



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Web Apps

Chandan Ravandur N

Website

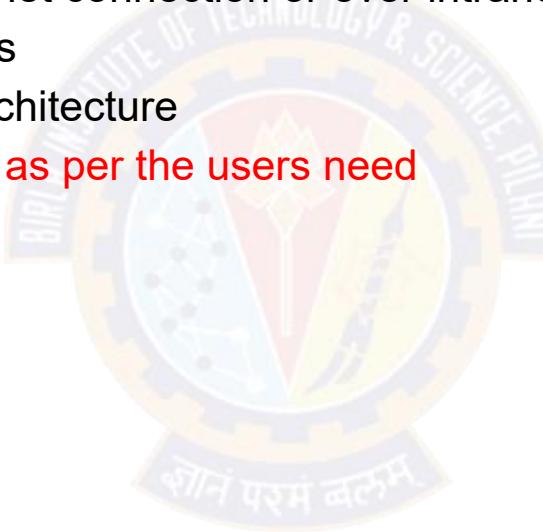
What & Why?

- 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

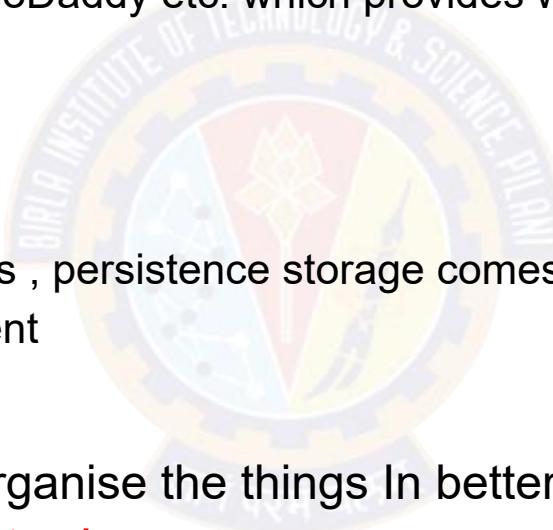
- Lets Visit!



Web Apps Architecture

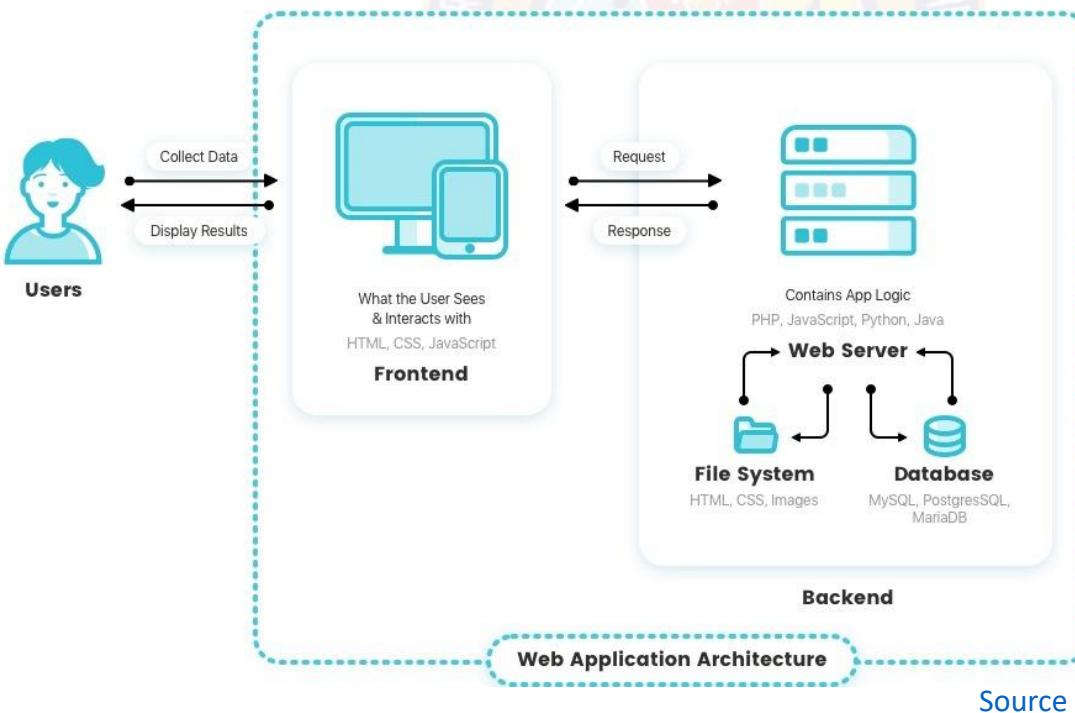
Why?

- 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

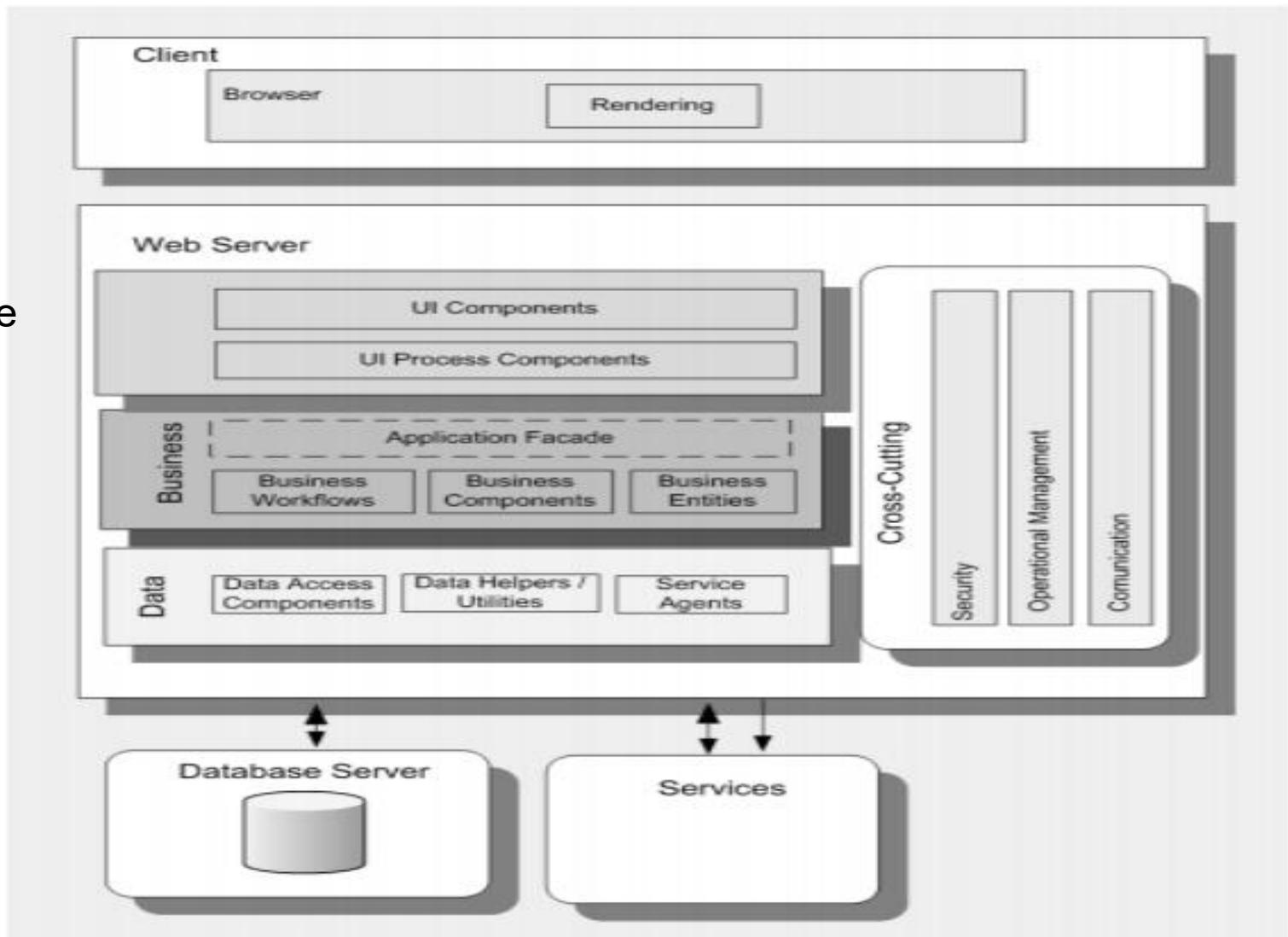
Web App Components

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

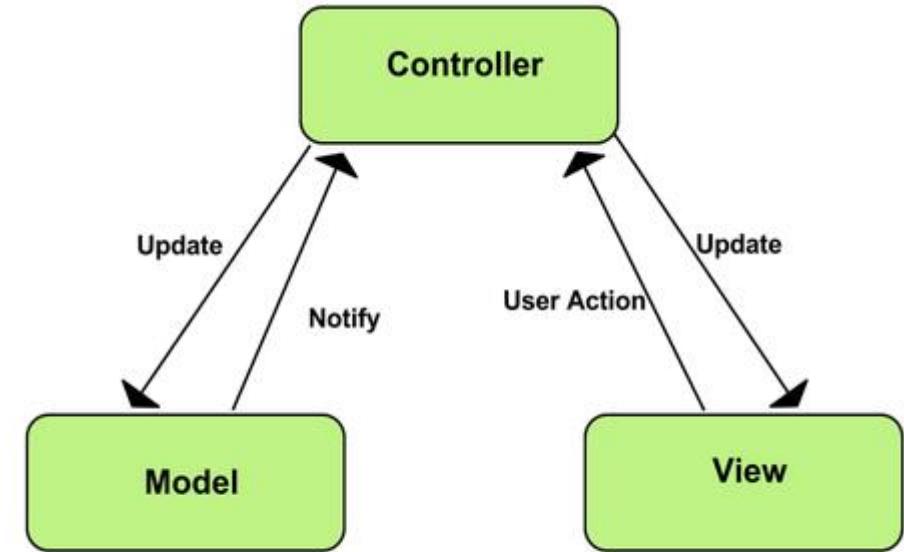
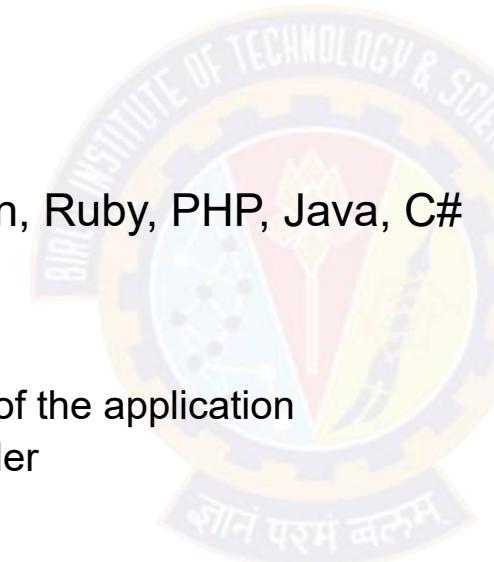
- The core of a Web application is
 - its server-side logic
- The Web application layer itself can be comprised of many distinct layers
 - Typically a three-layered architecture
 - comprised of
 - ❖ Presentation
 - ❖ Business
 - ❖ data layers



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

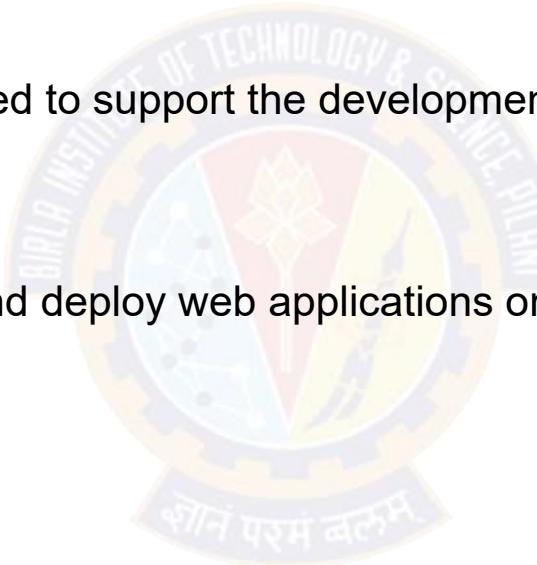
SSR Vs CSR

	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

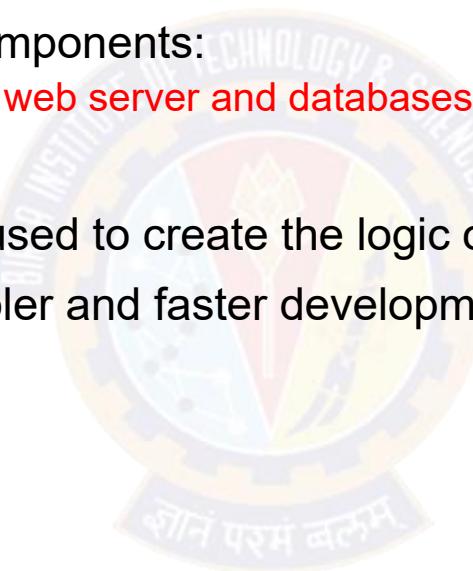
Frontend / User Interface

- 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

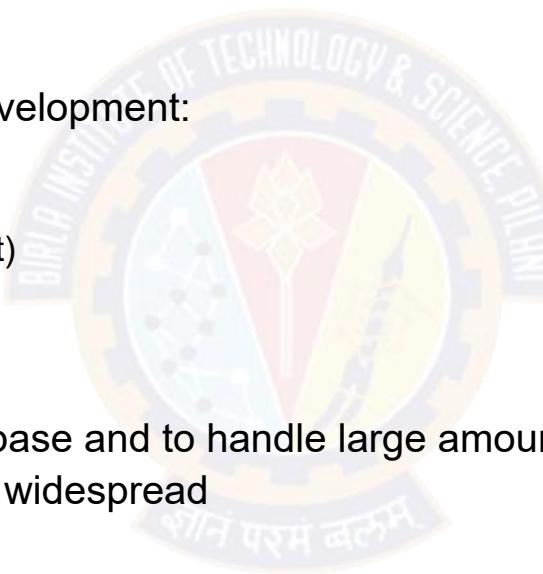
- Not visible to users, but it powers the client side
- Major server side technology stack components:
 - Programming language, Framework , web server and databases
- Server-side programming languages used to create the logic of applications
- Frameworks offer lots of tools for simpler and faster development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



Server-Side Programming

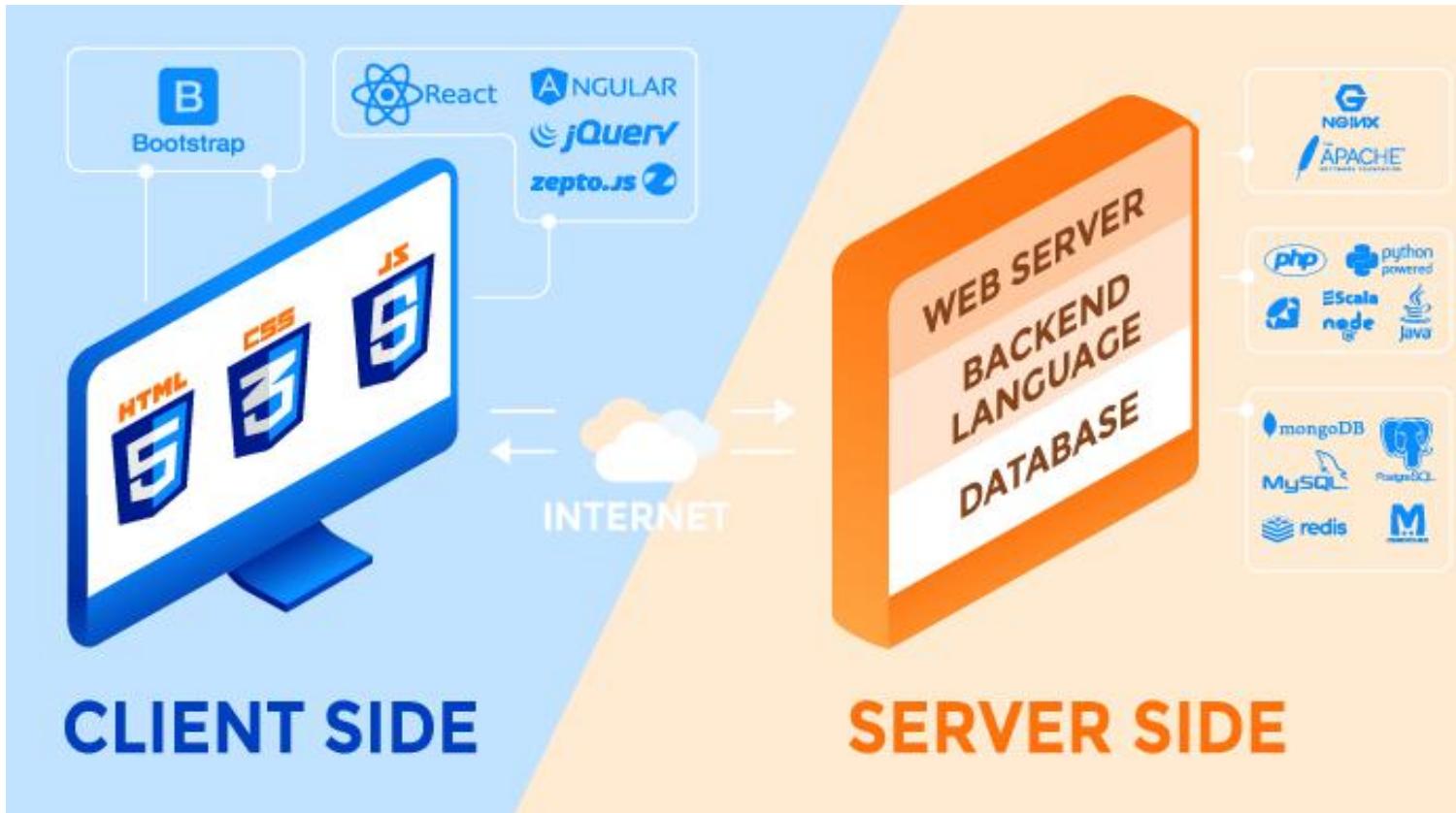
Other Components

- 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](#)

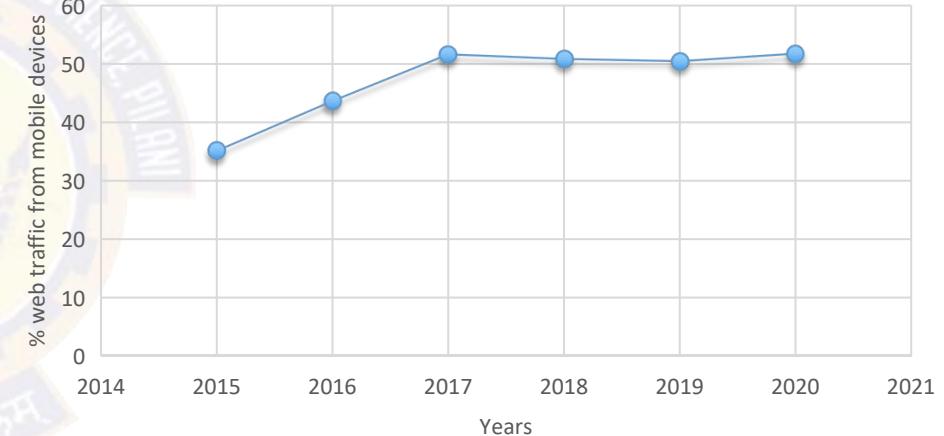
Mobile device website traffic

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 Web Traffic data



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

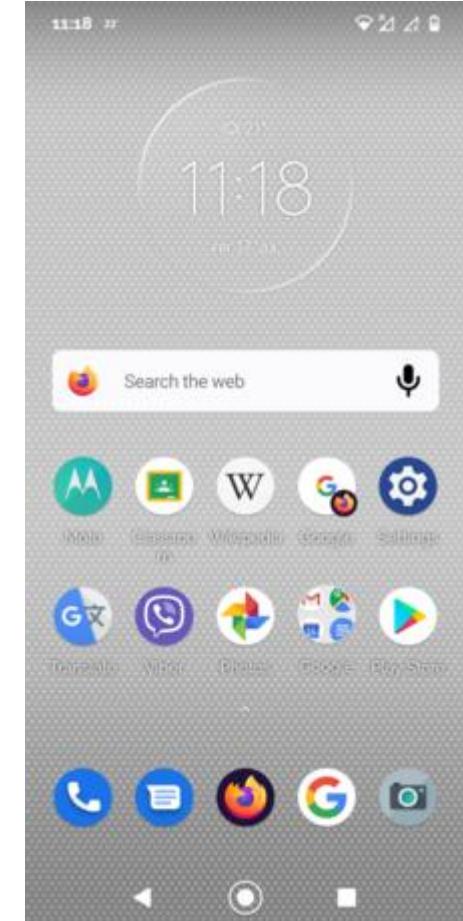


Image source : Wikipedia

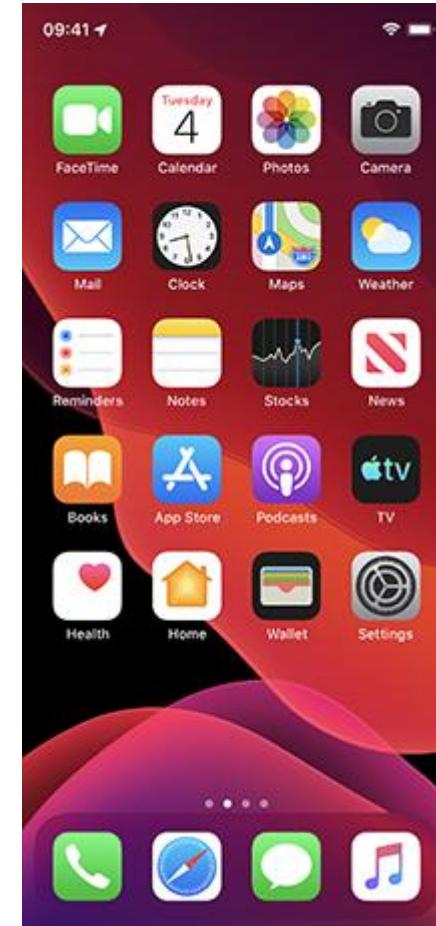
Mobile Apps

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



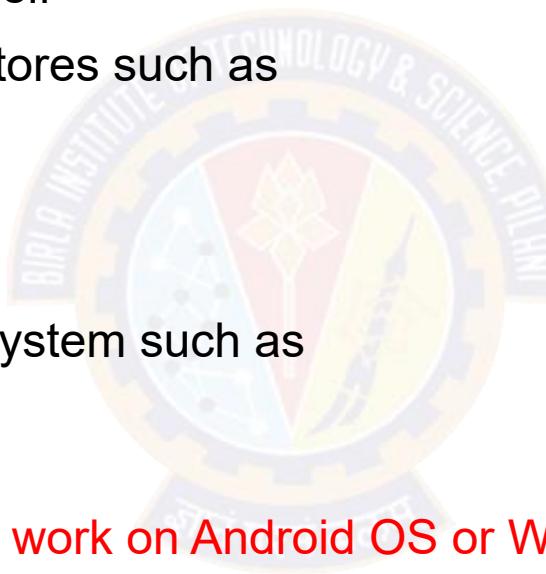
Mobile Apps - Types

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



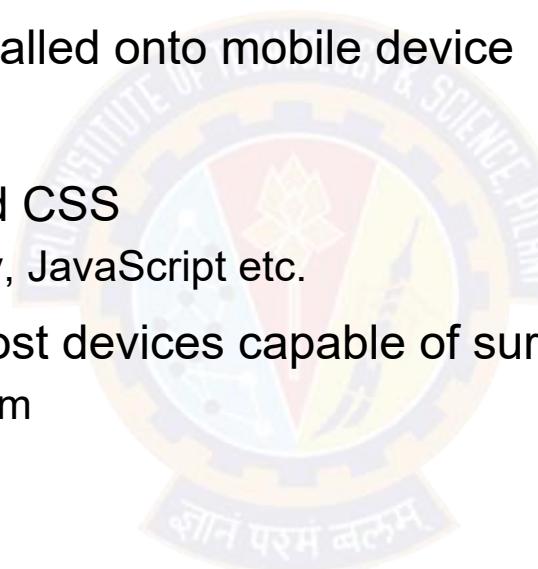
Native Apps

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

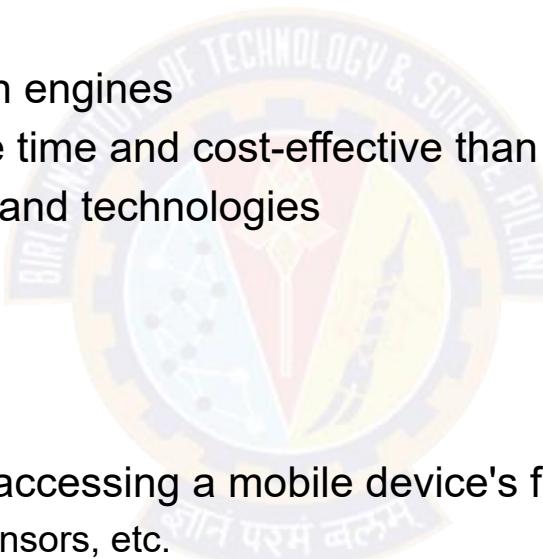


Web Apps

Pros and Cons

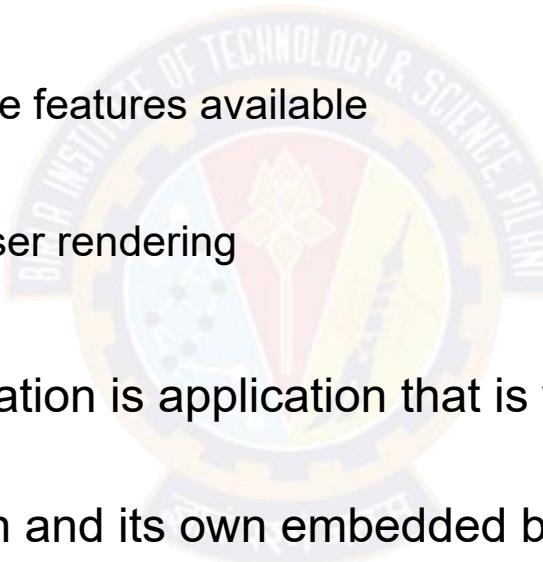
- Pros
 - Instantly accessible to users via a browser
 - Easier to update or maintain
 - Easily discoverable through search engines
 - Development is considerably more time and cost-effective than development of a native app
 - common programming languages and technologies
 - Much larger developer base.

- 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

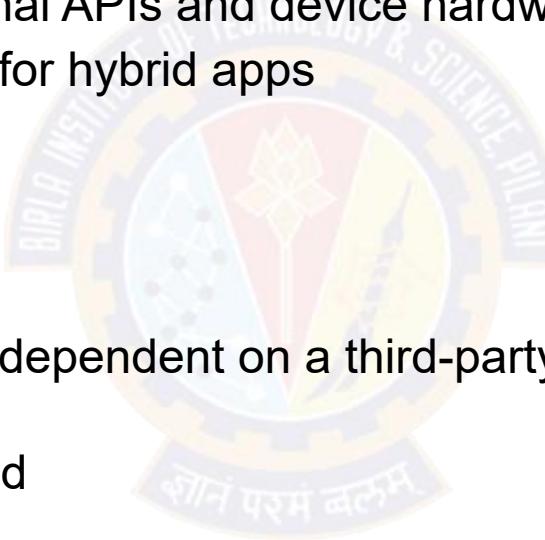
- Part native apps, part web apps
- Like native apps,
 - available in an app store
 - can take advantage of some device features available
- Like web apps,
 - Rely on HTML, CSS , JS for browser rendering
- The heart of a hybrid-mobile application is application that is written with HTML, CSS, and JavaScript!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

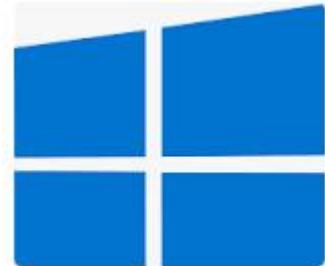
Computing Platform

- 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

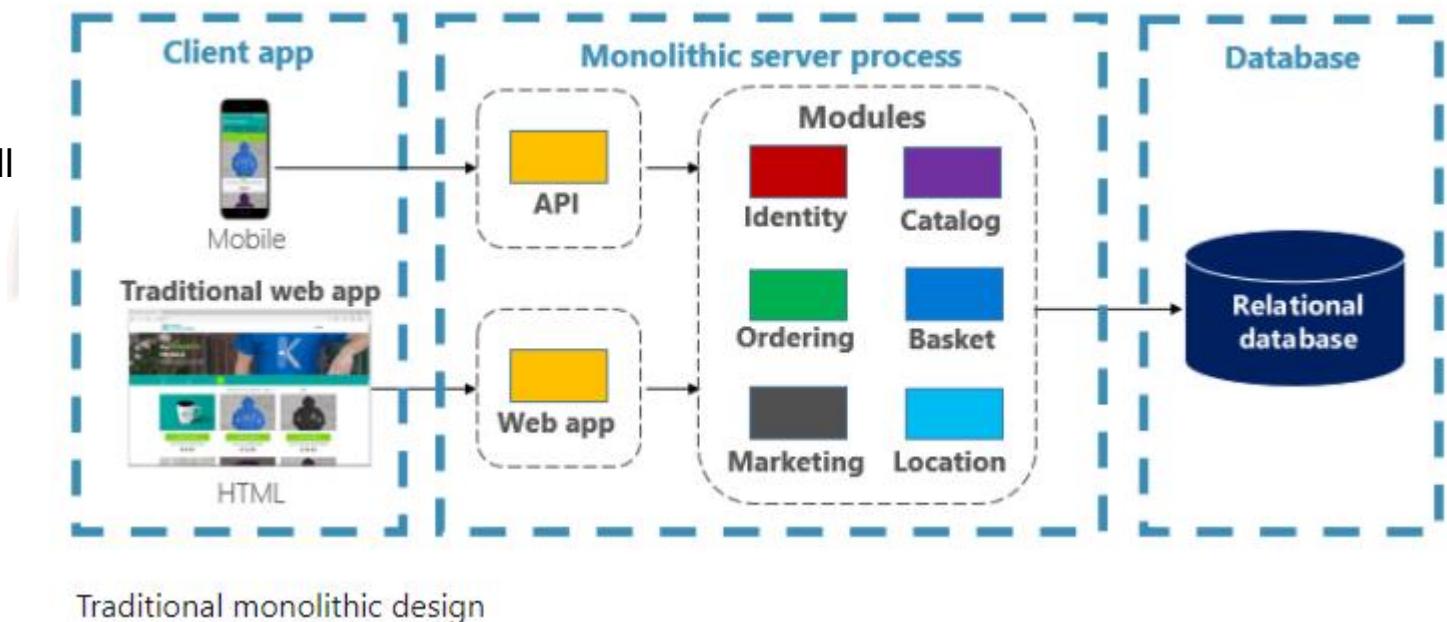
Disadvantages

- 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

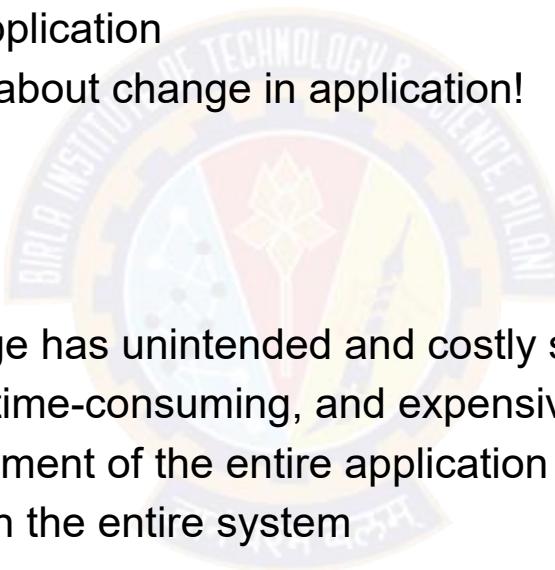
Conventional Layered Apps

- 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



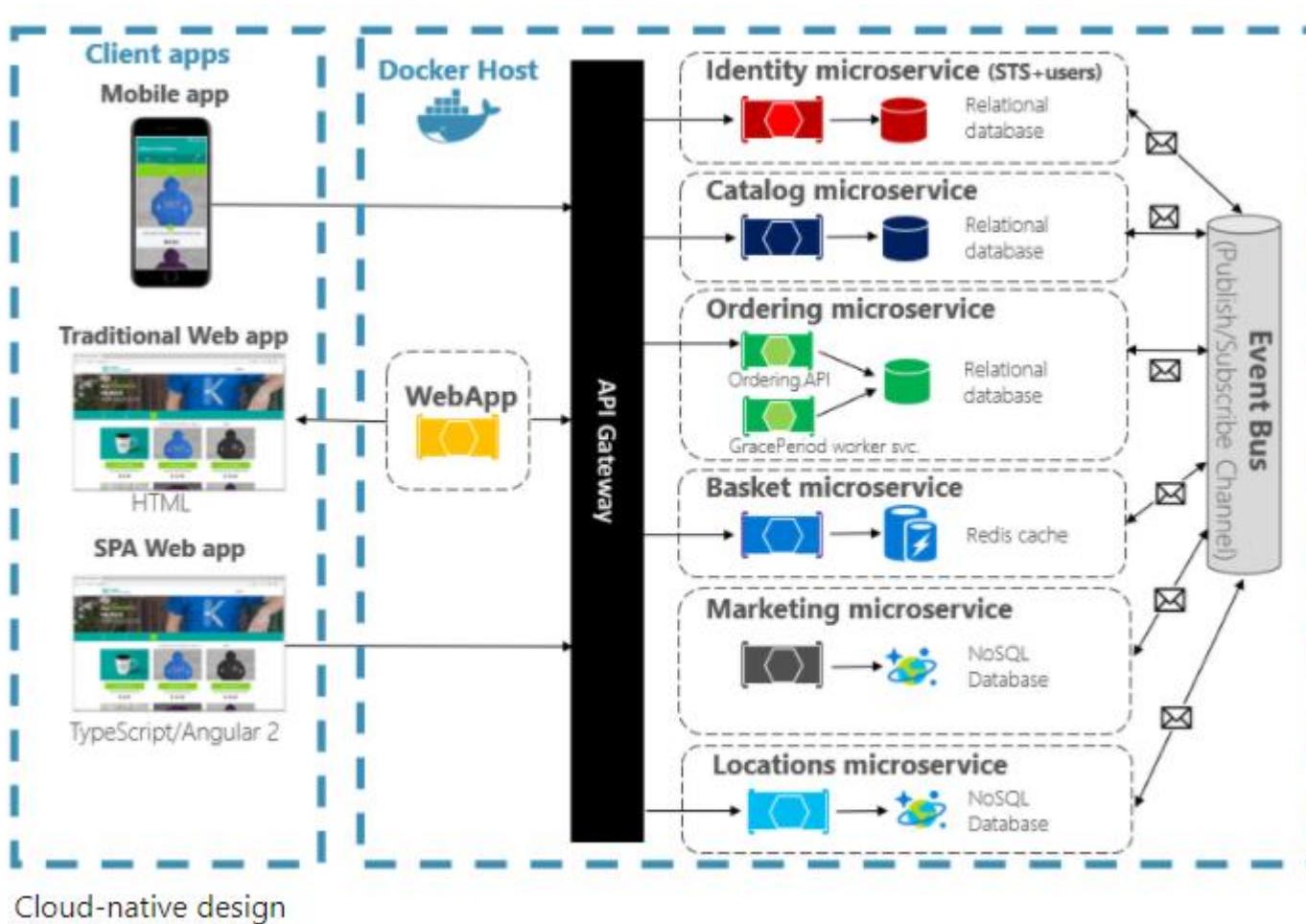
Monolithic Fear Cycle

- 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

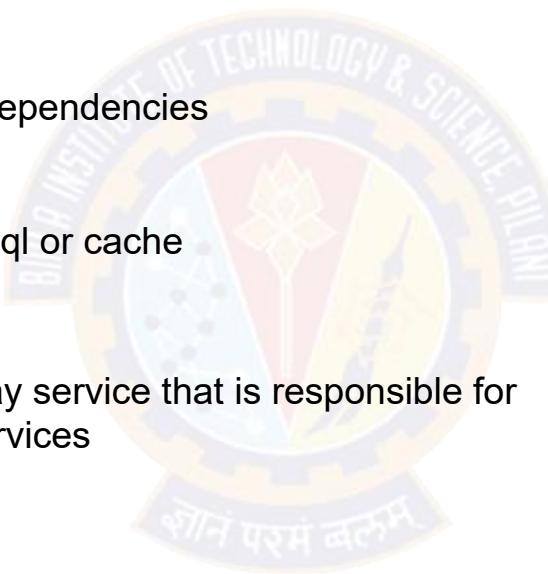


Source : Microsoft

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

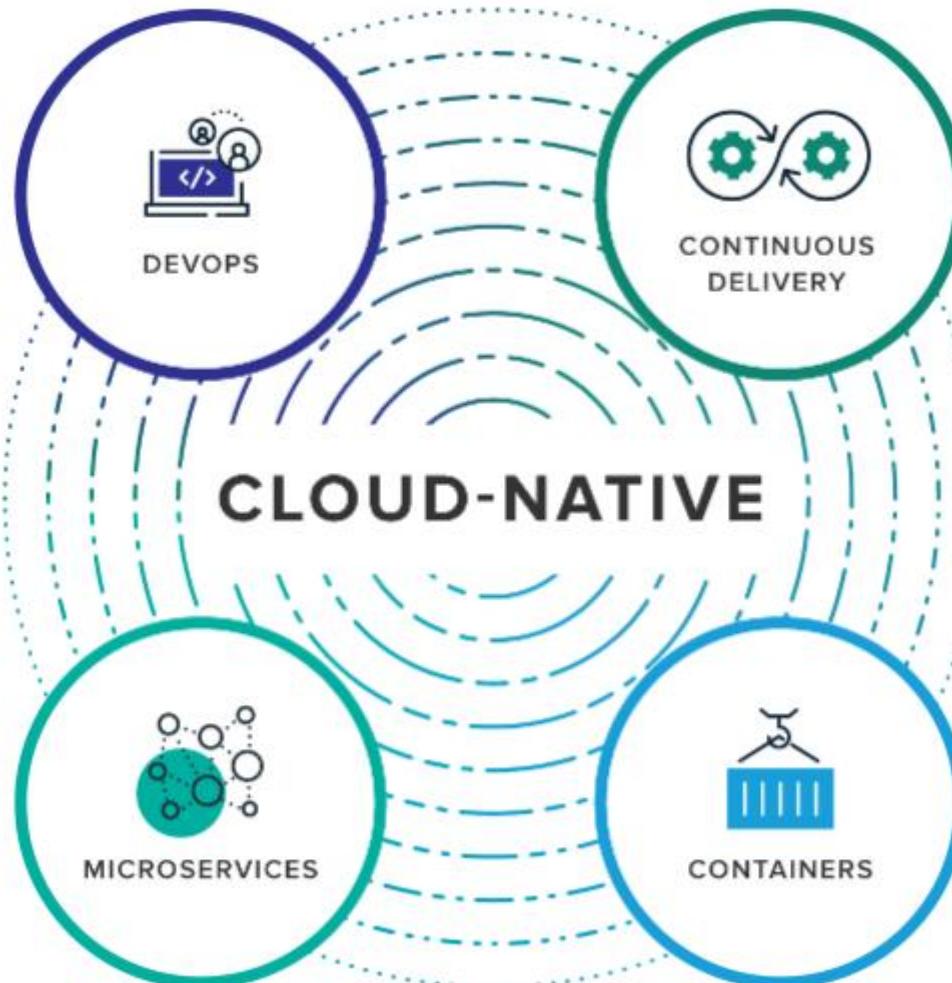


Cloud native

- 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

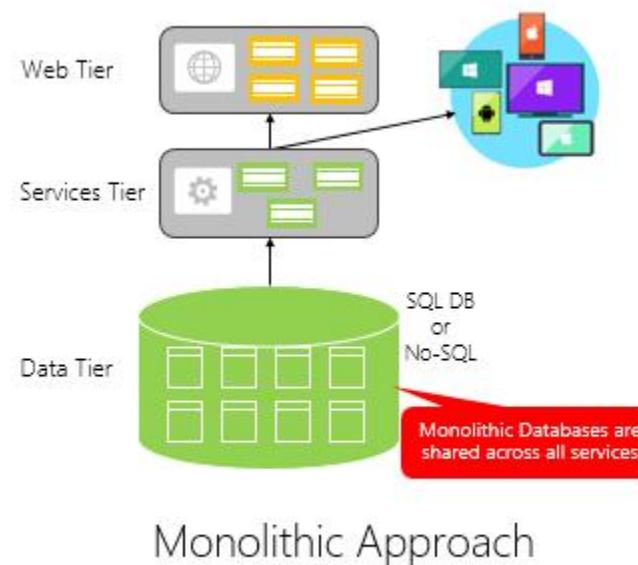
- "Containers are a great enabler of cloud-native software." - Cornelia Davis
- The Cloud Native Computing Foundation places microservice containerization as the first step in their Cloud-Native Trail Map - guidance for enterprises beginning their cloud-native journey.
- [Cloud Native Trail Map](#)



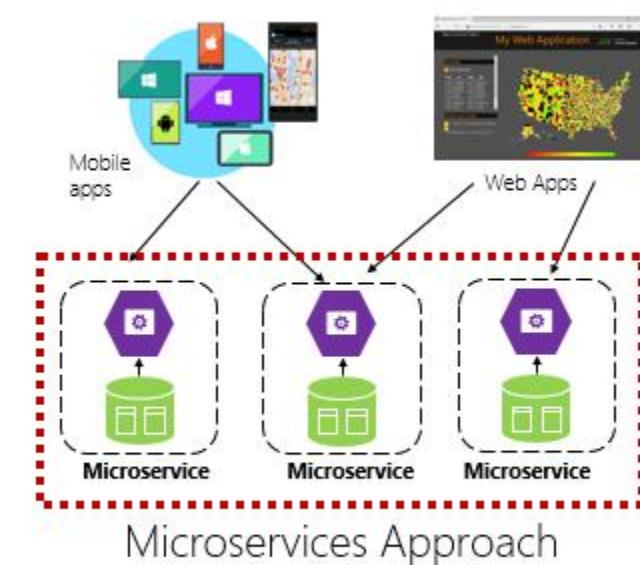
Microservices Architecture

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 Approach



Microservices Approach

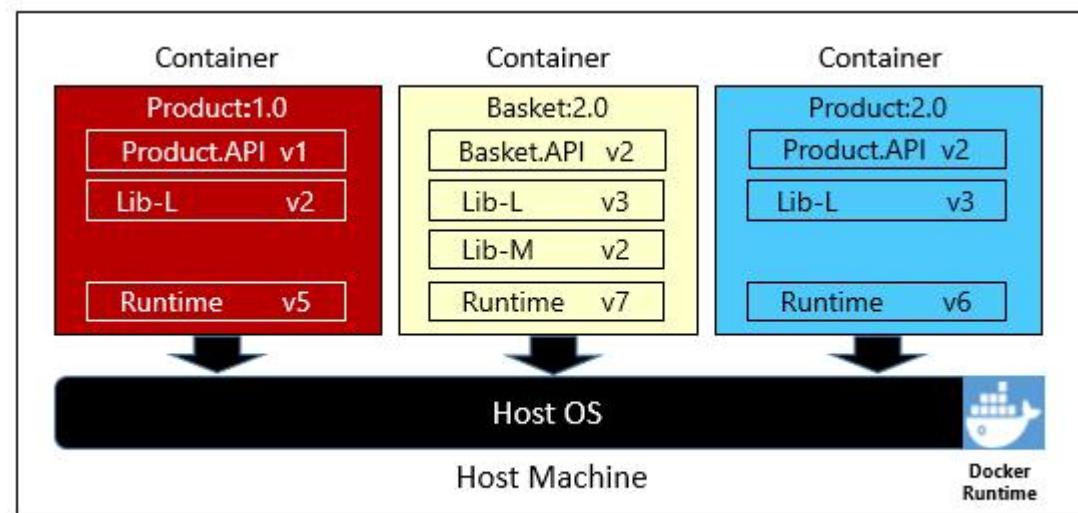
Monolithic deployment versus microservices

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

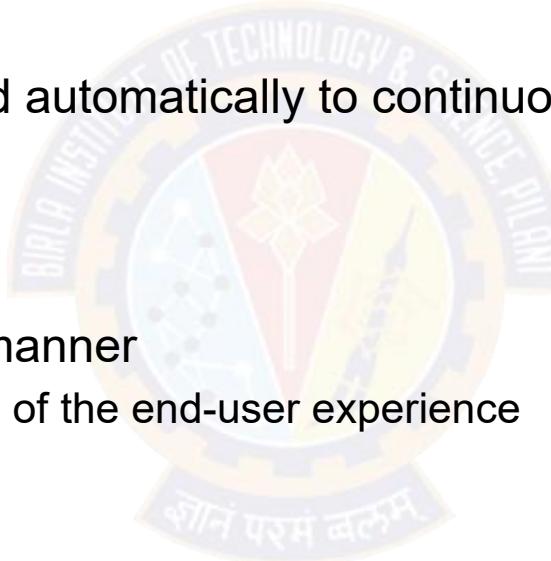


Multiple containers running on a container host

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

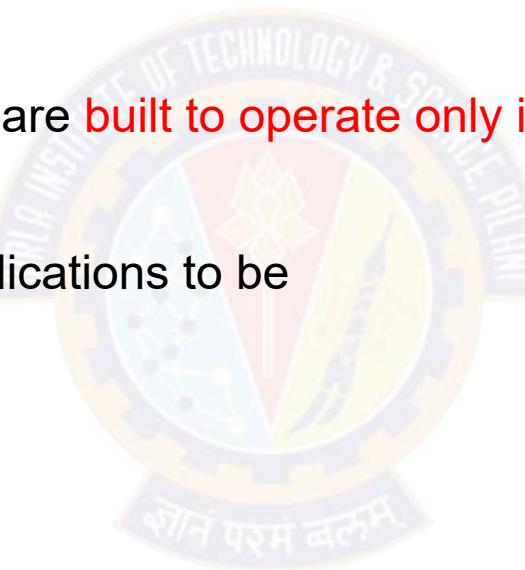
- Create the necessity of managing more elements
 - Rather than one large application, it becomes necessary to manage far more small, discrete services
- Demand additional toolsets
 - to manage the DevOps pipeline, replace traditional monitoring structures, and control microservices architecture
- Allow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

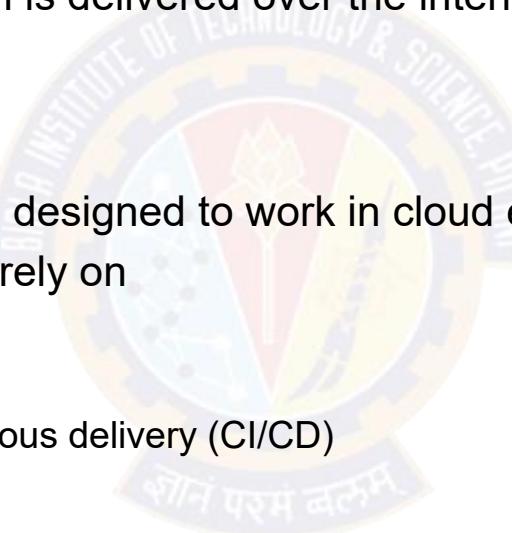
Cloud native vs. Cloud ready

- 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)

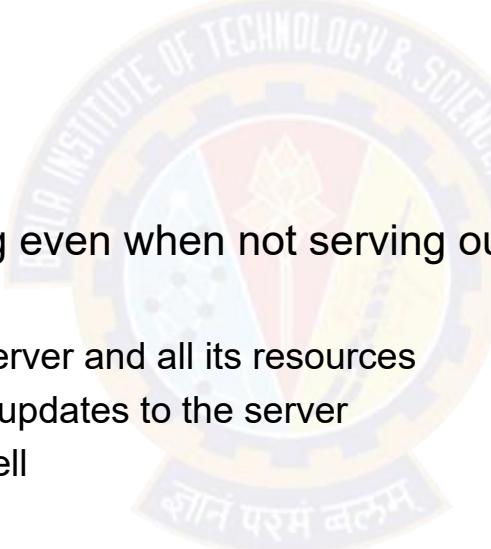
Cloud native vs. Cloud first

- 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

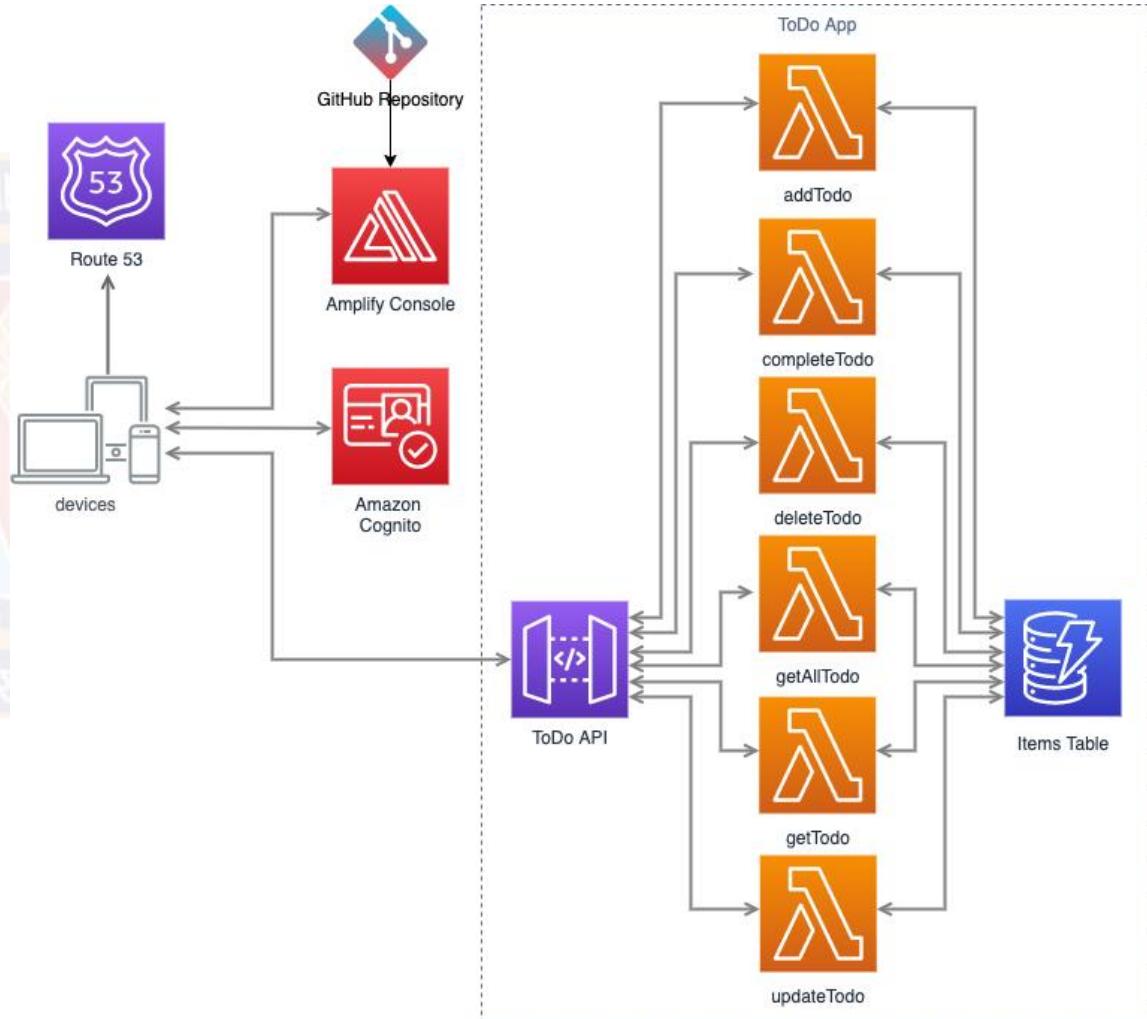
- AWS provides a set of fully managed services that can be used to build and run serverless applications
 - Serverless applications don't require provisioning, maintaining, and administering servers for backend components such as compute, databases, storage, stream processing, message queueing, and more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

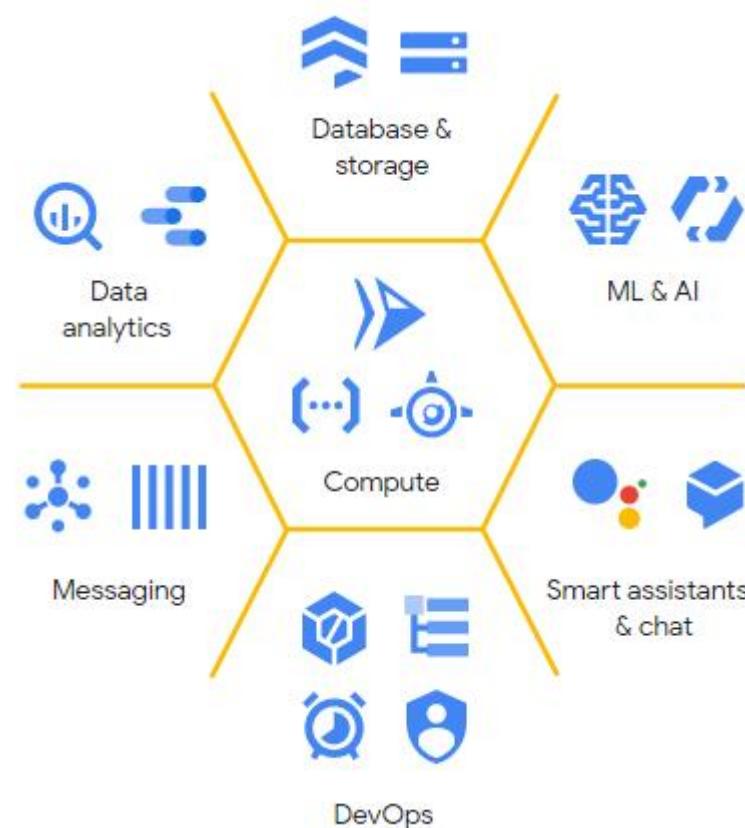


Source : AWS

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

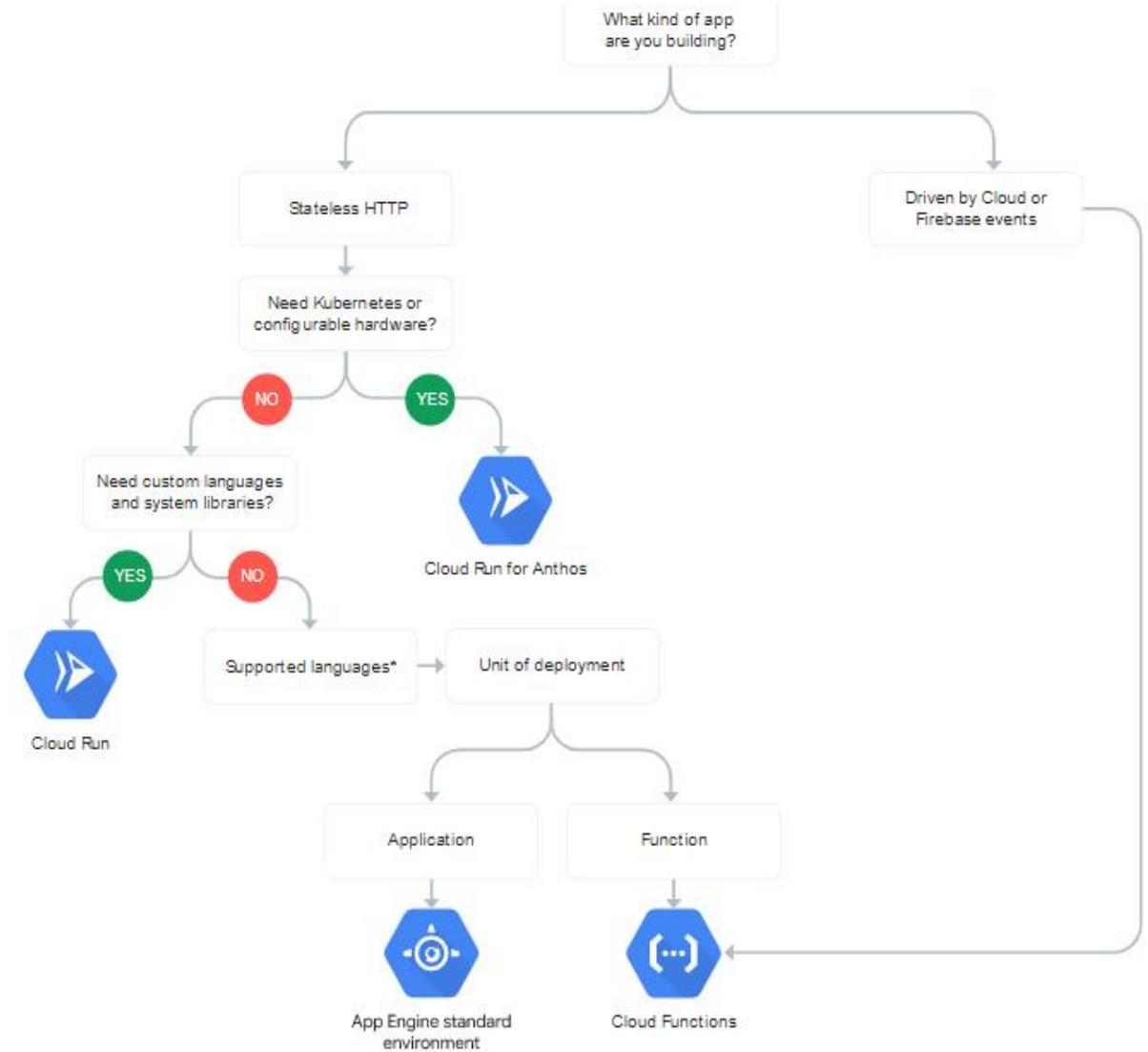
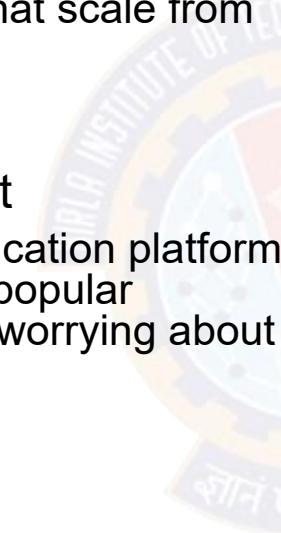


[Source : Google Product Page](#)

Google Cloud Platform (2)

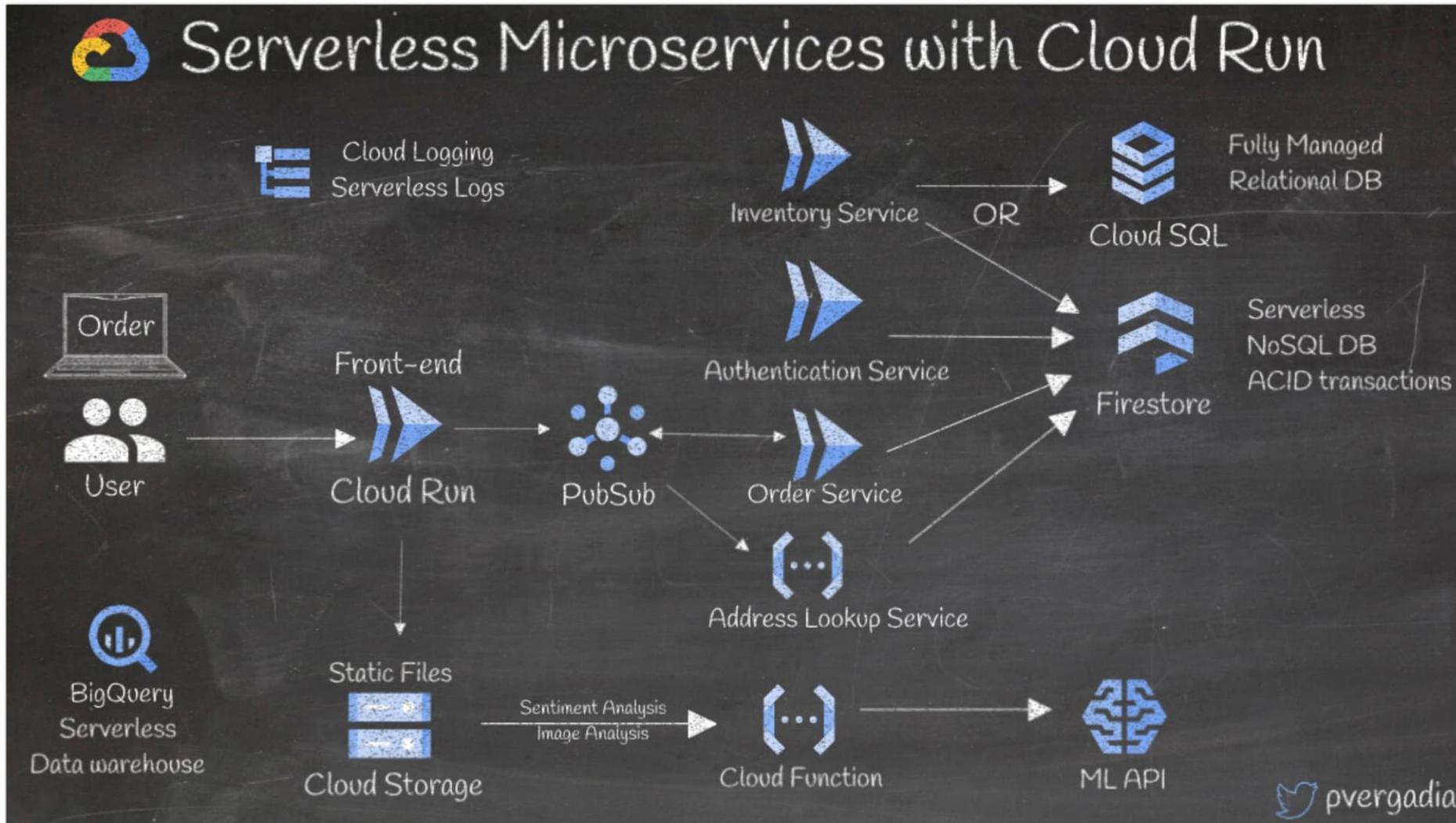
Services Offered

- 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



Source : Google

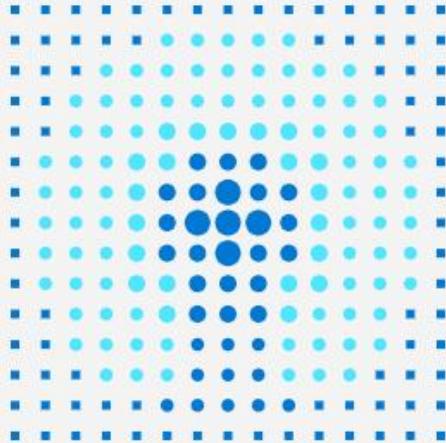
GCP Serverless Architecture (2)

Example - Flow

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

Microsoft Azure

Serverless application patterns



Serverless 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

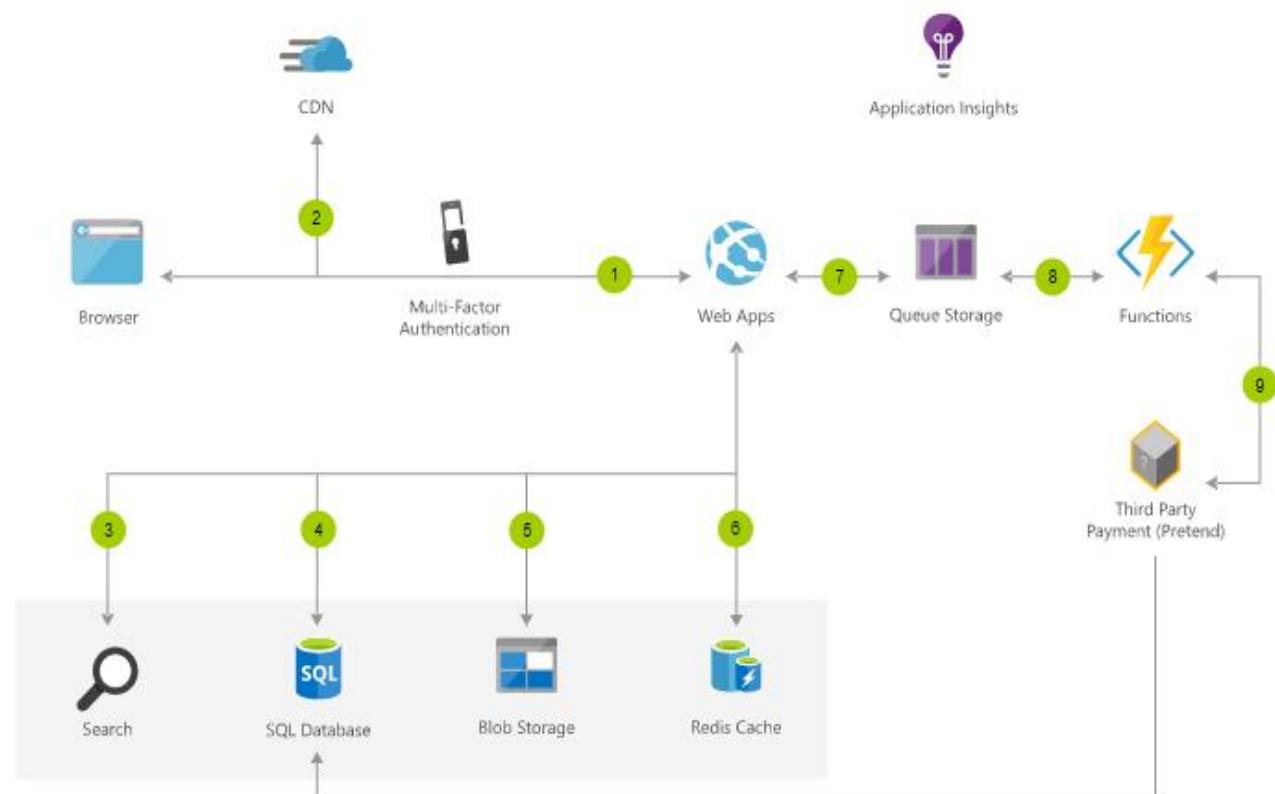
A serverless API gateway is a centralised, fully managed entry point for serverless backend services. It enables developers to publish, manage, secure and analyse APIs at global scale.

[Source : MS Serverless](#)

Microsoft Azure Serverless Architecture

Example – Architect scalable e-commerce web app

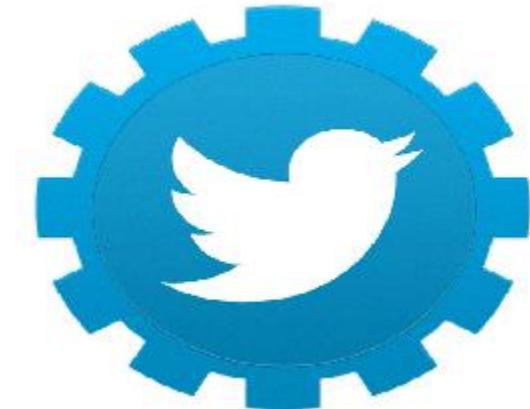
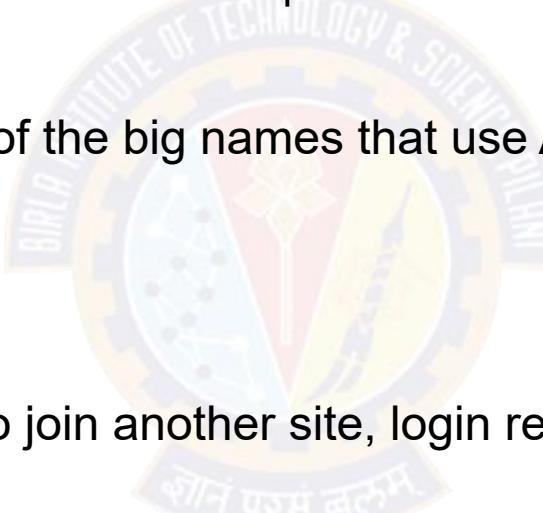
- 1 User accesses the web app in browser and signs in.
- 2 Browser pulls static resources such as images from Azure Content Delivery Network.
- 3 User searches for products and queries SQL database.
- 4 Web site pulls product catalog from database.
- 5 Web app pulls product images from Blob Storage.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

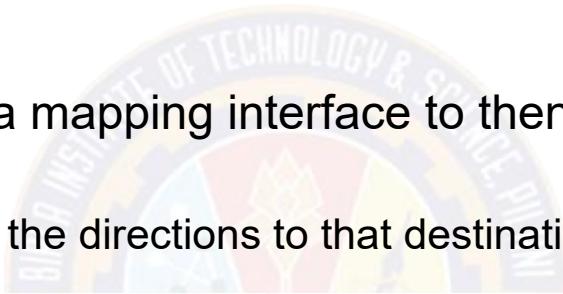
- Technically,
 - ✓ an API describes how to connect a dataset or business process with some sort of consumer application or another business process
- We are probably familiar with a lot of the big names that use APIs all the time
- For example,
- whenever use Facebook account to join another site, login 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.



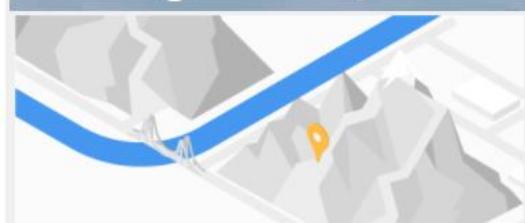
Mapping API

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



Google Maps Platform



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

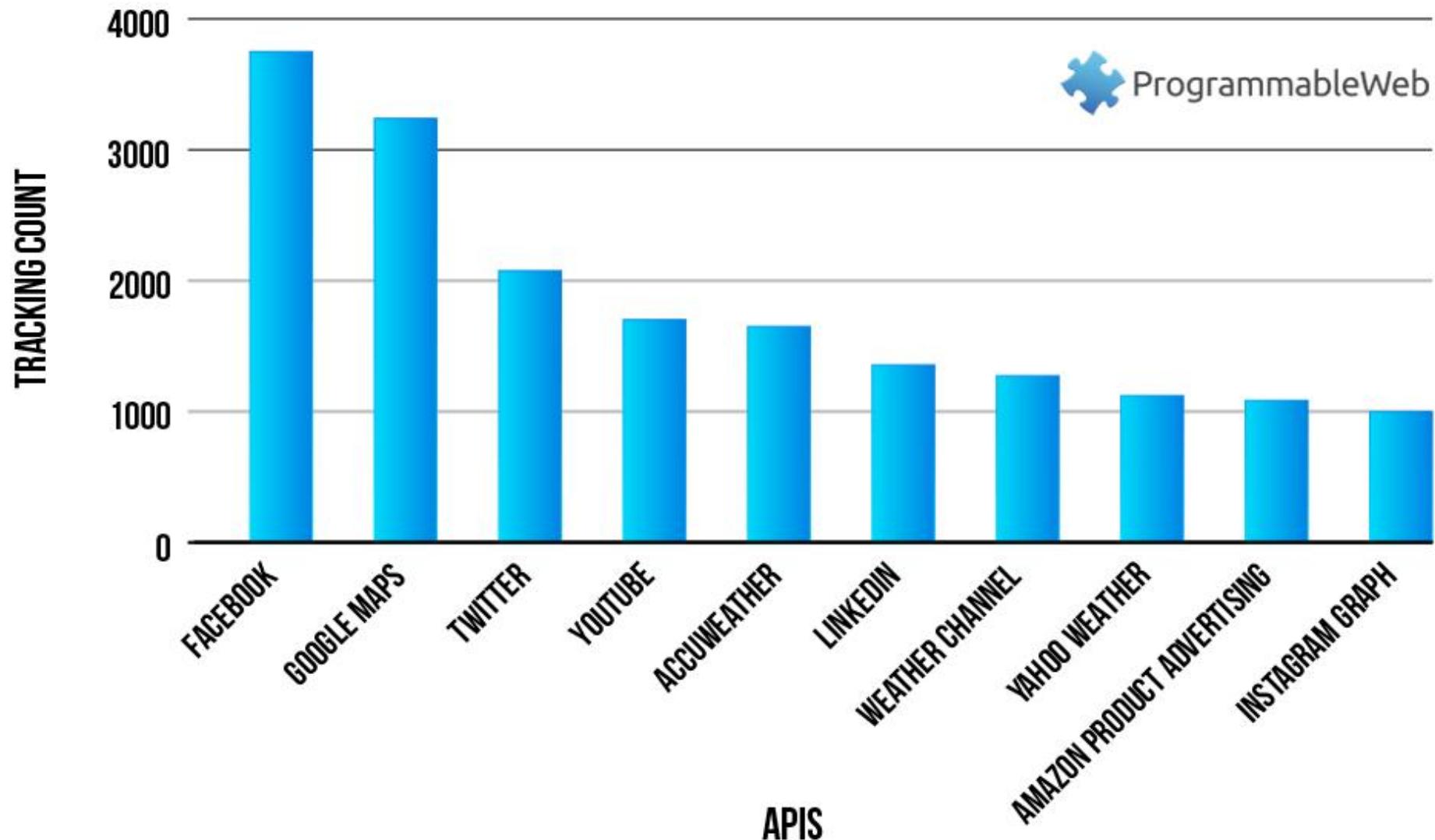
- Marketing APIs are a collection of API endpoints that can be used to help you advertise on social media platform like Facebook, Twitter etc.
- Amazon offers vendors and professional sellers the opportunity to advertise their products on Amazon



Source : pinterest

Popular APIs

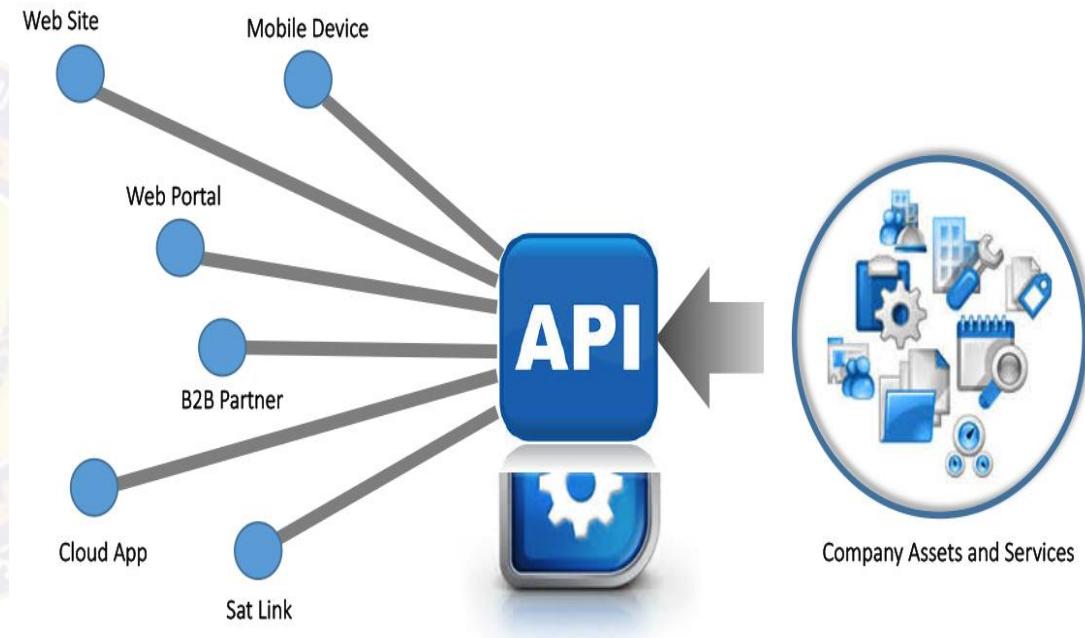
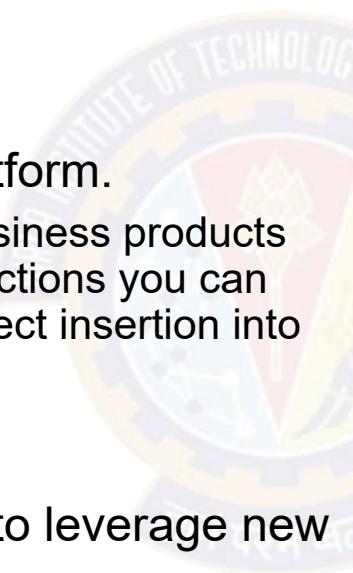
TOP TRACKED APIs OF ALL TIME



ProgrammableWeb

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

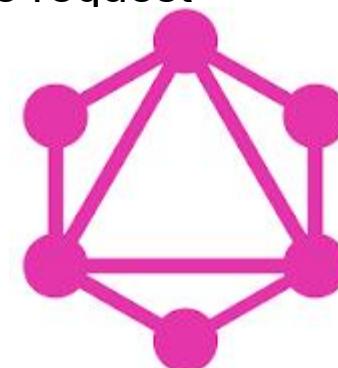
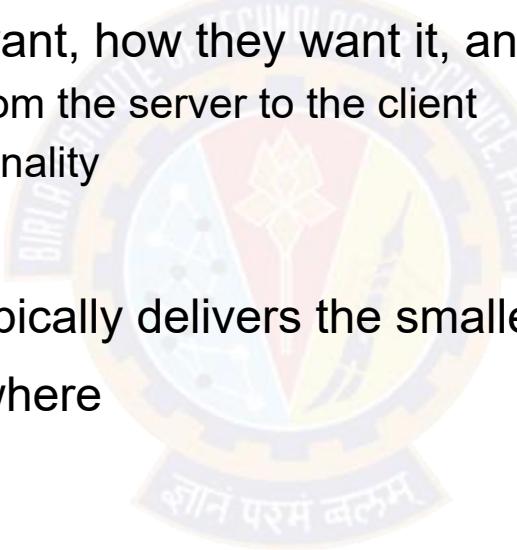


- Actually a new take on an old approach known as RPC, or Remote Procedure Call
- RPC is a method for executing a procedure on a remote server
- RPC functions upon an idea of contracts, in which the negotiation is defined and constricted by the client-server relationship rather than the architecture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





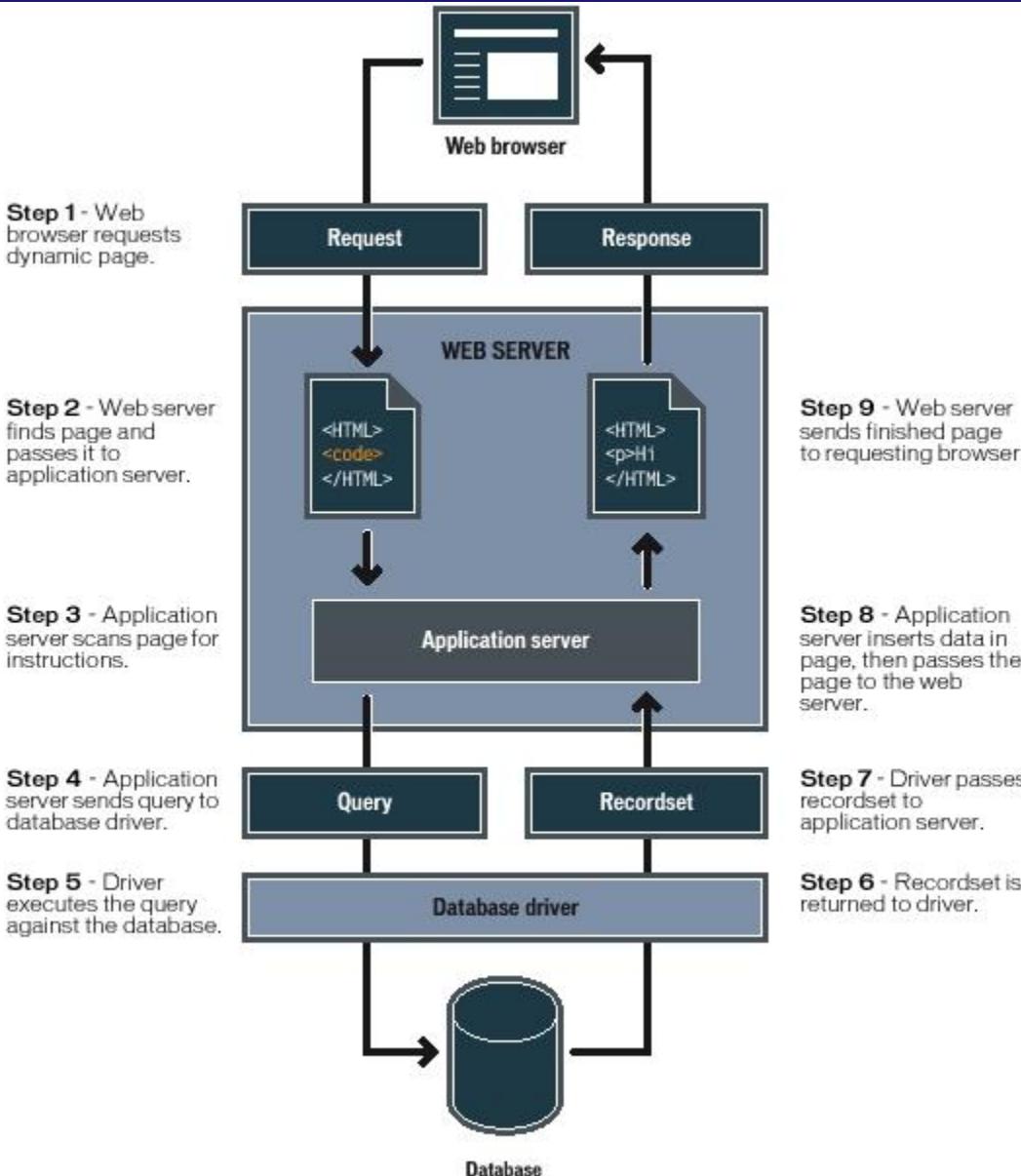
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

FrontEnds

Chandan Ravandur N

WebApps

Working

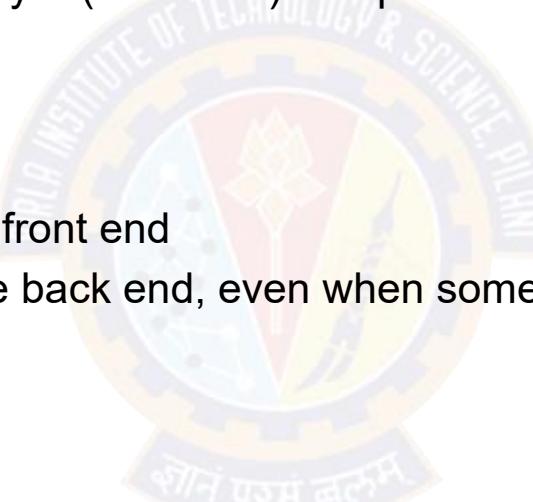


[Image source](#)

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

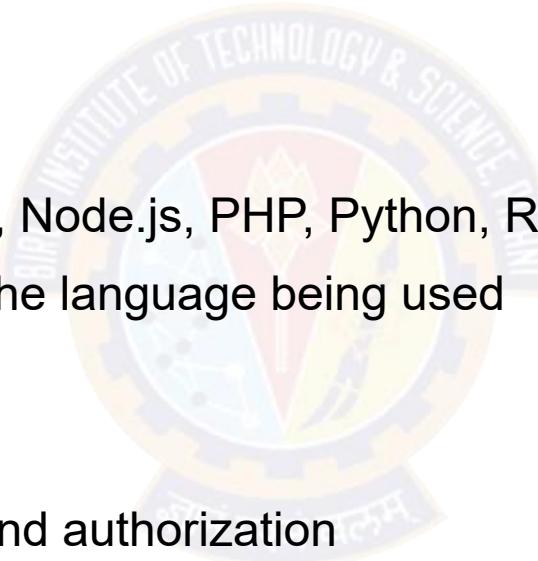
Focus is on

- 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

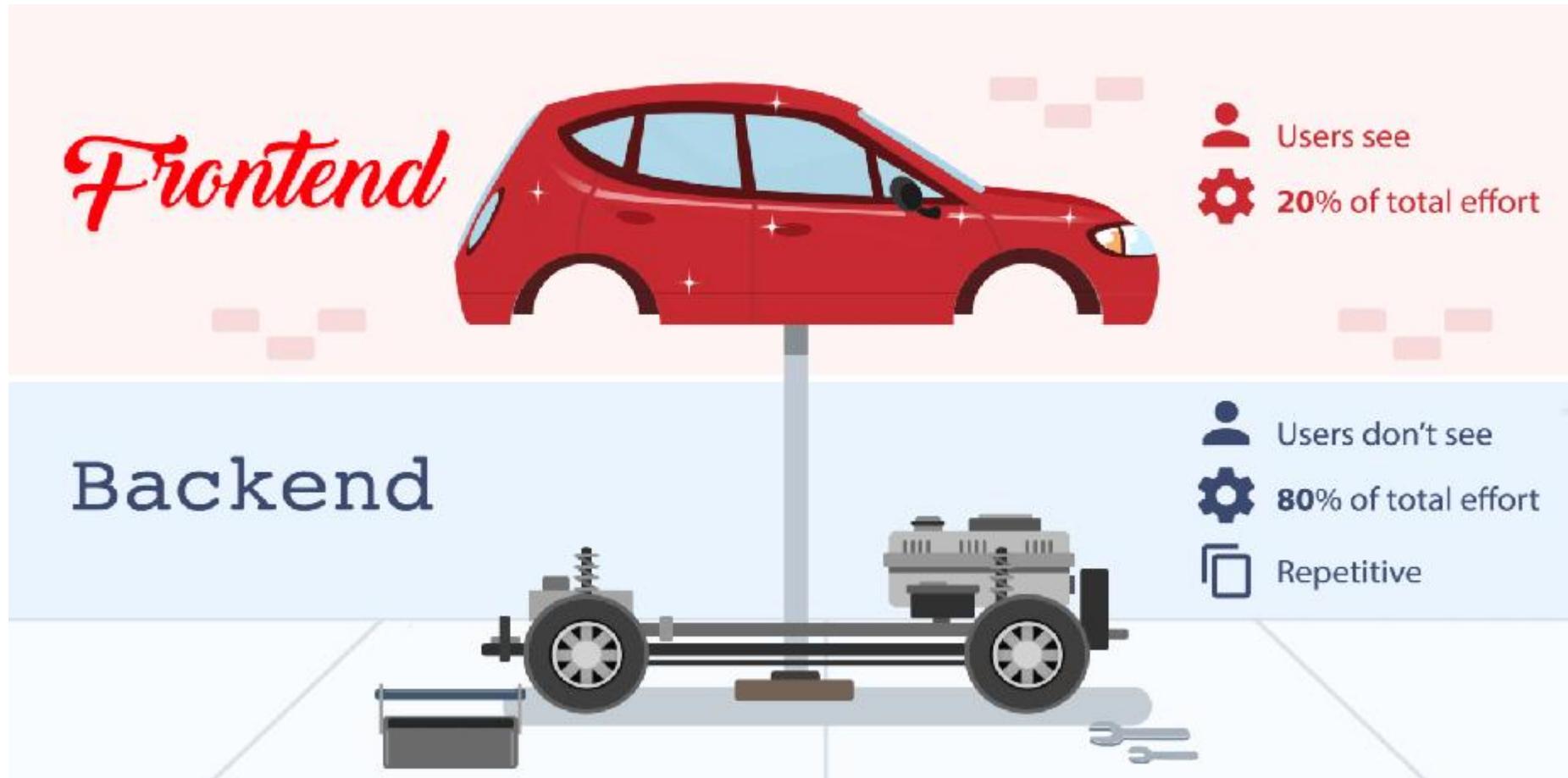
Back-end

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

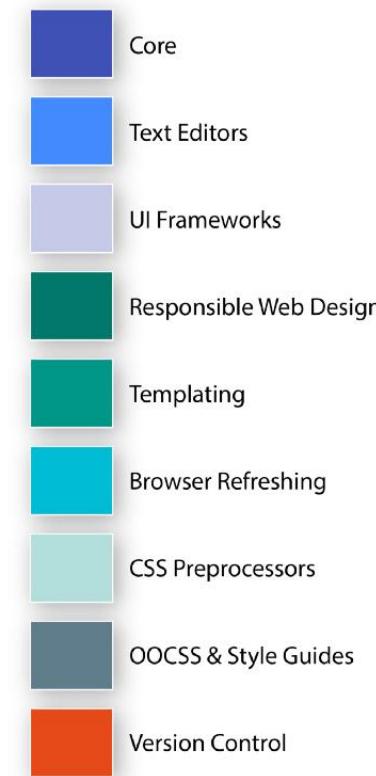
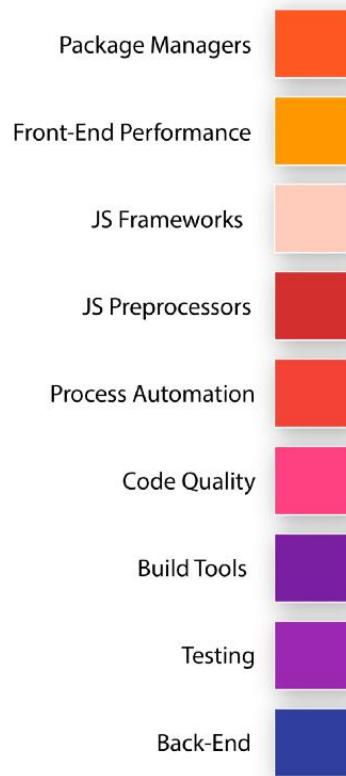
Interaction

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



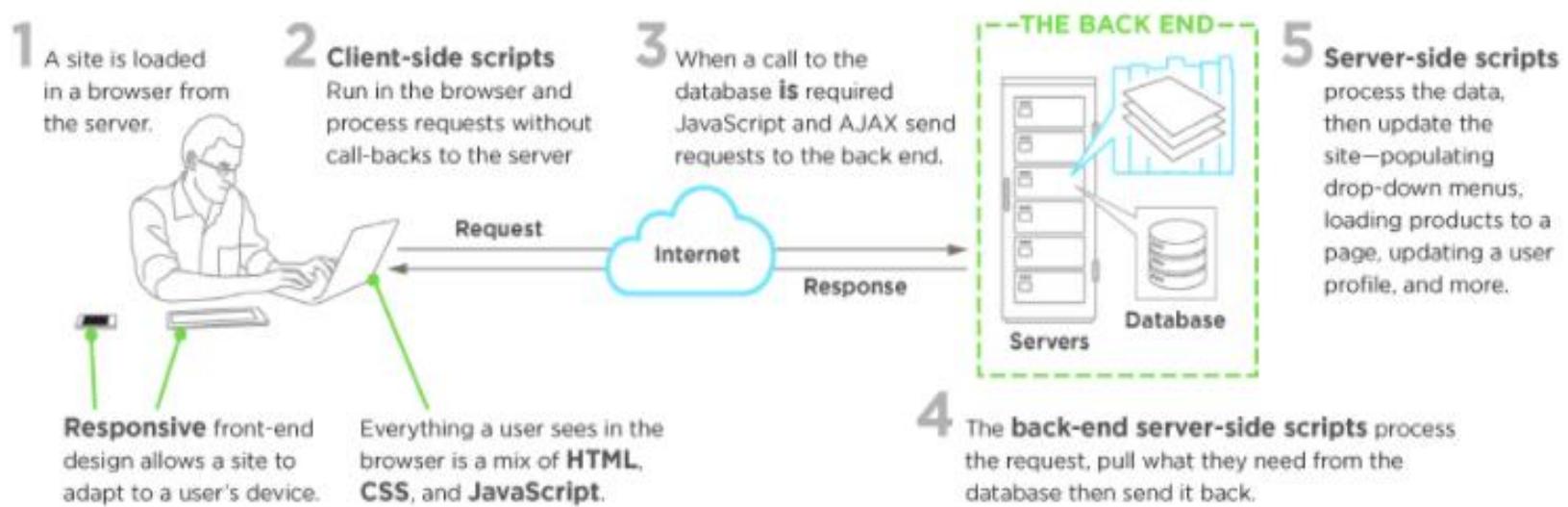
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

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](#)

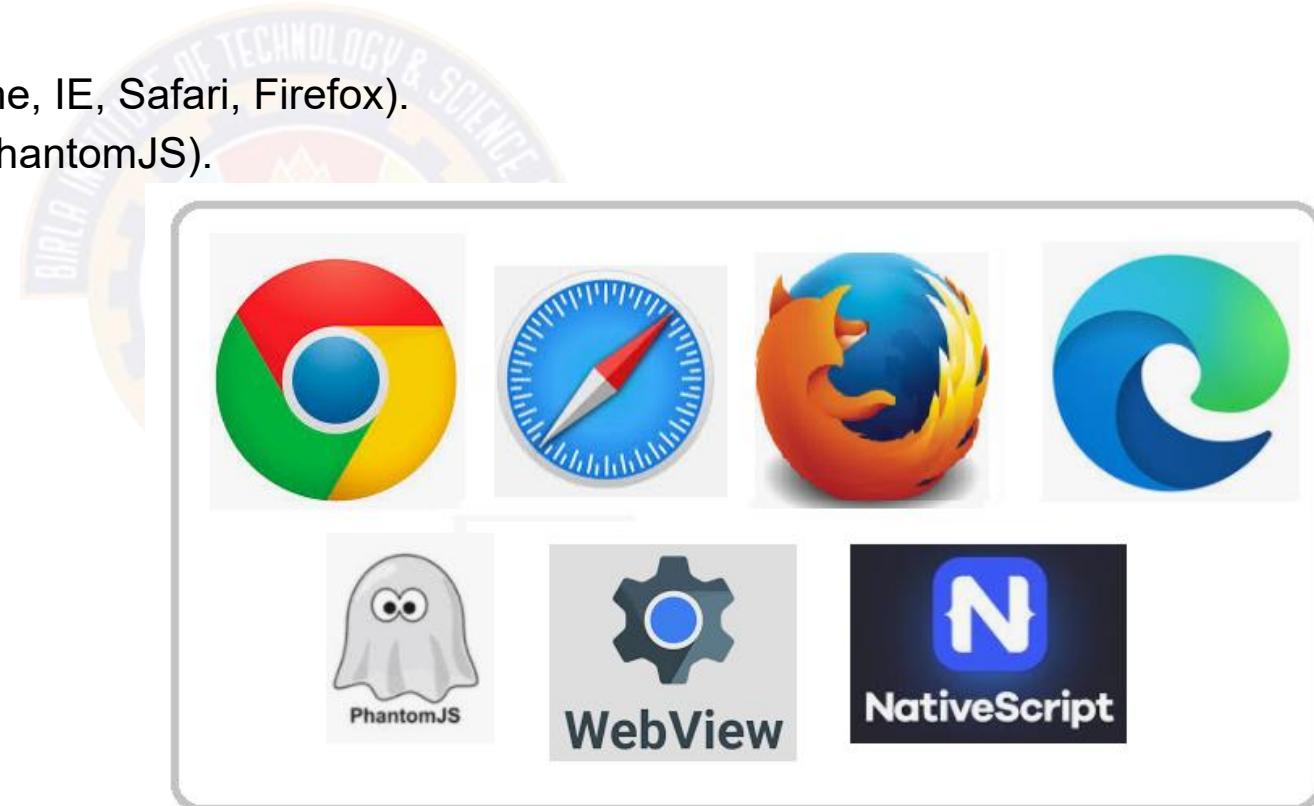
Role of a front end developer

- 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

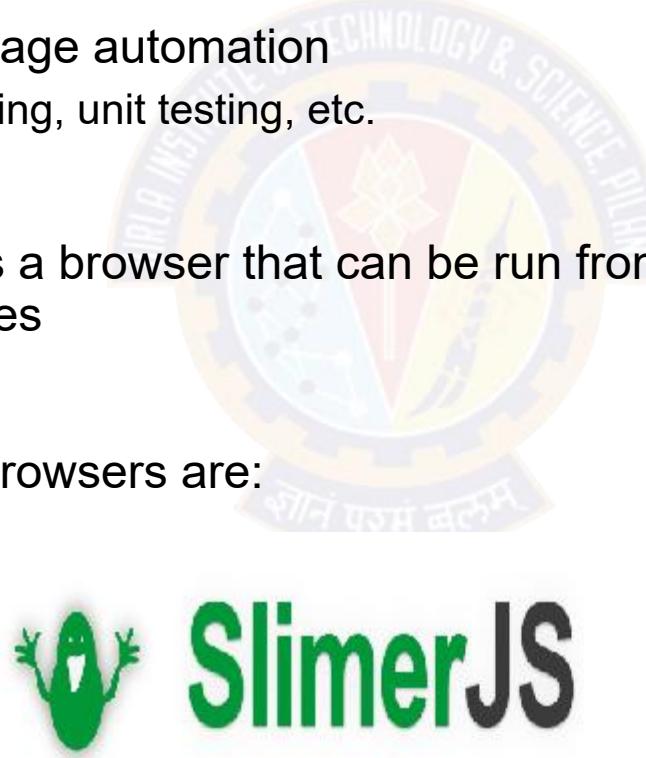
- Is software used to retrieve, present, and traverse information on the WWW
- Typically run on a desktop or laptop computer, tablet, or phone,
 - but as of late a browser can be found on just about anything (i.e., on a fridge, in cars, etc.).
- The most common web browsers are (shown in order of most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

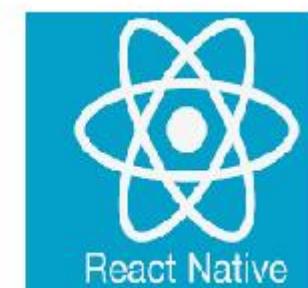
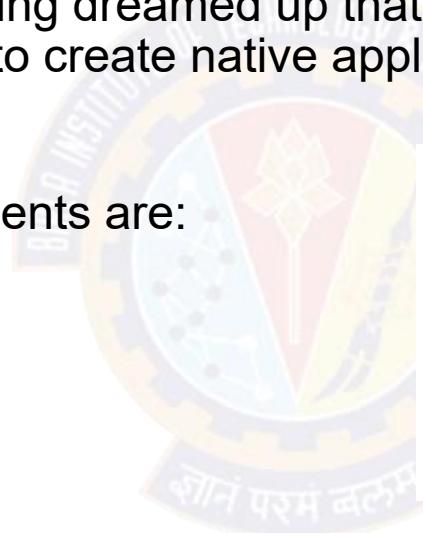
- Are used by a native OS, in a native application, to run web pages
- Think of a webview like an iframe or a single tab from a web browser that is embedded in a native application running on a device
 - e.g., webviews iOS, android, windows
- The most common solutions for webview development are:
 - 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





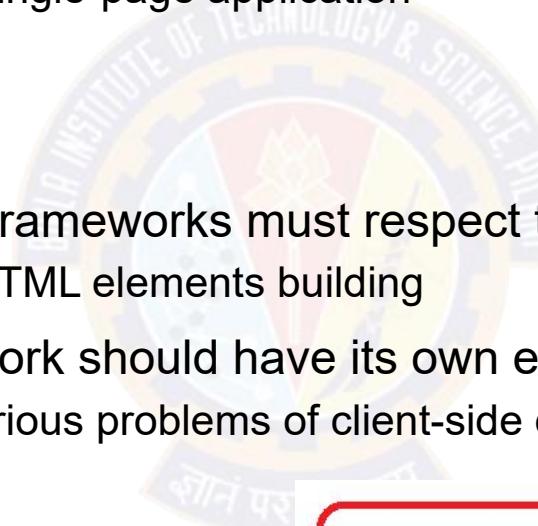
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

FrontEnd - Frameworks

Chandan Ravandur N

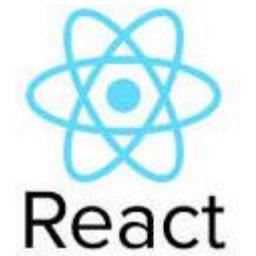
Why frameworks?

- 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

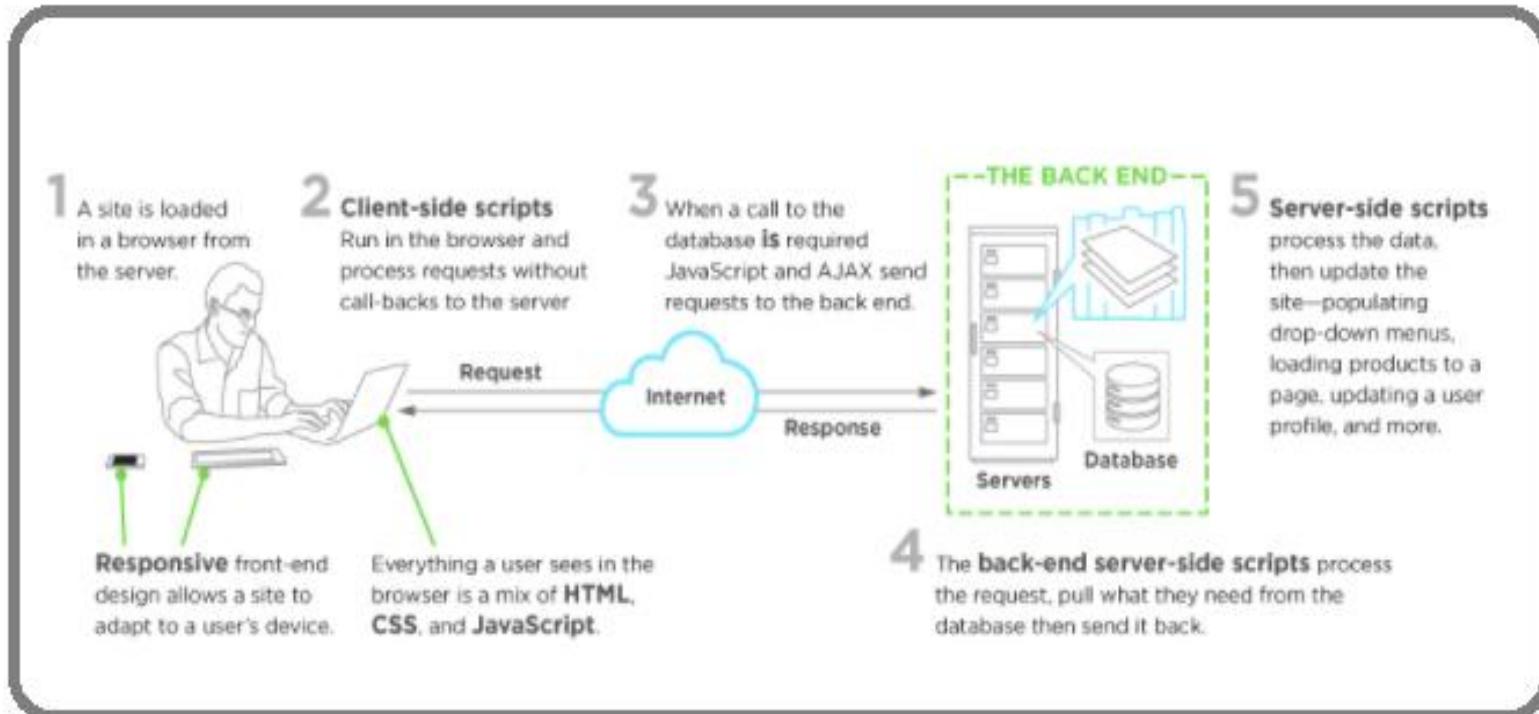
Backend

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

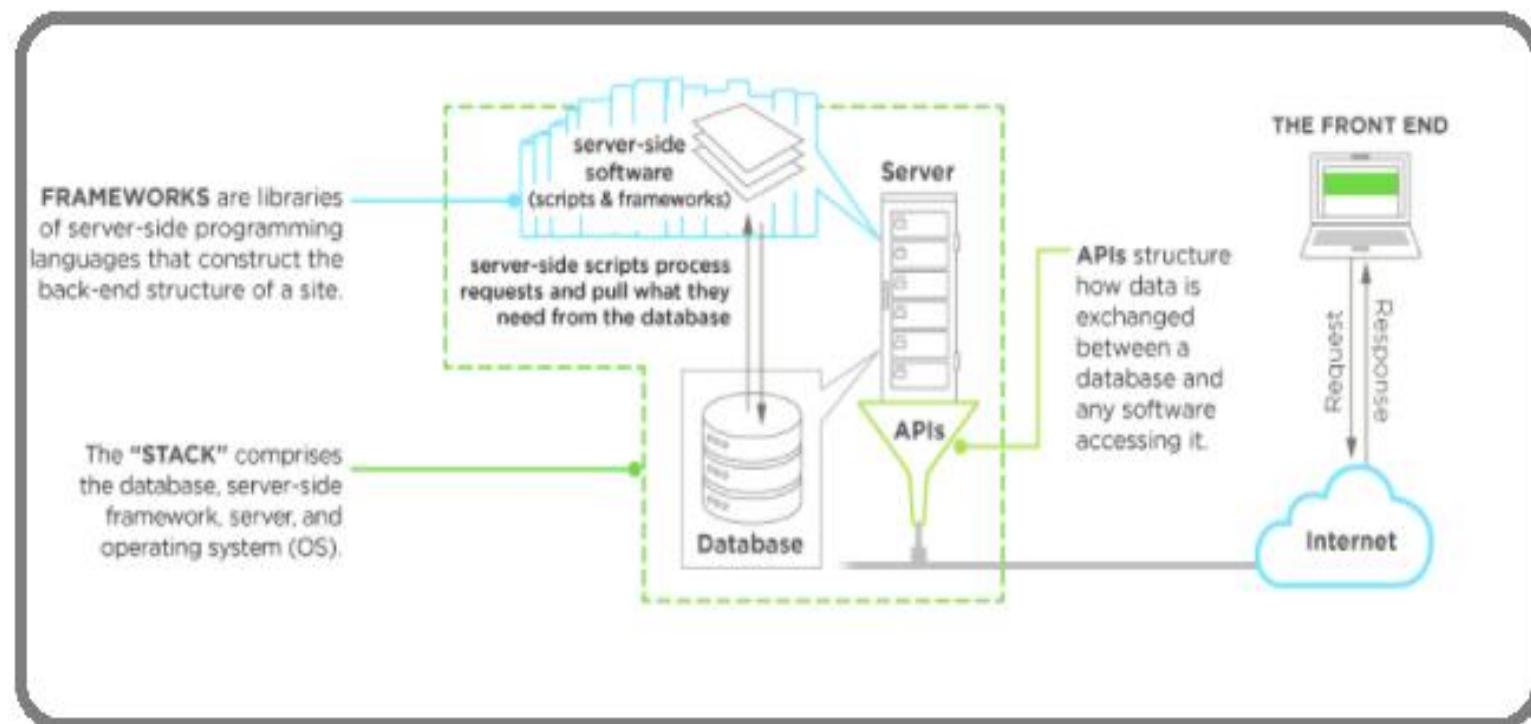
Interaction Explained

- 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

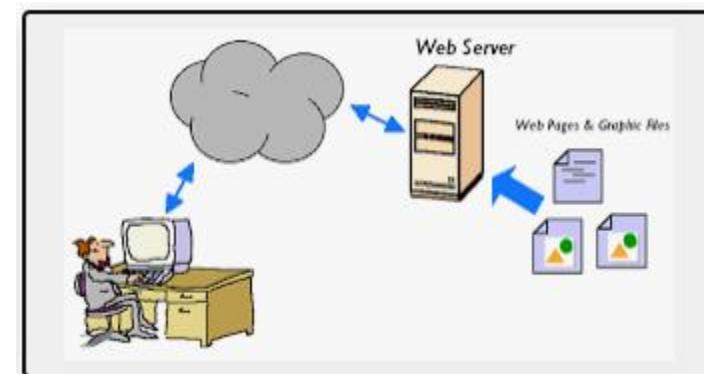
Backend - WebServers

Chandan Ravandur N

Web Server

Defined

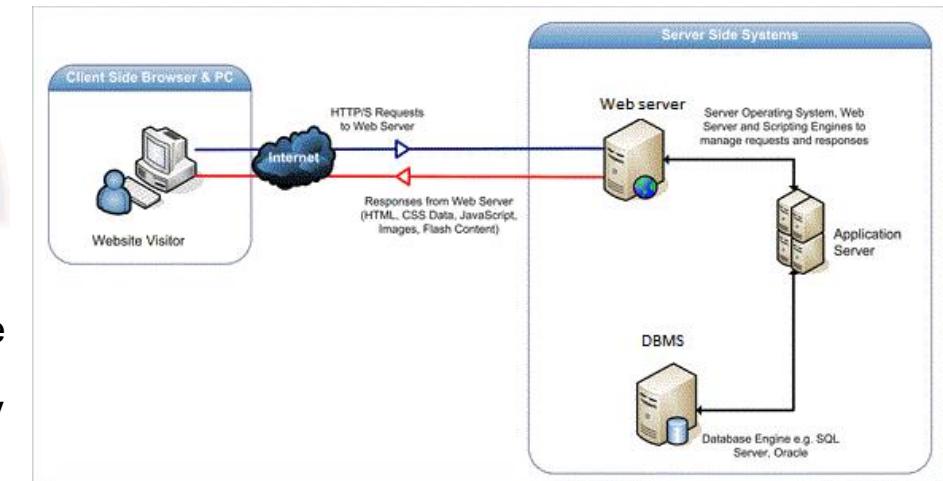
- All computers that host websites must have web server software
- Software and hardware that uses HTTP (Hypertext Transfer Protocol) to respond to client requests made over the World Wide Web
 - Main job of a web server is to display website content through storing, processing and delivering webpages 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

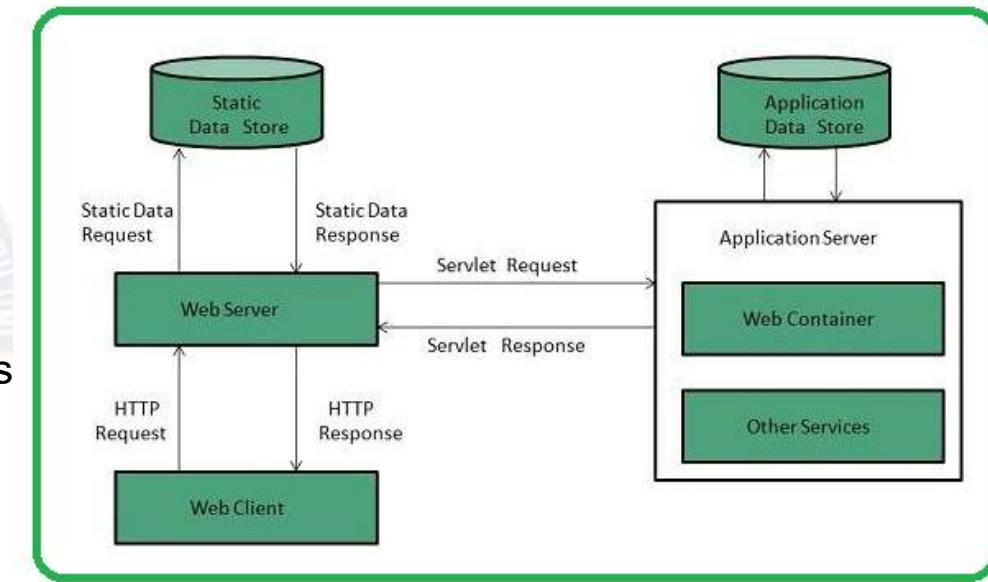
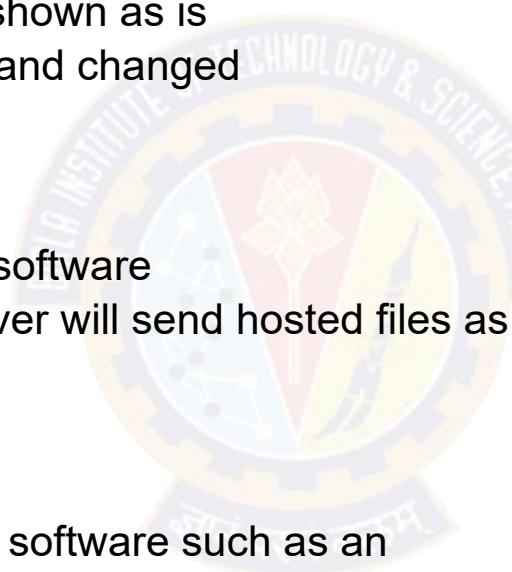


Image Source : [TutorialsPoint](#)

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

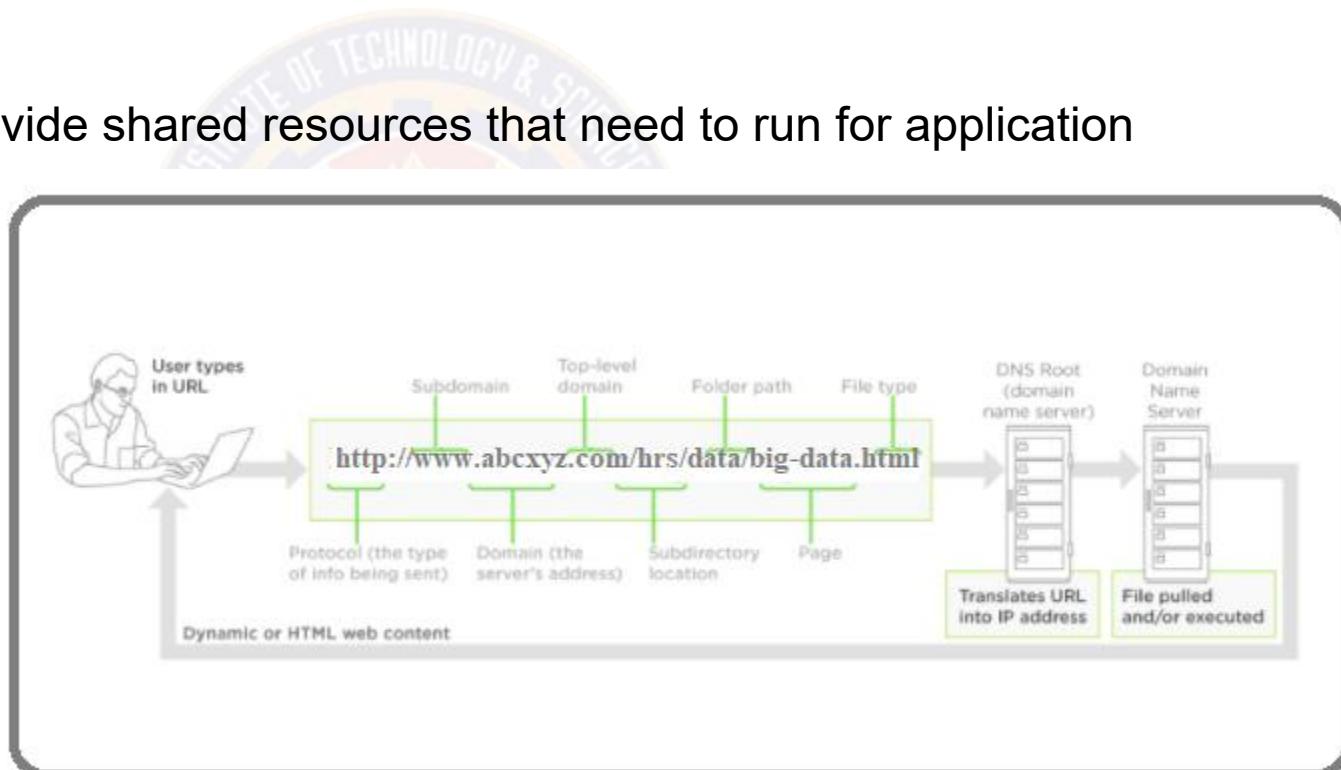
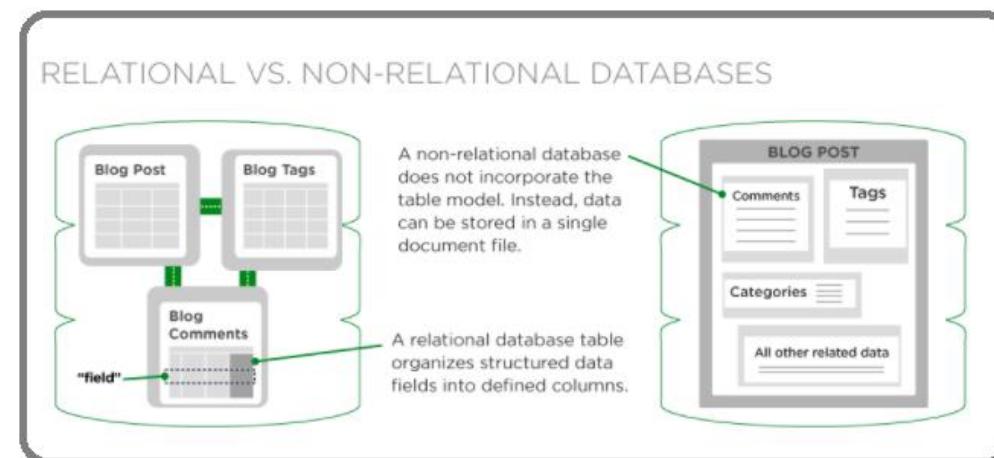


Image : Adapted from Upwork

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

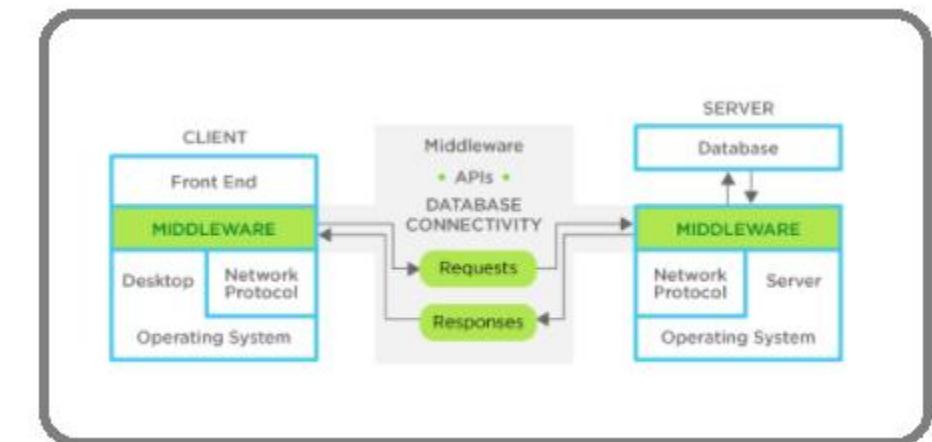
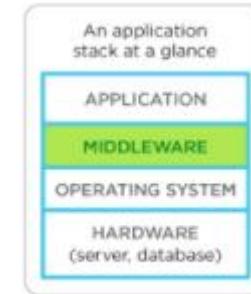


[Image : Upwork](#)

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

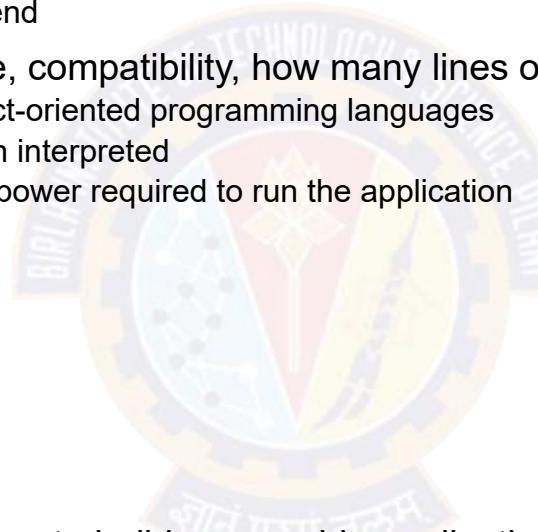


[Image : Upwork](#)

Server side Components

Programming languages & frameworks: The nuts & bolts

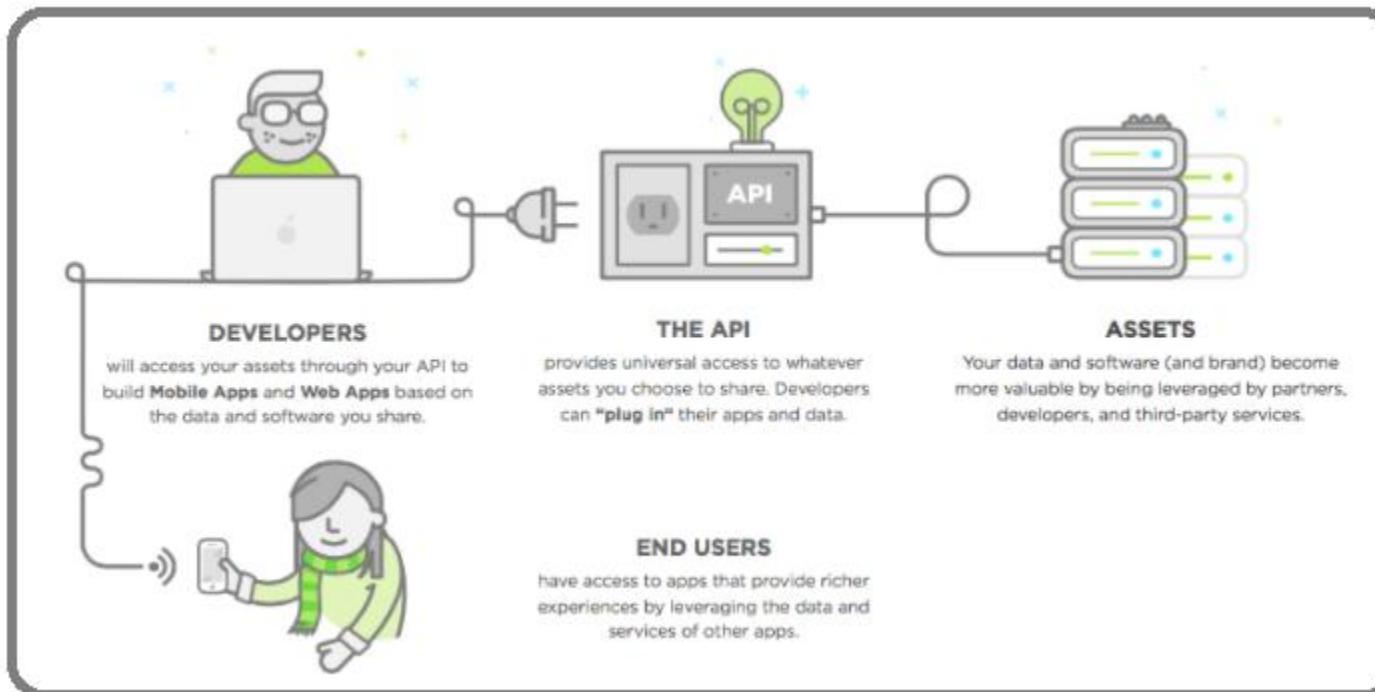
- Can choose from a variety of languages and frameworks depending on
 - the type of application being built
 - the specific processing requirements
 - other components already exist on the back end
- Languages will differ in file size, performance, compatibility, how many lines of code required, and 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:



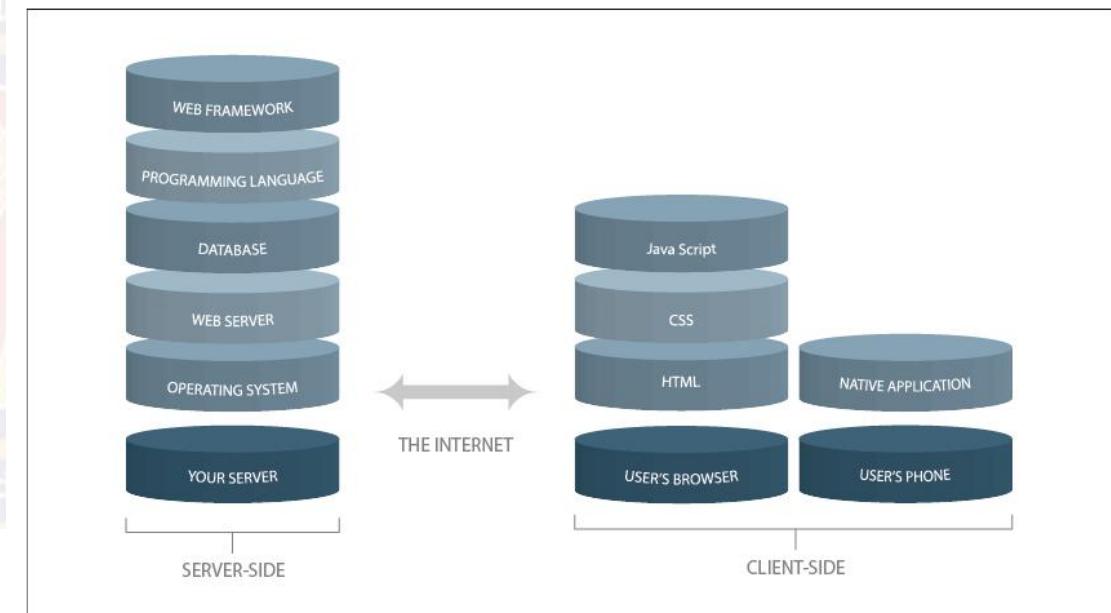
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

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

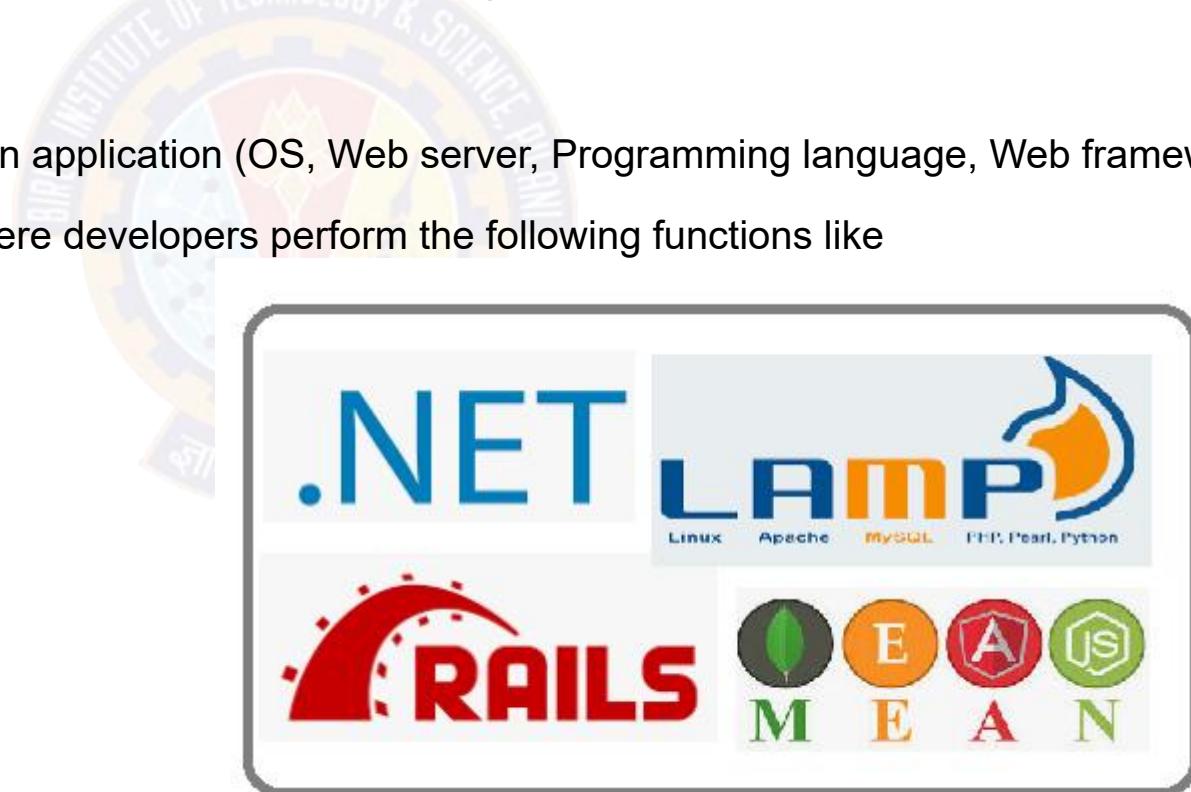


[Image source : hackernoon](#)

Tech stack

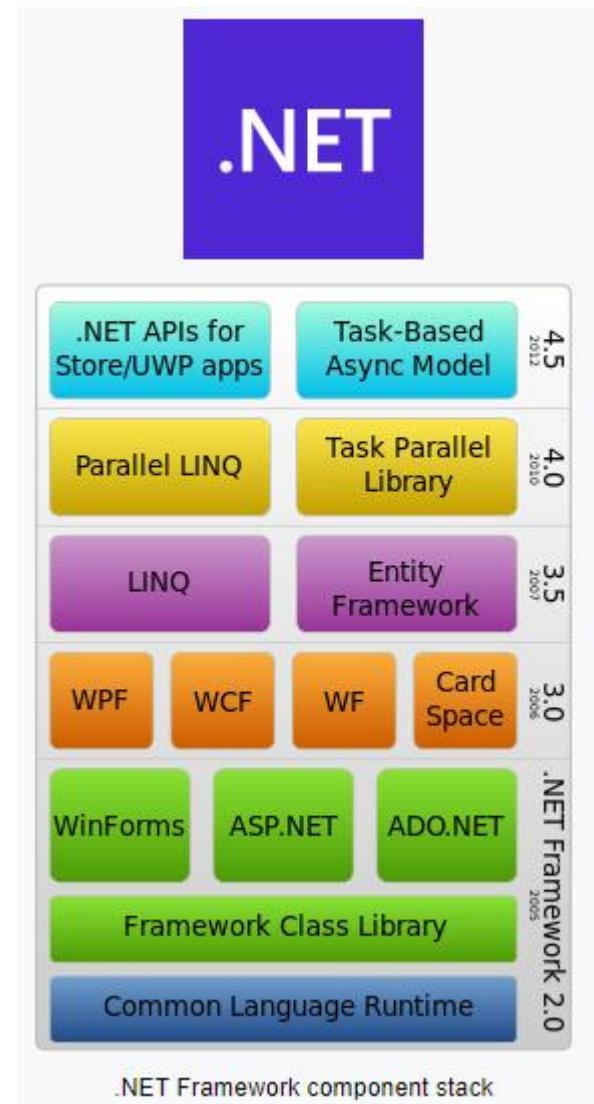
Deciphered

- 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

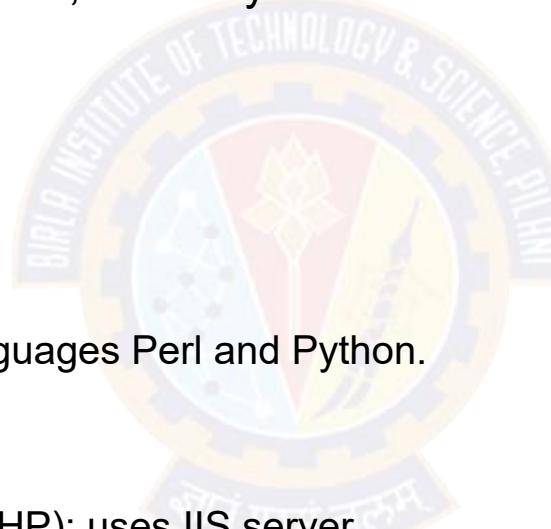


[Image source : wikipedia](#)

LAMP

Linux, Apache, MySQL, Python / Perl/ Php

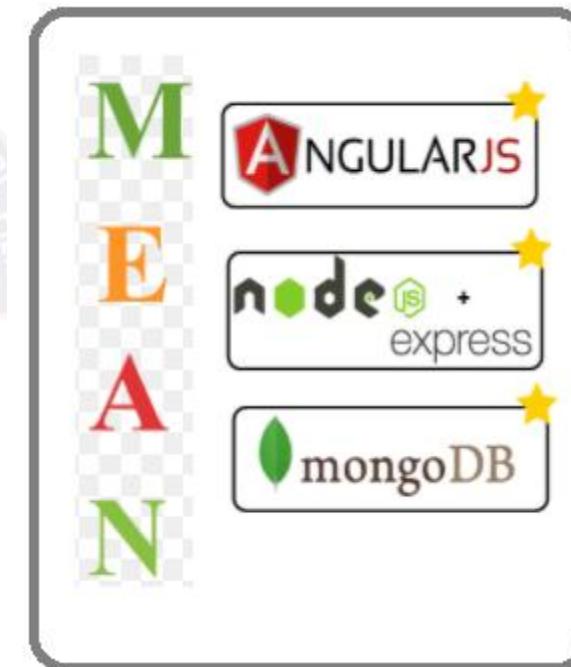
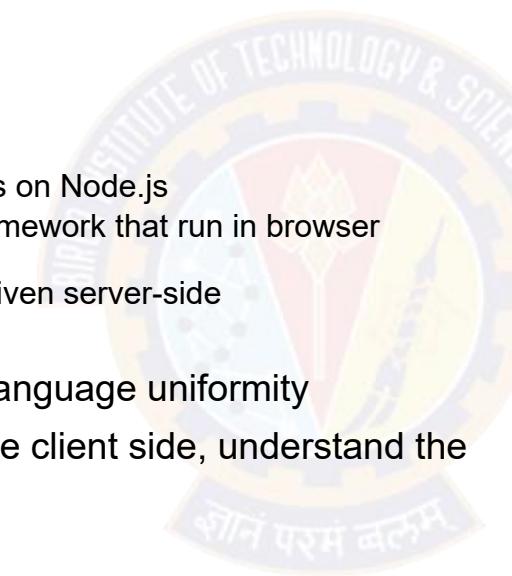
- Made of all open source software components is the one of the earliest stacks to get popular
- The most traditional stack model around, and very solid
- The main components are
 - Linux operating system,
 - Apache HTTP Server
 - MySQL RDBMS
 - PHP programming language
 - PHP is interchangeable with 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

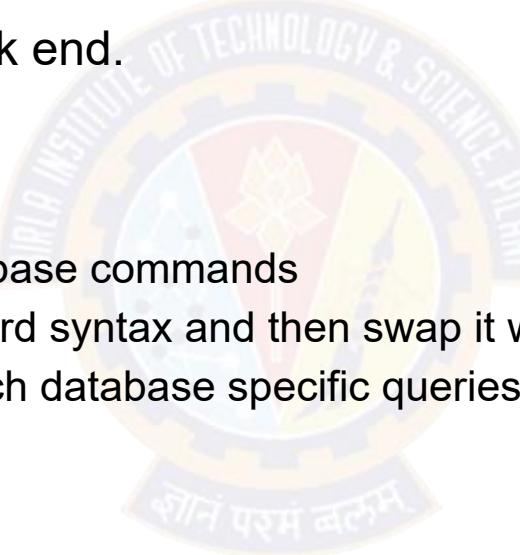
MongoDB, Express, Angular, Node

- 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

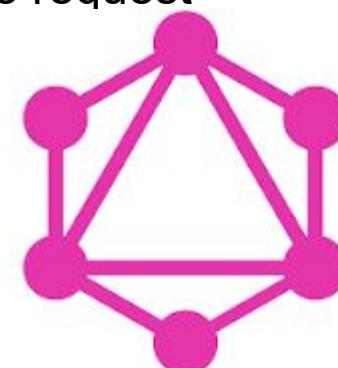
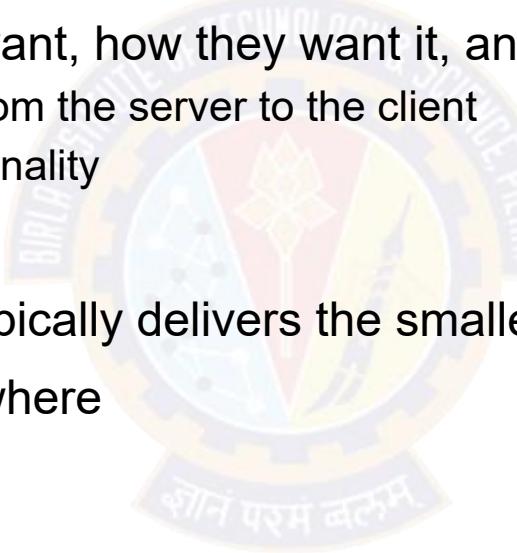


- Actually a new take on an old approach known as RPC, or Remote Procedure Call
- RPC is a method for executing a procedure on a remote server
- RPC functions upon an idea of contracts, in which the negotiation is defined and constricted by the client-server relationship rather than the architecture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



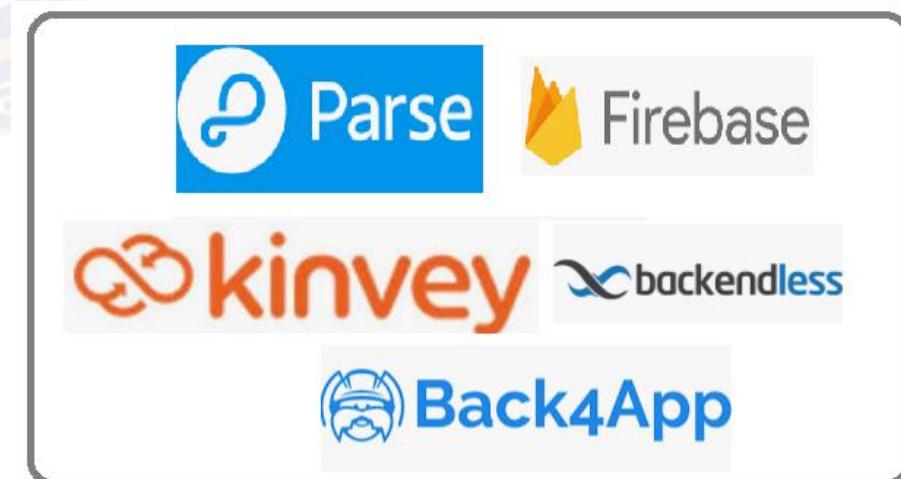
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

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

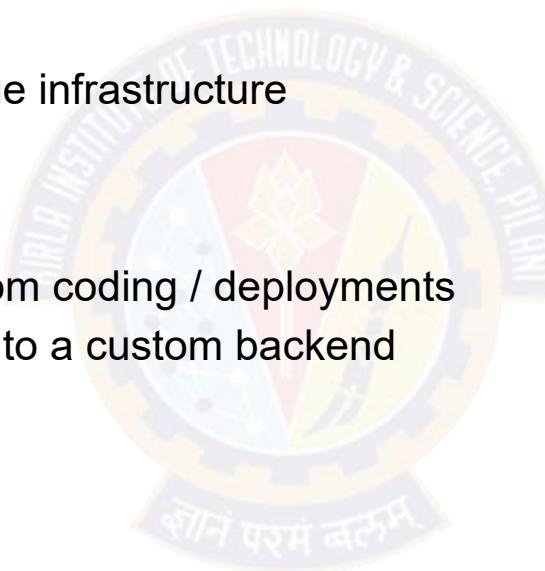
Technical reasons to use BaaS:

- 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

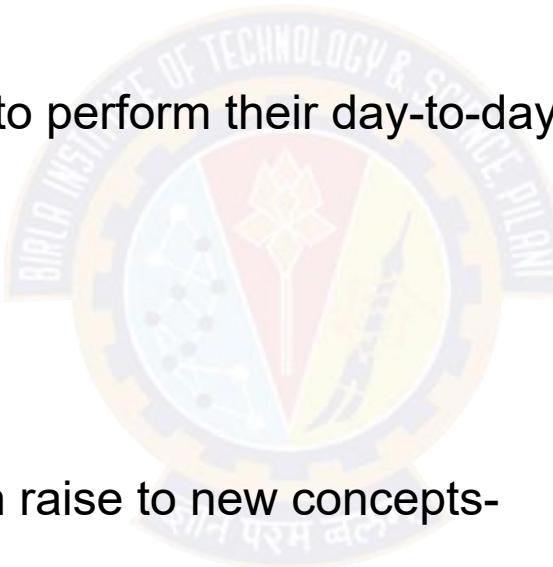
Issues

- 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

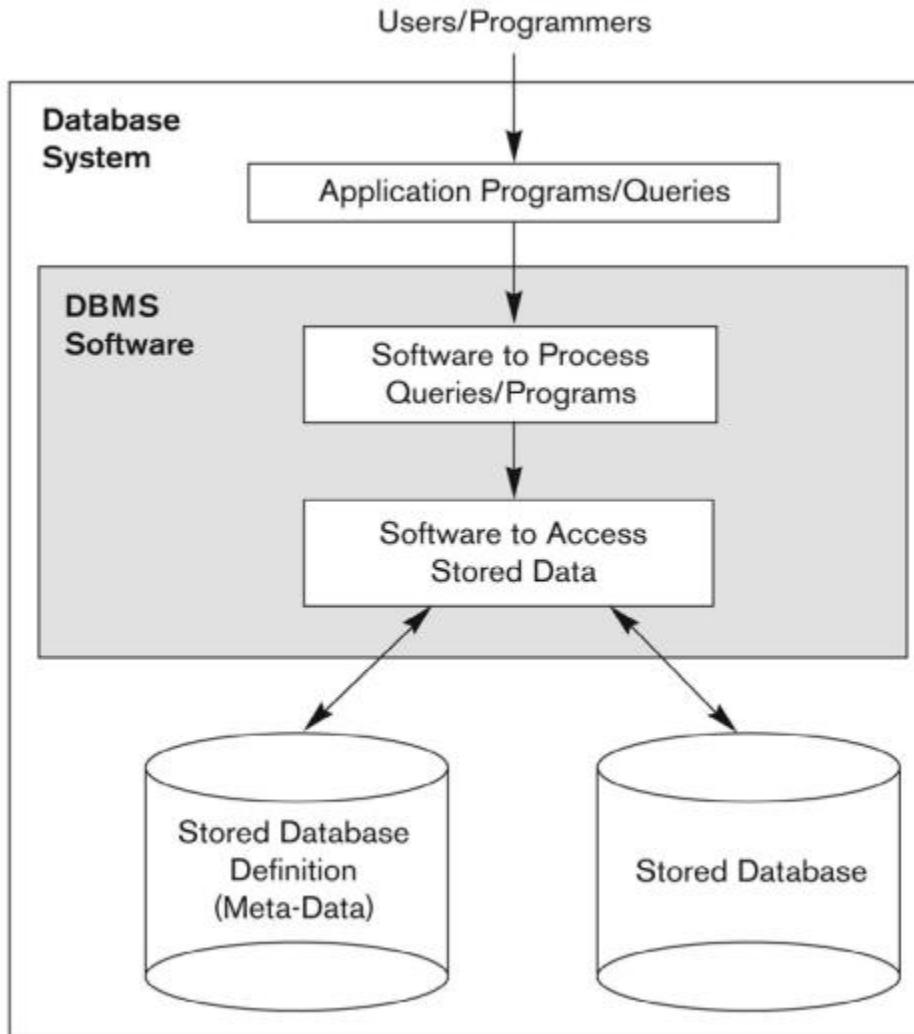
- means storing data into the database structures and storing on some storage medium

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

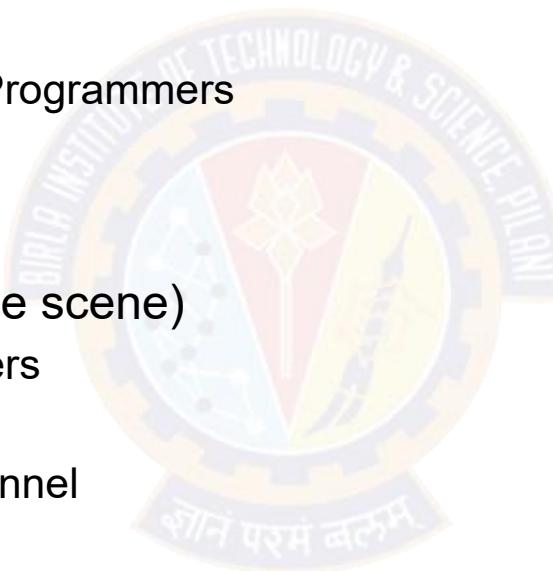
DBMS - Actors

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



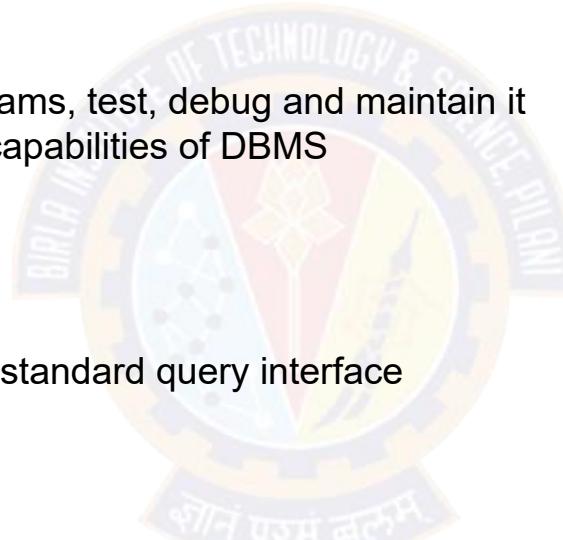
On the scene actors

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

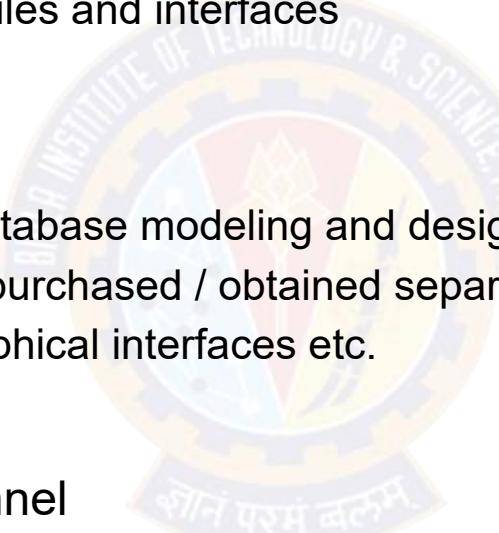
- System Analysts
 - Determine the requirements of end users and prepares the standard canned transactions
- Application Programmers
 - Implements the specifications as programs, test, debug and maintain it
 - Needs to be familiar with full range of capabilities of DBMS
- End Users
- Casual
 - occasionally 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



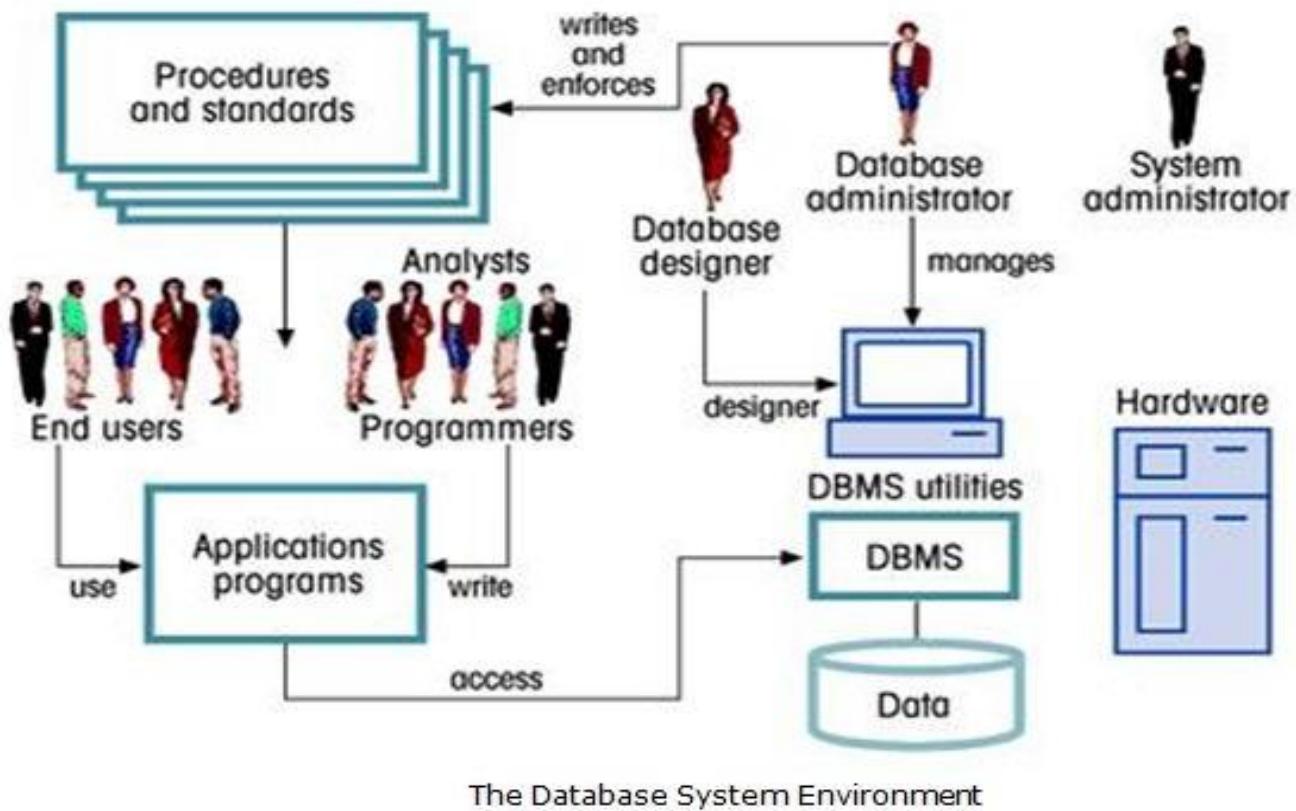
Behind the Scene

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

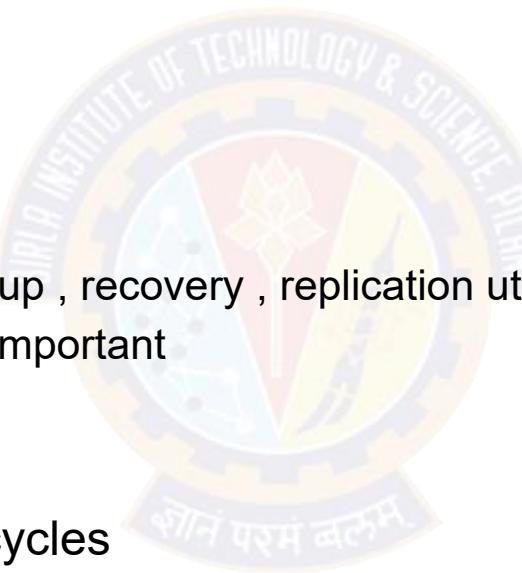
- Data independence
- Efficient data access
- Data integrity and security
- Easier Data Administration
- Concurrent access and Crash recovery
- Reduced application development time



DBMS

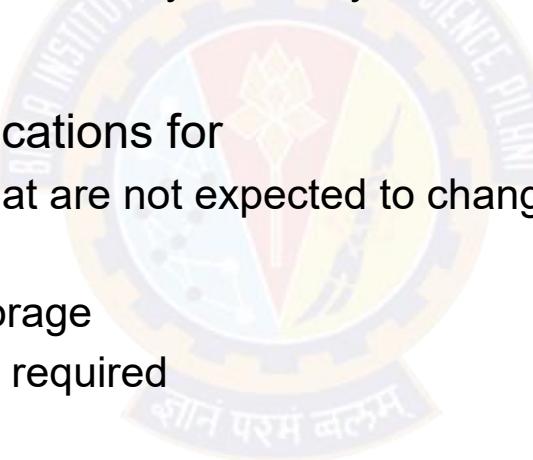
Disadvantages

- 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

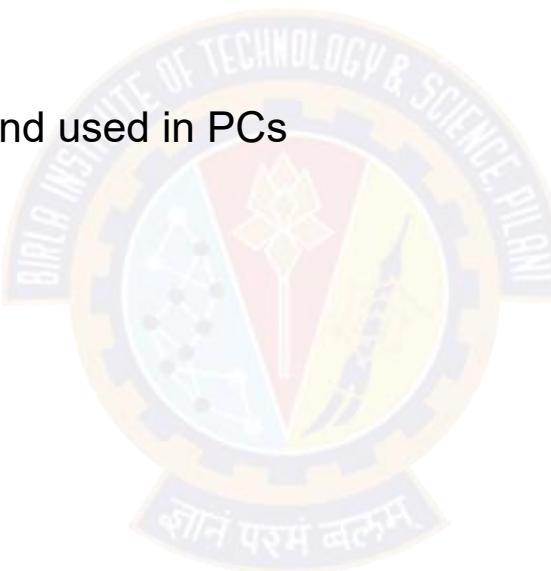
- Relational Data model
 - Quite common in commercial DBMS products
 - Aka SQL systems
 - Oracle , SQLServer, MySQL
- Object Data model
 - Not much usage - Tornado
- NoSQL Data model
 - Aka Big Data systems
 - Further classified as
 - ❖ Document based – MongoDB
 - ❖ Graph based – Neo4J
 - ❖ Column based – 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

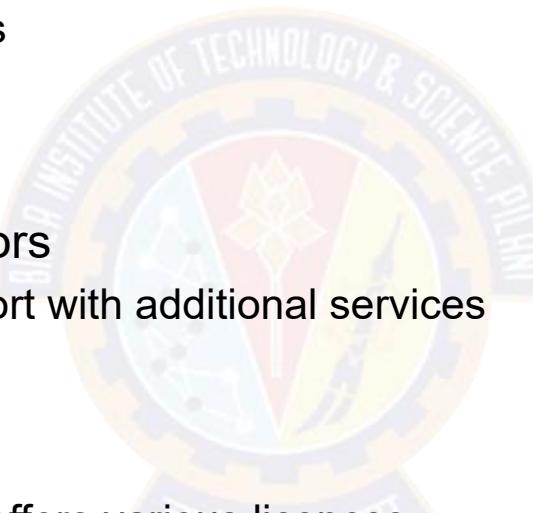
- Number of places where database is distributed
- Centralized
 - Data is stored at a single computer site
 - Can support multiple users
 - Prone to failure
- Distributed
 - Actual database and DBMS software distributed over many sites connected by network
 - Big Data systems are common with massive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



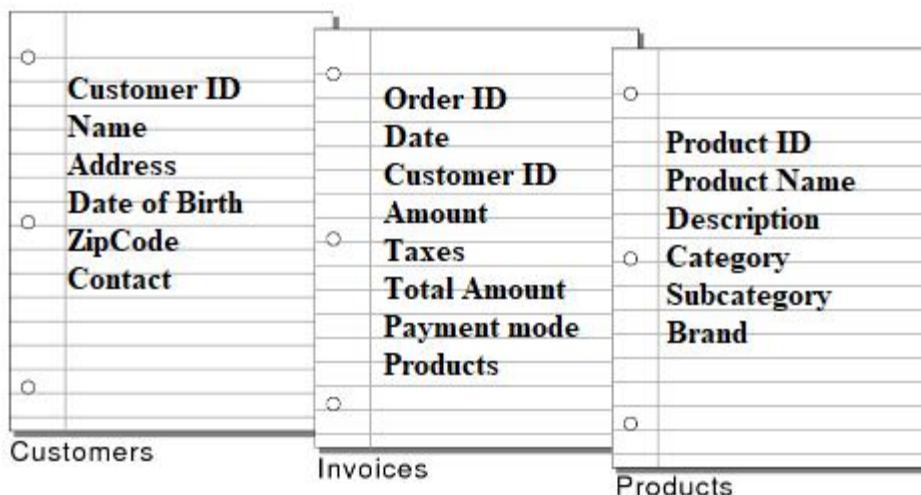
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

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

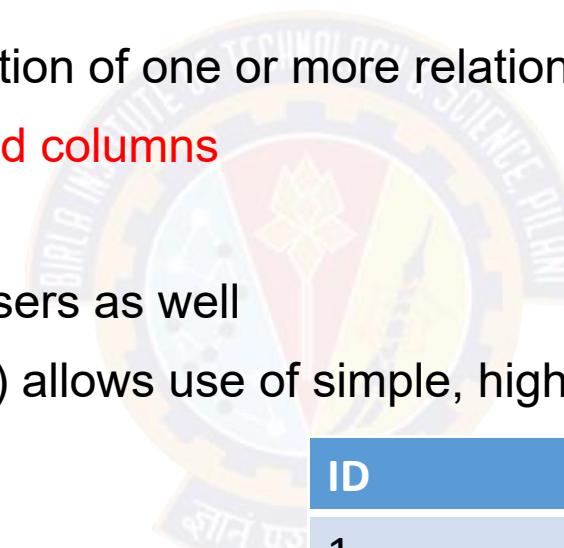


Relational model

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

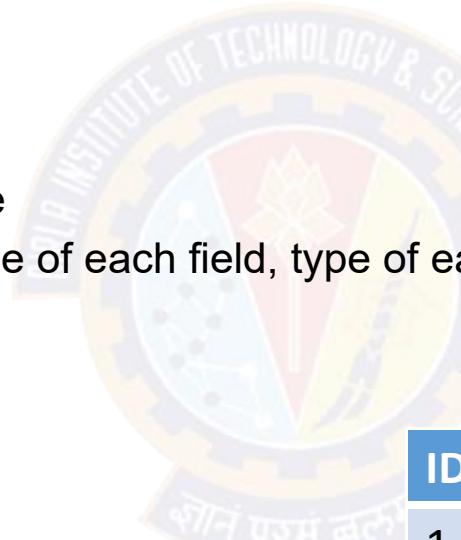
Relation

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



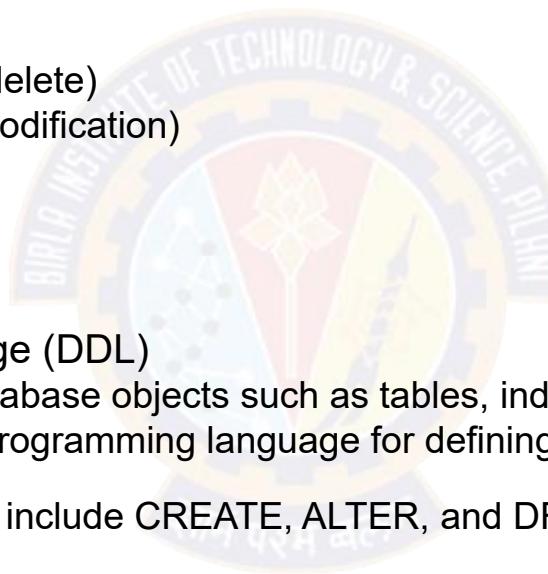
STUDENT		
STUDENT_ID (PK)	NUMBER(8,0)	NOT NULL
SALUTATION	VARCHAR2(5)	NULL
FIRST_NAME	VARCHAR2(25)	NULL
LAST_NAME	VARCHAR2(25)	NOT NULL
STREET_ADDRESS	VARCHAR2(50)	NULL
ZIP (FK)	VARCHAR2(5)	NOT NULL
PHONE	VARCHAR2(15)	NULL
EMPLOYER	VARCHAR2(50)	NULL
REGISTRATION_DATE	DATE	NOT NULL
CREATED_BY	VARCHAR2(30)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

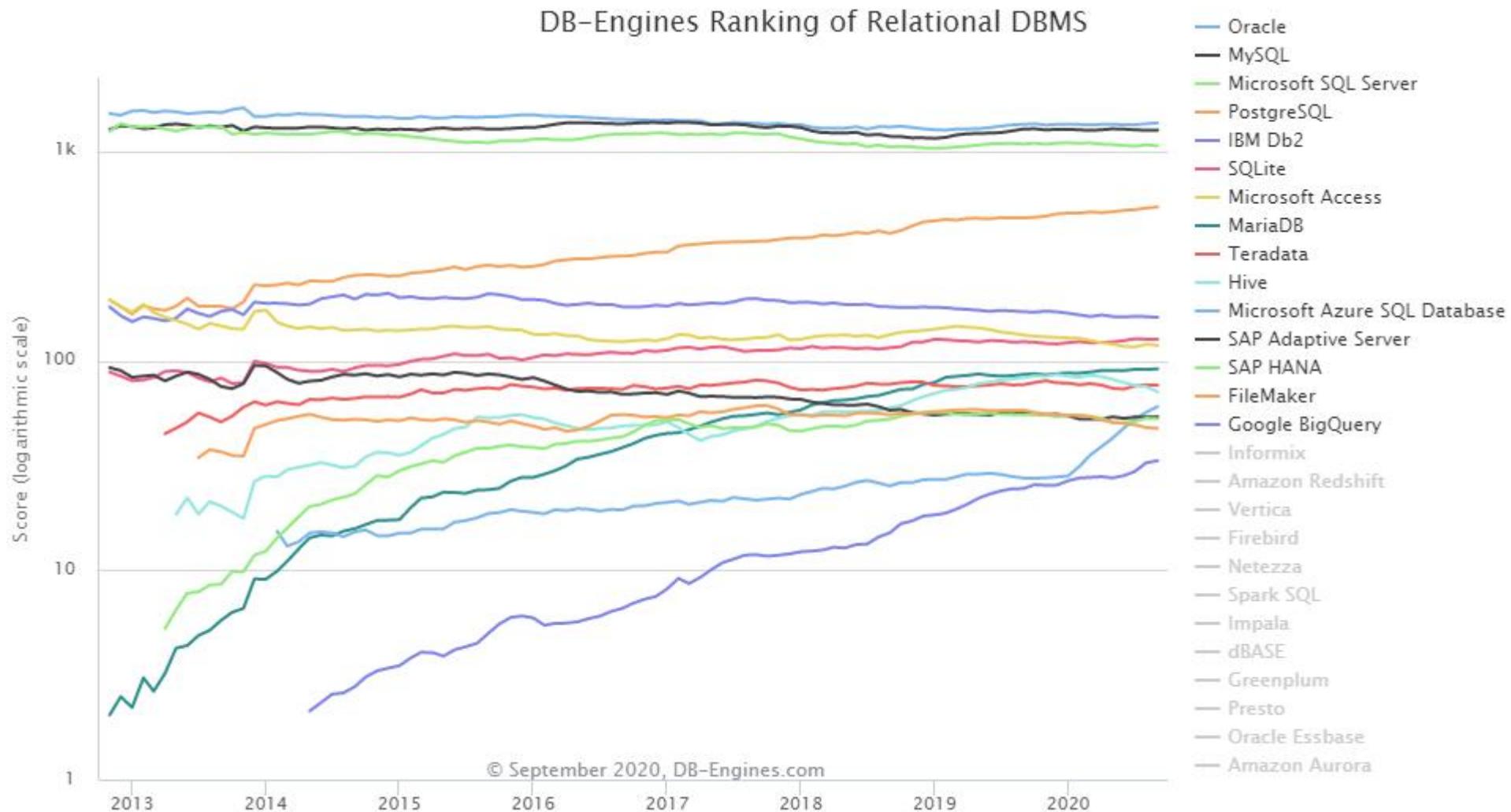


BITS Pilani
Pilani | Dubai | Goa | Hyderabad

RDBMS - Landscape

Chandan Ravandur N

RDBMS Landscape



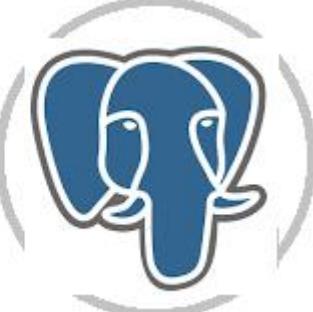
Source : [db-engines](https://db-engines.com)

Dominant Players

- Commercial Vendors



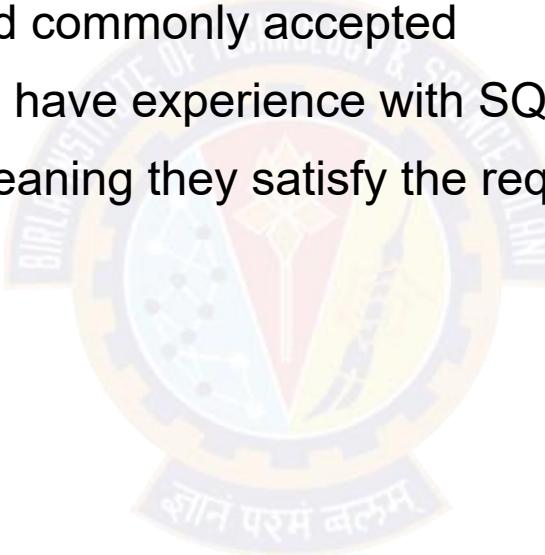
- Open Source Challengers



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 : DataVarsity

Emergence of NoSQL systems

Use cases

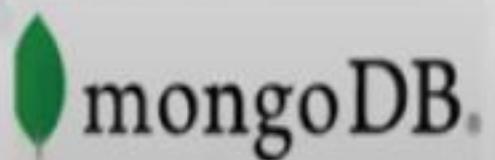
- Free email application like Gmail
 - Have millions of users, each user with thousands of emails – need for storage
 - SQL may not be appropriate because
 - ✓ It has too many services which are not required here
 - ✓ Structured data model is restrictive
- Facebook – social media platforms
 - Have millions 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)

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 21st century web applications



Categories (1)

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



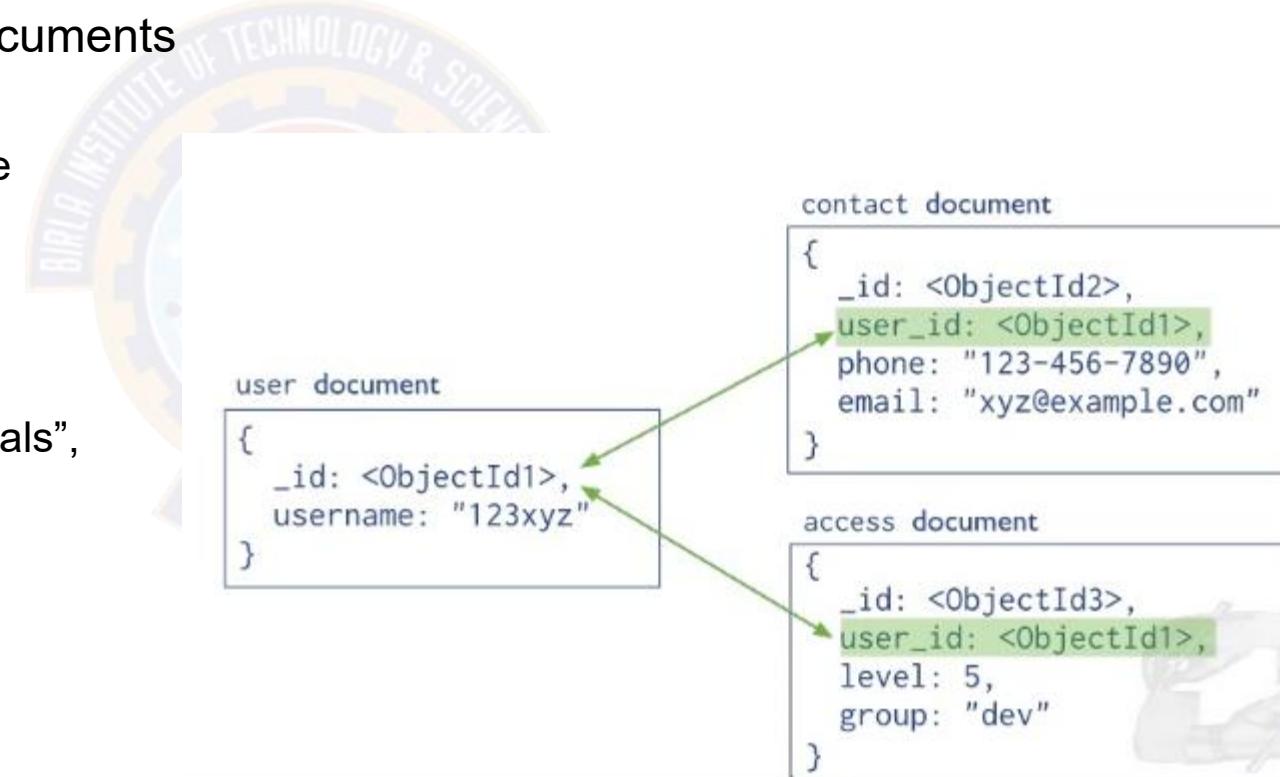
Key-value stores	Column-Oriented	Document based	Graph based
Riak	Cassandra	MongoDB	Neo4J
Redis	Hbase	CouchDB	InfiniteGraph
Membase	HyperTable	RavenDB	AllegroGraph

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"}
```

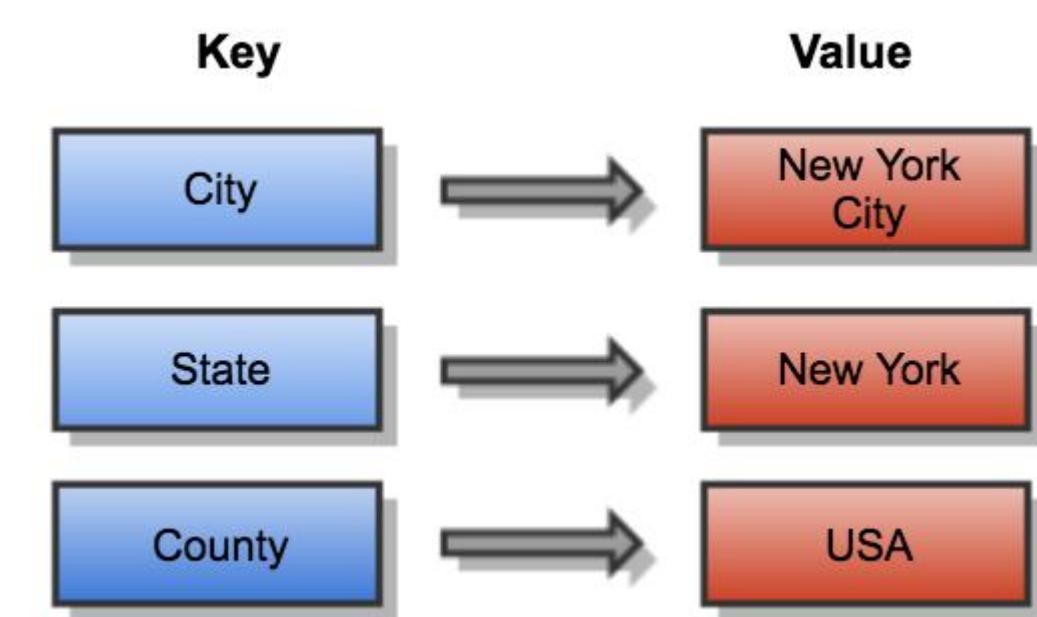


Categories (3)

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}



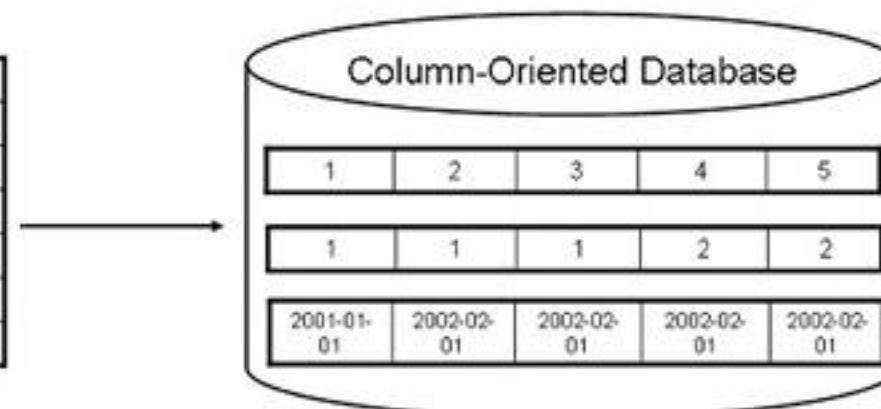
Categories (4)

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



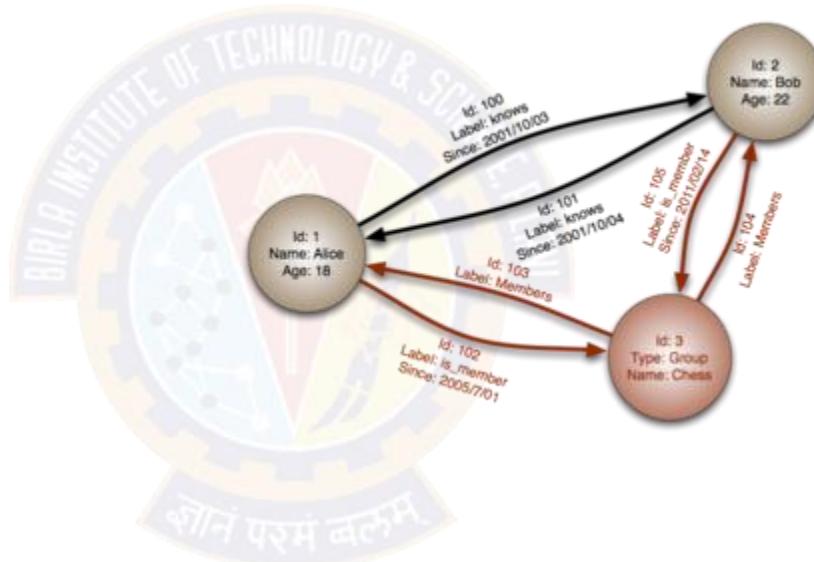
Emp_no	Dept_id	Hire_date	Emp_fn	Emp_ln
1	1	2001-01-01	Smith	Bob
2	1	2002-02-01	Jones	Jim
3	1	2002-05-01	Young	Sue
4	2	2003-02-01	Sternle	Bill
5	2	1999-06-15	Aurora	Jack
6	3	2000-08-15	Jung	Laura



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

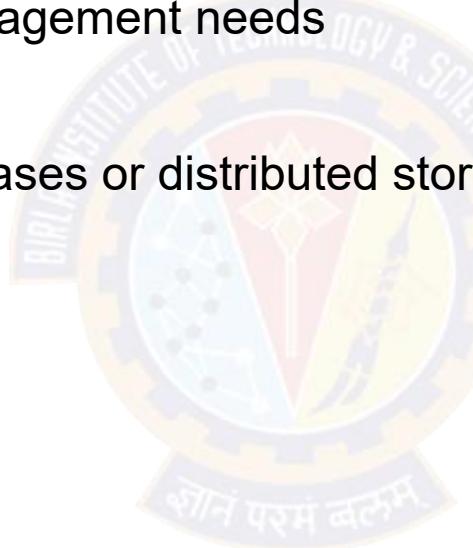


Image Source : DataVarsity

Emergence of NoSQL systems

Use cases

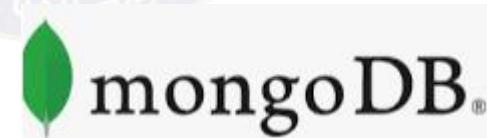
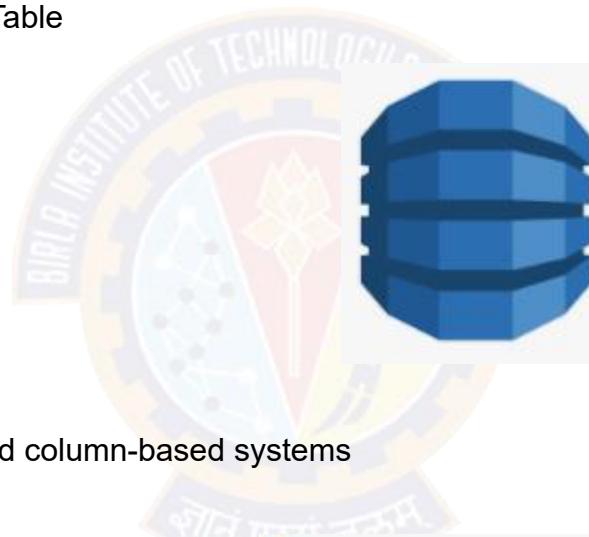
- Free email application like Gmail
 - Have millions of users, each user with thousands of emails – need for storage
 - SQL may not be appropriate because
 - ✓ It has too many services which are not required here
 - ✓ Structured data model is restrictive
- Facebook – social media platforms
 - Have millions 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 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
 - ✓ 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

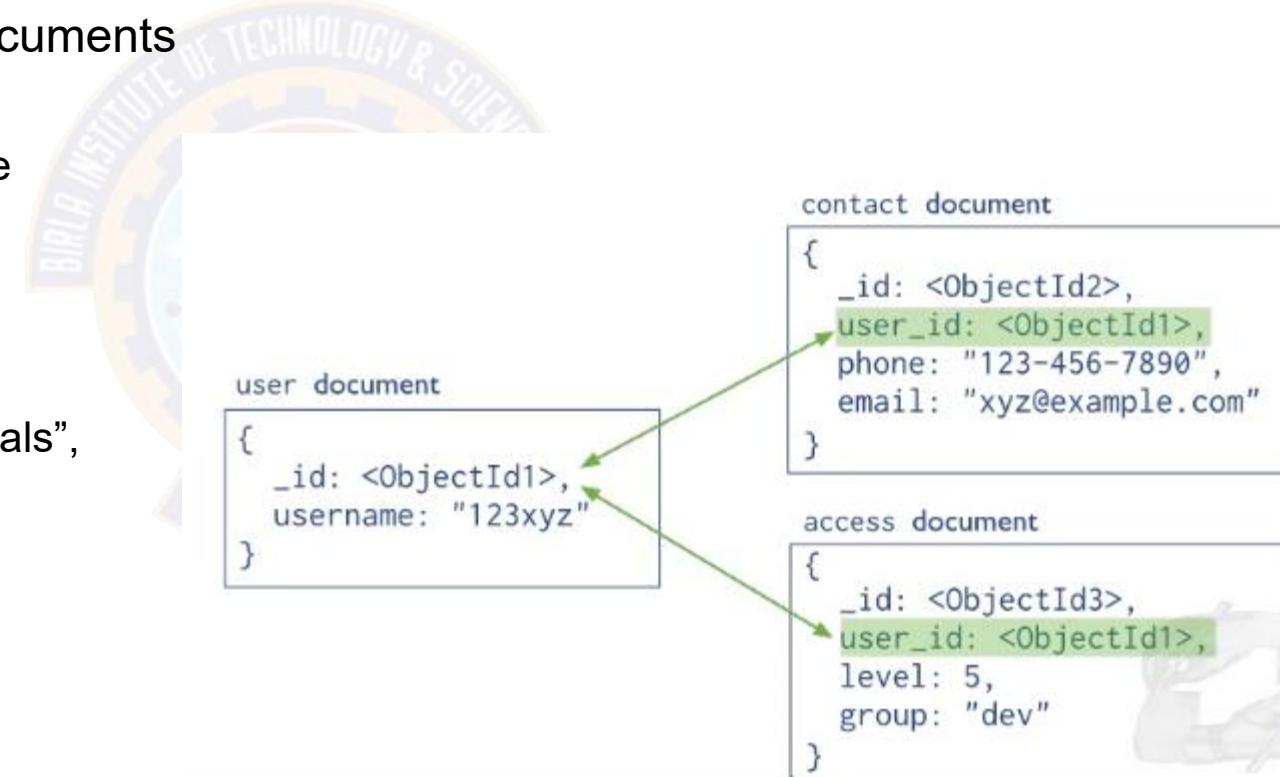
Chandan Ravandur N

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



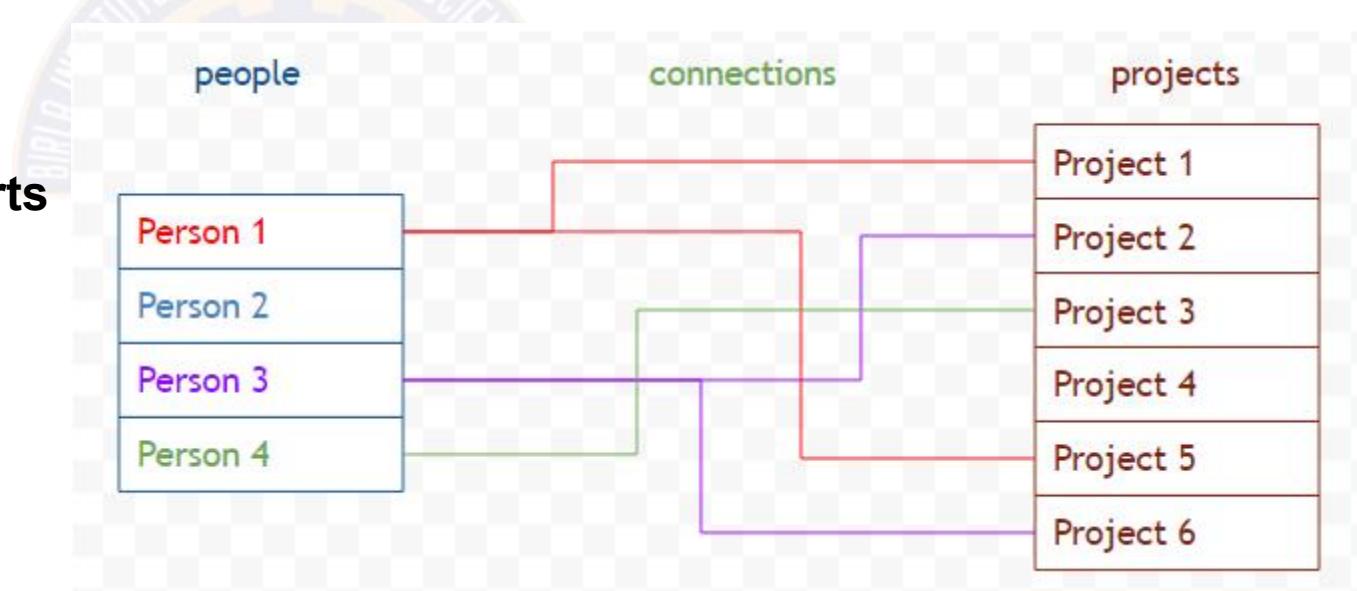
[Image Source : MongoDB](#)

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](#)

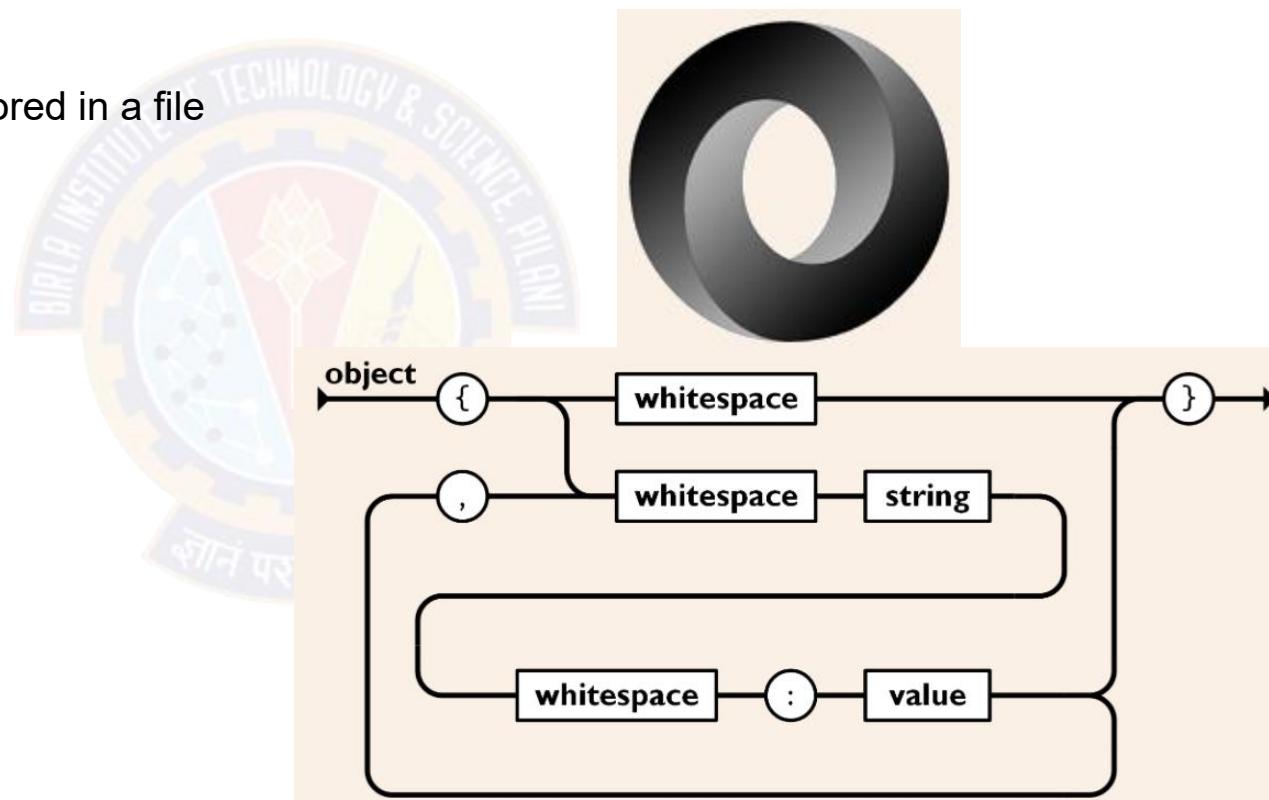
Why? (2)



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  
}
```

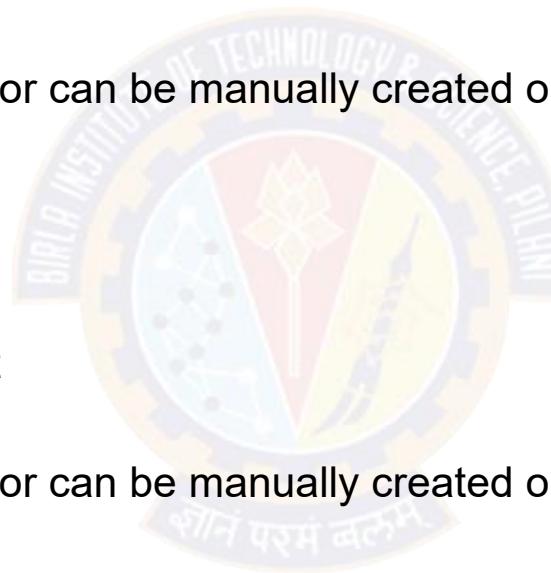


[Image Source : Json.org](http://Json.org)

MongoDB concepts

Database, Collection and Document

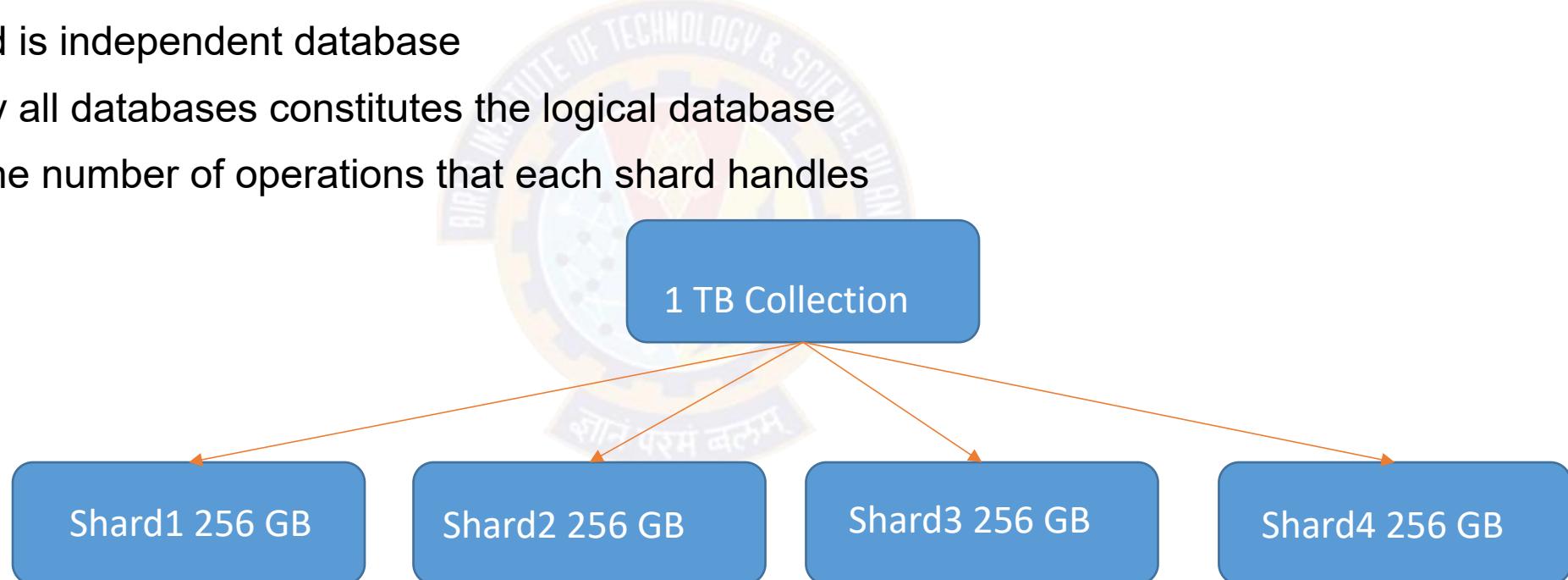
- Database
 - ✓ Collection of collections
 - ✓ Container for collections
 - ✓ Gets created when its referenced or can be manually created on demand
- Collection
 - ✓ Similar to table in RDBMS
 - ✓ Holds multiple documents within it
 - ✓ No enforcement of Schema
 - ✓ Gets created when its referenced or 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

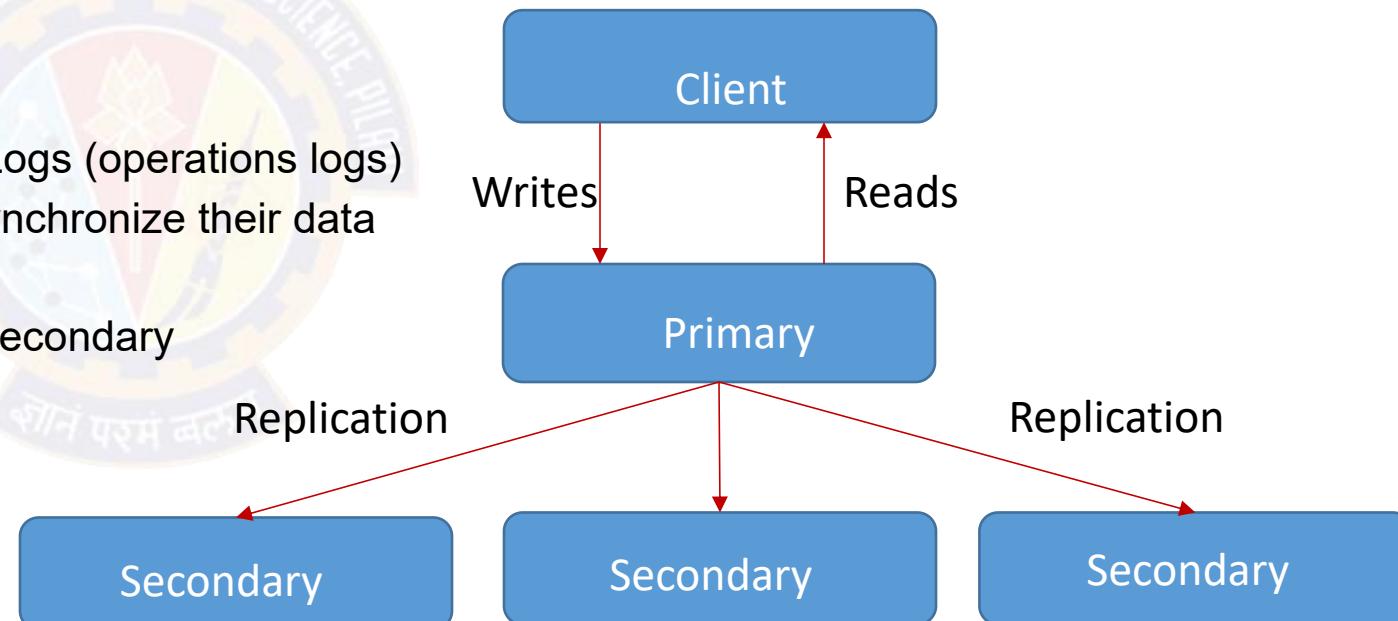
- For read and write operations (CRUD)
- Data Aggregation
- Text Search and Geospatial Queries



MongoDB Features(3)

Replication

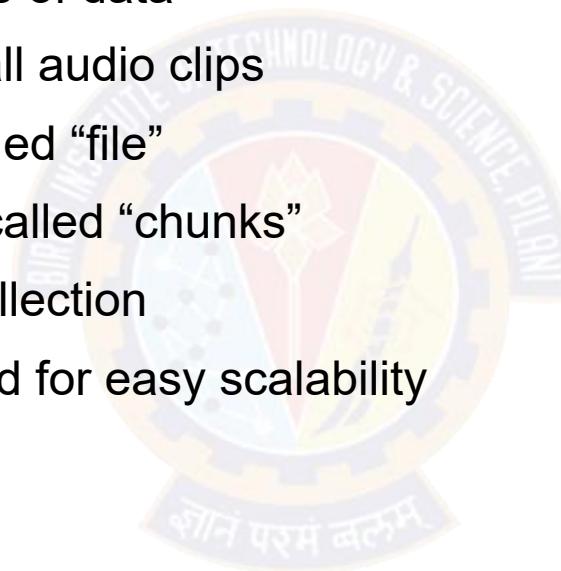
- Supports data redundancy and high availability
- Helps to recover from hardware failure and service interruptions
- Replica has single primary and several secondary's
- Working
 - ✓ Write is directed to primary
 - ✓ Primary then logs all write requests to OpLogs (operations logs)
 - ✓ OpLog is used 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



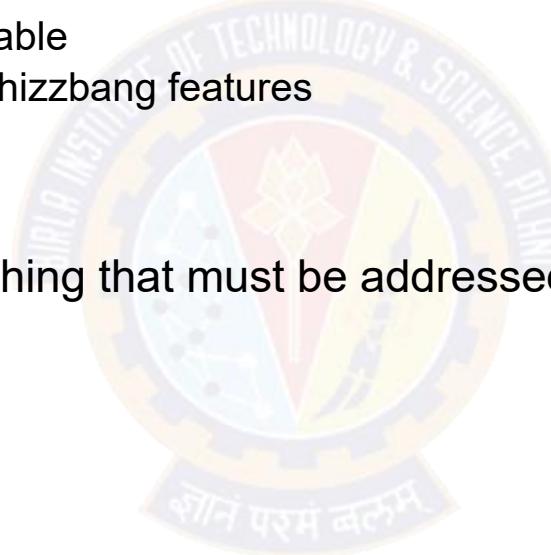
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Today's application requirements

Chandan Ravandur N

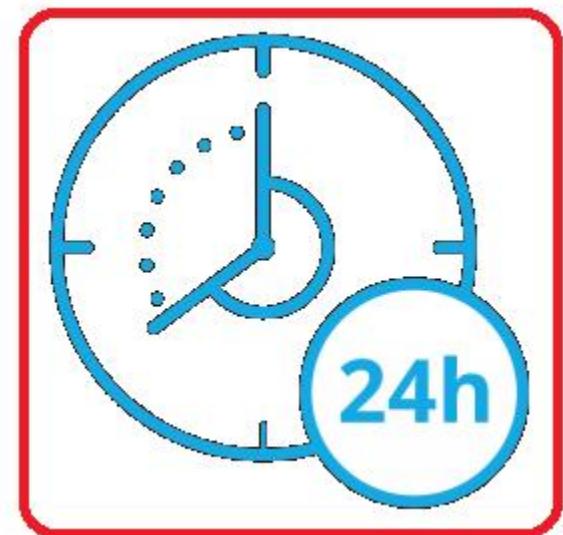
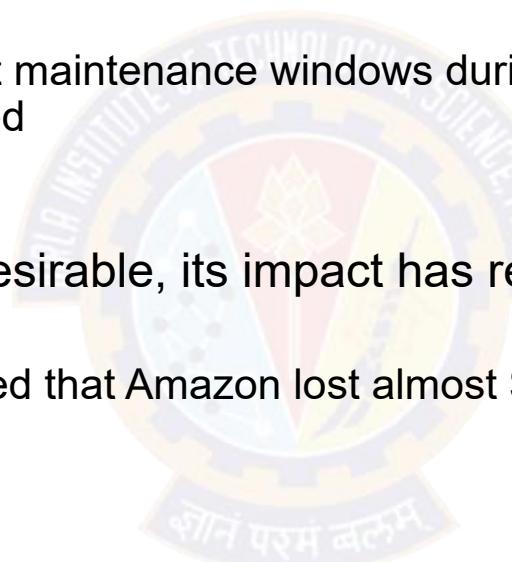
Today's application requirements

- 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



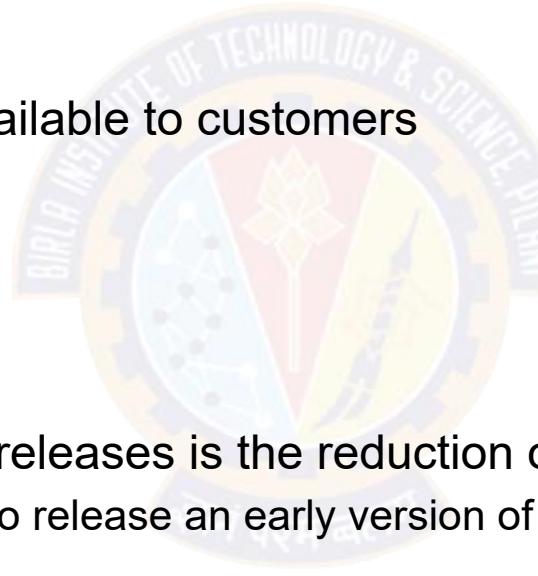
Zero downtime

- 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:



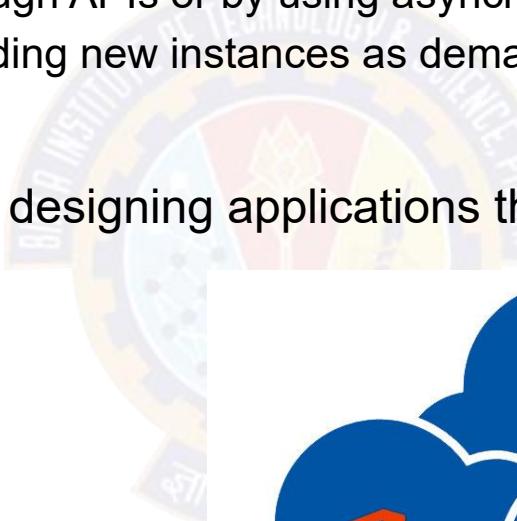
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

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

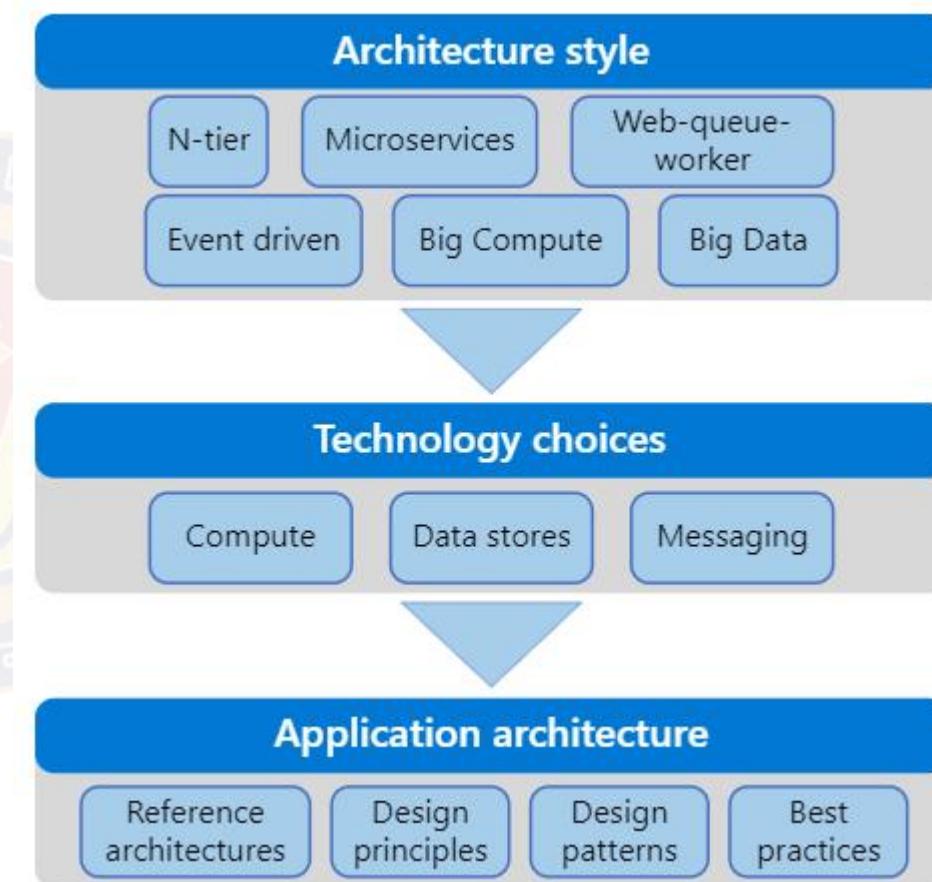
Compared

Traditional on-premises	Modern cloud
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

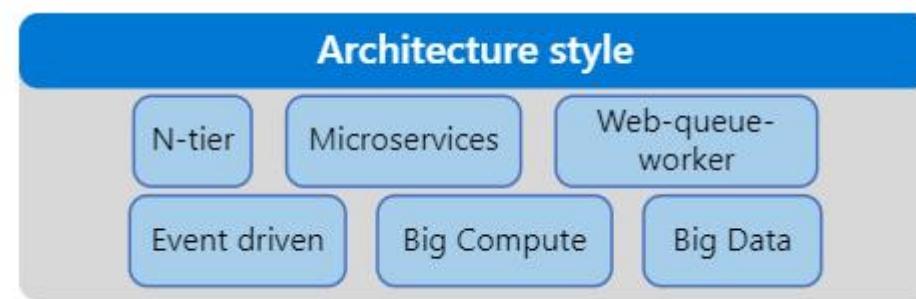
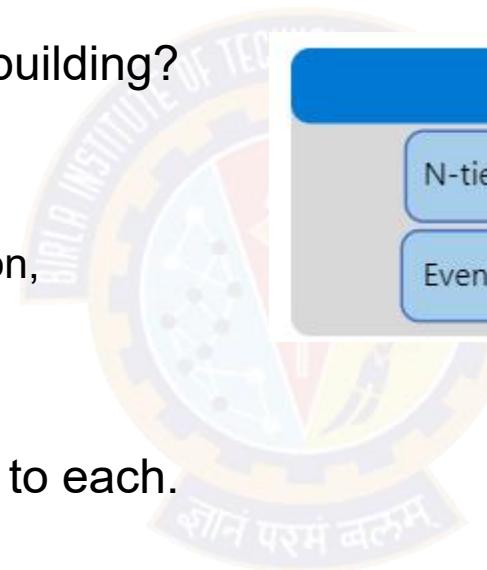
- Series of steps, from the architecture and design to implementation are involved
- For each step, there is supporting guidance needed to design application architecture



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.

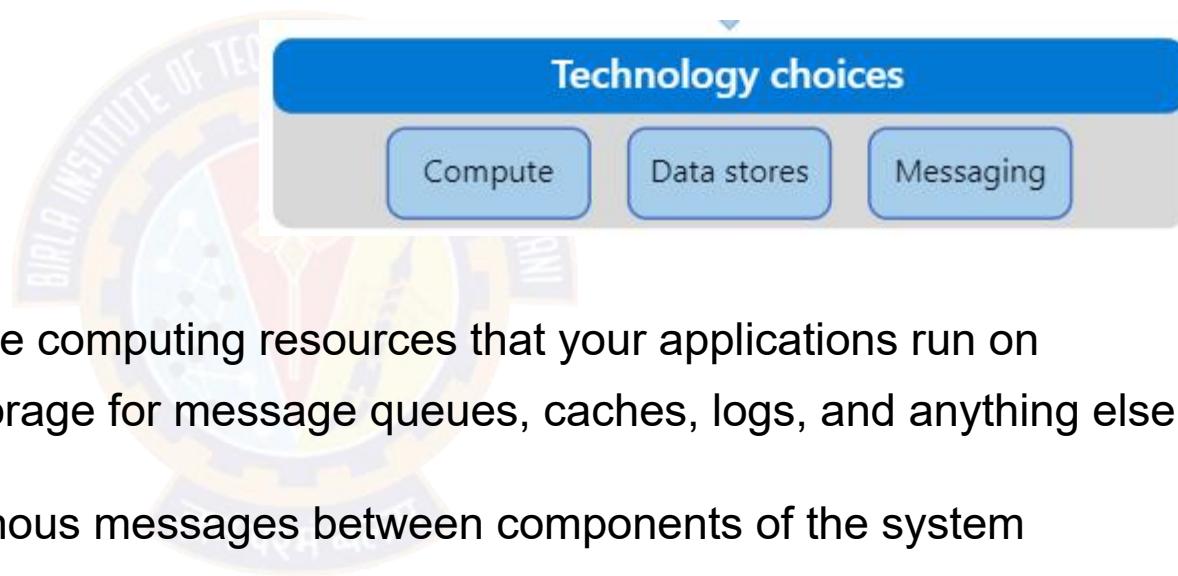


Technology choices

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

- Critical Technology choices are:

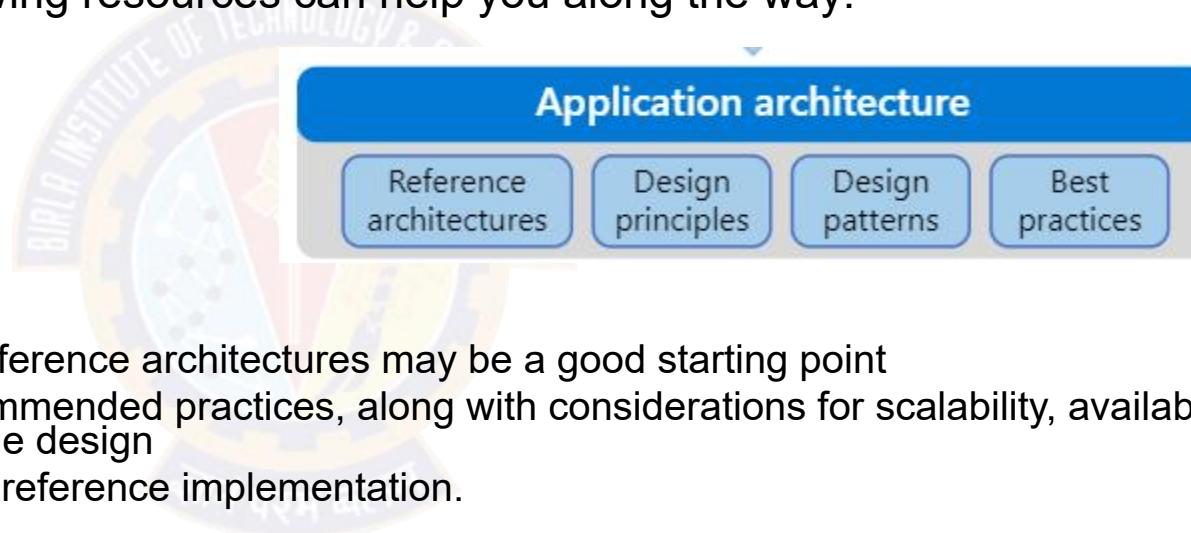
- ✓ Compute
- ✓ Data stores
- ✓ Messaging



- 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:



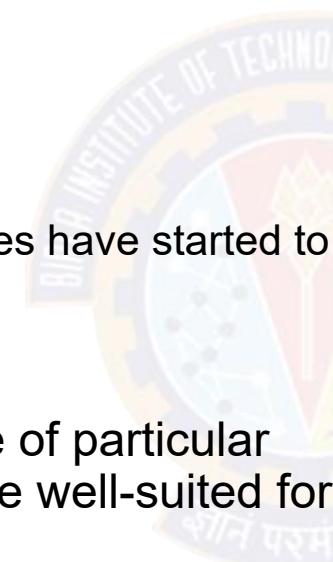
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

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

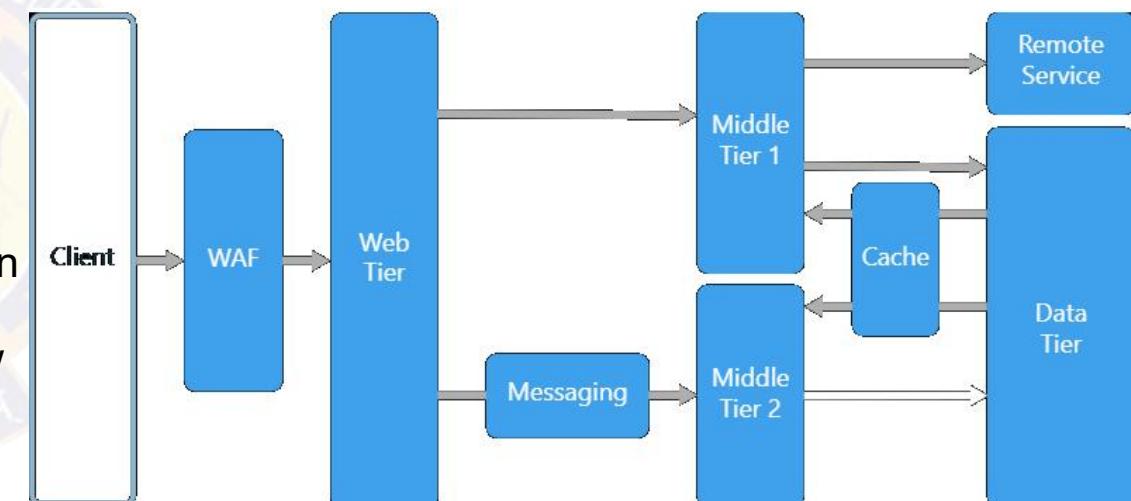
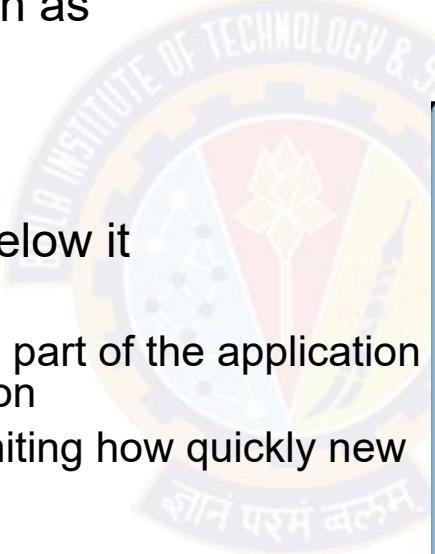
A quick tour of the styles

- 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

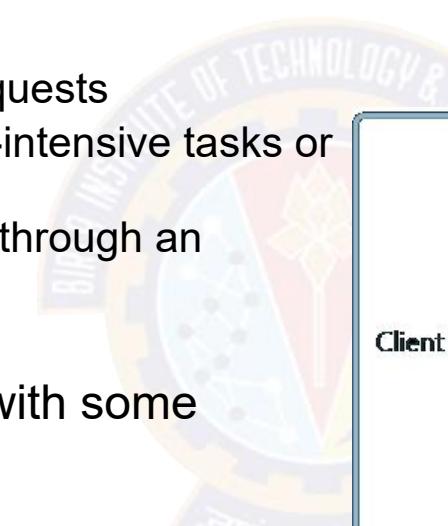
- A traditional architecture for enterprise applications
- Dependencies are managed by dividing the application into layers that perform logical functions, such as
 - ✓ presentation,
 - ✓ business logic,
 - ✓ and data access
- A layer can only call into layers that sit below it
 - ✓ this horizontal layering 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

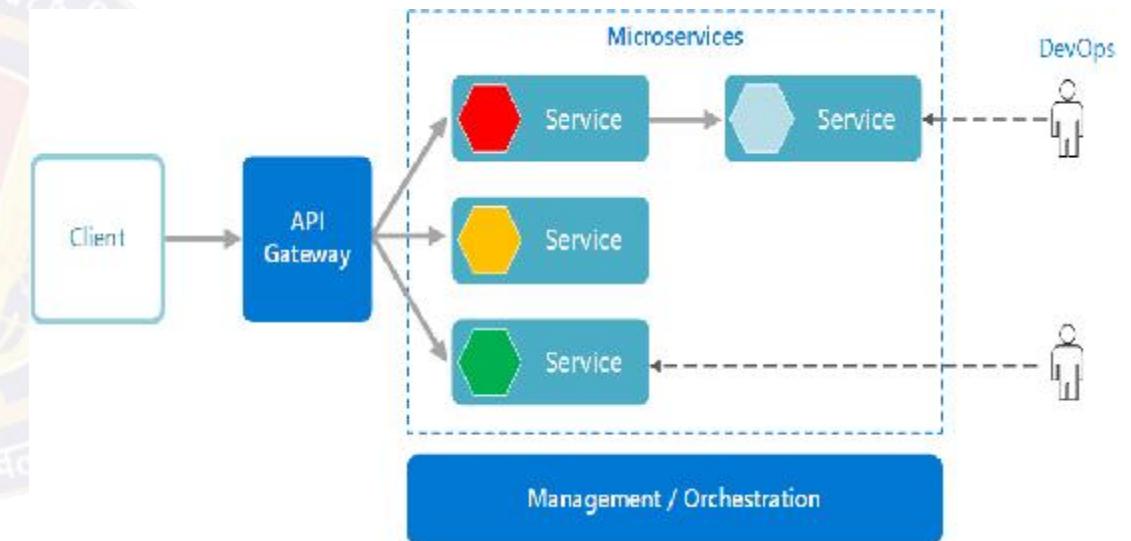
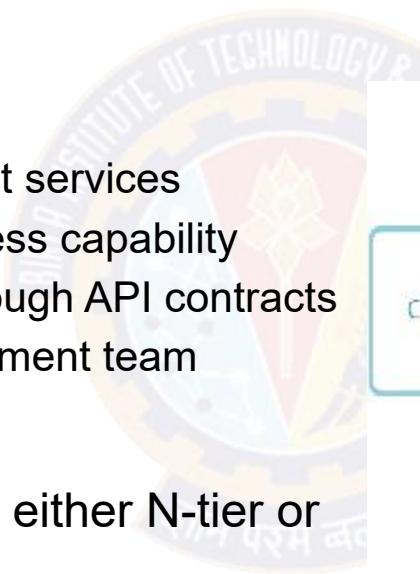
- For a purely PaaS solution, consider a Web-Queue-Worker architecture
- Application has
 - ✓ a web front end that handles HTTP requests
 - ✓ a back-end worker that performs CPU-intensive tasks or long-running operations
 - ✓ front end communicates to the worker through an asynchronous message queue
- 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

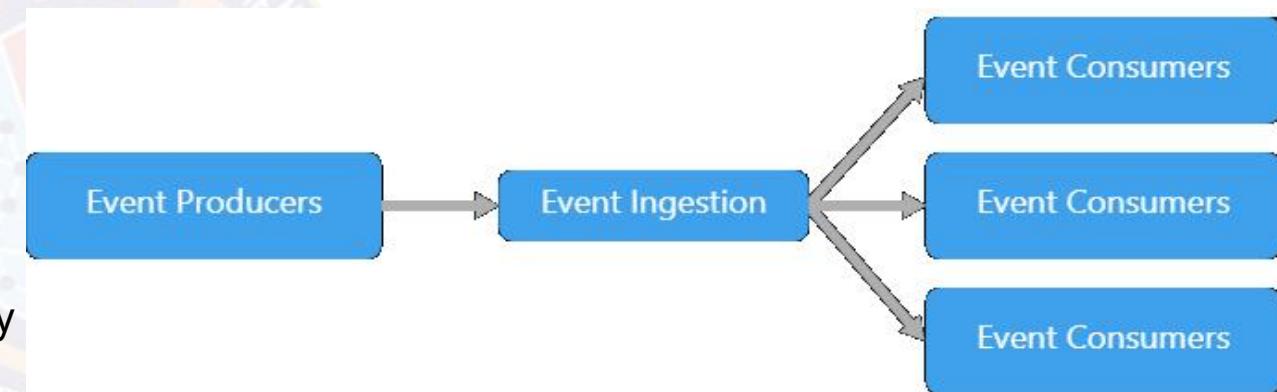
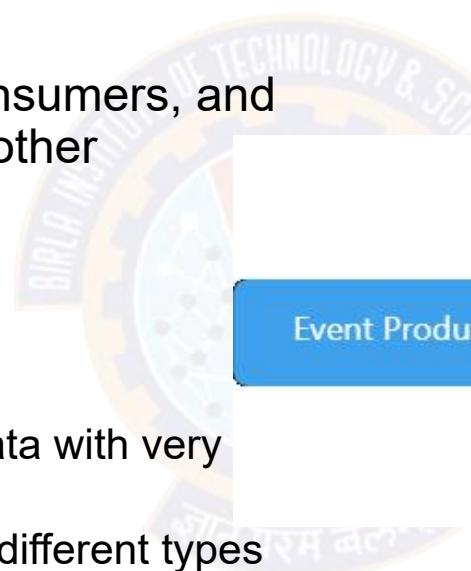
- If your application has a more complex domain, consider moving to a Microservices architecture
- Microservice Application
 - ✓ is composed of many small, independent services
 - ✓ each service implements a single business capability
 - ✓ are loosely coupled, communicating through API contracts
 - ✓ 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

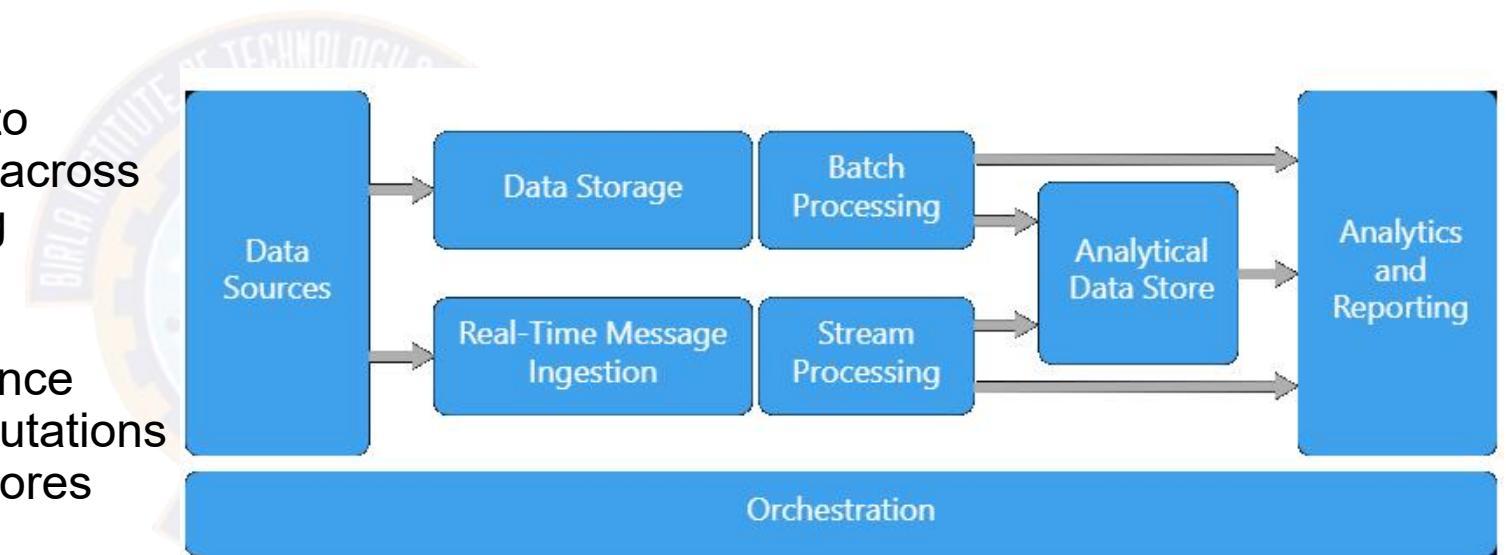
- Use a publish-subscribe (pub-sub) model, where producers publish events, and consumers subscribe to them
- Producers are independent from the consumers, and consumers are independent from each other
- Useful for applications that
 - ✓ ingest and process a large volume of data with very low latency, 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**

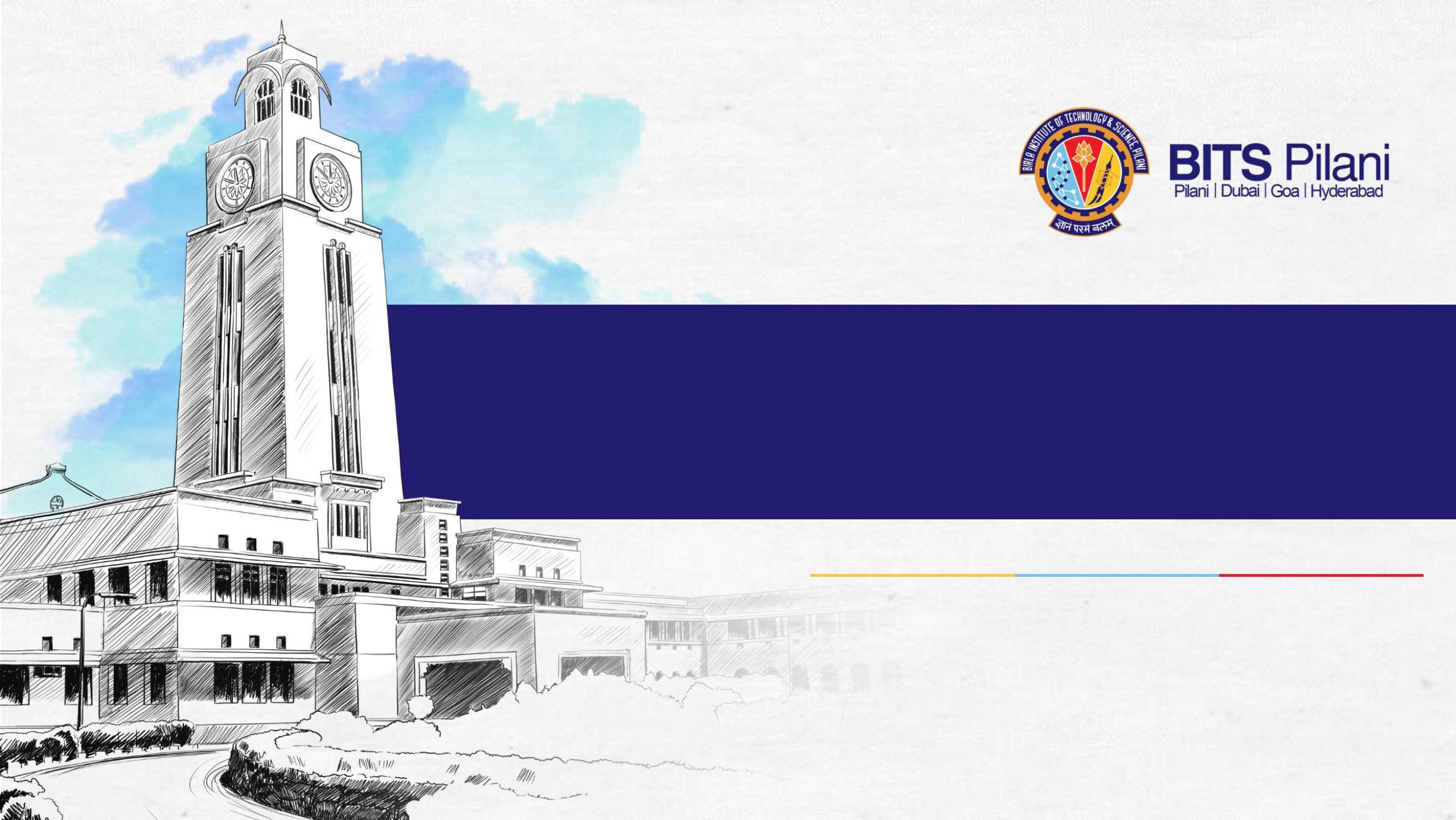




BITS Pilani
Pilani | Dubai | Goa | Hyderabad

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.

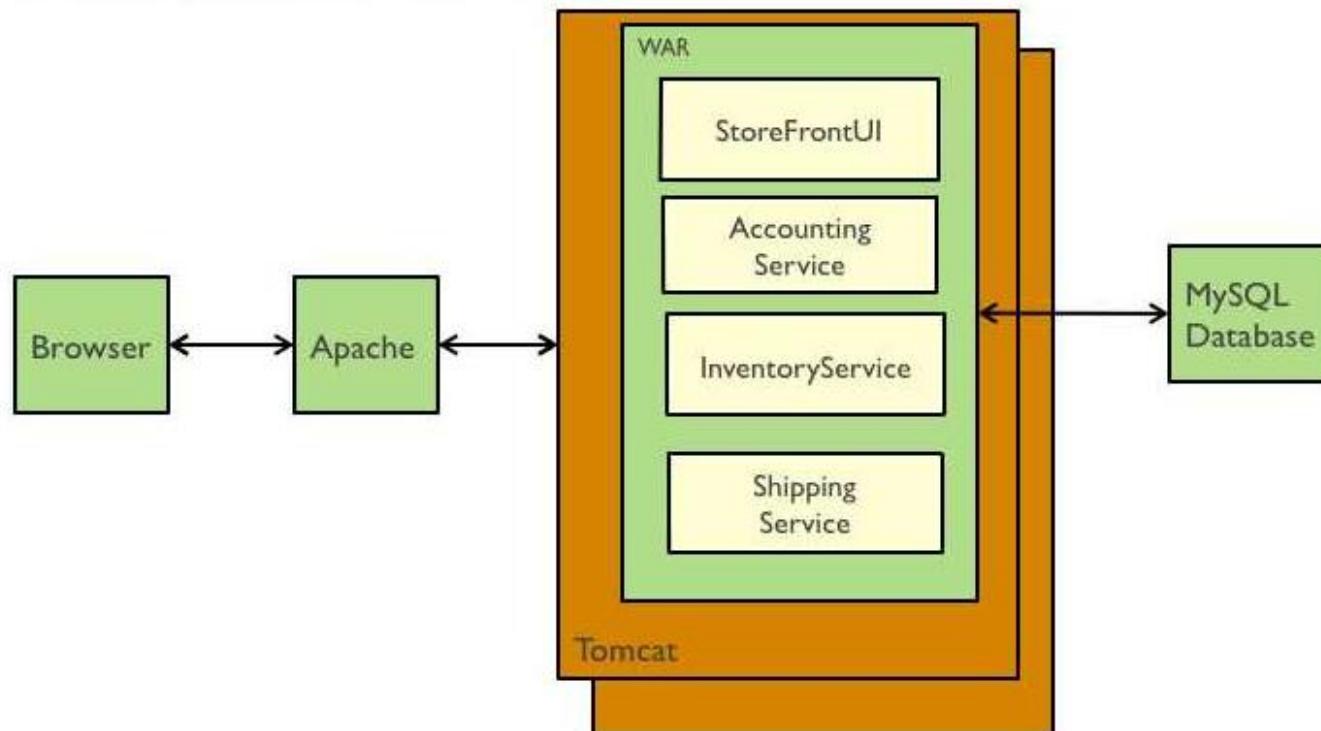
Problem (Forces) and Solution

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

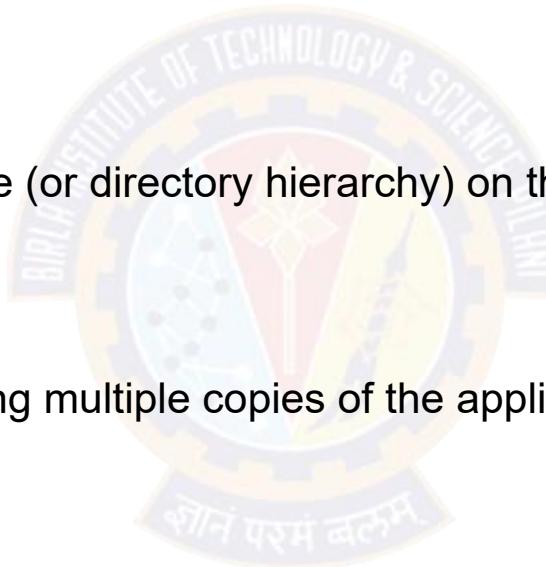
Example

- 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



Thank You!

In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Types of Monoliths

Chandan Ravandur N

Monolith

The unit of deployment

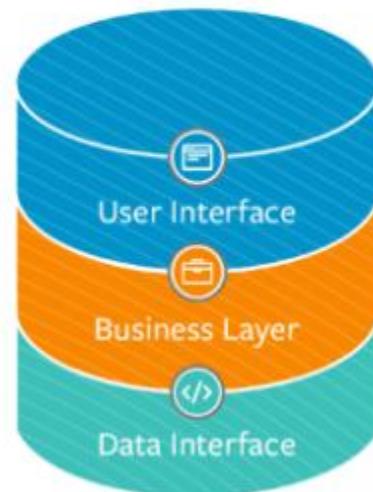
- When all the functionality in a system had to be deployed together
- Three types
 - ✓ Single process monolith
 - ✓ Distributed monolith
 - ✓ Third party black-box systems



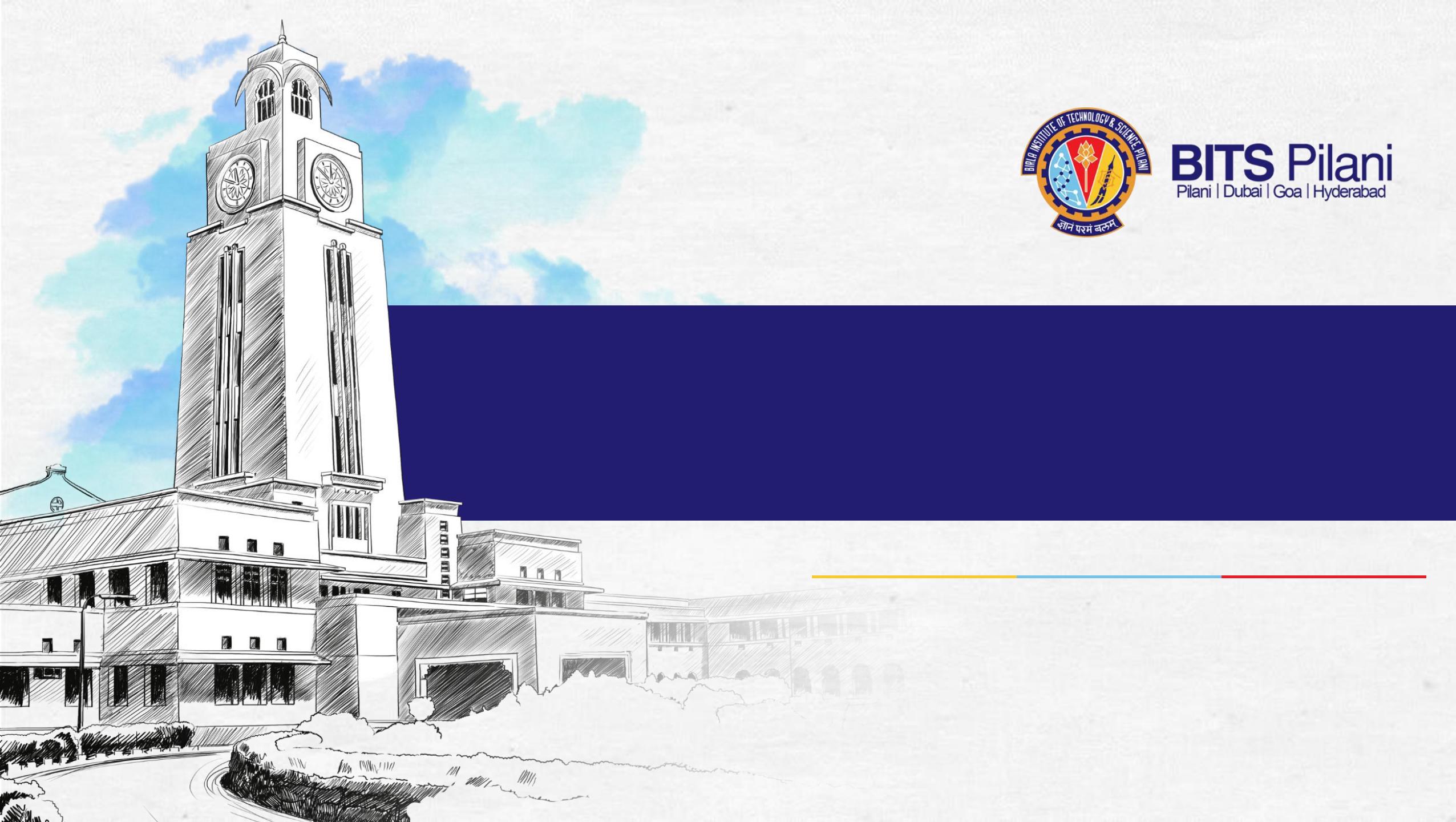
Single Process Monolith

- 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



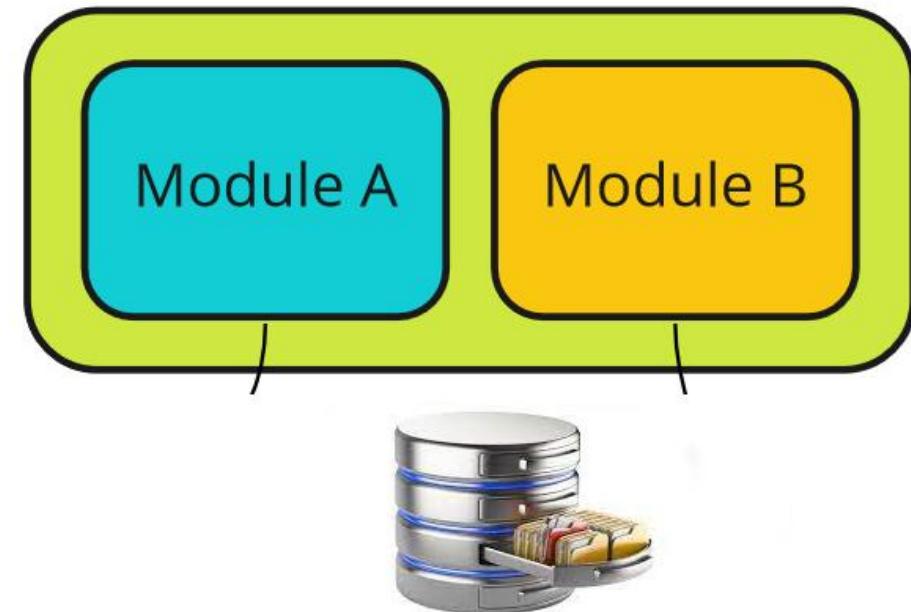
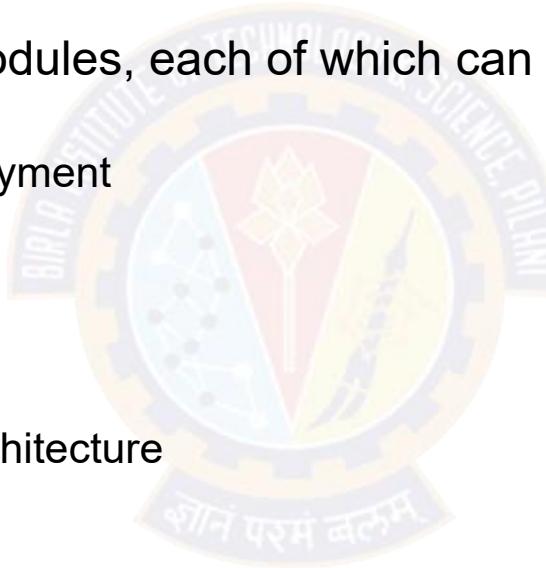
[source](#)



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

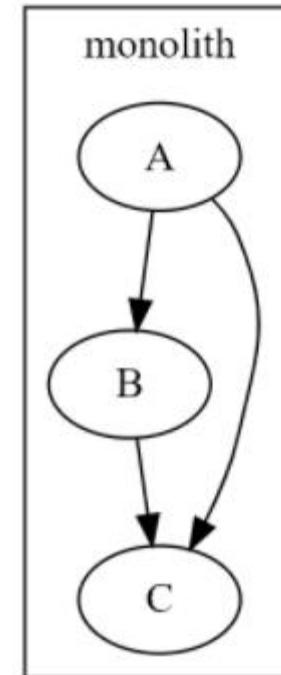
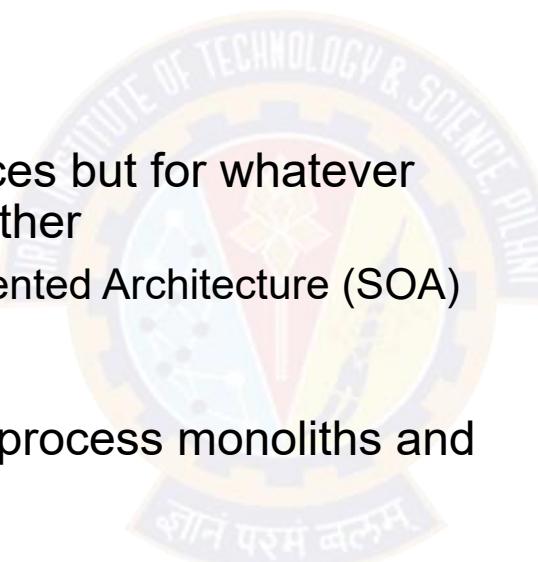
Modular monolith

- 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



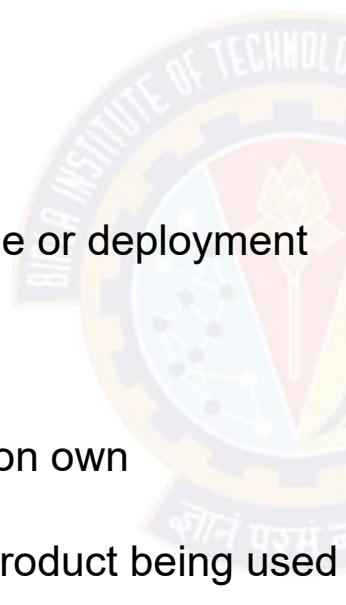
Distributed monolith

- 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

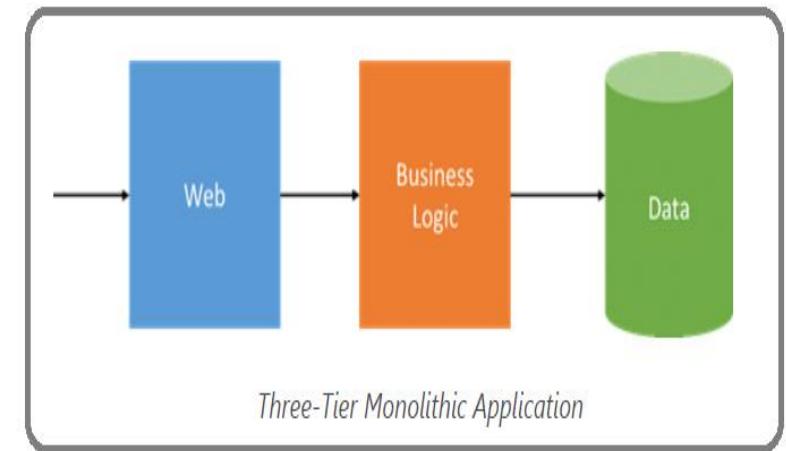
- Application development and IT system management revolution is driven by the cloud
- Operational efficiency and faster-time-to-market is being enabled by
 - Fast, agile, inexpensive, and massively scalable infrastructure
 - fully self-service applications
 - pay-as-you-go billing model
 - emergence 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

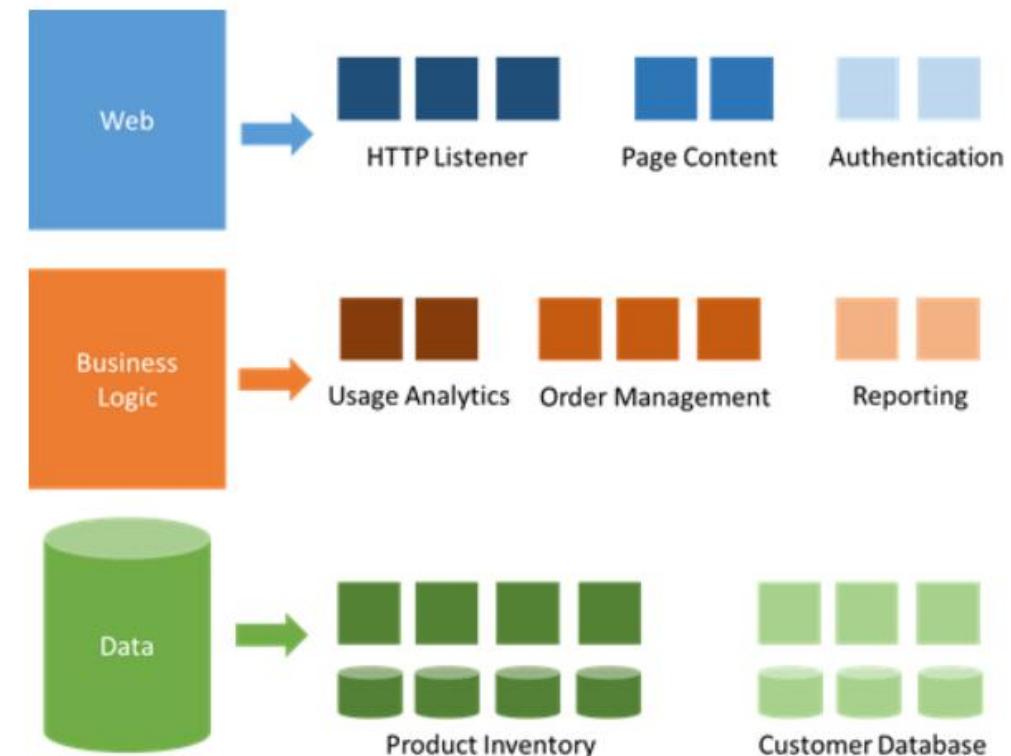
Challenges

- 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



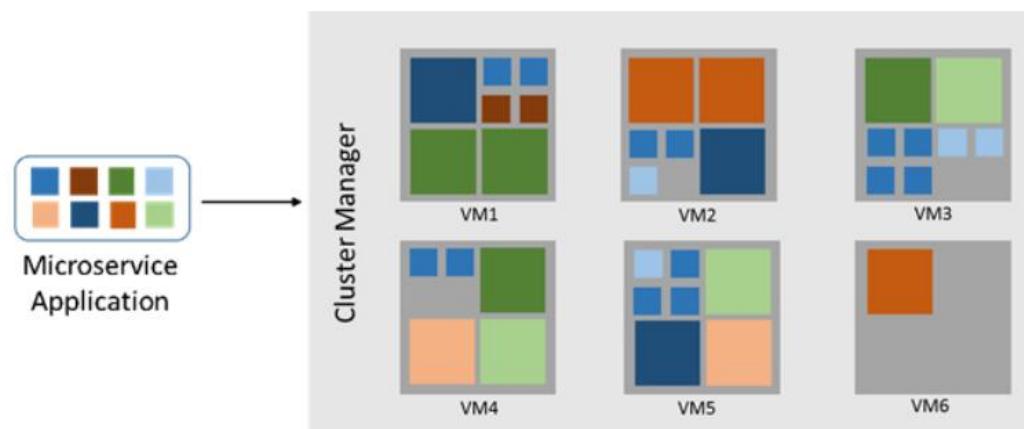
Breaking the Monolith into Microservices

[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:



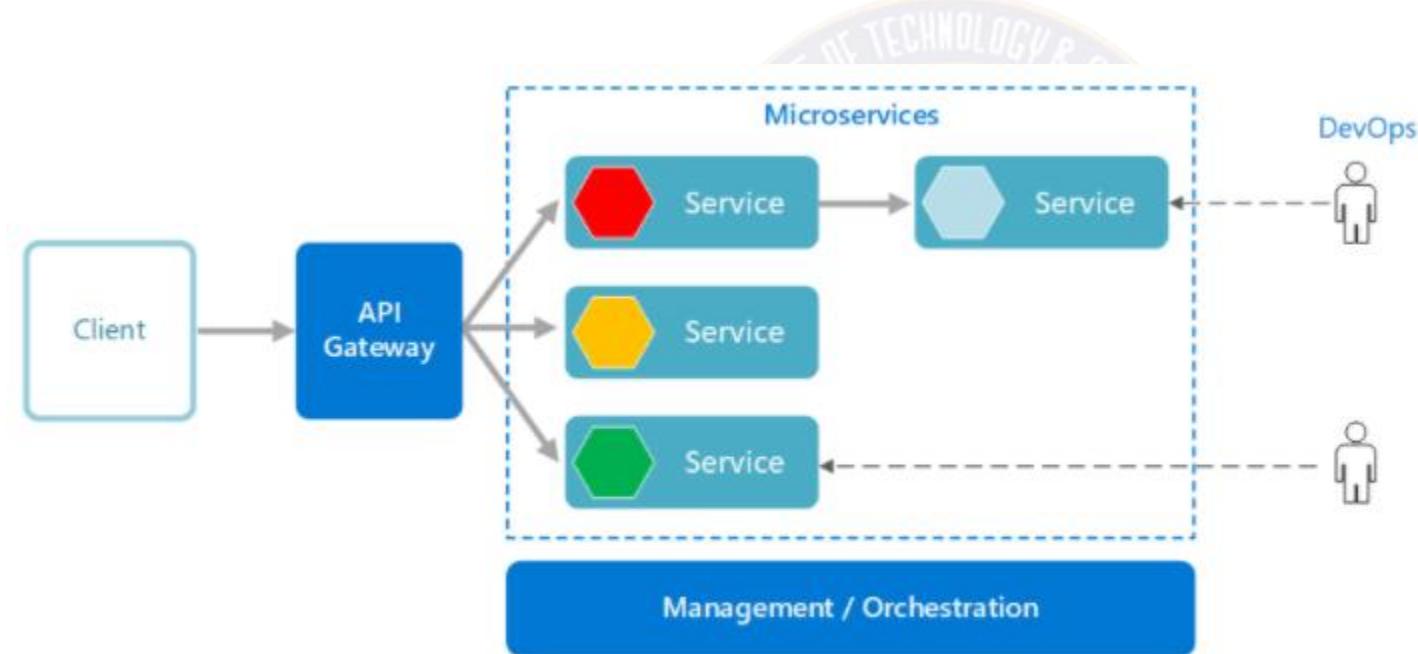
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

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)

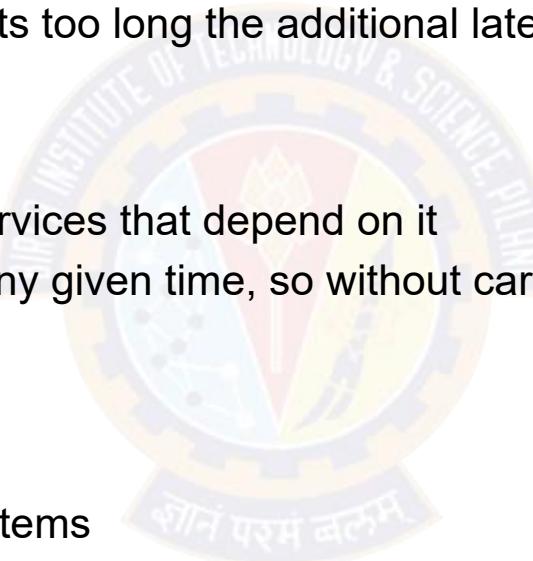
- Mix of technologies
 - Teams can pick the technology that best fits their service, using a mix of technology stacks as appropriate
- Fault isolation
 - If an individual microservice becomes unavailable, it won't disrupt the entire application, as long as any upstream microservices are designed to handle 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:

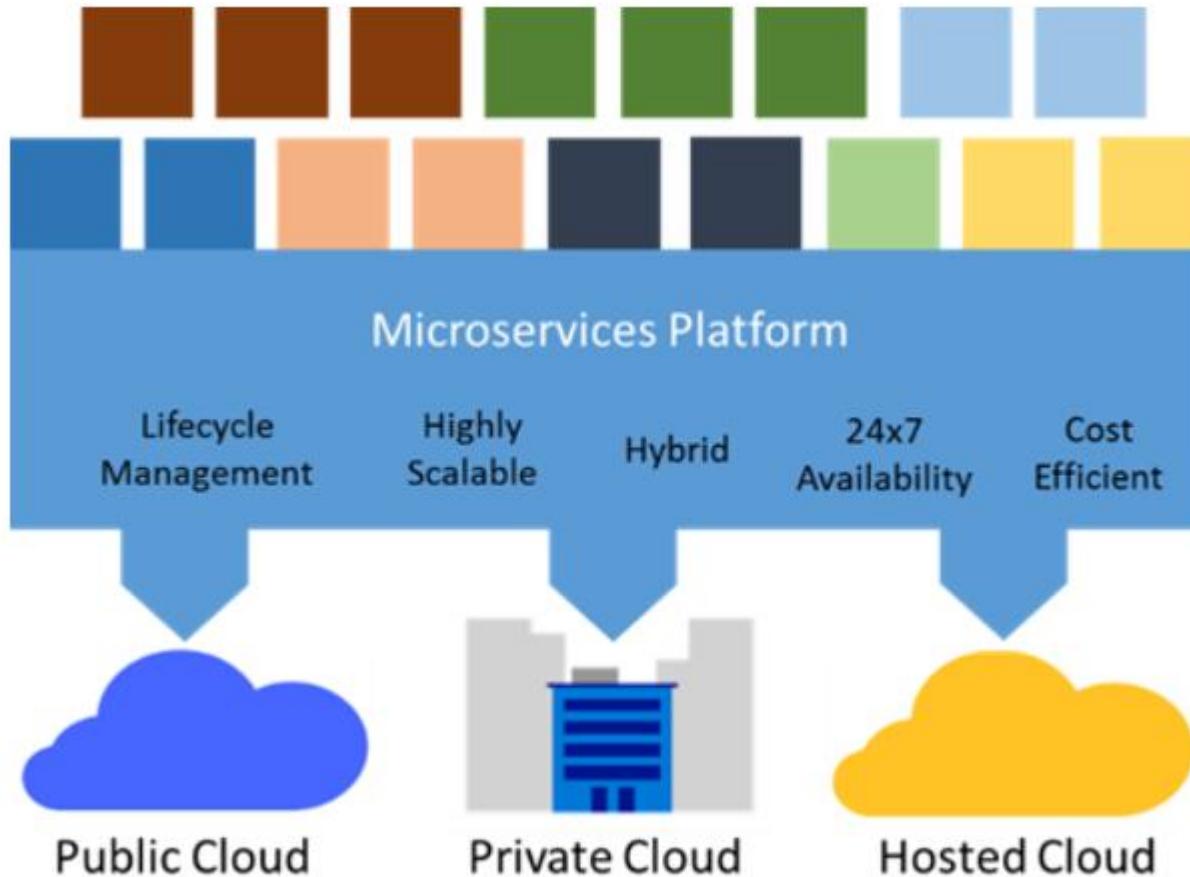


BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Microservice Application Platforms

Chandan Ravandur N

Microservice application platforms



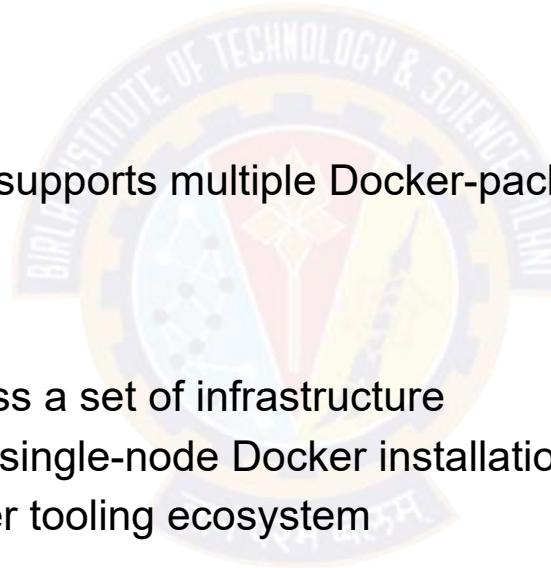
Microservices Platform

[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



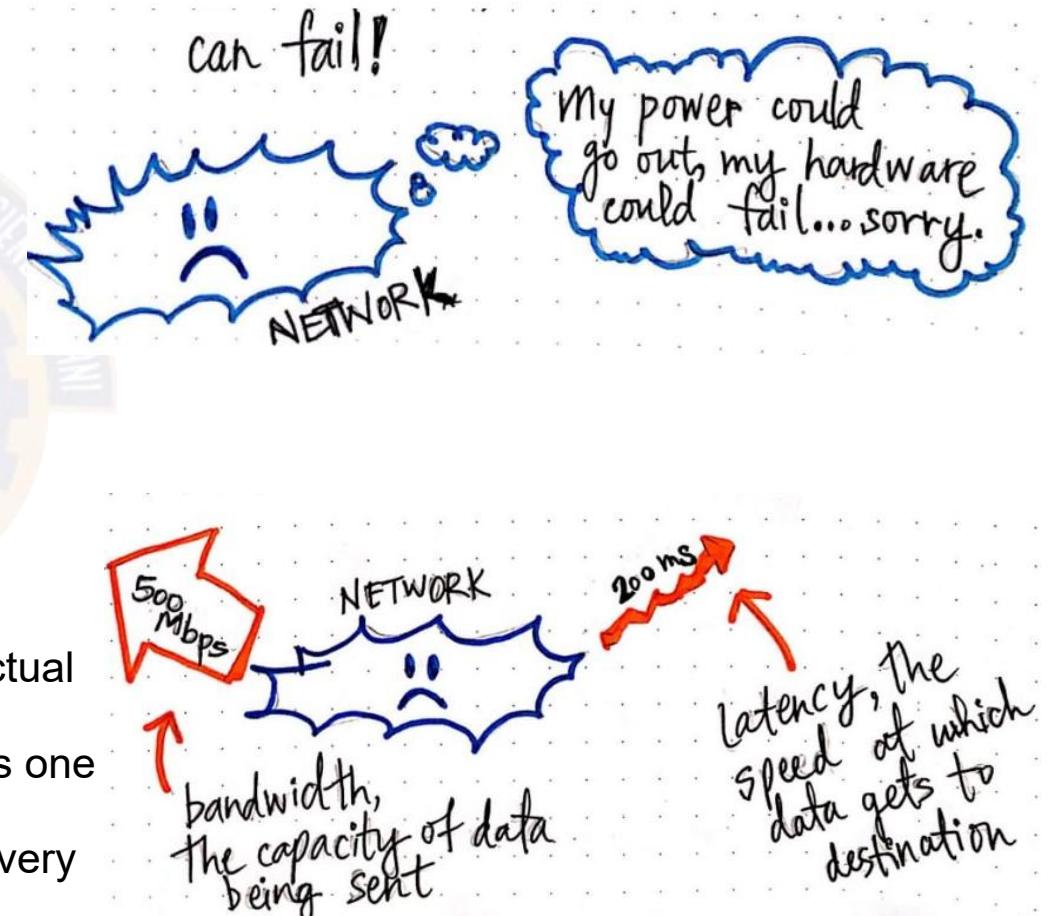
source

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.



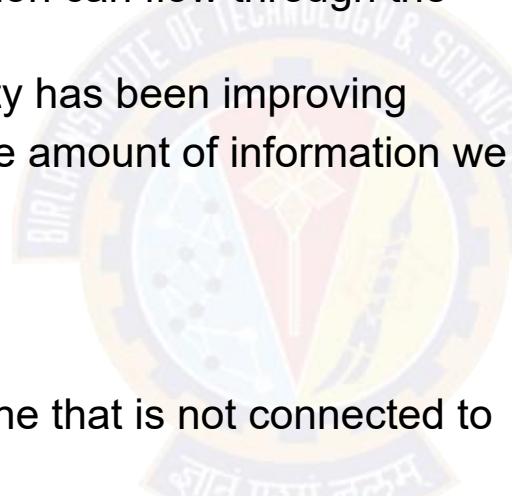
- Fallacy #2: Latency is zero
 - ✓ Latency is the time it takes between a request and the start of actual response data
 - ✓ Latency within development platforms networks can be as low as one millisecond
 - ✓ In production, when traffic is going over the internet, the story is very different
 - ✓ At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



Fallacies(2)

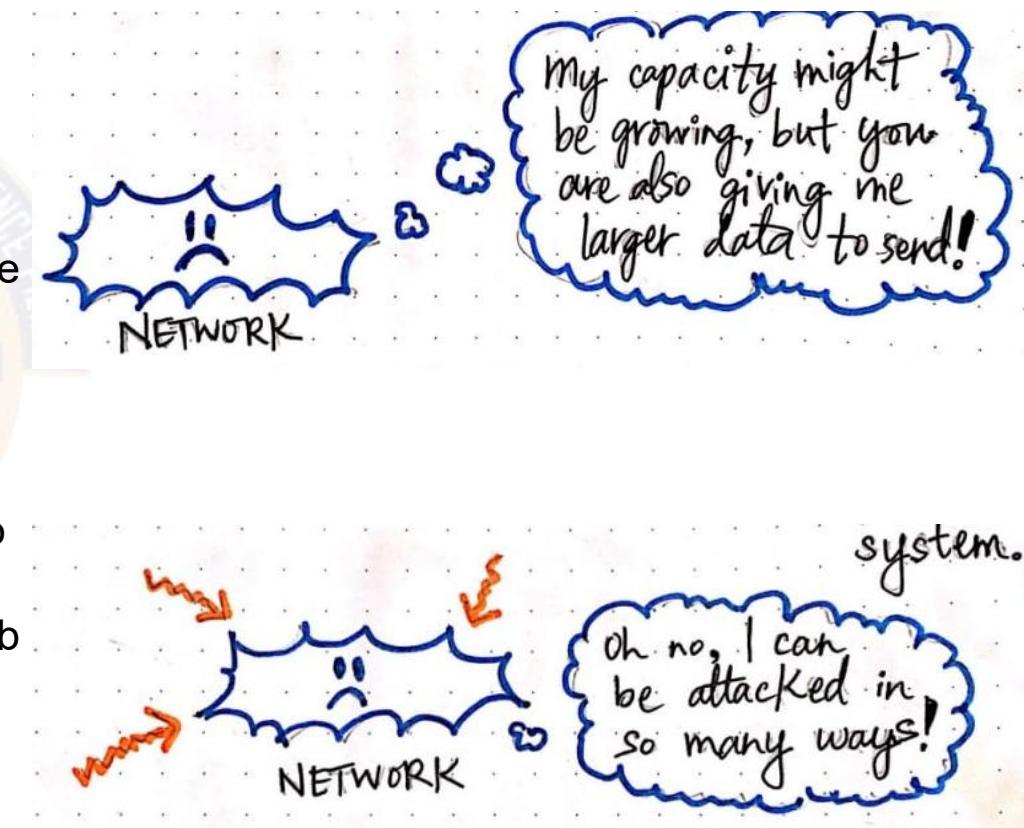
- Fallacy #3: Bandwidth is infinite

- ✓ 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

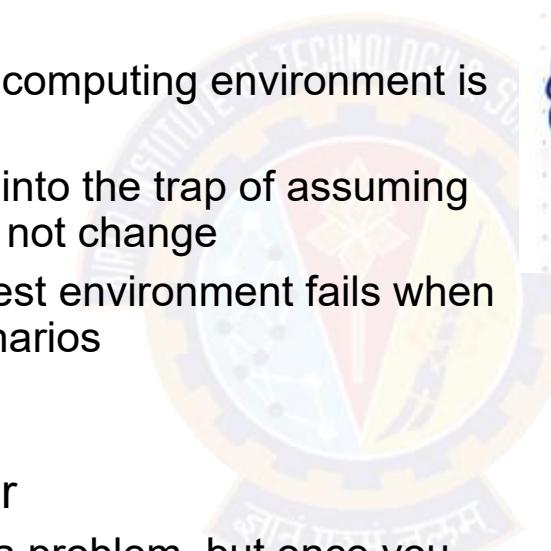
- ✓ The only completely secure system is one that is not connected to any network
- ✓ Keeping the network secure is a complex task and is a full-time job for many professionals
- ✓ There is a wide variety of both passive and active network attacks that can render network traffic unsafe from malicious users



Fallacies(3)

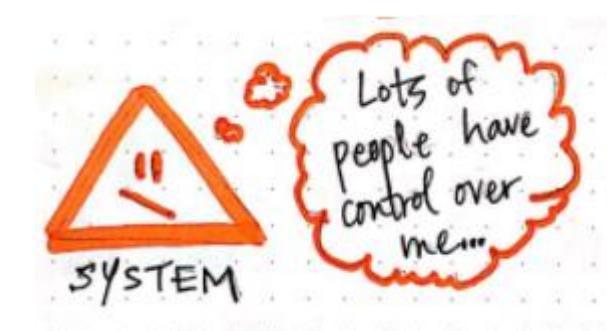
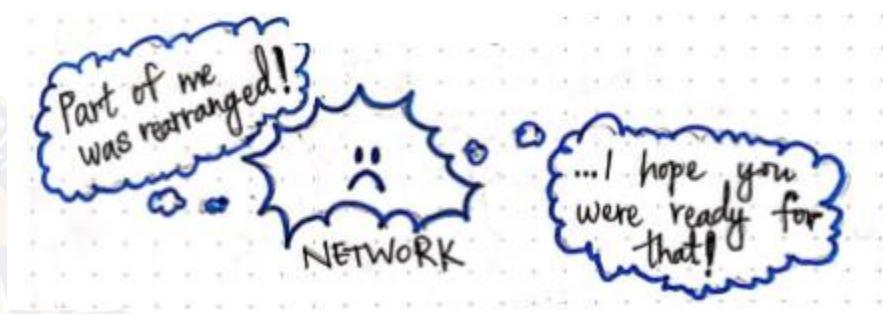
- Fallacy #5: Topology doesn't change

- ✓ 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



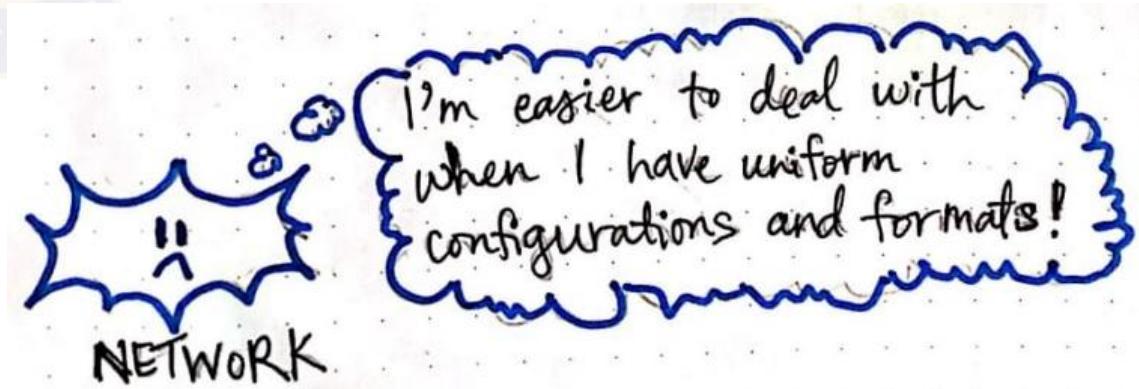
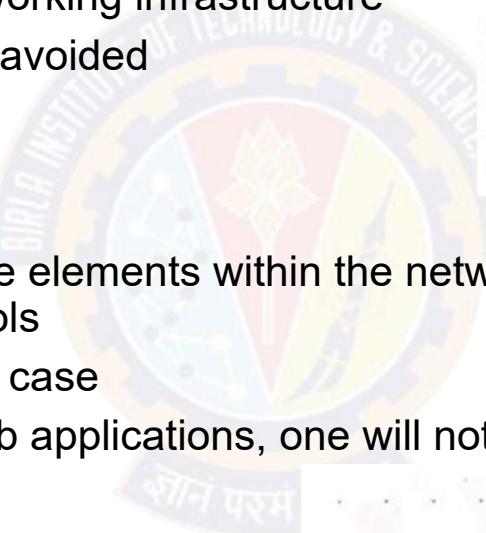
Fallacies(4)

- 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



SYSTEM

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

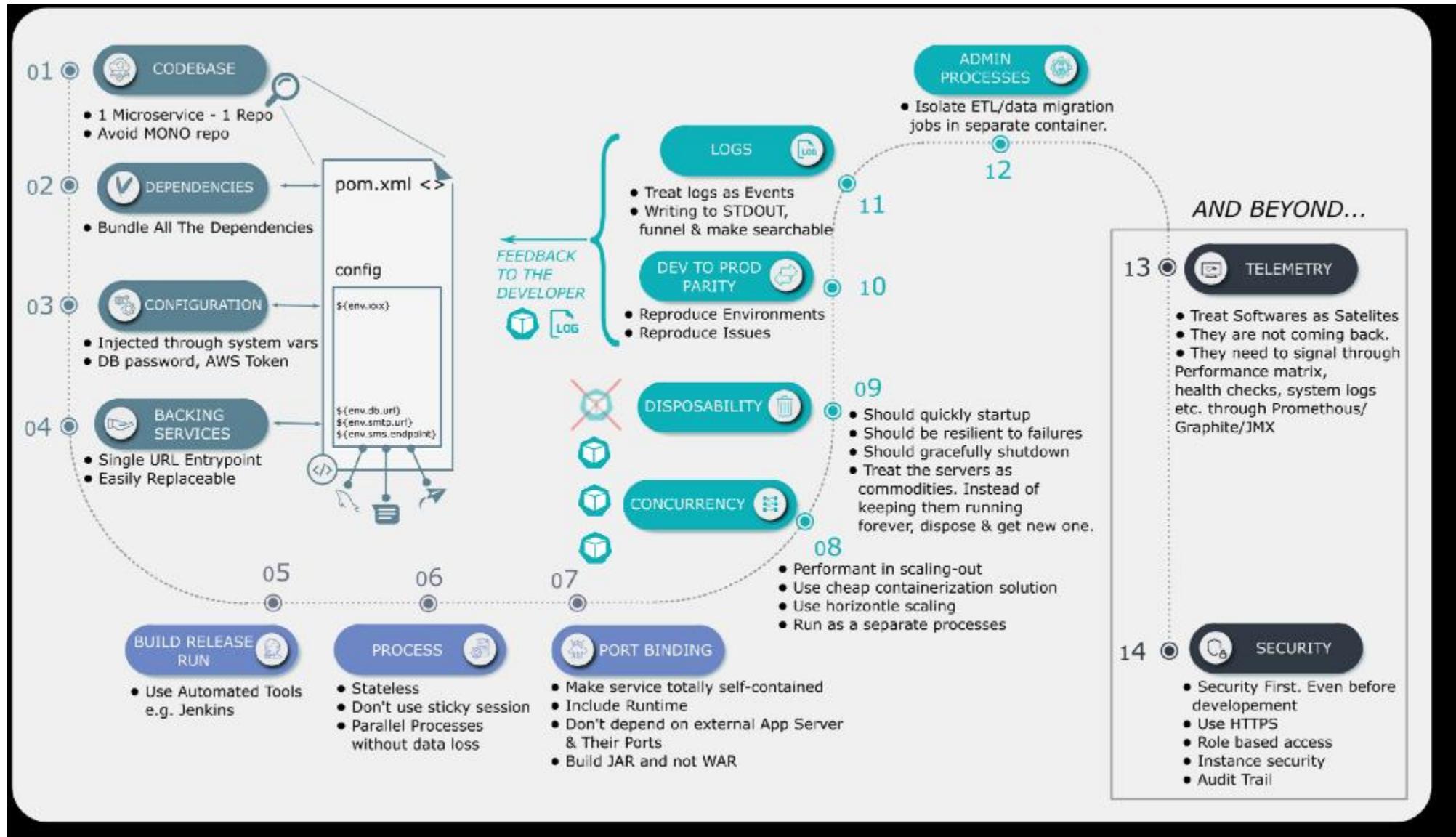
12 Factor App

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

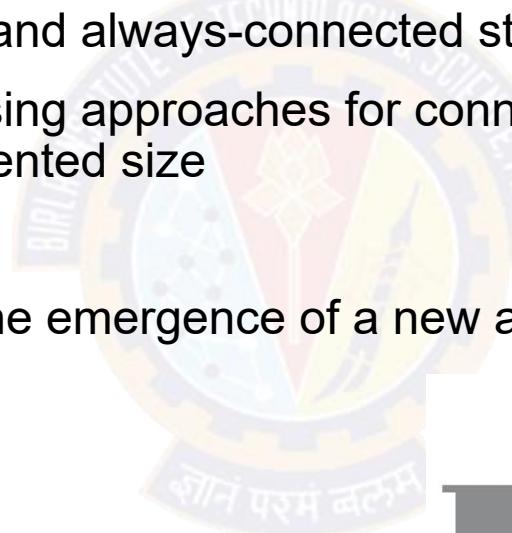
Cloud Native Architecture

Chandan Ravandur N

Introducing cloud-native software

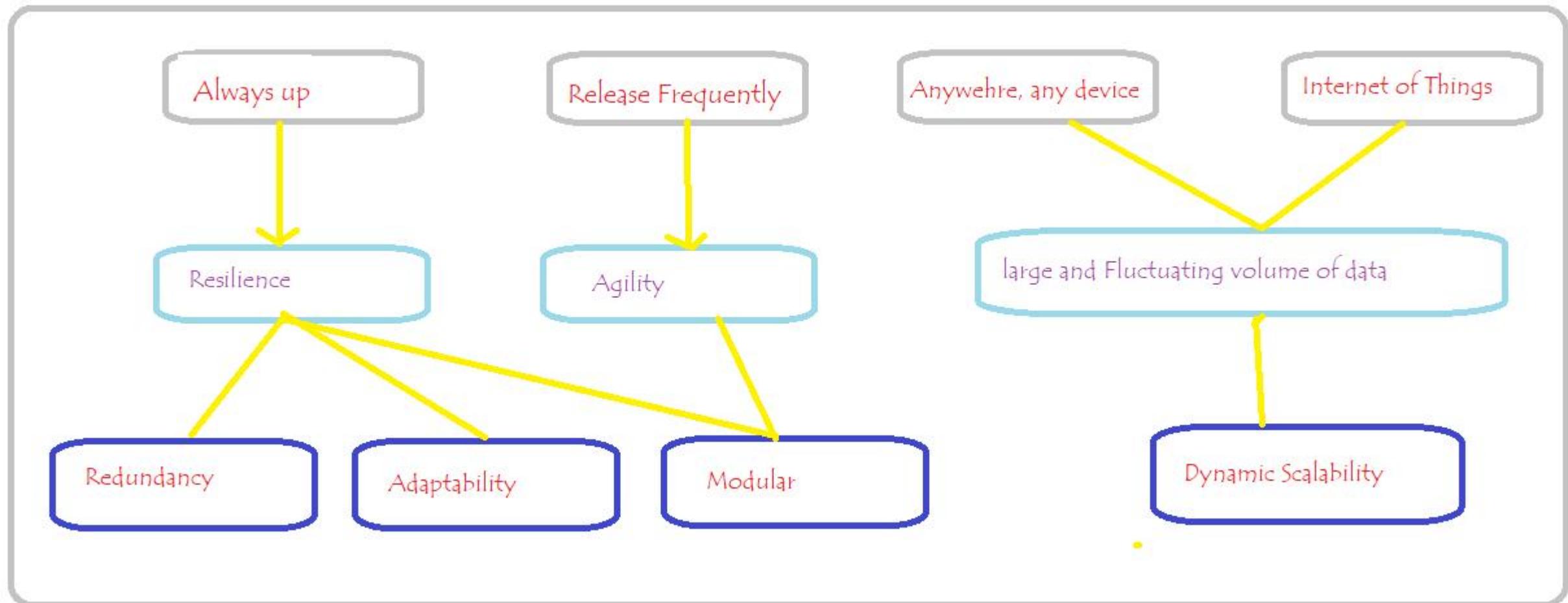
New Edge Applications Demands

- 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!**



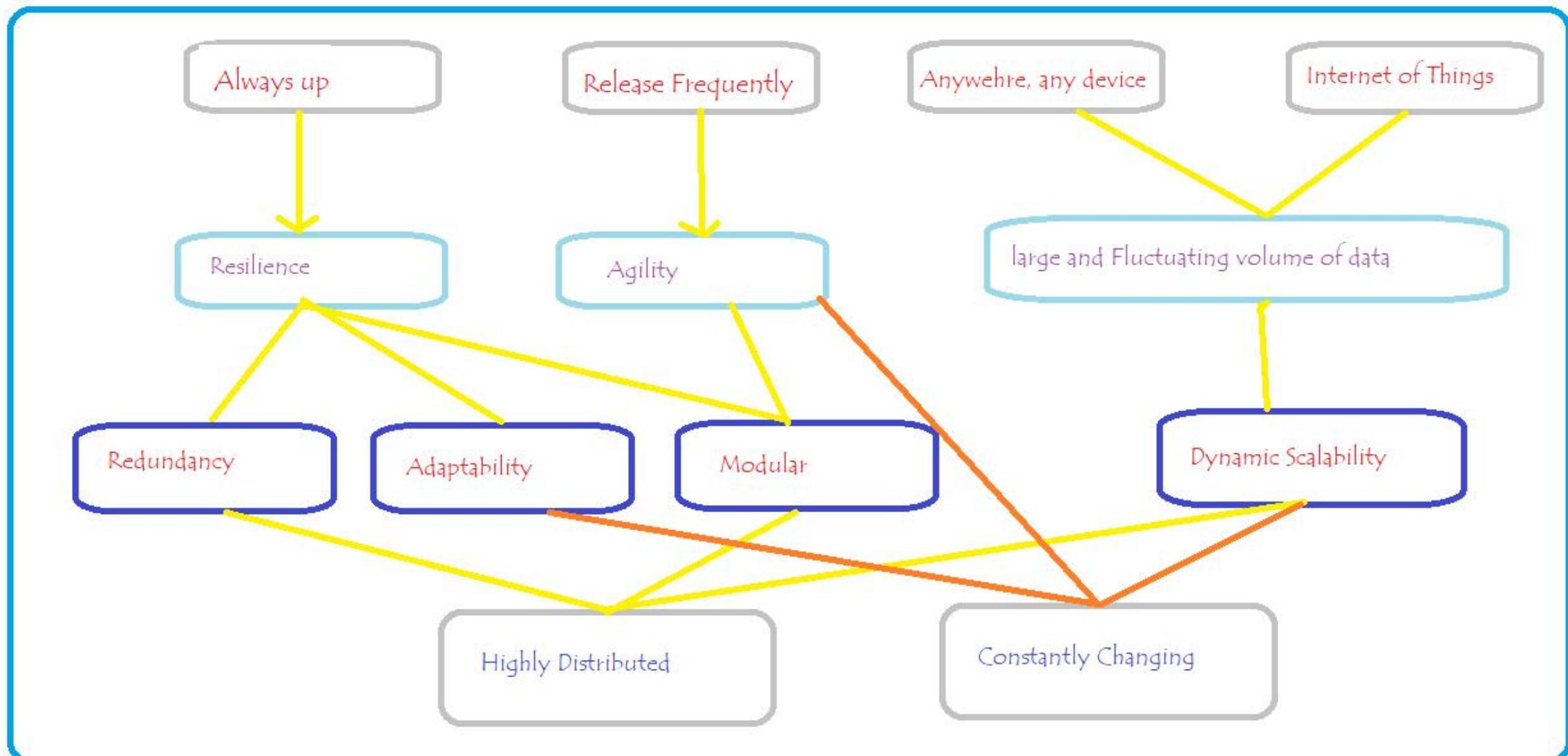
User requirements for software

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:



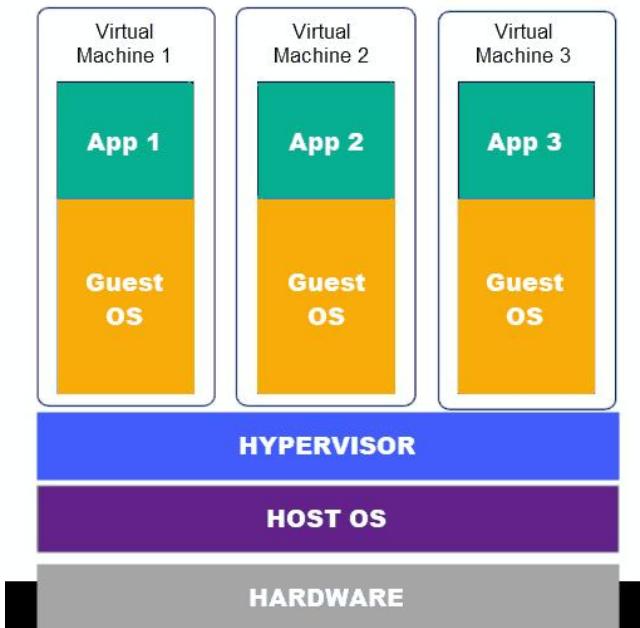
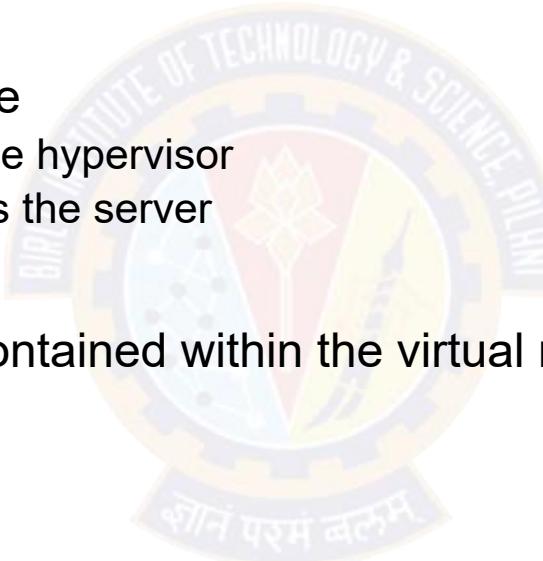
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Containers

Chandan Ravandur N

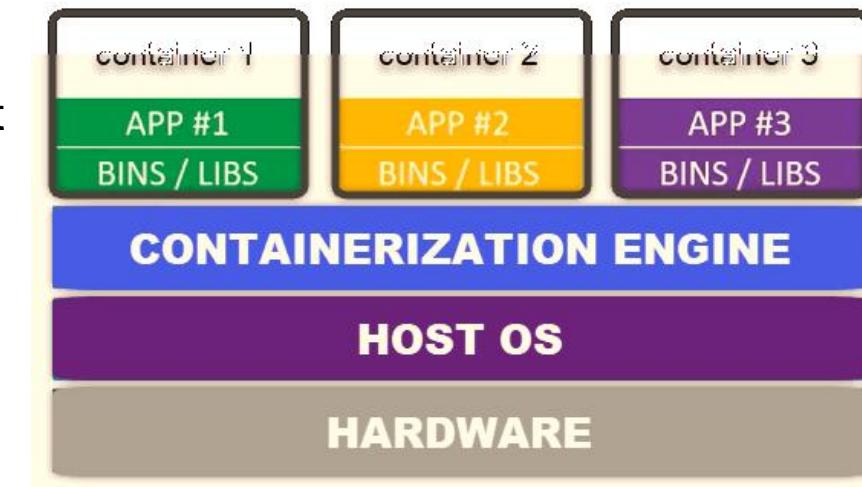
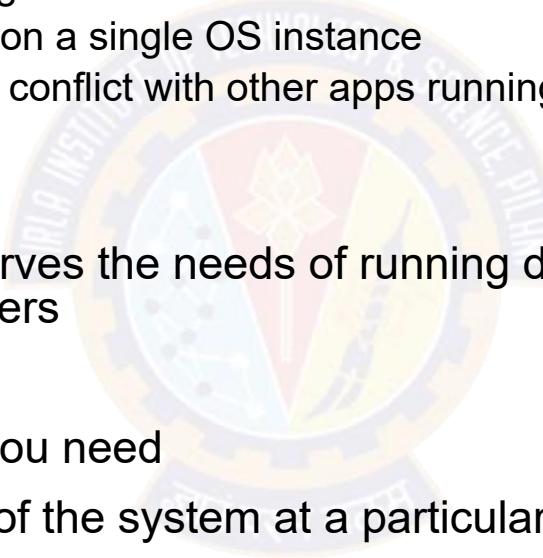
Virtual Machines

- 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



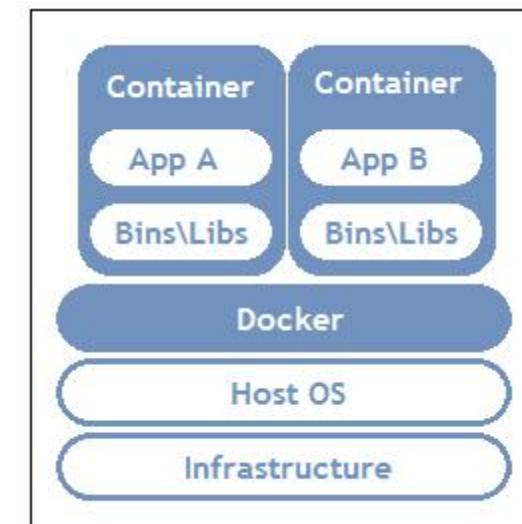
Containers

- 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



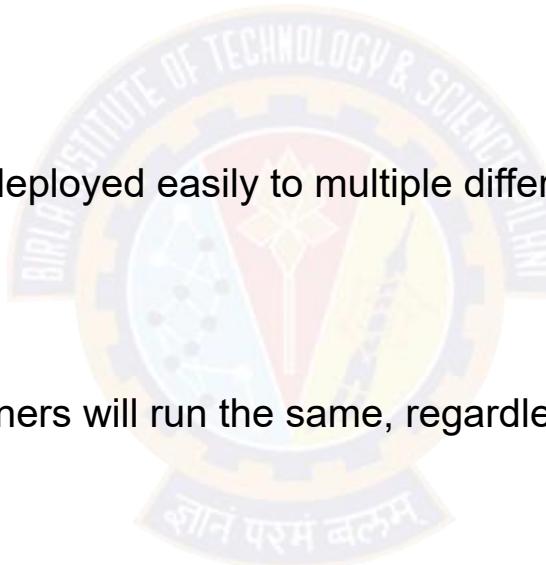
Containers(2)

- 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.

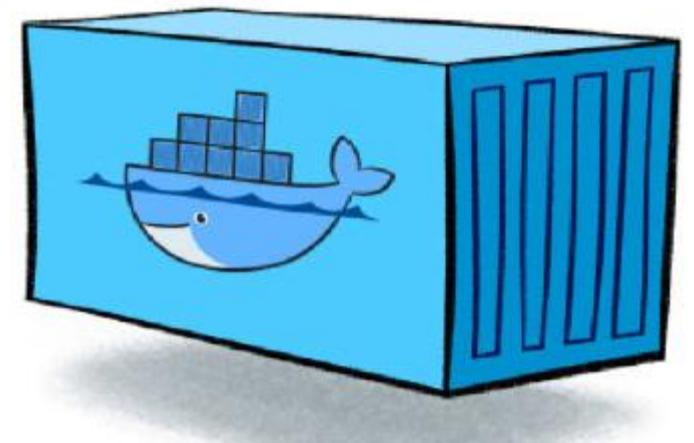
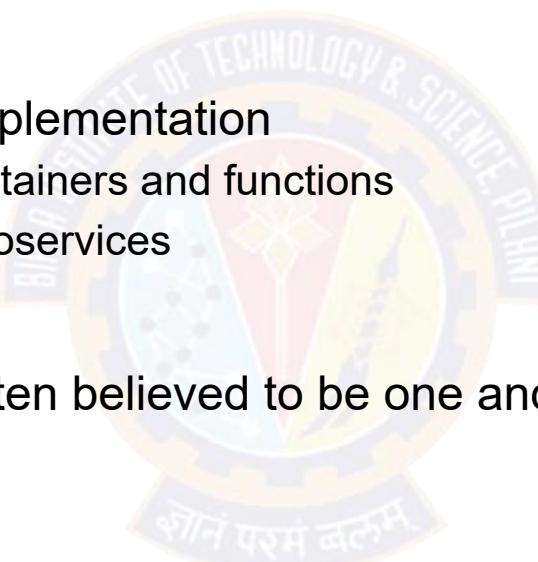


Container use cases

- “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

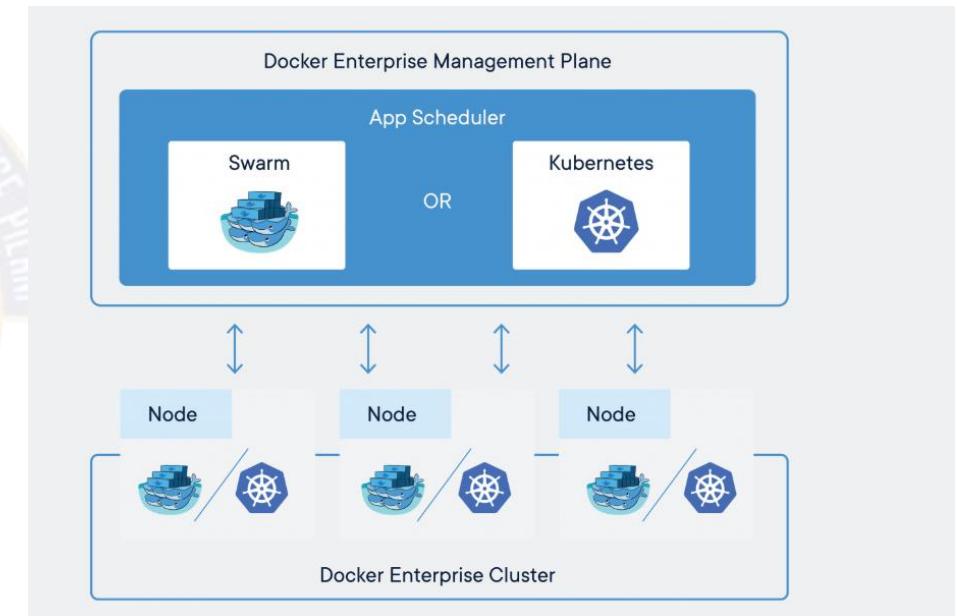
- Cloud native apps are distributed in nature and utilize cloud infrastructure
- Many technologies and tools used for implementation
 - ✓ From compute perspective – its only containers and functions
 - ✓ From architectural perspective – its microservices
- These terms are mistakenly used and often believed 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

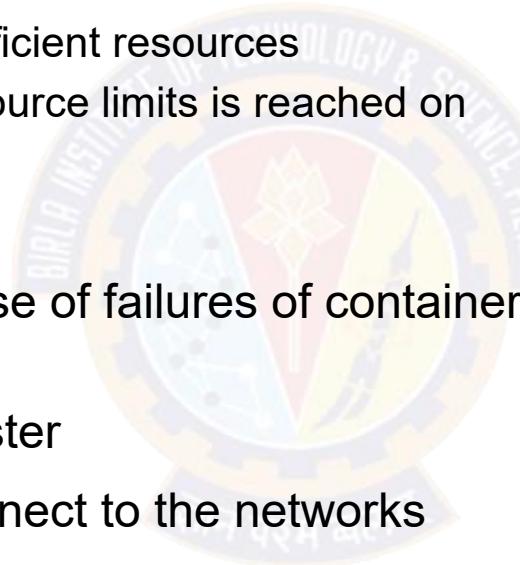


Source : medium

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:



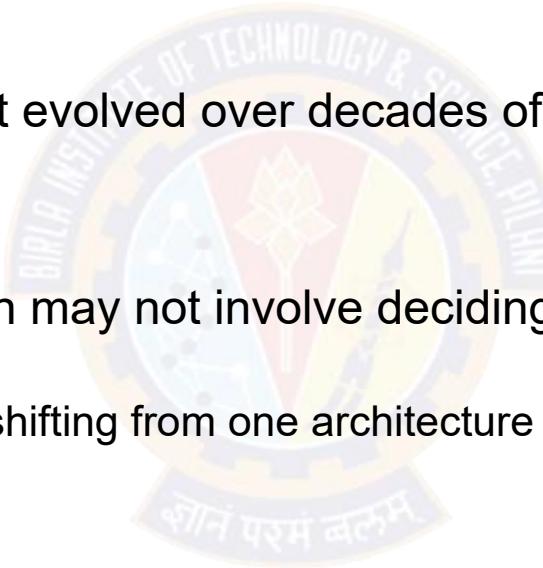
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Architecture deployment approaches - I

Chandan Ravandur N

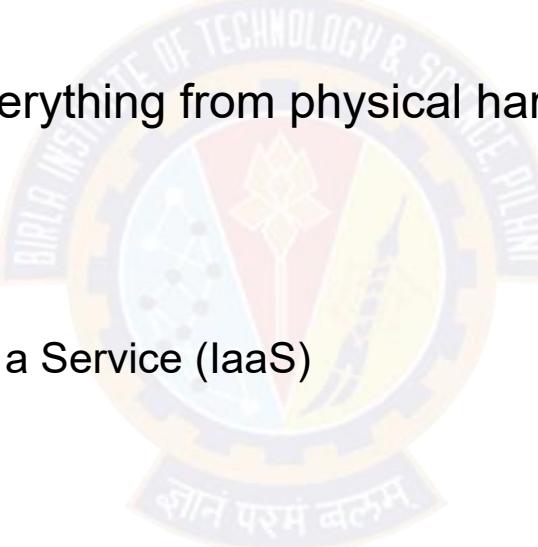
Architecture approaches

- 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

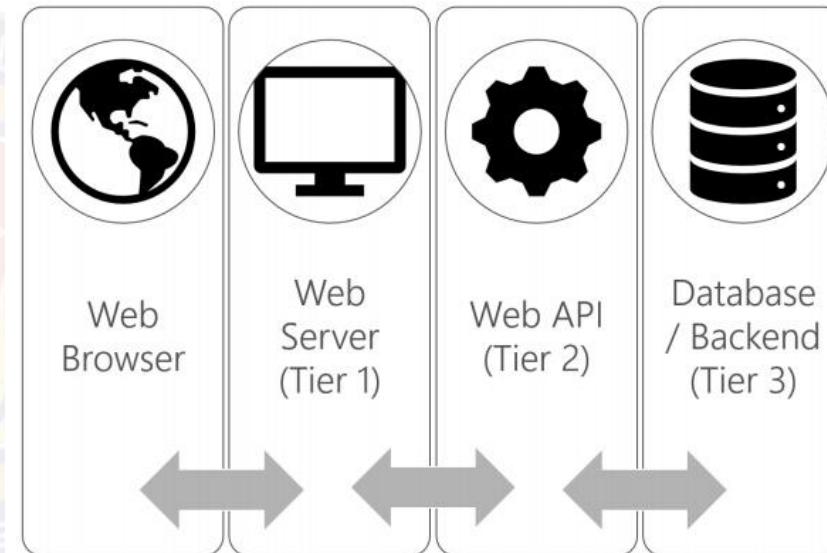
- Regardless of the architecture approach used to design a business application,
- the implementation, or deployment of those applications may vary
- Businesses host applications on everything from physical hardware to Serverless functions
- Conventional deployment patterns
 - ✓ N-Tier applications
 - ✓ 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

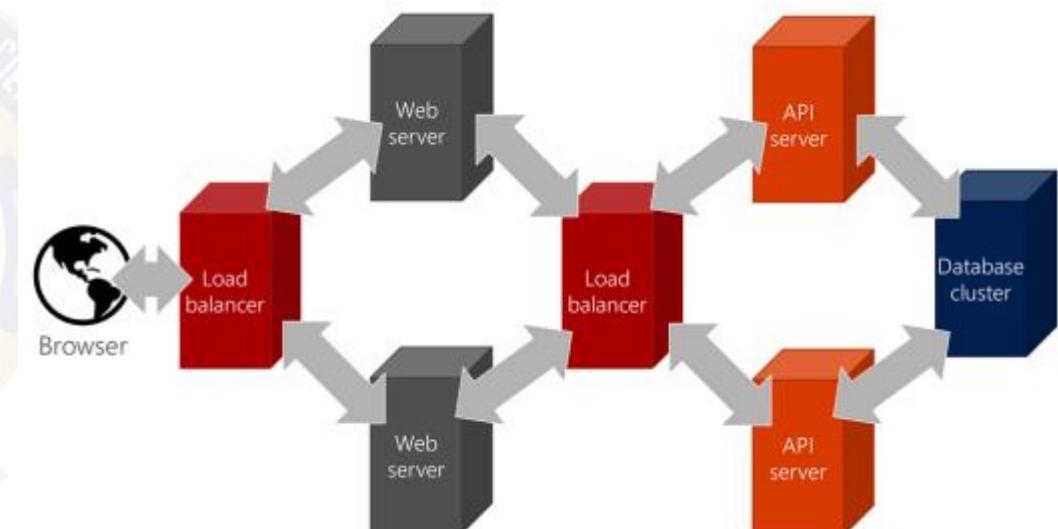
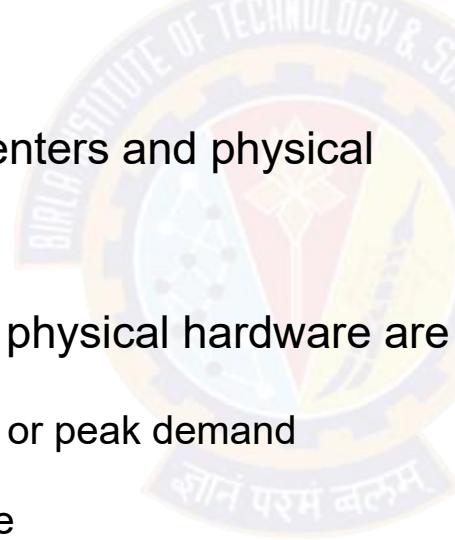


source : Microsoft

On-premises

On-premise hosting

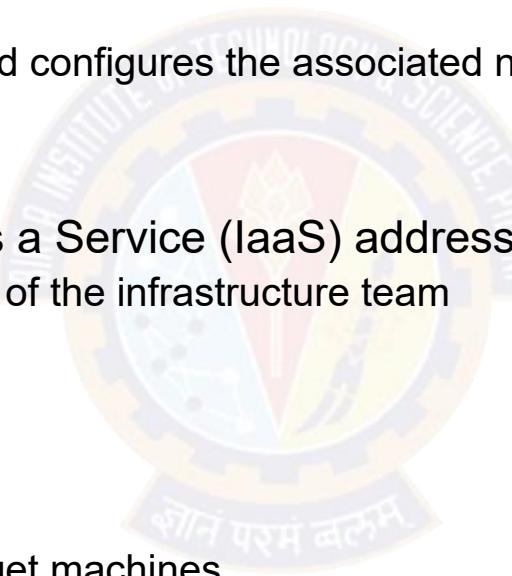
- The traditional approach to hosting applications requires
 - ✓ buying hardware
 - ✓ managing all of the software installations, including the operating system
- Originally this involved expensive data centers and physical hardware
- The challenges that come with operating physical hardware are many, 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)

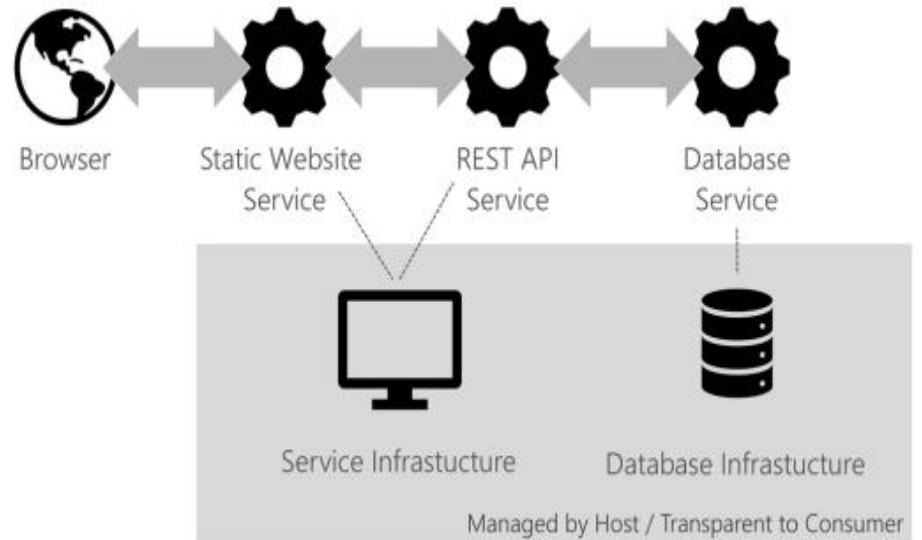
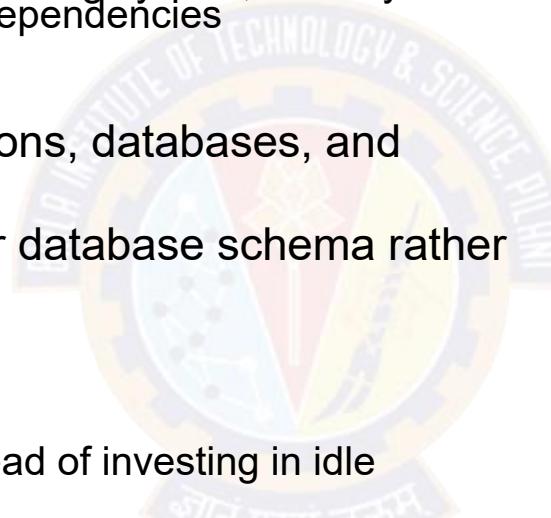
- Virtualization of hardware, via “virtual machines” enables Infrastructure as a Service (IaaS)
 - ✓ Host machines are effectively partitioned to provide resources to instances with allocations for their own memory, CPU, and storage
 - ✓ The team provisions the necessary VMs and configures the associated 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)

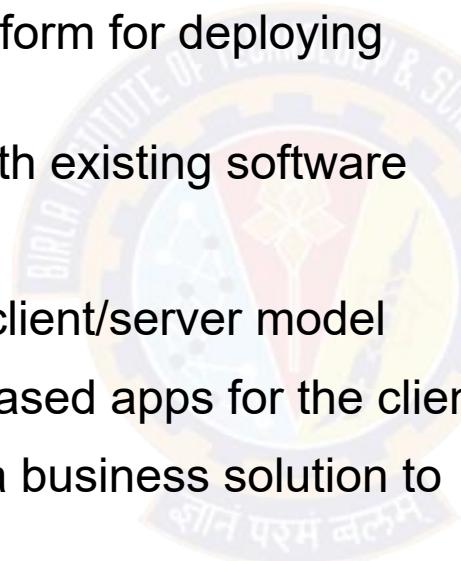
- Offers configured solutions that developers can plug into directly
 - ✓ Another term for managed hosting
 - ✓ Eliminates the need to manage the base operating system, security patches and in many cases any third-party dependencies
- Examples of platforms include web applications, databases, and mobile 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:



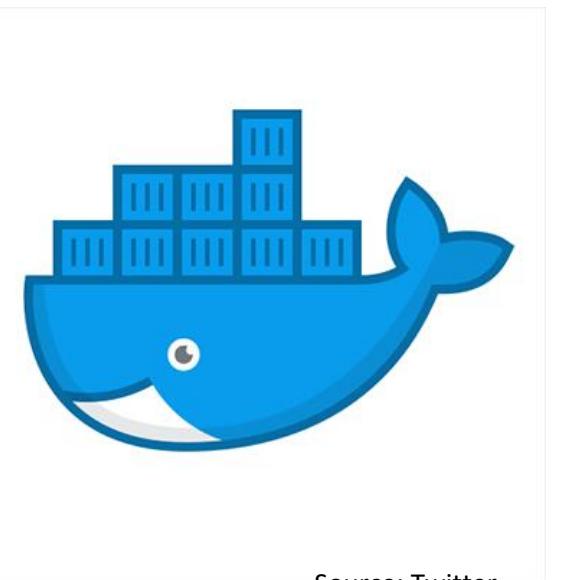
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Architecture deployment approaches - II

Chandan Ravandur N

Containers

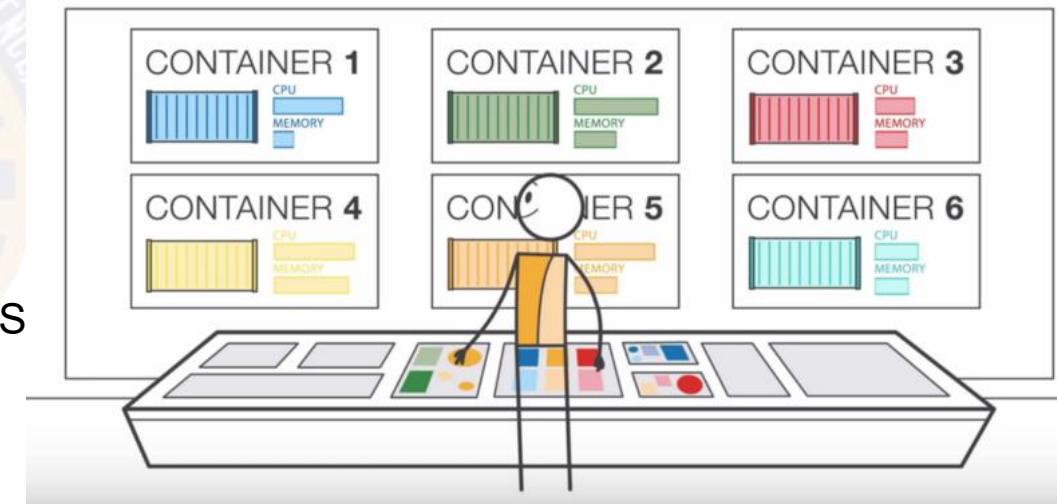
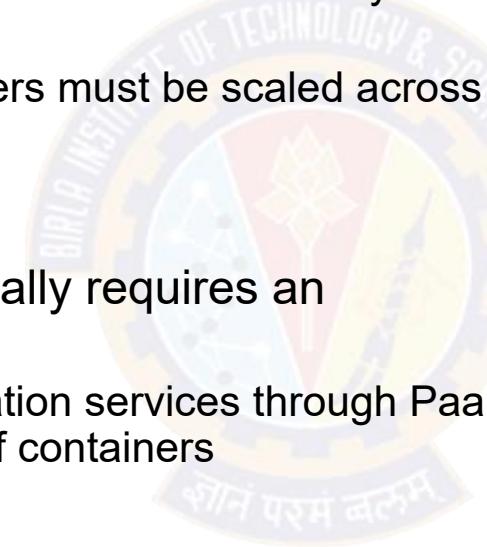
- Containers are an interesting solution that enables PaaS-like benefits without the IaaS overhead
- A container is essentially a runtime that contains the bare essentials needed to run a unique application
- The kernel or core part of the host operating system and services such as storage are shared 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



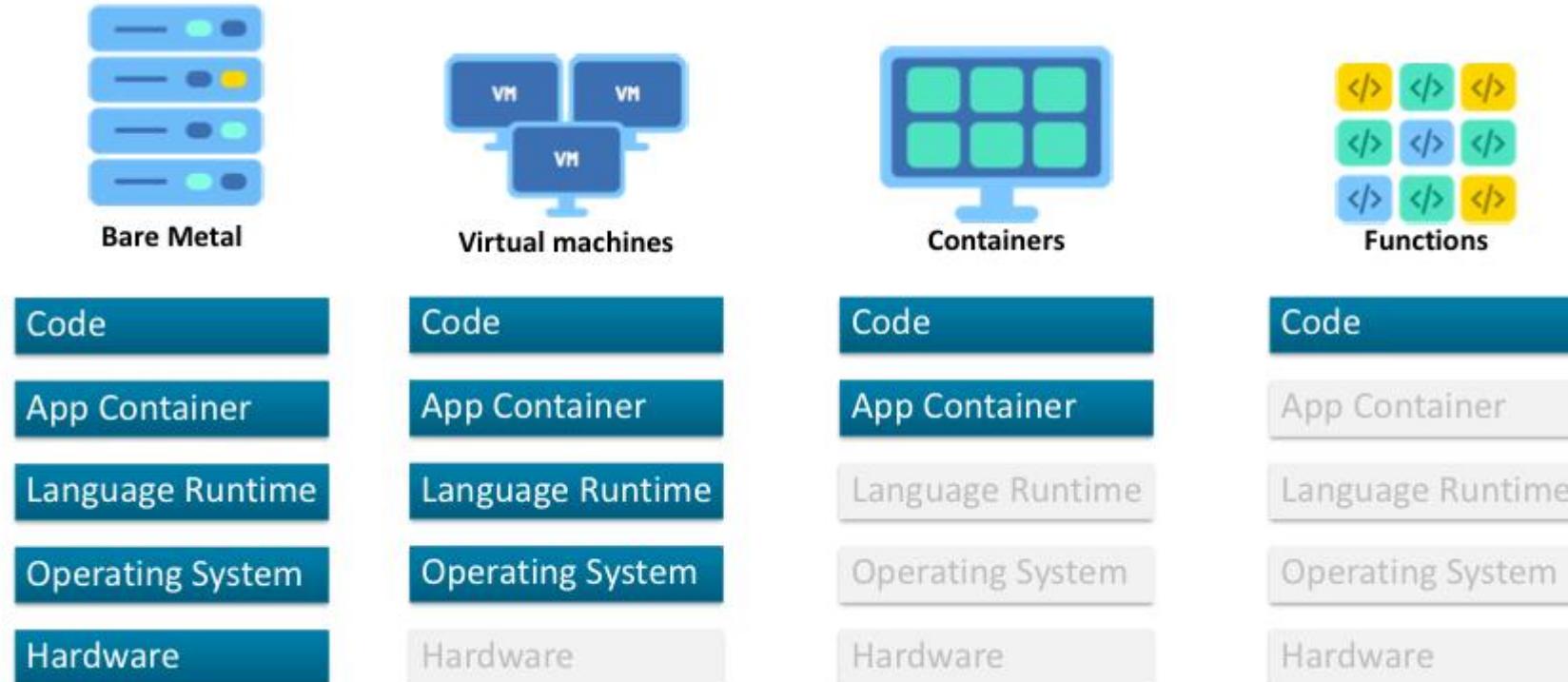
source:freecodecamp

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:



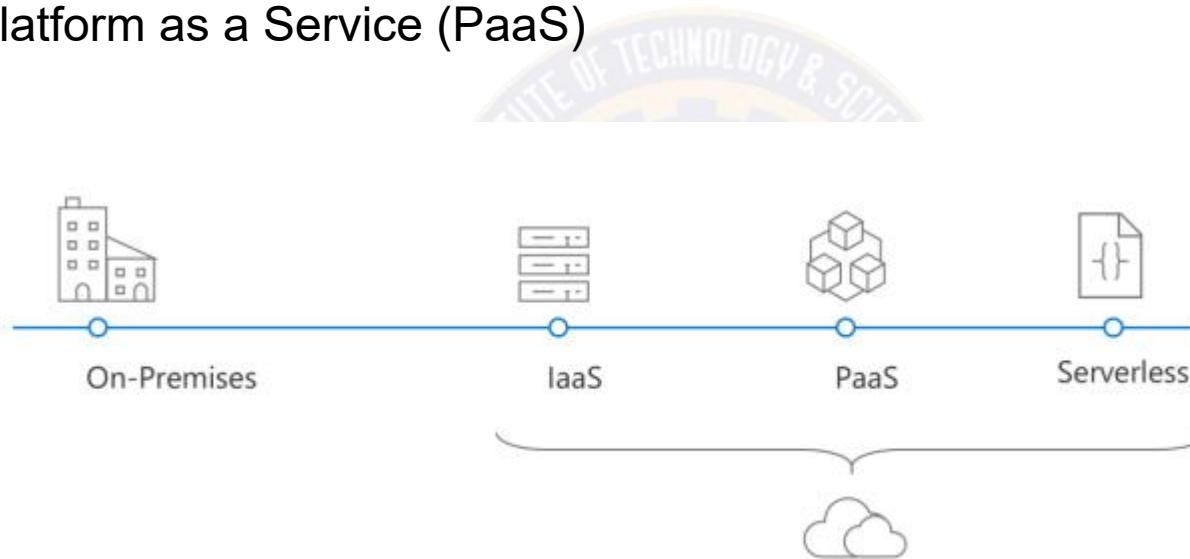
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

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

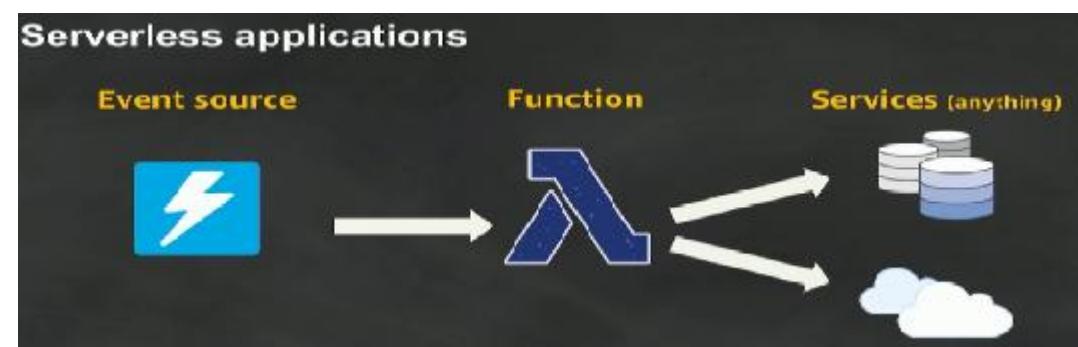
Serverless

- 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

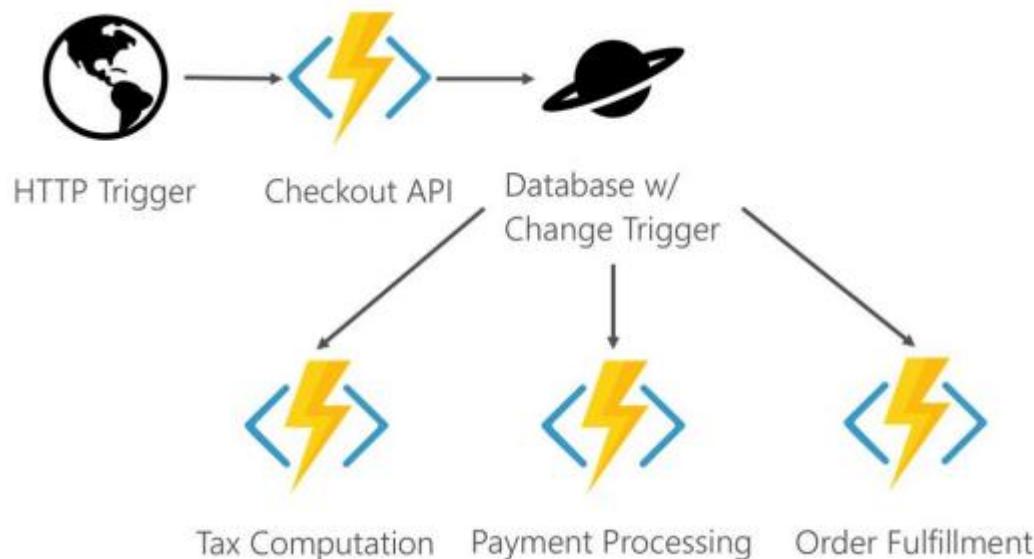
source : AWS



- It can be thought of as less server!

Serverless components

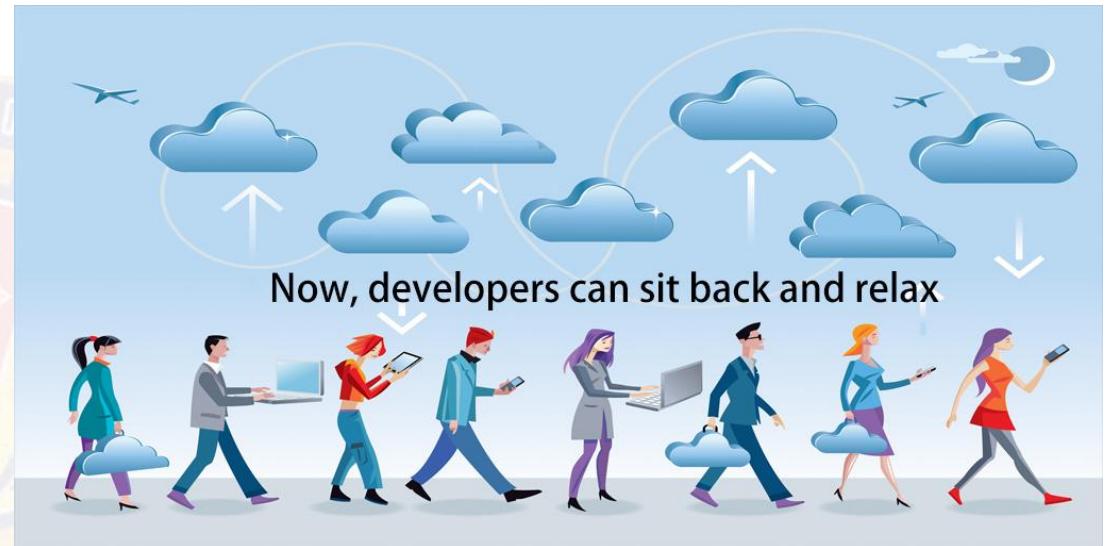
- An HTTP request causes the Checkout API code to run
- The Checkout API inserts code into a database
- The insert triggers several other functions to run to perform tasks like computing tasks and fulfilling the order.



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!

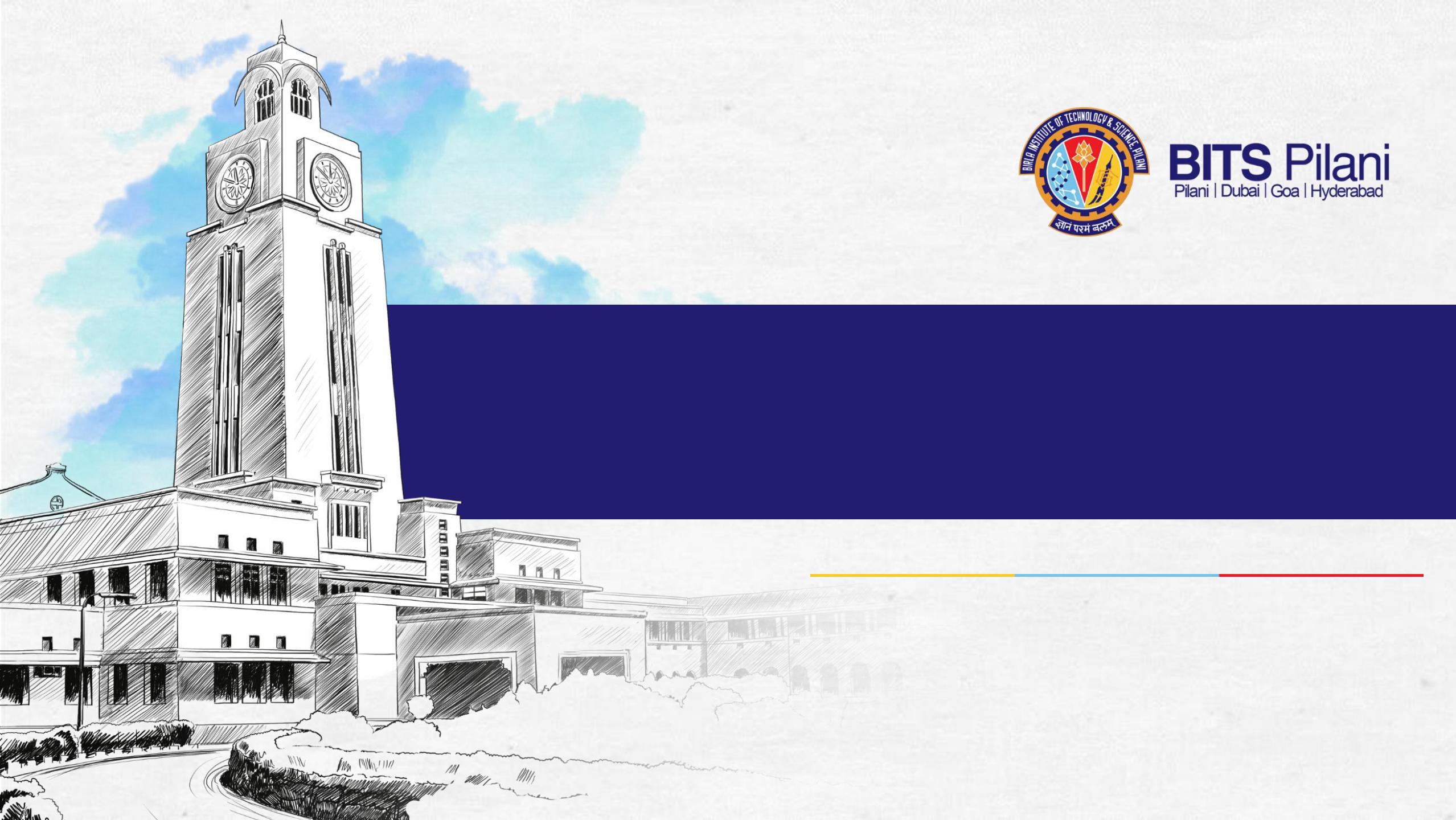
In our next session:



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Serveless App Examples

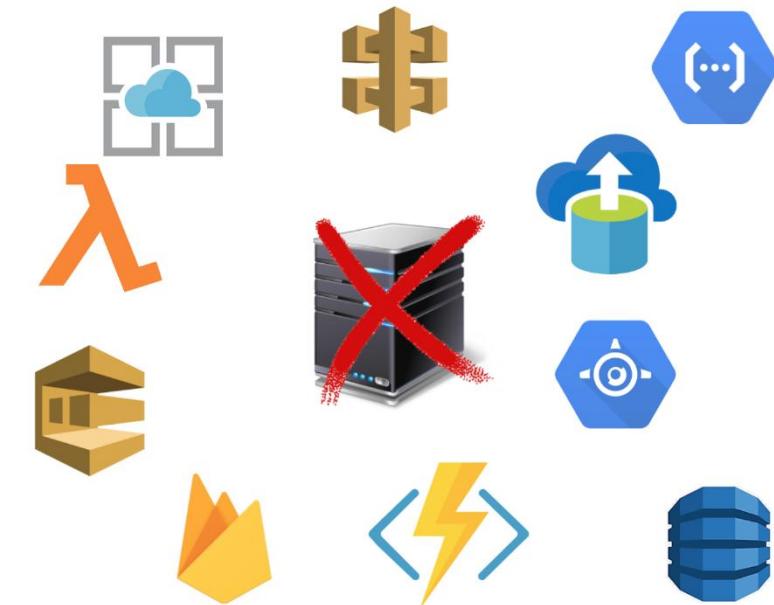
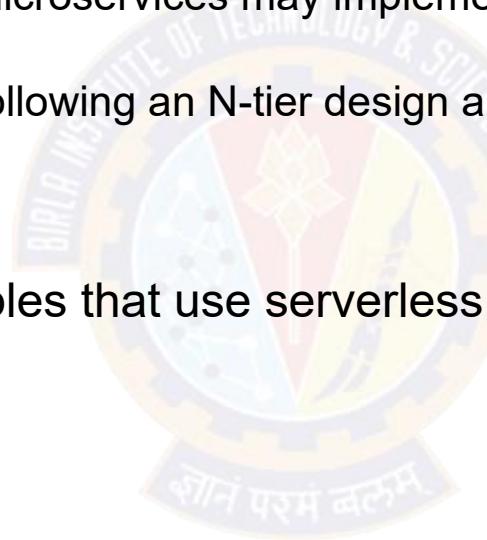
Chandan Ravandur N



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Common architecture examples for Serverless

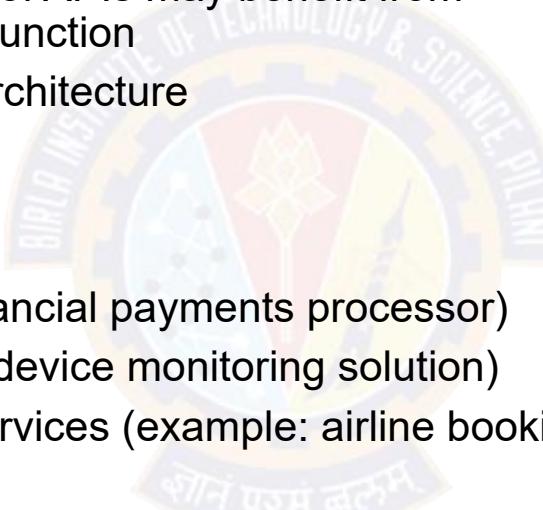
- Many approaches to using serverless architectures
 - ✓ Some projects may benefit from taking an “all-in” approach to serverless
 - ✓ Applications that rely heavily on microservices may implement all microservices using serverless technology
 - ✓ The majority of apps are hybrid, following an N-tier 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

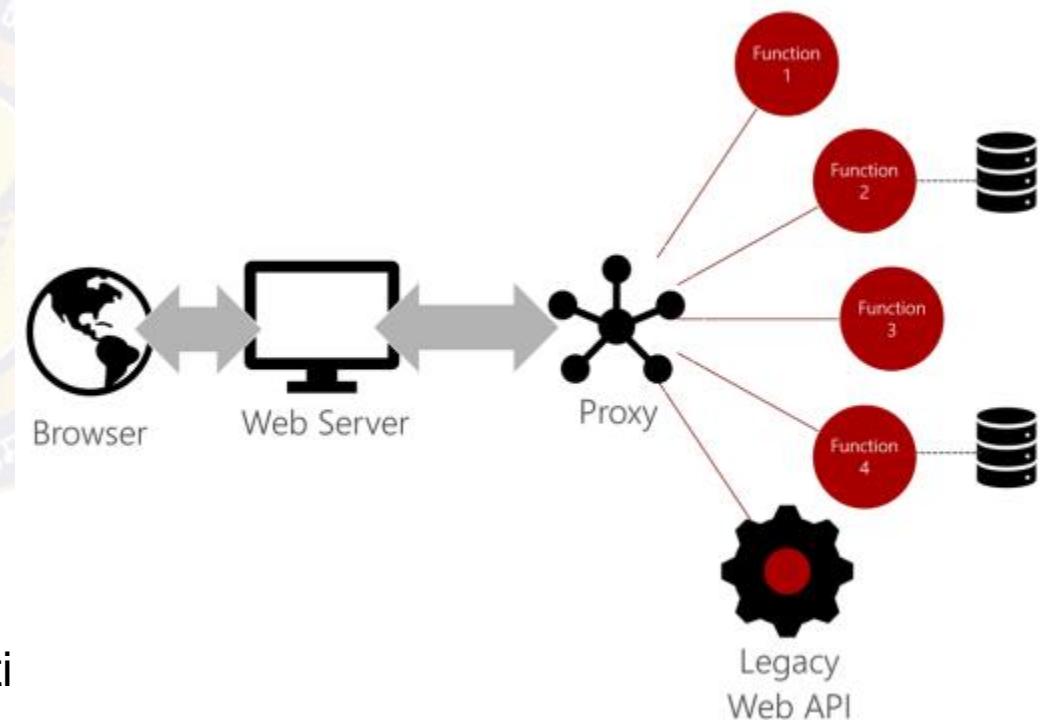
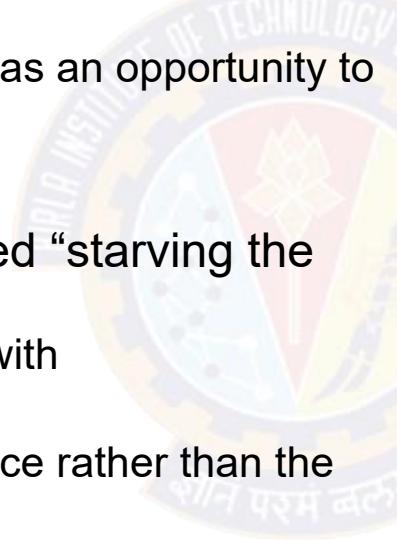
Full Serverless back end

- 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”

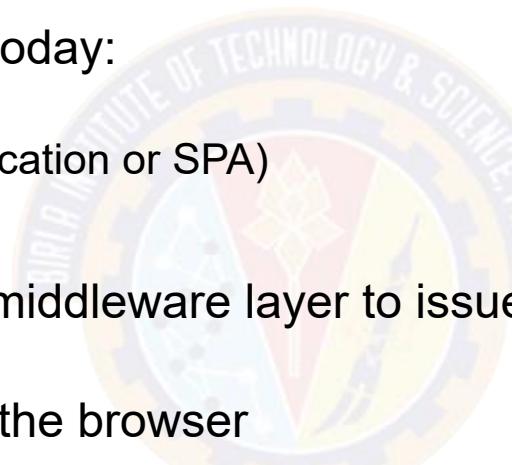
- A common challenge is migrating an existing monolithic application to the cloud
 - ✓ The least risky approach is to “lift and shift” entirely onto virtual machines
 - ✓ Many shops prefer to use the migration as an opportunity to modernize their code base
- A practical approach to migration is called “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



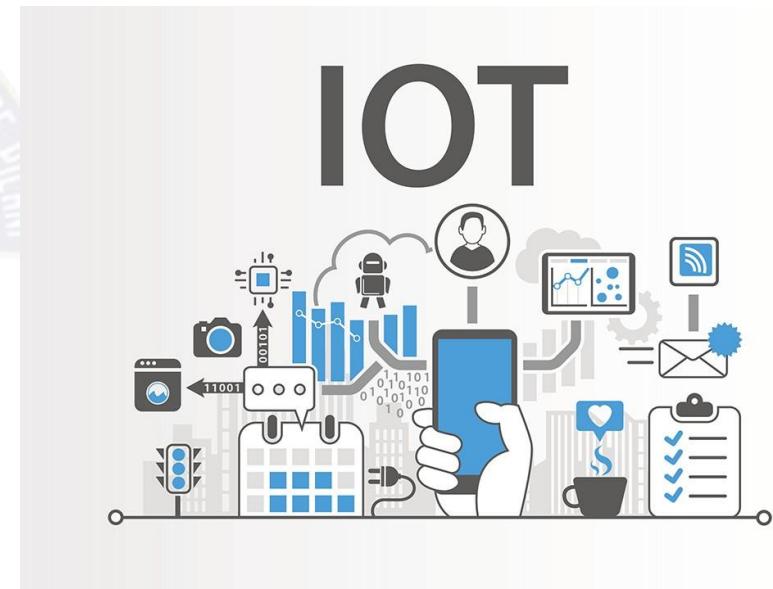
source : peerbits

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

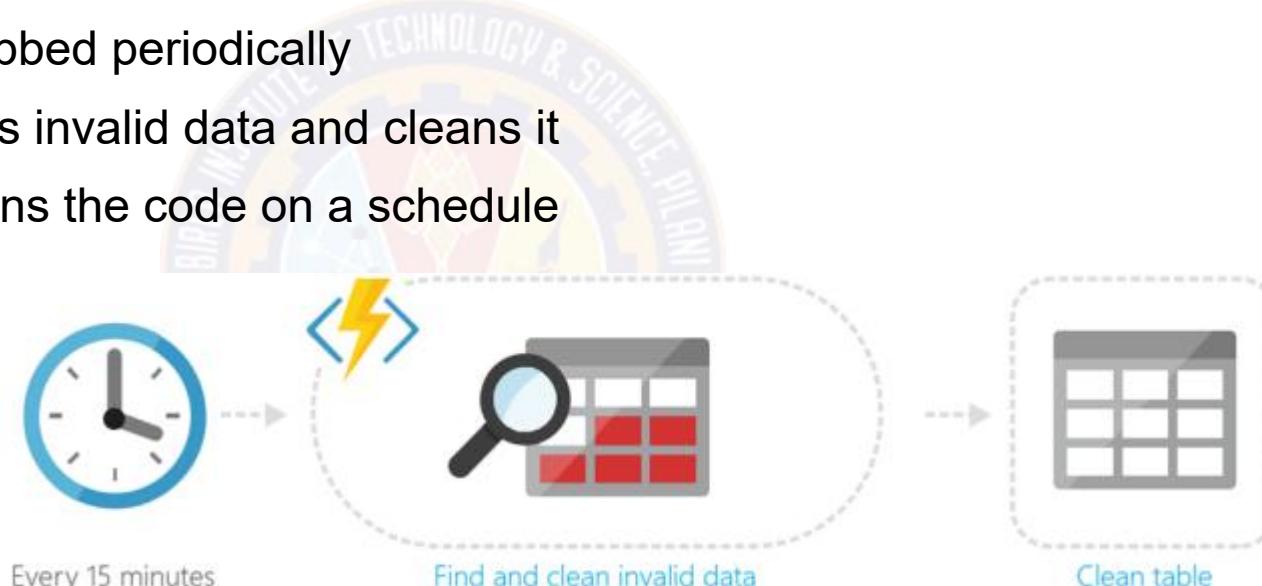
- Many design patterns exists for Serverless
- Commonality is the fundamental combination of an event trigger and business logic
- Patterns
 - ✓ Scheduling
 - ✓ Command and Query Responsibility Segregation (CQRS)
 - ✓ Event-based processing
 - ✓ File triggers and transformations
 - ✓ Web apps and APIs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

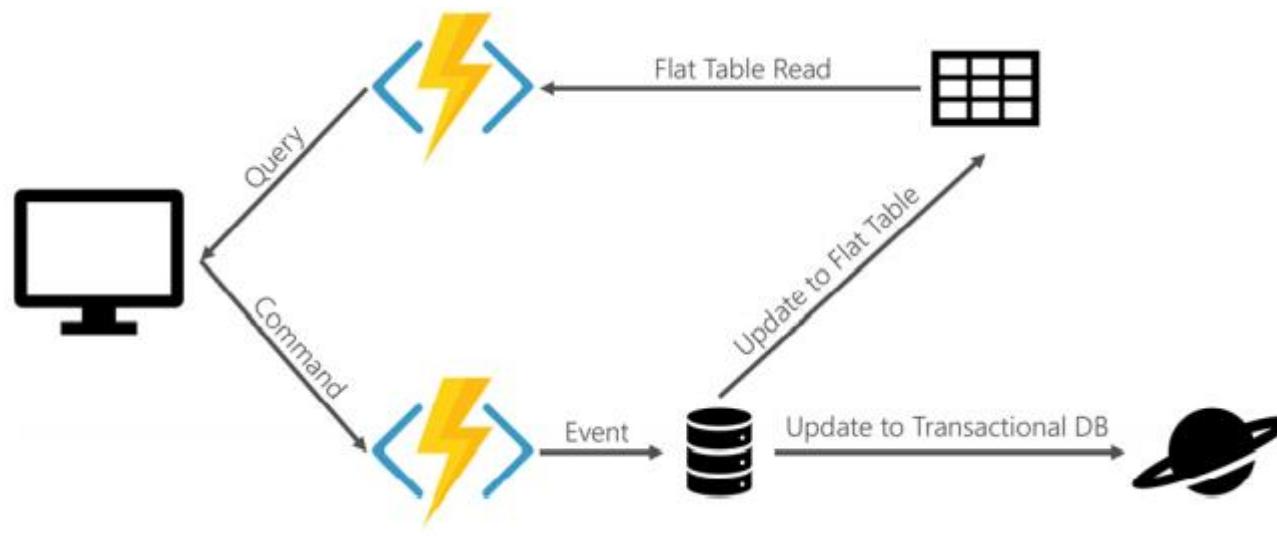


Source : Microsoft

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



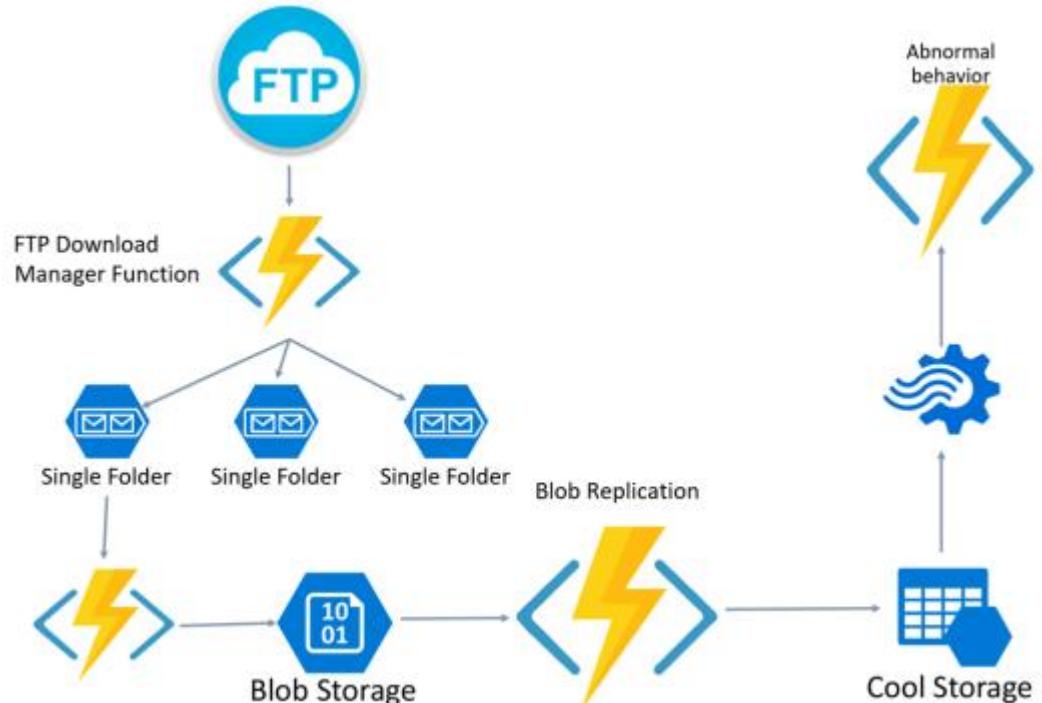
Source : Microsoft

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

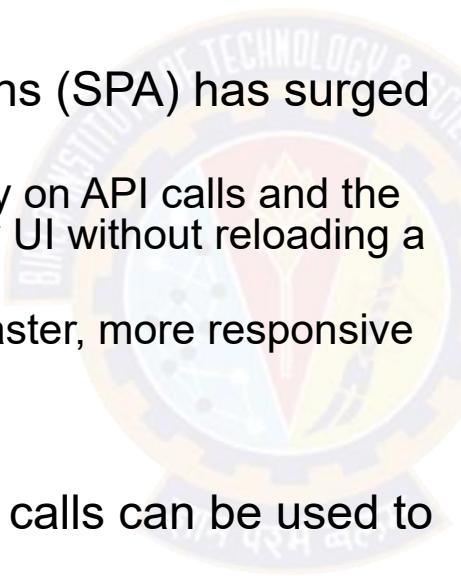
- Extract, Transform, and Load (ETL) is a common business function
- Serverless is a great solution for ETL because it allows code to be triggered as part of a pipeline
- Individual code components can address various aspects
 - ✓ One Serverless function may download the file
 - ✓ Another applies the 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

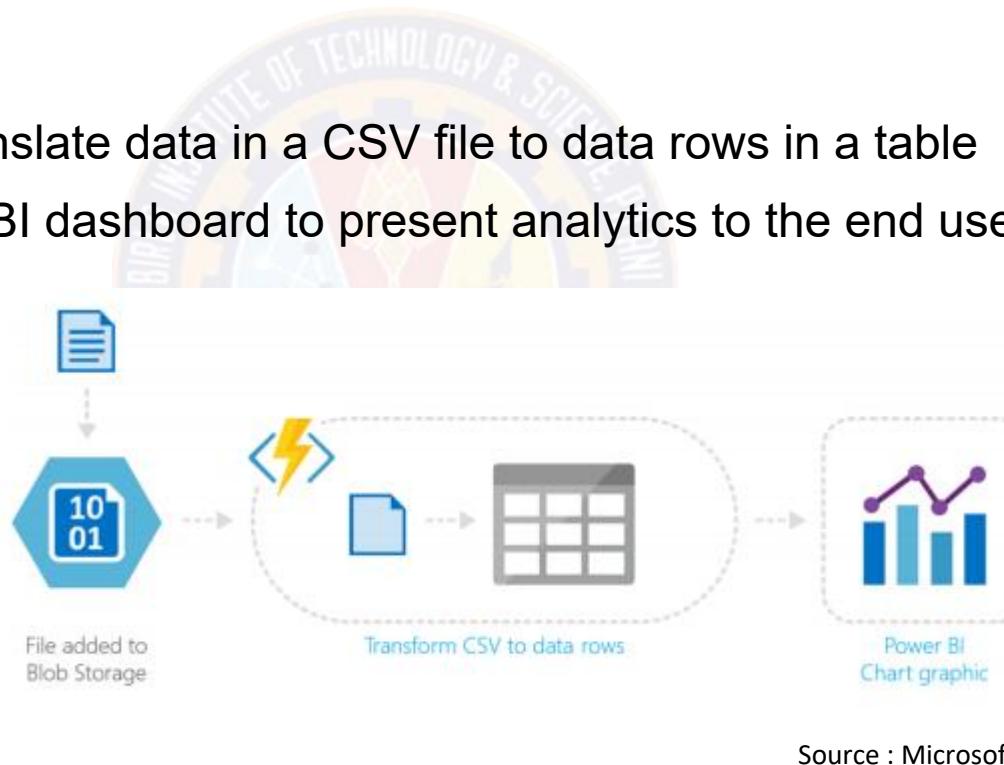
- A popular scenario for serverless is N-tier applications
 - ✓ most commonly ones where the UI layer is a web app
- The popularity of Single Page Applications (SPA) has surged recently
 - ✓ SPA apps render a single page, then rely on API calls and the returned data to dynamically render new UI without 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

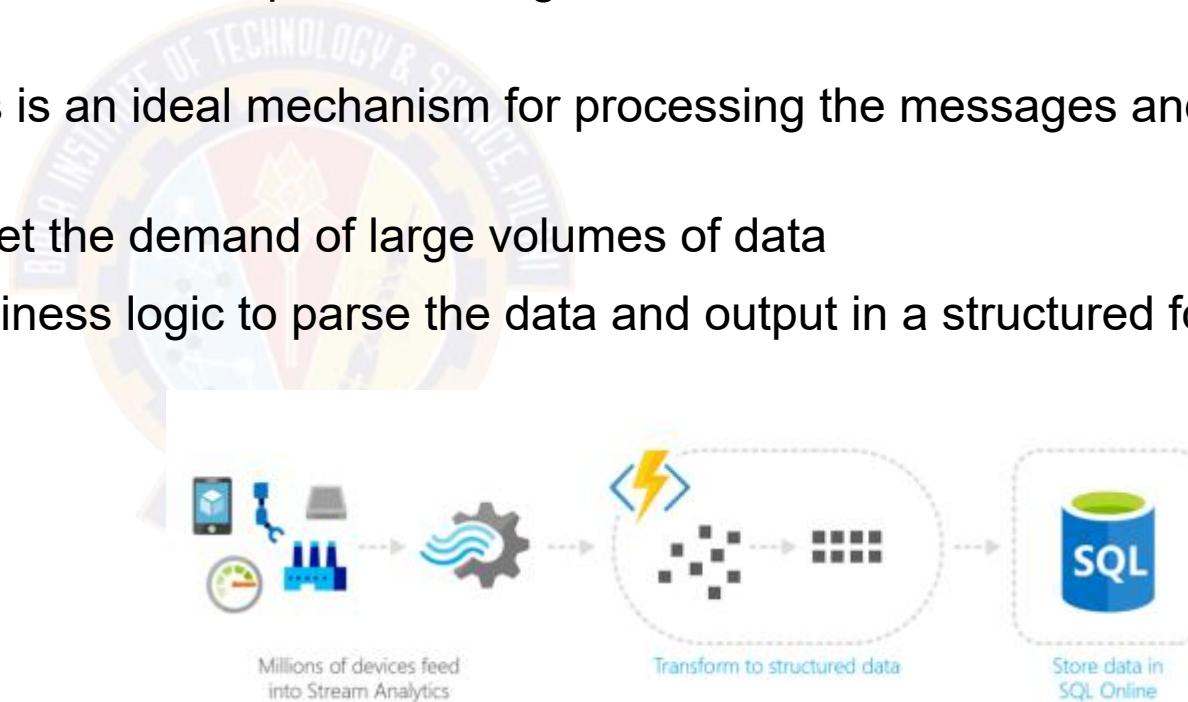
- Serverless functions can be used to facilitate a data pipeline
- For example,
- A file triggers a function to translate data in a CSV file to data rows in a table
- The organized table allows a BI dashboard to present analytics to the end user



Source : Microsoft

Stream processing

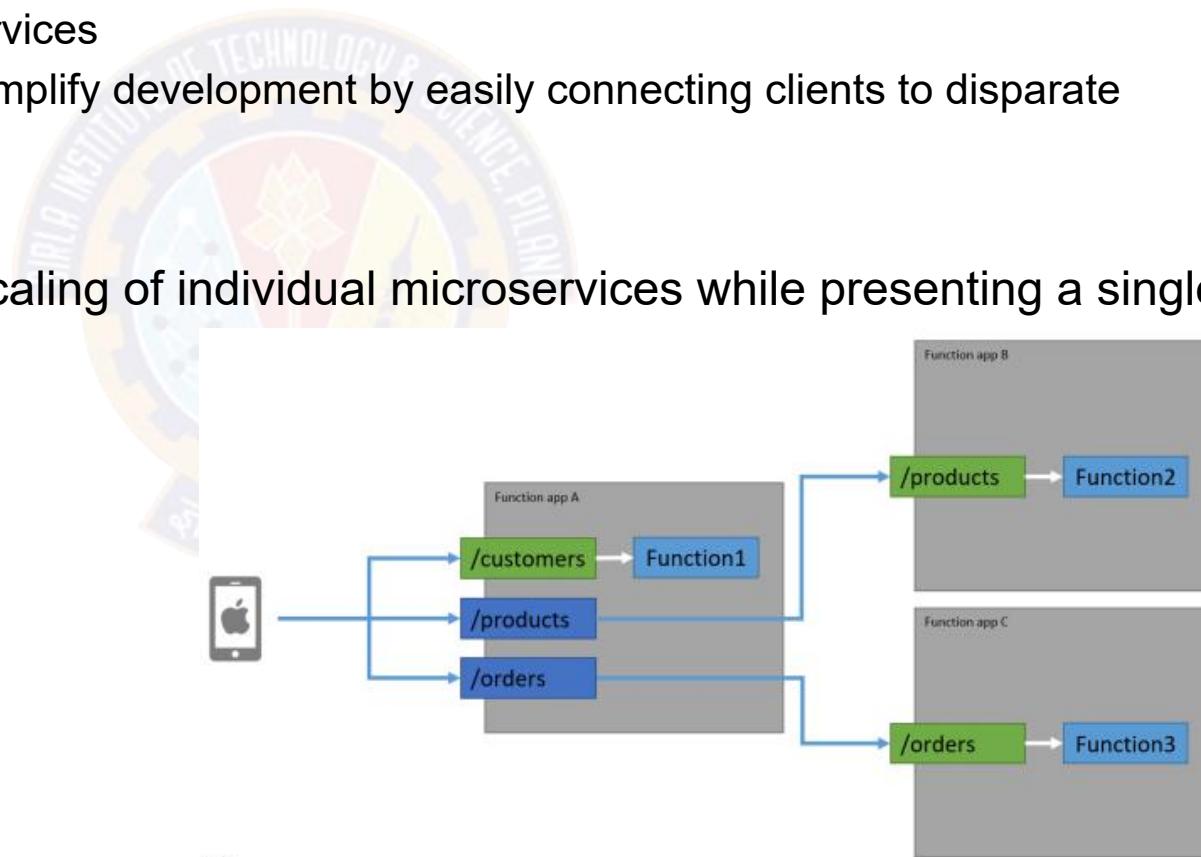
- Devices and sensors often generate streams of data that must be processed in real time
- There are a number of technologies that can capture messages and streams from Event Hubs and IoT Hub to Service Bus
- Regardless of transport, serverless is an ideal mechanism for processing the messages and streams 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

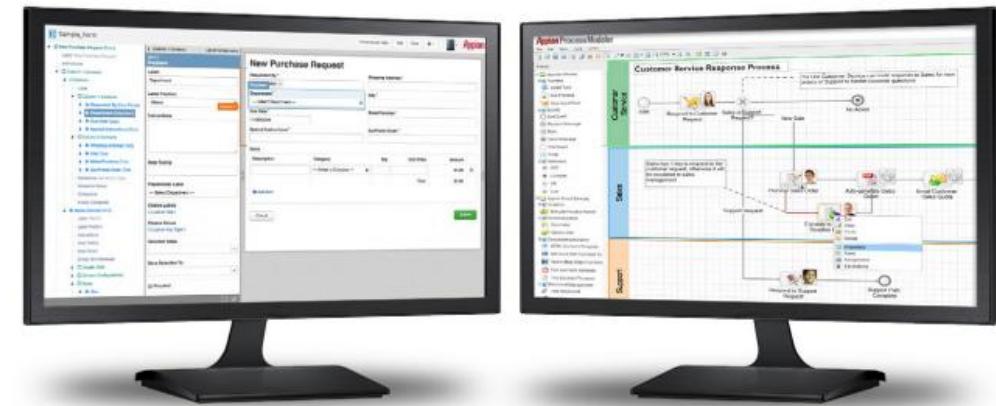
Chandan Ravandur N

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



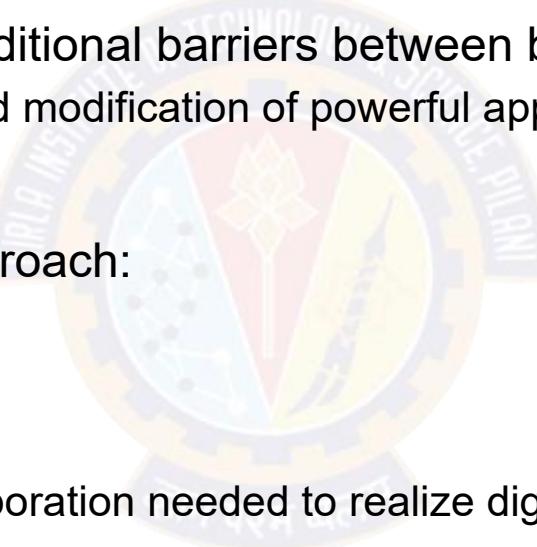
But why did low-code come about?

- 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

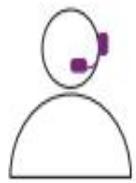
Chandan Ravandur N

Turn your business ideas into apps

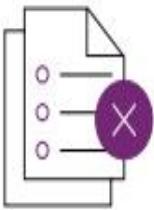
no coding experience required

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:



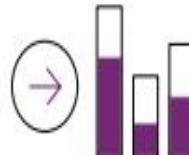
Support better customer
and employee experiences.



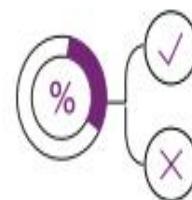
Eliminate paper-based
and manual processes.



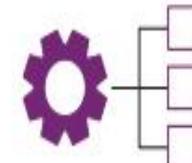
Keep databases and
files up to date.



Democratize dashboards
and analytics.



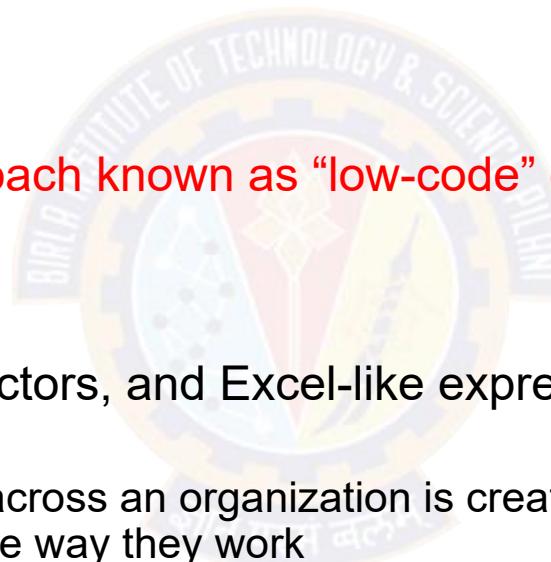
Enable data-based
decision-making.



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

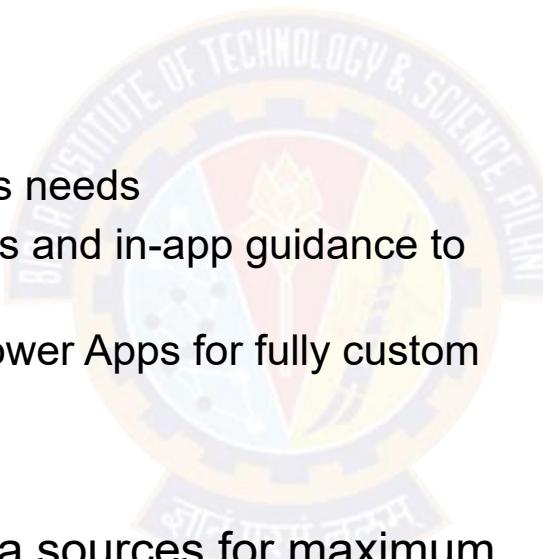
Developing Low Code Apps

- 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.



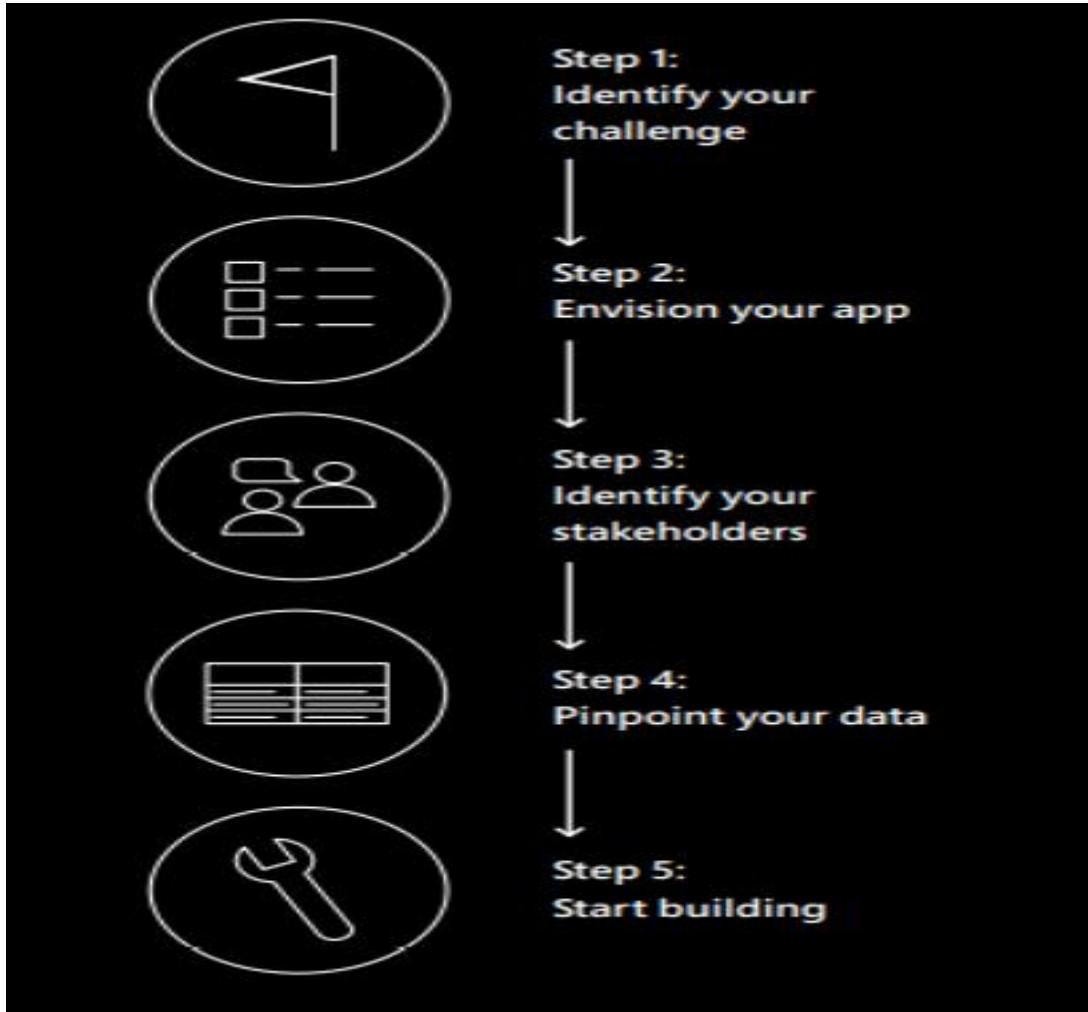
Citizen developer

- 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



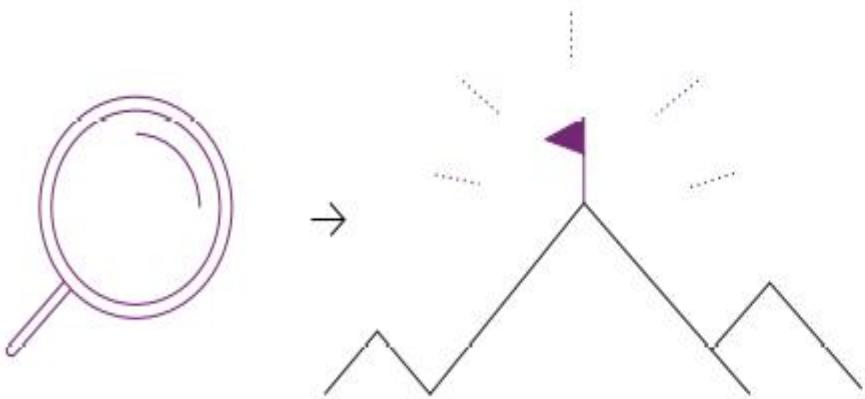
Source : youtube

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.

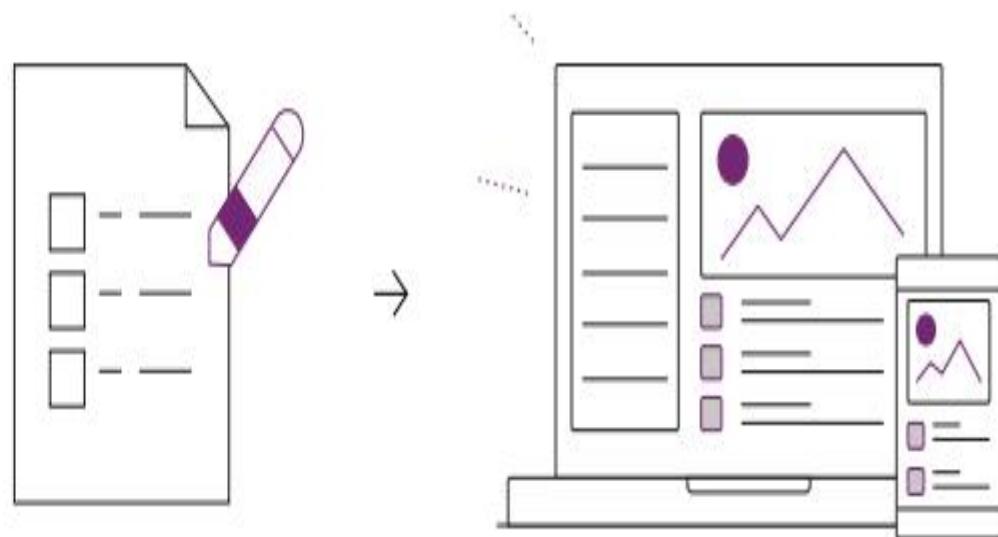


Common uses for Power Apps

- ✓ 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.

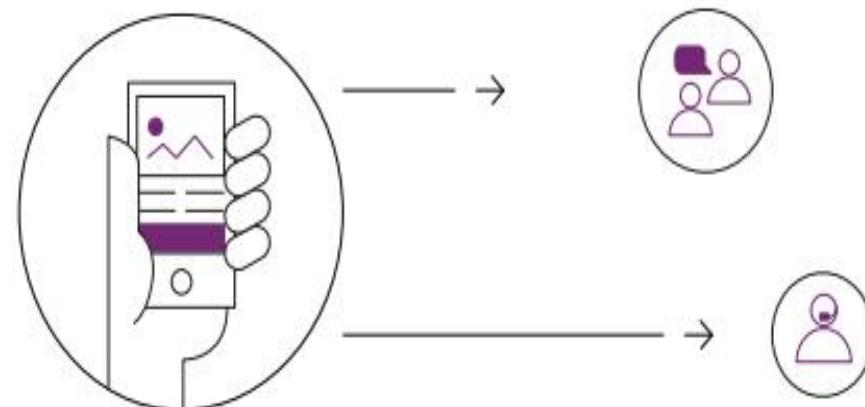


Step 3: Identify your stakeholders

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?



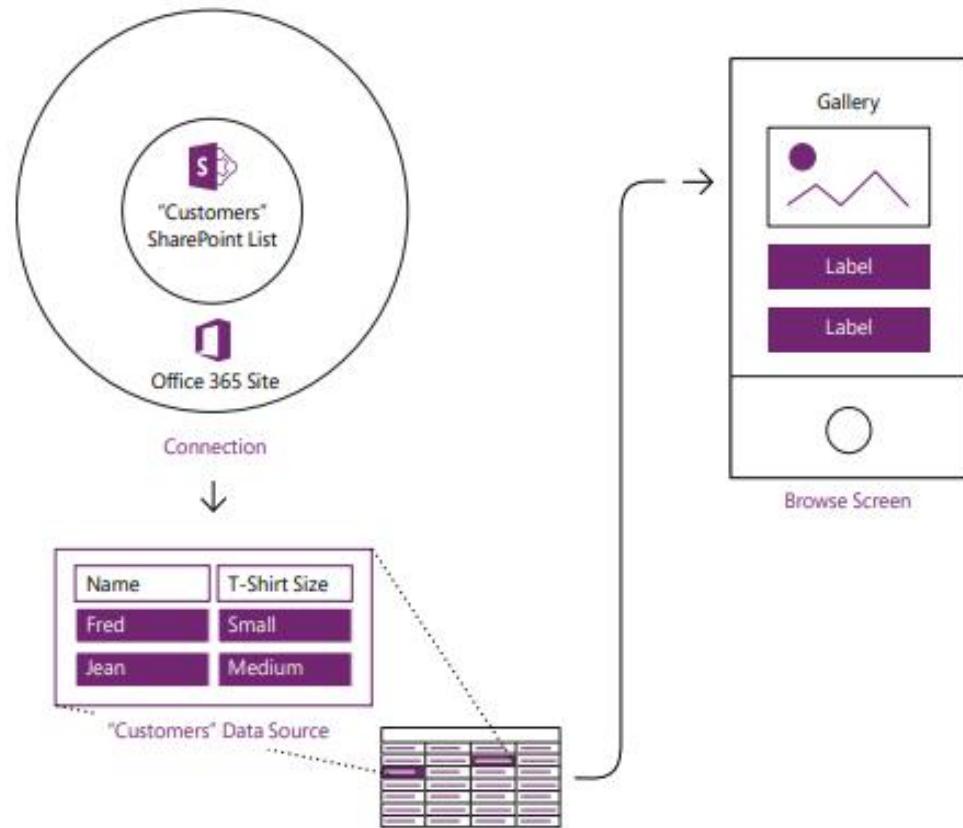
You don't have to address the needs of every stakeholder all at once. In fact, you'll want to start small. However, it's best to understand early on who could benefit from your app or provide valuable input along the way.

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.

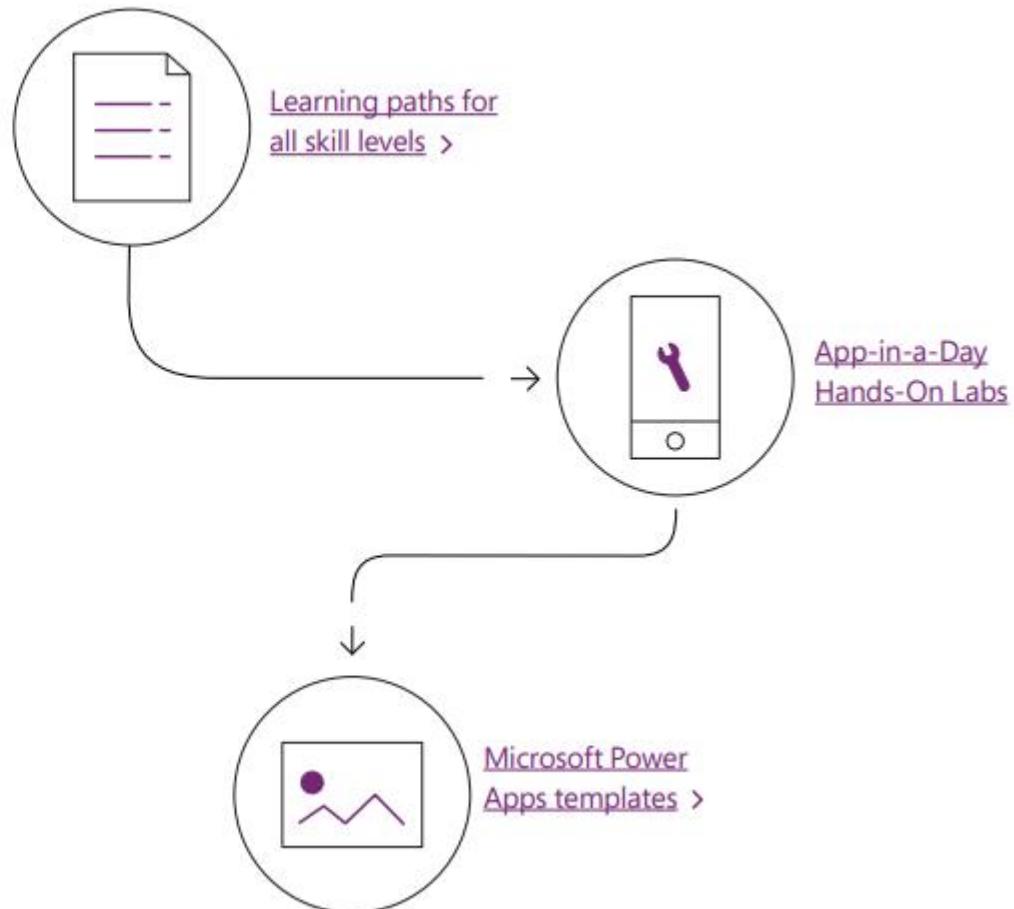
It's also important to consider whether the data might be sensitive or proprietary and work with the appropriate people in your organization to keep it secure and compliant.

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:



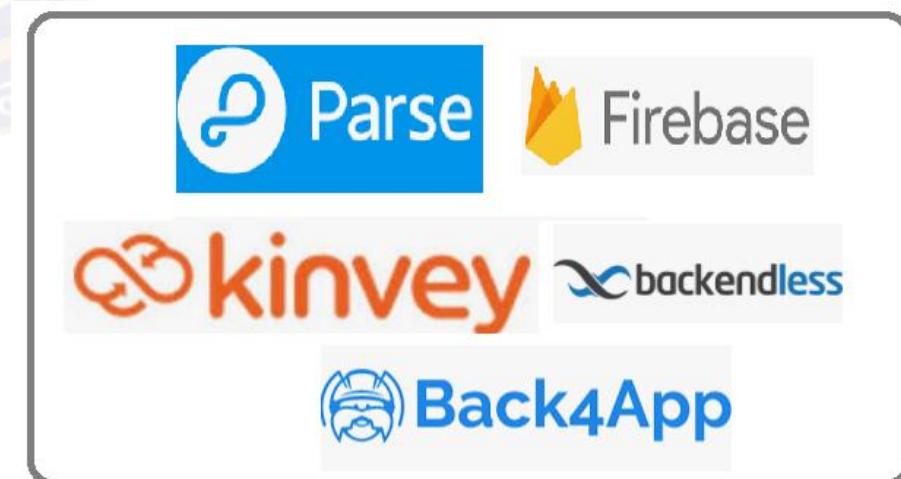
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

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

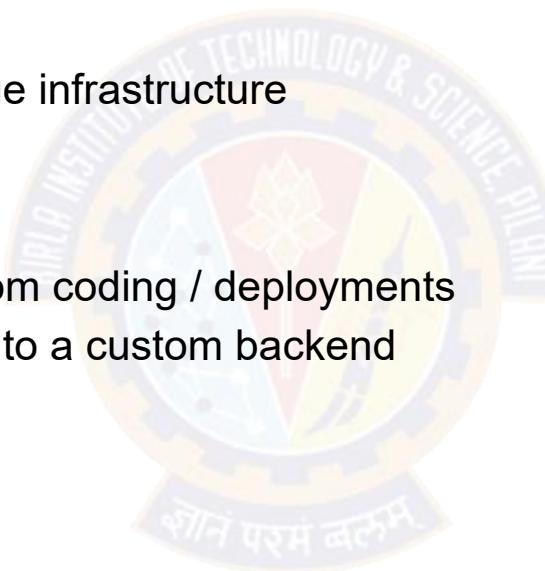
Technical reasons to use BaaS:

- 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

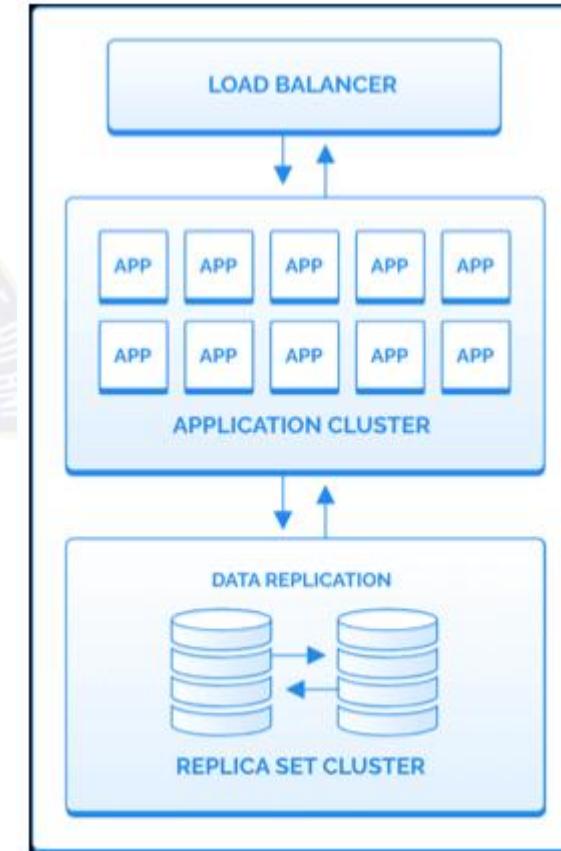
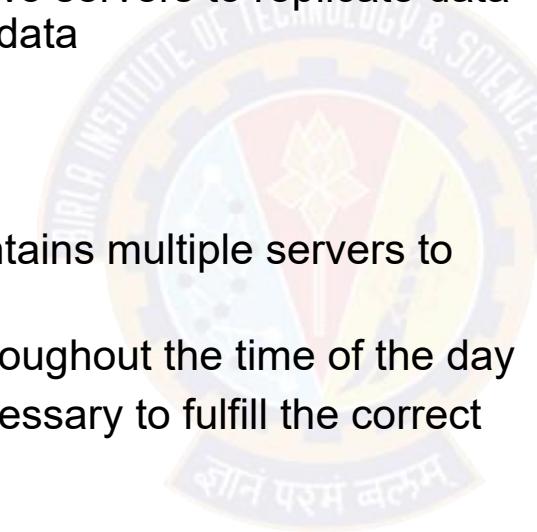
MBaaS

- 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

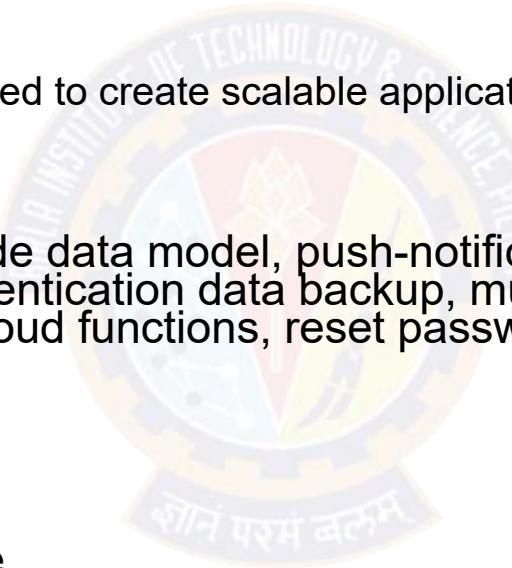


[Image source : Back4App](#)

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:



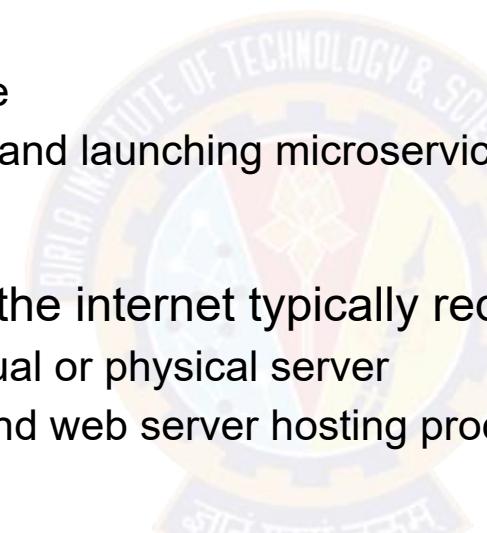
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

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

Functions

Application

Runtime

Containers

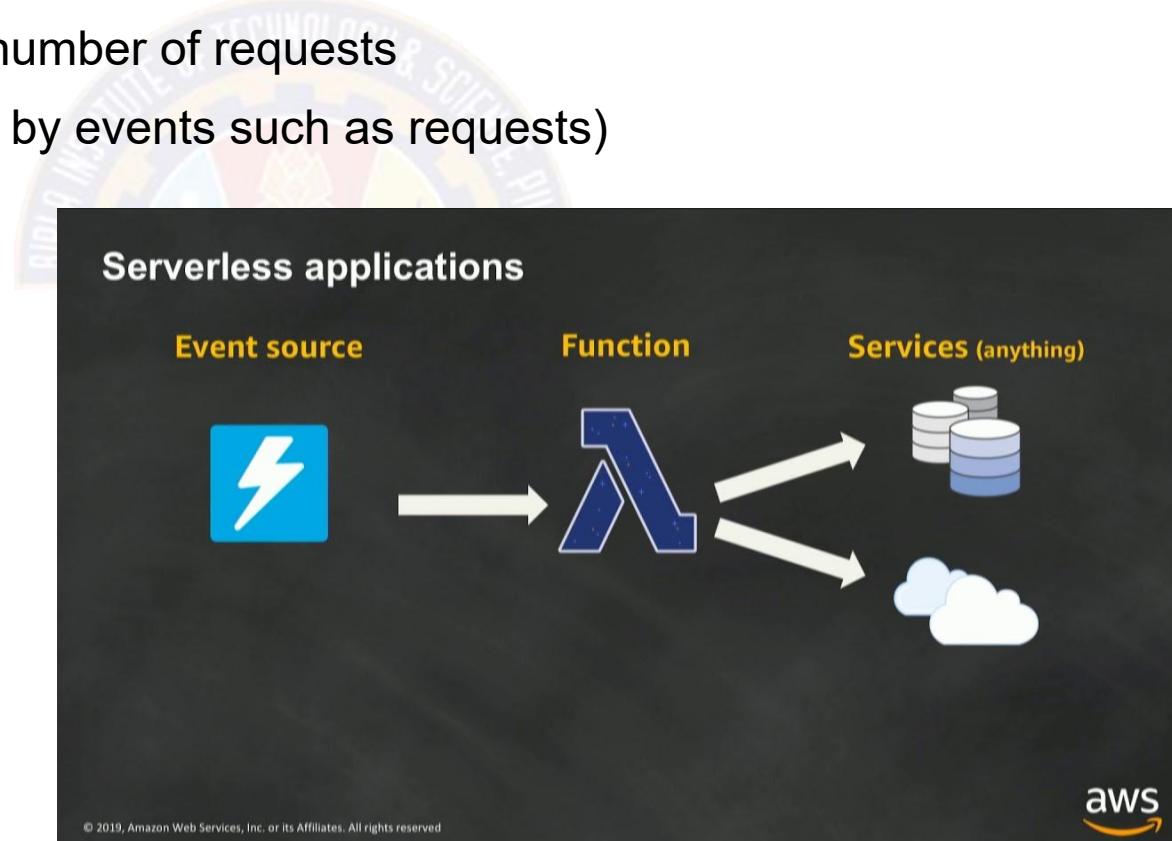
Operating System

Virtualization

Hardware

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)

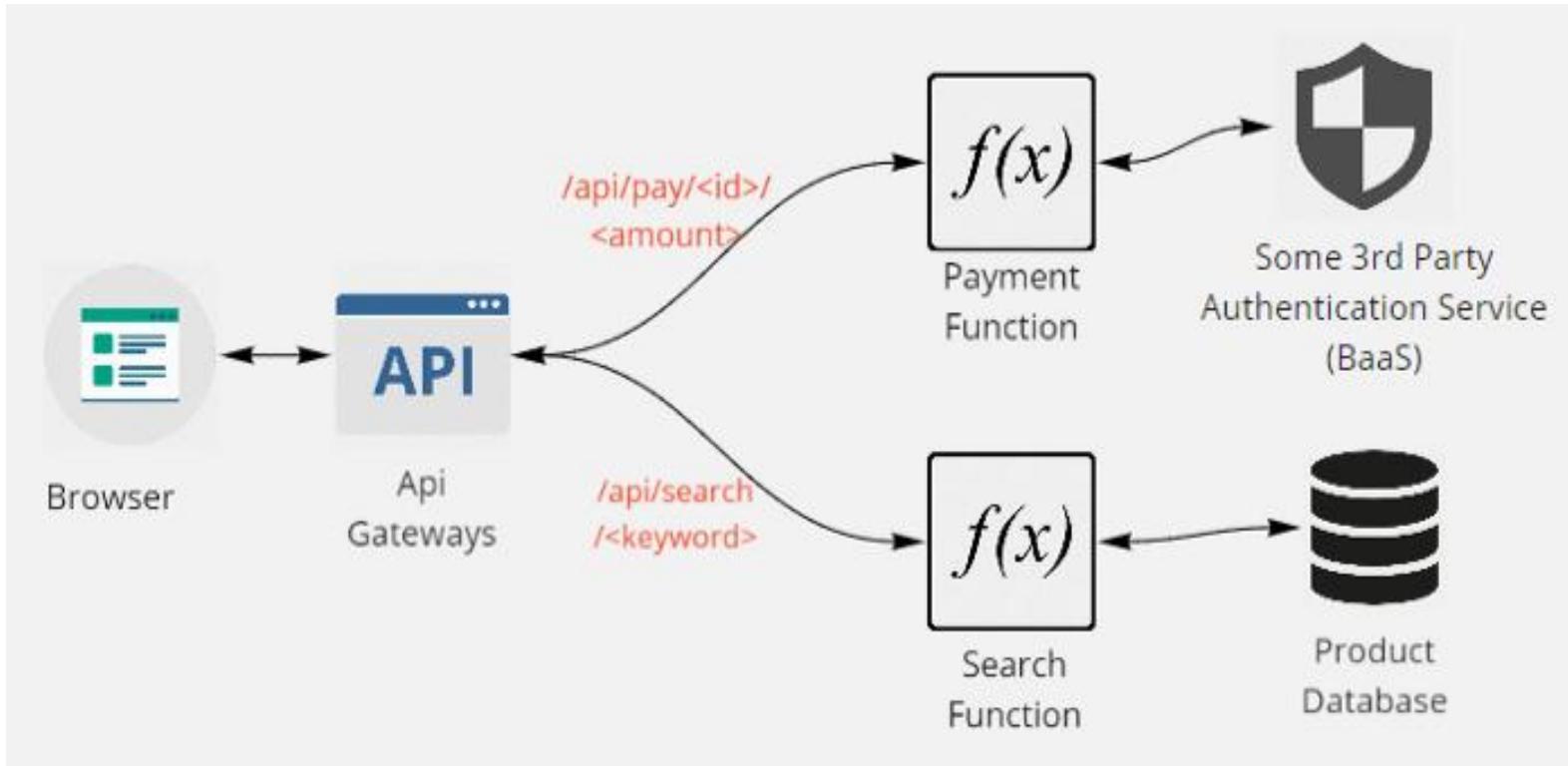


Benefits of FaaS

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

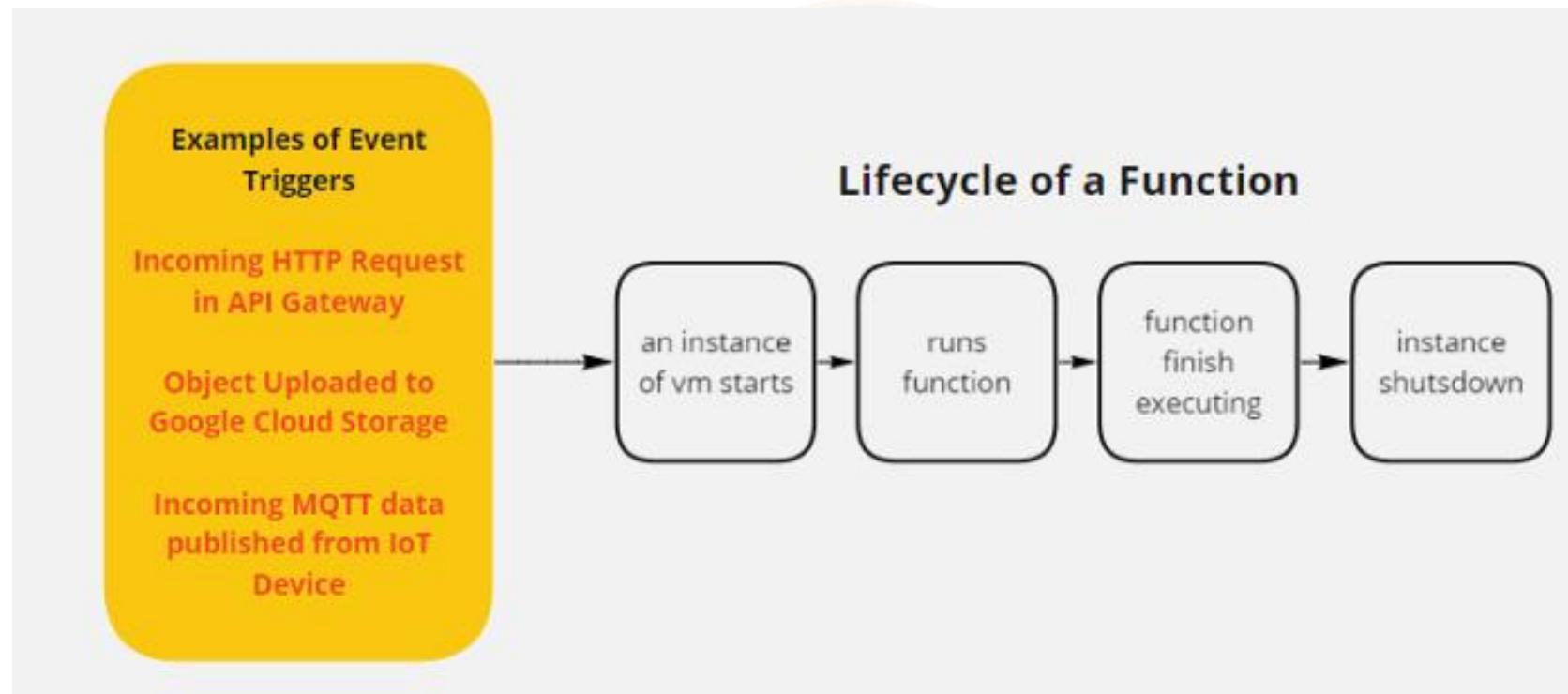
- Focus more on code, not infrastructure:
 - ✓ With FaaS, can divide the server into functions that can be scaled automatically and independently so don't have to manage infrastructure
 - ✓ This allows to focus on the app code and can dramatically reduce time-to-market
- Pay only for the resources you 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](#)

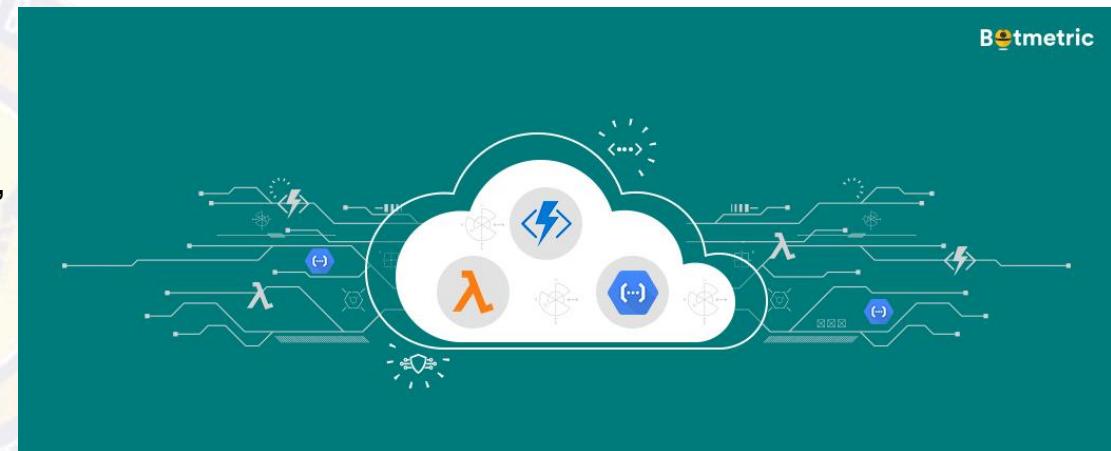
How it works?



Source : [medium](#)

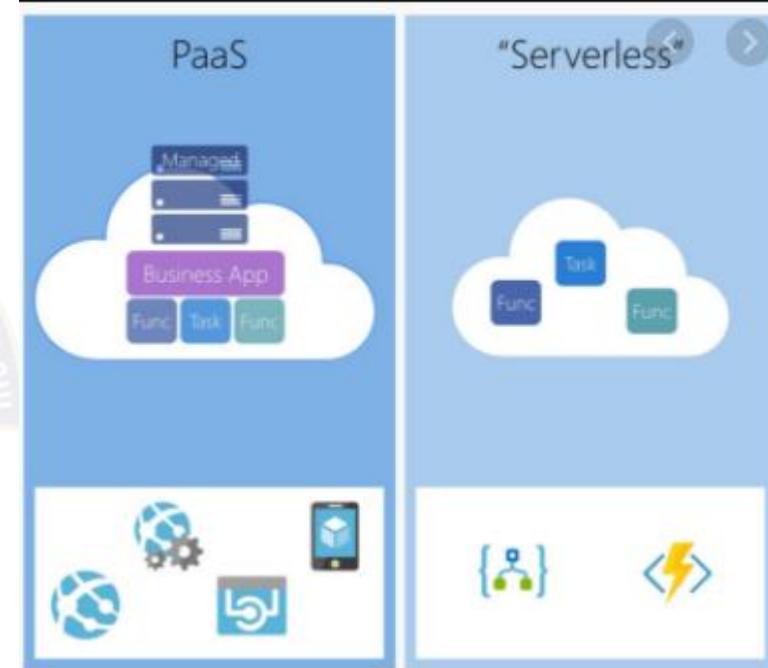
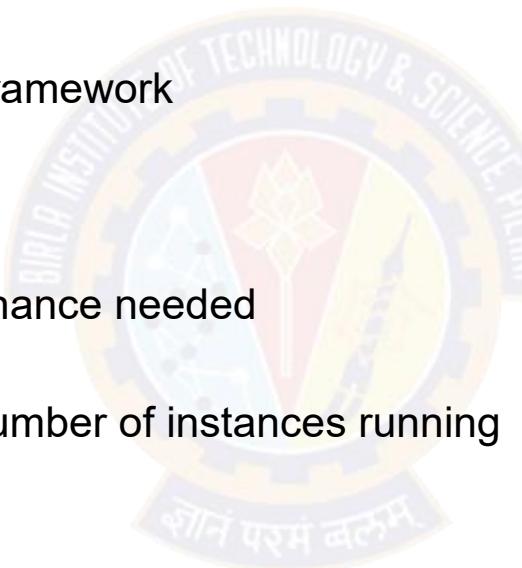
FaaS vs Serverless

- 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

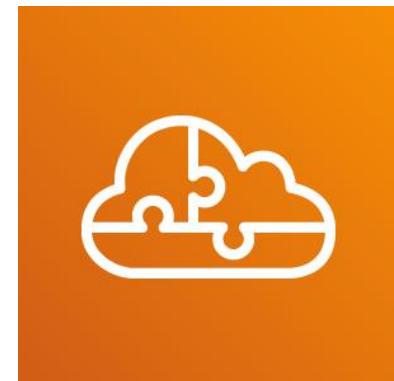
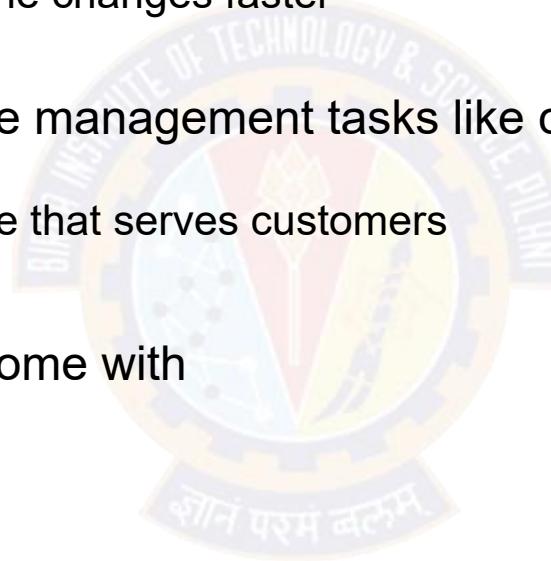
Serverless on AWS

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.

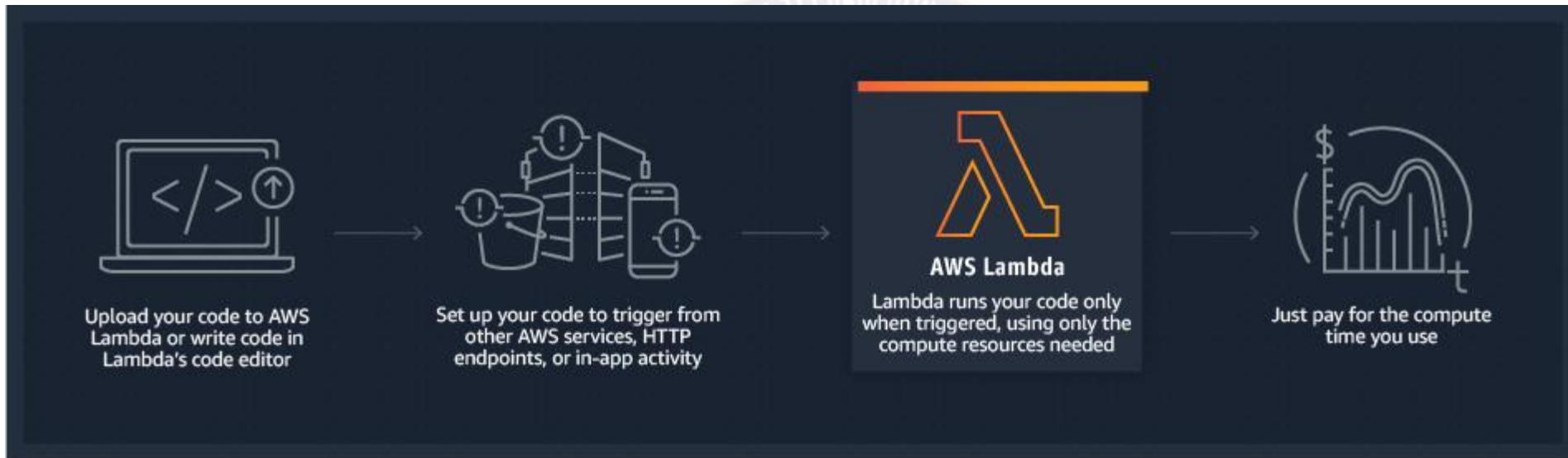
Cost optimized with millisecond metering

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

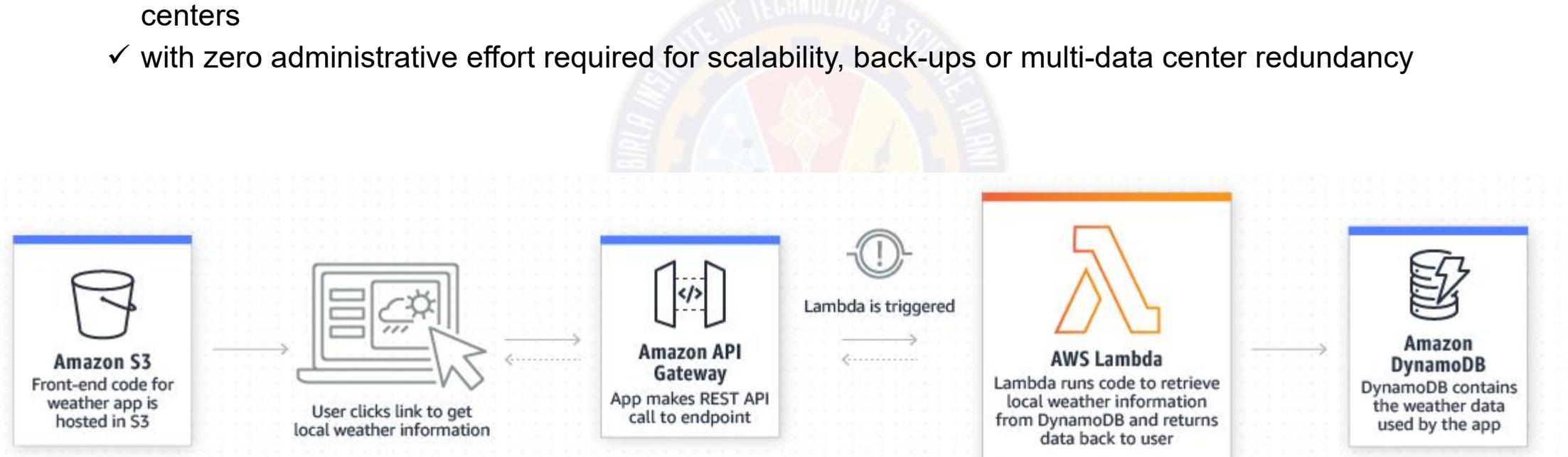
What can you build with AWS Lambda?

- 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

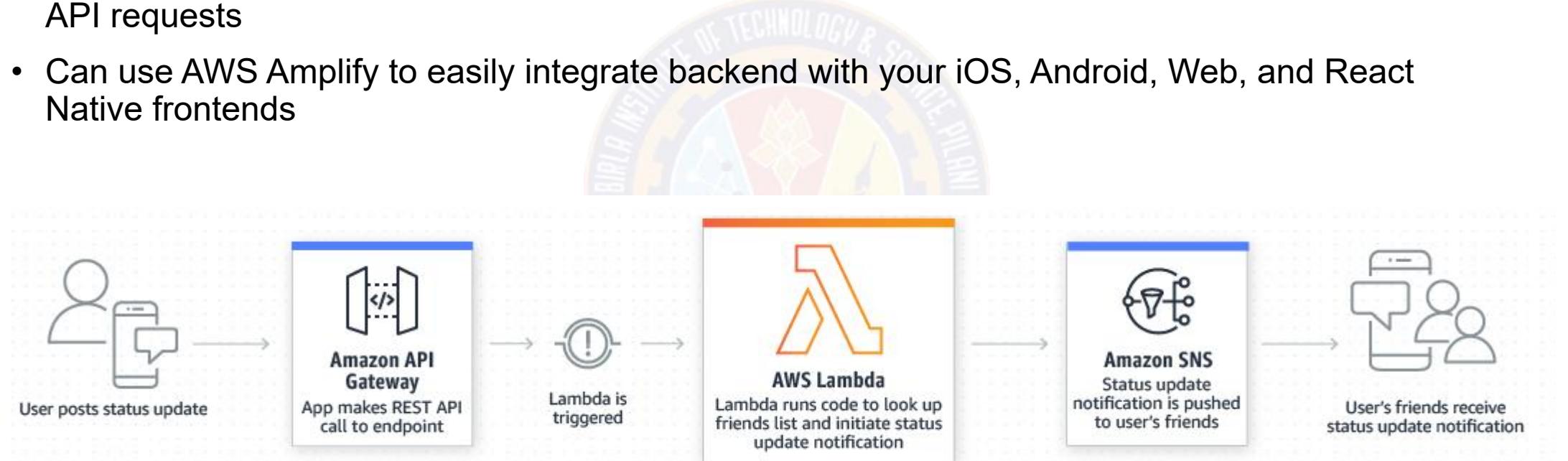
- By combining AWS Lambda with other AWS services, developers can build powerful web applications
 - ✓ that automatically scale up and down and run in a highly available configuration across multiple data centers
 - ✓ with zero administrative effort required for scalability, back-ups or multi-data center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

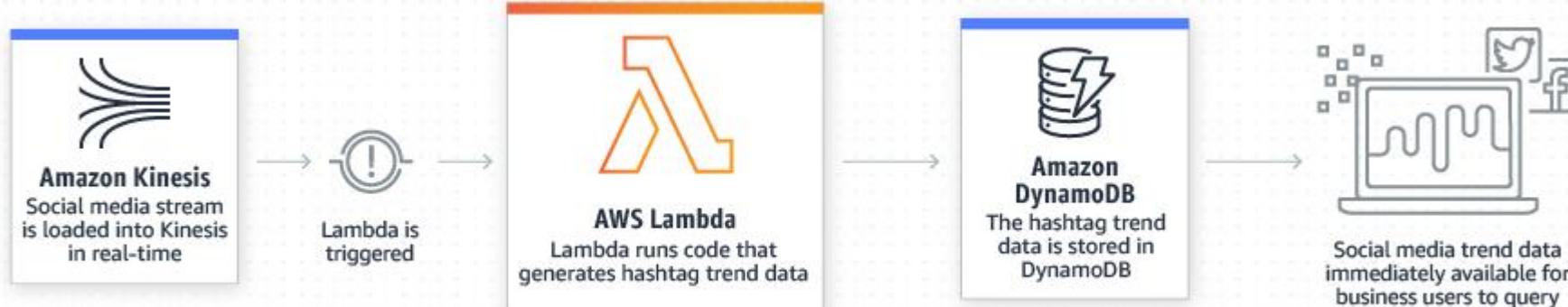
- Can use Amazon S3 to trigger AWS Lambda to process data immediately after an upload
- Can also connect to an existing Amazon EFS file system directly
- enabling massively parallel shared access for large scale file processing
- Can use Lambda to
 - ✓ thumbnail images
 - ✓ transcode videos
 - ✓ index 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

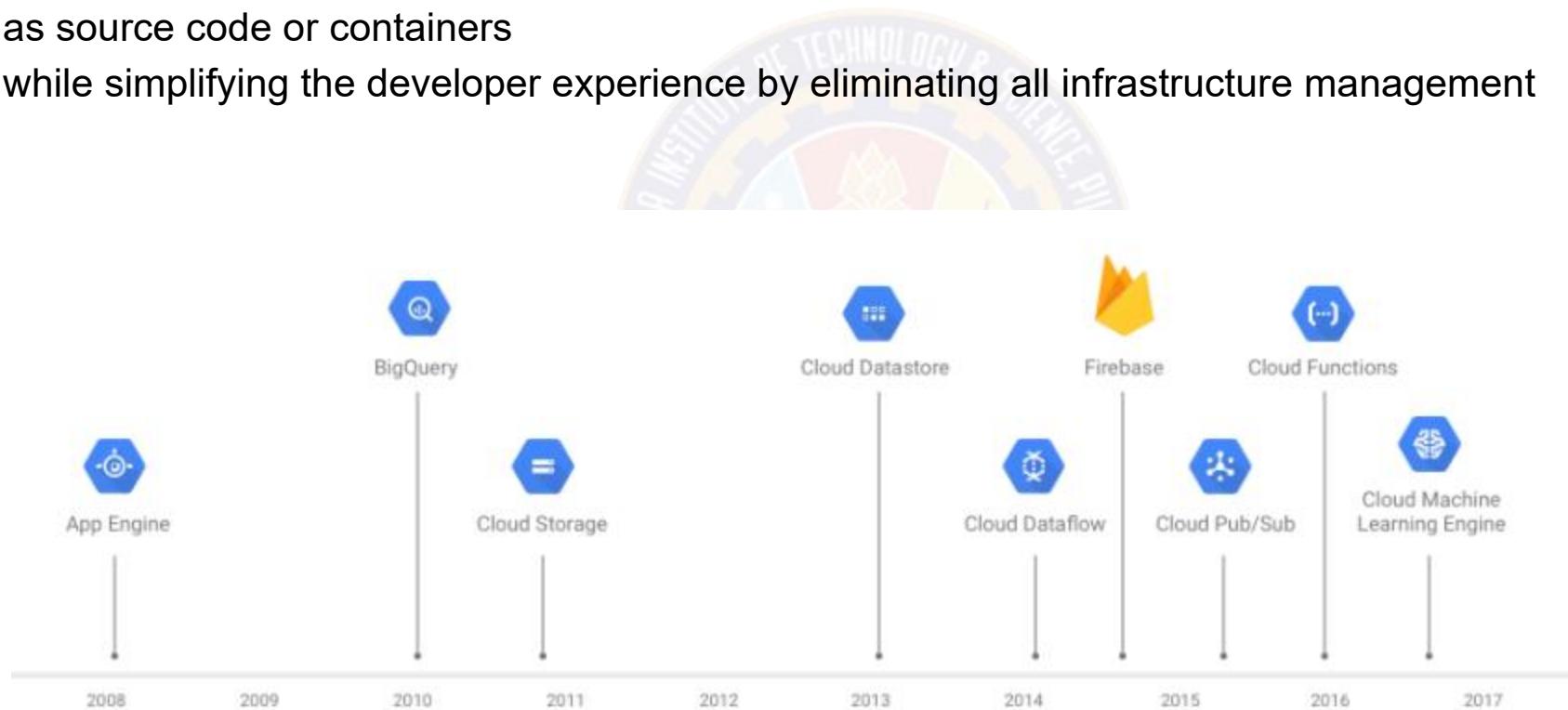
GCP Serverless computing

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

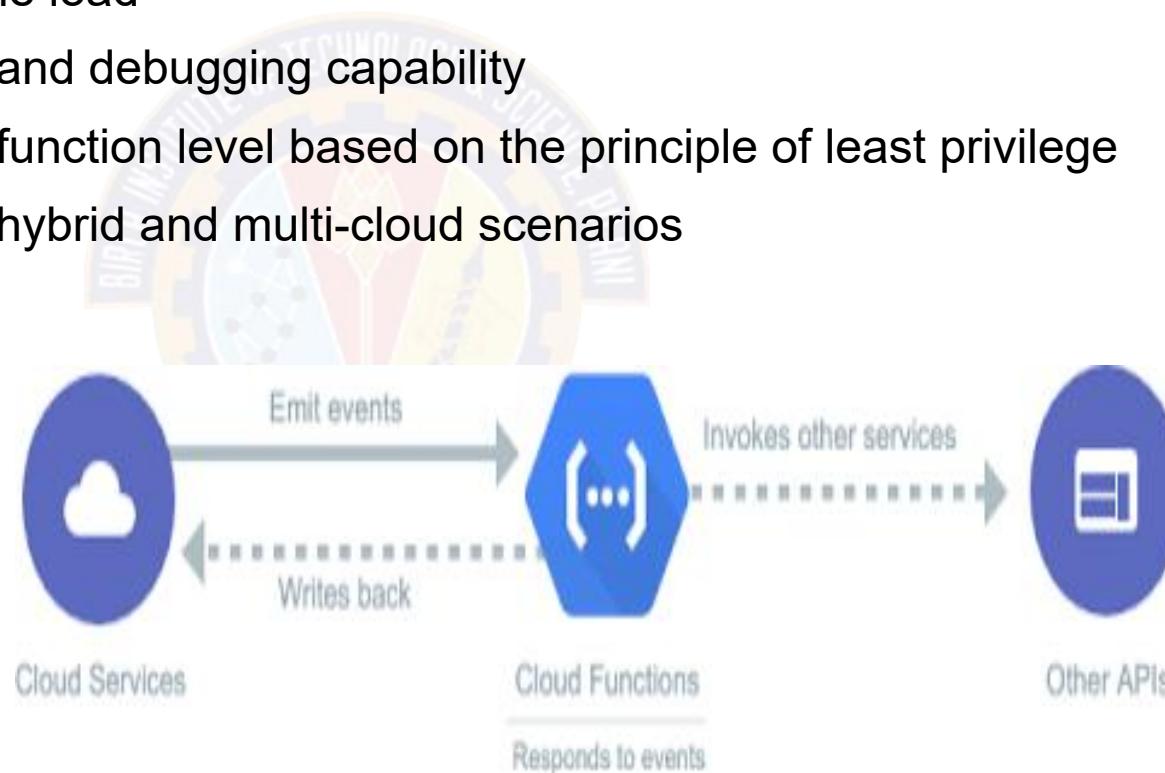
Google Cloud Functions

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



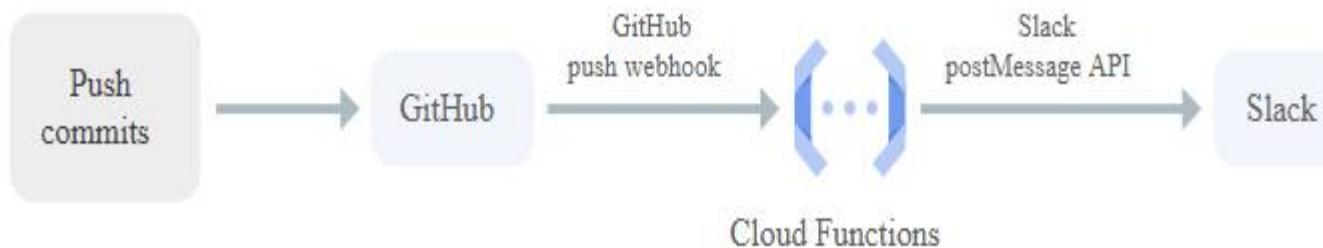
Key features

- 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

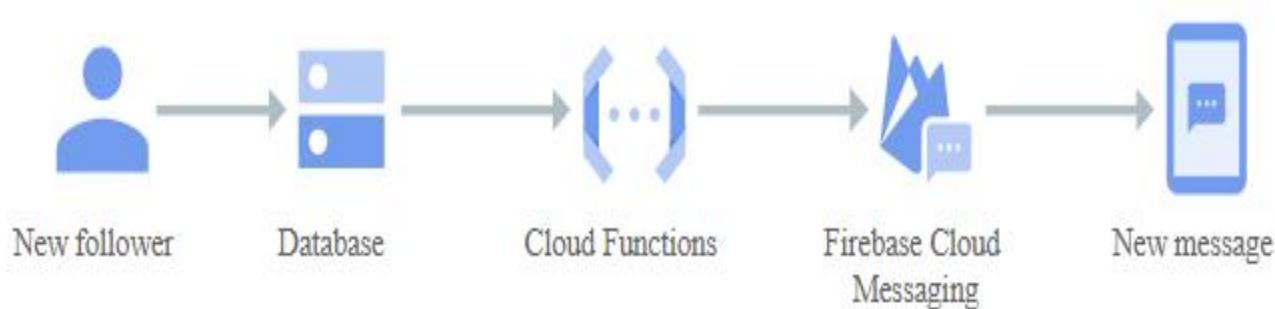
- Use Cloud Functions to
 - ✓ surface out microservices via HTTP APIs
 - ✓ or integrate with third-party services that offer webhook integrations
 - ✓ to quickly extend application with powerful capabilities such as
 - ❖ sending a confirmation email after a successful Stripe payment
 - ❖ or responding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



Use cases (2)

Serverless mobile back ends

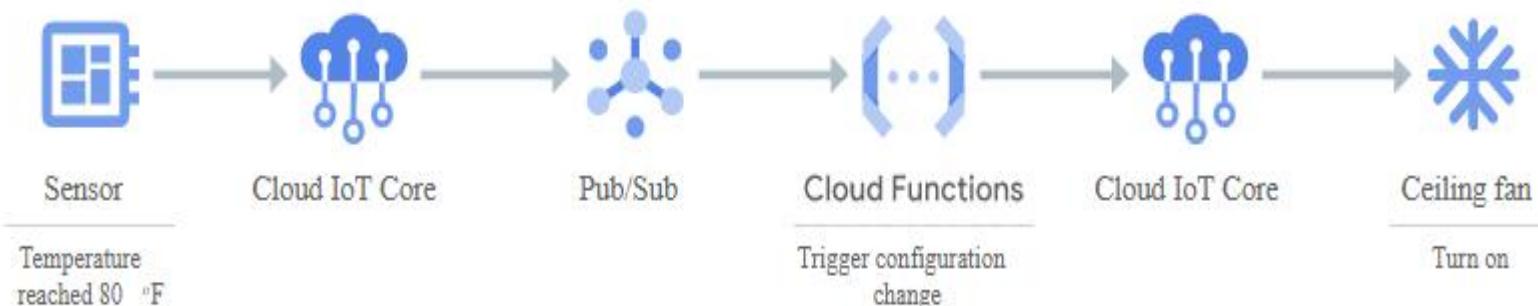
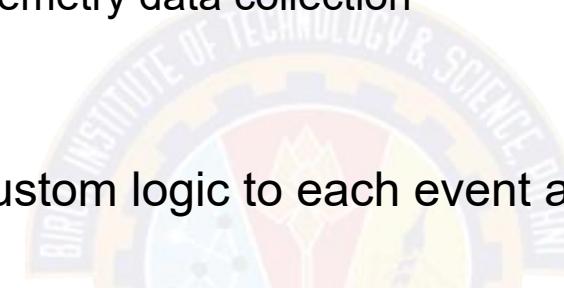
- Use Cloud Functions directly from Firebase to extend application functionality without spinning up a server
- Run code in response to user actions, analytics, and authentication events
 - ✓ to keep users engaged with event-based notifications
 - ✓ to offload CPU- and networking-intensive tasks to Google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



Use cases (3)

Serverless IoT back ends

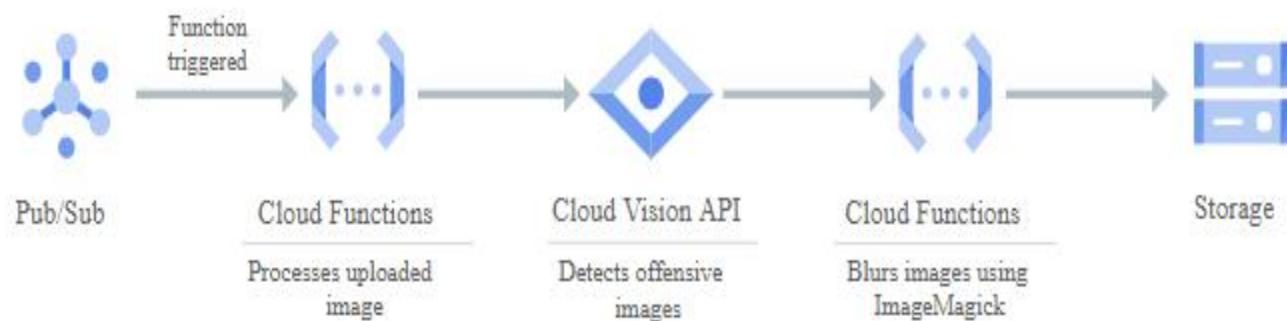
- Use Cloud Functions with Cloud IoT Core and other fully managed services to build back ends for
 - ✓ Internet of Things (IoT) device telemetry data collection
 - ✓ real-time processing
 - ✓ and analysis
- Cloud Functions allows to apply custom logic to each event as it arrives



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: