BITS Pilani
Pilani | Dubai | Goa | Hyderabad

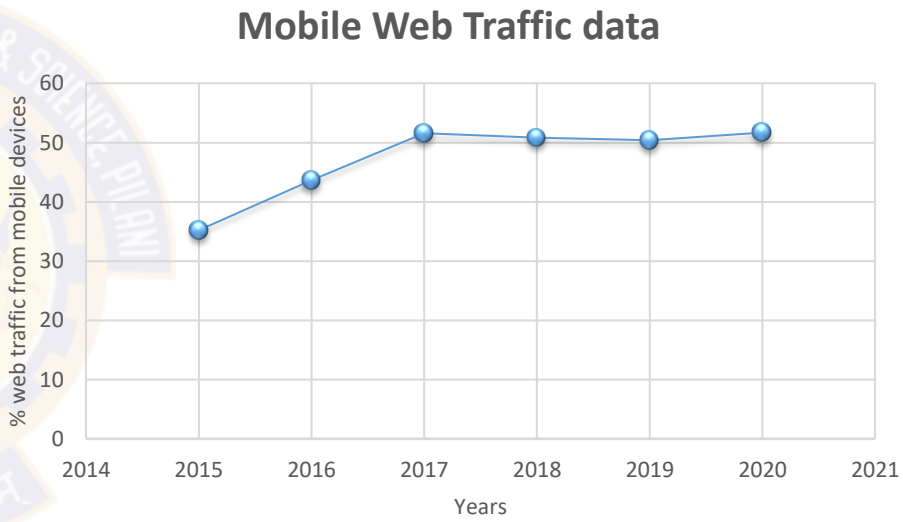
Mobile Apps

Mobile device website traffic

Increasing!

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

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



Mobile Apps vs Mobile Websites

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

Image source : Wikipedia

Mobile Apps

Aka Native Apps

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

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

Source : Wikipedia

Mobile Websites

Aka Responsive mobile websites

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

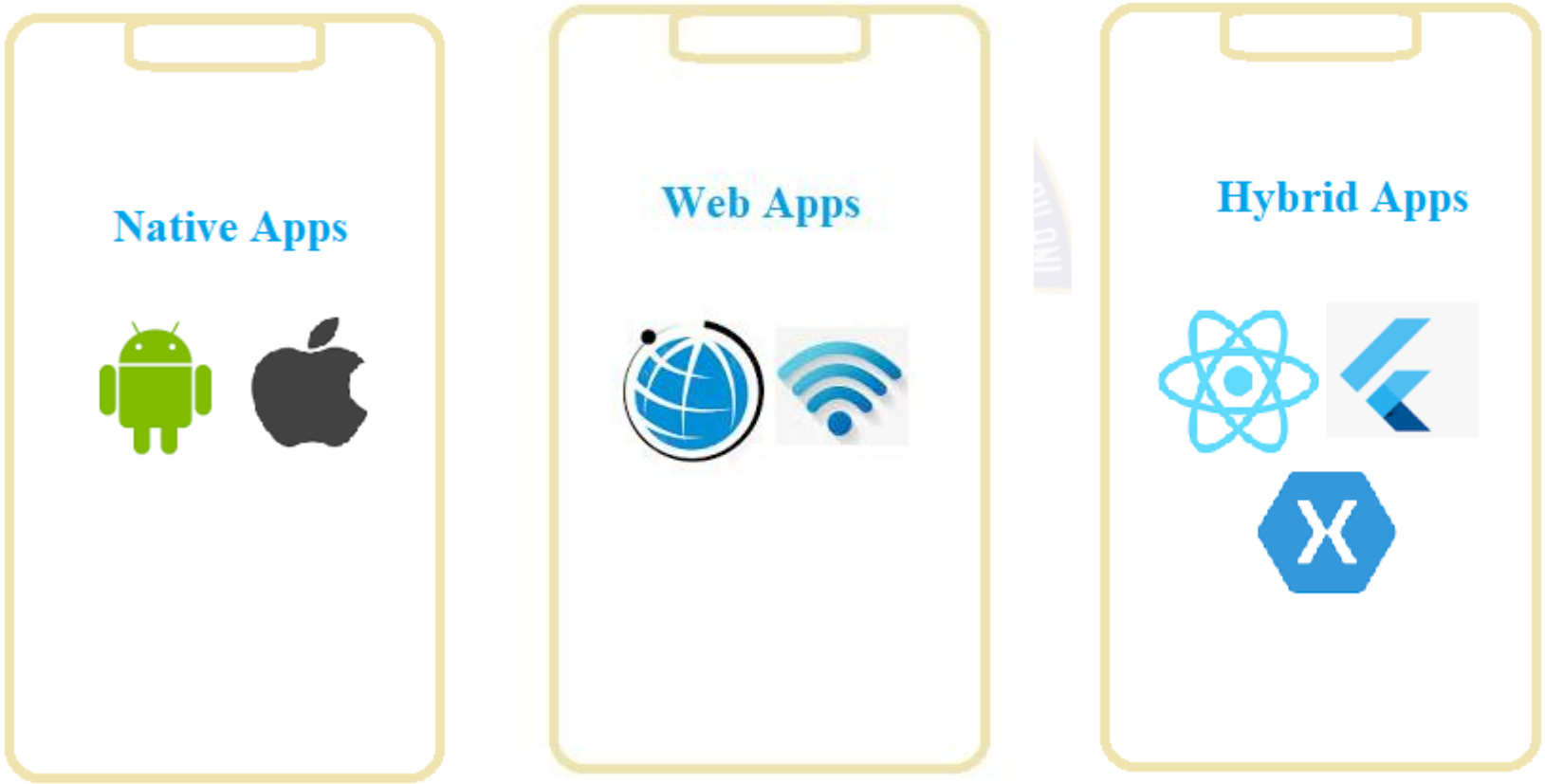
- Advantages
  - Available For All Users
  - Users Don’t Have To Update
  - Cost-Effective

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Mobile Apps - Types

Mobile Apps - Types

Three!



Native Apps

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

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

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

Native Apps

Pros and Cons

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

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

Web Apps

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

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

Web Apps

Pros and Cons

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

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

Hybrid Apps

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

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

Hybrid Apps

Pros and Cons

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

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

Compared!

Key Features: Native, Web, & Hybrid

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

Source : Ionic

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Cloud Native - Explored

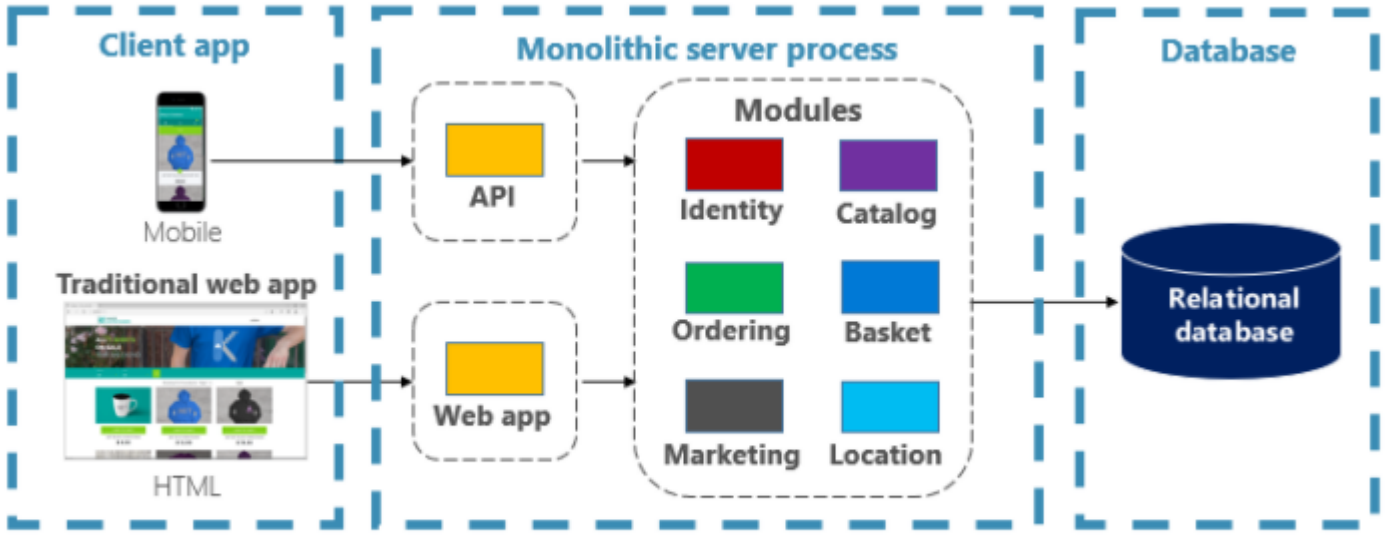
Design “Modern” Web Application

eCommerce App

- Required for start-up
- Should be cutting edge

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

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



Traditional monolithic design

Source : Microsoft

A monolithic application

Conventional Layered Apps

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

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

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

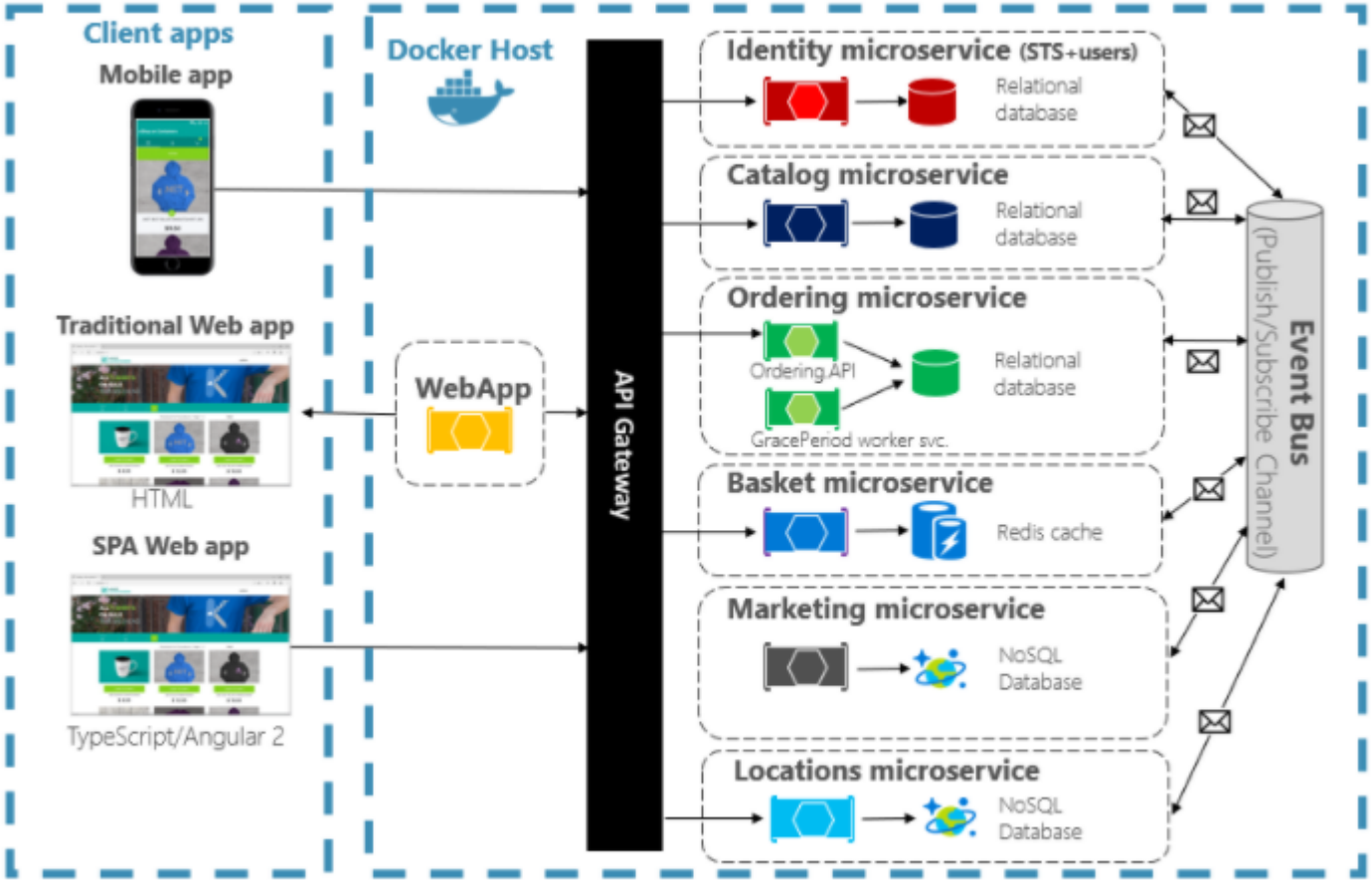
Monolithic Fear Cycle

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

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

Cloud-native applications

Solution



Cloud-native design

Source : Microsoft

Cloud-native design

Solution explained

- Application is decomposed across a set of small isolated microservices

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

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

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

  features found in modern cloud platforms

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Cloud Native Apps - Defined

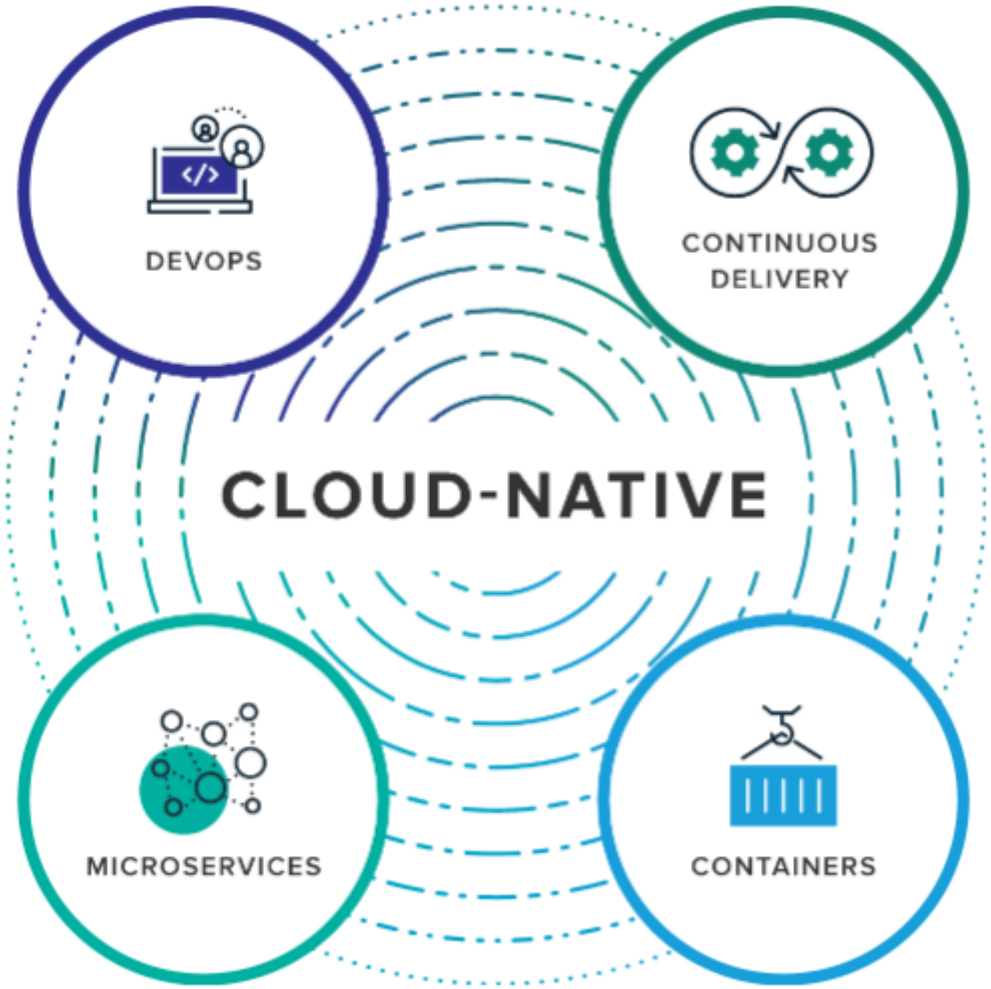
Cloud native

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

- The Cloud Native Computing Foundation provides an official definition:

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

Cloud native Building Blocks



Source : VMWare Blog

Cloud native Building Blocks (2)

DevOps, Continuous Delivery, Microservices & Containers

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

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

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

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

Source : VMWare Blog

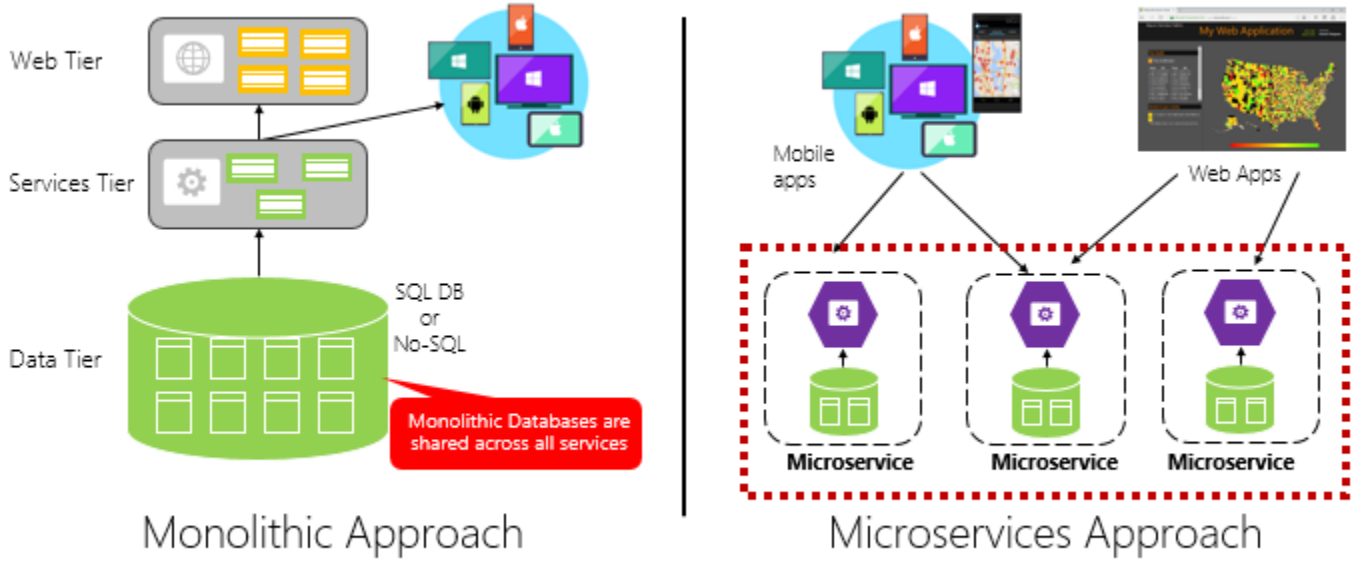
Containers

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

Microservices Architecture

Characteristics

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



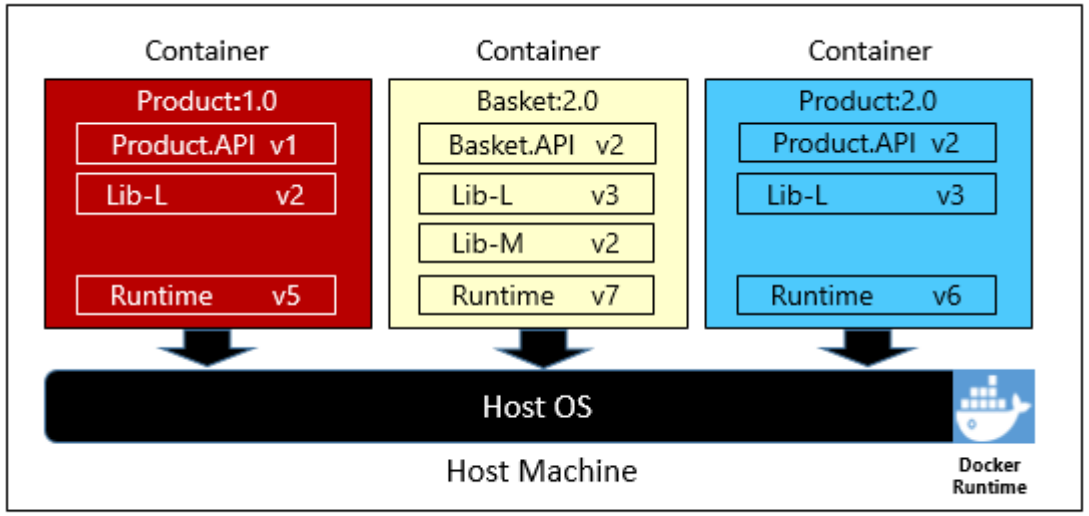
Monolithic deployment versus microservices

Source : Microsoft

Containerizing a Microservices

Simple and straightforward

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



Multiple containers running on a container host

Source : Microsoft

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Cloud Native Apps - Compared

Advantages

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

Source: IBM Blog

Disadvantages

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

Cloud native vs. traditional applications

Cloud native vs. Cloud enabled

- A cloud enabled application is an application that was developed for deployment in a traditional data center but was later changed so that it also could run in a cloud environment

- Cloud native applications, however, are built to operate only in the cloud

- Developers design cloud native applications to be
  - Scalable
  - platform agnostic
  - and comprised of microservices

Cloud native vs. traditional applications (2)

Cloud native vs. Cloud ready

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

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

Cloud native vs. traditional applications (3)

Cloud native vs. Cloud based

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

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

Cloud native vs. traditional applications (4)

Cloud native vs. Cloud first

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

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

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Serverless Computing

Conventional Web Apps

Issues

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

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

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

Serverless

Defined

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

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

- Servers are still running the code!

Source : Serverless

Serverless Apps

Two types

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

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

Serverless – Changes Required

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

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

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

Serverless - Compared

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

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

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Serverless Apps

AWS

Serverless Offering

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

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

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

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

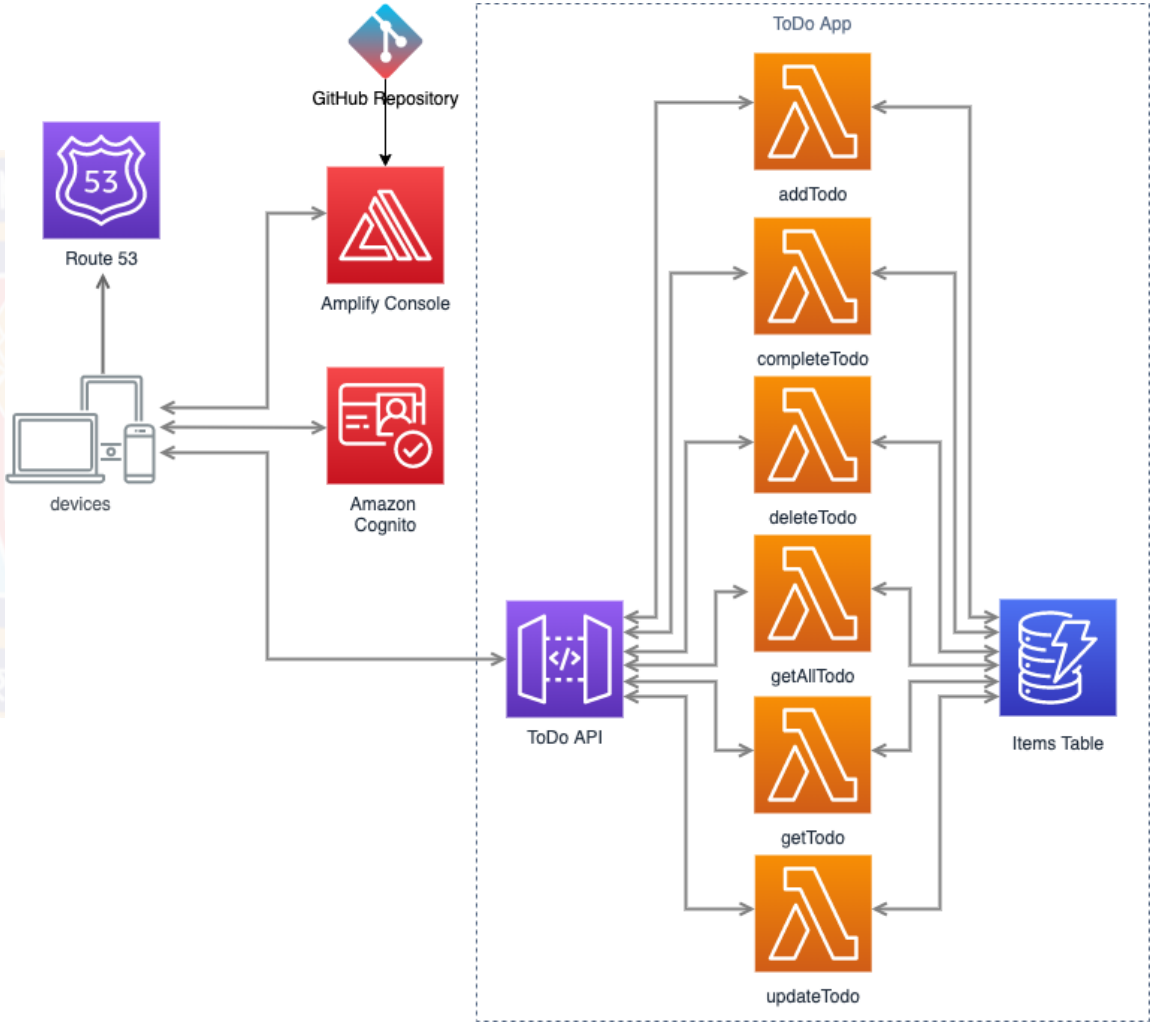
Source : AWS Serverless

AWS Serverless Architecture

Reference Architecture – Web Application

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

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

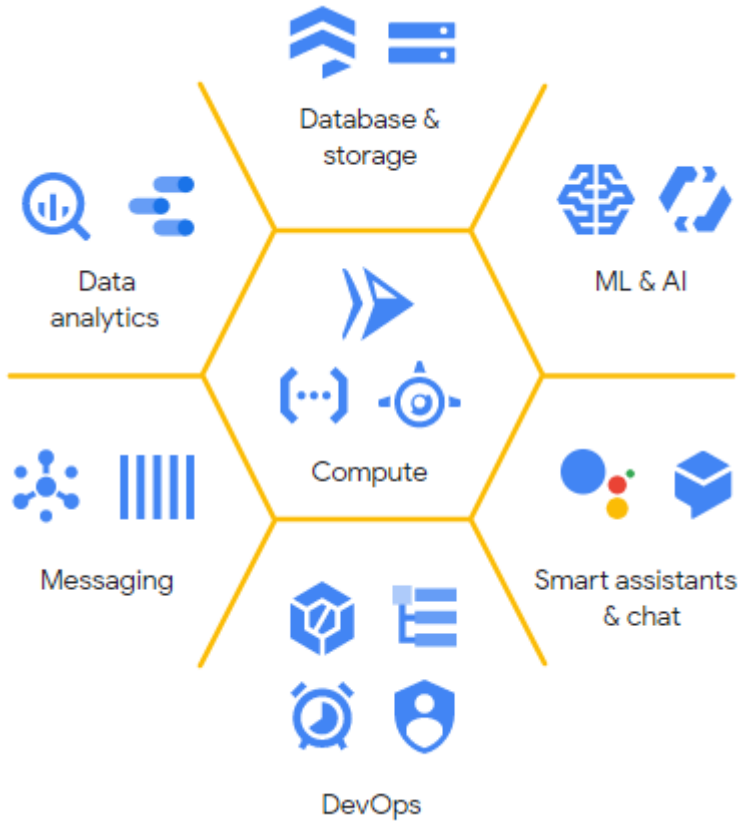


Source : AWS

Google Cloud Platform

Serverless computing

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

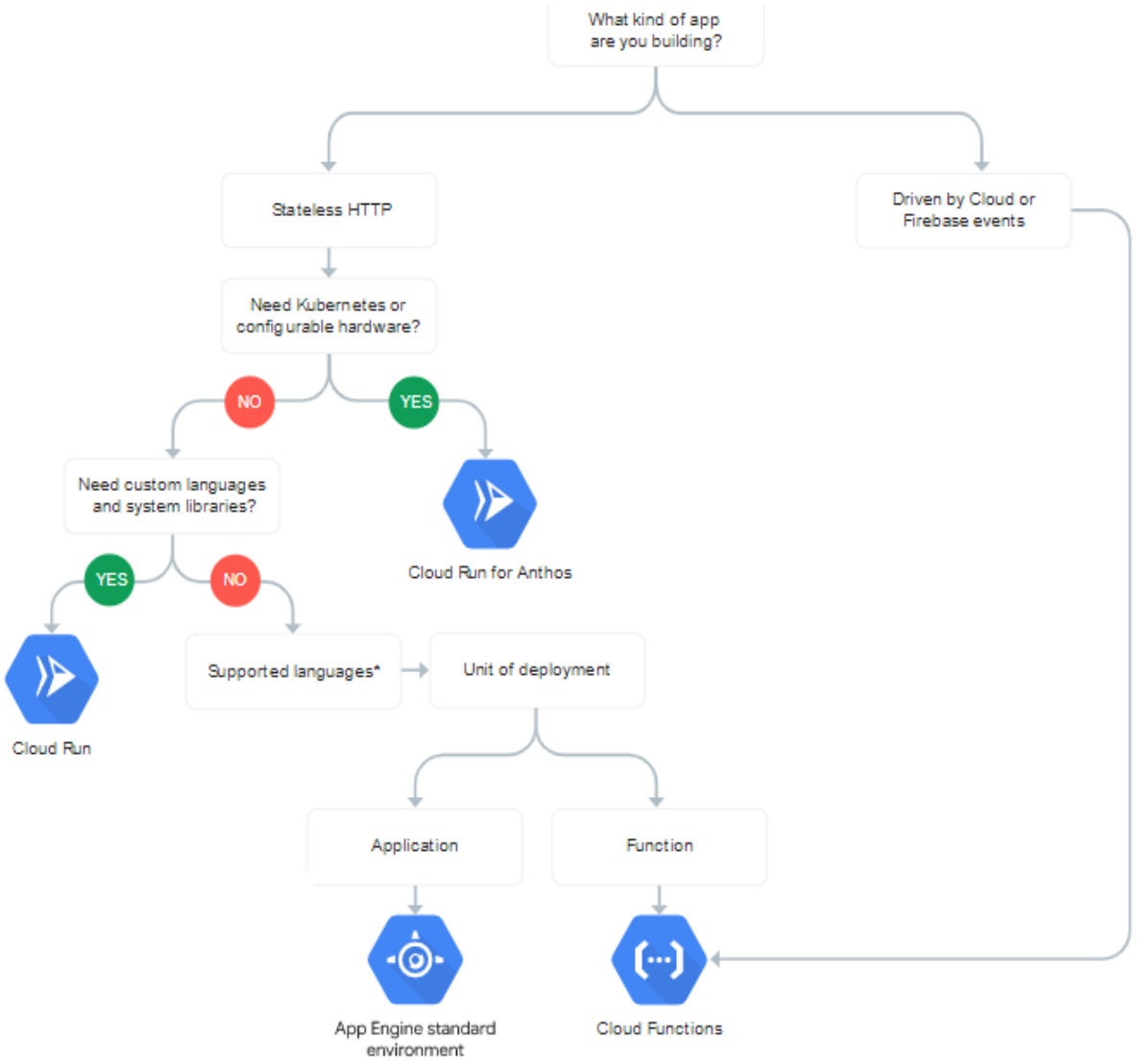


Source : Google Product Page

Google Cloud Platform (2)

Services Offered

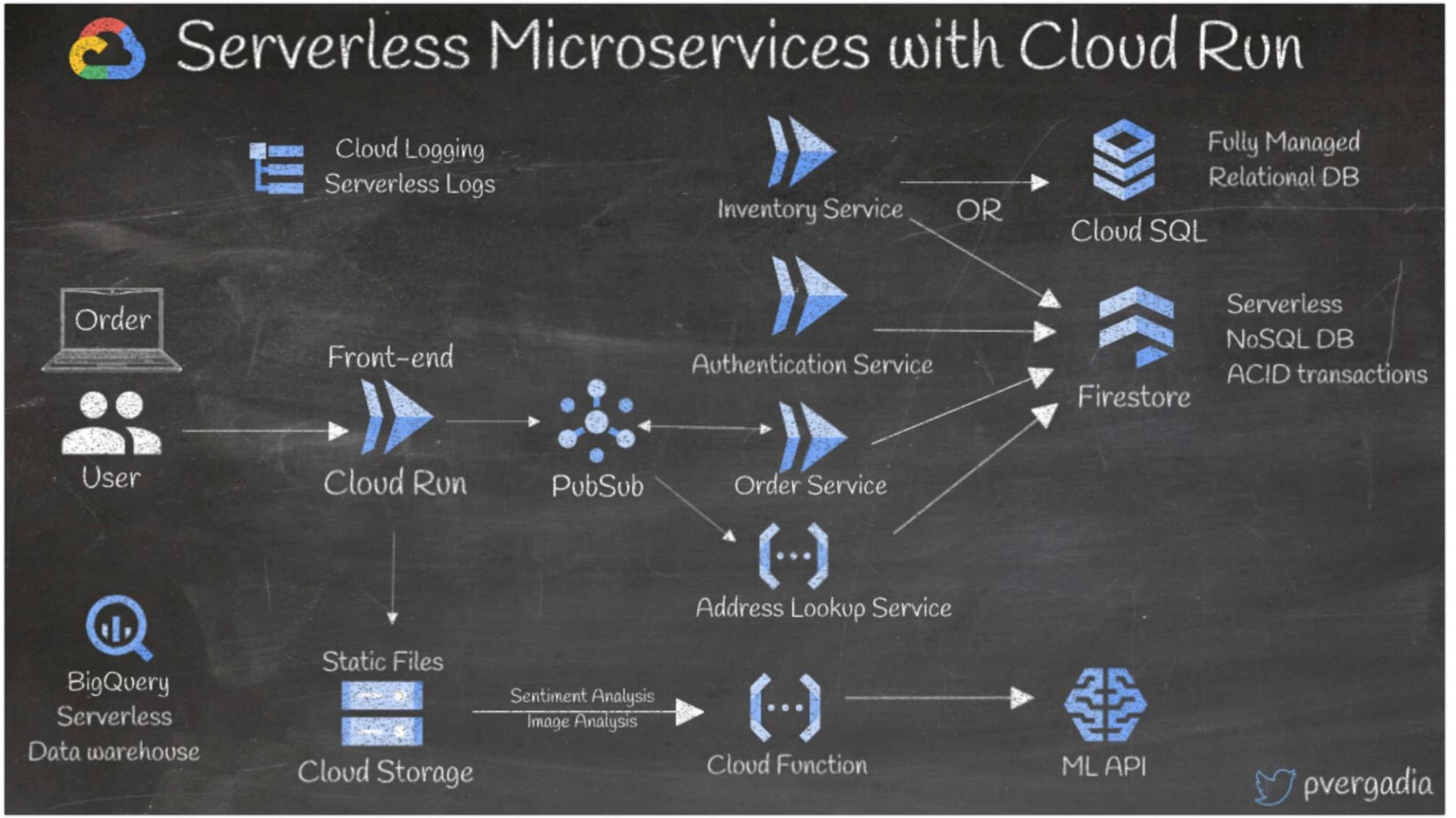
- Cloud Functions
  - An event-driven compute platform to easily connect and extend Google and third-party cloud services and build applications that scale from zero to planet scale.
- App Engine standard environment
  - A fully managed Serverless application platform for web and API backends. Use popular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



Source : Google Product Page

GCP Serverless Architecture

Example – Order Processing



Source : Google

GCP Serverless Architecture (2)

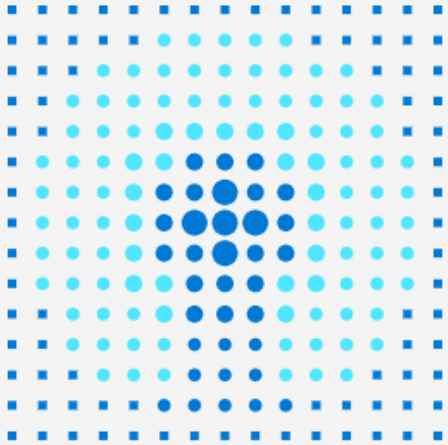
Example - Flow

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

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

Microsoft Azure

Serverless application patterns



Serverless functions

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

Serverless Kubernetes

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

Serverless workflows

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

Serverless application environments

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

Serverless API gateway

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

Source : MS Serverless

Microsoft Azure Serverless Architecture

Example – Architect scalable e-commerce web app

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

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

3 User searches forproducts and queries SQL database.

4 Web site pulls product catalog from database.

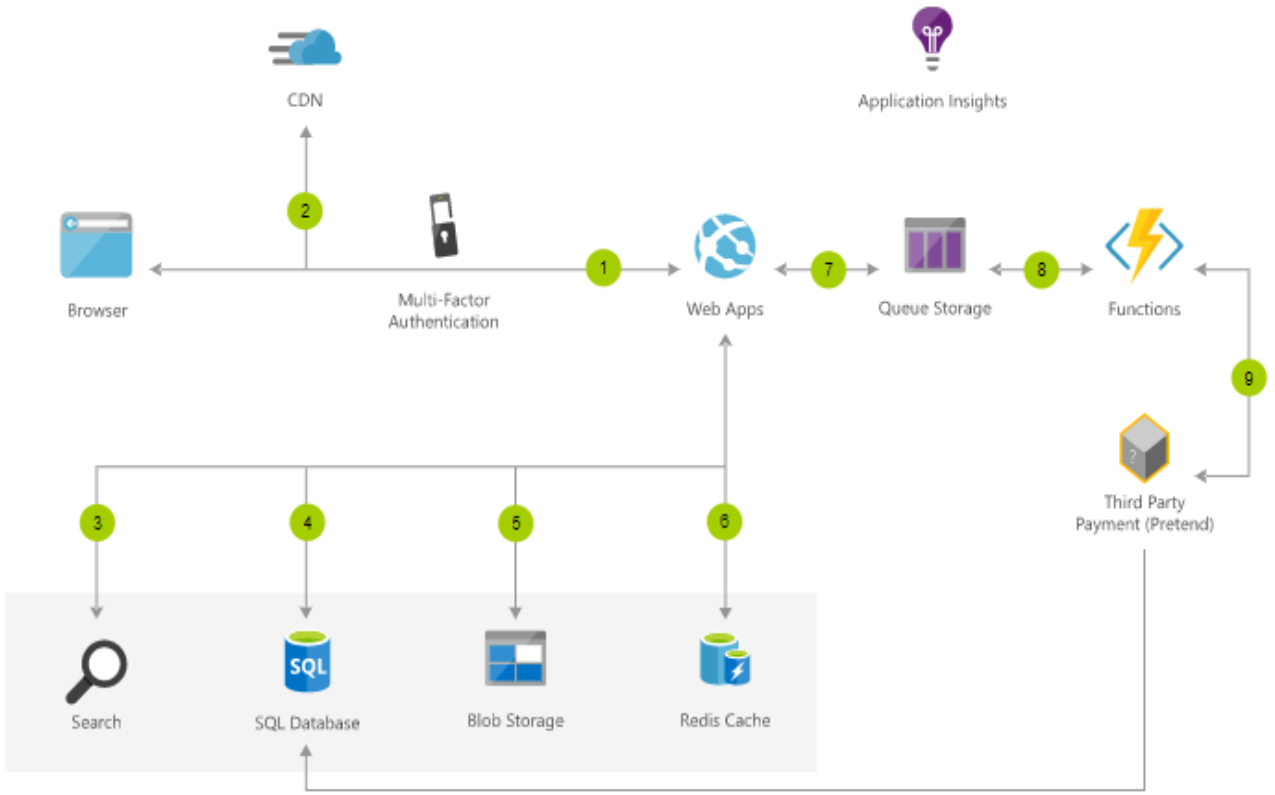
5 Web app pulls product images from Blob Storage.

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

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

8 Azure Functions processes order payment.

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



Source : MS Serverless

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Review Questions

Question

Scalability vs. Maintainability Trade-off

- How would you architect a web application capable of handling millions of concurrent users while maintaining code simplicity and ease of development? Discuss specific approaches and their limitations.

Microservices Architecture

Approach-1

- ***Approach**:* Decompose the application into smaller, loosely coupled services that can be developed, deployed, and scaled independently. Each service should focus on a specific functionality (e.g., user authentication, data processing, front-end rendering).
- ***Limitations**:* Microservices introduce complexities in terms of deployment, communication, and data consistency between services. Managing a large number of services can also increase operational overhead.

Containerization

Approach-2

- **Approach**: Use containerization technologies like Docker to package each microservice along with its dependencies into a lightweight, portable container. Containers can be easily deployed and managed across different environments.
- **Limitations**: Container orchestration adds another layer of complexity, especially when dealing with large-scale deployments. It requires expertise in tools like Kubernetes for managing containers effectively.

Asynchronous Communication

Approach-3

- **Approach**: Utilize asynchronous communication patterns such as message queues or event-driven architectures to decouple components and handle bursts of traffic more efficiently.
- **Limitations**: Implementing asynchronous communication adds complexity to the system architecture and can introduce challenges in ensuring message reliability and consistency.

Caching and Content Delivery Networks (CDNs)

Approach-4

- ***Approach**:* Cache frequently accessed data and use CDNs to deliver static assets closer to users, reducing latency and offloading traffic from the origin server.
- ***Limitations**:* Caching strategies need to be carefully designed to ensure data consistency and prevent stale data. CDNs may incur additional costs and may not be effective for dynamic content.

Other Approaches

Horizontal Scaling and Performance Optimization

- ***Approach**:* Design the application to scale horizontally by adding more instances of services to handle increased load. Utilize auto-scaling mechanisms to dynamically adjust resources based on demand.
- ***Limitations**:* Horizontal scaling requires stateless services and shared-nothing architecture, which may not be feasible for all components of the application. Managing a large number of instances adds complexity in monitoring, orchestration, and resource allocation.

Performance Optimization

- ***Approach**:* Identify and optimize performance bottlenecks through techniques such as code profiling, database indexing, and resource utilization optimization.
- ***Limitations**:* Performance optimization requires continuous monitoring and tuning, which adds overhead to development and maintenance processes. It may also involve trade-offs in terms of code simplicity and development speed.

Question-2

Serverless vs. Containerized Architectures

Compare and contrast serverless and containerized architectures for web applications.

When would you choose one over the other?

Discuss the trade-offs in terms of performance, cost, developer experience, and maintainability.

Answer

Serverless

- **Concept:** You write code (functions) triggered by events (HTTP requests, database changes). The cloud provider manages servers and scales them automatically.
- **Pros:**
  - **Highly scalable:** Auto-scaling handles traffic spikes without manual intervention.
  - **Cost-effective:** Pay only for resources used, minimizing idle server costs.
  - **Fast development:** Focus on code, not infrastructure management.
- **Cons:**
  - **Vendor lock-in:** Tied to a specific cloud provider's platform.
  - **Limited control:** Restricted customization options for complex applications.
  - **Cold starts:** Initial function invocation might have higher latency.

Answer (cont.)

Containers

- **Concept:** Package your application with dependencies in a container (Docker image). Containers run on servers you manage (on-prem or cloud).
- **Pros:**
  - **Flexible and portable:** Runs on any platform supporting Docker.
  - **Fine-grained control:** Customize environments and resources as needed.
  - **Better suited for stateful applications:** Easier to manage persistent data.
- **Cons:**
  - **Manual scaling:** Requires proactive infrastructure management.
  - **Potentially higher cost:** Pay for reserved server resources even during low traffic.
  - **Steeper learning curve:** Requires container orchestration knowledge.

Answer (cont.)

When to use?

- **Performance:**
  - For consistent, low-latency workloads, containers might have an edge.
  - For event-driven, bursty workloads, serverless excels with auto-scaling.
- **Cost:**
  - Serverless shines for unpredictable traffic, paying only for used resources.
  - Containers might be cheaper for predictable, sustained workloads.
- **Developer Experience:**
  - Serverless simplifies development with event-driven triggers and managed infrastructure.
  - Containers offer more control and flexibility, but require deeper infrastructure knowledge.
- **Maintainability:**
  - Serverless offers automatic scaling and patching, reducing maintenance overhead.
  - Containers require managing container images, orchestration, and server infrastructure.

Comparison

Aspect	Monolithic Architecture	Microservices Architecture	Serverless Architecture	Containerization and Orchestration
Description	Entire application is developed, deployed, and managed as a single unit.	Decomposes the application into smaller, independently deployable services.	Abstracts infrastructure management, focusing on writing code for individual functions.	Packages applications into lightweight, portable containers and manages their lifecycle.
Advantages	• Simple development and deployment. • Easy to understand and debug.	• Improved scalability and flexibility. • Faster development and deployment cycles • Enhanced fault isolation.	• Reduced operational overhead. • Pay-per-use pricing model. • Built-in scalability and fault tolerance.	• Consistent environment across environments. • Efficient resource utilization and scaling. • Support for microservices architectures.
Disadvantages	• Lack of scalability. • Limited flexibility. • Increased risk of downtime.	• Increased complexity. • Higher operational overhead. • Challenges in ensuring data consistency.	• Limited control. • Potential vendor lock-in. • Challenges in debugging and monitoring.	• Complexity in setting up and managing. • Learning curve. • Additional overhead.

Comparing Modern Application Landscape

Deployment Models

- **On-premise:** Traditional model with applications running on in-house hardware and software. Offers full control but high maintenance costs.
- **Cloud:** Applications run on remote servers managed by cloud providers like AWS, Azure, or GCP. Scalable, flexible, and pay-as-you-go pricing.
- **Hybrid:** Combines on-premise and cloud deployments for diverse application needs.

Comparing Modern Application Landscape

Architecture Styles

- **Monolithic:** Single, large application codebase. Simple to develop initially but less scalable and agile with growing complexity.
- **Microservices:** Application broken down into small, independent services that communicate through APIs. Highly scalable, agile, and resilient to failures.
- **Serverless:** Application functions triggered by events, managed by the cloud provider. Highly scalable and cost-effective but limited control and vendor lock-in.

Comparing Modern Application Landscape

Development Approaches

- **Waterfall:** Linear, sequential development stages with rigid planning. Predictable but slow and inflexible to changes.
- **Agile:** Iterative and incremental development with continuous feedback loops. Adaptable to changes but requires strong collaboration and planning.
- **DevOps:** Integrates development, operations, and security teams for faster deployment and smoother operations. Enhances collaboration and efficiency.

Comparing Modern Application Landscape

Technology Stack:

- **Programming Languages:** Diverse options like Python, Java, Javascript, Go, etc., each with strengths and weaknesses in performance, ease of use, and community support.
- **Databases:** Relational (MySQL, PostgreSQL) vs. NoSQL (MongoDB, Cassandra) options, each suited for different data types and access patterns.
- **Tools and Frameworks:** Numerous options for development, testing, deployment, and automation, impacting developer experience and workflow efficiency.