefits, will deliver a sub-par user experience



# Defining the Hybrid Development Approach

## A brief definition of hybrid

- Hybrid development is a way of building mobile apps using open web technologies
  - ✓ like JavaScript, HTML, and CSS
- The term hybrid refers to the fact that the apps are built using web technology
  - ✓ but still have access to all of the native device features
- From a user's perspective, hybrid apps
  - ✓ look and feel identical to a traditional native app
  - ✓ can be accessed and downloaded in the same way as a native app is via any platform's app store



# How do they work?

- Hybrid apps run in a full-screen browser called a web view
  - ✓ which is invisible to the user
  - ✓ feels just like a native app
- Through customizable plugins, the app can access the native features of specific mobile devices
  - ✓ such as the camera, GPS, or touch ID
  - ✓ without the core code being tied to that device
- Means that hybrid-built apps can run on any platform or device, all from a single codebase,
  - ✓ while still delivering a completely native experience

# Why Hybrid?

## More time to focus on features

- Can build apps for multiple platforms using a single codebase
  - ✓ reduces the overhead of building and maintaining separate codebases for each target platform
  - ✓ desktop web, iOS, Android, Progressive Web App, etc.
  - ✓ leaving more time to focus on shipping features and improving app quality



# Why Hybrid? (2)

## Seamless experiences across platforms

- What makes a good user experience?
  - ✓ Solid performance and design are often must-haves
- But users have also come to expect a seamless app experience
  - ✓ as they move from device to device - or from mobile to desktop
  - ✓ easy to get with a hybrid cross-platform approach
- Can easily deliver the same set of features and experiences across desktop and mobile, all from a single codebase
  - ✓ When a framework is used, the look and feel of app will automatically adhere to the unique design patterns of whatever device users are on at the time
  - ✓ whether it's Android, iOS, or Windows

# Why Hybrid? (3)

## Real-time app updates

- A hidden gem of the hybrid approach is the ability to update app on the fly
  - ✓ without having to submit a new version to the app stores or ask users to update their app
- How is this possible?
  - ✓ App store policies require that any changes to native code must go through the app stores
  - ✓ However, it is permissible to make changes to the parts of app that are considered web asset
  - ✓ Since 90 percent of hybrid apps exist at the web layer, that means can change most of app by pushing changes directly
    - ❖ includes new features and critical bug fixes, which can be sent in real-time
- The ability to send live updates is a significant benefit cited by real-world users
- So, with a hybrid approach, can make sure that users always get
  - ✓ the best experience
  - ✓ with real-time bug fixes
  - ✓ seamless updates

# Why Hybrid? (4)

## Speed and performance

- Mobile app performance is absolutely critical to delivering a great user experience.
- How does hybrid stack up when it comes to load times, scrolling, and other moments where speed is critical?
- The best way to answer that is to find out for yourself
  - ✓ by checking out the many successful consumer apps built using hybrid technology
  - ✓ From award-winning apps like Sanvello and Sworkit to highly popular apps like Shipt and Untappd
- Take?
  - ✓ Unless you're building a highly-graphic intensive app, such as a 3D game,
  - ✓ a hybrid approach will deliver the speed and performance you need to make your users happy

Reference:

**How Hybrid App Development Helps Deliver Great UX**

By Rachel Brown

Ionic Blog



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Ionic Framework

Chandan Ravandur N

# Ionic Framework

- Ionic Framework is an open source UI toolkit
- for building performant, high-quality mobile and desktop apps
- using web technologies — HTML, CSS, and JavaScript
- — with integrations for popular frameworks like Angular, React, and Vue
- Freely available, open-source, and a front-end SDK framework that enables to create
  - ✓ mobile-based applications for iOS, Windows, and Android phones using the same codebase
  - ✓ proves out to be a cross-platform tool for mobile development
- Allows to create hybrid mobile applications
  - ✓ has a collection of components that provides the functionality of a mobile platform

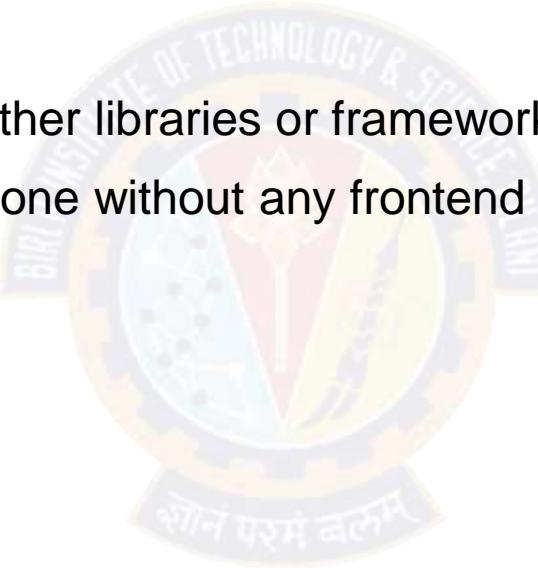


ionic

# Overview

## UI Framework

- Focuses on the frontend UX and UI interaction of an app
  - ✓ UI controls, interactions, gestures, animations
- Easy to learn, and integrates with other libraries or frameworks, such as Angular, React, or Vue
- Alternatively, it can be used standalone without any frontend framework using a simple script include



# Overview(2)

- One codebase, running everywhere
  - ✓ Ionic is the only mobile app stack that enables web developers to build apps
  - ✓ for all major app stores and the mobile web from a single codebase
  - ✓ With Adaptive Styling, Ionic apps look and feel at home on every device
- A focus on performance
  - ✓ Ionic is built to perform and behave great on the latest mobile devices with best practices
  - ✓ like efficient hardware accelerated transitions, and touch-optimized gestures
- Clean, simple, and functional design
  - ✓ Ionic is designed to work and display beautifully on all current mobile devices and platforms
  - ✓ With ready-made components, typography, and a gorgeous (yet extensible) base theme that adapts to each platform, will be building in style
- Native and Web optimized
  - ✓ Ionic emulates native app UI guidelines and uses native SDKs
  - ✓ bringing the UI standards and device features of native apps together with the full power and flexibility of the open web
  - ✓ Uses Capacitor (or Cordova) to deploy natively, or runs in the browser as a Progressive Web App

# Goals

- Cross-platform
  - ✓ Build and deploy apps that work across multiple platforms, such as native iOS, Android, desktop, and the web as a Progressive Web App - all with one code base
  - ✓ Write once, run anywhere
- Web Standards-based
  - ✓ Ionic Framework is built on top of reliable, standardized web technologies: HTML, CSS, and JavaScript
  - ✓ using modern Web APIs such as Custom Elements and Shadow DOM
  - ✓ Ionic components have a stable API, and aren't at the whim of a single platform vendor
- Beautiful Design
  - ✓ Clean, simple, and functional
  - ✓ Ionic Framework is designed to work and display beautifully out-of-the-box across all platforms
- Simplicity
  - ✓ Ionic Framework is built with simplicity in mind
  - ✓ creating Ionic apps is enjoyable, easy to learn, and accessible to just about anyone with web development skills

# Framework Compatibility

- Past releases of Ionic were tightly coupled to Angular
  - ✓ With version 4.x of the framework - re-engineered to work as a standalone Web Component library
  - ✓ with integrations for the latest JavaScript frameworks, like Angular
  - ✓ Ionic can be used in most frontend frameworks with success
  - ✓ including React and Vue, though some frameworks need a shim for full Web Component support
- JavaScript
  - ✓ One of the main goals with moving Ionic Framework to Web Components was to remove any hard requirement on a single framework to host the components
  - ✓ Made it possible for the core components to work standalone in a web page with just a script tag
  - ✓ While working with frameworks can be great for larger teams and larger apps
  - ✓ it is now possible to use Ionic as a standalone library in a single page even in a context like WordPress

# Framework Compatibility(2)

- Angular
  - ✓ Angular has always been at the center of what makes Ionic great
  - ✓ @ionic/angular includes all the functionality that Angular developers would expect coming from Ionic 2/3
- React
  - ✓ Ionic now has official support for the popular React library
  - ✓ Ionic React lets React developers use their existing web skills to build apps that target iOS, Android, the web, and the desktop
- Vue
  - ✓ Also has official support for the popular Vue 3 library
  - ✓ With @ionic/vue, can use all the core Ionic components, but in a way that feels like using native Vue components

# Tooling

- Ionic CLI
  - ✓ The official Ionic CLI, or Command Line Interface, is a tool
  - ✓ that quickly scaffolds Ionic apps and provides a number of helpful commands to Ionic developers
  - ✓ Comes with a built-in development server, build and debugging tools, and much more
  - ✓ Can be used to perform cloud builds and deployments with Appflow, and administer your account
- Appflow
  - ✓ To help build, deploy, and manage Ionic apps throughout their lifecycle
    - ❖ a commercial service for production apps called Appflow, which is separate from the open source Framework
  - ✓ Helps developers and teams compile native app builds and deploy live code updates to Ionic apps from a centralized dashboard
  - ✓ Optional paid upgrades are available for more advanced capabilities
  - ✓ like publishing directly to app stores, workflow automation, single sign-on (SSO) and access to connected services and integrations

# Ecosystem

- Ionic Framework is actively developed and maintained full-time by a core team
  - ✓ Ecosystem is guided by an international community of developers and contributors fueling its growth and adoption
  - ✓ Developers and companies small and large use Ionic to build and ship amazing apps that run everywhere
- Here are some communities :
  - ✓ Forum: A great place for asking questions and sharing ideas.
  - ✓ Slack: A lively place for devs to meet and chat in real time.
  - ✓ Twitter: Where post updates and share content from the Ionic community.
  - ✓ GitHub: For reporting bugs or requesting new features, create an issue here.
  - ✓ Content authoring: Write a technical blog or share your story with the Ionic community.

Reference:  
Ionic Framework documentation



# Thank You!

In our next session:



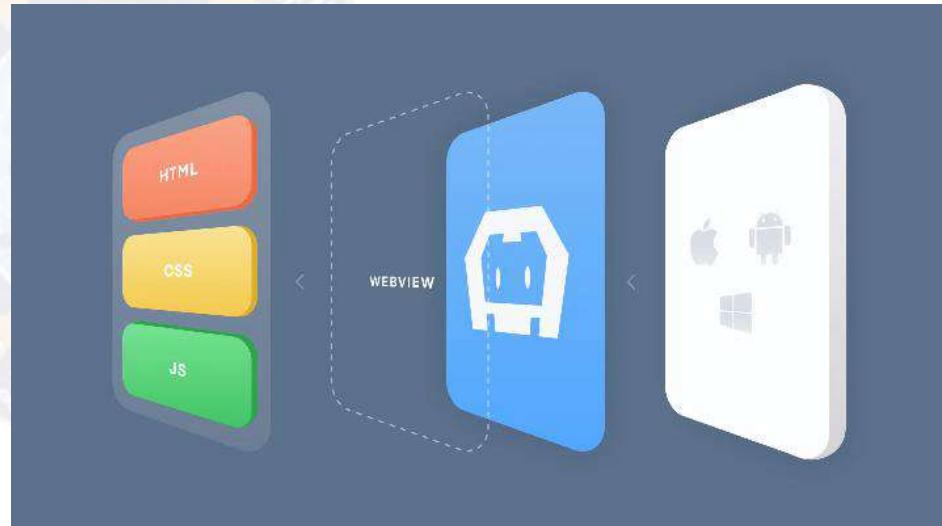
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Apache Cordova

Chandan Ravandur N

# Apache Cordova

- Apache Cordova is an open source framework
- enables web developers to use their HTML, CSS, and JavaScript content
- to create a native application for a variety of mobile platforms



ionic

# How it works?

- Cordova takes web application and renders it within a native WebView
- A WebView is an application component (like a button or a tab bar)
  - ✓ that is used to display web content within a native application
- WebView as a web browser without any of the standard user interface elements, such as a URL field or status bar
- The web application running inside this container
  - ✓ is just like any other web application that would run within a mobile browser
    - ❖ can open additional HTML pages
    - ❖ execute JavaScript code
    - ❖ play media files
    - ❖ and communicate with remote servers
- Often called a hybrid application



# How it works?(2)

- Typically, web-based applications are executed within a sandbox
  - ✓ do not have direct access to various hardware and software features on the device
- A good example of this is the contact database on mobile device
  - ✓ names, phone numbers, emails, and other bits of information is not accessible to a web app
- Besides providing a basic framework to run a web app within a native application
  - ✓ Cordova also provides JavaScript APIs to allow access to a wide variety of device features, like the contacts database
  - ✓ Capabilities are exposed through the use of a collection of plugins
  - ✓ Plugins provide a bridge between web application and the device's native features
- The Ionic team has created Ionic Native, with TypeScript interfaces for over 200 of the most common plugins

# The History of Cordova (aka PhoneGap)

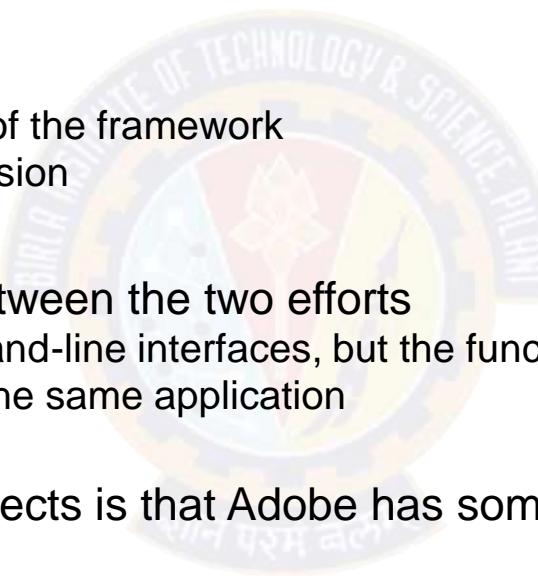
## Developers are often confused by the difference between Apache Cordova and PhoneGap

- In late 2008, several engineers from Nitobi, a web development company from Canada,
  - ✓ attended an iPhone development camp at the Adobe offices in San Francisco
  - ✓ explored the idea of using the native WebView as a shell to run their web applications in a native environment
  - ✓ it worked!
- Expanded their efforts and were able to leverage this solution to create a framework
  - ✓ resulted into the project PhoneGap since it allowed web developers the ability to bridge the gap between their web apps and the device's native capabilities
  - ✓ continued to mature, and more plugins were created, enabling more functionality to be accessed on the phone
  - ✓ Other contributors joined the effort, expanding the number of mobile platforms it supported
- In 2011, Adobe bought Nitobi, and the PhoneGap framework was donated to the Apache Foundation
  - ✓ project was eventually renamed Cordova
  - ✓ is actually the street name of Nitobi's office in Vancouver, Canada

# Apache Cordova versus Adobe PhoneGap

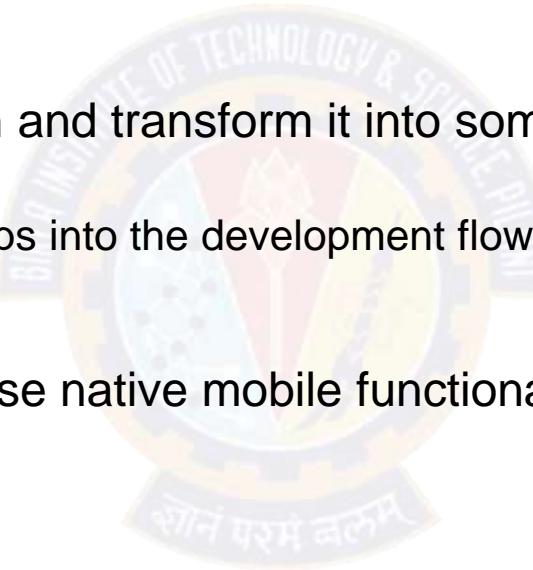
## quite easy to confuse the projects

- A good way to understand the difference is to think about how Apple has its Safari browser
  - ✓ but it is based on the open source WebKit engine
- The same is true here:
  - Cordova is the open source version of the framework
  - PhoneGap is the Adobe-branded version
- In the end, there is little difference between the two efforts
  - ✓ some slight differences in the command-line interfaces, but the functionality is the same
  - ✓ cannot do is mix the two projects in the same application
- The main difference between the projects is that Adobe has some paid services under the PhoneGap brand
  - ✓ most notably the PhoneGap Build service
  - ✓ a hosted service that enables to have application compiled into native binaries remotely
  - ✓ eliminating the need to install each mobile platform's SDKs locally
  - ✓ The PhoneGap command line interface tool (CLI) has the ability to utilize this service, while the Cordova CLI does not



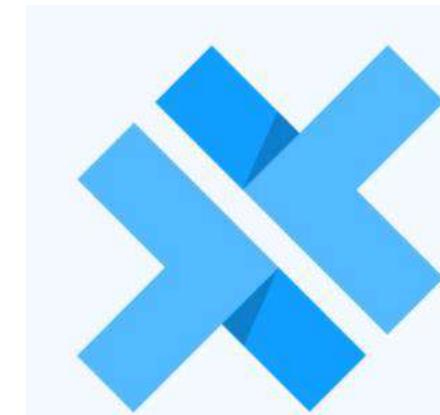
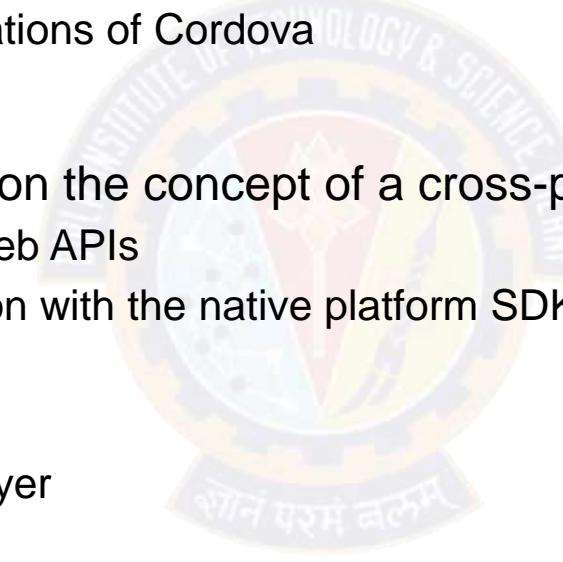
# Ionic and Cordova

- Ionic's primary role in the application development space is
  - ✓ to provide consistent, platform-specific user interface components
- How does one take their application and transform it into something that can be submitted to the various app stores?
  - ✓ That is where Apache Cordova steps into the development flow
- Cordova provides the solutions to use native mobile functionality and to create fully native applications
  - ✓ doesn't include a UI SDK
- Since the beginning of Ionic, Cordova has been an integral part of the project
  - ✓ With over 100 UI components, plus navigation and platform-specific styling
  - ✓ Ionic allows to develop high performing, native-like apps



# Capacitor: An Alternative to Cordova

- Now, Cordova is not the only solution for this
- Ionic recently released a successor to Cordova, under an open source project called Capacitor
  - ✓ aims to improve upon the foundations of Cordova
- Capacitor is a more modern take on the concept of a cross-platform native runtime for the web
  - ✓ taking advantage of the latest Web APIs
  - ✓ providing much deeper integration with the native platform SDKs
  - ✓ allowing greater customization
  - ✓ easier troubleshooting
  - ✓ and fewer issues at the native layer



Reference

What is Apache Cordova?

By Chris Griffith

Ionic blog



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Capacitor

Chandan Ravandur N

# Capacitor

- Capacitor was first released in 2019 as a new and revolutionary way to deliver
  - ✓ native mobile apps and Progressive Web Apps (PWAs) from the same codebase
  - ✓ using web technology in place of native programming languages
- The Ionic team created Capacitor as a spiritual successor to Apache Cordova and Adobe PhoneGap
  - ✓ with inspiration from other popular cross-platform tools like React Native and Turbolinks
  - ✓ but focused entirely on enabling modern web apps to run on all major platforms with ease
- Open source project that runs modern Web Apps natively on iOS, Android, Electron, and Web
  - ✓ while providing a powerful and easy-to-use interface for accessing Native SDKs and Native APIs on each platform
- As an alternative to Cordova, Capacitor delivers the same cross-platform benefits,
  - ✓ but with a more modern approach to app development
  - ✓ taking advantage of the latest Web APIs and native platform capabilities

# Project flow with Capacitor

- Capacitor embraces npm to manage plugins, Swift for iOS, Java for Android,
  - ✓ TypeScript has full support for Progressive Web Apps
- Capacitor makes it possible to build one app for iOS, Android, and the Web, all with one code base
- Capacitor has first-class support for building plugins
  - ✓ in fact the plugin authoring experience has been a focus from the very beginning
- Capacitor treats native projects as source-artifacts, meaning native code can quickly be added
  - ✓ all plugins are available as soon as the page starts loading
- Today, Capacitor is the most cost effective path to deploying a single app to multiple platforms

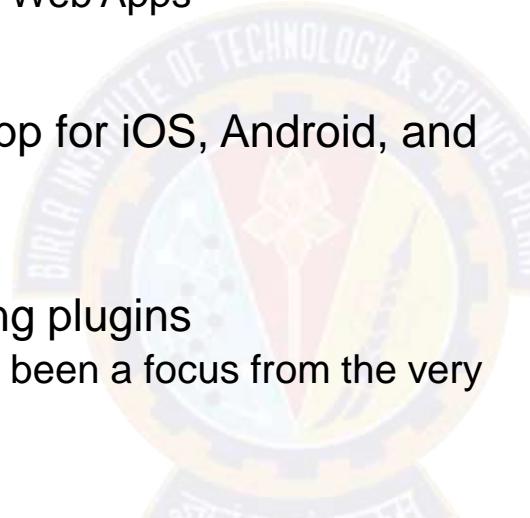
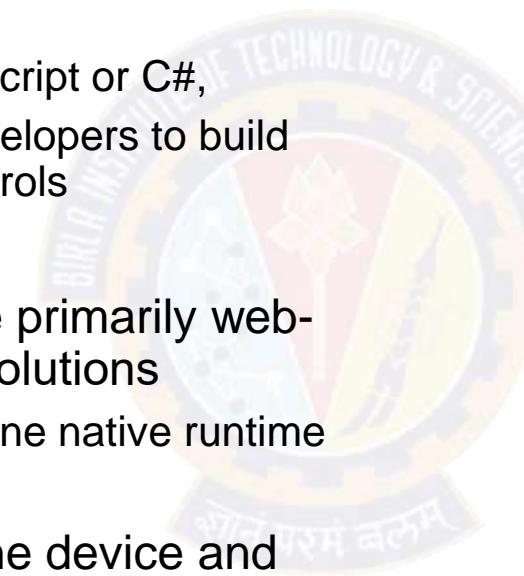


Figure 1 - Project flow with Capacitor

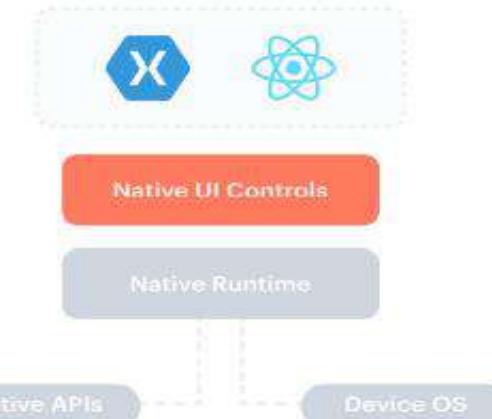
Capacitor whitepaper

# How it works?

- “Cross-platform native” solutions like React Native and Xamarin are tightly coupled with the native mobile platforms
  - ✓ Written and controlled largely in JavaScript or C#,
  - ✓ boast solid performance and allow developers to build mobile experiences with native UI controls
- In contrast, Cordova and Capacitor are primarily web-based in nature - referred as “hybrid” solutions
  - ✓ in reference to the fact that they combine native runtime features with a web UI layer
- Although hybrid apps run natively on the device and interact with any available native APIs
  - ✓ the UI layer of the app is executed primarily in a web browser, known as a WebView, that is invisible to the user



Xamarin/React Native



Capacitor/Cordova



# Easy access to native device features

## Plugins

- Developers have access to the full Native SDK on each platform, and can easily deploy to App Stores, and the web
- Adding native functionality is easy with a simple Plugin API
  - ✓ for Swift on iOS, Java on Android, and JavaScript for the web
- Capacitor apps access native and web features using a series of platform APIs (AKA plugins)
- While the base Capacitor experience includes a series of core plugins, along with OSS contributed plugins,
  - ✓ can easily add own custom device functionality by writing a plugin of own
- Using Capacitor, developers can build one app and target a single set of APIs regardless of the platform the app is running on
  - ✓ as opposed to managing multiple APIs for each target platform
- For example , accessing the Camera uses the same code on iOS/Android as it does on Electron and on the web

# User Interface (UI) layer?

- While Capacitor gives the native runtime environment to run app on mobile, still need to think about the UI layer of app
- Couple of options :
- Bring your own library or framework
  - ✓ UI of a Capacitor app runs primarily in the browser
  - ✓ Opens up a whole world of possibilities for web developers and teams
  - ✓ who want to bring their web UI components and applications to mobile
  - ✓ Means any popular UI framework like Bootstrap, Material, or Tailwind will work great in Capacitor
  - ✓ existing web-based library will now run natively on any iOS or Android device, and on the web as a PWA
- Use a mobile-ready UI framework like Ionic
  - ✓ Mobile styling, navigation, and performance on mobile devices are all very tricky
  - ✓ As an alternative to rolling own solution from scratch
  - ✓ can use a mobile-ready UI library like Ionic Framework instead
  - ✓ can then incrementally add own custom components as needed
  - ✓ Or
  - ✓ start with UI library as the foundation
  - ✓ incrementally add Framework components as needed

Reference:  
Cordova vs Capacitor  
Max Lynch  
Ionic Blog



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Intro to APIs

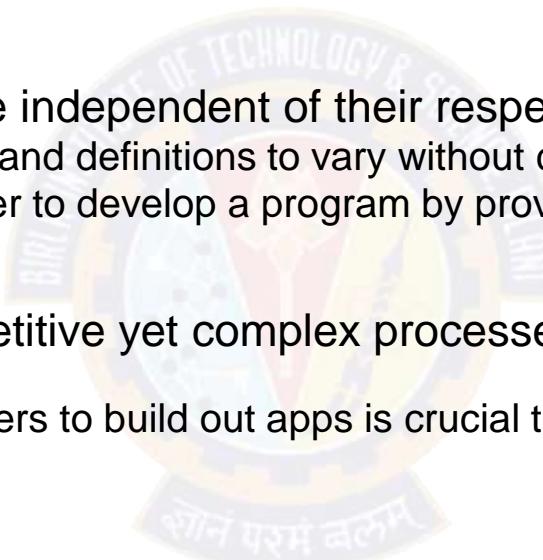
Chandan Ravandur N

# What exactly is an API?

- API stands for application programming interface
  - ✓ are the little pieces of code that make it possible for digital devices, software applications, and data servers to talk with each other
  - ✓ are the essential backbone of so many services we now rely on!
- An easy way to understand the definition of an API is to think about the applications that you use every day
- In an internet-connected world, web and mobile applications are designed for humans to use
  - ✓ while APIs are designed for other digital systems and applications to use
- Websites and APIs both do the same things, like return data, content, images, video, and other information
  - ✓ But APIs don't return all the details that are needed to make things look pretty for the human eye
  - ✓ you only get the raw data and other machine-readable information needed behind the scenes
  - ✓ to put the resources being delivered to work, with very little assistance from a human

# What is an API?

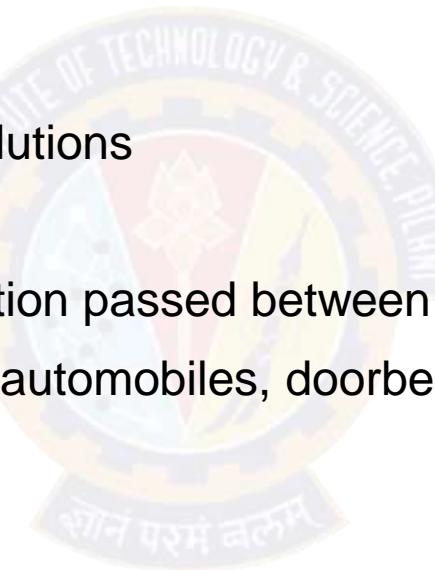
- An API is a software intermediary that allows two applications to talk to each other
  - ✓ API is the messenger that delivers your request to the provider that you're requesting it from and then delivers the response back to you
- An API defines functionalities that are independent of their respective implementations
  - ✓ which allows those implementations and definitions to vary without compromising each other
  - ✓ Therefore, a good API makes it easier to develop a program by providing the building blocks
- APIs enable developers to make repetitive yet complex processes highly reusable with a little bit of code
  - ✓ The speed that APIs enable developers to build out apps is crucial to the current pace of application development
- Developers are now much more productive than they were before when they had to write a lot of code from scratch
  - ✓ With an API they don't have to reinvent the wheel every time they write a new program
  - ✓ Instead, they can focus on the unique proposition of their applications while outsourcing all of the commodity functionality to APIs



# What are APIs used for?

**For lots and lots and lots of things,**

- APIs power desktop applications
- APIs are behind most web applications
- APIs make mobile applications possible
- APIs are the integrations for no code solutions
- APIs connect devices to the internet
- APIs define the networks or the information passed between applications, systems, and devices
- APIs even connect everyday things like automobiles, doorbells, dishwashers, and wearable devices



# Why should you care about APIs?

## A very brief list

- APIs help you access the data you need to get your work done and do daily tasks
  - ✓ whether you're a business user, a student, or using an application just for fun
- APIs make it possible to integrate different systems together
  - ✓ like Customer Relationship Management systems, databases, or even school learning management systems
- APIs help different departments, teams, and groups become more agile
- APIs help organizations, schools, government agencies, and nonprofits strengthen relationships with other organizations, research institutes, and agencies

# A real example of an API

## How are APIs used in the real world?

- When you search for flights online, you have a menu of options to choose from
  - ✓ choose a departure city and date, a return city and date, cabin class
  - ✓ and other variables like your meal, your seat, or baggage requests
- To book your flight, you need to interact with the airline's website to access the airline's database
  - ✓ to see if any seats are available on those dates, and what the cost might be based on the date, flight time, route popularity, etc.
- You're interacting with it from the website or an online travel service that aggregates information from multiple airlines
- Alternatively, you might be accessing the information from a mobile phone
  - ✓ In any case, you need to get the information, and so the application must interact with the airline's API, giving it access to the airline's data
- The API is the interface that, like your helpful waiter, runs and delivers the data from the application you're using to the airline's systems over the Internet
  - ✓ It also then takes the airline's response to your request and delivers right back to the travel application you're using
  - ✓ Moreover, through each step of the process, it facilitates the interaction between the application and the airline's systems
  - ✓ from seat selection to payment and booking!
- APIs do the same for all interactions between applications, data, and devices
  - ✓ allow the transmission of data from system to system, creating connectivity
  - ✓ provide a standard way of accessing any application data, or device
  - ✓ whether it's accessing cloud applications like Salesforce, or shopping from your mobile phone

Reference:

Intro to APIs: What Is an API?

Postman Blog



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# How API works?

Chandan Ravandur N

# API Working

## The customer, a waiter, and a restaurant kitchen!

- APIs work by sharing data and information between applications, systems, and devices—making it possible for these things to talk with each other
- Sometimes the easiest way to think about APIs is to think about a metaphor, and a common scenario
- The customer, a waiter, and a restaurant kitchen!
  - ✓ A customer talks to the waiter and tells the waiter what she wants
  - ✓ The waiter takes down the order and communicates it to the kitchen
  - ✓ The kitchen does their work, creating the food
  - ✓ Then the waiter delivers the order back to the customer



[medium](#)

# API Working(2)

## The customer, a waiter, and a restaurant kitchen!

- In this metaphor, a customer is like a user, who tells the waiter what she wants
- The waiter is like an API,
  - ✓ receiving the customer's order
  - ✓ translating the order into easy-to-follow instructions that the kitchen then uses to fulfill that order—often following a specific set of codes
  - ✓ or input, that the kitchen easily recognizes
- The kitchen is like a server that does the work of creating the order in the manner the customer wants it, hopefully!
- When the food is ready, the waiter picks up the order and delivers it to the customer.



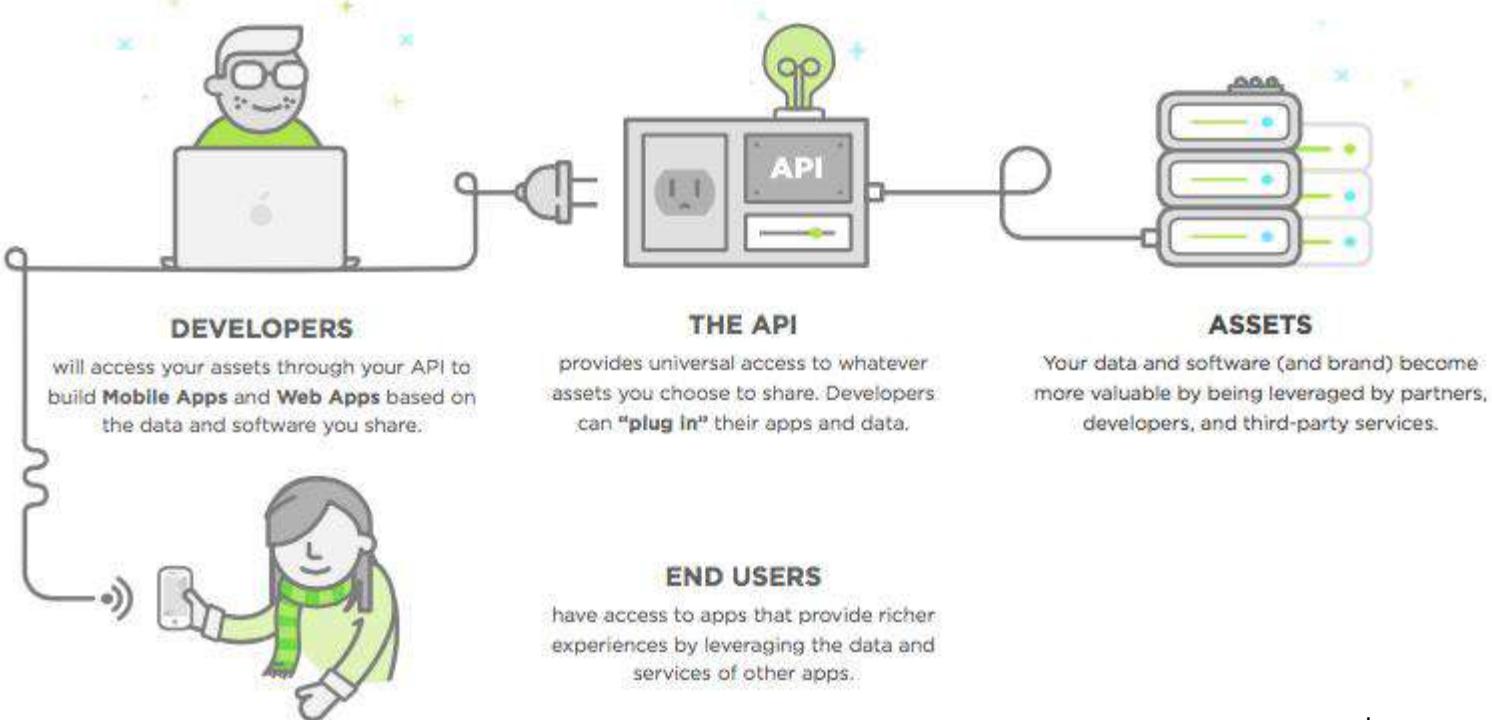
# API Working(3)

## Explained!

- An Application Programming Interface (API) defines the allowed interactions between two pieces of software
  - ✓ just like a User Interface defines the ways in which a user can interact with a program
- An API is composed of the list of possible methods
  - ✓ to call (requests to make), their parameters, return values and any data format they require (among other things)
  - ✓ equivalent to how a user's interactions with a mobile phone app are limited to the buttons, sliders and text boxes in the app's User Interface
- APIs can be local, where both interacting parties run on the same machine.
  - ✓ For example
  - ✓ the Windows API offered by the Operating System to its applications,
  - ✓ the Standard C Library offered by any C compiler to the programs being compiled
  - ✓ the TensorFlow API offered by this machine learning library to programs using it
- Remote APIs allows interacting parties run on separate machines and communicate over a network
  - ✓ For example,
  - ✓ a public weather service offering up machine-readable forecasts to be consumed by web pages mobile applications
  - ✓ Twitter allowing third-party applications to send messages through its network
- Party offering up its services through an API is called the provider
- Party requesting these services is the consumer!

# API Working(4)

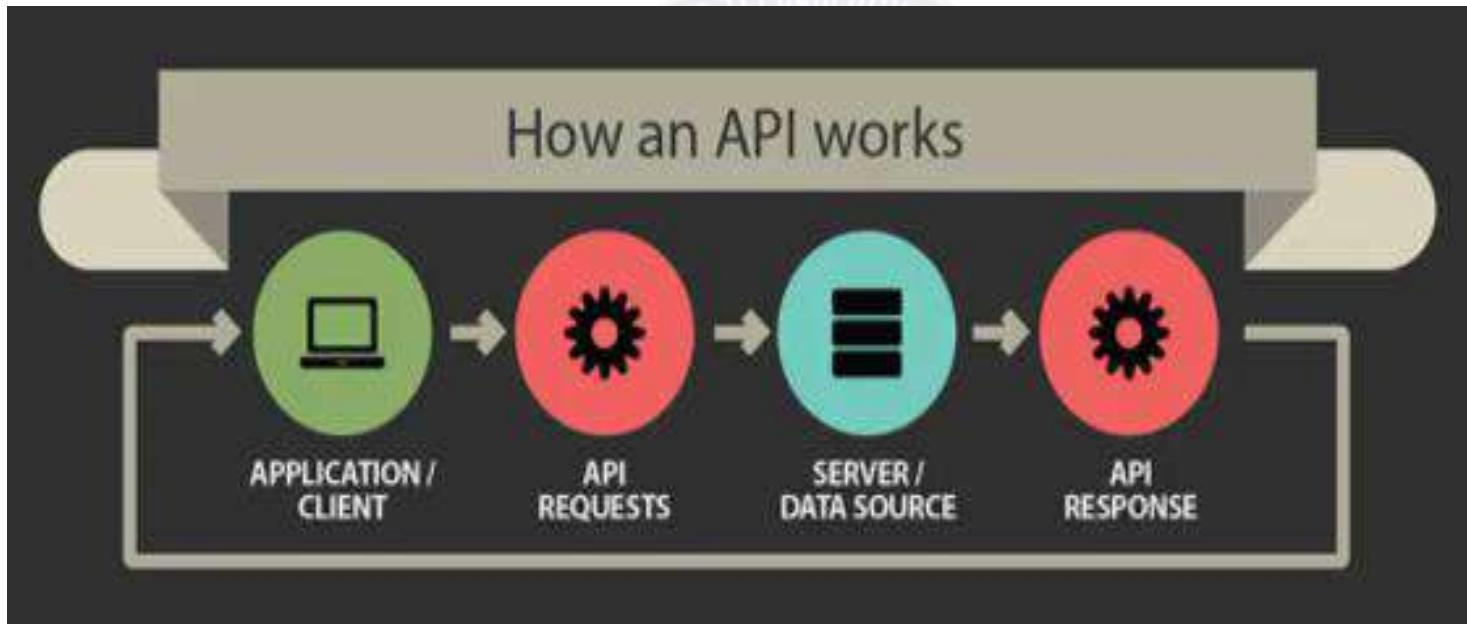
## Parties involved



upwork

# API Working(5)

## Summarized



ujudebug



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# History of APIs

Chandan Ravandur N

# API Starting

- In order to understand why APIs are so important, need to take a look at how we got here
- While web APIs have a relatively brief history
  - ✓ it's resulted in them being behind almost every aspect of how we do business online
- When you hear the acronym “API”, it is almost always in reference to modern approach,
  - ✓ in that we use HTTP to provide access to machine readable data in a JSON or XML format
  - ✓ often simply referred to as “web APIs.”
- APIs have been around almost as long as computing, but modern web APIs began taking shape in the early 2000s
  - ✓ A popular dissertation on REST by Roy Fielding and a handful of emerging technology companies including Salesforce, eBay, and Amazon
  - ✓ led to the definition of web APIs we use today

# A very commercial API beginning

## The “three”

- Web APIs got their start by putting the “commercial” in “.com”
  - ✓ powering the vision of emerging commerce startups looking to change the way we do business on the web
  - ✓ took advantage of this new medium to make products and services available to customers via a single website
  - ✓ ensured that partners and third party resellers could extend the reach of their platform
- The commerce age of web APIs was dominated by three companies
  - ✓ Salesforce, eBay, and Amazon!
  - ✓ Twenty years later, they remain commercial powerhouses and continue to shape the world of APIs
- Salesforce
  - ✓ officially launched its API on February 7, 2000 at the IDG Demo conference
  - ✓ introduced an enterprise-class, web-based, Salesforce automation: “Internet as a Service”
  - ✓ XML APIs were a fundamental part of how Salesforce did business from day one
- eBay
  - ✓ On November 20, 2000, eBay launched the eBay Application Program Interface (API) along with the eBay Developers Program
  - ✓ originally rolled out to only a select number of licensed eBay partners and developers
  - ✓ but ultimately shifted how goods are now sold on the web
- Amazon
  - ✓ On July 16, 2002, Amazon launched Amazon.com Web Services
  - ✓ allowed developers to incorporate Amazon.com content and features into their own websites
  - ✓ let third party sites search and display products from Amazon.com in an XML format



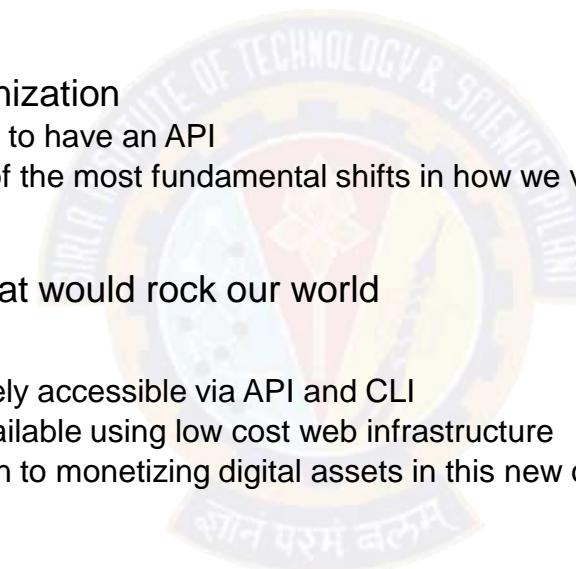
# Making the web much more social

- In 2004, a shift in the API landscape began to emerge - a new breed of API providers started to pop up
  - ✓ This new group changed how we use the web and share information with the people around us, both in the real world and virtually
  - ✓ These new APIs weren't as directly linked to commercial value as their counterparts
  - ✓ but they provided value to their organizations, and became lucrative platforms down the road
- Around this time there were four major developments that set the stage for the incredible growth that happened from 2006 through 2012.
- Flickr
  - ✓ In February 2004, the popular photo sharing site Flickr launched
  - ✓ Six months later they produced their now infamous API, which allowed users to easily embed their photos on web pages and social media
  - ✓ quickly became the image platform of choice for the emerging social media movement
- Facebook
  - ✓ In August 2006, Facebook launched its long-awaited development platform and API
  - ✓ Version 1.0 allowed developers to access Facebook users' friends, photos, events, and profile information
  - ✓ helped Facebook become one of the most popular social networks to date
- Twitter
  - ✓ In September 2006, Twitter introduced its own API in response to developers increasingly scraping content and data from the platform
  - ✓ Twitter incorporated APIs into almost every feature of the product we know today, from its mobile application to the share button



# Moving everything into the Cloud

- As this new social reality unfolded, a seismic shift occurred that would completely transform how business was done online
  - ✓ Having used APIs to power their commercial visions with unprecedented success
  - ✓ Amazon became the model for the next generation of startups and enterprises alike
- Amazon's model was API-focused across its organization
  - ✓ Internally, all shared digital resources were required to have an API
  - ✓ As companies followed their lead, thus began one of the most fundamental shifts in how we view digital resources
- Two new Amazon Web Services soon emerged that would rock our world
- Amazon Simple Storage (S3)
  - ✓ began offering a basic storage service that was solely accessible via API and CLI
  - ✓ Suddenly essential digital resources were made available using low cost web infrastructure
  - ✓ A pay-as-you-go model was introduced as a solution to monetizing digital assets in this new online economy
- Amazon Elastic Compute (EC2)
  - ✓ Just six months after S3 was released, Amazon would release another service called EC2
  - ✓ Provided servers that developers could leverage to deploy the necessary infrastructure for the next generation of applications
- Amazon Web Services changed everything
  - ✓ demonstrated that web APIs could be used to deploy infrastructure, generate revenue, and fundamentally change the way companies do business



# Everything is becoming more mobile

- In 2007, Apple launched the iPhone
- radically changed not only how we engaged with our mobile phones, but how we engaged with the online world
  - ✓ Same year, Google responded by launching Android: an open source mobile platform
  - ✓ These developments spurred a massive investment in new startups
  - ✓ looking to provide much-needed resources and applications to meet the growing public demand
  - ✓ A few of these API-first startups became the blueprint for how APIs are delivered today
- Google Maps
  - ✓ In 2006, Google launched a new mapping solution called Google Maps
  - ✓ Six months later, the Google Maps API was made available in direct response to the number of rogue applications
  - ✓ Location became a major topic of the API conversation that we were about to have on every mobile device sold in the next couple of years
- Instagram
  - ✓ The mobile evolution of the Internet was underway when on October 6, 2010, Instagram launched its photo-sharing iPhone application
  - ✓ Less than three months later, it had one million users
  - ✓ Instagram focused on delivering a powerful but simple iPhone app that solved common problems with the quality of mobile photos as well as user frustration around sharing them
  - ✓ Immediately, many users complained about the lack of an Instagram API
  - ✓ By January, Instagram had shut down the rogue API, but had also announced that it was building one of its own
- Twilio
  - ✓ In 2007, a new API-as-a-product platform launched called Twilio
  - ✓ introduced a voice API, which allowed developers to make and receive phone calls via any cloud application, and capitalized on the growing need for voice-enabled apps
  - ✓ Over the next decade, Twilio became synonymous with the essential resources we need on our phones like voice, SMS, email, and other messaging and communication applications
- The mobile evolution of the web made APIs what they are today
  - ✓ Commerce, social, and the Cloud laid the foundation, but mobile put the web in our pockets
  - ✓ API-powered mobile applications have led to waves of evolution not just in mobile phones
  - ✓ but in any object that can be connected to the Internet!



# APIs powering next-generation devices

- Some developers also began thinking about how everyday objects could be connected to the web
  - ✓ If devices such as cameras, thermostats, speakers, microphones, and sensors could connect to WiFi or cellular networks
  - ✓ they could also send or receive data, content, media, and other digital resources via this magical new place called “the Cloud”
  - ✓ Common objects were granted a new life by allowing users to connect in ways they could never have imagined just a few years before

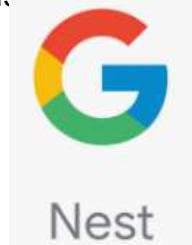
- Fitbit

- ✓ was established in March of 2007 and set out to deliver a range of products including wireless-enabled wearable technology devices
  - ✓ that could measure data like step count, heart rate, quality of sleep, and various other fitness metrics
  - ✓ connected our health and activity to the cloud, and opened up a whole new industry around connected, wearable applications powered by APIs



- Nest

- ✓ Now owned by Google, Nest began in 2010 as Nest Labs, and first produced a connected home thermostat
  - ✓ Later on, Nest released a smoke detector and series of home cameras, all available via a rich API developer ecosystem



- Alexa

- ✓ In November 2014, Amazon announced Alexa alongside their Echo product, dubbed as a smart speaker
  - ✓ Alexa is a virtual assistant that is capable of voice interaction, music playback, making to-do lists, setting alarms, streaming podcasts, playing audiobooks, and providing weather, traffic, sports, and other real-time updates!
  - ✓ Each feature leverages the API developer ecosystem to deliver a new generation of voice-enabled applications



- Connectivity showcases how APIs can be about much more than just building desktop, web, and mobile applications

- ✓ can be put to work creating what many call an “Internet of Everything”
  - ✓ APIs are no longer just at startups—due to their success, mainstream corporations, organizations, institutions, and government agencies are beginning to adopt them at scale.

# We are only just beginning the journey

- Represents less than twenty years of history
- It's only the beginning, but we have the building blocks for a connected world
  - ✓ Web APIs can be used to connect almost anything to the Internet
  - ✓ are regularly being used to invent entirely new products and services, as well as constructing ecosystems
- In 2010, innovative startups were putting APIs to work, and in 2019, mainstream industries are putting APIs to work
  - ✓ The collective momentum of industry change that is being set into motion will continue to shift the landscape
  - ✓ making it a challenge for some to keep up!
- Those who master the deployment and integration of APIs, who are able to adapt and transform
  - ✓ will lead the way into the future!

Reference:

Intro to APIs: History of APIs

Postman Blog



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

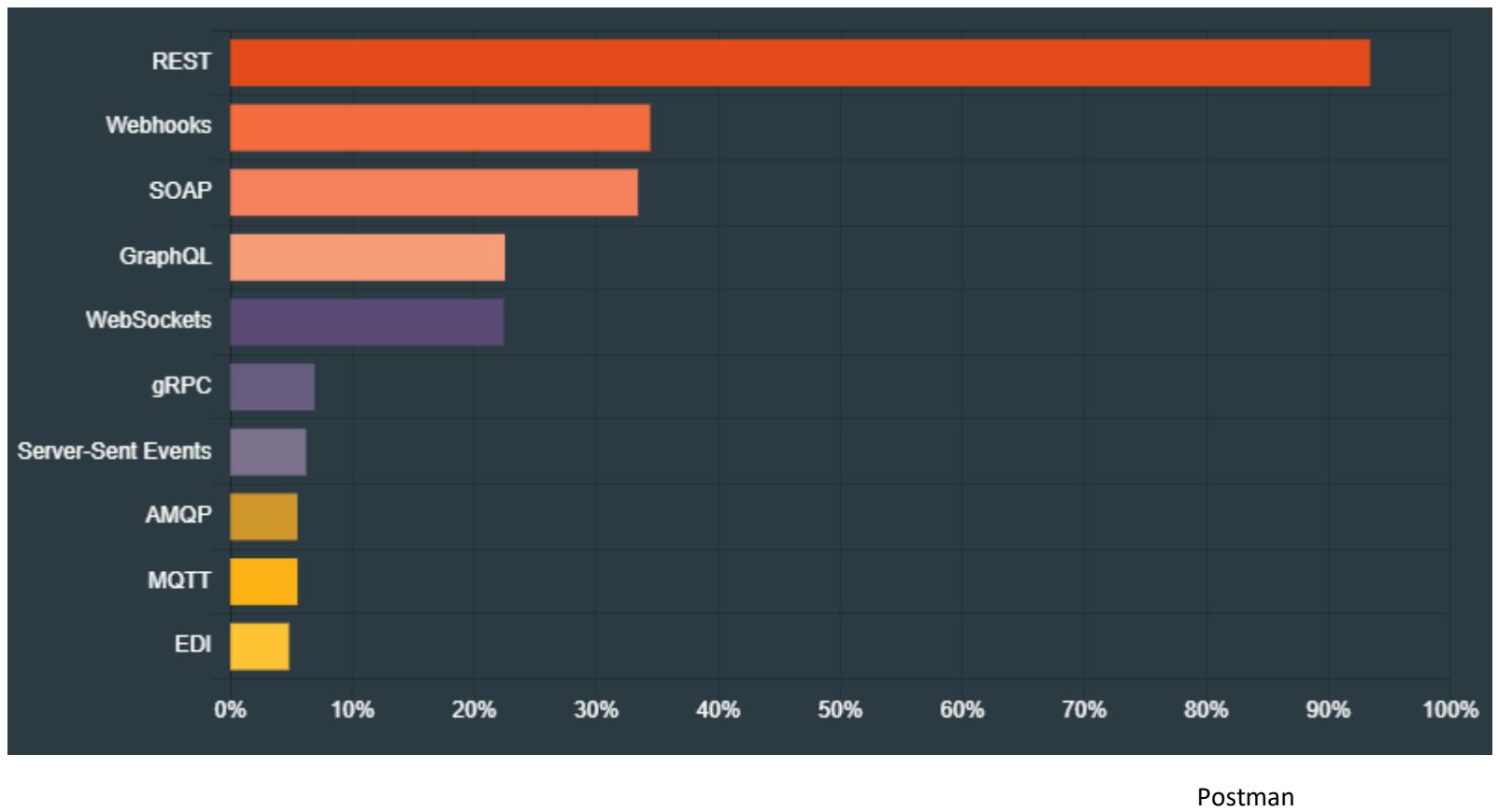
# API Technologies and Tooling

Chandan Ravandur N

# API Technologies

## API architectural styles

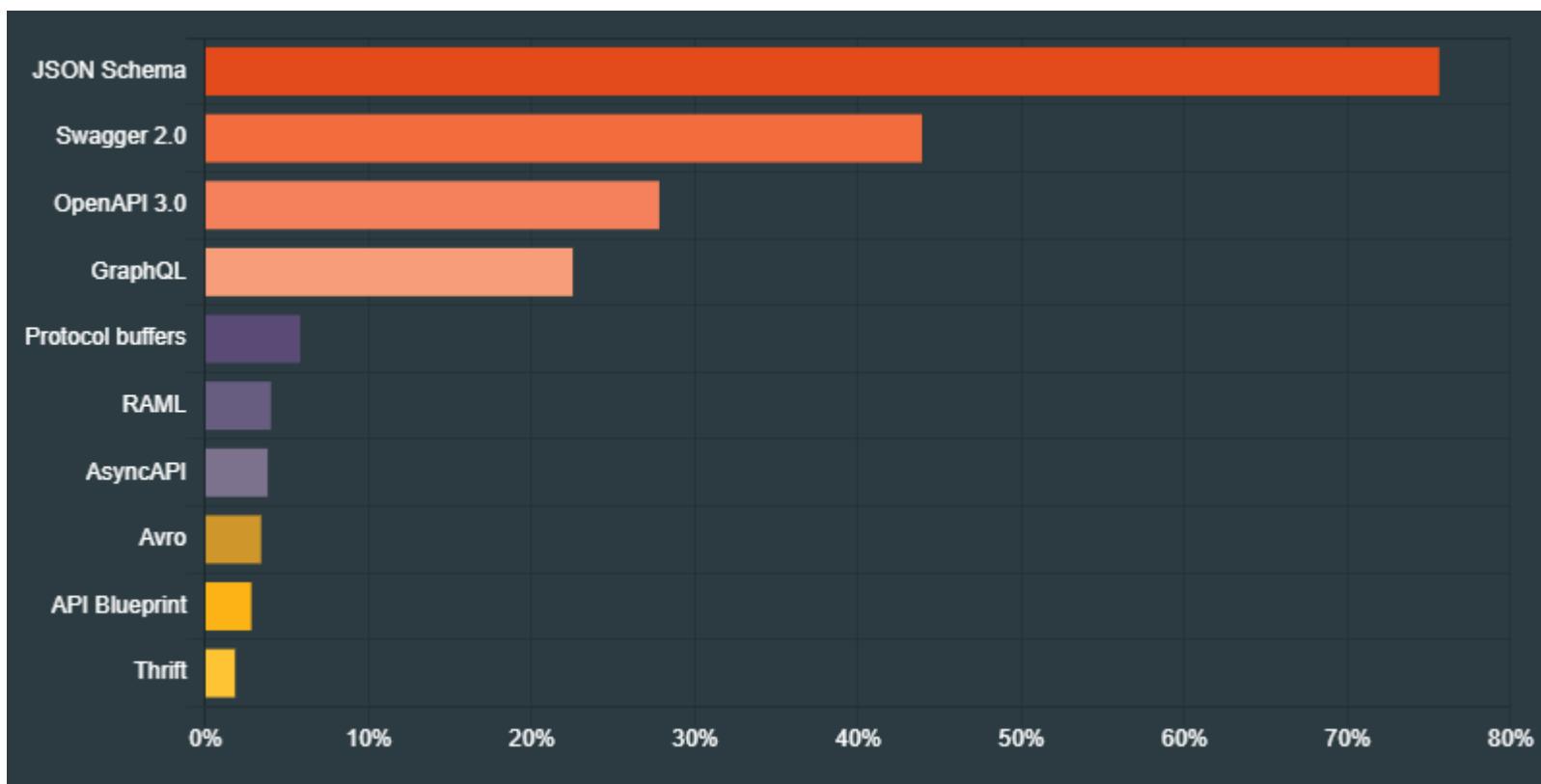
- As far as architectural styles for APIs are concerned
  - ✓ a sweeping majority of respondents (93.4%) were most familiar with REST
  - ✓ More than one-third mentioned webhooks and almost one-quarter mentioned WebSockets
    - ❖ which may point toward an event-driven future
  - ✓ Rounding out the top five are SOAP at 33.4% and GraphQL at 22.5%
- Respondents with 6+ years of API development experience were more likely to use REST than those with 0–5 years of experience.



# API Technologies(2)

## Specifications

- JSON Schema was by far the top specification in use in 2020, at 75.6%
  - ✓ Swagger 2.0 was next, followed by OpenAPI 3.0
  - ✓ GraphQL also had some significant reported usage at 22.5%
  - ✓ The rest fell in line significantly behind at 5.8% and less.
- Respondents with 6+ years of API development experience were more likely to use JSON Schema than those with 0–5 years of experience.

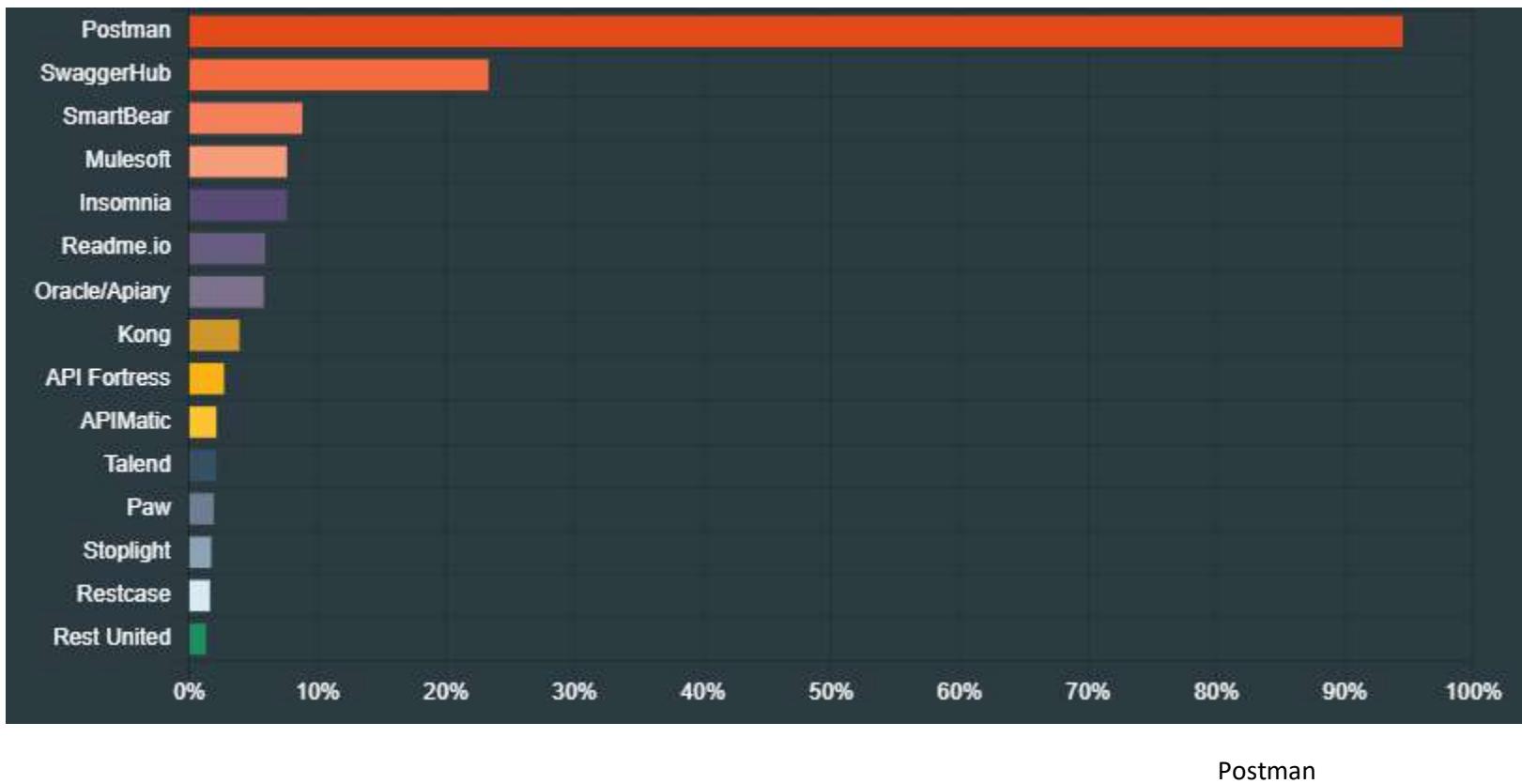


Postman

# Tooling for APIs and Development

## API tools

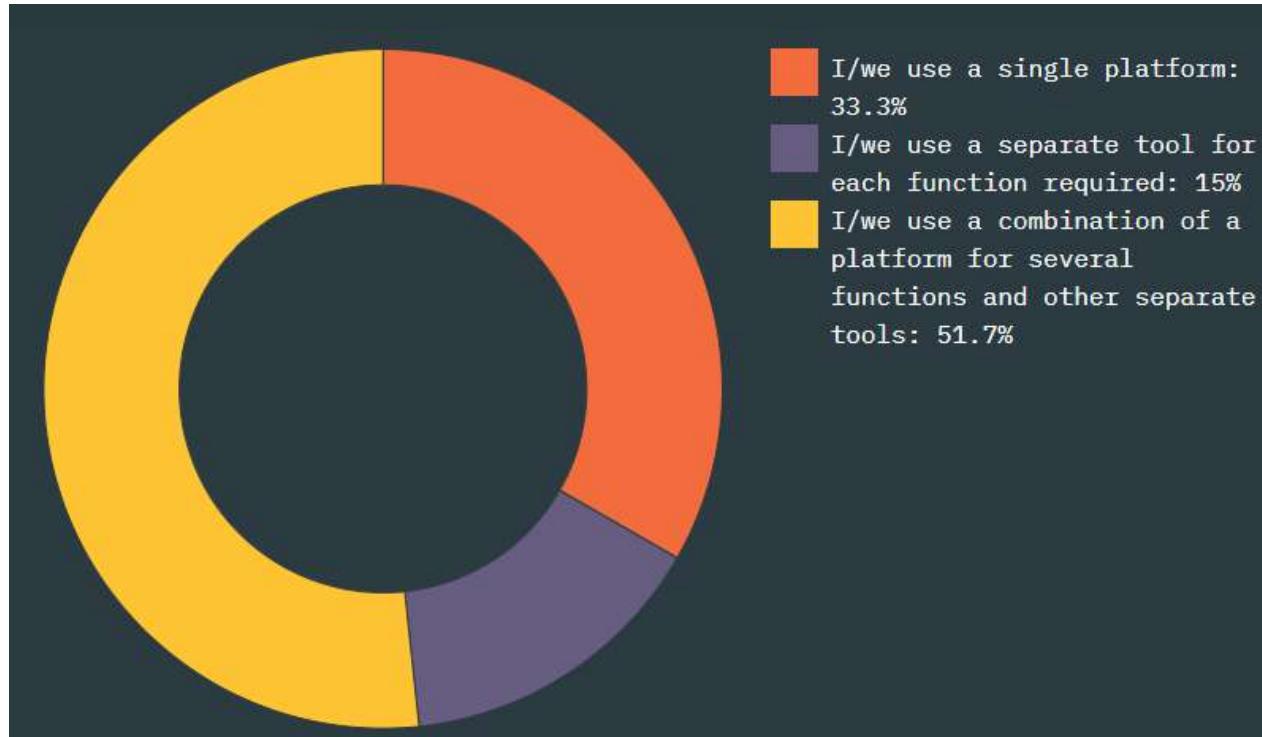
- Postman came out on top, garnering mentions from 94.5% of responses
  - ✓ SwaggerHub was the only other tool earning above single digits, with 23.3%
  - ✓ SmartBear (8.8%), Mulesoft (7.6%), and Insomnia (7.6%) rounded out the top five
- API-first leaders were more likely to mention Postman and SwaggerHub than others
  - ✓ Respondents with 6+ years of API development experience were more likely to mention Postman, SwaggerHub, and SmartBear than those with 0–5 years of experience.



# Tooling for APIs and Development (2)

## Platform vs separate tools

- Whether they prefer a single platform or a mix of tools to design, document, test, and deliver APIs
- A combination of both was the most popular answer, garnering more than 50% of the responses
- Respondents with 6+ years of API development experience were more likely to report using a combination than those with 0–5 years of experience

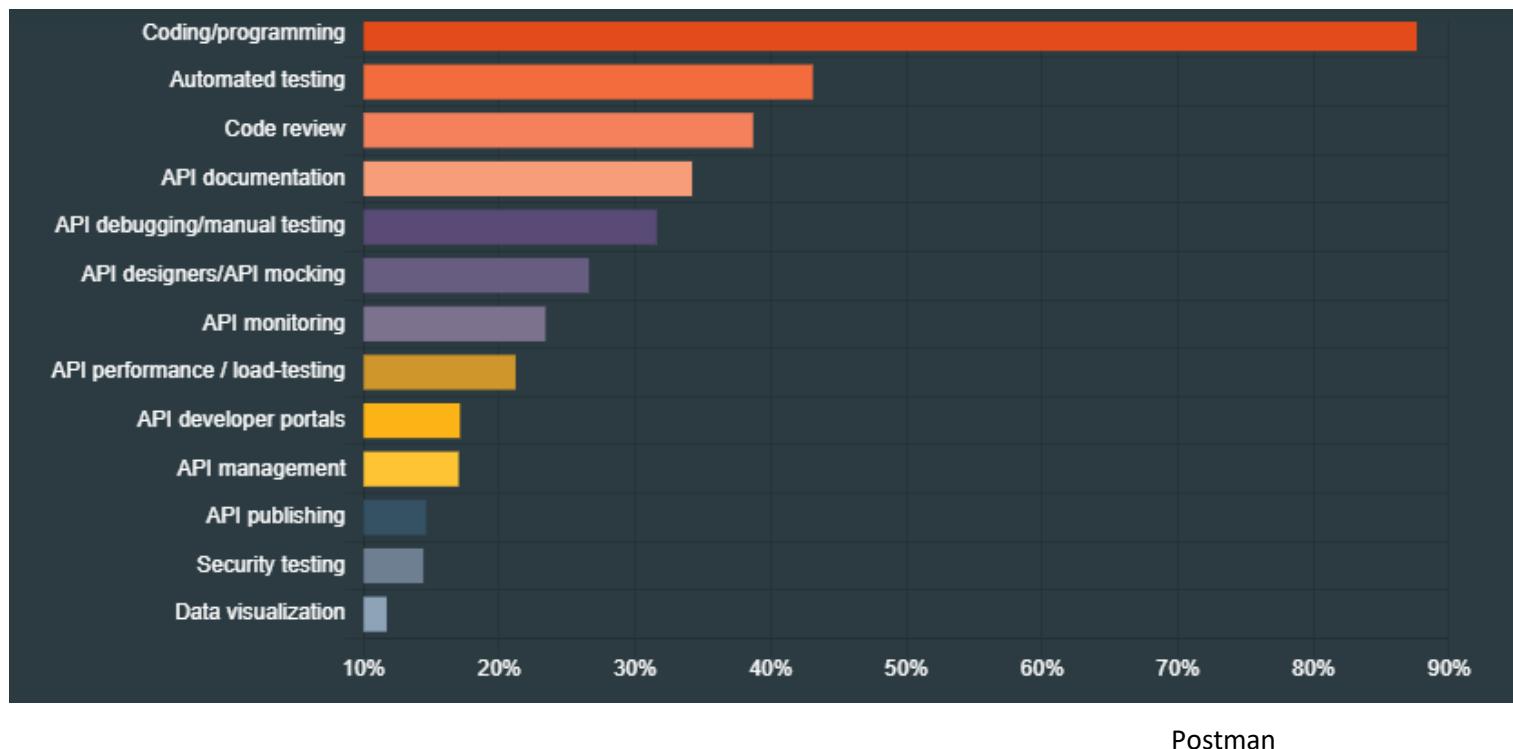


Postman

# Tooling for APIs and Development (3)

## Types of tooling for producing APIs

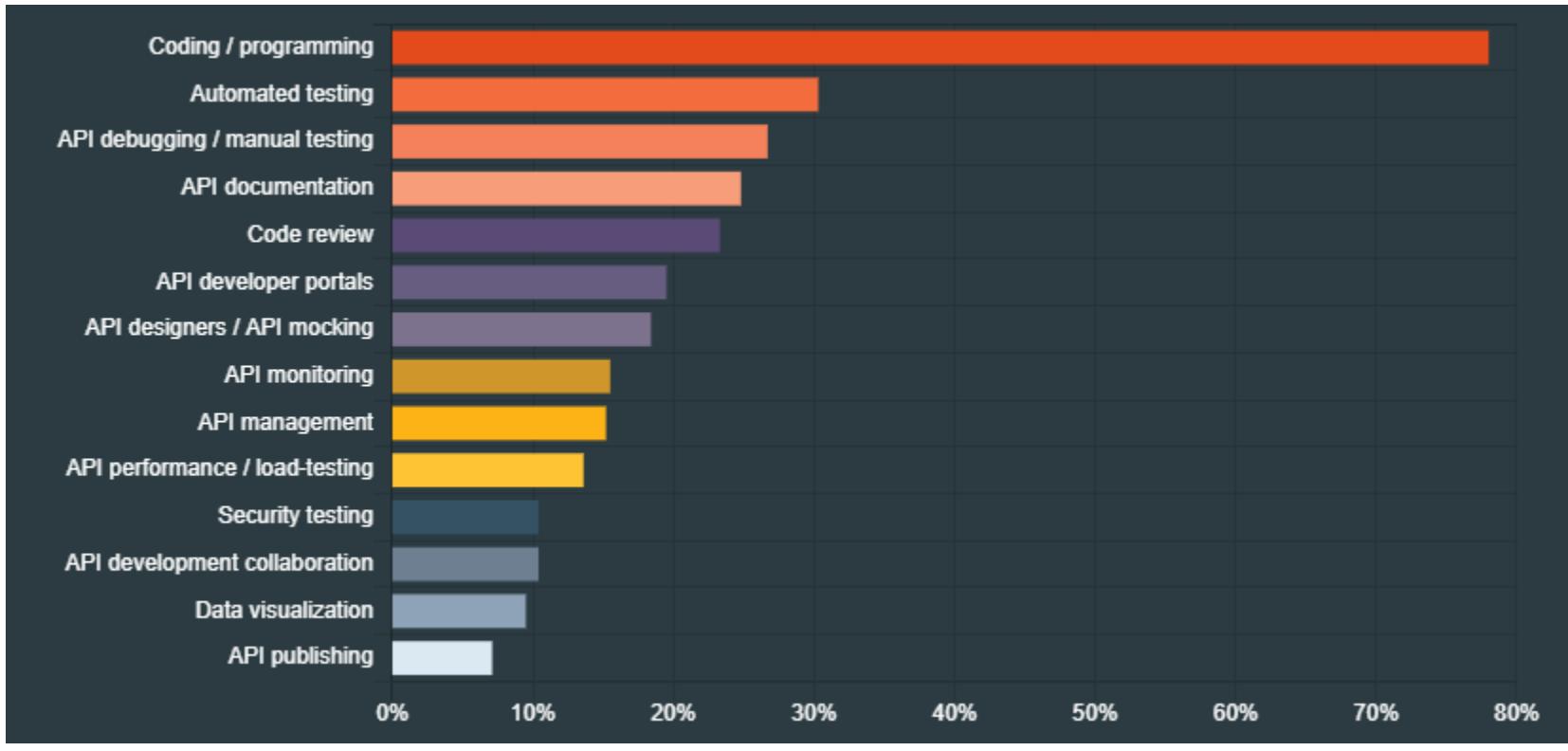
- Coding/programming tools (87.6%) received twice as many mentions as the next most popular, automated testing (43.1%)
- Code review tools were the third most popular at 38.7%
- The remainder of the tools mentioned were primarily API-specific tools
- Interestingly, respondents with 6+ years of API development experience were more likely to report using coding/programming tools to produce APIs than those with 0–5 years of experience



# Tooling for APIs and Development (4)

## Types of tooling for consuming APIs

- Mentions of coding/programming tools dwarfed all other API consumption tools, with 78%, compared to 30.3% for automated testing, the second-highest
- Respondents with 6+ years of API development experience were also more likely to report using coding/programming tools to consume APIs than those with 0–5 years of experience

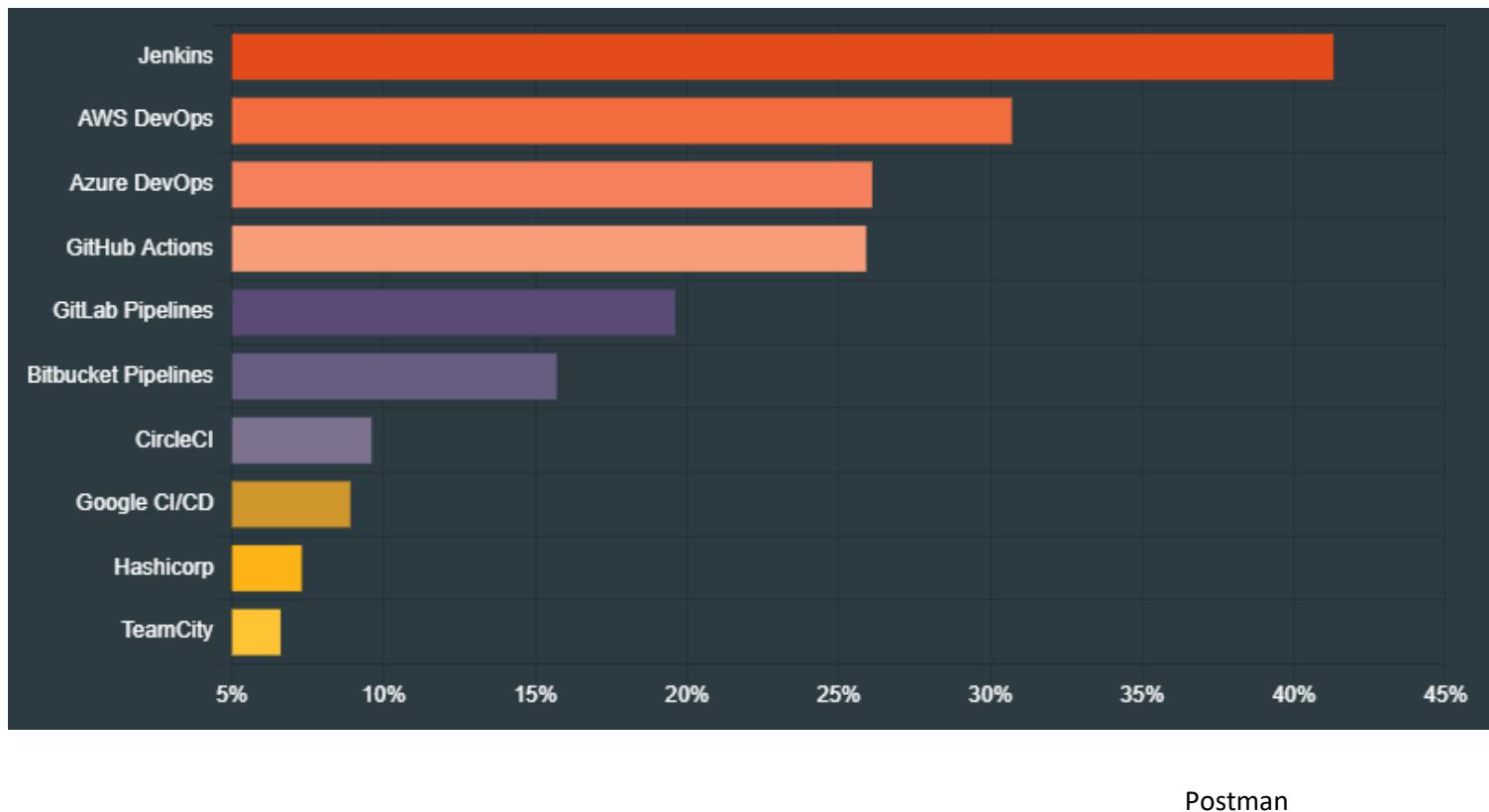


Postman

# Tooling for APIs and Development(5)

## DevOps tooling

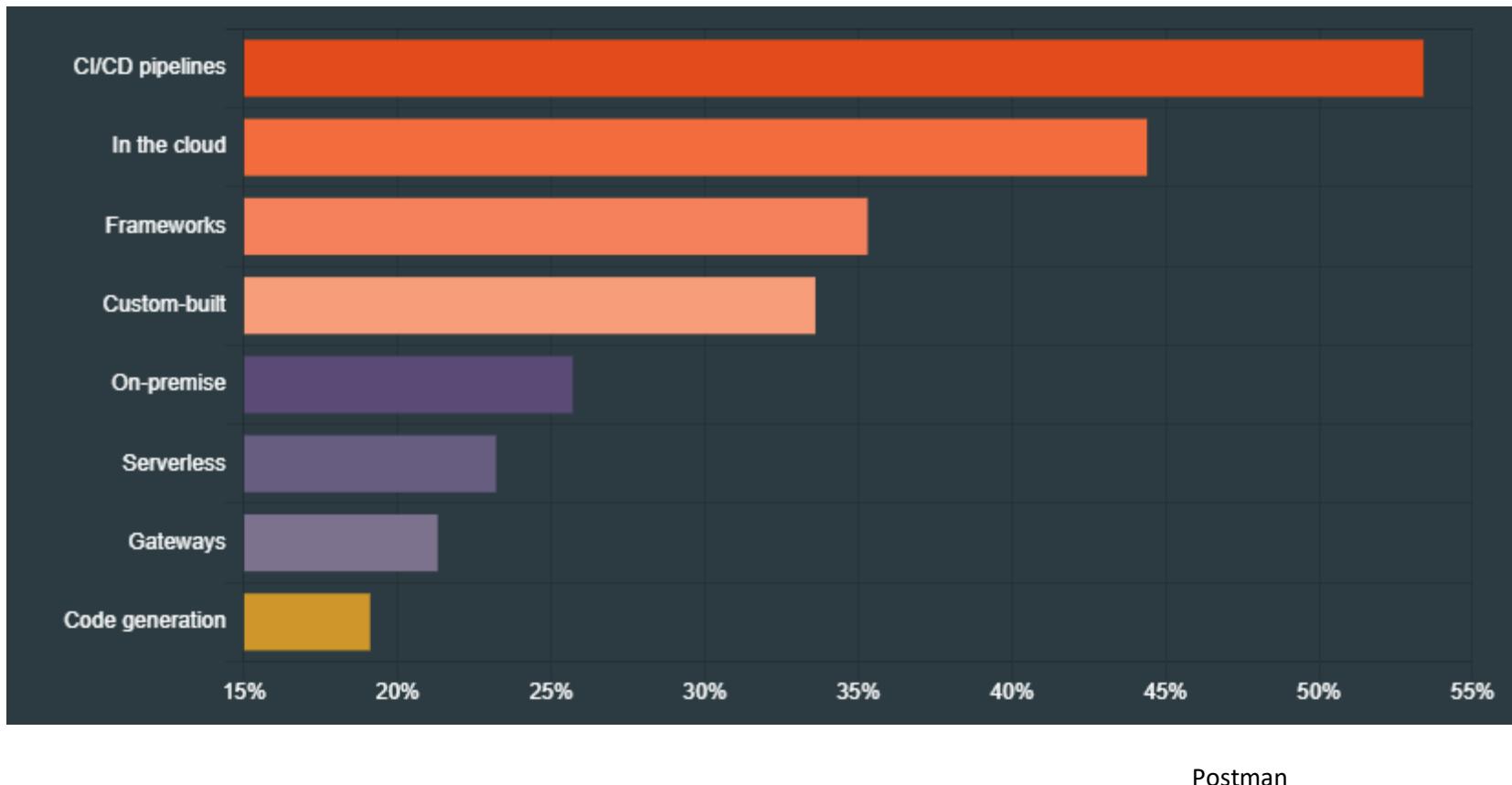
- DevOps practitioners rely on a number of tools, with Jenkins standing out and leading the way at 41.3%
- AWS DevOps (30.7%) and Azure DevOps (26.1%) registered second and third, respectively
- GitHub Actions, GitLab Pipelines, and Bitbucket Pipelines round out the remaining tools that received responses totaling 15% or higher.
- API-first leaders and respondents with 6+ years of API development experience were also more likely to mention Jenkins.



# Tooling for APIs and Development (6)

## Deploying APIs

- Respondents deploying APIs reported using a number of different approaches
- CI/CD pipelines were the most popular, at 53.4%, followed by deploying APIs in the cloud
- API-first leaders were more likely to use all of the approaches offered when deploying APIs than others
- Similarly, respondents with 6+ years of API development experience were more likely to use all of the approaches offered when deploying APIs than those with 0–5 years of experience



Reference

2020 State of the API Report

Report by Postman



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

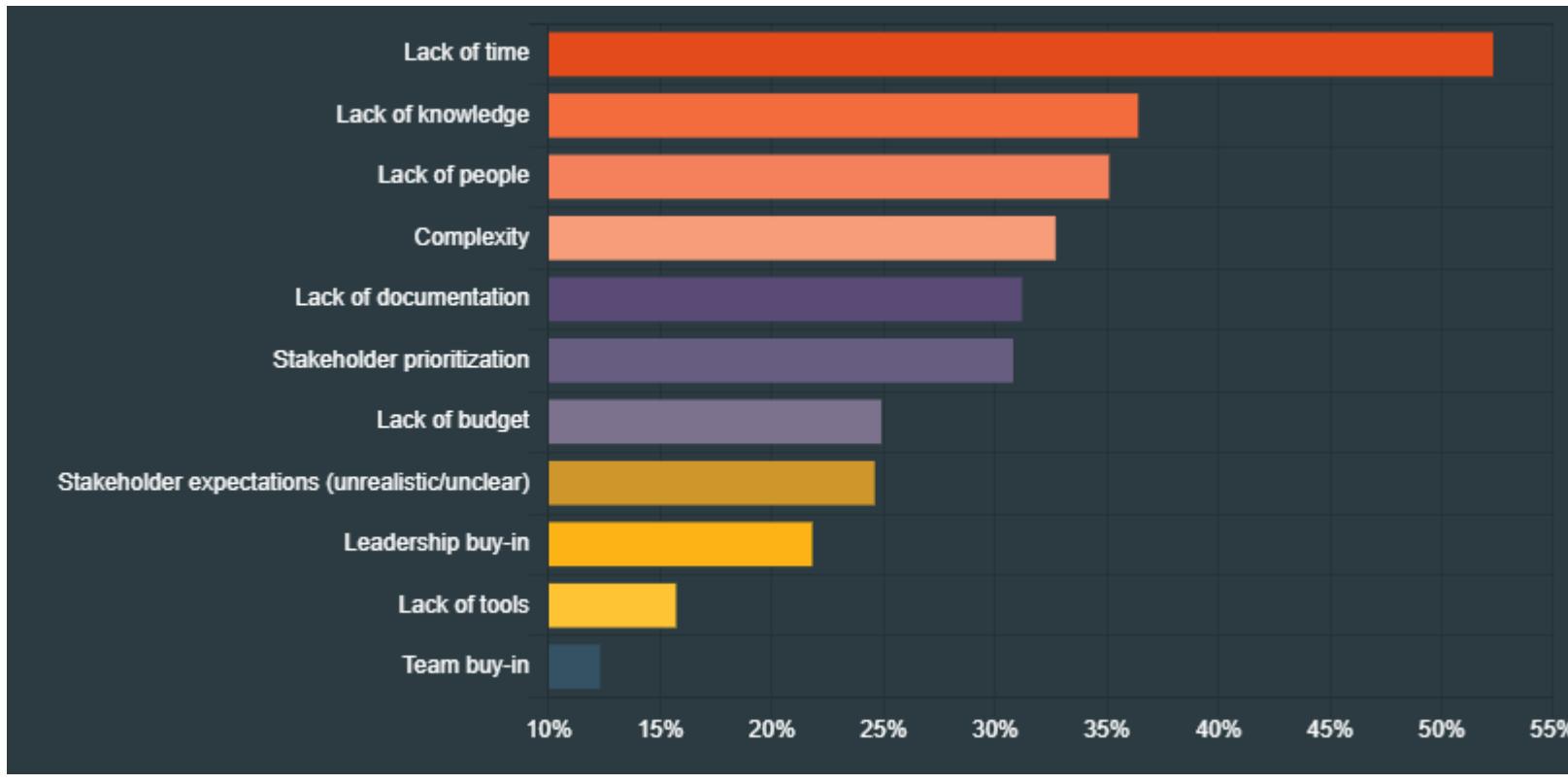
# API Challenges

Chandan Ravandur N

# Challenges of producing APIs and consuming APIs

## Obstacles to producing APIs

- When Developers reported about the obstacles to producing APIs,
  - ✓ lack of time is by far the leading obstacle, with 52.3% of respondents listing it
  - ✓ Lack of knowledge (36.4%) and people (35.1%) were the next highest.
- Looking at the top three mentions, API-first leaders were more likely to mention lack of time, lack of knowledge, and lack of people than others
  - ✓ Respondents with 6+ years of API development experience were less likely to mention lack of time, lack of knowledge, and lack of people than those with 0–5 years of experience.

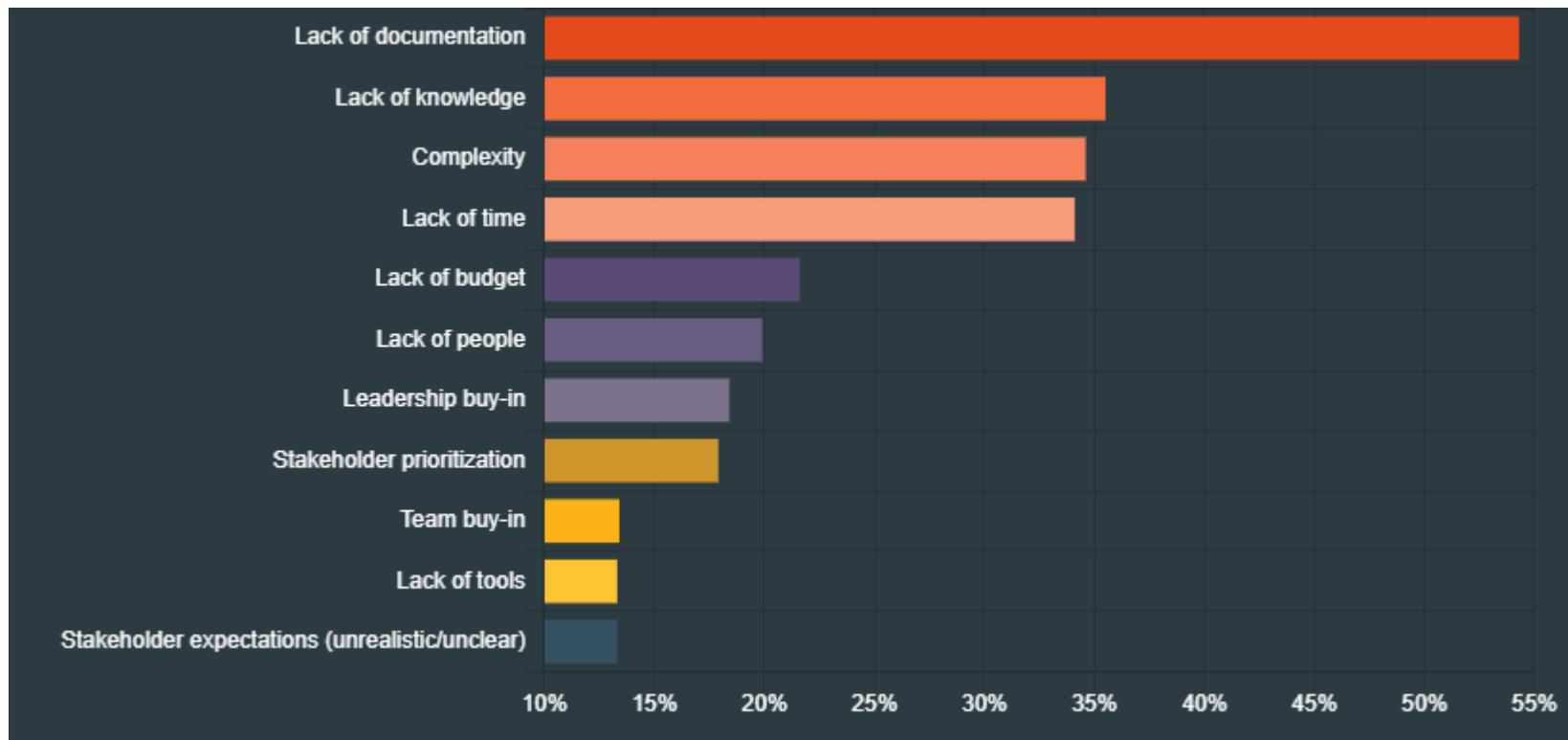


Postman

# Challenges of producing APIs and consuming APIs

## Obstacles to consuming APIs

- When asked about the biggest obstacle to consuming APIs,
  - ✓ lack of documentation clocked in the highest obstacle to consuming APIs (54.3%), by an extremely wide margin
  - ✓ Other top obstacles to consuming APIs are lack of knowledge, complexity, and lack of time, all cited by a little over one-third of respondents.
- Respondents with 6+ years of API development experience were more likely to mention lack of documentation than those with 0–5 years of experience.



Postman

# The most common challenges

- Lack of time
  - ✓ The number one obstacle for API producers, as well as an obstacle for many API consumers, is lack of time
  - ✓ Building and consuming APIs can be a time-intensive effort
    - ❖ depending on what you want APIs to do or how you use them
- Lack of documentation
  - ✓ The number one obstacle for API consumers is documentation
  - ✓ API documentation refers to the information API providers provide to API consumers to help them understand how to use APIs
- Lack of knowledge
  - ✓ Another obstacle for many producers and consumers is lack of knowledge
  - ✓ In some cases individuals working with APIs need deeper technical knowledge
  - ✓ while in other cases individuals are technically adept, but lack knowledge about the features of specific APIs

# The most common challenges (2)

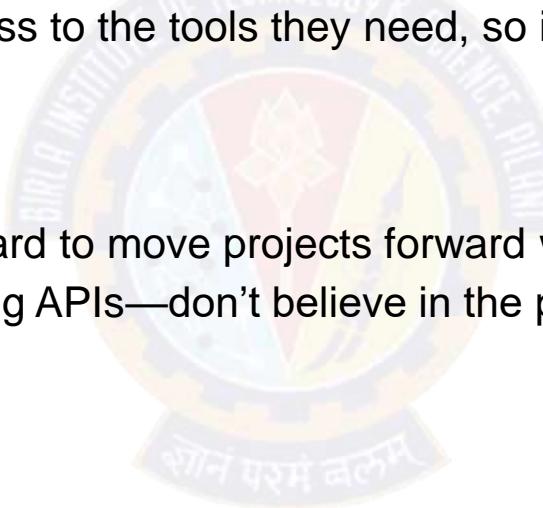
- Lack of people
  - ✓ Lack of people is closely related to lack of time and knowledge
  - ✓ Many teams find it challenging to produce and consume APIs with their existing team infrastructure
  - ✓ need additional people on their team to both expand their technical knowledge base as well as expand the amount of time individuals can spend on API efforts
- Complexity
  - ✓ While the concept of APIs is relatively simple, the implementation of APIs can be extremely complex
  - ✓ Many individuals who both consume and product APIs find complexity to be a major obstacle
- Stakeholder prioritization
  - ✓ Stakeholders, or individuals who have a “stake” in projects and efforts, are often the ones who control the resources, staff, and funds that are allocated to projects
  - ✓ If stakeholders do not prioritize API projects, that can be an obstacle to consumers and producers alike

# The most common challenges (3)

- Lack of budget
  - ✓ For most teams, it takes a budget to produce and consume APIs
  - ✓ Budget is spent on staff and training, sometimes on tooling, and some APIs require funds to use the API on a regular basis
- Stakeholder expectations (unrealistic/unclear)
  - ✓ Often stakeholders don't have technical backgrounds, and are not clear about what they're trying to achieve with APIs,
  - ✓ even worse, unrealistic about what it takes!
- Leadership buy-in
  - ✓ It can be really hard for teams to work on something that organization leadership does believe in, or "buy-in" to
  - ✓ When leaders don't buy-in to APIs, projects don't get prioritized and they fall by the wayside

# The most common challenges (4)

- Lack of tools
  - ✓ Very few developers open up a blank notepad and begin coding
  - ✓ There are a myriad of tools available to developers to help them code smarter, faster, and better
  - ✓ But, not every developer has access to the tools they need, so it becomes an obstacle
- Team buy-in
  - ✓ Like leadership buy-in, it can be hard to move projects forward with the teams themselves
  - ✓ the ones actually building and using APIs—don't believe in the project, making forward progress unlikely, or even impossible



Reference:  
2020 State of the API Report  
By Postman



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# API Paradigm

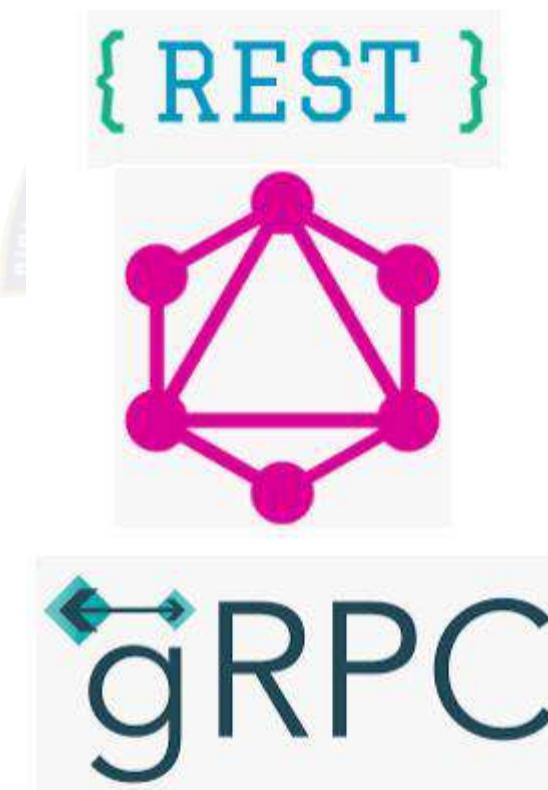
Chandan Ravandur N

# API Paradigm

- API paradigm defines the interface exposing backend data of a service to other applications
- Initially, organizations/developers do not always consider all the factors that will make an API successful
  - ✓ Afterwards becomes difficult to manage the feature addition in APIs
- Unfortunately, once developers starts using it, changing an API is difficult
- Selecting the right API paradigm is important
  - ✓ to save time, effort, and headaches
  - ✓ to leave room for new and exciting feature
  - ✓ to give some thought to protocols, patterns, and a few best practices
- Helps in designing an API that allows to make the changes want in the future

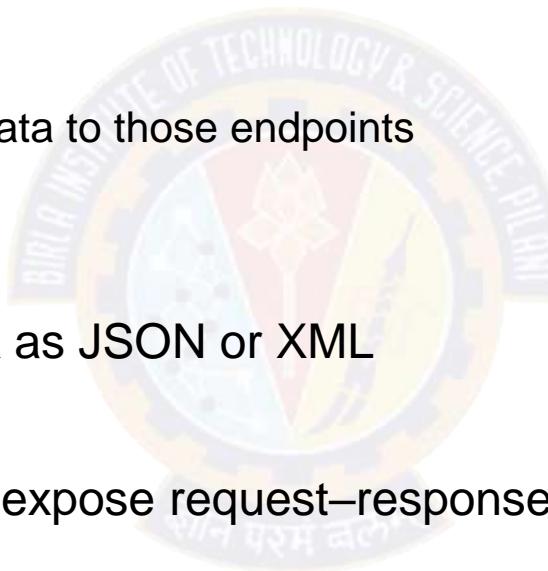
# API Paradigm (2)

- Over the years, multiple API paradigms have emerged such as
  - ✓ REST
  - ✓ RPC
  - ✓ GraphQL
  - ✓ WebHooks
  - ✓ and WebSockets
- Broadly can be classified as
  - ✓ Request-Response APIs
  - ✓ Event Driven APIs



# Request-Response APIs

- Typically expose an interface through a HTTP-based web server
- APIs define a set of endpoints
  - ✓ Clients make HTTP requests for data to those endpoints
  - ✓ Server returns responses
- The response is typically sent back as JSON or XML
- Three common paradigms used to expose request-response APIs:
  - ✓ REST
  - ✓ RPC
  - ✓ and GraphQL



# Event Driven APIs

- Apps which want to stay up to date with the changes in data on server side often end up polling the API
- With polling, apps constantly query API endpoints at a predetermined frequency and look for new data
  - ✓ If poll is done at a low frequency, apps will not have data about all the events happened since last poll
  - ✓ If poll is done at a high frequency, would lead to a huge waste of resources, as most API calls will not return any new data
- To share data about events in real time,
  - ✓ three common mechanisms are used :
    - ❖ WebHooks
    - ❖ WebSockets
    - ❖ and HTTP Streaming



# Thank You!

In our next session:



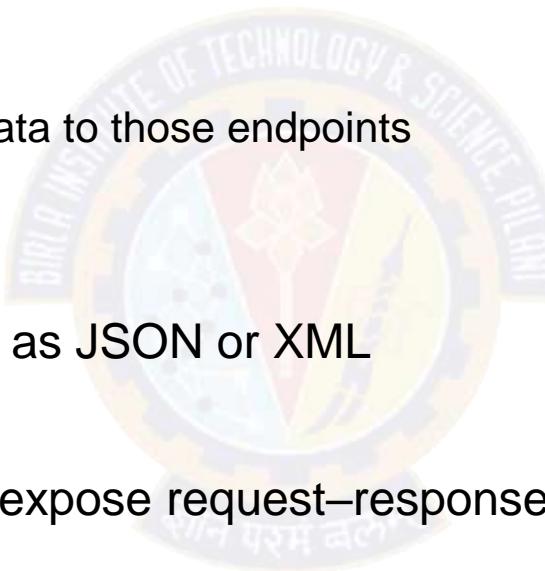
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Request - Response APIs

Chandan Ravandur N

# Request-Response APIs

- Typically expose an interface through a HTTP-based web server
- APIs define a set of endpoints
  - ✓ Clients make HTTP requests for data to those endpoints
  - ✓ Server returns responses
- The response is typically sent back as JSON or XML
- Three common paradigms used to expose request-response APIs:
  - ✓ REST
  - ✓ RPC
  - ✓ and GraphQL



# Representational State Transfer

## REST

- The most popular choice for API development lately
  - ✓ Used by providers like Google, Stripe, Twitter, and GitHub
- REST is about resources
  - ✓ A resource is an entity that can be identified, named, addressed, or handled on the web
  - ✓ REST APIs expose data as resources
  - ✓ Use standard HTTP methods to represent Create, Read, Update, and Delete (CRUD) transactions against these resources
- General rules REST APIs follow:
  - ✓ Resources are part of URLs, like /books
  - ✓ For each resource, generally two URLs are implemented
    - ❖ one for the collection, like /books
    - ❖ one for a specific element, like /books/B123
  - ✓ Nouns are used instead of verbs for resources
    - ❖ instead of /getBoooksInfo/B123, use /books/B123.
  - ✓ HTTP methods like GET, POST, UPDATE, and DELETE inform the server about the action to be performed
  - ✓ Standard HTTP response status codes are returned by the server indicating success or failure
    - ❖ codes in the 2XX range indicate success
    - ❖ 3XX codes indicate a resource has moved
    - ❖ codes in the 4XX range indicate a client-side error
    - ❖ codes in the 5XX range indicate server-side errors
  - ✓ REST APIs might return JSON or XML responses
    - ❖ Due to its simplicity and ease of use with JavaScript, JSON has become the standard for modern APIs

# Representational State Transfer (2)

## CRUD operations, HTTP verbs, and REST conventions

Operation	HTTP verb	URL: /books	URL: /books/123
Create	POST	Create new book	Not applicable
Read	GET	Get list of all books	Get one particular book with given ID
Update	PUT / PATCH	Batch update books	Update a particular book with given ID
Delete	DELETE	Delete all books	Delete a particular book with given ID

- GITHUB API for getting user profile data

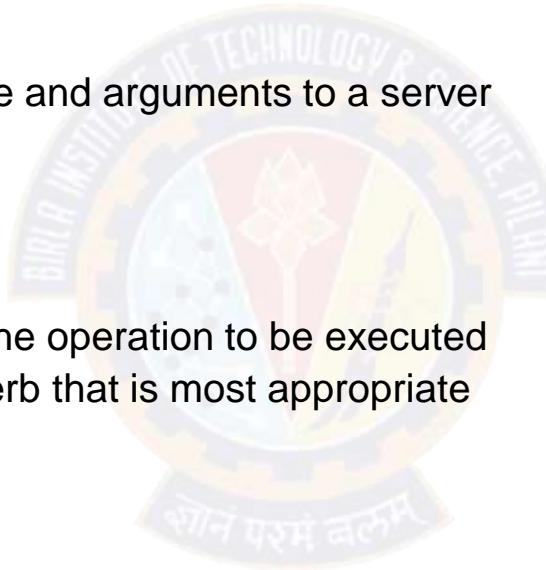
```
# GET /users/defunkt
$ curl https://api.github.com/users/defunkt

> {
>   "login": "defunkt",
>   "id": 2,
>   "url": "https://api.github.com/users/defunkt",
>   "html_url": "https://github.com/defunkt",
>   ...
> }
```

# Remote Procedure Call

## RPC

- One of the simplest API paradigms, in which a client executes a block of code on another server
- RPC is about actions
  - ✓ Clients typically pass a method name and arguments to a server
  - ✓ Receive back JSON or XML
- Simple rules:
  - ✓ The endpoints contain the name of the operation to be executed
  - ✓ API calls are made with the HTTP verb that is most appropriate
  - ✓ GET for read-only requests
  - ✓ POST for others
- Works great for APIs that
  - ✓ expose a variety of actions which have complications that can be encapsulated with CRUD
  - ✓ for which there are side effects unrelated to the “resource” at hand
  - ✓ accommodate complicated resource models or actions upon multiple types of resources



# Remote Procedure Call (2)

## Using the Slack Web API

- The Slack Web API is an interface for querying information from and enacting change in a Slack workspace
- The Web API is a collection of HTTP RPC-style methods
- All with URLs in the form [https://slack.com/api/METHOD\\_FAMILY.method](https://slack.com/api/METHOD_FAMILY.method)

### admin.analytics

Method	Description
<a href="#">admin.analytics.getFile</a>	Retrieve analytics data for a given date, presented as a compressed JSON file

### admin.apps

Method	Description
<a href="#">admin.apps.approve</a>	Approve an app for installation on a workspace.
<a href="#">admin.apps.clearResolution</a>	Clear an app resolution
<a href="#">admin.apps.restrict</a>	Restrict an app for installation on a workspace.

### admin.apps.approved

Method	Description
<a href="#">admin.apps.approved.list</a>	List approved apps for an org or workspace.

# Remote Procedure Call (3)

## Protocols

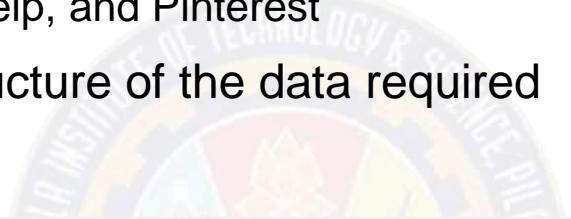
- RPC-style APIs are not exclusive to HTTP
- Other high performance protocols that are available for RPC-style API
  - ✓ Apache Thrift
  - ✓ gRPC
- JSON options available for gRPC
  - ✓ Both Thrift and gRPC requests are serialized
  - ✓ Thrift and gRPC also have built-in mechanisms for editing the data structures



Apache Thrift

# GraphQL

- GraphQL is a query language for APIs that has gained significant momentum recently
  - ✓ Developed internally by Facebook, publicly released in 2015
  - ✓ Adopted by API providers like GitHub, Yelp, and Pinterest
- GraphQL allows clients to define the structure of the data required
- Server returns exactly that structure



```
{  
  hero {  
    name  
    # Queries can have comments!  
    friends {  
      name  
    }  
  }  
}
```

```
{  
  "data": {  
    "hero": {  
      "name": "R2-D2",  
      "friends": [  
        {  
          "name": "Luke Skywalker"  
        },  
        {  
          "name": "Han Solo"  
        }  
      ]  
    }  
  }  
}
```

- GraphQL APIs need only a single URL endpoint
  - ✓ Do not need different HTTP verbs to describe the operation
  - ✓ Instead, indicate in the JSON body whether you're performing a query or a mutation

# GraphQL(2)

## Github Example

Untitled Query 1

GraphQL Endpoint <https://api.github.com/graphql> Method POST Edit HTTP Headers

GraphiQL Prettify Docs

```
query {
  organization(login: "nasa") {
    name
    url
  }
}
```

```
{
  "data": {
    "organization": {
      "name": "NASA",
      "url": "https://github.com/nasa"
    }
  }
}
```

- Fetch Rails Repository

```
query {
  organization(login: "rails") {
    name
    url
    repository(name: "rails") {
      name
    }
  }
}
```

```
{
  "data": {
    "organization": {
      "name": "Ruby on Rails",
      "url": "https://github.com/rails",
      "repository": {
        "name": "rails"
      }
    }
  }
}
```

# Comparison of request-response API paradigms

	REST	RPC	GraphQL
What?	Exposes data as resources Uses standard HTTP methods to represent CRUD operations	Exposes action-based API methods—clients pass method name and arguments	Exposes action-based API methods—clients pass method name and arguments
Providers	Stripe, GitHub, Twitter, Google	Slack, Flickr	Slack, Flickr
Example usage	GET /books/<id>	GET /books.get?id=<id>	query (\$id: String!) { book() { name author } }
HTTP Verbs	GET, POST, PUT, PATCH, DELETE	GET, POST	GET, POST

# Comparison of request–response API paradigms (2)

	REST	RPC	GraphQL
Pros	<ul style="list-style-type: none"><li>• Standard method name, arguments format, and status codes</li><li>• Utilizes HTTP features</li><li>• Easy to maintain</li></ul>	<ul style="list-style-type: none"><li>• Easy to understand</li><li>• Lightweight payloads</li><li>• High performance</li></ul>	<ul style="list-style-type: none"><li>• Saves multiple round trips</li><li>• Avoids versioning</li><li>• Smaller payload size</li><li>• Strongly typed</li><li>• Built-in introspection</li></ul>
Cons	<ul style="list-style-type: none"><li>• Big payloads</li><li>• Multiple HTTP round trips</li></ul>	<ul style="list-style-type: none"><li>• Discovery is difficult</li><li>• Limited standardization</li><li>• Can lead to function explosion</li></ul>	<ul style="list-style-type: none"><li>• Requires additional query parsing</li><li>• Backend performance optimization is difficult</li><li>• Too complicated for a simple API</li></ul>
When to use?	For APIs doing CRUD like operations	For APIs exposing several actions	When you need querying flexibility; great for providing querying flexibility and maintaining consistency

Reference:

Designing Web APIs – Building APIs that developers love

Jin, Sahni and Shevat



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Event Driven APIs

Chandan Ravandur N

# Event Driven APIs

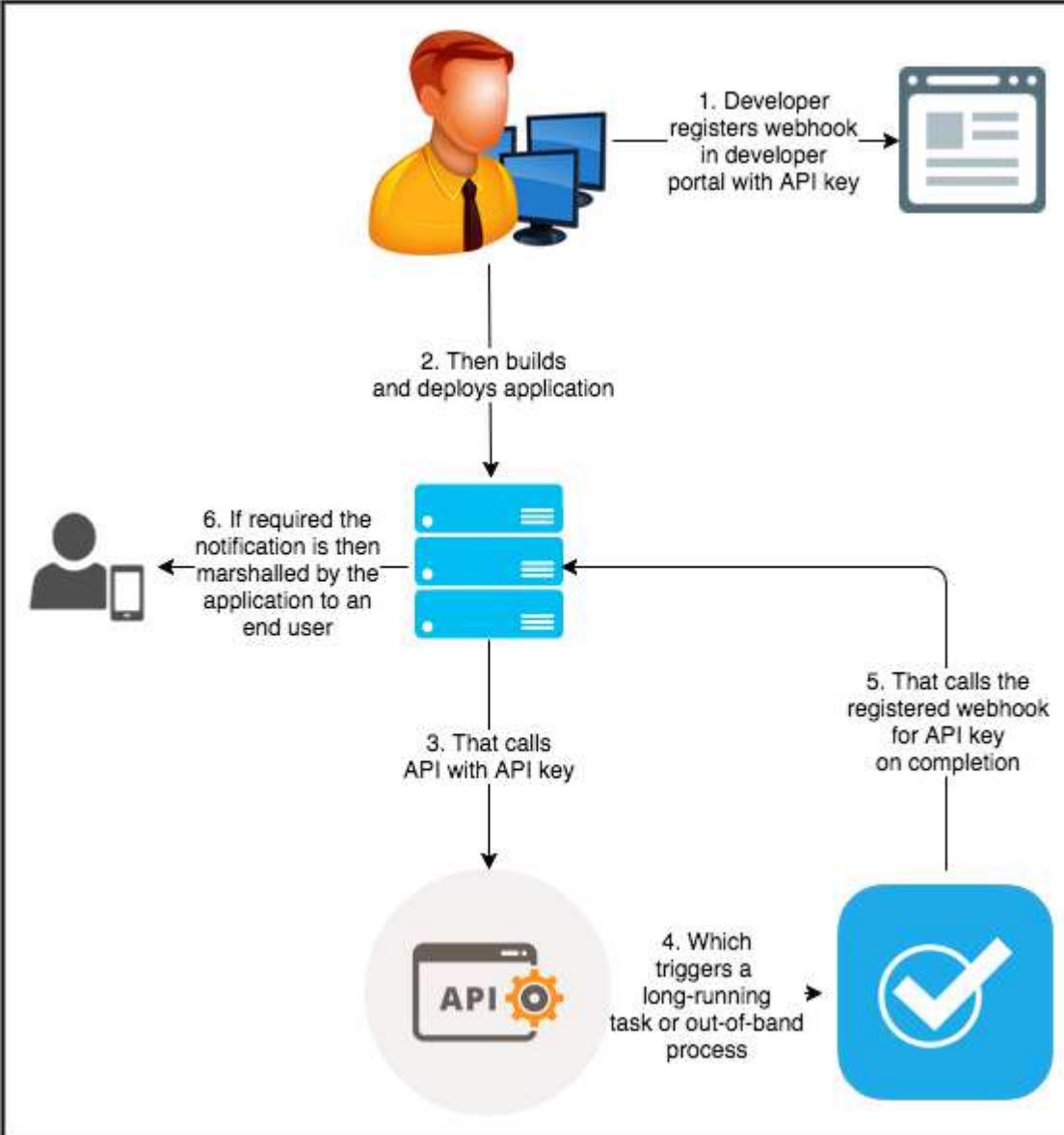
- Apps which want to stay up to date with the changes in data on server side often end up polling the API
- With polling, apps constantly query API endpoints at a predetermined frequency and look for new data
  - ✓ If poll is done at a low frequency, apps will not have data about all the events happened since last poll
  - ✓ If poll is done at a high frequency, would lead to a huge waste of resources, as most API calls will not return any new data
- To share data about events in real time,
  - ✓ three common mechanisms are used :
    - ❖ WebHooks
    - ❖ WebSockets
    - ❖ and HTTP Streaming

# WebHooks

- With WebHooks, app can receive updates in real time
  - ✓ Just a URL that accepts an HTTP POST (or GET, PUT, or DELETE)
  - ✓ An API provider implementing WebHooks will simply POST a message to the configured URL when something happens
- Several API providers, like Slack, Stripe, GitHub, and Zapier, support WebHooks
- For instance,
  - ✓ if you want to keep track of “channels” in a Slack team with Slack’s Web API, you might need to continuously poll the API for new channels
  - ✓ by configuring a WebHook, you can simply receive a notification whenever a new channel is created

# WebHooks (2)

## Working

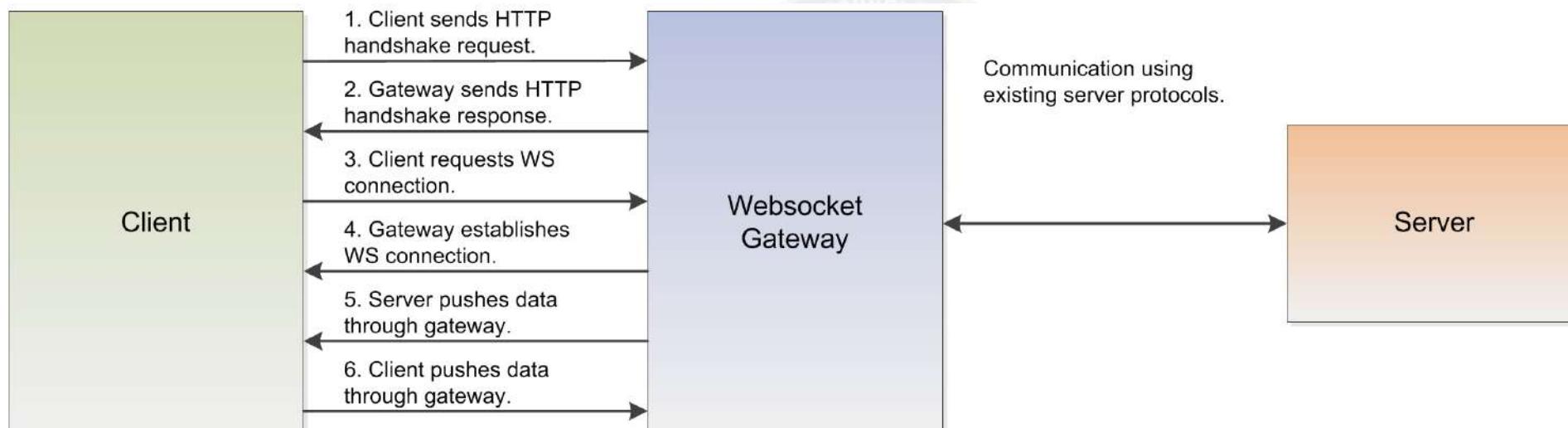


# WebSockets

- WebSocket is a protocol used to establish a two-way streaming communication channel over a single Transport Control Protocol (TCP)
  - ✓ Protocol is generally used between a web client (e.g., a browser) and a server
  - ✓ sometimes used for server-to-server communication
- Supported by major browsers and often used by real-time applications
  - ✓ Slack uses WebSockets to send all kinds of events happening in a workspace to Slack's clients
  - ✓ Trello uses WebSockets to push changes made by other people down from servers to browsers listening on the appropriate channels
  - ✓ Blockchain uses its WebSocket API to send real-time notifications about new transactions and blocks
- Enable full-duplex communication between server and client
  - ✓ Great for fast, live streaming data and long-lived connections
  - ✓ be wary if planning to make these available on mobile devices or in regions where connectivity can be spotty
  - ✓ Clients are supposed to keep the connection alive

# WebSockets (2)

## How Does WebSockets Work?



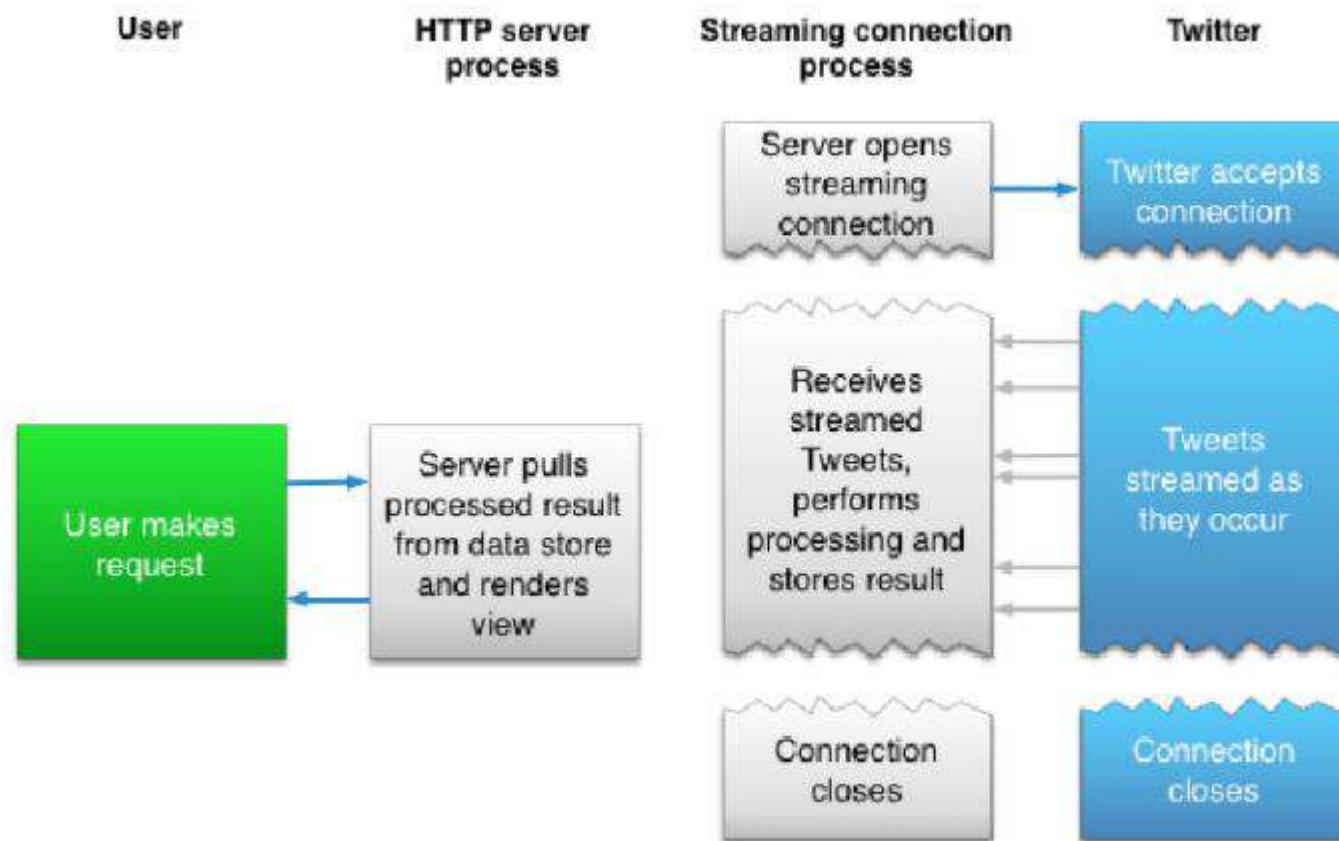
programmableweb

# HTTP Streaming

- With the HTTP request–response APIs
  - ✓ clients send an HTTP request and the server returns an HTTP response of a finite length
- With HTTP Streaming,
  - ✓ possible to make the length of this response indefinite
  - ✓ the server can continue to push new data in a single long-lived connection opened by a client
- Two options:
  - ✓ to set the Transfer-Encoding header to chunked - indicates to clients that data will be arriving in chunks of newline-delimited strings
  - ✓ to stream data via server-sent events (SSE)- great for clients consuming these events in a browser
- Twitter utilizes the HTTP Streaming protocol to deliver data through a single connection opened between an app and Twitter's streaming API
  - ✓ Developers don't need to poll the Twitter API continuously for new tweets
  - ✓ Twitter's Streaming API can push new tweets over a single HTTP connection instead of a custom protocol

# HTTP Streaming (2)

## Twitter Streaming API



Source : researchgate

# Comparison of event-driven APIs

	WebHooks	WebSockets	HTTP Streaming
What?	Event notification via HTTP callback	Two-way streaming connection over TCP	Long-lived connection over HTTP
Example services	Slack, Stripe, GitHub, Zapier, Google	Slack, Trello, Blockchain	Twitter, Facebook
Pros	<ul style="list-style-type: none"><li>Easy server-to-server communication</li><li>Uses HTTP protocol</li></ul>	<ul style="list-style-type: none"><li>Two-way streaming communication</li><li>Native browser support</li><li>Can bypass firewalls</li></ul>	<ul style="list-style-type: none"><li>Can stream over simple HTTP</li><li>Native browser support</li><li>Can bypass firewalls</li></ul>
Cons	<ul style="list-style-type: none"><li>Do not work across firewalls or in browsers</li><li>Handling failures, retries, security is hard</li></ul>	<ul style="list-style-type: none"><li>Need to maintain a persistent connection</li><li>Not HTTP</li></ul>	<ul style="list-style-type: none"><li>Bidirectional communication is difficult</li><li>Reconnections required to receive different events</li></ul>
When to use	<ul style="list-style-type: none"><li>To trigger the server to serve real-time events</li></ul>	<ul style="list-style-type: none"><li>For two-way, real-time communication between browsers and servers</li></ul>	<ul style="list-style-type: none"><li>For one-way communication over simple HTTP</li></ul>

Reference:

Designing Web APIs – Building APIs that developers love

Jin, Sahni and Shevat



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Private-Partner-Public APIs

Chandan Ravandur N

# API Mindset

- Application Programming Interfaces (APIs) are at the center of a digital transformation
  - ✓ enabling businesses to do more with less, to reach new markets, and to speed up product and service development time
- Launching a successful API starts with the right mindset
- Should you develop your API
  - ✓ for internal use
  - ✓ for partners,
  - ✓ for the general public
  - ✓ or a mix of these?
- Several questions :
  - ✓ Should you plan to be closed at first
  - ✓ but then open up over time?
  - ✓ What are the consequences of this strategy?
  - ✓ What aspects of your API program should you prioritize at what points?



# API adoption patterns

## three key API adoption patterns

- Private APIs
  - ✓ AKA internal or enterprise APIs
- Partner APIs that facilitate integration between a business and their partners
- Public or open APIs



Nordic APIs

# Private API Models

## Two types of Private APIs

- The first type are those that are set up internally to link two datasets or processes together
  - ✓ Paired with business logic in a very tight manner
  - ✓ Often set in place by a top-down management process, and their use is enforced behind the scenes by an IT department
  - ✓ Used to expose a business' data to its own mobile applications
  - ✓ Lets a business get out from behind the desktop (or laptop) and provide access to the data from any remote device
- The second type of Private APIs are those akin to Web services, provided as a part of a Service-Oriented Architecture(SOA)
  - ✓ Provides an integration component that is made available to anyone within the business
  - ✓ Encourage and facilitate reuse
  - ✓ Often be a bottom-up or horizontally-instituted API strategy
  - ✓ Created to encourage other departments to share data or processes, and to facilitate team collaboration across the business
  - ✓ Forms a sort of private library of APIs

# Partner API Approaches

- Companies are also beginning to turn to partner-based APIs as a way of collaborating effectively
  - ✓ to leverage business relationships in a distributed environment
- Allows partners to utilize the customer relationships of one business with another
- As a result, the Partner APIs enables broader adoption of the company's Public APIs
  - ✓ This type of multifaceted API strategy is indicative of one developed by an organization that has progressed beyond one of API provider to that of an API platform.
- For example,
  - ✓ Increasingly banks are looking to differentiate their offering through “vertical integration”
  - ✓ This is where banks will use API's to integrate 3rd party products and services to augment what they provide
  - ✓ This may be purely financial providers or others that help the bank target very specific customer journeys or segments



Source :economicstimes

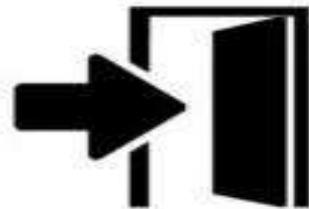
# Public API Releases

- Public, or open APIs, can rapidly grow a business' market share and customer base
- Companies are finding they can monetize their API as a new commercial product or service
  - ✓ with developer customers willing to pay for access to the stream of data or functionality opened up via an API
- Are publicly available to developers and other users with minimal restriction
  - ✓ May require registration, use of an API Key or OAuth, or maybe completely open
  - ✓ They focus on external users, to access data or services
- For example,
  - ✓ IMDb API helps to query for all information about films, actors, characters, etc... as on official websites
  - ✓ Access movie and TV information
  - ✓ Get Title, Year, Metascore Rating, IMDB rating, Release date, Runtime, Genre, Directors, Writers, Actors, Plot, Awards, Posters,etc...



# Summarized

THERE ARE THREE MAIN TYPES OF APIs



## Open

Open APIs allow companies to **publicly expose information and functionalities** of one or various systems and applications to **third parties** that do not necessarily have a business relationship with them.

### Advantages:

- Delegated R&D
- Increased reach, traffic
- New revenue stream



## Partner

Partner APIs are used to **facilitate** communication and integration of software **between a company and its business partners**.

### Advantages:

- Value-added service
- Up sell
- Must have for business partners



## Private

Private APIs are used **internally** to facilitate the **integration** of different applications and systems used by a company.

### Advantages:

- Rationalized infrastructure
- Reduced costs
- Increase flexibility: "real-time" business
- Improved internal operations

Reference:

Developing the API Mindset

Preparing Your Business for Private, Partner, and Public APIs

By Nordic APIs



# Thank You!

In our next session:



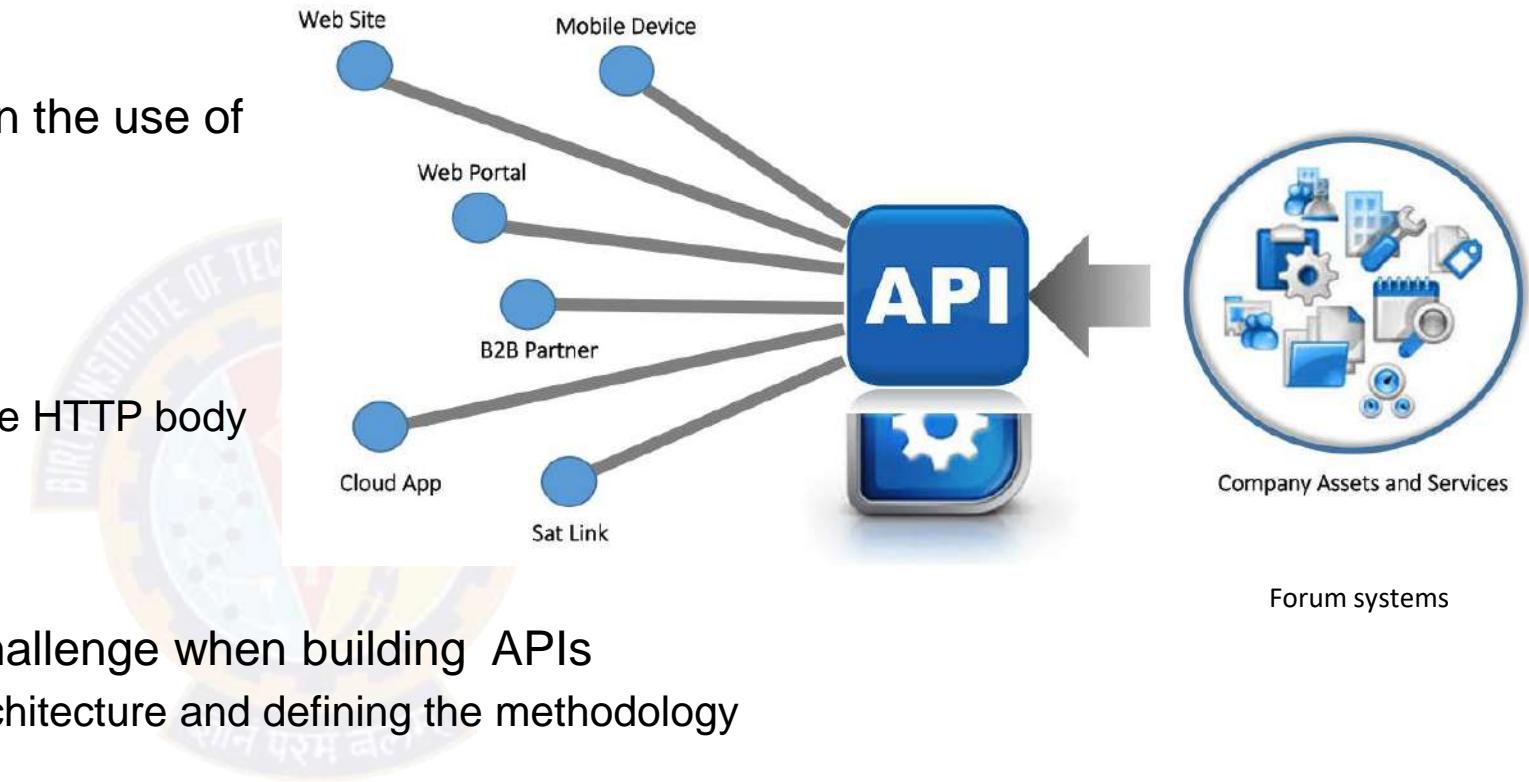
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# API Platform Architecture

Chandan Ravandur N

# What is API Architecture?

- Most of the times API design focus is on the use of
  - ✓ HTTP methods
  - ✓ URL design
  - ✓ HTTP status codes
  - ✓ HTTP headers
  - ✓ and the structure of the resources in the HTTP body



- In reality, this is actually the smallest challenge when building APIs
  - ✓ The real challenge is finding an API architecture and defining the methodology
- API architecture is way more than the correct application of REST principles
  - ✓ API Architecture spans the bigger picture of APIs and can be seen from several perspectives

# Different perspectives

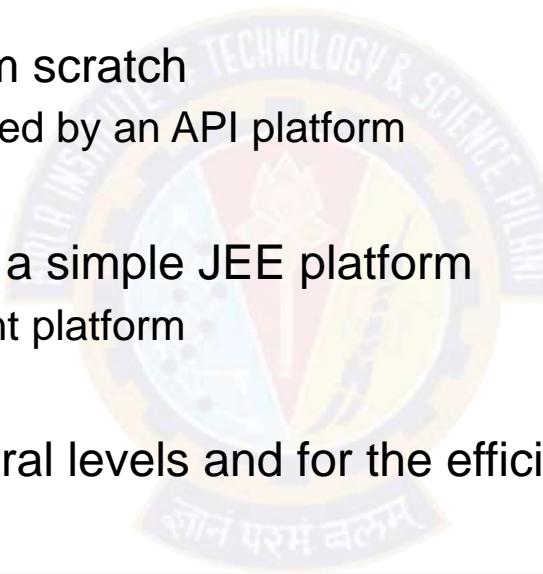
## 4 types

- API solution architecture
  - ✓ API architecture may refer to the architecture of the complete solution
  - ✓ consisting not only of the API itself, but also of an API client such as a mobile app and several other components
  - ✓ API solution architecture explains the components and their relations within the software solution
- API platform
  - ✓ API architecture may refer to the technical architecture of the API platform
  - ✓ When building, running and exposing not only one, but several APIs, it becomes clear that certain building blocks of the API, runtime functionality and management functionality for the API need to be used over and over again
  - ✓ An API platform provides an infrastructure for developing, running and managing APIs
- API portfolio architecture
  - ✓ API architecture may refer to the architecture of the API portfolio
  - ✓ The API portfolio contains all APIs of the enterprise and needs to be managed like its product
  - ✓ API portfolio architecture analyzes the functionality of the API and organizes, manages and reuses the APIs.
- API proxy architecture
  - ✓ API architecture may refer to the design decisions for a particular API proxy
  - ✓ To document the design decisions, API description languages are used

# API Platform Architecture

## Defined

- API platforms are used by API providers to realize new APIs efficiently
- New APIs are typically not built from scratch
  - ✓ but from the building blocks provided by an API platform
- This platform can be anything from a simple JEE platform
  - ✓ to a fully-featured API management platform
- A platform allows for reuse on several levels and for the efficient development of APIs



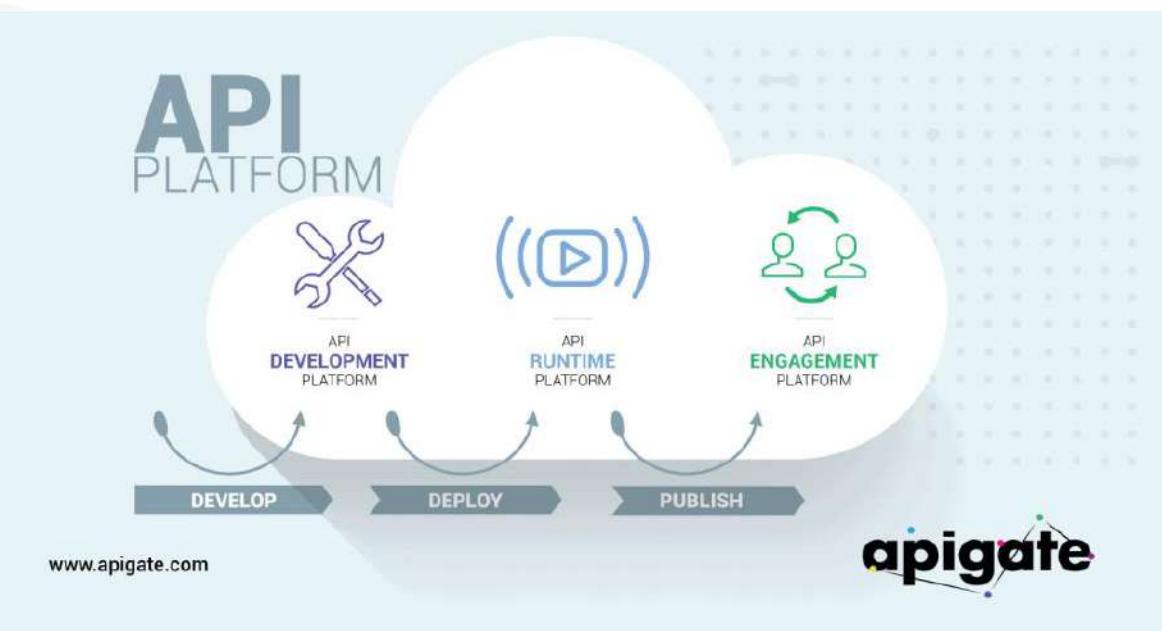
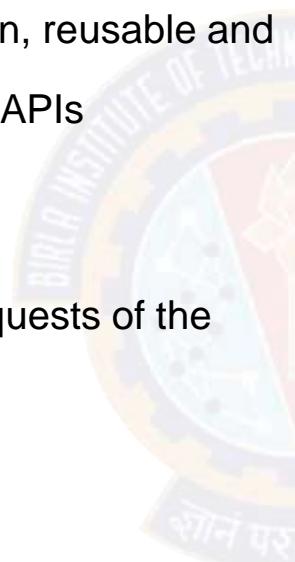
Api architecture



# An API platform

## Components

- API Development Platform
  - ✓ enables API providers to develop APIs quickly and with high quality
  - ✓ offers API building blocks, which are proven, reusable and configurable
  - ✓ offers tools for development and design of APIs
- API Runtime Platform
  - ✓ primarily executes the APIs
  - ✓ serves API responses for incoming API requests of the consumers
  - ✓ with favorable non-functional properties
    - ✓ such as high throughput and low latency
- API Engagement Platform
  - ✓ allows API providers to manage their interaction with API consumers
  - ✓ offers API documentation, credentials and rate plans for API consumers
  - ✓ For API providers it offers product management and configuration capabilities



# API Development Platform

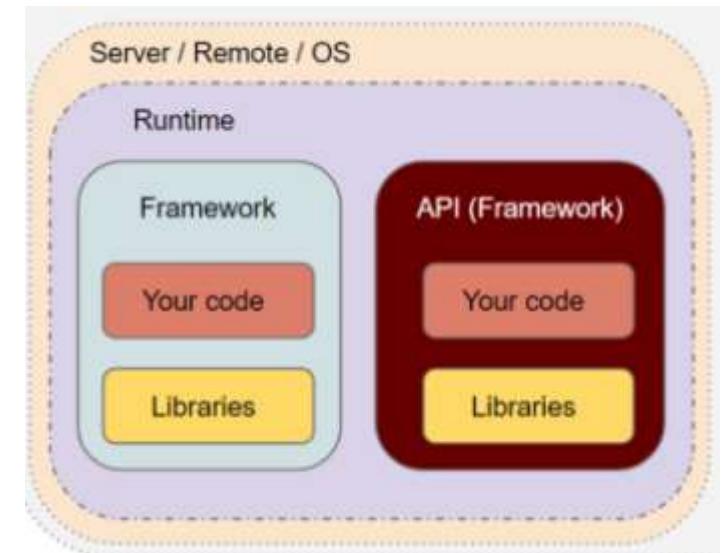
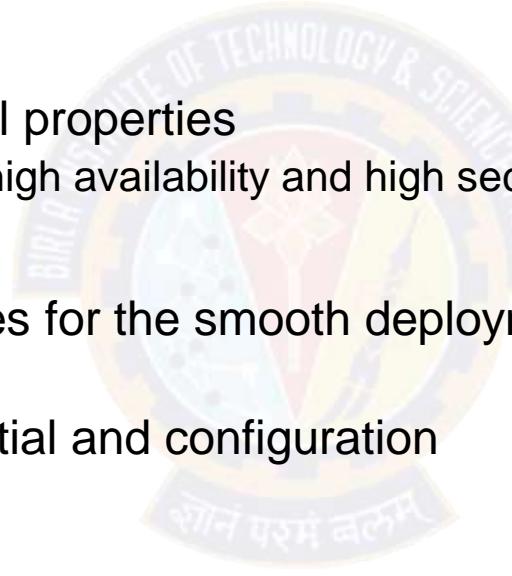
- Offers a toolbox for API design and development
  - ✓ Contains API building blocks, which are proven, reusable and configurable
  - ✓ APIs are constructed by composing these building blocks
  - ✓ There is no need to reinvent the wheel for the development of each new API
- Offers an integrated engineering environment with tools for the development and design of APIs
- The toolbox consists of:
  - ✓ Library of API building blocks
  - ✓ Language for implementing APIs
  - ✓ IDE for API development with editor, debugger and deployment tools
  - ✓ Language for designing APIs
  - ✓ Design tool for creating API interface designs
  - ✓ Tool for generating documentation and code skeletons based on the design.
- Targeted at the API developers, who work for the API provider
  - ✓ Supports the developers and enables them to develop APIs quickly and with high quality



appinventive

# API Runtime Platform

- Primarily executes the APIs
  - ✓ Enables the APIs to accept incoming requests from API consumers and to serve responses through HTTP server
- Responsible for favorable non-functional properties
  - ✓ such as high stability, high throughput, high availability and high security
- Also responsible for providing capabilities for the smooth deployment of new and maintenance of existing APIs
- Needs to support capabilities for credential and configuration management
- The runtime infrastructure provides capabilities for:
  - ✓ Load balancing, connection pooling and caching
  - ✓ Monitoring, logging and analytics
  - ✓ Deployment and maintenance
  - ✓ Credential and configuration management



medium

# API Engagement Platform

## API Provider Perspective

- Used by the API provider to interact with its community of API consumers
- The platform offers API providers the capabilities of product management and configuration
- API providers use the following capabilities:
  - ✓ API management
    - ❖ configuration and reconfiguration of existing APIs without the need for deployment of the API
  - ✓ API discovery
    - ❖ a mechanism for clients to obtain information about the APIs
  - ✓ Consumer onboarding (Client ID/App Key generation, Interactive API console)
  - ✓ Community management (Blogs, Forums, Social features etc.)
  - ✓ Documentation
  - ✓ Version management
  - ✓ Management of monetization and service level agreements (SLA)

# API Engagement Platform(2)

## API Consumer Perspective

- For API consumers, the API engagement platform is the information hub for
  - ✓ inspecting the API portfolio
  - ✓ accessing specific API documentation
  - ✓ managing credentials and managing rate plans
- From the perspective of the API consumers, the platform offers:
  - ✓ An overview of the API portfolio
  - ✓ A source of inspiration for API solutions
  - ✓ Documentation of APIs
  - ✓ Possibility to try APIs interactively
  - ✓ Example source code for integration
  - ✓ Self service to get access to APIs (credentials and rate plans)
  - ✓ Service announcements
  - ✓ Client tooling, such as code generators for clients

Reference :  
API Architecture  
Matthias Biehl



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# API Development Platform

Chandan Ravandur N

# API Development Platform

- Offers a toolbox for API design and development
  - ✓ Contains API building blocks, which are proven, reusable and configurable
  - ✓ APIs are constructed by composing these building blocks
  - ✓ There is no need to reinvent the wheel for the development of each new API
- Offers an integrated engineering environment with tools for the development and design of APIs
- The toolbox consists of:
  - ✓ Library of API building blocks
  - ✓ Language for implementing APIs
  - ✓ IDE for API development with editor, debugger and deployment tools
  - ✓ Language for designing APIs
  - ✓ Design tool for creating API interface designs
  - ✓ Tool for generating documentation and code skeletons based on the design.
- Targeted at the API developers, who work for the API provider
  - ✓ Supports the developers and enables them to develop APIs quickly and with high quality



appinventive

# Library of API Building Blocks

- When developing APIs, certain functionality is needed over and over again
- Extremely helpful to have this functionality available in a shared library of building blocks
  - ✓ Tested and proven in practice, so bugs are extremely seldom
  - ✓ Configurable and can be adapted for many purposes
  - ✓ Composable, i.e. an API can be built from a collection of properly configured blocks
- The building blocks has the following features:
  - ✓ Processing of HTTP requests and HTTP responses
    - ❖ including header parameters, query parameters, URI processing, HTTP status code, HTTP methods
  - ✓ Security
    - ❖ threat protection, IP-based access limitation, location-based access limitation, time-based access limitation
  - ✓ Frontend authentication and authorization with OAuth, Basic Authorization, API key
  - ✓ Frontend protocols
    - ❖ REST, SOAP, XML-RPC, JSON-RPC, WebSockets,
  - ✓ Protocol mediation - SOAP to REST, REST to SOAP
  - ✓ Data format transformation - XML to JSON, JSON to XML
  - ✓ Aggregation and orchestration of multiple APIs and/or multiple backend services
  - ✓ Throttling to protect backends and platforms - rate limitation and throughput limitation
  - ✓ Load balancing for incoming requests to the API platform and outgoing requests to the backends
  - ✓ Cache for incoming requests to the API platform and outgoing requests to the backends
  - ✓ Hooks for logging, analytics
  - ✓ Monetization capabilities and enforcement



apievangelist

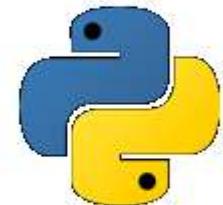
# Language for Designing APIs

- The design of the API needs to be described
- API development platforms provide two types of languages:
  - ✓ a low-level API implementation language
  - ✓ a high-level API description language
- API description language
  - ✓ Used to express the "what" of APIs
- API development Platform provides
  - ✓ design tools for creating API interface designs
  - ✓ tools for generating documentation from the API description
  - ✓ tools for generating implementations
- An API description language is a domain-specific language for expressing API design
  - ✓ Provides the well-defined semantics and the tooling ecosystem



# Language for Implementing APIs

- A language is used for composing the building blocks
  - ✓ Can be either a general purpose programming language or a domain-specific API development language
- General purpose programming languages
  - ✓ such as Java or JavaScript have the advantage of wide-spread use and many generic supporting tools
    - Node.js or Express.js for JavaScript and JAX-RS, Jersey, Restlet or Spring for Java
- Domain-specific configuration languages intentionally limit the expressibility
  - ✓ only offer a handful of language constructs, but all of them are relevant for API development
  - ✓ easier to learn, understand and validate them to discover erroneous or missing configurations
  - ✓ offer the advantage that APIs can be built by pure configuration
  - ✓ resulting in much smaller, more compact and more understandable implementations
- Usually offer their own integrated development environment with editor, compiler, debugger and deployment capabilities
  - ✓ DSLs are typically linked with a specific API development platform and are product-specific
- The existing skill set of your developers needs to be considered
  - ✓ usually more efficient to write APIs in a language that the API developers are familiar with and already productive in



Reference :  
API Architecture  
Matthias Biehl



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# API Platform Configurations and Interactions

Chandan Ravandur N

# Different Environments

- The components of the API platform should be available in different environments
  - ✓ Especially the runtime platform and the engagement platform
- Only one instance of the development platform is needed
  - ✓ can be shared among all environments
- APIs need to be deployed on different stages, called as environments
  - ✓ Each of the stages has a specific purpose
  - ✓ is separated from the other stages to isolate potential errors to a single stage
  - ✓ deployed APIs are tested and validated
- Ideally, there is a smooth migration of the API and its configuration from one stage to the next
  - ✓ after all the tests in the previous stage have been completed

# Different Stages

## Stages, ordered by increasing maturity:

- Sandbox or Simulation
  - ✓ Used for playing with interface design, provide mocks or simulation of an API, allow interaction with consumers
- Development
  - ✓ Used for development, which will eventually go to production
- Continuous Integration
  - ✓ Used for automated testing of the latest version
  - ✓ intended to provide feedback on the latest development version with short turnaround times
- Testing
  - ✓ Used for manual black box testing and integration testing
- Pre-Production
  - ✓ Used as a practice for production and for acceptance testing
- Production
  - ✓ Used as a real system for consumers

# API Platform Deployment Models

- The API platform may be deployed on premise or in the cloud
- Most API platforms are on premise solutions
  - ✓ since the required backend systems are also on premise
  - ✓ For security reasons the backend systems should not be available via the internet
  - ✓ The API layer is an additional protection and security layer for the backend systems
- API platforms, which are available in the cloud, are typically elastic and scale well
  - ✓ API platforms in the cloud make sense, if they do not need to connect to secured backend systems in a private network
  - ✓ if the backends of the APIs are publicly available services or APIs on other cloud systems

# Interactions between the Platforms

- Three components can be used separately
  - ✓ However, there are some dependencies between the platform components
- Design and Development
  - ✓ Design and development activity is only performed on the development platform
  - ✓ does not affect any other platform
- Deployment
  - ✓ When the development of an API proxy is finished, it is ready for deployment
  - ✓ Deployment transfers the API proxy (in a compiled or interpretable format) from the development platform to the runtime platform
  - ✓ The runtime platform should have
    - ❖ Configurations and credentials
    - ❖ All the building block libraries
    - ❖ External dependencies
  - ✓ API proxy is not available for consumers directly after the initial deployment
- Publishing
  - ✓ An API proxy that has been deployed for the first time, needs to be published on the engagement platform
  - ✓ Only published API proxies are available on the engagement platform
  - ✓ Once published, the API
    - ❖ is listed in the portfolio of the engagement platform
    - ❖ its documentation is generated from the API description,
    - ❖ overall monitoring and analytics are started



Reference :  
API Architecture  
Matthias Biehl



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

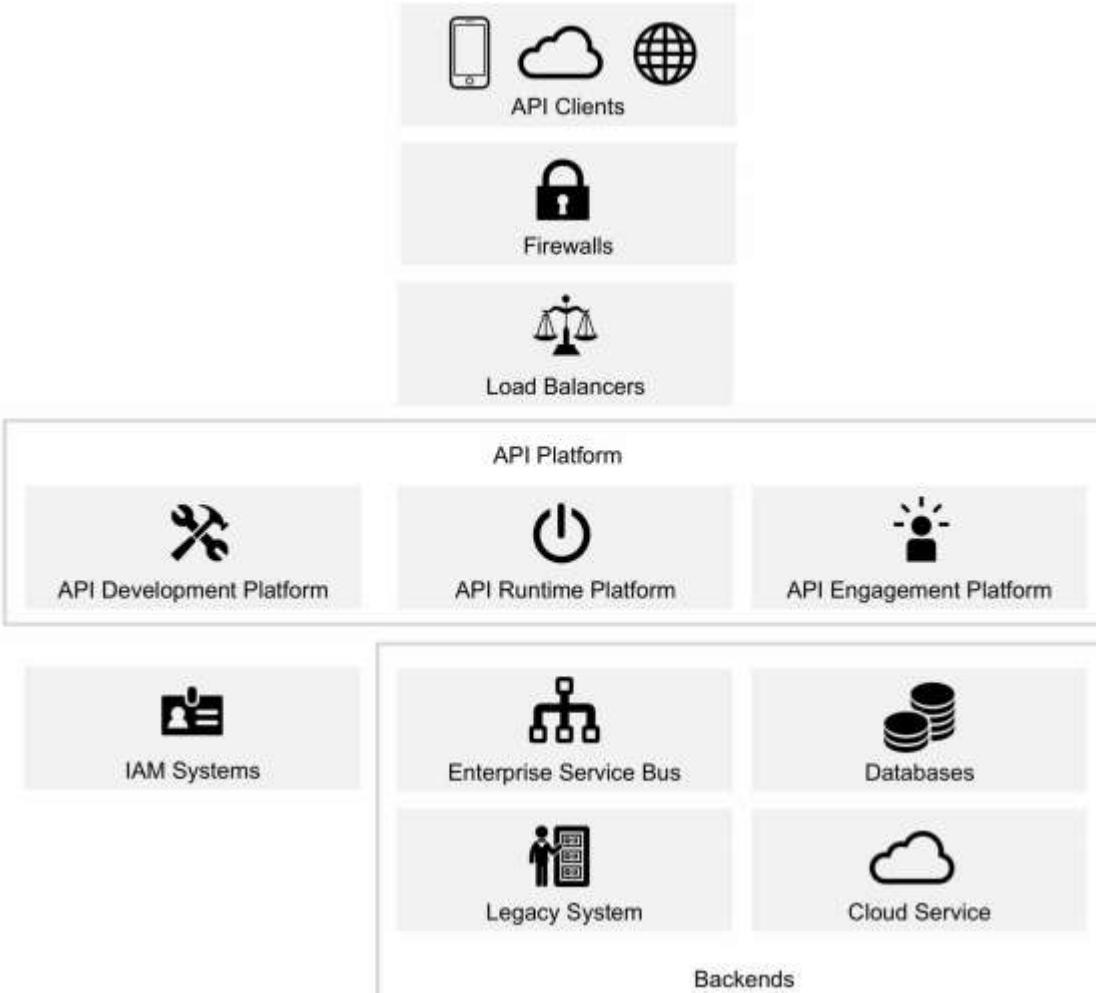
# Surrounding Systems

Chandan Ravandur N

# Surrounding Systems

- The API platform is not an isolated system
  - ✓ but it needs to be integrated into the existing architecture of the enterprise
- The size and complexity of the architecture surrounding the API platform typically depend on the size of the company
- Large enterprises have many legacy systems
  - ✓ grown their architecture organically
  - ✓ technical architecture of an enterprise is complex and thus the API platform becomes complex
- Small startups typically have no legacy systems
  - ✓ can form their architecture around the API platform
  - ✓ some of the building blocks may not even exist in a startup
- API clients are outside the scope of the surrounding systems of the API platform
  - ✓ Consumers develop clients to access the APIs
  - ✓ can be mobile apps, web applications, cloud services or embedded devices for the internet of things

# Surrounding Systems (2)



The api architecture

# Surrounding Systems (3)

- The surrounding systems are
  - ✓ partly before the API platform i.e. between the clients and the API platform
  - ✓ partly are hidden behind the API platform
- Before the API platform, there are
  - ✓ typically firewalls to improve the security
  - ✓ load balancers to improve the performance
- Behind the API platform there are
  - ✓ IAM systems for managing identity information
  - ✓ backend systems for providing the core functionality of the enterprise

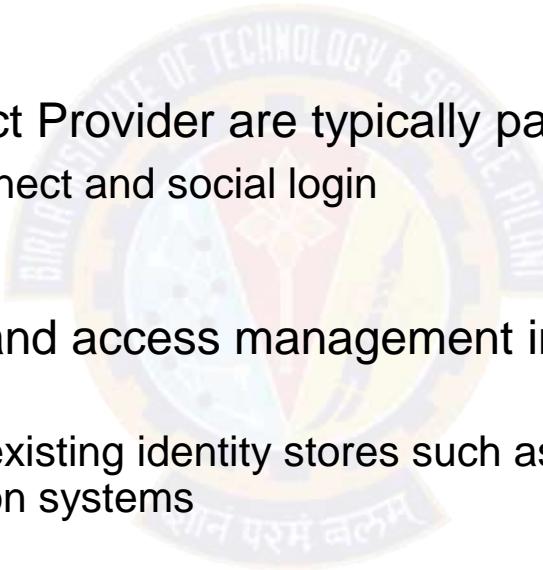


# Load Balancers and Firewalls

- Load balancers are usually placed between the API platform and the internet
  - ✓ Some API platforms may offer load balancing capabilities out of the box
  - ✓ others may rely on external component
- Load balancers are
  - ✓ Used to route the traffic from the internet to one of several nodes of an API platform running as a cluster
  - ✓ Spreads the requests equally among the nodes
- Another place for load balancers is between the API platform and the backend system
  - ✓ The API platform sends all requests for a specific backend system to a load balancer
  - ✓ which balances the requests over several instances of the backend system
- Firewalls are usually placed between the internet and the API platform
  - ✓ IP level filtering (ISO/OSI level 3) is usually performed by a separate security device
  - ✓ application level filtering (ISO/OSI level 7) is performed by the API runtime platform

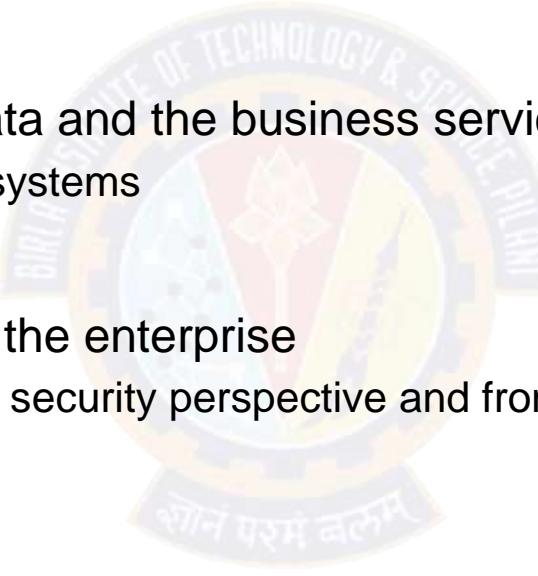
# Identity and Access Management Infrastructure

- APIs need to authenticate and authorize the users
  - ✓ The API platform provides building blocks for authentication and authorization
- OAuth provider and an OpenID Connect Provider are typically part of the runtime platform
  - ✓ to be able to offer OAuth, OpenID Connect and social login
- At the same time, the existing identity and access management infrastructure needs to be reused as much as possible
  - ✓ The API platform needs to connect to existing identity stores such as LDAP or Active Directory and to existing authentication and authorization systems



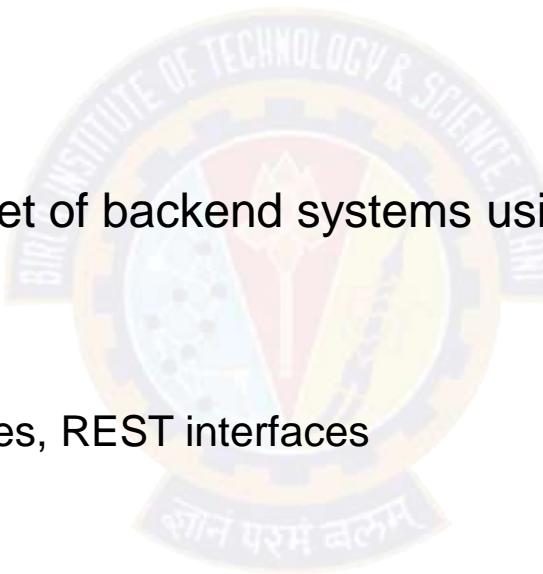
# Backends

- APIs typically do not implement the core business logic of the enterprise,
  - ✓ nor do they store customer or business data.
- ✓ APIs need to have access to the data and the business services for logic
  - Which typically reside in backend systems
- Backend systems form the heart of the enterprise
  - ✓ needs to be protected, both from a security perspective and from a performance & availability perspective
- The responsibility of the API platform is to
  - ✓ secure the access to the backend system
  - ✓ limit the load on the backends to a healthy and manageable level



# Backends (2)

- Backend systems typically do not provide the data and services in an easily digestible form
  - ✓ typically big, ugly, complicated
  - ✓ not easily digestible
  - ✓ simply not customer friendly
- Enterprises typically have a large set of backend systems using a variety of technologies
  - ✓ databases
  - ✓ applications
  - ✓ enterprise service buses
  - ✓ web services using SOAP interfaces, REST interfaces
  - ✓ message queues
- The responsibility of the API to
  - ✓ aggregate, filter and process the data into an easily-consumable form
  - ✓ offer the data and services in a homogeneous, modern protocol and data format



Reference :  
API Architecture  
Matthias Biehl



# Thank You!

In our next session:



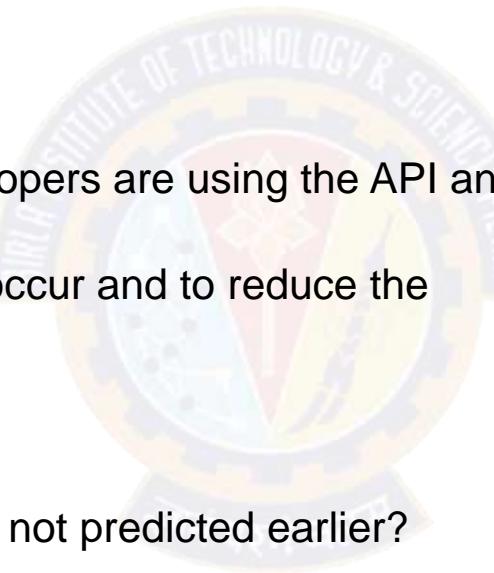
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Evolving API Design

Chandan Ravandur N

# Evolving API Design

- Initial API design might not scale with the growth of user base or with the increased adoption and usage of API
- Important
  - ✓ to determine the main reasons that developers are using the API and where the issues are
  - ✓ to get insights into bottlenecks that may occur and to reduce the number of API calls
- Few questions to be asked
  - ✓ Are developers using the API for reasons not predicted earlier?
  - ✓ Is polling a problem?
  - ✓ Is the API returning too much data?



crossroadsantigua

## Expert Advice

Grow and work with your users. Keep a good and open channel with your developers/users. Gain feedback and tune the API to solve their main pain points.

—Ido Green, developer advocate at Google

# Introducing New Data Access Patterns

- Developers might begin using it in ways that never anticipates
  - ✓ To address scaling challenges, might want to consider alternative ways of sharing data
- If polling is one of API scaling problems and have only REST APIs
  - ✓ should explore options like WebSockets and WebHooks
  - ✓ no need to poll for changes but can wait for new data to be delivered in real time
- For example,
- In the past, Twitter applications could only receive new tweets in near real time by frequently polling the Twitter API
  - ✓ Led to increased traffic to the REST APIs, and added to existing scaling challenges
  - ✓ Introduced a streaming API that delivers new data and cuts down on polling
  - ✓ Developers can subscribe to selected keywords or users and can receive new tweets over a long-lived connection
- In the past, GitHub found that its responses were bloated and were sending too much data
  - ✓ but even with the large payloads, they still weren't including all of the data that developers needed
  - ✓ Developers would make separate calls to assemble a complete view of a resource
  - ✓ GitHub launched the GraphQL API
  - ✓ Using GraphQL, developers can batch multiple API calls into a single API call and fetch only the items that they needed

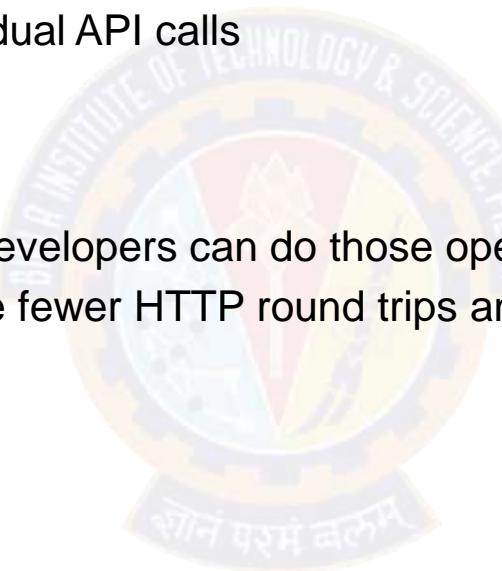


# Adding New API Methods

- Another way to address scalability and performance problems is by adding new API methods
- If some expensive APIs are there, might want to dive deeper into the use cases they are serving
  - ✓ Developers might need only a small subset of the data from API responses
  - ✓ Developers might be working hard to assemble data that isn't easily accessible with the existing APIs
- If developers can only either request everything or nothing,
  - ✓ they may end up receiving full responses and ignoring most of the payloads
  - ✓ possible that the data they don't need is expensive for you to compute
- Difficult to remove or change APIs after they are in use, but adding new methods is easy
  - ✓ New methods can deliver the data that developers need
  - ✓ while addressing the performance and scale issues of existing APIs

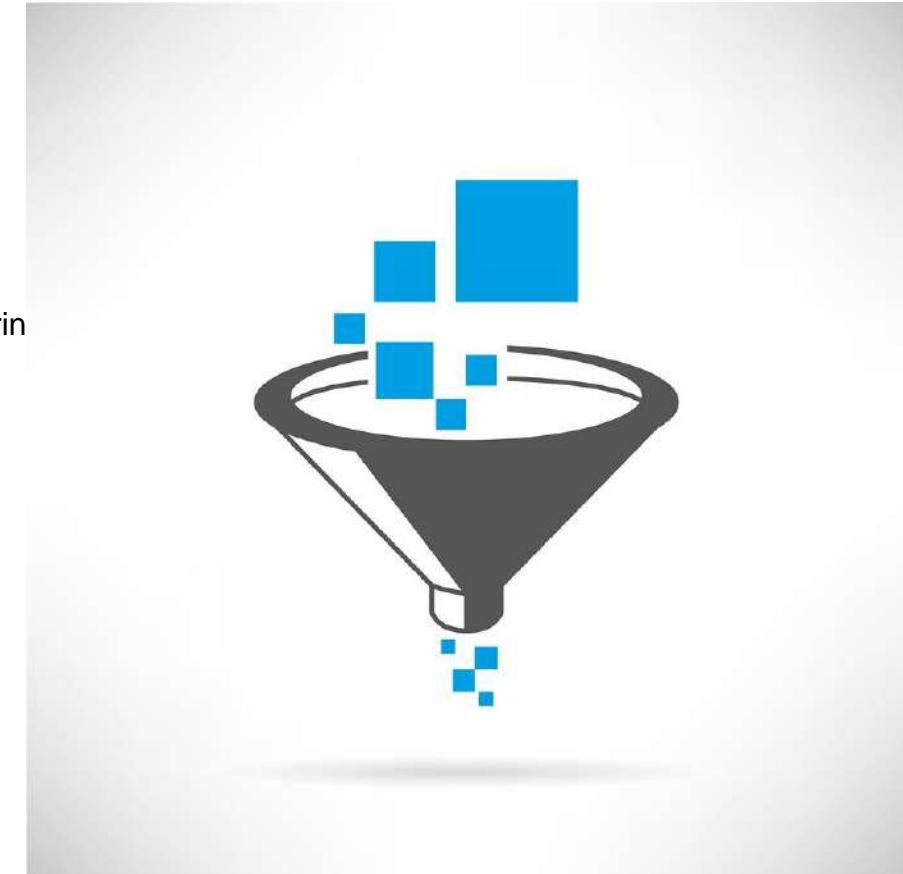
# Supporting Bulk Endpoints

- Developers need to do the same operation on several items
  - ✓ like looking up or updating multiple user
  - ✓ Often requires doing several individual API calls
- Can be helpful for scaling
  - ✓ to support bulk endpoints so that developers can do those operations in fewer API calls
  - ✓ more efficient because they require fewer HTTP round trips and can even help in reducing load on a database



# Adding New Options to Filter Results

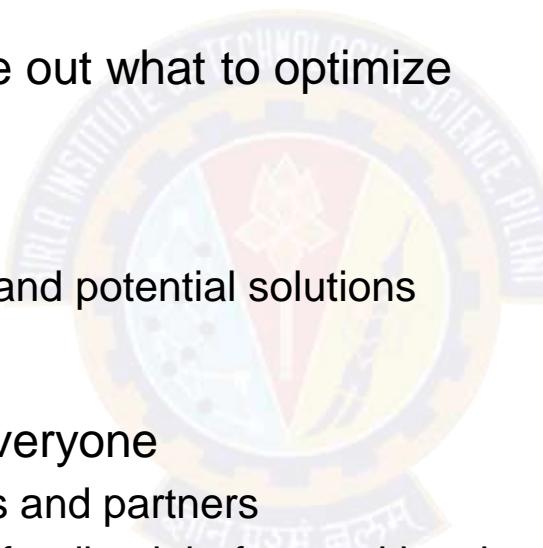
- Important to consider providing options to filter the results returned by APIs
  - ✓ Different APIs need different filters depending on how they are used
- Developers can limit the number of objects returned to those that they actually need
  - ✓ makes API more scalable
- Common filters:
- Search filter
  - ✓ can specifically request the results they are looking for using similar words, regex, or matching strings
  - ✓ otherwise developers could end up requesting and parsing a lot more results than they need
- Date filter
  - ✓ Need only new results since the last time they requested results from the API
  - ✓ Can return only the results from after or before the given timestamp
  - ✓ The Twitter timeline, the Facebook News Feed
- Order filter
  - ✓ enables developers to order a set of results by a given property
  - ✓ can reduce the number of results that developers need to request and process
  - ✓ The Amazon Product Advertising API supports sorting by popularity, price, and condition
- Options to indicate which fields to return or not return
  - ✓ Unnecessary fields can also significantly increase the response payload size
  - ✓ Can offer developers an option to exclude or include certain fields
  - ✓ Twitter timeline API provides filters to trim included user objects and not return tweets



# Evolving API Design Best Practices

## Best practices for evolving API design that will help with scaling

- Important to ensure that do not introduce surprising breaking changes to developers
- Analyze API usage and patterns to figure out what to optimize
- Talk to developers and partners
  - ✓ will provide good insights into problems and potential solutions
- Before launching new API patterns for everyone
  - ✓ try them out with a handful of developers and partners
  - ✓ can iterate on the design based on their feedback before making the patterns generally available



Reference:

Designing Web APIs – building APIs that developers love

Jin, Sahni and Shevat



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# API Description Language

Chandan Ravandur N

# API Description Languages

- APIs needs to be described in some form
  - ✓ especially when it needs to be communicated among various stakeholders
  - ✓ should be as easy as possible to describe APIs
- API description languages
  - ✓ Specialized languages can support the crafting of useful API descriptions by providing appropriate abstractions and language concepts
- Original purpose of API description languages was a language for creating API documentation
  - ✓ similar as JavaDoc provides a language for documenting Java programs
- Today, API description languages can be used for many additional purposes during the design and development process of APIs
  - ✓ are machine readable specifications of the API
  - ✓ can be used for automating tasks in API development
  - ✓ automation has the potential to increase the productivity of API development
  - ✓ can also be used for the partially automated generation of code for the API
- API description languages are very powerful tools that can be far more than just languages
  - ✓ can serve as the "single source of truth"
  - ✓ can be used as the main reference for all aspects of API design and development



# What are API Description Languages?

- Domain specific languages, which are especially suited for describing APIs
  - ✓ both human readable and machine readable languages, much like programming languages
- Are intuitive languages that can be easily written, read and understood by API developers and API designers alike
  - ✓ are also precise, leave little room for ambiguity and are very expressive and powerful
  - ✓ have a well-defined syntax, which makes it possible to process them automatically by software
- Compared to programming languages or API implementation languages
  - ✓ use a higher level of abstraction and a declarative paradigm
  - ✓ can be used to express the "what" instead of the "how"
- For example,
  - ✓ define the data structure of the possible responses (the "what")
  - ✓ instead of describing how the response is computed (the "how")
  - ✓ very well suited for expressing the architecture of each API proxy and the design of the API portfolio as a whole



<https://petstore.swagger.io/v2/swagger.json>

Explore

## Swagger Petstore

[ Base URL: [petstore.swagger.io/v2](https://petstore.swagger.io/v2) ]  
<https://petstore.swagger.io/v2/swagger.json>

This is a sample server Petstore server. You can find out more about Swagger at <http://swagger.io> or on <irc://freenode.net/#swagger>. For this sample, you can use the api key: `special-key` to test the authorization filters.

[Terms of service](#)

[Contact the developer](#)

Apache 2.0

[Find out more about Swagger](#)

Schemes

HTTPS

Authorize 

**pet** Everything about your Pets

Find out more: <http://swagger.io> 

**POST** /pet Add a new pet to the store

**PUT** /pet Update an existing pet

**GET** /pet/findByStatus Finds Pets by status

/pet/findByTags Finds Pets by tags

**GET** /pet/{petId} Find pet by ID

**POST** /pet/{petId} Updates a pet in the store with form data

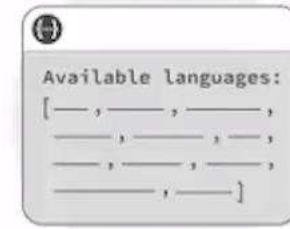
**DELETE** /pet/{petId} Deletes a pet

# API Description Language vs. API Development Language

- An API platform should provide two types of languages
- First language that should be provided is a higher-level language - API description language
  - ✓ can be used for designing APIs and for expressing the "what"
- Second language that should be provided by an API platform is a lower-level language - API development language
  - ✓ used for implementing APIs and for expressing the "how".
- Relation between API description languages and API development languages
  - ✓ First, APIs should be designed using an API description language
  - ✓ Some API platforms have support for API description languages built in
  - ✓ means that the platform supports the parsers, and code generators for a specific API description language
  - ✓ The generated implementations should be expressed in the API development language



Design



Build



Document

# Language Features

- API description is a technical contract between API provider and API consumer
  - ✓ important that the designed contract is unambiguous and clear
  - ✓ should provide clarity to all involved stakeholders
  - ✓ should enable simultaneous development of both the consuming and providing software components
- Properties:
  - ✓ Intuitive, Compactness, Precision and Relevance
  - ✓ Communication
  - ✓ Agility
  - ✓ Quick Validation and Iteration

# Options

- Today, a number of API description languages are available
- 
- The most popular and widespread API description languages
  - ✓ Swagger, RAML, Blueprint, Mashery I/O Docs, WADL and WSDL
- WADL and WSDL
  - ✓ have been around for a long time, their use is widespread, they can be used for API design,
  - ✓ however, they have not been created specifically for API design and thus are lacking important features
- Swagger (OpenAPI), RAML, API Blueprint and Mashery I/O Docs
  - ✓ have been created specifically for RESTful API design



Swagger™



apiblueprint

RAML

Reference:  
API architecture  
MATTHIAS BIEHL



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# API Languages - Usage

Chandan Ravandur N

# API Languages - Usage

## Several use cases for API descriptions

- API descriptions can support all phases of API design and development
  - ✓ use cases are not complete, but they are chosen to convey the central role of API descriptions
- Use cases
  - ✓ Communication and Documentation
  - ✓ Design Repository
  - ✓ Contract Negotiation
  - ✓ API Implementation
  - ✓ Client Implementation
  - ✓ Discovery
  - ✓ Simulation

# Communication and Documentation

- API is an interface connecting two or more software systems
  - ✓ important that the API is understood by the involved developers on all sides
  - ✓ Developers are on the side of the API provider and busy developing the API
  - ✓ Developers are on the consumer side - develop apps that use the API
- The idea of loose coupling of services is great
  - ✓ as long as it is ensured that the services are well understood by the developers on API provider side and on API consumer side
- To provide a shared understanding of an API, the API needs to be well documented
  - ✓ developers are not co-located and can quickly share their insight
  - ✓ they are spread out over different companies, countries, continents and time zones
  - ✓ An appropriate documentation can help in this case
- This is why the documentation of APIs is extremely important
  - ✓ for both developers of the server-side API implementation and for the client-side API consumers



The screenshot shows the OpenWeatherMap API documentation page on Swagger UI. At the top, there's a navigation bar with tabs for 'swagger' (selected), 'http://api.openweathermap.org/2.5/weather?lat=40&lon=-37&appid=00000000000000000000000000000000', and 'Explore'. Below the header, the title 'OpenWeatherMap API' is displayed with a '4.0.0' badge. A note below the title says: 'Get current weather info for 10 days, and 5 hourly forecast for your city. API is free, reliable, and it's easy to integrate with your application. Interactive mode allows you to play with requests, responses, and even your local environment. Data is available in JSON, XML, or JSONP format. Note: This API is based on the current OpenWeatherMap API version 2.5.' There are several sections below, including 'Definitions', 'Responses', and 'API Documentation'. On the right side, there's a 'Authorize' button. At the bottom, there are sections for 'Server' (set to 'http://api.openweathermap.org/2.5/weather') and 'Current Weather Data' (with a dropdown showing 'Gva, Germany'). A 'GET /weather' button is also visible.

# Design Repository

- The API description of an API proxy is the central reference of truth for this API
- API description is the definitive, authoritative point of reference
  - ✓ If you are ever in doubt, which version of the API accepts a certain parameter or which status codes are returned by the API
- Contains all the important design decisions for that API proxy
- Complete API portfolio, including the API descriptions of all API proxies should be documented
- To provide a history and synchronized access in a distributed development team
  - ✓ the API descriptions should be put under version control, e.g. in a GIT or SVN repository



Swagger™



# Contract Negotiation

- In contract-first design, the precise description of the design contract is essential
  - ✓ the strength of API description languages.
- API description can serve as a design contract
  - ✓ can be used for agreements between API designer and API developer or
  - ✓ as a contract between API consumer and API provider
  - ✓ Both contracting parties negotiate this contract, decide on it
  - ✓ rely on this contract during the implementation and maintenance phases

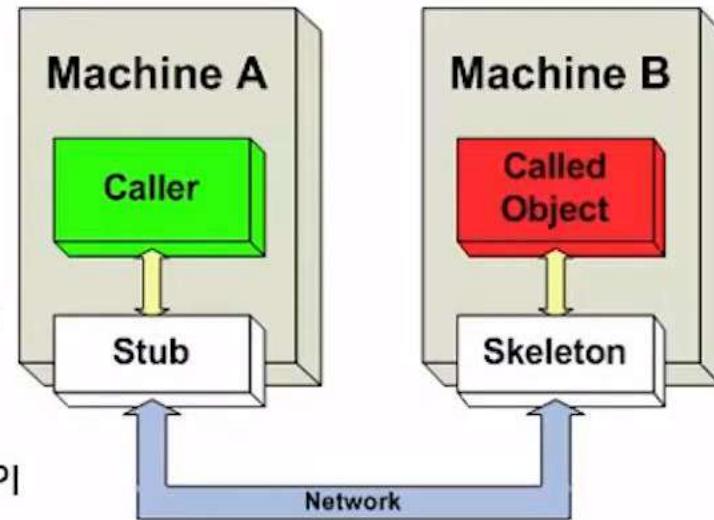


Health careers

- Traditionally, app developers would need to wait for the API to be finished
- Contract first design allows starting the implementation of the app by the consumer before the provider has finished building the API
  - ✓ allows for a very efficient development process with a much quicker turn around time
  - ✓ allows app developers to bring their apps to market quicker than before

# API and Client Implementation

- On the API provider side,
  - ✓ API description is machine readable, it can be used for automating tasks in software development
  - ✓ have the potential to increase the productivity of software development
- Can be used by the API provider to automatically generate API skeletons
  - ✓ An API skeleton contains some important pieces of the implementation, it is, however, not complete
  - ✓ The skeleton needs to be extended and filled with manual implementation before the API can be used
  - ✓ contribute to a higher speed for API implementation as well as to a higher quality of the API implementation
- On the API consumer side,
  - ✓ API description can be used for generating client stubs for accessing the API
  - ✓ With an appropriate code generator, the client stub can be generated for the programming language used by the consumer
- By generating the client stub for accessing the API, it is ensured that the implementation actually matches the specified contract
  - ✓ For the API consumer, code generation speeds up the development process

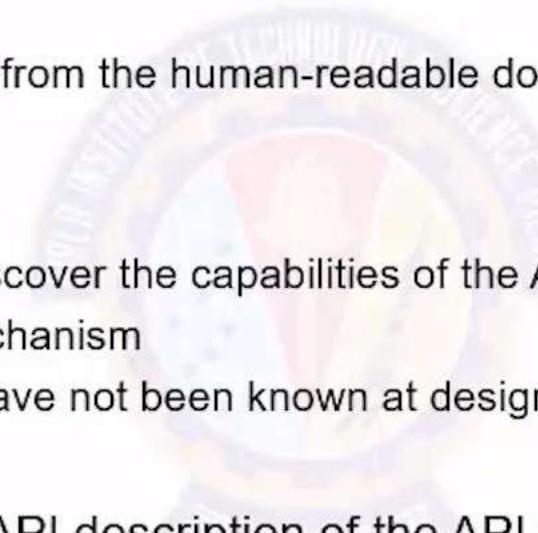


wikipedia

# Discovery

## How does the client know about the capabilities of the API?

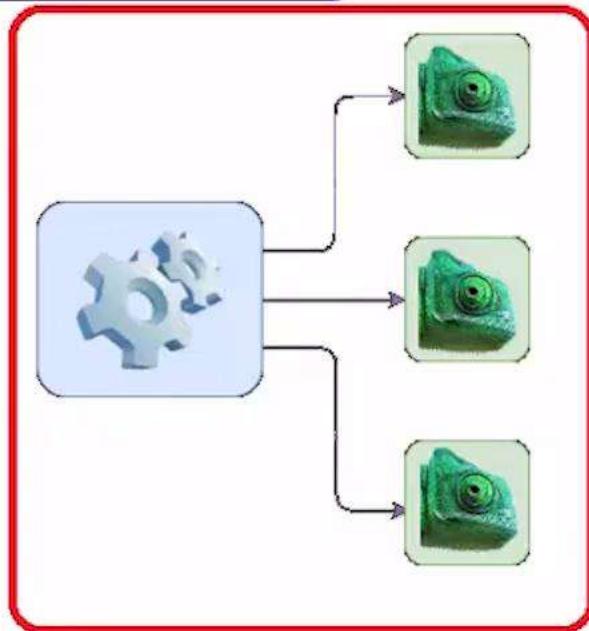
- One answer
  - ✓ client does not need to know, since the API needs to be understood by the API consumer and he develops the client
  - ✓ The consumer can learn about the API from the human-readable documentation
- Another answer
  - ✓ client needs to be able to explore or discover the capabilities of the API programmatically
  - ✓ With such an automated discovery mechanism
  - ✓ an app may include new APIs, which have not been known at design time
- To enable such an implementation, an API description of the API portfolio should be served by a specific endpoint
  - ✓ allows the caller to discover each API within the portfolio by downloading and parsing the API description
  - ✓ A precondition is that the API provider made the API description available to the consumers



dzone

# Simulation

- A simulation can provide a first impression of the finished API to the consumer
- In the early phases of API design,
  - ✓ a simulation can be presented to pilot consumers for eliciting their initial feedback
  - ✓ The pilot consumers can even base first demos of their apps on the API simulation
- A model of the real world is needed to create a simulation
  - ✓ An API description contains such a model
- Model is provided in the form of the interface specifications that are necessary to build a simple simulation or mockup
  - ✓ A specification of the input data in the form of
    - ❖ query parameters
    - ❖ form parameters
    - ❖ header parameters path parameters
    - ❖ a data structure in the message body is included in an API description
  - ✓ A specification of the error messages
    - ❖ can produce error behavior according to the specification.
  - ✓ An example response is specified, which can be served by the simulated API.



Reference:  
API architecture  
MATTHIAS BIEHL



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

**Swagger**

**Chandan Ravandur N**

# Swagger

- Swagger is an Interface Description Language for describing RESTful APIs expressed using JSON
- Swagger is used together with a set of open-source software tools
  - to design, build, document, and use RESTful web services
- Swagger includes
  - automated documentation
  - code generation (into many programming languages)
  - and test-case generation tools



Swagger	
Developer(s)	SmartBear Software
Initial release	2011; 10 years ago
License	Apache License 2.0
Website	<a href="https://swagger.io">swagger.io</a>

# History

- During the development of Wordnik's products
  - ✓ the need for automation of API documentation and client SDK generation became a major source of frustration
- Swgger was created in 2011 by Tony Tam technical co-founder of the dictionary site Wordnik
- The Swagger API project was made open source in September 2011
- Soon after release, a number of new components were added to the project
  - ✓ including a stand-alone validator, support for Node.js, and Ruby on Rails
- In Swagger's early years, modest traction came from small companies and independent developers
- RESTful APIs typically did not have a machine-readable description mechanism
  - ✓ Swagger provided a simple and discoverable way to do so
- Helped by the use of the Apache 2.0 open-source license, a number of products and online services began including Swagger in their offerings
  - ✓ which accelerated quickly after adoption by Apigee, Intuit, Microsoft, IBM and others who began to publicly endorse the Swagger project



## History (2)

- Shortly after Swagger was created, alternative structures for describing RESTful APIs were introduced
  - ✓ the most popular being API Blueprint in April 2013 and RAML in September 2013
  - ✓ these competing products had stronger financial backing than Swagger, they initially focused on different use cases from Swagger
- In November 2015, SmartBear Software, the company that maintained Swagger, announced that
  - ✓ it was helping create a new organization, under the sponsorship of the Linux Foundation, called the OpenAPI Initiative
  - ✓ A variety of companies, including Google, IBM, and Microsoft are founding members
- On 1 January 2016, the Swagger specification was renamed to OpenAPI Specification
  - ✓ was moved to a new software repository on GitHub
- While the specification itself was not changed
  - ✓ this renaming signified the split between the API description format and the open-source tooling



# Usage

## Swagger's open-source tooling usage can be broken up into use cases

- Development, interaction with APIs, and documentation
- Developing APIs
  - ✓ When creating APIs, Swagger tooling may be used to automatically generate an Open API document based on the code itself
  - ✓ This embeds the API description in the source code of a project
  - ✓ Informally called code-first or bottom-up API development
- Alternatively, using Swagger Codegen, developers can decouple the source code from the Open API document
  - ✓ Generate client and server code directly from the design
  - ✓ This makes it possible to defer the coding aspect
- Interacting with APIs
  - ✓ Using the Swagger Codegen project, end users generate client SDKs directly from the OpenAPI document
  - ✓ reducing the need for human-generated client code
  - ✓ supported over 50 different languages and formats for client SDK generation
- Documenting APIs
  - ✓ When described by an OpenAPI document, Swagger open-source tooling may be used to interact directly with the API through the Swagger UI
  - ✓ This project allows connections directly to live APIs through an interactive, HTML-based user interface
  - ✓ Requests can be made directly from the UI and the options explored by the user of the interface

Swagger Open Source

Editor

Codegen

UI

Swagger Codegen

Swagger Editor

Swagger UI

Reference:  
API architecture  
MATTHIAS BIEHL



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Open API- Specification

Chandan Ravandur N

# OpenAPI - Specification

## Basic Structure

- Can write OpenAPI definitions in YAML or JSON
- All keyword names are case-sensitive.
- A sample OpenAPI 3.0 definition written in YAML looks like:

```
openapi: 3.0.0
info:
  title: Sample API
  description: Optional multiline or single-line description in [CommonMark] (http://commonmark.org/help/) or HTML.
  version: 0.1.9
servers:
  - url: http://api.example.com/v1
    description: Optional server description, e.g. Main (production) server
  - url: http://staging-api.example.com
    description: Optional server description, e.g. Internal staging server for testing
paths:
  /users:
    get:
      summary: Returns a list of users.
      description: Optional extended description in CommonMark or HTML.
      responses:
        '200': # status code
          description: A JSON array of user names
          content:
            application/json:
              schema:
                type: array
                items:
                  type: string
```

# Metadata

- Every API definition must include the version of the OpenAPI Specification that this definition is based on:
  - ✓ openapi: 3.0.0
- The OpenAPI version defines the overall structure of an API definition –
  - ✓ what you can document and how you document it
- OpenAPI 3.0 uses semantic versioning with a three-part version number.
  - ✓ available versions are 3.0.0, 3.0.1, 3.0.2, and 3.0.3

- The info section contains API information:
  - ✓ title, description (optional), version

```
openapi: 3.0.0
info:
  title: Sample API
  description: Optional multiline or single-line description
  version: 0.1.9
```

- title is your API name
- description is extended information about your API
- version is an arbitrary string that specifies the version of your API

- can use semantic versioning like major.minor.patch, or an arbitrary string like 1.0-beta or 2017-07-25
- info also supports other keywords for contact information, license, terms of service, and other details.

# Servers

- The servers section specifies the API server and base URL
  - ✓ can define one or several servers, such as production and sandbox

```
servers:  
  - url: http://api.example.com/v1  
    description: Optional server description, e.g. Main (production) server  
  - url: http://staging-api.example.com  
    description: Optional server description, e.g. Internal staging server for testing
```

- All API paths are relative to the server URL
- In the example above, /users means `http://api.example.com/v1/users` or `http://staging-api.example.com/users`
  - ✓ depending on the server used

# Paths

- The paths section defines individual endpoints (paths) in API, and the HTTP methods (operations) supported by these endpoints
- An operation definition includes
  - ✓ parameters,
  - ✓ request body (if any),
  - ✓ possible response status codes (such as 200 OK or 404 Not Found)
  - ✓ and response contents



- For example, GET /users can be described as:

```
paths:  
  /users:  
    get:  
      summary: Returns a list of users.  
      description: Optional extended description in CommonMark or HTML.  
      responses:  
        '200':  # status code  
          description: A JSON array of user names  
          content:  
            application/json:  
              schema:  
                type: array  
                items:  
                  type: string
```

# Parameters

- Operations can have parameters passed via
  - ✓ URL path (/users/{userId}),
  - ✓ query string (/users?role=admin),
  - ✓ headers (X-CustomHeader: Value)
  - ✓ or cookies (Cookie: debug=0)
- You can define the parameter
  - ✓ data types,
  - ✓ format,
  - ✓ whether they are required or optional,
  - ✓ and other details

```
1. paths:
2.   /users/{userId}:
3.     get:
4.       summary: Returns a user by ID.
5.       parameters:
6.         - name: userId
7.           in: path
8.           required: true
9.           description: Parameter description in CommonMark or HTML.
10.          schema:
11.            type : integer
12.            format: int64
13.            minimum: 1
14.          responses:
15.            '200':
16.              description: OK
```

# Request Body

- If an operation sends a request body, use the `requestBody` keyword to describe the body content and media type.

```
1. paths:
2.   /users:
3.     post:
4.       summary: Creates a user.
5.       requestBody:
6.         required: true
7.         content:
8.           application/json:
9.             schema:
10.               type: object
11.               properties:
12.                 username:
13.                   type: string
14.             responses:
15.               '201':
16.                 description: Created
```

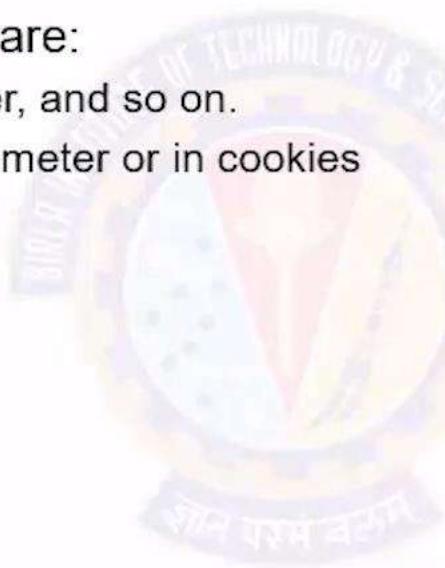
# Responses

- For each operation, you can define
  - ✓ possible status codes, such as 200 OK or 404 Not Found,
  - ✓ and the response body schema.
  - ✓ Schemas can be defined inline or referenced via \$ref
- You can also provide example responses for different content types

```
responses:  
  '200':  
    description: A user object.  
    content:  
      application/json:  
        schema:  
          type: object  
          properties:  
            id:  
              type: integer  
              format: int64  
              example: 4  
            name:  
              type: string  
              example: Jessica Smith  
  '400':  
    description: The specified user ID is invalid (not a number).  
  '404':  
    description: A user with the specified ID was not found.  
  default:  
    description: Unexpected error
```

# Authentication

- The securitySchemes and security keywords are used to describe the authentication methods used in your API.
- Supported authentication methods are:
  - ✓ HTTP authentication: Basic, Bearer, and so on.
  - ✓ API key as a header or query parameter or in cookies
  - ✓ OAuth 2
  - ✓ OpenID Connect Discovery



```
components:  
  securitySchemes:  
    BasicAuth:  
      type: http  
      scheme: basic  
  
  security:  
    - BasicAuth: []
```

Reference:  
API architecture  
MATTHIAS BIEHL



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

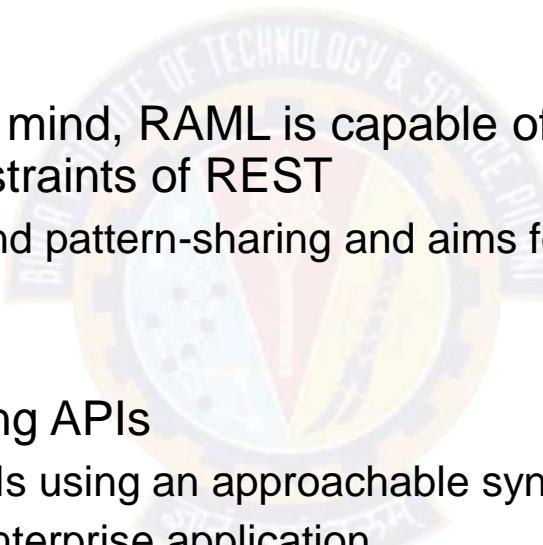
**RAML**

**Chandan Ravandur N**

# RAML

## RESTful API Modeling Language (RAML)

- A YAML-based language for describing RESTful APIs
  - ✓ Provides all the information necessary to describe RESTful APIs
- Although designed with RESTful APIs in mind, RAML is capable of describing APIs that do not obey all constraints of REST
  - ✓ Encourages reuse, enables discovery and pattern-sharing and aims for merit-based emergence of best practices
- A simple but powerful syntax for modelling APIs
  - ✓ RAML enables rapid development of APIs using an approachable syntax
  - ✓ which can scale from hobby project to enterprise application



# History

- First proposed in 2013
- The initial RAML specification was authored by Uri Sarid, Emiliano Lesende, Santiago Vacas and Damian Martinez
  - ✓ Garnered support from technology leaders like MuleSoft, AngularJS, Intuit, Box, PayPal, Programmable Web and API Web Science, Kin Lane, SOA Software, and Cisco
  - ✓ Development is managed by the RAML Workgroup
- The current workgroup signatories include technology leaders from
  - ✓ MuleSoft, AngularJS, Intuit, Airware, Programmable Web and API Science
  - ✓ SOA Software , Cisco , VMware , Akamai Technologies and Restlet
- Very few existing APIs meet the precise criteria to be classified as RESTful APIs
- Consequently, like most API initiatives in the 2010s, RAML has initially focused on the basics of practically RESTful APIs
  - ✓ including resources, methods, parameters, and response bodies that need not be hypermedia
- There are plans to move towards more strictly RESTful APIs as the evolution of technology and the market permits

# RAML

## Open Sourced

- Number of reasons why RAML has broken out from being a proprietary vendor language and has proven interesting to the broader API community:
  - ✓ RAML has been open-sourced along with tools and parsers for common languages
  - ✓ The development of RAML will be overseen by a steering committee of API and UX practitioners
  - ✓ There is an emerging ecosystem of third party tools being developed around RAML
- MuleSoft originally started using Swagger (now OpenAPI Specification), but decided it was best suited to documenting an existing API,
  - ✓ not for designing an API from scratch
- RAML evolved out of the need to support up-front API design in a succinct, human-centric language
  - ✓ introduced language features that support structured files and inheritance that address cross-cutting concerns
- Open API Initiative
  - ✓ A new organization, under the sponsorship of the Linux Foundation was set up in 2015 to standardize the description of RESTful APIs
  - ✓ A number of companies including SmartBear, Google, IBM and Microsoft were founding members
  - ✓ SmartBear donated the Swagger specification to the new group
  - ✓ RAML and API Blueprint are also under consideration by the group

# Example

```
1  #%RAML 0.8
2
3  title: World Music API
4  baseUri: http://example.api.com/{version}
5  version: v1
6  traits:
7      - paged:
8          queryParameters:
9              pages:
10                 description: The number of pages to return
11                 type: number
12             - secured: !include http://raml-example.com/secured.yml
13 /songs:
14     is: [ paged, secured ]
15     get:
16         queryParameters:
17             genre:
18                 description: filter the songs by genre
19     post:
20     /{songId}:
21         get:
22             responses:
23                 200:
24                     body:
25                         application/json:
26                             schema: |
27                             { "$schema": "http://json-schema.org/schema",
28                               "type": "object",
29                               "description": "A canonical song",
30                               "properties": {
31                                 "title": { "type": "string" },
32                                 "artist": { "type": "string" }
33                               },
34                               "required": [ "title", "artist" ]
35                             }
36                         application/xml:
37         delete:
38             description: |
39                 This method will *delete* an **individual song**
```

# API gateways supporting RAML

- Apigee
- MuleSoft
- AWS API Gateway by AWS (through AWS API Gateway Importer)
- Akana
- Restlet
- Can convert RAML specification to either OpenAPI or API Blueprint using APIMATIC
  - ✓ thus enabling to use further API gateways



Reference:  
Wikipedia



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

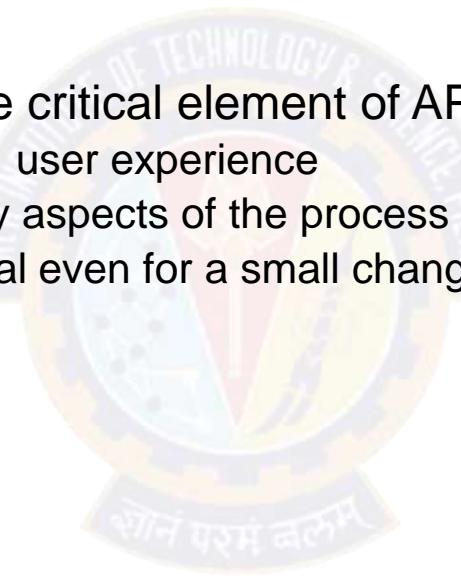
# Continuous API improvement

Chandan Ravandur N

# API Change

**Not necessary to change. Survival is not mandatory. – W. Edwards Deming**

- Truly Said! If you don't want to survive, don't change.
  - ✓ Change is the only constant thing in life!
- Managing changes to APIs turns to be critical element of API management
  - ✓ Have big impact on software product, user experience
  - ✓ Breaking change can impact lot many aspects of the process
  - ✓ Change management becomes critical even for a small change to be carried out!
- You need to change API to
  - ✓ Fix a bug in API implementation
  - ✓ Improve developer experience
  - ✓ Optimize the implementation
- Depends on scope as well
  - ✓ Interface can change
  - ✓ Implementation can change
  - ✓ Instance can change
  - ✓ Supporting assets can change



# Changing interface

- Every API has interface – abstraction over the details
  - ✓ Describes behaviour of API to external world without exposing internal details
  - ✓ Includes info about protocols, messages, vocabulary etc.
  - ✓ Not implemented – can be hand drawn or defined using languages
  - ✓ Needs to be consistent with documentation
- Interface model changes are highly impactful, inevitable for any API product
  - ✓ Need to add support for new feature
  - ✓ Need to make a change to improve API usability
  - ✓ Need to deprecate part of interface because of changes in business model
- Has larger impact on the consumer application code that uses APIs
  - ✓ Depends heavily on coupling between apps code and APIs interface
  - ✓ Larger the bonding, major is impact
- Lesser the coupling, easier is to change interface
  - ✓ Better to share interface only after its completely “ready”
  - ✓ Once its public – its like a diamond, forever – Joshua Bloch

# Changing Implementation

- Implementation is magic behind the APIs – brings life into it
  - ✓ Enables parties / components to interact with each other
  - ✓ Consists of code, configuration, data, infrastructure etc.
  - ✓ Consumers of API don't need to worry about these internal / private details of API
  - ✓ Also should not be impacted negatively by the change in the implementation
- Need to maintain API over its life – so change is inevitable things – need to change implementation
  - ✓ When model changes – implementation has to change – no excuse
  - ✓ Sometimes implementation can change – not triggered by change in model
  - ✓ Fixing bug, performance improvement change, code rewriting etc.
- Impact of change is hidden behind interface of API
  - ✓ Client should not need to change the app code – unless interface has changed
  - ✓ Downside is that the change can affect instance and supporting assets – needs to be updated as well

# Changing instance

- Implementation expresses model as callable, usable interface
- Instance of API is managed, running expression of interface
  - ✓ Instance is running machine through which over the network API is made available to consumer
- Change in API is not done until its not reflected onto the instances
  - ✓ Interface change or implementation change – both has to be applied on the instance
  - ✓ Then only available to the clients / consumers of the APIs
- Can change instance independently without interface or implementation change
  - ✓ Changing a system configuration value
  - ✓ Cloning a machine
  - ✓ Destroying a machine
- Mostly done for improving availability, performance , reliability of systems

# Changing supporting assets

- Have least cascading impact, can produce highest impact on user experience
  - ✓ Can be done irrespective of changes in interface, implementation or instance
- Better developer experience is crucial for any API products success – which can be achieved easily by investing into supporting assets of product such as
  - ✓ Documentation
  - ✓ Developer portal
  - ✓ Troubleshooting tools
  - ✓ Key material – tutorial, guides , demos
  - ✓ Human support staff to resolve problems raised by consumers
- Over life of API, all these are bound to change as interface, implementation or instance going to change
  - ✓ Greater will be cost when large number of supporting assets are generated
  - ✓ Can be done independently – like changing look and feel of documentation pages
  - ✓ Has greater impact on user experience but no impact on other API elements

Reference:  
Continuous API management  
By Medjaoui, Wilde, Mitra, Amundsen



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# API and its management

Chandan Ravandur N

# State of APIs

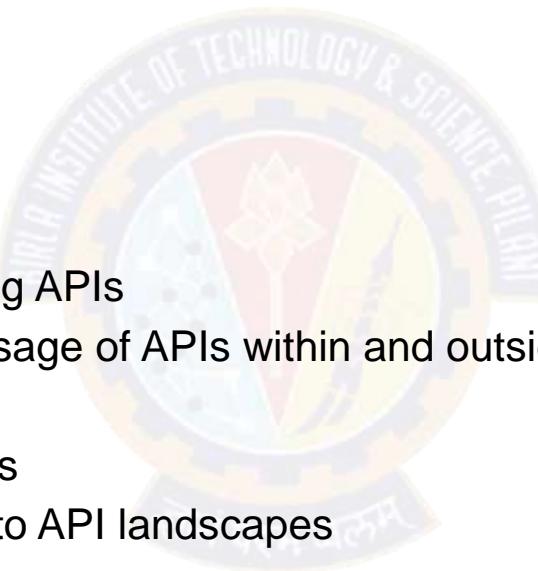
- Per Coleman Parkes survey
  - ✓ 9 out of 10 global enterprises have some API program
  - ✓ Organizations are seeing benefits from these programs
  - ✓ Time-to-market is one of notable benefit
- Only half of companies uses full-fledge API management program!
- Difficult to get knowledge out from companies successfully implementing API programs
  - ✓ Its hidden deep inside the companies work – hard to extract out
  - ✓ People are too busy to share the knowledge
  - ✓ Companies are sceptic to share it – as they gain competitive advantage out of it
  - ✓ Information shared is not easy to consume directly

# What is an API?

- Interface or implementations or both ?
  - ✓ Sometimes people refer it as contract between two parties – interface
  - ✓ Sometimes talks about exact functionality of APIs – implementations
  - ✓ Sometimes mentions various versions of same APIs – instances
- For example,
  - ✓ Consider User On boarding Scenario
    - New user can be created, then updated
    - Existing users can be edited
    - Account status can be changed
  - ✓ Interface defines what can be done with User entity – create / update / delete /get details
  - ✓ Implementation involves concrete code using programming language to carry out the action
  - ✓ Instance defines the combination of both – running instance of interface and implementation- actually executes the action

# API Management

- Involves just more than governing
  - ✓ Design
  - ✓ Implementation
  - ✓ Release of APIs
- Involves
  - ✓ Dealing with ecosystem surrounding APIs
  - ✓ Taking decision about controlling usage of APIs within and outside organization
  - ✓ Managing security of APIs
  - ✓ Maintaining the versions of the APIs
  - ✓ Process of moving existing APIs into API landscapes



# More than just API

- API involves technical details like interface , implementation and instance
  - ✓ Design – Build – Deploy are critical in lifecycle of API
  - ✓ But its just a part of whole API game!
- Managing involves
  - ✓ Testing
  - ✓ Documenting
  - ✓ Publishing
  - ✓ Learning to use
  - ✓ Securing
  - ✓ Monitoring the usage
  - ✓ Maintaining APIs over the lifetime
- All these are additional pillars that all API program managers need to consider!
  - ✓ Can be scaled from one API to multiple easily



# API Maturity Stages

- APIs undergoes a cycle – moves through several stages during life!
  - ✓ Design – Build – Publish – Maintain - Retire
- Important to understand the whole journey of APIs to
  - ✓ Determine where exactly you are standing on
  - ✓ Understand how much more time and resources needs to be allocated
  - ✓ Prepare for and respond to varying needs of time and investment at each stage
- Not all pillars have same significance in each stage
  - ✓ In early stages – design and build aspects are important – less focus on documentations, security
  - ✓ In later stages – monitoring , securing gains attention
- Understanding these stages and pillars helps to allocate resources for maximum advantage

# Not only a single API

- Organizations who has seen benefits in APIs are going to them more often
  - ✓ Just Like a tiger who has tested the blood
- Slowly focus will moved from a single API to the thousands of APIs
  - ✓ AKA API landscapes
- Things starts changing when moving from API to API landscape – both in terms of scope and scale
- More focus is on
  - ✓ details of how these APIs coexist in ecosystem
  - ✓ How to assure same level of consistency across all APIs without much complexity
- When API program grows, just does not get bigger , also changes in shape
  - ✓ Needs to deal with scope and scale of both while managing APIs

Reference:  
Continuous API management  
By Medjaoui, Wilde, Mitra, Amundsen



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Challenge of API management

Chandan Ravandur N

# Challenges

- Only 50% of companies have investments into proper API management platforms
  - ✓ What challenges they faced? How they overcame it?
  - ✓ Will provide essential knowledge when deciding upon API management solution
- Significant changes that happen when moving from a single API to API landscape
  - ✓ Scope
  - ✓ Scale
  - ✓ Standardization
- Need to take them into consideration before moving further!

# Scope

## How to set proper level of control when APIs grows in number?

- As API program changes, control over the APIs starts loosing
- Initially, teams have better control over design, implementations
  - ✓ Have freedom and are responsible for taking decisions on their own
  - ✓ Resulted into “API Best practices” or “Guidelines”
- As teams started increasing, the problem they are addressing with API also starts varying a lot
  - ✓ Difficult to maintain consistency across all API products
  - ✓ Some teams because of their requirement may not be able to adhere to the guidelines
- Certainly some common guidelines needs to be adhered to but not all
  - ✓ Over the period requirement changes, scenarios changes , a lot of evolution happens
  - ✓ Tools, Practices, Guidelines also needs to react to the evolution as the scope changes!

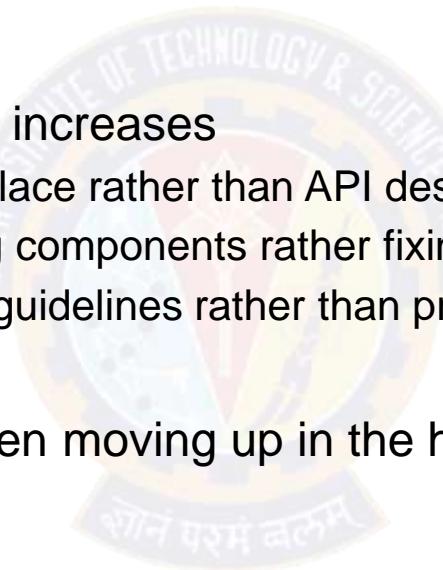
# Scale

## How to deal with changes in scale over time?

- One of the biggest challenge for health of API program
- As the system matures – moves from single API to API landscape
  - ✓ Processes to monitor and manage APIs at runtime are needed
  - ✓ Tooling varies a lot for both of cases
- API landscapes comes with their own set of challenges
  - ✓ When need to scale the processes, tooling needs to change appropriately to handle the change
  - ✓ More APIs means more interaction between apps and APIs as well as among APIs as well
  - ✓ Growing complexity will easily go out control and result into unexpected behaviour
  - ✓ Needs to focus on fixing / resolving / mitigating these errors

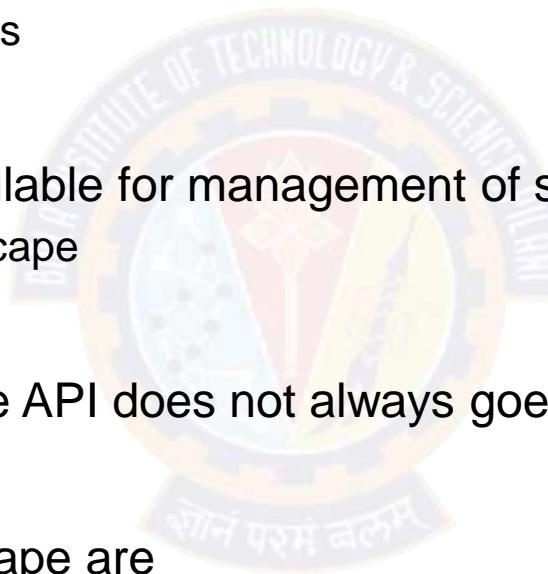
# Standardization

- Power of standards can not be ignored when moving from API to Landscape
  - ✓ Provides consistency among the various APIs and their implementations
- With growing APIs – coordination cost increases
  - ✓ Needs to have general standards in place rather than API design constraints
  - ✓ Standardize the coordination's among components rather fixing them with constraints
  - ✓ More emphasis on providing general guidelines rather than providing implementation details
- Can be tough initially but essential when moving up in the hierarchy of APIs



# Managing API landscape

- Two key challenges in API management space
  - ✓ Managing life cycle of a single API
  - ✓ Managing landscape of all the APIs
- Much guidance / knowledge is available for management of single API
  - ✓ But not is the case with API landscape
- Things which works well with single API does not always goes well for Landscape of APIs
- Unique challenges with API landscape are
  - ✓ Scaling technology
  - ✓ Scaling teams
  - ✓ Scaling governance



Reference:  
Continuous API management  
By Medjaoui, Wilde, Mitra, Amundsen



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# API Product lifecycle

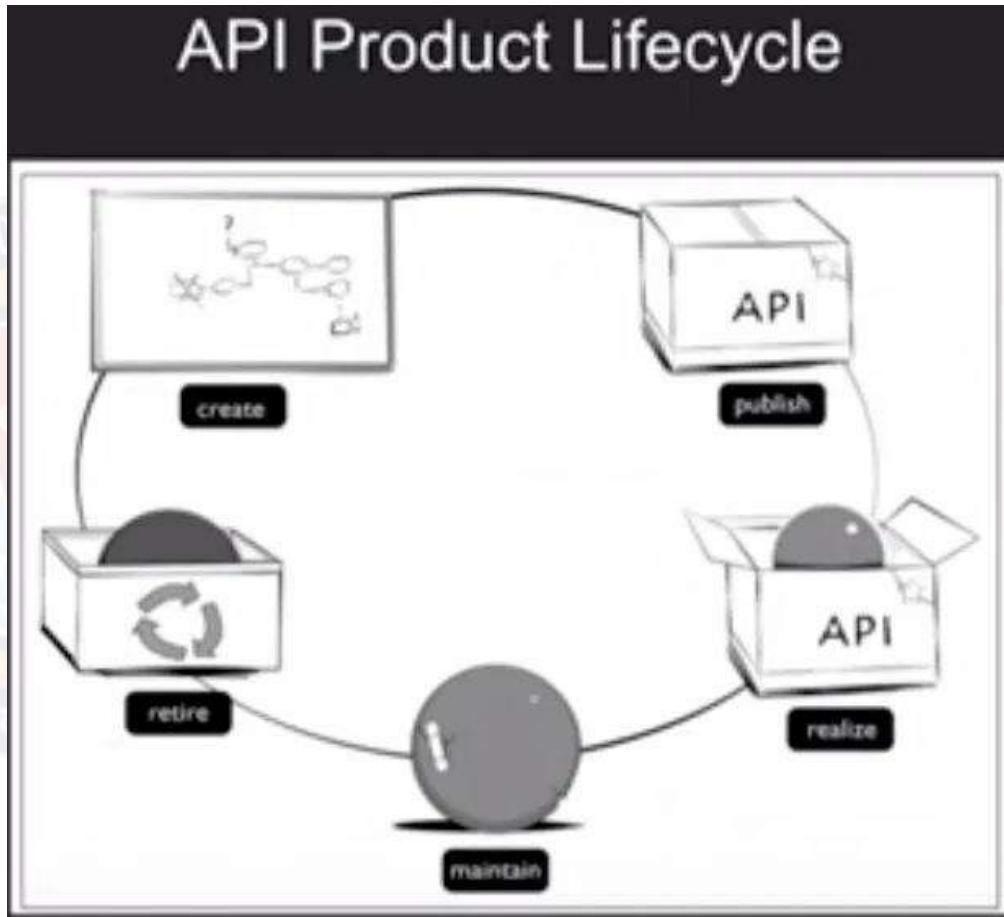
Chandan Ravandur N

# API maturity

- There are situations when need to invest lots of time and effort to get an API product ready for consumption
  - ✓ taking shortcuts or skipping critical steps will likely to result in poorly designed, hard-to-use APIs
  - ✓ that fail to realize their intended value
- At other times, mature APIs that are returning consistent value need to be “left alone”
  - ✓ No need to spend valuable time and resources making changes to an already successful API
  - ✓ the product should be optimized and tweaked in small ways
  - ✓ to maximize its value for the company without requiring substantial investment
- Finally, like all products, APIs will eventually outlive their usefulness and deserve to be granted an end-of-life plan
  - ✓ eventually removed from the system
  - ✓ Not all APIs reach their intended usefulness and no API lasts forever
  - ✓ Removing unused APIs from system will eliminate clutter, improve maintenance efforts
  - ✓ and often reduce infrastructure costs

# Five stages of the API product lifecycle

- Similar to any product lifecycle
  - ✓ API products also have lifecycle stages
- Five stages
  - ✓ Create
  - ✓ Publish
  - ✓ Realize
  - ✓ Maintain
  - ✓ Retire
- Product lifecycle is superset of release cycle
  - ✓ Stage many undergo many releases
  - ✓ Release does not result into stage change always



Continuous API Management

# Create

- It is important to have an active design process for API products
  - ✓ one that includes a business rationale for the API itself
  - ✓ Designing, implementing, and maintaining APIs in production is a costly endeavor
  - ✓ Doing all that work without a clear use case (and way to measure success) can lead to needless expense
- At the start, APIs go through a design, prototyping, and test phase
  - ✓ APIs might even be “complete,” but they haven’t been released into production
  - ✓ are not considered available for wide use
- While in the create phase,
  - ✓ the interface is likely to change
  - ✓ exhibit errors
  - ✓ and experience periods of non-responsiveness while work is being carried out to handle the edge cases
- Characteristics
  - ✓ A new API or a replacement of API that no longer exists
  - ✓ Has not been deployed in a production environment
  - ✓ Has not been made available for reliable use

# Publish

- The API is placed into production and made available to one or more developer communities for use
  - ✓ considered reliable, the interface is stable
  - ✓ the proper security and scalability elements are in place
- Publishing an API is more than just pushing it onto production servers
- Making an API public likely means investing in
  - ✓ documentation
  - ✓ training courses
  - ✓ sales materials
  - ✓ and even staffing up support forums and online communities
- Wide adoption of the API can hinge on whether the publishing effort was adequate for the target community
- Characteristics
  - ✓ API instance has been deployed to a production environment
  - ✓ Made available to one or more developer communities
  - ✓ Strategic value of API is not yet being realized

# Realize

- Goes into the phase focused on realizing the initial reason that product was created
  - ✓ might change the interface to better meet goals,
  - ✓ tweak performance, availability, scalability, etc.
  - ✓ All these changes are driven by the desire to realize the initial purpose of the product
- Good APIs solve a problem
- A good API program keeps track of just how well those APIs are doing at solving their stated problems
- The work of realizing an API is
  - ✓ actually the work of “proving” that the API product is doing a good job
  - ✓ solving the problem it was designed to address
- That means need a clear way to track the APIs behavior
  - ✓ compare that to the planned results documented during the create phase
- Characteristics
  - ✓ Published API instance exists and is available
  - ✓ Being used in a way that realizes its objective
  - ✓ Realized value is generally trending upward

# Maintain

- Assuming API meets our intended goal
  - ✓ it is realizing its planned value
  - ✓ can eventually place it into “maintenance mode”
- Should focus on getting the most value from the product without investing a great deal in changes to it
  - ✓ may do some minor optimizations, look for ways to reduce operational costs, etc.
  - ✓ but it is important to resist making any major changes or costly investments to the product
- Essentially, just counting the repeated incoming revenue (or cost savings) the API was designed to produce
- It is important to identify when the API reaches the maintain phase
  - ✓ since that will affect how you judge whether it is worth it to invest more time and effort into changing it
  - ✓ Placing APIs into maintenance mode means you can dial back on adding new features and focus on improving performance, reducing operational costs, and maximizing return
- Characteristics
  - ✓ Being actively used by one or more consuming applications
  - ✓ Realized value is stagnant or downward trending
  - ✓ No longer actively being improved

# Retire

- Finally, as the realized value starts to wane, as the operational costs outweigh the revenue/saving benefits
  - ✓ time to place the product into retirement
- May begin a program to migrate remaining users of this product to a new, more valuable, alternative
  - ✓ or if the service is no longer needed, simply work out an end-of-life plan for the API
  - ✓ begin the “wind-down” process
- Retiring an API frees up resources (infrastructure, staff, support, etc.)
  - ✓ and helps to clear out little-used parts of the ecosystem
- Too easy to forget about actively managing the retirement phase of APIs
  - ✓ But ignoring unused APIs can lead to needless costs in infrastructure and support
  - ✓ means have less money for new, important API products in the future
- Characteristics
  - ✓ Published API instance exists and is available
  - ✓ Realization value is no longer enough to justify continued maintenance
  - ✓ End of life decision has been made

Reference:  
Continuous API management  
By Medjaoui, Wilde, Mitra, Amundsen

Five stages of the API product lifecycle  
Mike Amundsen



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Product Pillars to Lifecycle Mapping

Chandan Ravandur N

# API Pillars

- Each of the API pillar forms a boundary for a work domain
  - ✓ Defines works needs to be done to build and maintain API product
  - ✓ Need to put time and effort in understanding and managing these foundation blocks
  - ✓ Some pillars can be stronger than others in certain phases of product life cycle
- Ten pillars of API
  - ✓ Strategy
  - ✓ Design
  - ✓ Documentation
  - ✓ Development
  - ✓ Testing
  - ✓ Deployment
  - ✓ Security
  - ✓ Monitoring
  - ✓ Discovery
  - ✓ Change Management



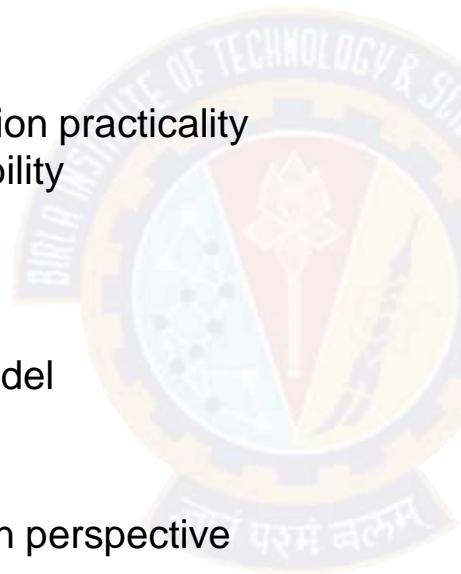
# Applying product pillars to Lifecycle

- API product lifecycle is useful way of understanding maturity of API product
  - ✓ Helpful in estimating changeability cost of API in each stage
  - ✓ Helps to manage the work that needs to be carried out for APIs
- Ten pillars have significance in every lifecycle stage
  - ✓ Will not be able to ignore them
  - ✓ Not every pillar needs to be considered in every stage of API
  - ✓ Certainly some pillar will dominate a phase as compared to the others
  - ✓ Provides pointers where focus should be kept during that stage of API

# Stage 1 : Create

**Focus is on developing the best API model before releasing to users**

- Special focus on strategy, design, development, testing and security work
- Strategy
  - ✓ Design initial strategy
  - ✓ Test strategy for design and implementation practicality
  - ✓ Update goals and tactics based on feasibility
- Design
  - ✓ Design initial interface model
  - ✓ Test the design from user perspective
  - ✓ Validate implement ability of interface model
- Development
  - ✓ Develop prototypes
  - ✓ Test interface design from implementation perspective
  - ✓ Develop initial implementation of API
- Testing
  - ✓ Define and execute testing strategy for interface model
  - ✓ Define strategy for implementation testing
- Security
  - ✓ Define security requirements
  - ✓ Validate interface design from security perspective

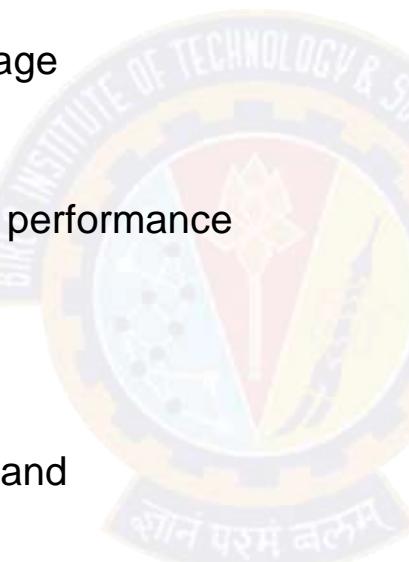


	Create
Strategy	✓
Design	✓
Development	✓
Deployment	
Documentation	
Testing	✓
Security	✓
Monitoring	
Discovery	
Change Management	

# Stage 2 : Publish

## Door opening for API product – makes API available for consumers

- Focus is on design, development, deployment, documentation, monitoring and discovery
- Design
  - ✓ Analyse the usability of interface
  - ✓ Test design assumptions made in create stage
  - ✓ Improve interface model based on findings
- Development
  - ✓ Optimize implementation for scalability and performance
  - ✓ Optimize implementation for changeability
- Deployment
  - ✓ Deploy API instance
  - ✓ Focus on making API available
  - ✓ Plan and design deployment for future demand
- Documentation
  - ✓ Publish documentation
  - ✓ Improved docs based on actual usage
- Monitoring
  - ✓ Design and implement strategic measures for API
  - ✓ Build monitoring system that can be used during realization
- Discovery
  - ✓ Invest in marketing, engagement and findability of API

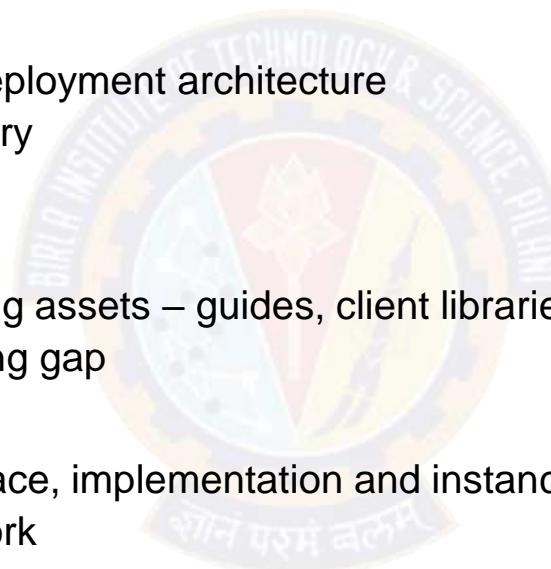


	Publish
Strategy	
Design	✓
Development	✓
Deployment	✓
Documentation	✓
Testing	
Security	
Monitoring	✓
Discovery	✓
Change Management	

# Stage 3 : Realize

## Goal of API product to reach to this stage – increase value from API

- Focus is on deployment, documentation, testing, discovery and change management
- Deployment
  - ✓ Making sure instances are available
  - ✓ Continually improve and optimize deployment architecture
  - ✓ Improve implementation as necessary
- Documentation
  - ✓ Continue to improve docs
  - ✓ Experiment with additional supporting assets – guides, client libraries, blogs, demos etc.
  - ✓ Drive new usage by reducing learning gap
- Testing
  - ✓ Implement testing strategy for interface, implementation and instance changes
  - ✓ Continually improve testing framework
- Discovery
  - ✓ Continue to invest in API marketability, engagement and findability
  - ✓ Invest more in high value user communities
- Change management
  - ✓ Design and implement change management system
  - ✓ Carefully communicate changes to users, maintainers and stakeholders

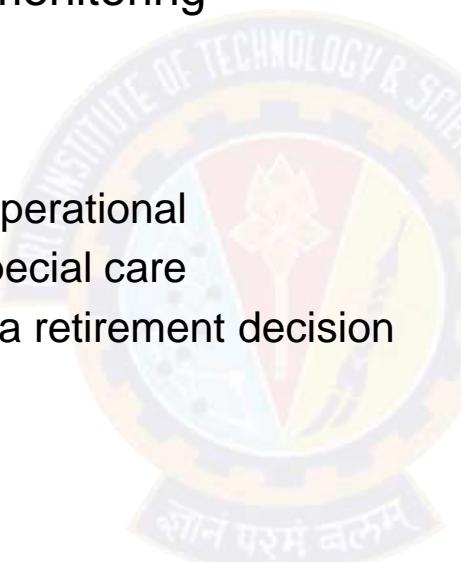


	Realize
Strategy	
Design	
Development	
Deployment	✓
Documentation	✓
Testing	✓
Security	
Monitoring	
Discovery	✓
Change Management	✓

# Stage 4 : Maintain

**Not getting any new value from API, but don't want to harm existing users**

- Keep the engine running and maintain it
- Most important work is involved in monitoring
- Monitoring
  - ✓ Ensure that monitoring system is operational
  - ✓ Identify patterns that will require special care
  - ✓ Observe metrics that could trigger a retirement decision

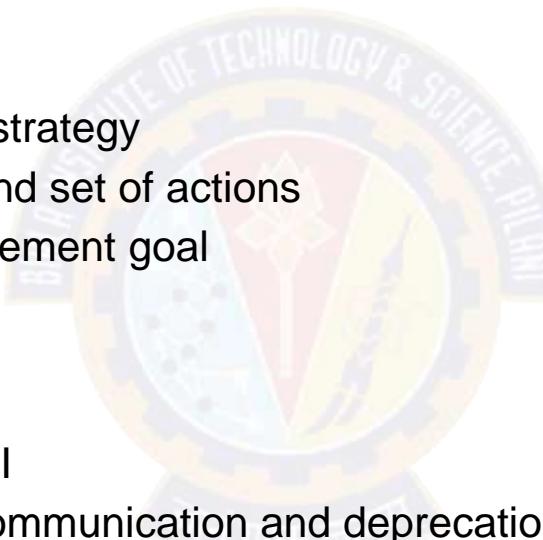


	Maintain
Strategy	
Design	
Development	
Deployment	
Documentation	
Testing	
Security	
Monitoring	✓
Discovery	
Change Management	

# Stage 5 : Retire

## API not yet gone – needs to be deprecated in planned manner

- Most important pillars are strategy and change management
- Strategy
  - ✓ Define a retirement (or transition) strategy
  - ✓ Identify a new goal, tactical plan and set of actions
  - ✓ Measure progress toward this retirement goal
- Change management
  - ✓ Assess the impact of retiring of API
  - ✓ Design and implement a plan of communication and deprecation
  - ✓ Manage implementation and instance changes to support that deprecation



	Retire
Strategy	✓
Design	
Development	
Deployment	
Documentation	
Testing	
Security	
Monitoring	
Discovery	
Change Management	✓

# Summarized

	Create	Publish	Realize	Maintain	Retire
Strategy	✓				✓
Design	✓	✓			
Development	✓	✓			
Deployment		✓	✓		
Documentation		✓		✓	
Testing	✓		✓		
Security	✓				
Monitoring		✓			✓
Discovery		✓	✓		
Change Management			✓		✓

From Continuous API Management

Reference:

Continuous API management

By Medjaoui, Wilde, Mitra, Amundsen

Five stages of the API product lifecycle

Mike Amundsen



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# API - Business Roles

Chandan Ravandur N

# API Teams

- “Becoming a software architect is not something that simply happens overnight or with a promotion. It’s a role, not a rank.” – Simon Brown
- Same is applicable for API teams
  - ✓ No standardization across the organizations, teams
  - ✓ Every organization has its own way of organizing divisions, products, services and teams
  - ✓ Makes it hard to come up with recommendations for API teams
- But some general practices, patterns can be discussed
  - ✓ Titles / Designations may vary from company to company
  - ✓ But Roles are fairly consistent across companies – handling a task
  - ✓ No matter what titles people have, same kind of work needs to be done

# API Teams(2)

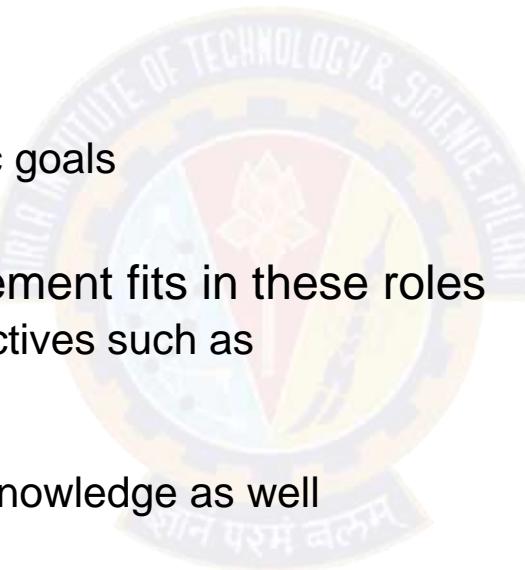
- Exact makeup of team varies based on API maturity stage
  - ✓ In create stage, no need to focus on QA or DevOps
  - ✓ In maintain stage, no need to focus on frontend developers
- Mix of API roles will be often realized in organizations
- Engineering the engineers
  - ✓ Teams needs to interact with each other
  - ✓ Team of teams helps to keep all teams , manage interoperability, encourages collaboration

# API Roles

- Common roles needs to be defined for the API teams
  - ✓ Can be considered as Set of job titles – but that's not standardized
  - ✓ Program manager / API owner / Product owner – are used for same role definition
- Whatever is title – needs to define scope of that role
  - ✓ Work of associating those roles can be done easily
  - ✓ Quite possible, not all roles will be applicable in one organization
- Two types of roles –
  - ✓ Business roles – tending more towards meeting business objectives
  - ✓ Technical roles – tending more towards meeting technical objectives

# API – Business Roles

- Primarily focused on business side of APIs
- Often have responsibility of
  - ✓ speaking customers voice
  - ✓ Aligning product with clear strategic goals
- Mostly people from Product management fits in these roles
  - ✓ Have more focus on business objectives such as
    - ❖ Promoting new products
    - ❖ Improving sell-through etc.
  - ✓ But some roles requires technical knowledge as well
- Example Roles
  - ✓ API product manager
  - ✓ API designer
  - ✓ API technical writer
  - ✓ API evangelist
  - ✓ Developer Relations



# API Roles - Defined

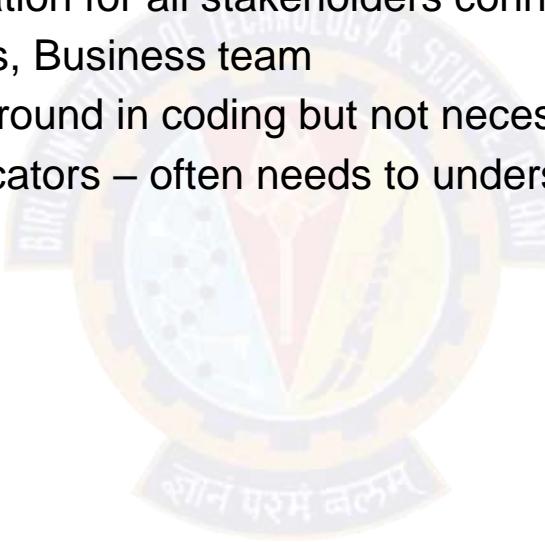
- API Product Manager
  - ✓ Product manager / owner is main owner and contact point for API product and team
  - ✓ Responsible for making sure that all moving parts are going together in one direction
  - ✓ Defines and sets objectives for the team to achieve it within specified boundaries
  - ✓ Defines “what” for the rest of teams – what needs to be done and monitors it
  - ✓ Needs to ensure that developer experience is delivered as per necessity of API consumers
  
- API Designer
  - ✓ First line of contact for consumers and may become the voice of the customers
  - ✓ Needs to understand how API is going to consumer and based on upon that one design the interface
    - ❖ Making sure its functional, useful and providing necessary developer experience
  - ✓ Needs to interact with technical teams to see API design can be translated into reality with technology in hand
  - ✓ Needs to ensure that overall API design matches with Company wide style guidelines

# API Roles – Defined(2)

- API Evangelist
  - ✓ Responsible for promoting and supporting API practices in organization
    - ❖ Largely suitable for big organizations where interactions can be quite cumbersome
  - ✓ Ensures that all the internal API developers understand and accomplish their goals for API
  - ✓ Listens to customer voice and provides feedback to rest of teams
  - ✓ May create samples, demos, training materials and supporting activities to maximize dev experience
- Developer Relations / DevRel / Developer Advocate
  - ✓ Focused on external use of APIs – outside the organization
  - ✓ Responsible for creating samples, demos, training materials and other assets to promote use of APIs
  - ✓ Listens to customer voice and provides feedback to rest of teams
  - ✓ Also tasked with selling API product to a wider audience by
    - ❖ Participate in customer onsite, presales activities and ongoing product support
    - ❖ Speaking at public event, writing blogs or articles on how to use product
    - ❖ Brand awareness activities to reach set goals for the teams

# API Roles – Defined(3)

- API Technical Writer
  - ✓ Closely associated with API product owner and designer
  - ✓ Responsible for writing documentation for all stakeholders connected with product
  - ✓ API consumers, API internal teams, Business team
  - ✓ Many of them has technical background in coding but not necessary
  - ✓ Required to be effective communicators – often needs to understand views of API providers and consumers



Reference:  
Continuous API management  
By Medjaoui, Wilde, Mitra, Amundsen



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# API - Technical Roles

Chandan Ravandur N

# API - Technical Roles

- Responsible for “how” part of APIs
  - ✓ Focused on technical details of actually implementing the APIs design, testing and deployment
  - ✓ Followed by maintenance throughout the lifecycle of API
  - ✓ Responsible for achieving KPIs as well as helping business staff reach their objectives
- Usual roles
  - ✓ API Architect
  - ✓ Lead API Engineer
  - ✓ Frontend Developer
  - ✓ Backend Developer
  - ✓ Test/QA Engineer
  - ✓ DevOps Engineer



# API - Technical Roles - Defined

- API Architect
  - ✓ Advocates overall software architecture and design for overall organization
  - ✓ Responsible for the architecture used for API product allowing easy interaction with the systems
  - ✓ Taking following aspects into consideration
    - ❖ Security,
    - ❖ Stability and reliability metrics
    - ❖ Protocol and I/O format specifications
    - ❖ Scalability etc.



# API - Technical Roles – Defined (2)

- API Lead Engineer / API tech lead (TL)
  - ✓ Technically equivalent to product manager role
  - ✓ Key person for all work related to development , testing, monitoring, deployment of API
  - ✓ PM is responsible for “what” part of product – TL is responsible for “how” part of product
  - ✓ Knows technical details of what it takes to build, deploy and maintain an API
  - ✓ Also coordinates with other technical members of teams and other teams
- Frontend Developer / Engineer (FE)
  - ✓ Responsible for making sure APIs offers quality consumer experience
  - ✓ Helps to implement API registry, consumer portal
- Backend Developer / Engineer (BE)
  - ✓ Implements vision of PM and Designer of what API should do and how it should do it
  - ✓ Responsible for implementing the actual interface of APIs, connecting it to other services
  - ✓ Needs to make sure that API is reliable, stable and consistent in production scenarios

# API - Technical Roles – Defined (3)

- Test/QA Engineer
  - ✓ Takes care of validation of design and checking out functionalities, safety and stability of APIs
  - ✓ Writes actual test cases, running them effectively and efficiently
  - ✓ Makes use of test frameworks, automation tools
- DevOps Engineer
  - Deals with every aspect of building and deployment of API including
    - ✓ API performance monitoring
    - ✓ Working on delivery pipeline tooling, build scripts and artifacts
    - ✓ Maintaining several environments for desktop, build, staging and production etc.
    - ✓ Supporting releases , roll back broken releases
  - Maintains real time dashboard for production release related data, problems / issues faced

Reference:  
Continuous API management  
By Medjaoui, Wilde, Mitra, Amundsen



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# API Teams and Maturity

Chandan Ravandur N

# API Teams and Maturity

- Business and technical roles – both needs to be present as part of API teams
  - ✓ Defines scopes of responsibility of each role in API life cycle
- One person can assume more than one roles
  - ✓ API evangelist and DevRel or QA engineer and DevOps engineers role in small team
- Not necessarily all the roles to be present in all the API teams
  - ✓ In each stage of API lifecycle , some roles will have more presence as compared to others
  - ✓ Also known as – primary and secondary roles in each stage
  - ✓ Primary roles are responsible for the work that needs to be done for API
  - ✓ In publish stage – DevOps role has to take ownership of it

# Stage 1 : Create

- Primary Role(s)
  - ✓ Product Manager, designer, API lead

- Secondary Role(s)
  - ✓ API evangelist, DevOps, API architect, backend developer

- Primary Activities

• Activity	• Role
Develop the strategy	Product Manager
Design Interface model	Designer
Develop initial implementation	API architect, API lead, backend developer

- Secondary Activities

• Activity	• Role
Develop Prototypes	API lead, Backend developer
Test the implement ability, security of design	API architect, lead, developers, QA engineer
Test usability , marketability of design	API evangelist, DevRel, Designer

# Stage 2 : Publish

- Primary Role(s)
  - ✓ Product Manager, API technical Writer, DevOps
- Secondary Role(s)
  - ✓ Frontend / backend developer, Designer, API evangelist, DevRel
- Primary Activities

• Activity	• Role
Write and publish documentation	Technical Writer
Design deployment architecture and deploy APIs	DevOps
Publish API ( make it officially discoverable)	Product Manager

- Secondary Activities

• Activity	• Role
Design and implement a portal	Frontend developer
Market API, gather feedback	API evangelist , DevRel
Improve interface design	Designer
Collect usage information from deployed instances	API lead, DevOps
Improve and optimize the implementation	API lead, backend developer

# Stage 3 : Realize

- Primary Role(s)
  - ✓ Product Manager, DevOps
- Secondary Role(s)
  - ✓ Designer, Test Engineers, Architect, Lead, Front/backend developers, tech writer, DevRel, API evangelist
- Primary Activities

• Activity	• Role
Improve and optimize deployment architecture	DevOps
Manage and prioritize changes	Product Manager

- Secondary Activities

• Activity	• Role
Improve interface design	Designer
Improve and optimize tests	Test/QA Engineer
Improve and optimize implementations	API architect, lead, backend developer
Test security of implementation and deployment	API architect, test / QA engineer
Improve and optimize onboarding and learning experience	Frontend developer, tech writer, DevRel
Market the API	API evangelist, DevRel

# Stage 4 : Maintain

- Primary Role(s)
  - ✓ DevOps, DevRel, API architect
- Secondary Role(s)
  - ✓ Product Manager, API lead, backend developer
- Primary Activities

• Activity	• Role
Support existing users	DevRel
Identify system changes that will deteriorate API quality	API architect

- Secondary Activities

• Activity	• Role
Plan and schedule implementation changes	Product Manager
Make required implementation change	API lead, backend developer
Make required deployment changes	DevOps, backend developer

# Stage 5 : Retire

- Primary Role(s)
  - ✓ Product Manager
- Secondary Role(s)
  - ✓ DevRel, API evangelist, API architect, DevOps, API lead
- Primary Activities

• Activity	• Role
Develop a retirement strategy	Product manager

- Secondary Activities

• Activity	• Role
Communicate retirement plan and help users transition	DevRel, API evangelist, Technical Writer
Design technical retirement strategy	API architect, API lead
Update deployment architecture and remove instances gracefully	DevOps, API lead

Reference:  
Continuous API management  
By Medjaoui, Wilde, Mitra, Amundsen



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Anatomy of an API management solution

Chandan Ravandur N

# Need for API management solution

- Customers of every business are engaging with companies on a variety of devices and channels
  - ✓ including in-store, web, smartphones, tablets, laptops, and even connected devices IoT
- Customer and developer expectations are increasingly driving enterprise IT
  - ✓ to employ new approaches serving the needs of a diverse mix of users and experiences
- APIs are the foundation upon which digital business is built
  - ✓ allowing app developers to create apps that can serve the needs of a specific segment of users
- With the explosion of apps and experiences required in the digital world, and new customer-centric IT organizations,
  - ✓ companies across industries need better solutions than ever to manage their APIs and API-driven businesses
- API management enables you to create, manage, secure, analyze, and scale APIs
  - ✓ IT organizations are moving toward more efficient, agile development frameworks for external and internal use

# Anatomy of an API management solution

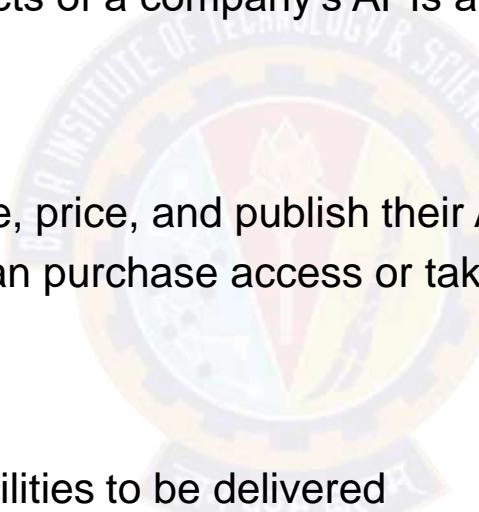
**An API management solution needs to include at least the following capabilities:**

- Developer portal
  - ✓ to attract and engage application developers
  - ✓ enabling them to discover, explore, purchase (or profit from)
  - ✓ to test APIs and register to access and use the APIs
- API gateway
  - ✓ to secure and mediate the traffic between
    - ❖ clients and backends,
    - ❖ between a company's APIs and the developers,
    - ❖ customers, partners, and employees who use the APIs
- API lifecycle management
  - ✓ to manage the process of designing, developing, publishing, deploying, and versioning APIs

# Anatomy of an API management solution(2)

**More sophisticated API management provides additional capabilities including:**

- An analytics engine
  - ✓ provides insights for business owners, operational administrators, and application developers
  - ✓ enabling them to manage all aspects of a company's APIs and API programs
- API monetization
  - ✓ to enable API providers to package, price, and publish their APIs
  - ✓ so that partners and developers can purchase access or take part in revenue sharing
- Provision
  - ✓ typical for API management capabilities to be delivered
    - ❖ in the cloud as a SaaS (Software as a Service) solution
    - ❖ or on premises in a private cloud,
    - ❖ or sometimes using a hybrid approach



apigee

Reference:

The Definitive Guide to API Management  
apigee ebook



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# API Platform - Essentials

Chandan Ravandur N

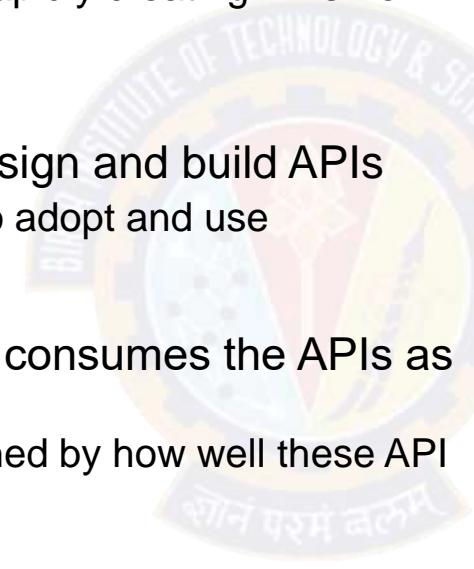
# API lifecycle management

- The core API management capability is API lifecycle management
- Involves
  - ✓ consideration for both the API provider and the API consumer (most often the app developer)
- API providers manage the processes for
  - ✓ designing,
  - ✓ developing,
  - ✓ publishing,
  - ✓ deploying,
  - ✓ versioning,
  - ✓ governance,
  - ✓ monitoring availability,
  - ✓ and measuring performance
- API consumers
  - ✓ discover new APIs,
  - ✓ understand versioning and API updates,
  - ✓ easily register for access to APIs,
  - ✓ test and register apps built against the APIs,
  - ✓ and communicate and collaborate with other developers and the API provider



# Design and develop APIs that developers love

- API management enables API developers, who expose assets via APIs
  - ✓ to unlock the value of business assets by rapidly creating APIs from existing data and services
- API management provides the ability to design and build APIs
  - ✓ that are intuitive and easy for developers to adopt and use
- An API's job is to make the developer who consumes the APIs as successful as possible
  - ✓ The success of an API program is determined by how well these API consumers adopt the APIs
- The developer is the lynchpin of the entire API strategy
  - ✓ Any API management solution needs to help API providers see from the developer's perspective when designing and building APIs
  - ✓ that are easy to use and follow best practices
  - ✓ will ultimately maximize the productivity of the developers who build on the API



apigee

# API lifecycle management - Capabilities

**Key capabilities of the management product in this area include:**

- Security
  - ✓ lets you protect APIs, messages, and backends with configurable policies
  - ✓ such as OAuth, API key verification, XML/JSON threat protection, access control etc.
- Protocol transformation
  - ✓ enables the transformation of enterprise data and services into usable, scalable, and secure APIs
  - ✓ Configurable policies include SOAP to REST, XML to JSON, JSON to XML, and XSL Transformation
- Support for Java, JavaScript, Node.js, and Python
  - ✓ extend the programmability of the API management solution for developers who prefer coding over configuration
- Versioning
  - ✓ supported at multiple levels
  - ✓ Backend service versions can be “hidden” behind the API facade
  - ✓ Versioning can be applied at the URI level, following best practices and internal corporate standards

# Publish APIs and enable developer productivity

## Developer and partner productivity depends on an efficient onboarding experience

- A key capability of any API management solution is a developer portal
  - ✓ enabling companies to provide everything that internal, partner, and third-party developers need be effective and productive building on the APIs
- Enables an API provider to deliver an enhanced developer and community experience
  - ✓ that accelerates API adoption, simplifies learning, and increases the business value of APIs
- The best developer portals provide a complete, self-service developer experience
  - ✓ enable developers to
  - ✓ register their applications
  - ✓ select the APIs and the service levels they need
  - ✓ get secure access
  - ✓ monitor their API usage
  - ✓ and even monetize and participate in revenue sharing with the API provider

# Publish APIs and enable developer productivity(2)

## Documentation

- The ability to provide documentation and a developer feedback mechanism
  - ✓ is an important consideration when publishing API products
- Developer portals with social publishing features are increasingly being used for communicating static content
  - ✓ such as interactive API documentation and terms-of-use
  - ✓ as well as dynamic community-contributed content, such as blogs and forums, as well as customer support features



# API traffic management

## API gateway

- API management solution should enable to manage the API traffic generated by the apps
  - ✓ that developers and partners have built against them
- Traffic management capabilities include:
  - ✓ Caching
    - ❖ to improve API and app performance by storing data from backend resources in a cache
    - ❖ from where they can be retrieved quickly
  - ✓ Quotas and rate limits
    - ❖ to limit the number of connections apps can make via the API to the backend
  - ✓ Spike arrest capabilities
    - ❖ to protect backend systems against severe traffic spikes and denial-of-service attacks

Reference:

The Definitive Guide to API Management  
apigee ebook



# Thank You!

In our next session:



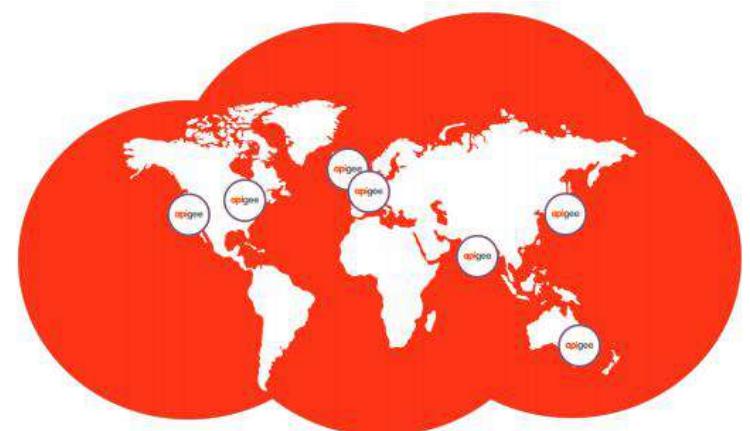
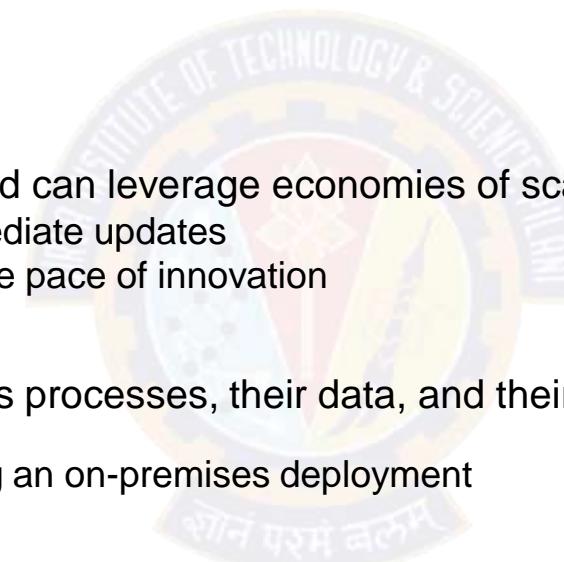
**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# API Platform - Other Aspects

Chandan Ravandur N

# Deployment

- It is typical for API management capabilities to be delivered
  - ✓ in the cloud
  - ✓ or on premises (in a private cloud)
  - ✓ or sometimes using a hybrid approach
- A cloud deployment aggregates many users and can leverage economies of scale
  - ✓ provides the ability to do seamless and immediate updates
  - ✓ results into improving responsiveness and the pace of innovation
- Some companies still require that their business processes, their data, and their customers' data
  - ✓ be controlled within their enterprise, requiring an on-premises deployment
- Geographical redundancy is important both for high availability and also for latency and performance considerations
- API management solutions typically support a multi-region, multi-datacenter deployment
  - ✓ ensures the highest level of availability and distribution



apigee

# API Security

- API management solutions secure and mediate the traffic between
  - ✓ a company's APIs and the developers, customers, partners, and employees who use those APIs
- Digital business is built on the design principle that internal and external users will use the system
  - ✓ OAuth is one of the most widely used forms of authentication for consumer or partner-facing apps
- Security is foundational to API infrastructure
  - ✓ both from the APIs to the backend services and from the API to the apps — across the entire digital value chain
  - ✓ critical for this core function to be easily configurable and to enable security at all points of engagement
- Most modern apps require some social component
  - ✓ API management solutions that provide third-party sign-in can improve user experience
  - ✓ while increasing adoption, giving the API provider access to valuable information from social networks and services
- Auditing and compliance processes dictate that RBAC (Role Based Access Control) be supported by enterprise platforms
  - ✓ allowing for an audit trail and administrative accountability
  - ✓ aids in the software development life cycle (SDLC) by limiting the potential for one team's work to interfere with the work of another team.

# API Security (2)

## At different stages



# API Analytics

- Robust API management tools empower businesses to answer questions like
  - ✓ How is your API traffic trending over time?
  - ✓ Who are your top developers?
  - ✓ When is API response time fastest? Slowest?
  - ✓ Are you attracting more developers?
  - ✓ Geographically where do you see the most API traffic?
- Helps enterprises improve their APIs, attract the right app developers, troubleshoot problems
  - ✓ ultimately, make better business decisions related to the API program
- API management solutions typically provide the
  - ✓ visualization tools
  - ✓ dashboards
  - ✓ and reportsto help measure a broad spectrum of data that flows across APIs
- This information is most useful in today's dynamic API economy
  - ✓ when it is gathered, analyzed, and provided to the business in real time
  - ✓ both the ops and business teams should be able to gain deep visibility into the performance of the API program

# API monetization

- Digital assets and services that provide value to customers, partners, and end users
  - ✓ can be a source of revenue
- Companies can charge for data or services as part of their business model
  - ✓ or they can share revenue with partner companies and developers
- For example,
  - ✓ Content providers can offer valuable content such as maps and images that partners and developers will pay to access
- In the pre-API world, these transactions were done via contractual processes, data sharing agreements
  - ✓ or, in some instances, the data was given away for free
- With APIs, a company can make data and services available to front-end applications and partners
  - ✓ in an easy and scalable manner while tracking usage and billing in real time
- API Monetization Services, provides the following capabilities:
  - ✓ Rate plans
  - ✓ Reporting and billing
  - ✓ Setting limits

Reference:

The Definitive Guide to API Management  
apigee ebook



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# API Platform Alternatives

Chandan Ravandur N

# Magic Quadrant for Full Life Cycle API Management

- API usage, and the need to govern it, is increasing rapidly
  - ✓ Driven by remote working, platforms, ecosystems, innovations, digital transformations and regulations
- Application leaders needs to find the right API life cycle management vendor for their organization's needs



# Market Definition/Description

- Gartner defines the full life cycle API management market as the market for offerings that support the
  - ✓ planning,
  - ✓ design,
  - ✓ implementation,
  - ✓ testing,
  - ✓ publication,
  - ✓ operation,
  - ✓ consumption,
  - ✓ versioning
  - ✓ and retirement of APIs
- Includes:
  - ✓ Developer portals for targeting, marketing to, and governing ecosystems of developers who produce and consume APIs
  - ✓ API gateways for runtime management, security and gathering of usage data
  - ✓ Policies for operational management, security, format translation
  - ✓ Analytics to collect technical metrics associated with the usage of APIs
- For full life cycle API management, following four functional categories are considered as core
  - ✓ Planning and initial design
  - ✓ Implementation and testing
  - ✓ Deploy and run (basic)
  - ✓ Versioning and retirement



# Google (Apigee)

- Google (Apigee) is a Leader in this Magic Quadrant
  - ✓ in the last iteration of this research it was also a Leader
- The core Apigee API management platform is available for
  - ✓ public cloud, private cloud or data center and hybrid (customer-managed runtime and Google-managed control plane) deployment
  - ✓ also provides Apigee Sense (for bot protection), Apigee API Monetization and Apigee Advanced API Ops
- Google's roadmap for Apigee includes
  - ✓ delivering a fully managed API platform for multicloud and hybrid deployments
  - ✓ building ecosystems of citizen developers
  - ✓ integrating with marketplaces
  - ✓ and extending Google technologies such as artificial intelligence (AI) and machine learning (ML) to API management
- Most Google (Apigee) clients are located in the U.S., Europe, Australia, New Zealand, India and Southeast Asia
  - ✓ The Apigee team markets its offering as a cross-cloud API platform and as a platform for digital business



# MuleSoft

- MuleSoft is a Leader in this Magic Quadrant
  - ✓ in the last iteration of this research it was also a Leader
- MuleSoft, which Salesforce acquired in 2018, offers the Anypoint Platform as its full life cycle API management offering
  - ✓ which combines API management and integration capabilities in a single platform
  - ✓ A packaged option providing only API management is also available
- In 2019, MuleSoft introduced Anypoint API Community Manager
  - ✓ to create and grow an ecosystem of API consumers and drive adoption of API products
- MuleSoft sells its platform both directly and through an ecosystem of partners
  - ✓ has midsize and large customers worldwide



- Software AG is a Leader in this Magic Quadrant
  - ✓ in the last iteration of this research it was also a Leader
- Its offerings for full life cycle API management are
  - ✓ the webMethods API management platform (for on-premises deployment)
  - ✓ the webMethods.io API (for the cloud)
- The API management platform and its cloud equivalent comprise gateway, portal and microgateway
  - ✓ also offers CentraSite (a registry and repository), the webMethods CloudStreams connection framework, a service mesh and a cloud-based engagement platform
- Plans to focus more strongly on microservices and a service mesh, the developer user experience and multicloud
  - ✓ sells to large companies and government clients in Europe, North America, Australia and Japan



# Kong

- Kong is a Leader in this Magic Quadrant
  - ✓ in the last iteration of this research it was a Visionary
- Kong's open-source API gateway for REST APIs is based on an NGINX proxy server with OpenResty
- Also includes
  - ✓ Kong Vitals (for monitoring the Kong platform itself)
  - ✓ Kong Brain and Kong Immunity, which use AI and ML to help automate the API and service development life cycle
  - ✓ Insomnia (an API design, documentation and testing tool)
- Kuma, an open-source project created by Kong, is a platform-agnostic control plane that uses Envoy
  - ✓ donated Kuma to the Cloud Native Computing Foundation (CNCF)
  - ✓ released Kong Mesh, an enterprise service mesh built on top of Kuma and Envoy
- Kong's solution is available both on-premises and in the cloud
  - ✓ Has a vision that its Service Control Platform will intelligently broker information between all services
- Kong has over 250 paying enterprise customer accounts in North America, Western Europe, Japan, Australia and Singapore



Reference:

The 2020 Gartner Magic Quadrant for Full Life Cycle API Management



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# API Analytics overview

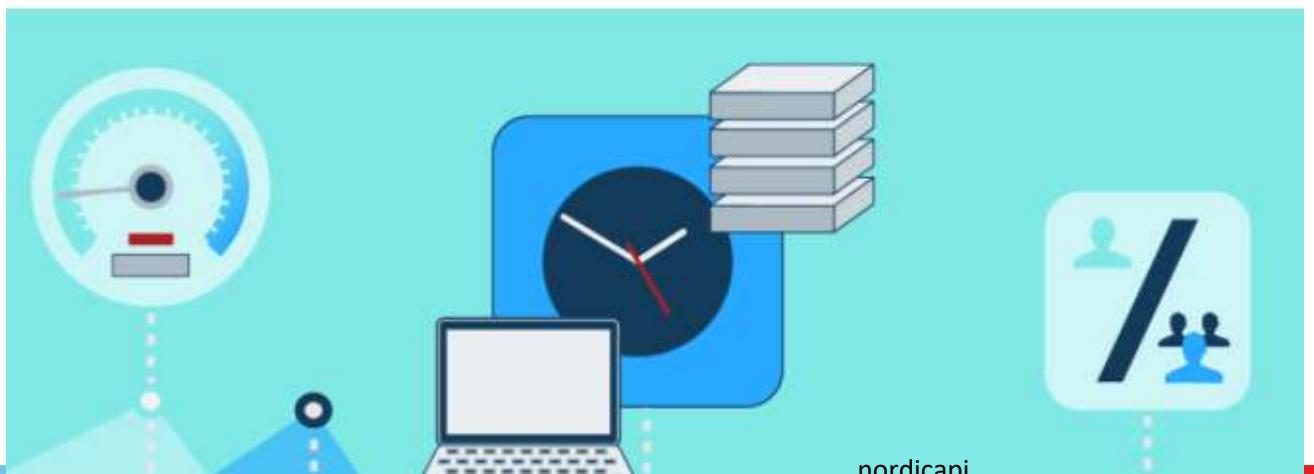
Chandan Ravandur N

# Analytics

- Per Wikipedia
- [Analytics] is used for the discovery, interpretation, and communication of meaningful patterns in data
- Analyzing data can determine the relationship between different parts of data and whether patterns exists
  - ✓ different ways to analyze data based on the type and origin
  - ✓ analyst has to choose a certain analysis to perform - preferably one that fits the type of data they have collected
- The two main types of data are qualitative or quantitative
  - ✓ Qualitative data can be an observation that is good, bad, colorful, tall, strong, etc.
  - ✓ Quantitative data are observed as numbers
- API analytics are all the ways that we can record qualitative and quantitative data from our API requests to be used in a meaningful way

# API Analytics and Monitoring

- APIs are enabling entirely new business models such as
  - ✓ API as a Product, developer platforms and ecosystems, and new partner opportunities
- However if companies want to out-innovate competition and quickly grow API platform,
  - ✓ need the right data to make informed decisions
- API Analytics are not just valuable for engineering teams
  - ✓ but across the organization including product owners, customer relations, marketing, and sales teams
- API Analytics and Monitoring includes both engineering focused metrics such as performance and uptime
  - ✓ but also tracking customer and product metrics such as engagement, retention, and developer conversion
- A variety of methods to perform such analysis - includes basic SQL and Excel to purpose built API analytics platforms

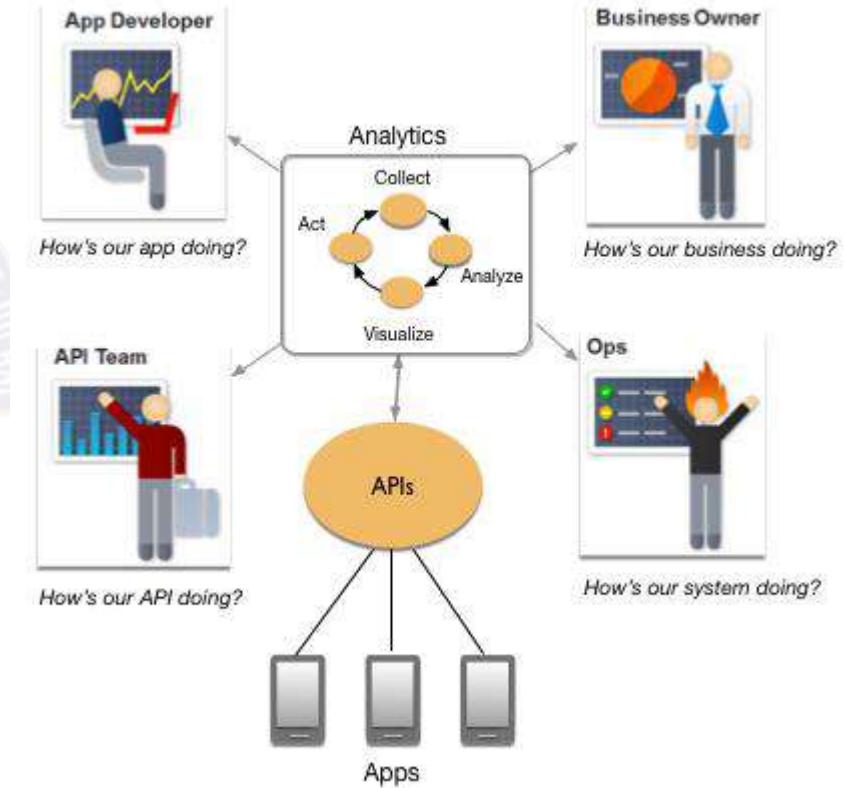
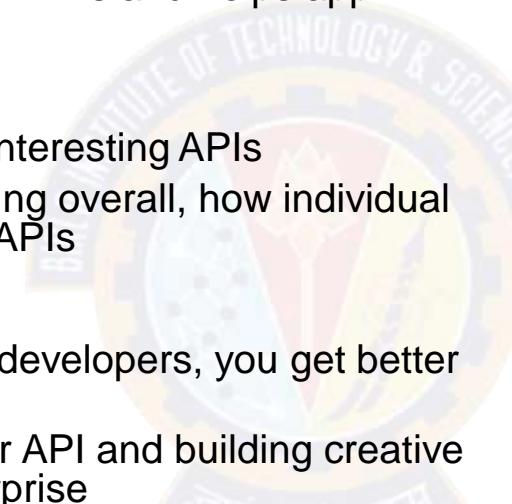


# Why Collect Qualitative Analytics?

- A quantitative analyst may measure the dimensions of the paintings and record numbers
- A qualitative analyst at an art show may describe the quality of a painting by observing it and then writing down words
- Quantitative analysis
  - ✓ If wanted to know how long an API request took, could measure the latency
  - ✓ Latency is the time interval between events - usually measured in milliseconds (ms.)
- Qualitative analysis
  - ✓ An example of qualitative data in API analyses is the HTTP method of the request
  - ✓ can use qualitative data to help categorize quantitative data
- For example
  - ✓ User is complaining that a certain action on their app takes a long time
  - ✓ In analysis, notice that a POST request (a type of HTTP method) is sent when the user performs the action
  - ✓ that it has an abnormally high latency
  - ✓ the use of both qualitative and quantitative API analytics produced a meaningful discovery in the data

# API Analytics helps everyone improve

- Through a continual process of collecting, analyzing, and visualizing data
  - ✓ API Analytics helps API team improve their APIs and helps app developers improve their apps
- API Team
  - ✓ is tapping into internal systems to create interesting APIs
  - ✓ wants to know how the API program is doing overall, how individual APIs are doing, and how to improve their APIs
- App Developers
  - ✓ By sharing analytics information with app developers, you get better apps
  - ✓ These developers are innovating with your API and building creative apps that help drive revenue to your enterprise
  - ✓ Analytics help app developers know how their apps are doing and how much they are contributing to the bottom line of your enterprise
- Ops Team
  - ✓ wants to understand traffic patterns and anticipate when to add backend resources or make other critical adjustments
- Business Owner
  - ✓ wants to see how their API investment is paying off and where to invest API dollars in the future



Google apigee

# Defining Key API Metrics

- Each team needs to track different KPIs when it comes to APIs
- The API metrics important to infrastructure teams will be different than
  - ✓ what API metrics are important to API product or API platform teams
- Metrics can also be dependent on where the API is in the product lifecycle
  - ✓ API recently launched will focus more on improving design and usage
  - ✓ while sacrificing reliability and backwards compatibility
- API Analytics collects and analyzes a broad spectrum of data that flows across API proxies such as:
  - ✓ Response time
  - ✓ Request latency
  - ✓ Request size
  - ✓ Target errors
  - ✓ API product name
  - ✓ Developer email address
  - ✓ App name
  - ✓ Many others

Reference:

API Analytics overview  
by Google apigee

API Analytics and Monitoring: Best Practices to Grow API Platforms  
Moesif blog



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# API Metrics

Chandan Ravandur N

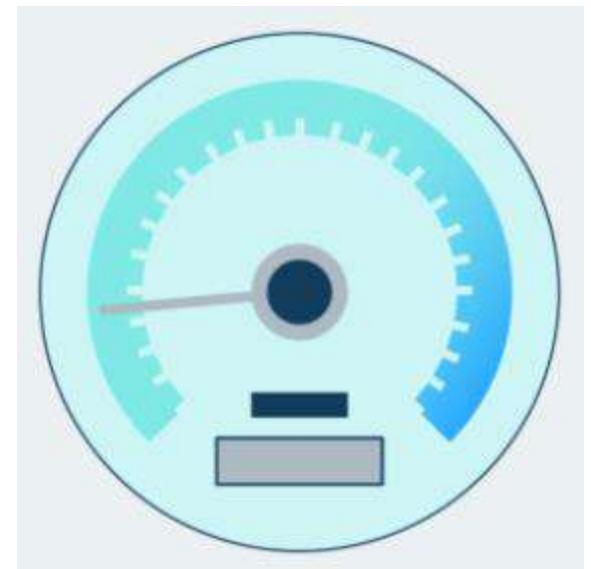
# API metrics

- Are perhaps the single most important factor in improving any API system
  - ✓ Metrics are inherently valuable – tracking data on API usage, availability, uptime, and other insights
  - ✓ is pivotal to keeping a consistently healthy platform
- An unfortunate truth that many developers do not leverage API analytics to their full power
- API metrics undoubtedly serve a very important business role in the modern API landscape
  - ✓ can be leveraged to greater heights, serving to amplify business choices and technical solutions
  - ✓ in a true approach towards platform empowerment
- Data is extremely valuable as it can be used to make an intricate system more transparent
  - ✓ While this is obviously a benefit to web API hosts, it can be hard to pinpoint what specific indicators should be monitored.

# Availability and Uptime

## Track API uptime and availability

- The most important metrics when it comes to something as living and high-demand as an API
  - ✓ Uptime describes whether the service is “on” or “off,”
  - ✓ Availability is a bit more nuanced, as it tracks how often the service has failed, for how long, and for what purpose
- Let’s imagine a server that is serving media data via an encoded stream
  - ✓ When discussing uptime, we would want to know a percentage of the time where the service was active
  - ✓ might be 99.999%, which is an absolutely monster metric
  - ✓ mean that in a year, the system was only down for a total of 5 minutes and 15.6 seconds
- That’s not really a useful metric, however – it only tells us the total amount of time in which the resource was technically unreachable
- Availability is more concerned with not only whether the resource was available or not
- but whether or not the access was full, unrestricted, and unhampered
  - ✓ While an API could have 99.999% availability simply by being able to call the API endpoint
  - ✓ if the authentication server is overwhelmed by connections and rejected 1 in 5 connections
  - ✓ in theory that uptime is maintained while the availability is terrible.
- Uptime is not everything
- Without uptime, availability cannot be determined – they must work in concert with one another.



nordicapi

# Responsiveness and Latency

## Discover latency issues to optimize API performance

- Adds a whole secondary layer to the concept of availability
  - ✓ by strictly considering whether or not data was easily intractable
  - ✓ and was indeed responsive to requests within a normal, specifically defined time set
- If the latency between call made, call responded to, and data sent is too high
  - ✓ responsiveness itself so astronomically low, the actual usability of an API is questionable
- Could have 99.999% uptime, and we could even have 99.999% effective call service, but if each call took 5 hours to respond to
  - ✓ then user experience is not what might be extrapolated from the availability and uptime metrics by themselves
- Responsiveness, however, does – especially when this data is broken out geographically
  - ✓ enables to identify failures in data processing per geographic location
  - ✓ then contextualizing this within the low and high responsive values can help contextualize the absolute values created by analyzing availability and uptime
- In other words, availability and uptime tell you if there's a problem – responsiveness and latency are cues into identifying why problems exist.



nordicapi

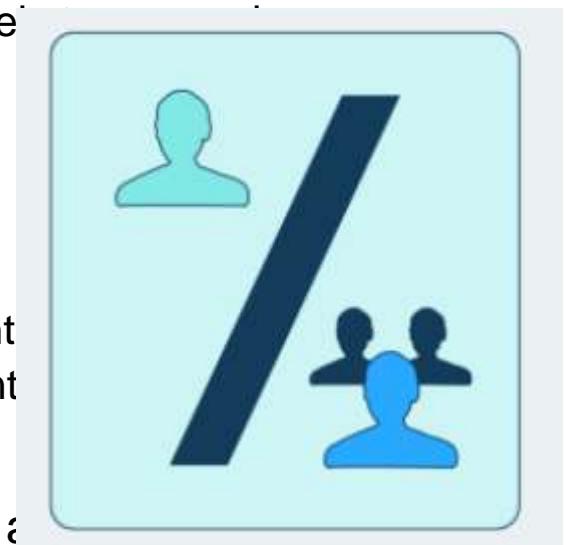
# Endpoint Valuations

- A hugely powerful metric is the KPI of endpoint valuation
  - ✓ really asking is this simple question – what's going on with the endpoints
  - ✓ a general question, and it entails a lot of data that is hugely valuable
- For endpoint analytics, consider finding answers to questions like:
  - Frequency
    - ✓ Do we have endpoints that are used more often than others?
    - ✓ At what rate are they used?
  - Utilization comparison
    - ✓ What are our least used endpoints, and do we have any data as to why they might be so under-utilized?
  - Traffic
    - ✓ What endpoints are hit the most with malicious traffic?
  - Vulnerabilities
    - ✓ When data breach does occur, or when penetration testing has shown vulnerability, what endpoints are responsible?
- The responses can inform providers about the health of an API in general, including platform insights such as:
  - Bloat
    - ✓ If we have a great number of unused endpoints, it suggests we are supporting large portions of codebase that is no longer needed
    - ✓ – this is unnecessary bloat
  - Service Efficiency
    - ✓ If we have one or two endpoints that are constantly hit, we need to look at them
    - ✓ If they cover multiple services, those services should be broken into their own endpoints for a true microservices approach
  - Security
    - ✓ If we have vulnerable endpoints, we have an issue in our codebase allowing for vulnerabilities to be easily detected and exploited

# Conversions

## User conversion is also an important metric, albeit business-related

- There are also business KPIs that inform us as to the API developer experience
  - ✓ One such data metric is the idea of conversions, or the rate at which a consumer takes a desired action
  - ✓ In the API space, it can be everything from registering for a premium account to submitting social information
  - ✓ The key concept in conversion is that if the user sees value in a product, they are more likely to convert
- Say we have an API that is streaming video content to a user
  - ✓ The API is called via an applet embedded in a JavaScript space on a webpage
  - ✓ serves data from a randomized server based on availability
  - ✓ The user has the ability to select the server of choice if they register for a premium account
  - ✓ or they can temporarily choose to register a credentialled account by sharing social account
- Overwhelming conversion data suggests that users are willing to register for the free account
  - ✓ but are unwilling to convert to a premium account
- Increase in user experience could result in not only greater premium account adoption (and thereby an increase in revenue), but, in theory, a better user experience in general



nordicapi

Reference:

Using API Analytics to Empower the Platform

Kristopher Sandoval, Nordic APIs



# Thank You!

In our next session: