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

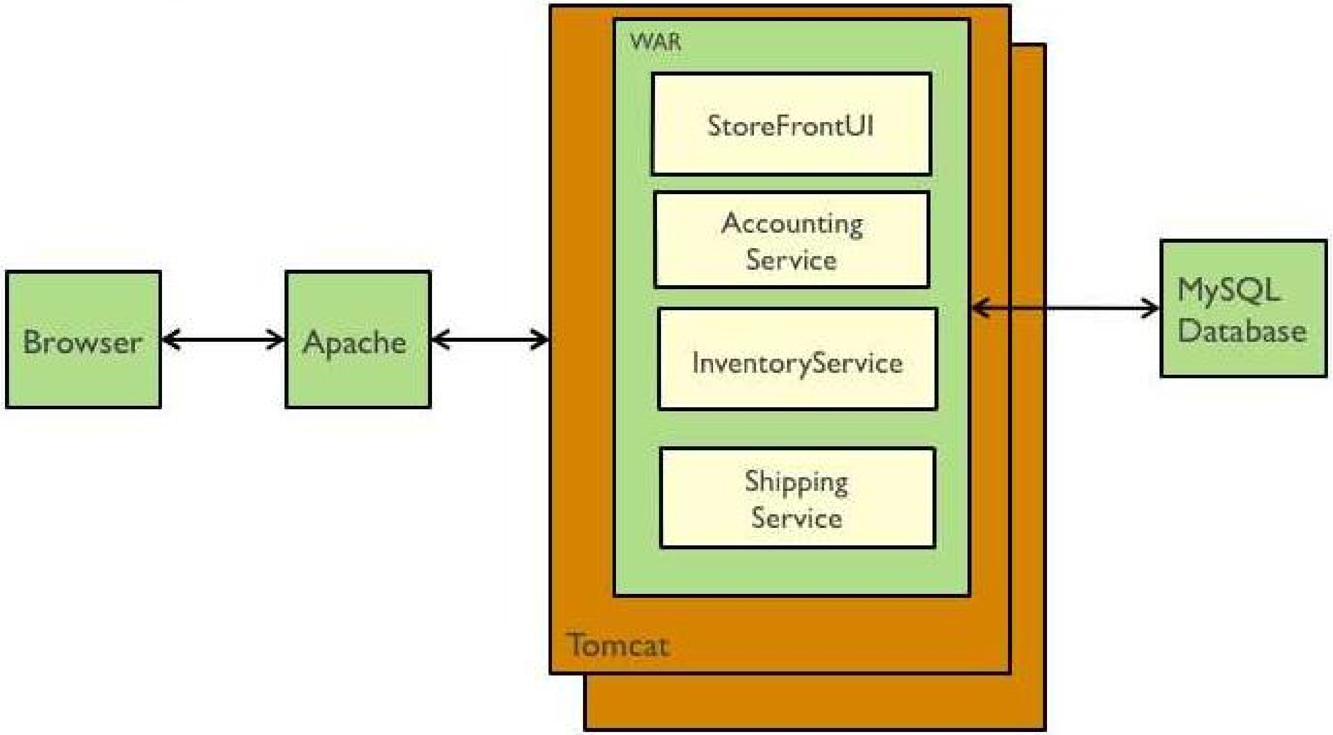
What’s the application’s deployment architecture?

- 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

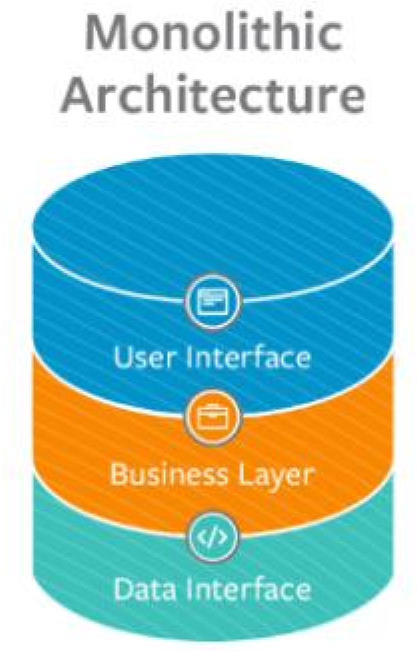
- Three types
  - ✓ Single process monolith
  - ✓ Distributed monolith
  - ✓ Third party black-box systems

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



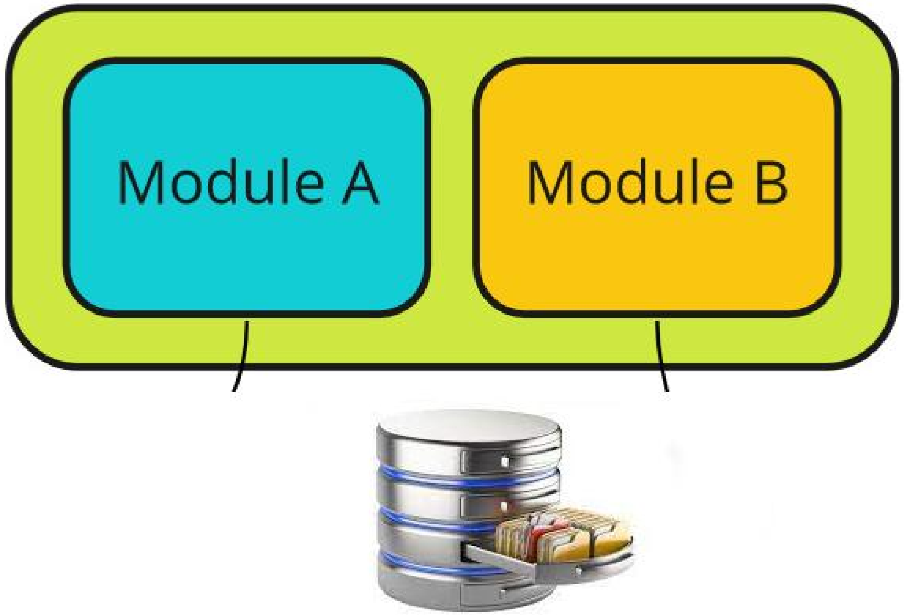
source

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

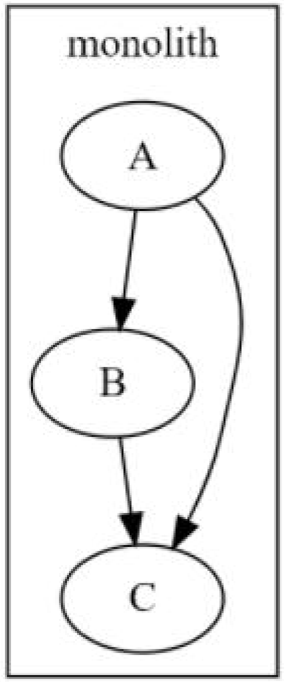


Distributed monolith

- 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

source

- Integration is the key here!

Source :
Monolith to Microservices by Sam Newman

Thank You!

In our next session:

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!

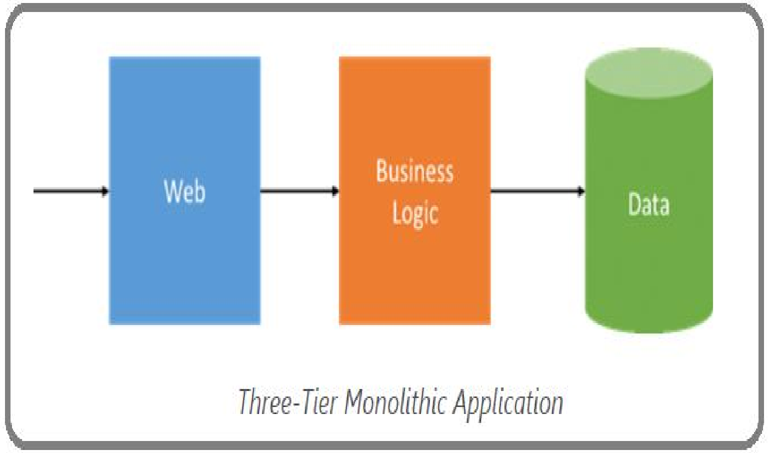
Monolithic application models

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

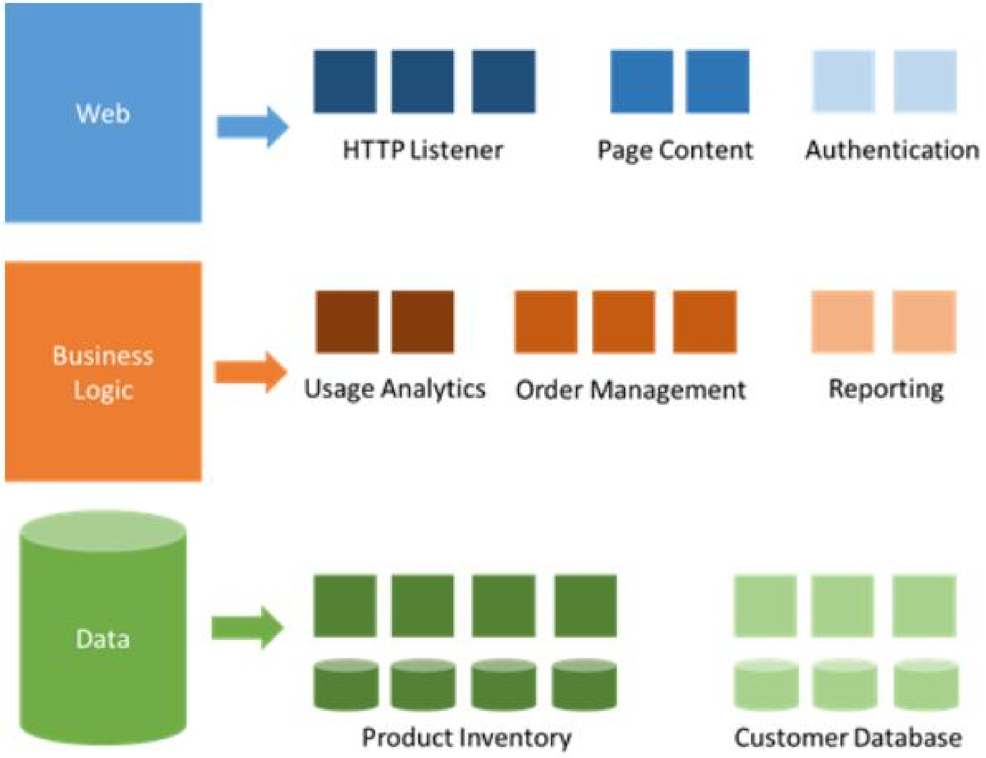
- Challenges
  - A tight coupling between unrelated application services within tiers
  - A change to any application service, even small ones, required its entire tier to be retested and redeployed
  - A simple update could have unforeseen effects on the rest of the tier - changes are risky
  - Lengthening 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



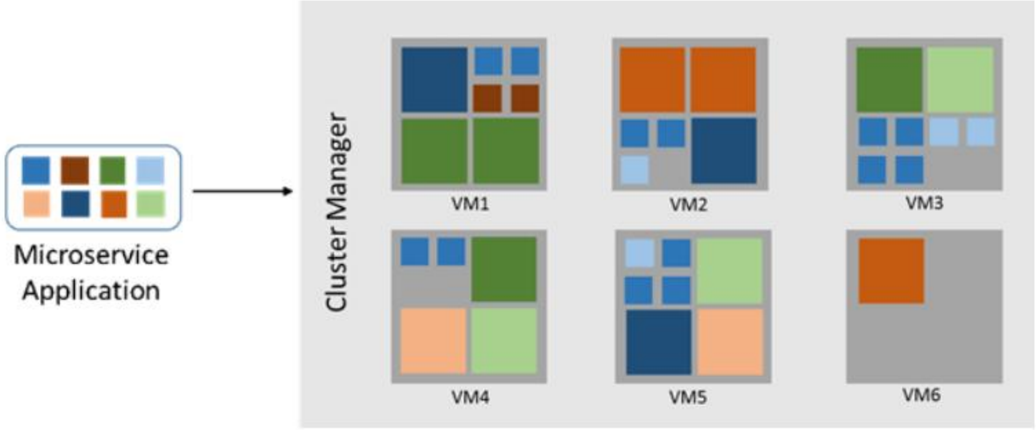
Breaking the Monolith into Microservices

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:

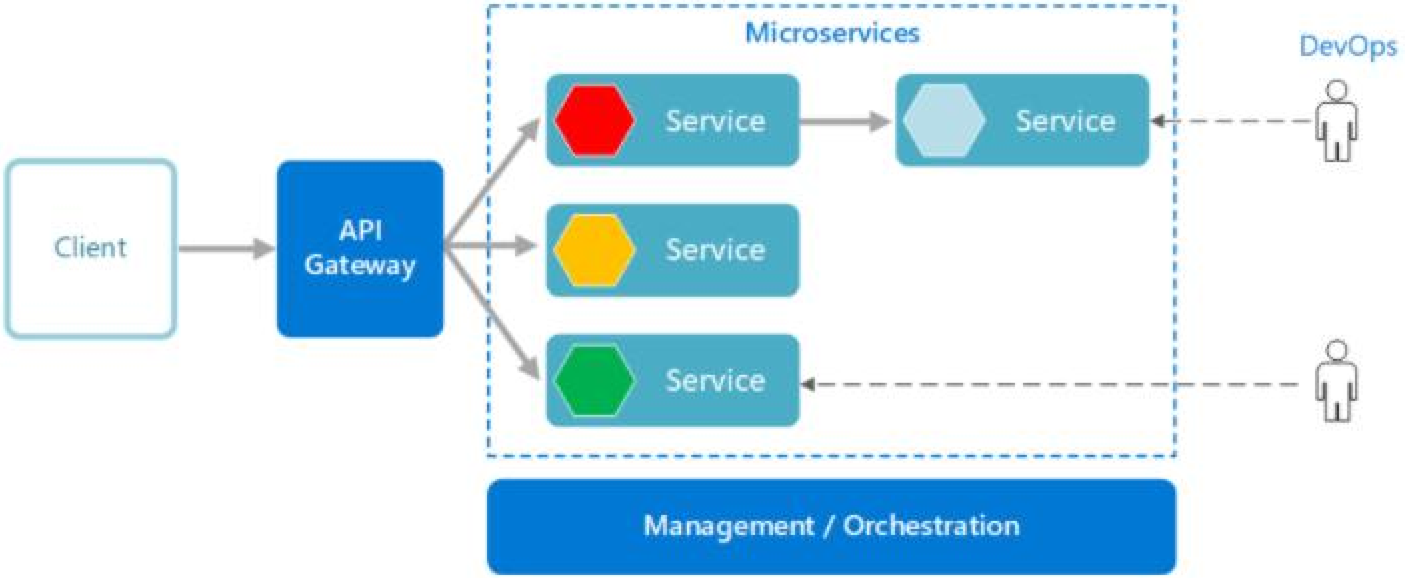
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

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:

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!

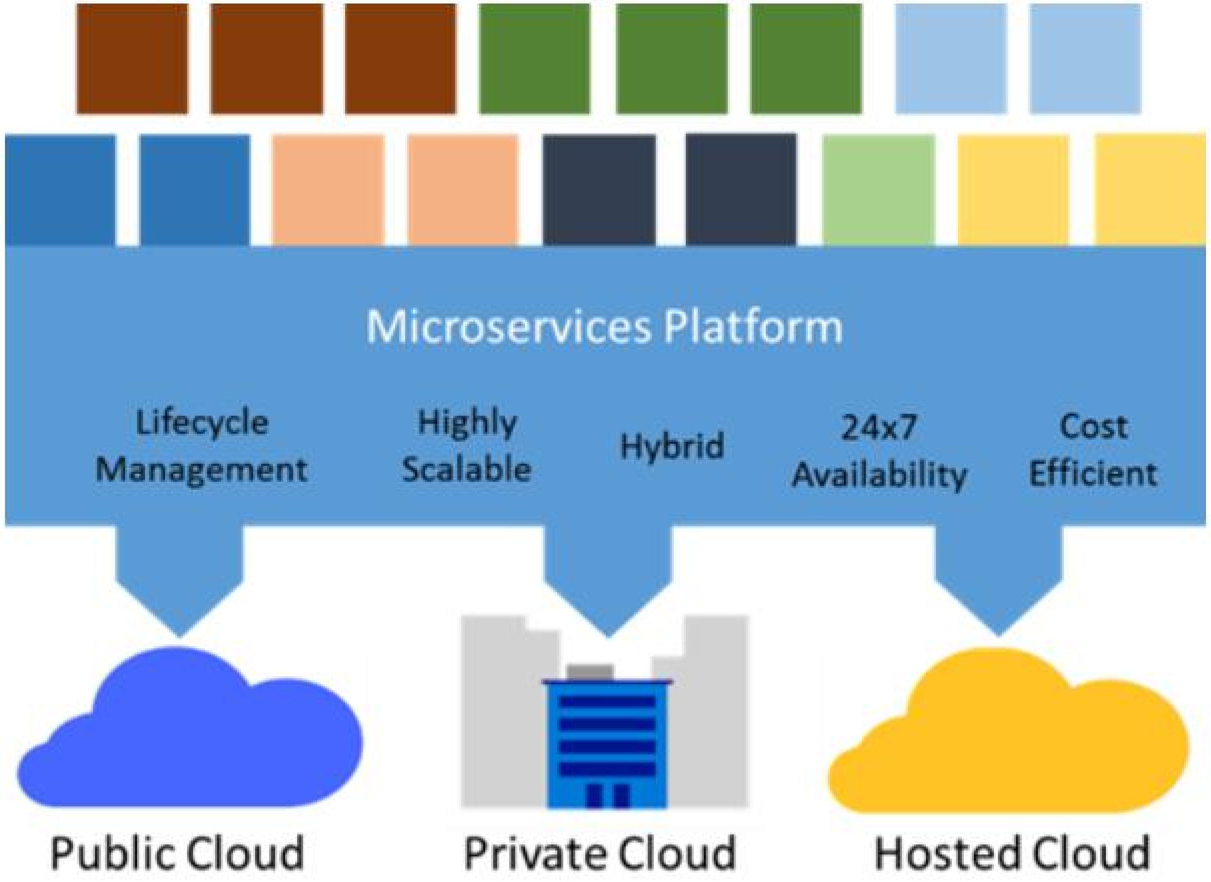
In our next session:

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Microservice Application Platforms

Chandan Ravandur N

Microservice application platforms



Microservices Platform

Source : Microsoft

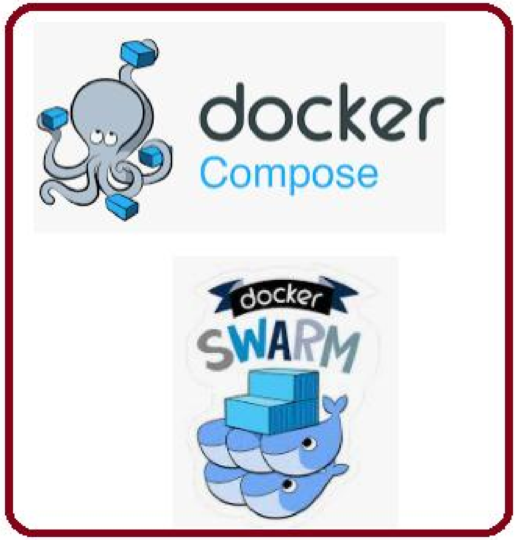
Microservice application platforms

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

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

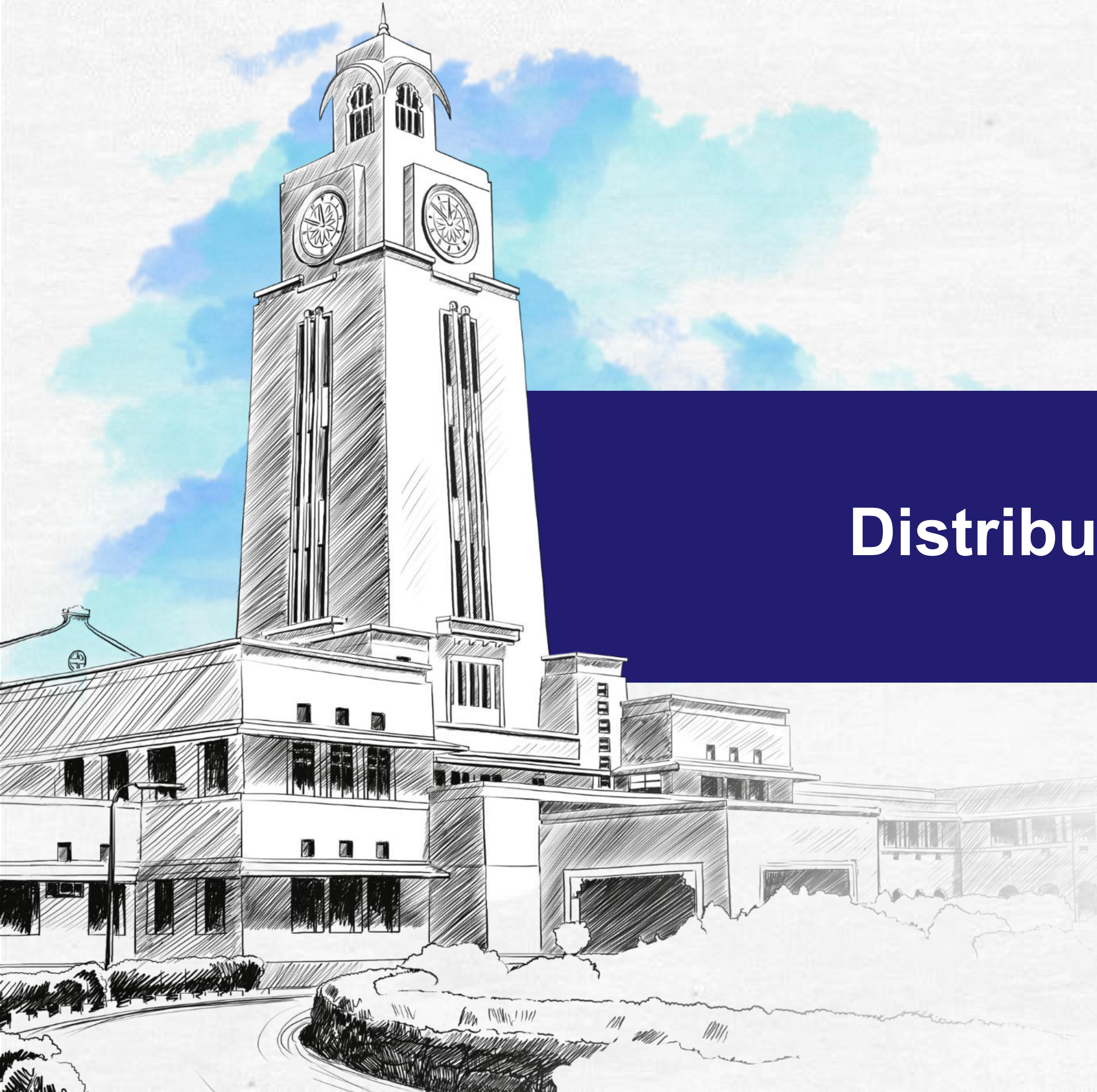
- Backed by Red Hat
- Platform-as-a-service offering that leverages Docker container-based packaging for
  - deploying container orchestration and compute management capabilities for Kubernetes
  - enabling users to run containerized JBoss Middleware
  - multiple programming languages, databases and 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

RED HAT OPENSHIFT

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!

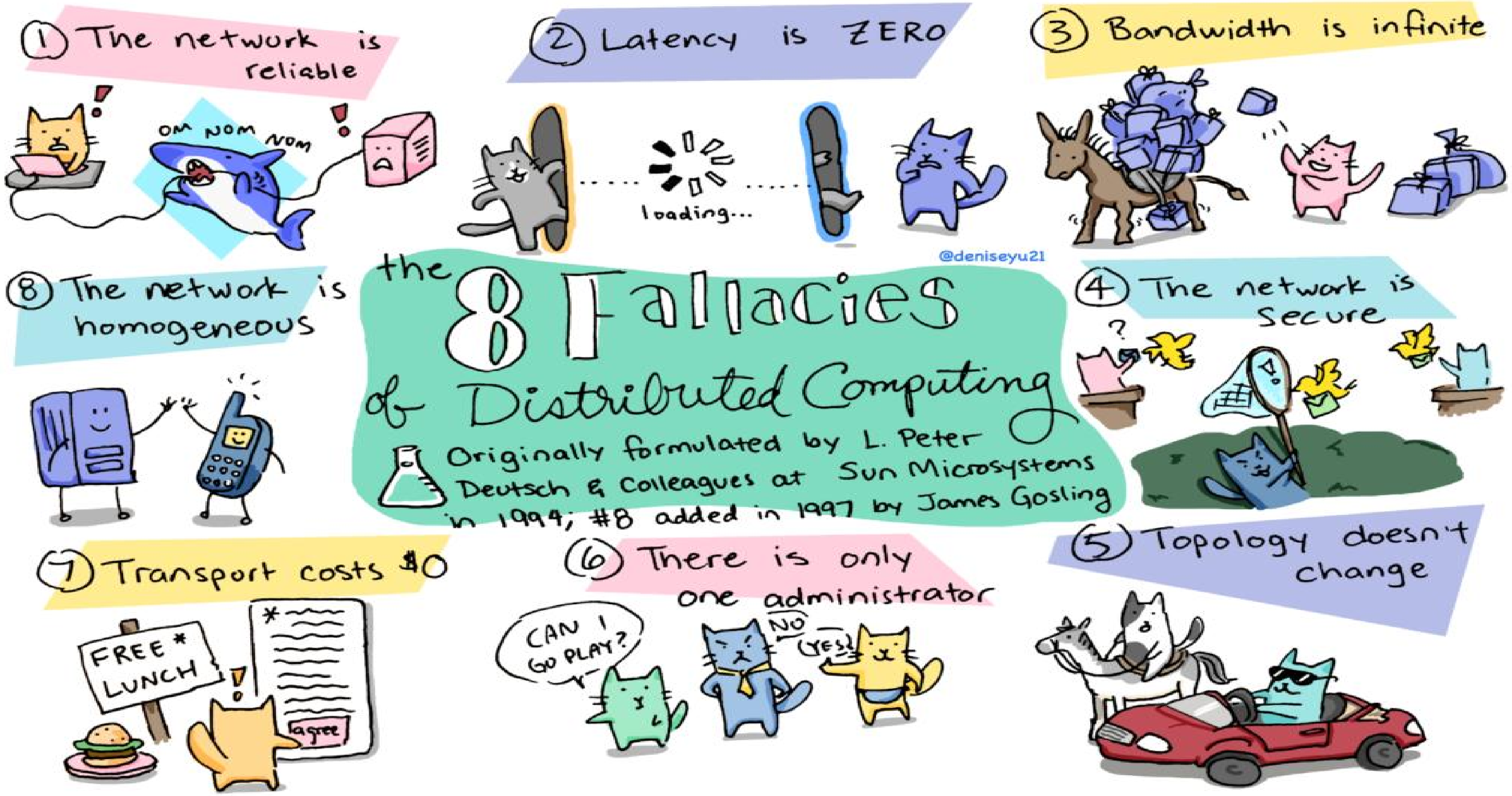
- Distributed system is system in which individual computers are connected through a network and appear as single computer to the outside world.
  - ✓ Provides power across machines to accomplish scalability, reliability and economics

- 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

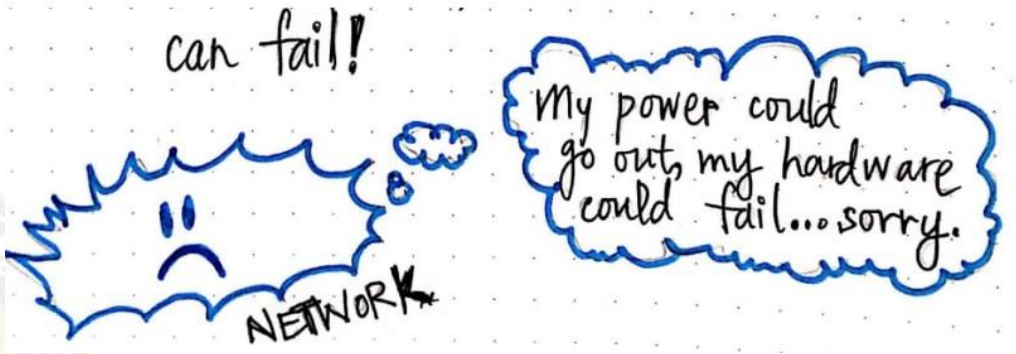


① The network is reliable
OM NOM NOM
② Latency is ZERO
loading...
③ Bandwidth is infinite
@deniseyu21
the 8 Fallacies of Distributed Computing
Originally formulated by L. Peter Deutsch & Colleagues at Sun Microsystems in 1994; #8 added in 1997 by James Gosling
⑧ The network is homogeneous
④ The network is Secure
?
⑦ Transport costs $0
FREE* LUNCH
*
agree
⑥ There is only one administrator
CAN I GO PLAY?
NO
YES!
⑤ Topology doesn't change

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.

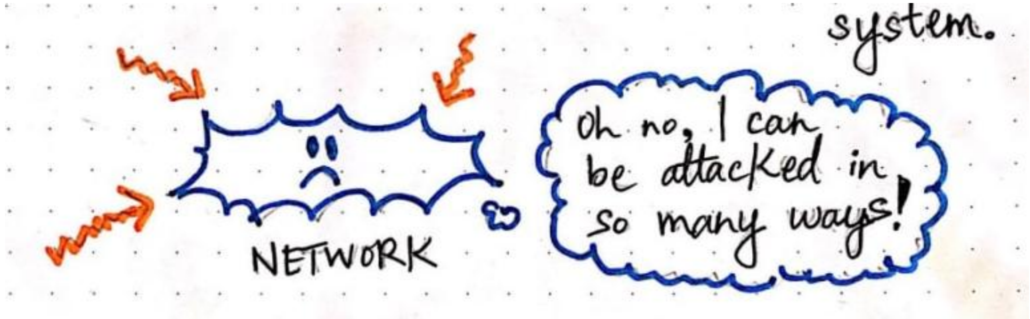
500 Mbps
NETWORK
200 ms
bandwidth, the capacity of data being sent
latency, the speed at which data gets to destination

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

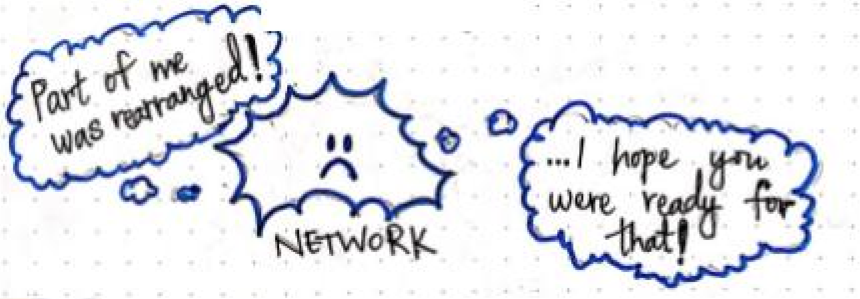
My capacity might be growing, but you are also giving me larger data to send!
NETWORK

- 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

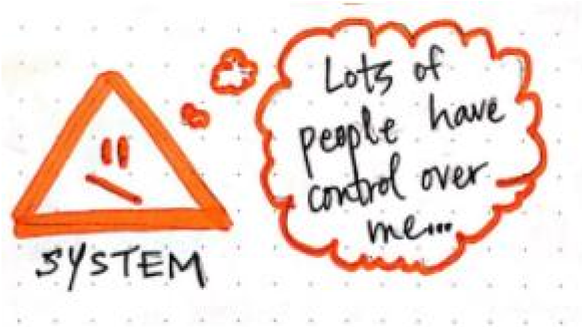


Fallacies(3)

- 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

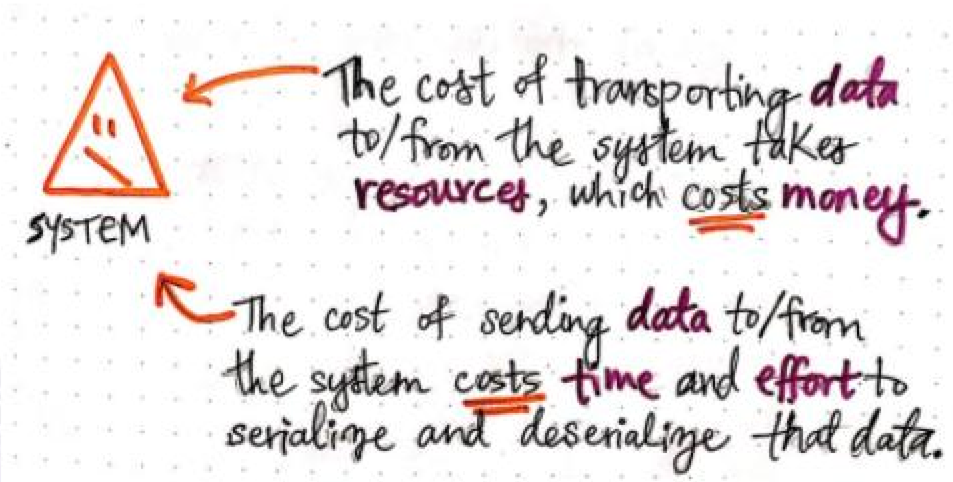

Part of me was rearranged!
NETWORK
...I hope you were ready for that!

- 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

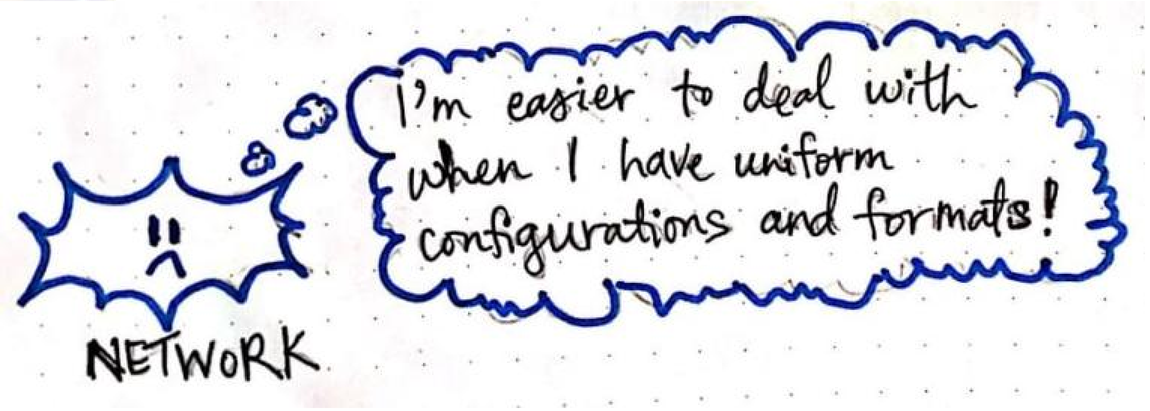


Fallacies(4)

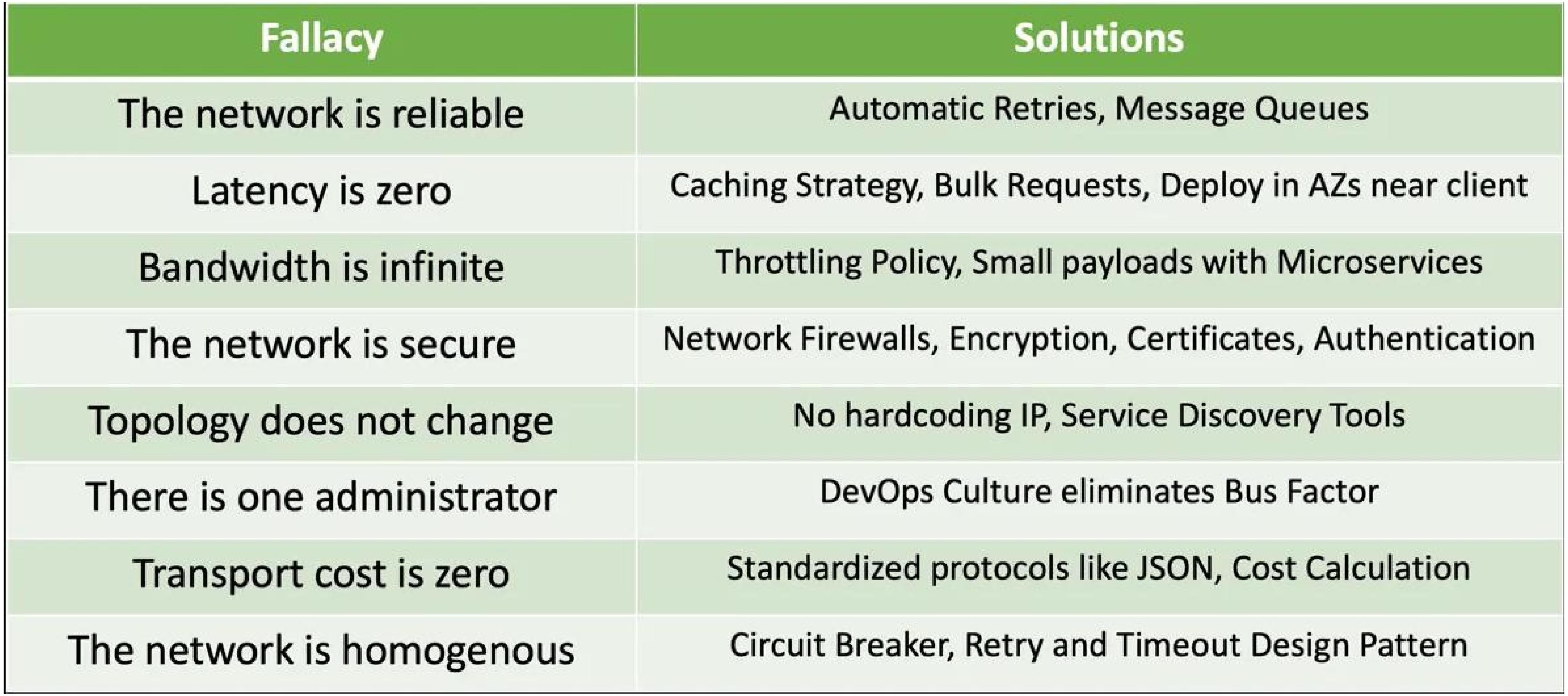
- Fallacy #7: Transport cost is zero
  - ✓ Need to do with the overhead of serializing data into the network
  - ✓ Other has to do with the costs of the networking infrastructure
  - ✓ Both of these are realities that cannot be avoided


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.

- 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 predict
    - ❖ the devices that will connect
    - ❖ the protocols used to connect
    - ❖ the operating systems, browsers, etc


I'm easier to deal with when I have uniform configurations and formats!
NETWORK

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

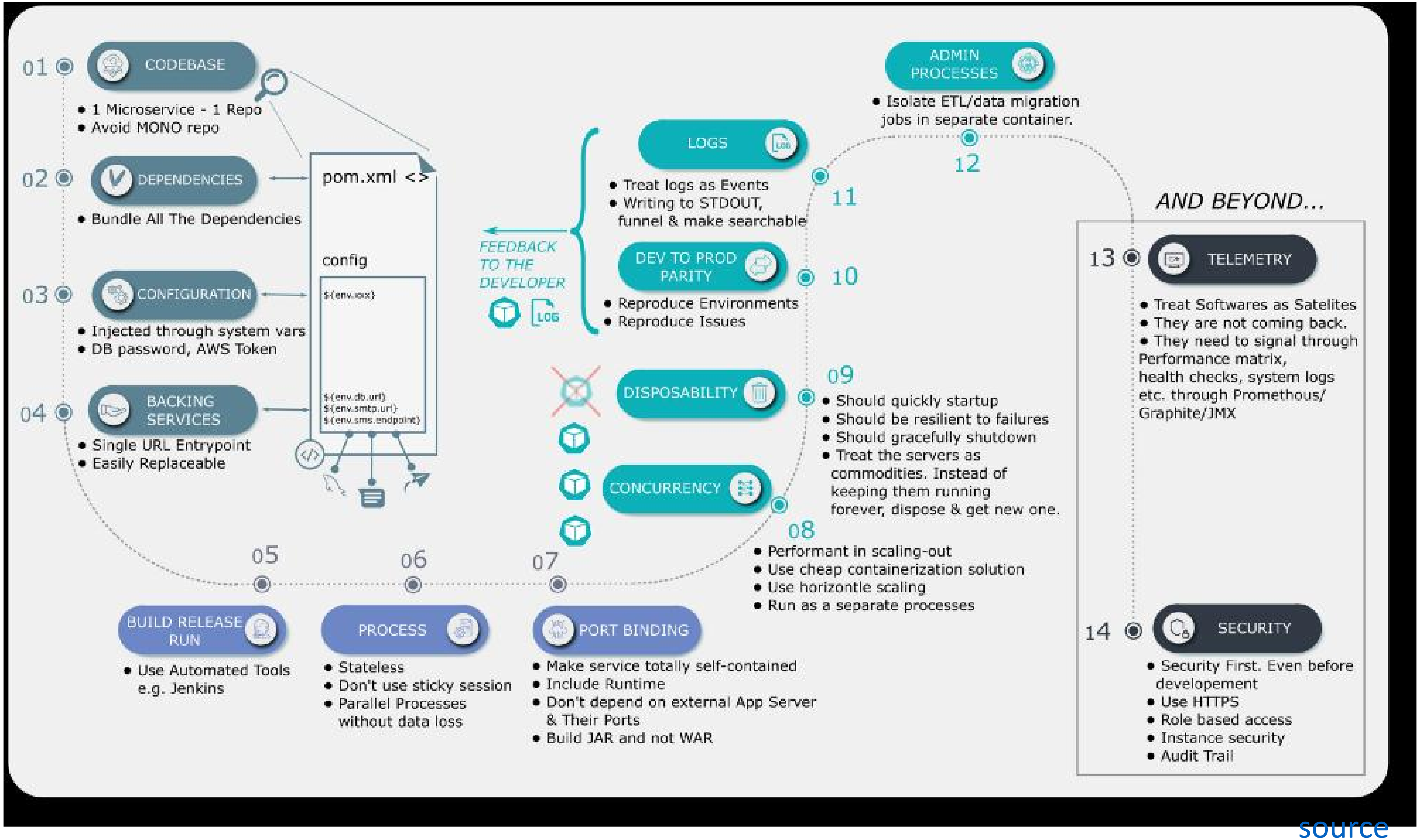
Chandan Ravandur N

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



01 CODEBASE
- 1 Microservice - 1 Repo
- Avoid MONO repo

02 DEPENDENCIES
- Bundle All The Dependencies

03 CONFIGURATION
- Injected through system vars
- DB password, AWS Token

04 BACKING SERVICES
- Single URL Entrypoint
- Easily Replaceable

05 BUILD RELEASE RUN
- Use Automated Tools e.g. Jenkins

06 PROCESS
- Stateless
- Don't use sticky session
- Parallel Processes without data loss

07 PORT BINDING
- Make service totally self-contained
- Include Runtime
- Don't depend on external App Server & Their Ports
- Build JAR and not WAR

08 CONCURRENCY
- Performant in scaling-out
- Use cheap containerization solution
- Use horizontle scaling
- Run as a separate processes

09 DISPOSABILITY
- Should quickly startup
- Should be resilient to failures
- Should gracefully shutdown
- Treat the servers as commodities. Instead of keeping them running forever, dispose & get new one.

10 DEV TO PROD PARITY
- Reproduce Environments
- Reproduce Issues

11 LOGS
- Treat logs as Events
- Writing to STDOUT, funnel & make searchable

12 ADMIN PROCESSES
- Isolate ETL/data migration jobs in separate container.

AND BEYOND...

13 TELEMETRY
- Treat Softwares as Satelites
- They are not coming back.
- They need to signal through Performance matrix, health checks, system logs etc. through Promethous/ Graphite/JMX

14 SECURITY
- Security First. Even before developement
- Use HTTPS
- Role based access
- Instance security
- Audit Trail

source

Thank You!

In our next session:

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Cloud Native Architecture

Chandan Ravandur N

Introducing cloud-native software

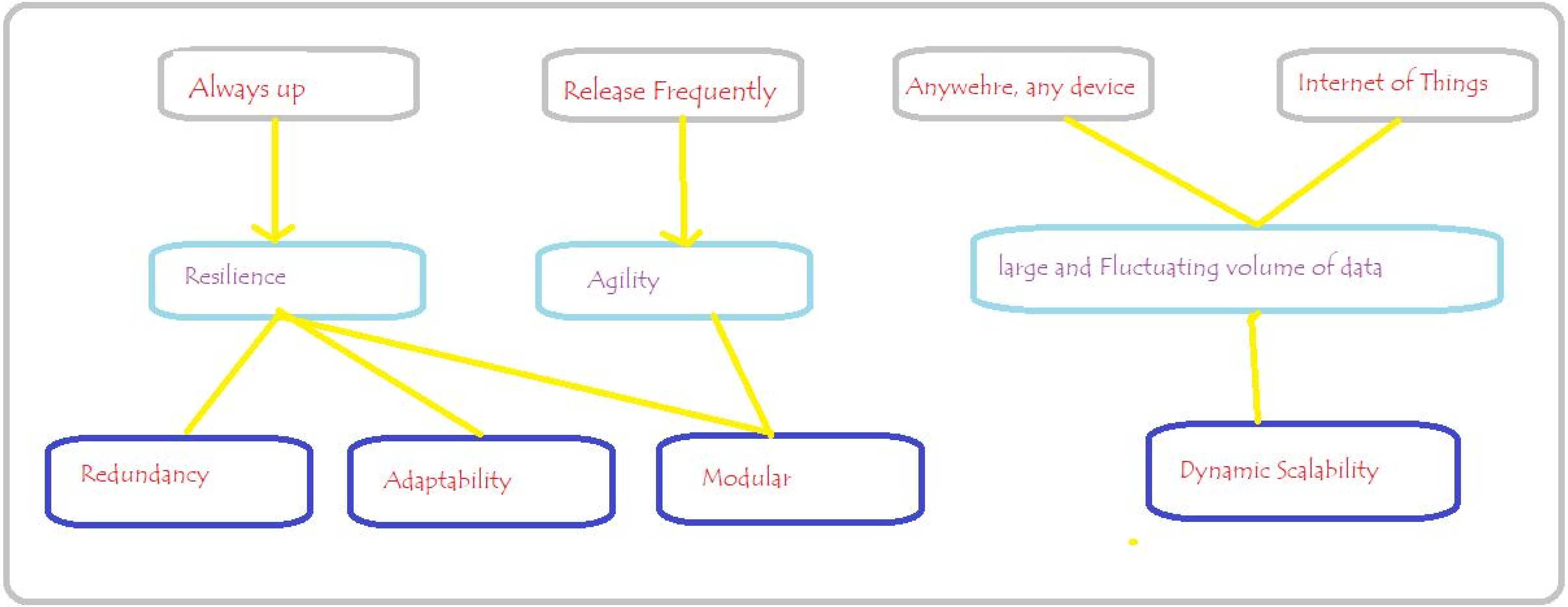
New Edge Applications Demands

- 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!

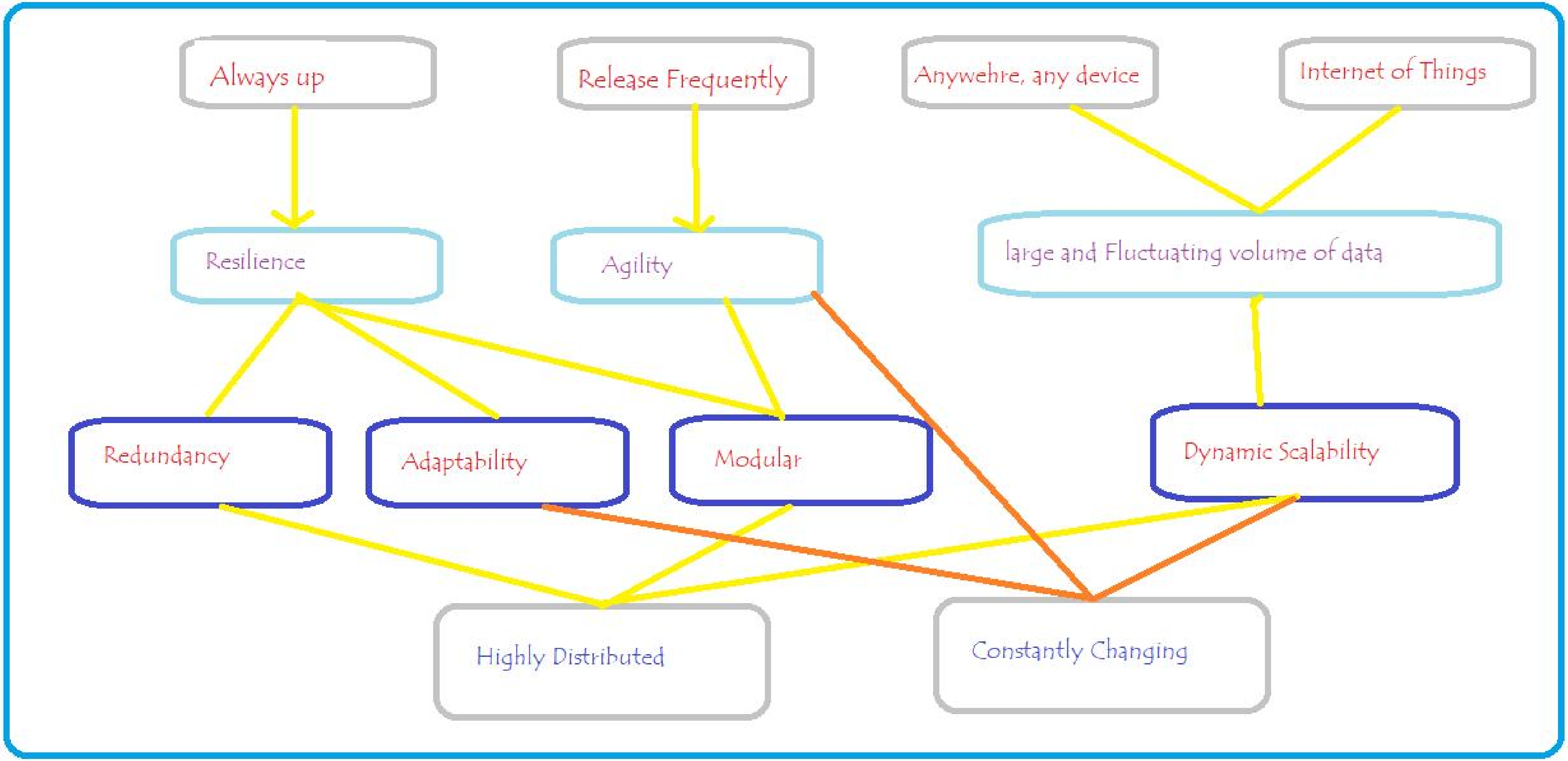
User requirements for software

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:

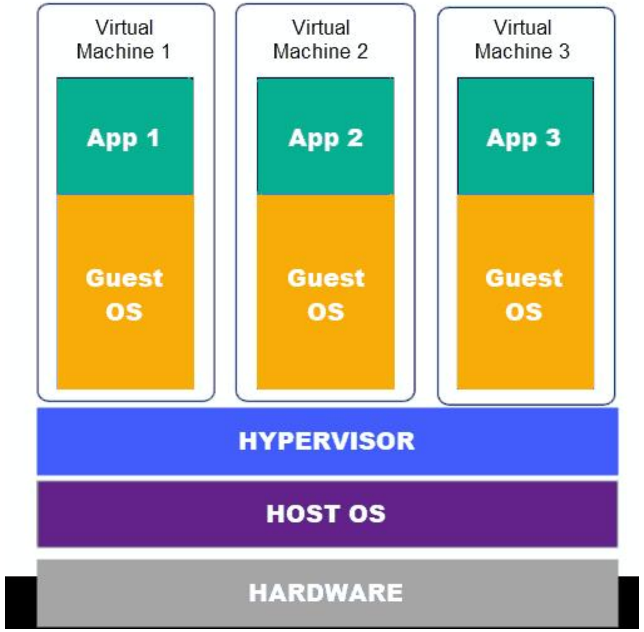
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

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

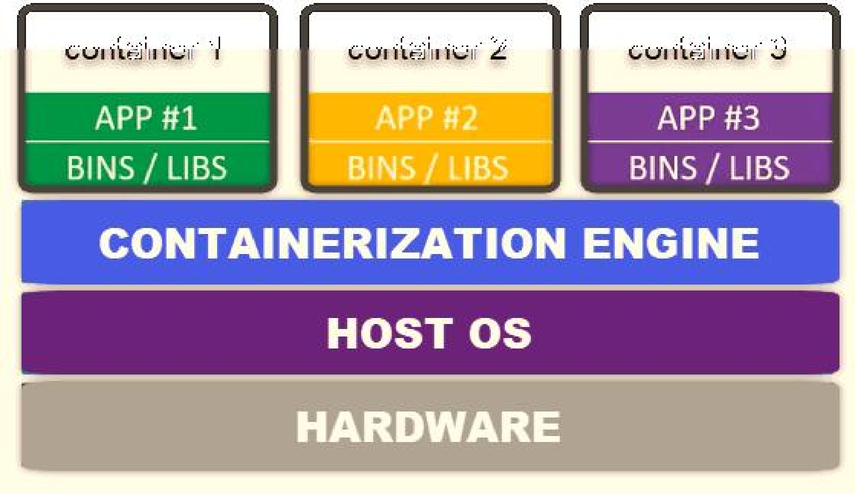
Containers

- 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

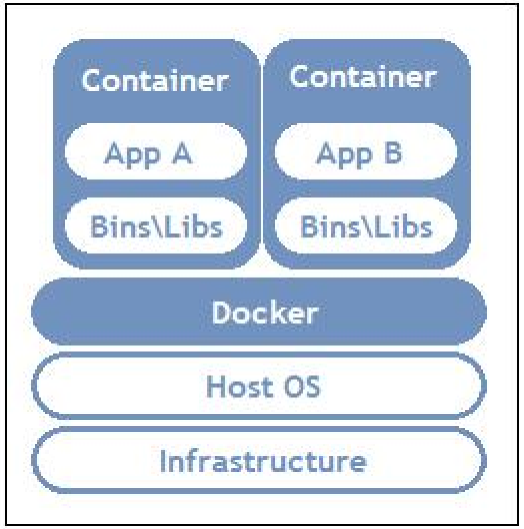


Containers(2)

- 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.

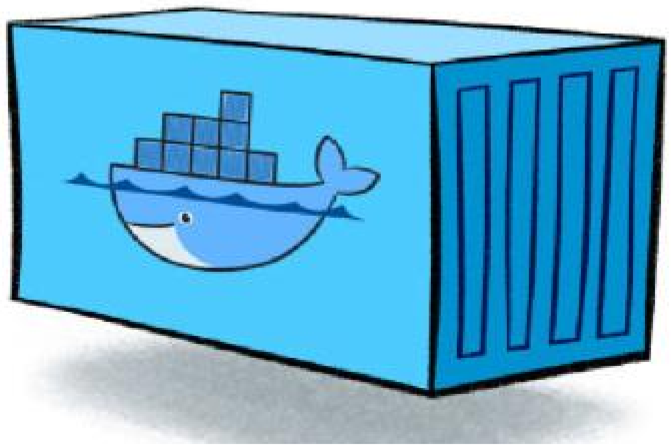
Containers in Cloud native

- 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



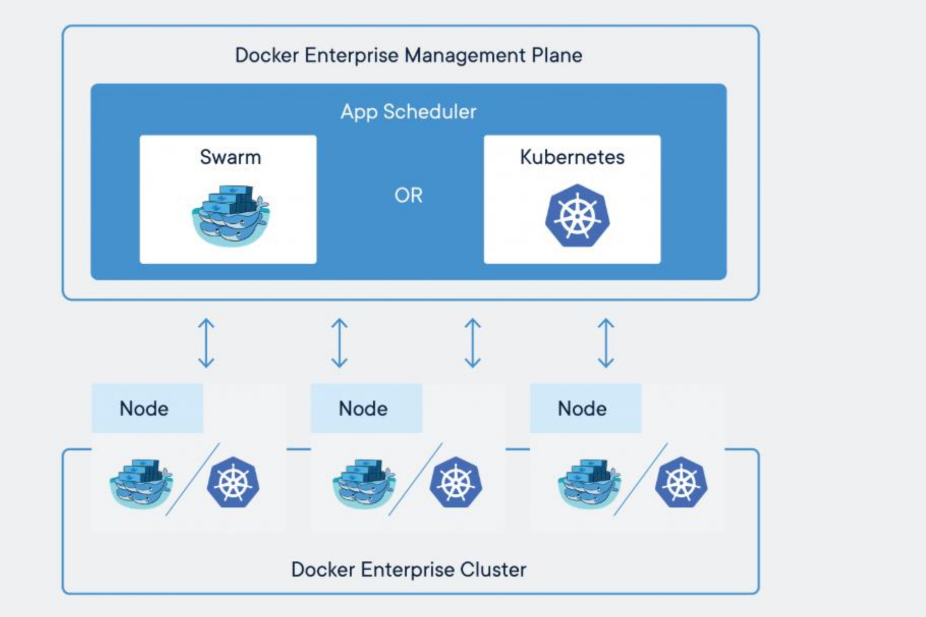
source

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

RANCHER
docker
SWARM
Nomad
MESOS
kubernetes

source

Thank You!

In our next session:

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

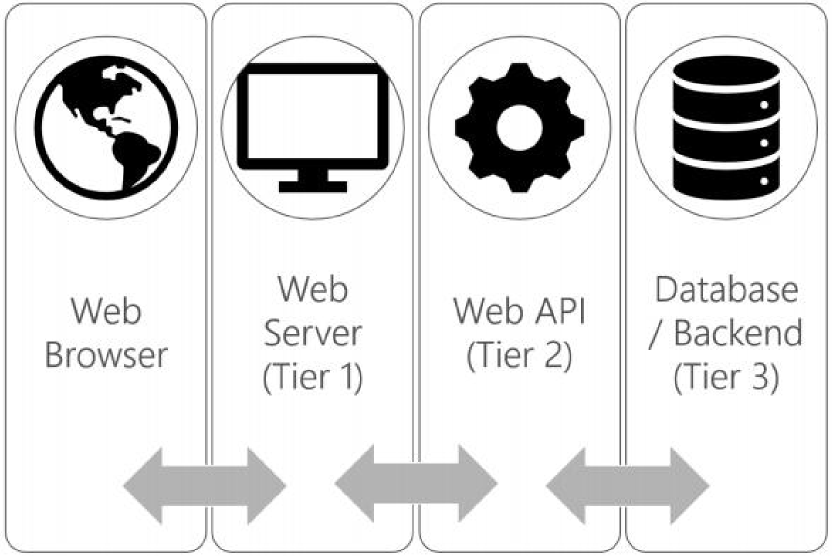
N-Tier applications

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

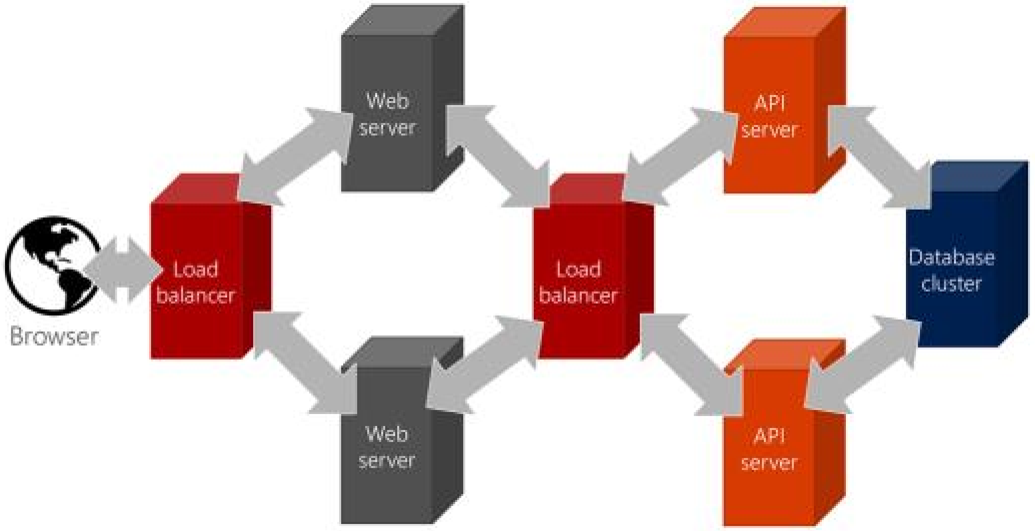
On-premises

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

source:FileCloud

- 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.

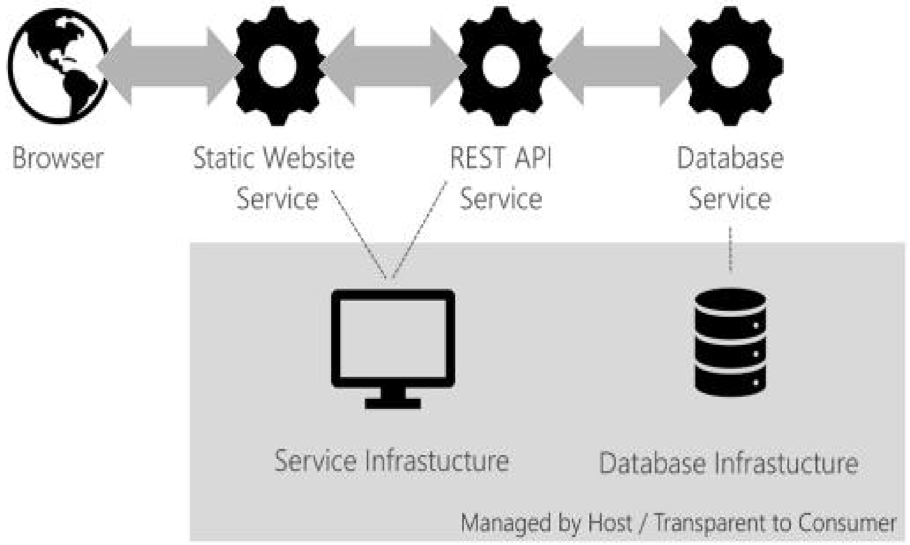
Platform as a Service (PaaS)

- 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

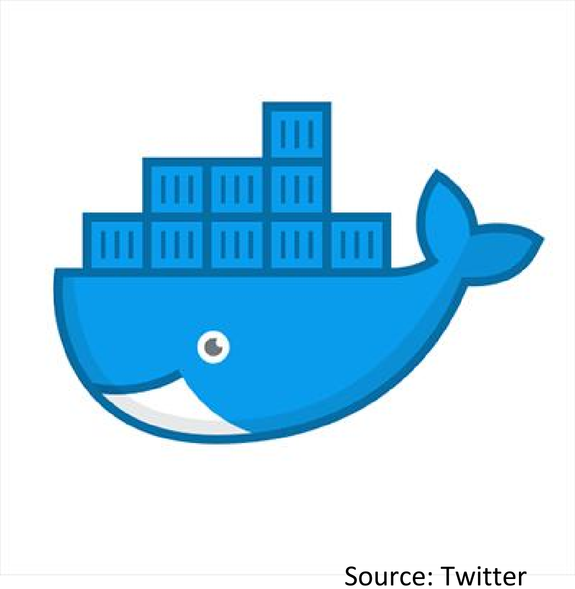
Chandan Ravandur N

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



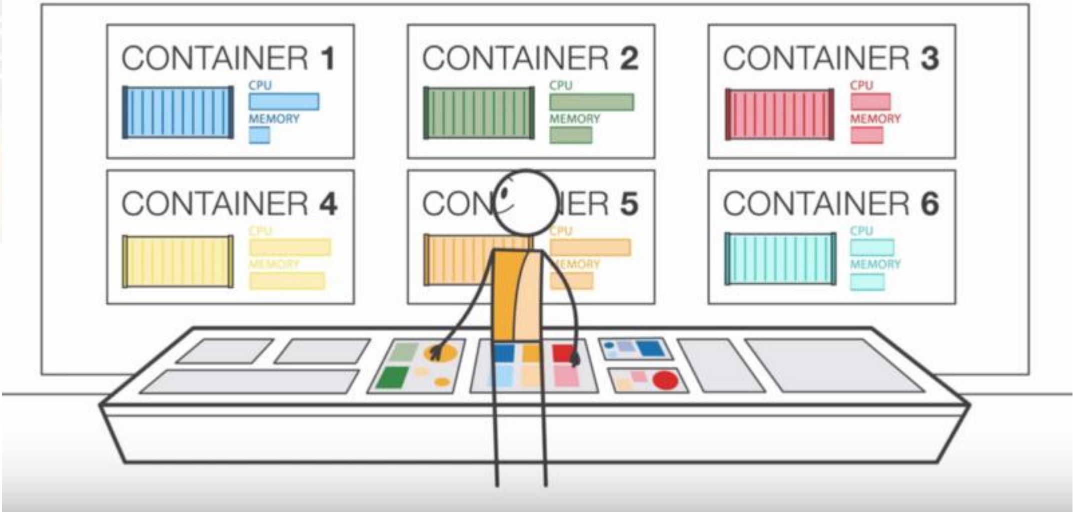
Source: Twitter

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



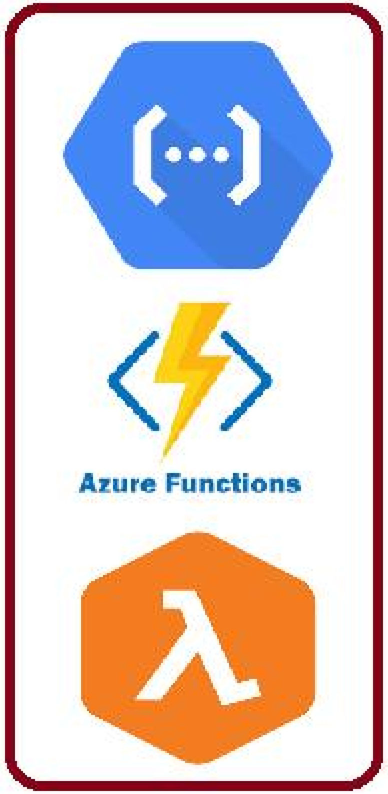
source:freecoecamp

Functions as a Service (FaaS)

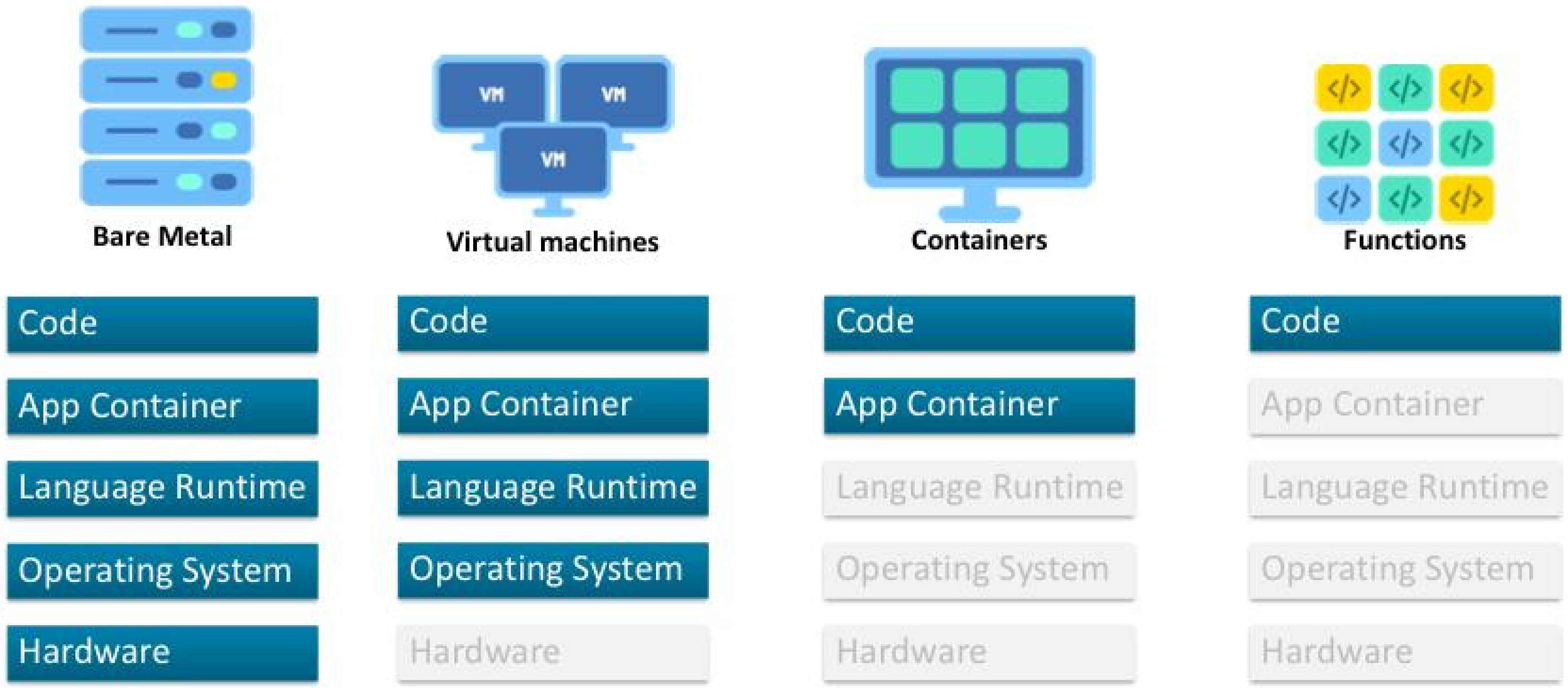
- A category of cloud computing services that provides a platform allowing customers to develop, run, and manage application functionalities
  - ✓ without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app

- 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



Bare Metal	Virtual machines	Containers	Functions
Code	Code	Code	Code
App Container	App Container	App Container	App Container
Language Runtime	Language Runtime	Language Runtime	Language Runtime
Operating System	Operating System	Operating System	Operating System
Hardware	Hardware	Hardware	Hardware

source : Oracle blog

Reference:
Serverless apps Architecture patterns and Azure implementation
By Microsoft

Thank You!

In our next session:

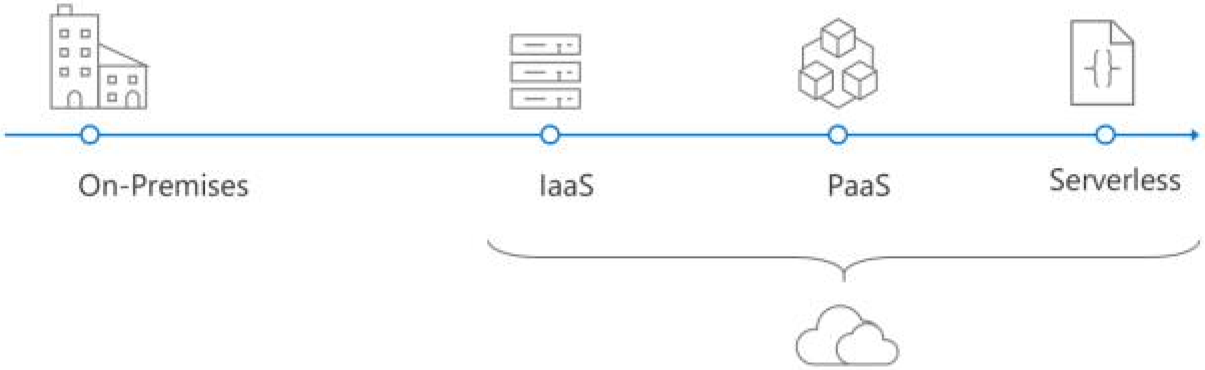
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

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!

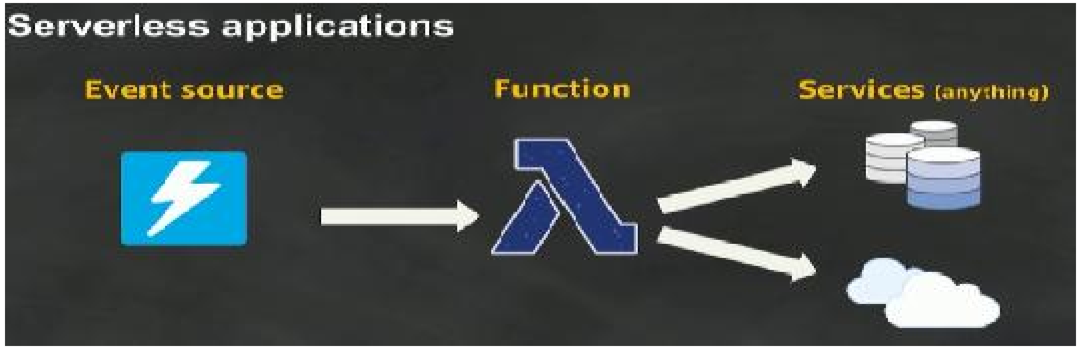
Serverless

- 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

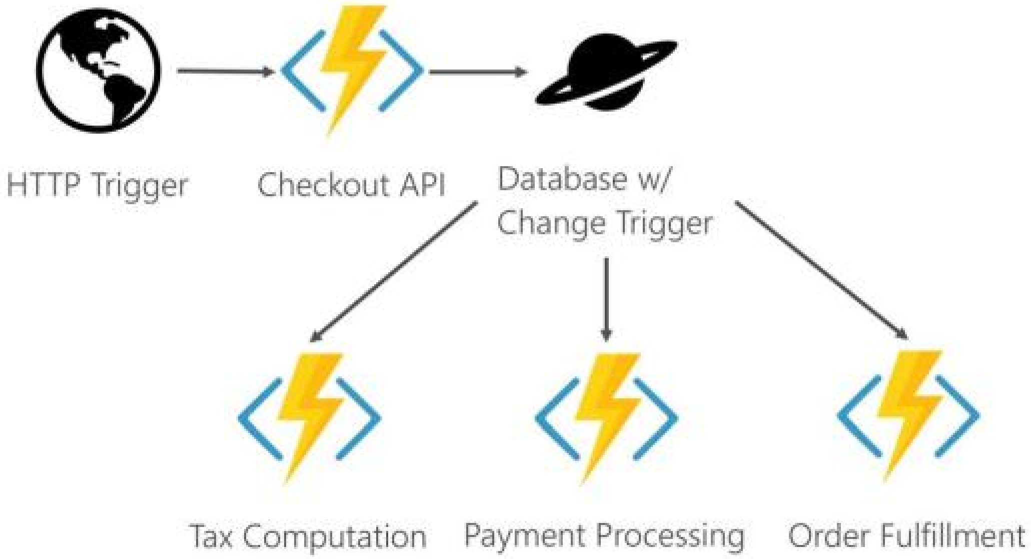
- It can be thought of as less server!

source : AWS



Serverless components

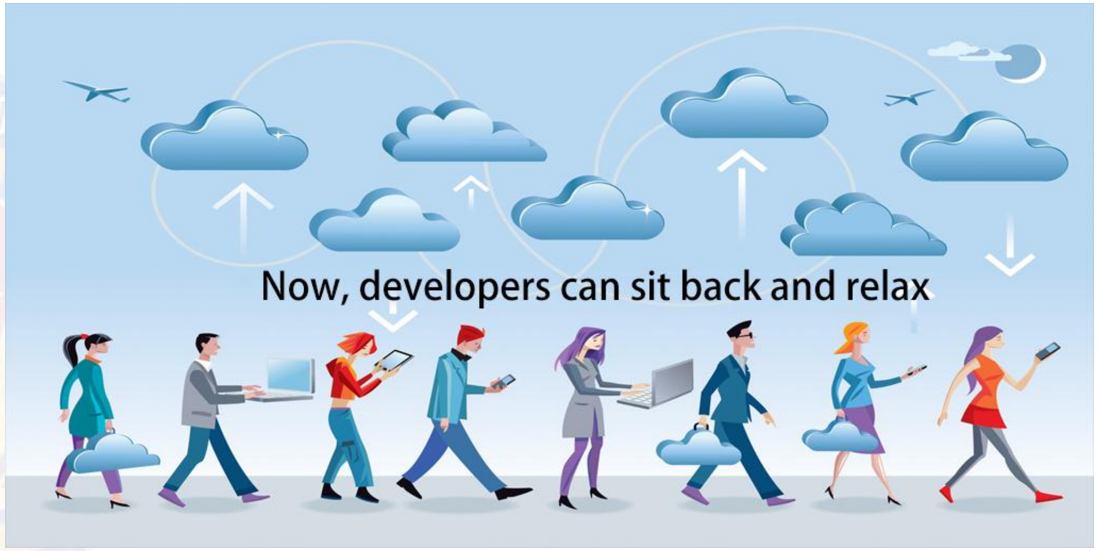
- An HTTP request causes the Checkout API code to run
- The Checkout API inserts code into a database
- The insert triggers several other functions to run to perform tasks like computing tasks and fulfilling the order.



source : Microsoft

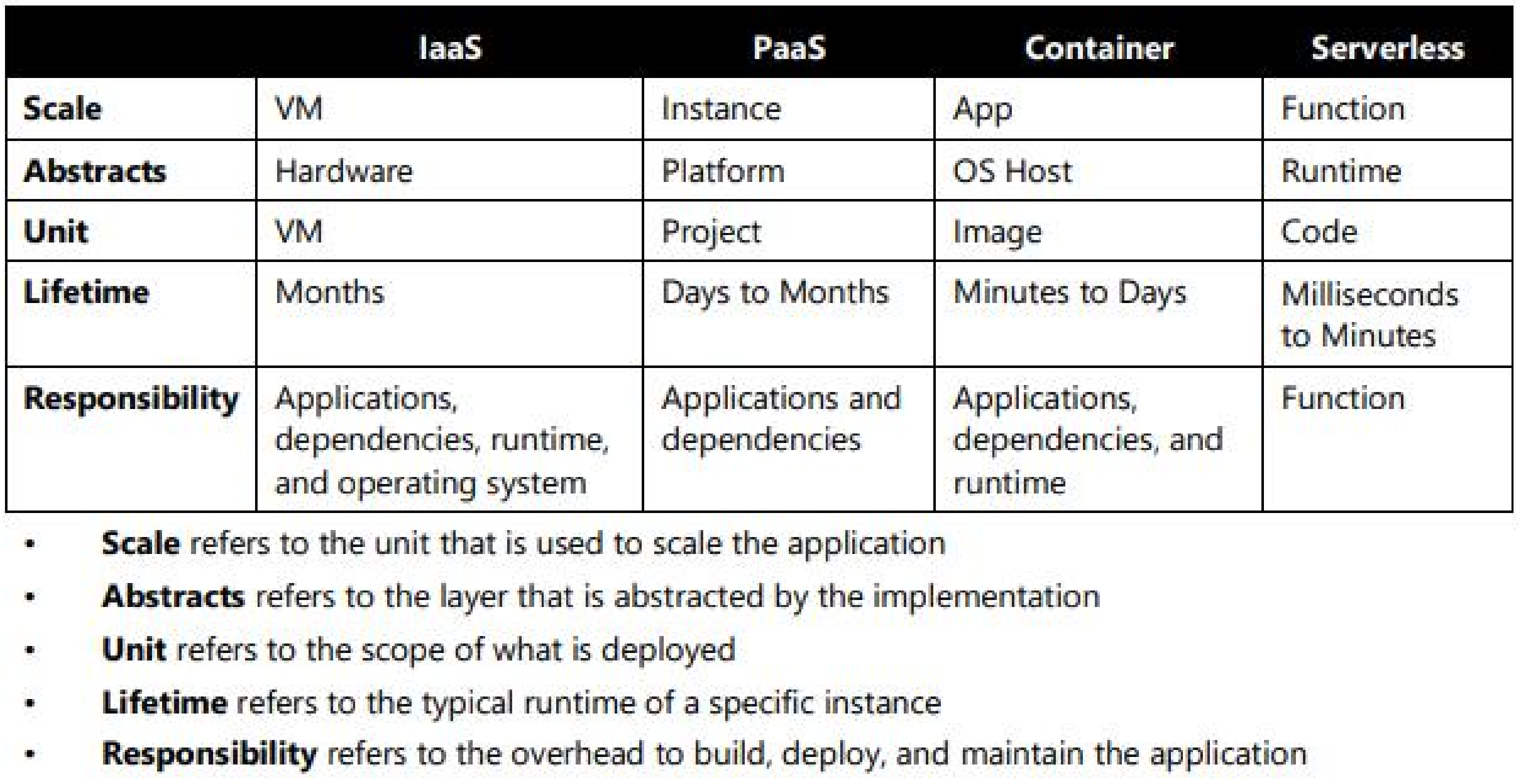
Advantages of Serverless

- 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

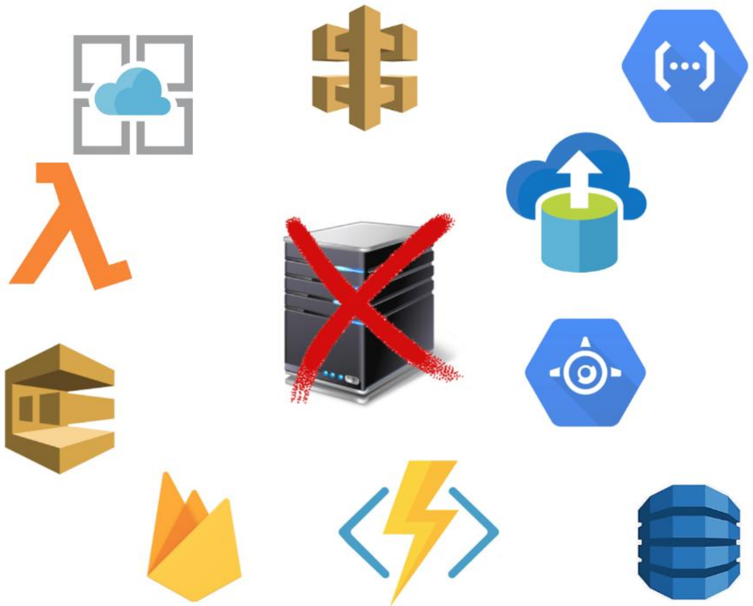
BITS Pilani

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

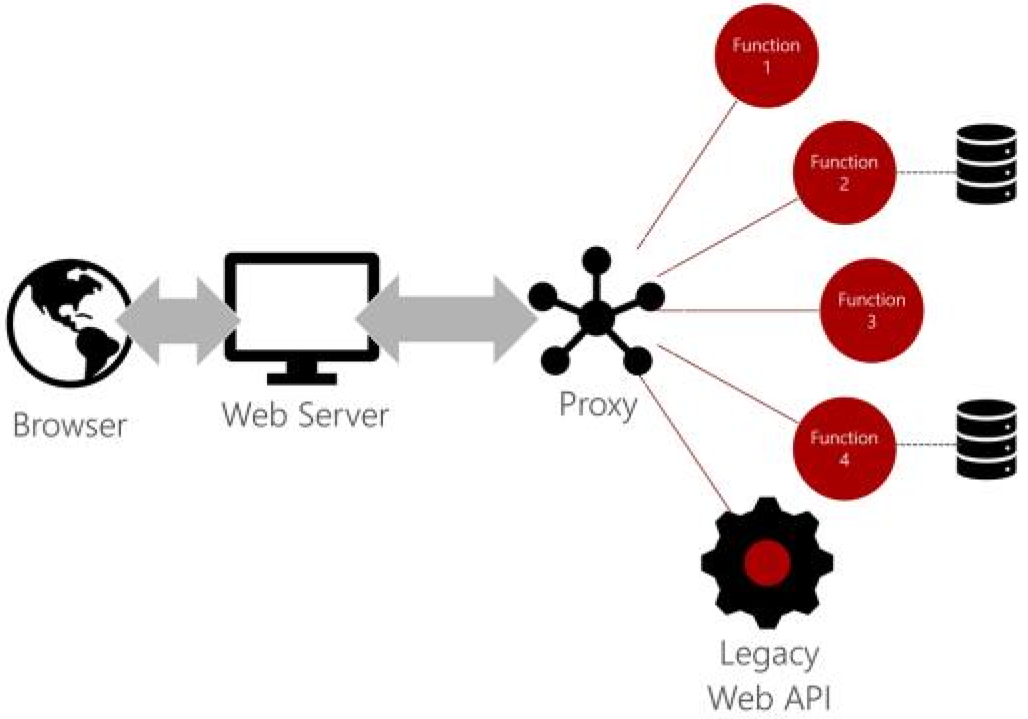
backendless

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) unti 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

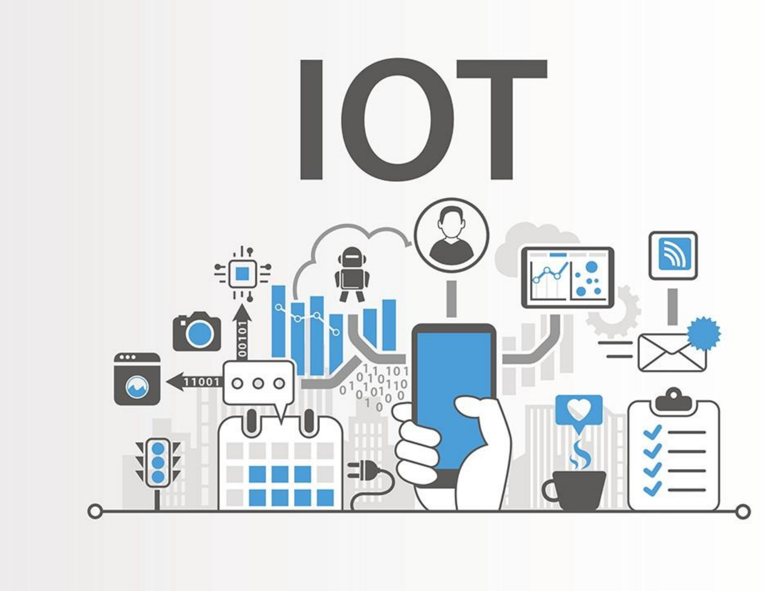
- For example,
  - ✓ A mobile developer who builds apps using a JavaScript framework can build serverless functions with JavaScript as well
  - ✓ The serverless host manages the operating system, a Node.js instance to host the code, package dependencies, and more
  - ✓ The developer is provided a simple 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

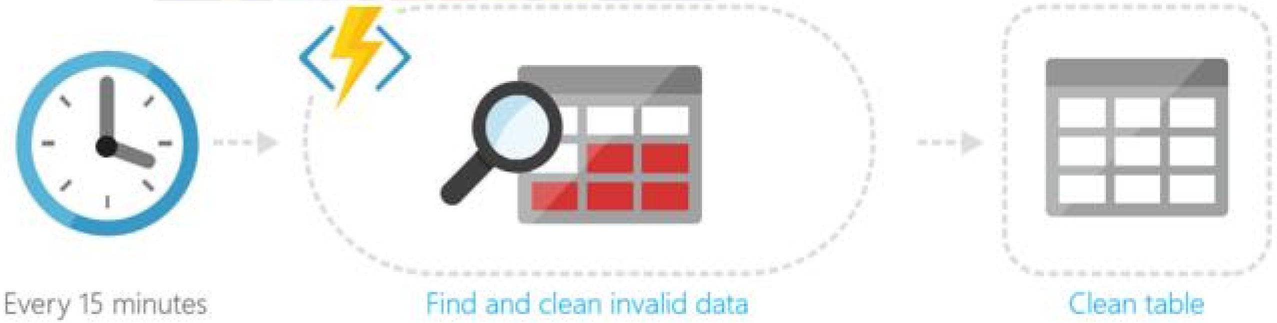
- Many design patterns exists for Serverless
- Commonality is the fundamental combination of an event trigger and business logic

- 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)

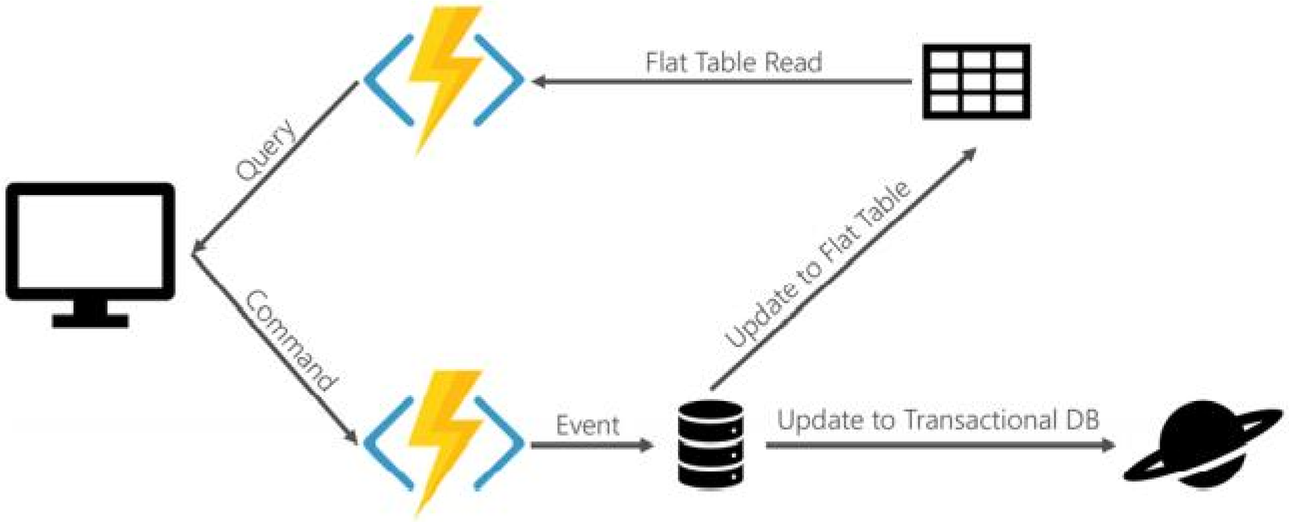
- A pattern that provides different interfaces for reading (or querying) data and operations that modify data

- 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

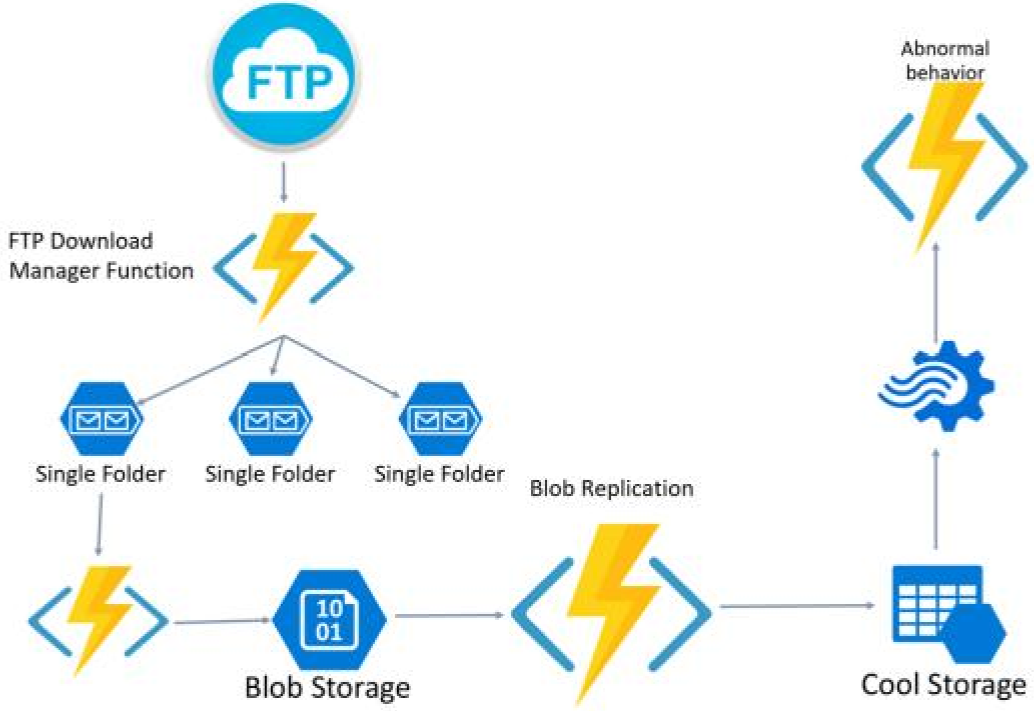
Event-based processing

- 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.



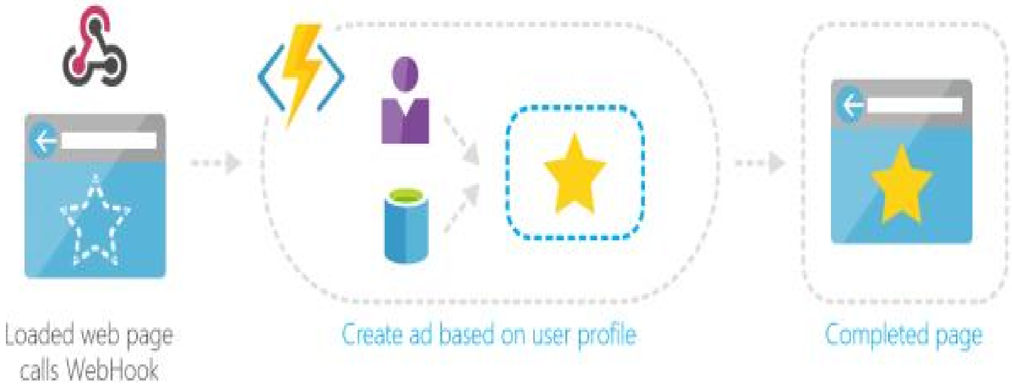
Source : Microsoft

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

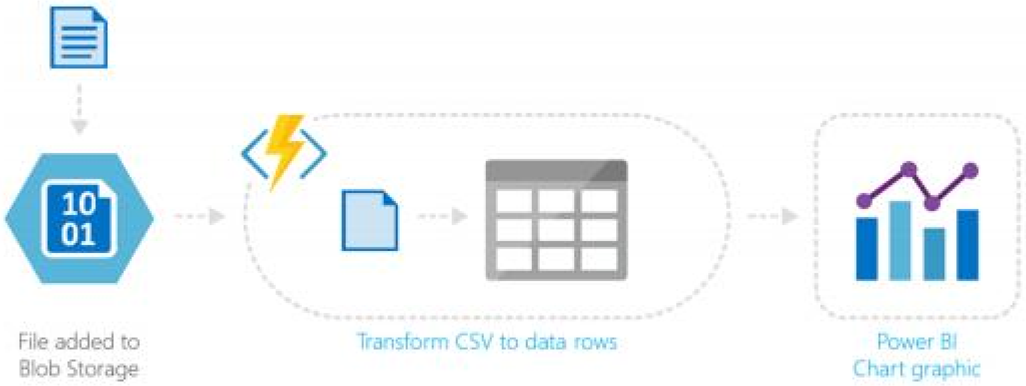

Loaded web page calls WebHook
Create ad based on user profile
Completed page

Source : Microsoft

Data pipeline

- Serverless functions can be used to facilitate a data pipeline

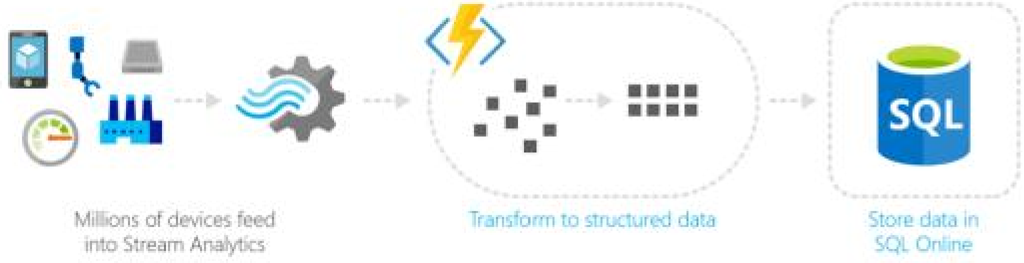
- For example,
- A file triggers a function to translate data in a CSV file to data rows in a table
- The organized table allows a BI dashboard to present analytics to the end user



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

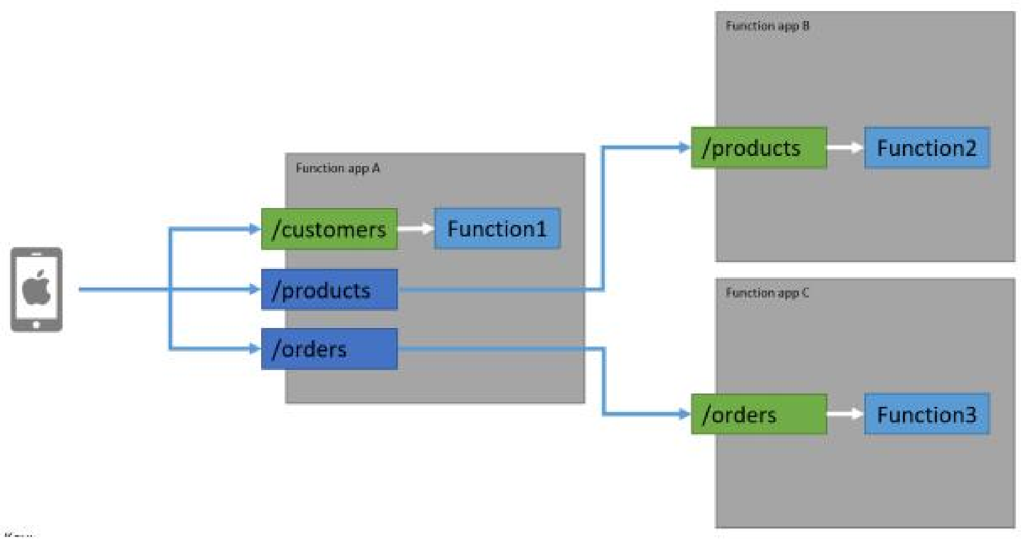

Millions of devices feed into Stream Analytics
Transform to structured data
SQL
Store data in SQL Online

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:

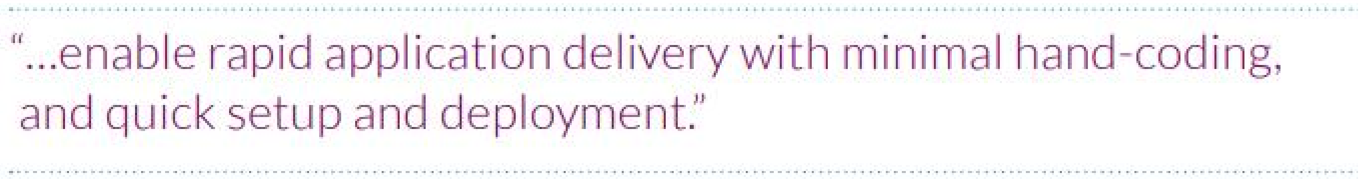
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Low code development

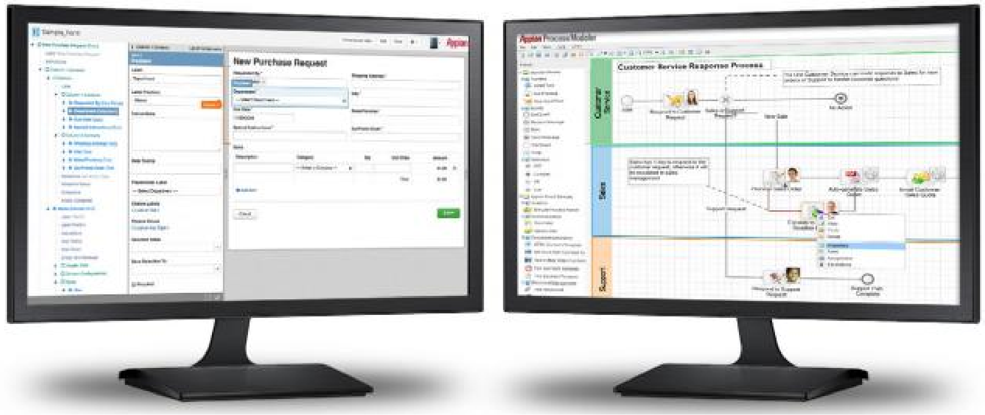
Chandan Ravandur N

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."



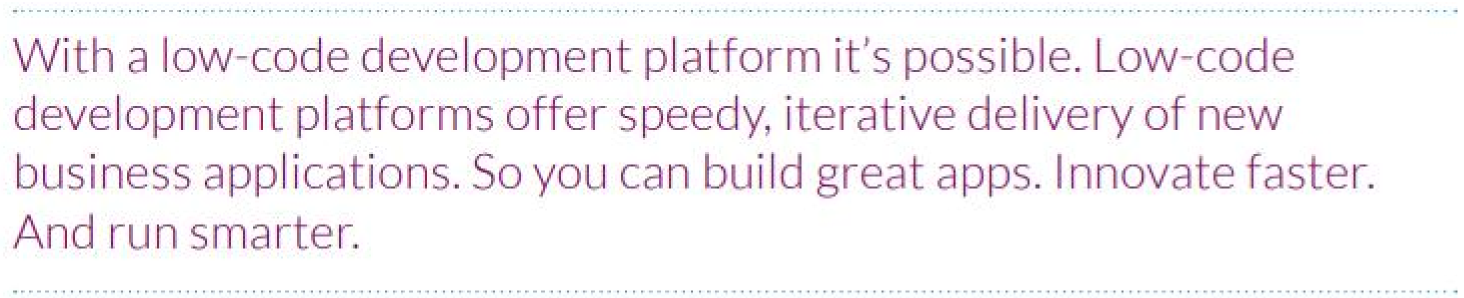
- Low-code development has evolved to take advantage of visual design tools like
  - ✓ drag-and-drop modelers
  - ✓ point-and-click interface creation
- —to enable the rapid creation, launch, use and change of powerful business apps

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?



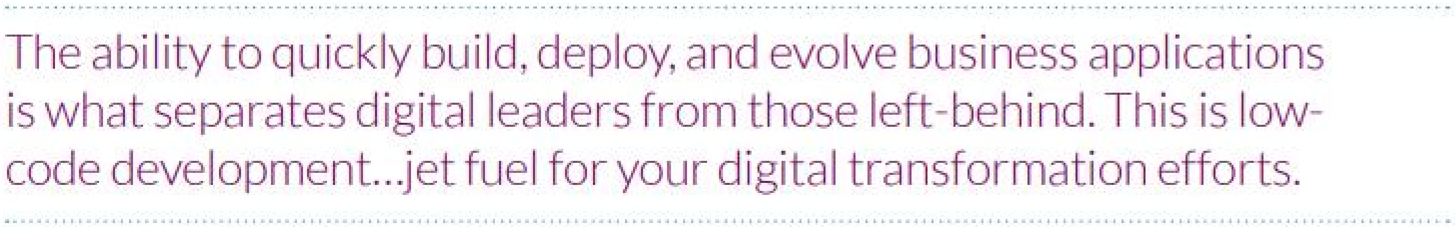
With a low-code development platform it's possible. Low-code development platforms offer speedy, iterative delivery of new business applications. So you can build great apps. Innovate faster. And run smarter.

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



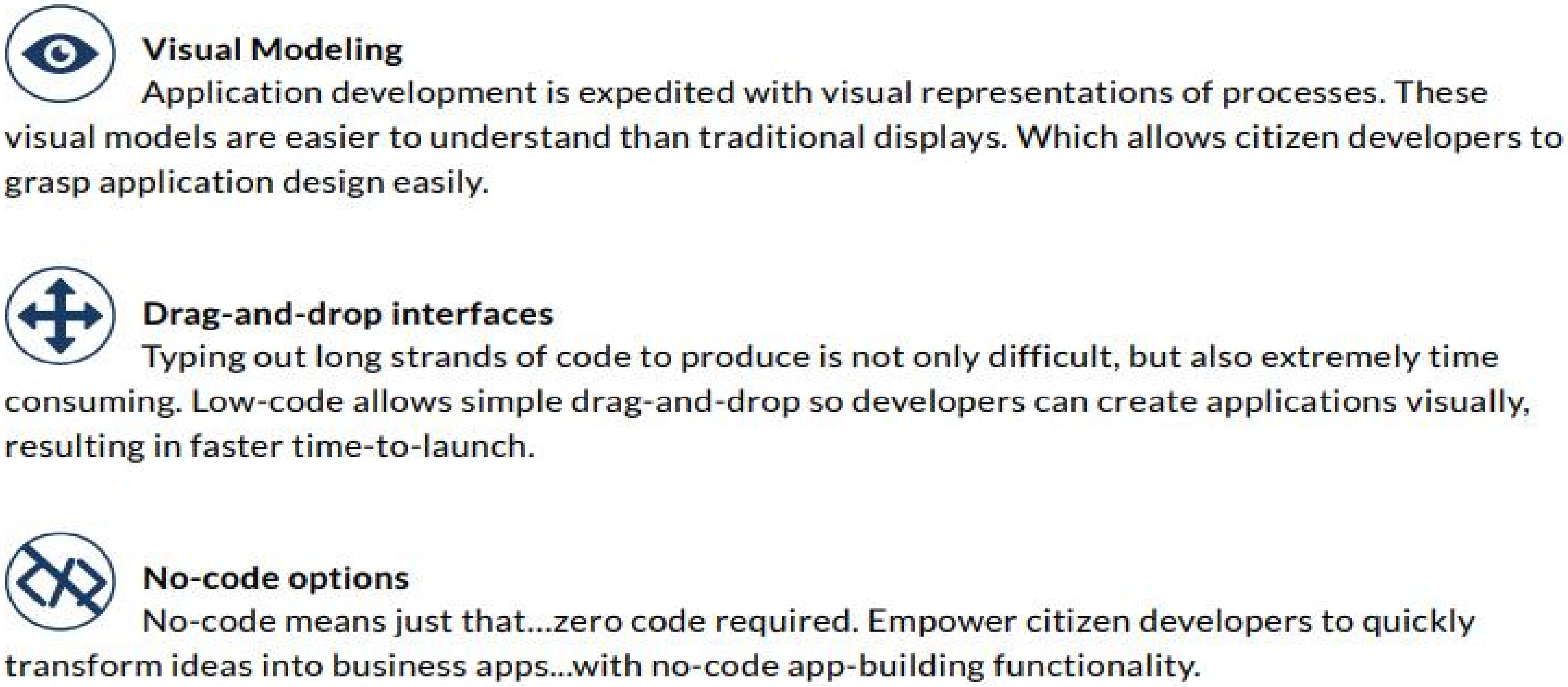
The ability to quickly build, deploy, and evolve business applications is what separates digital leaders from those left-behind. This is low-code development…jet fuel for your digital transformation efforts.

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.

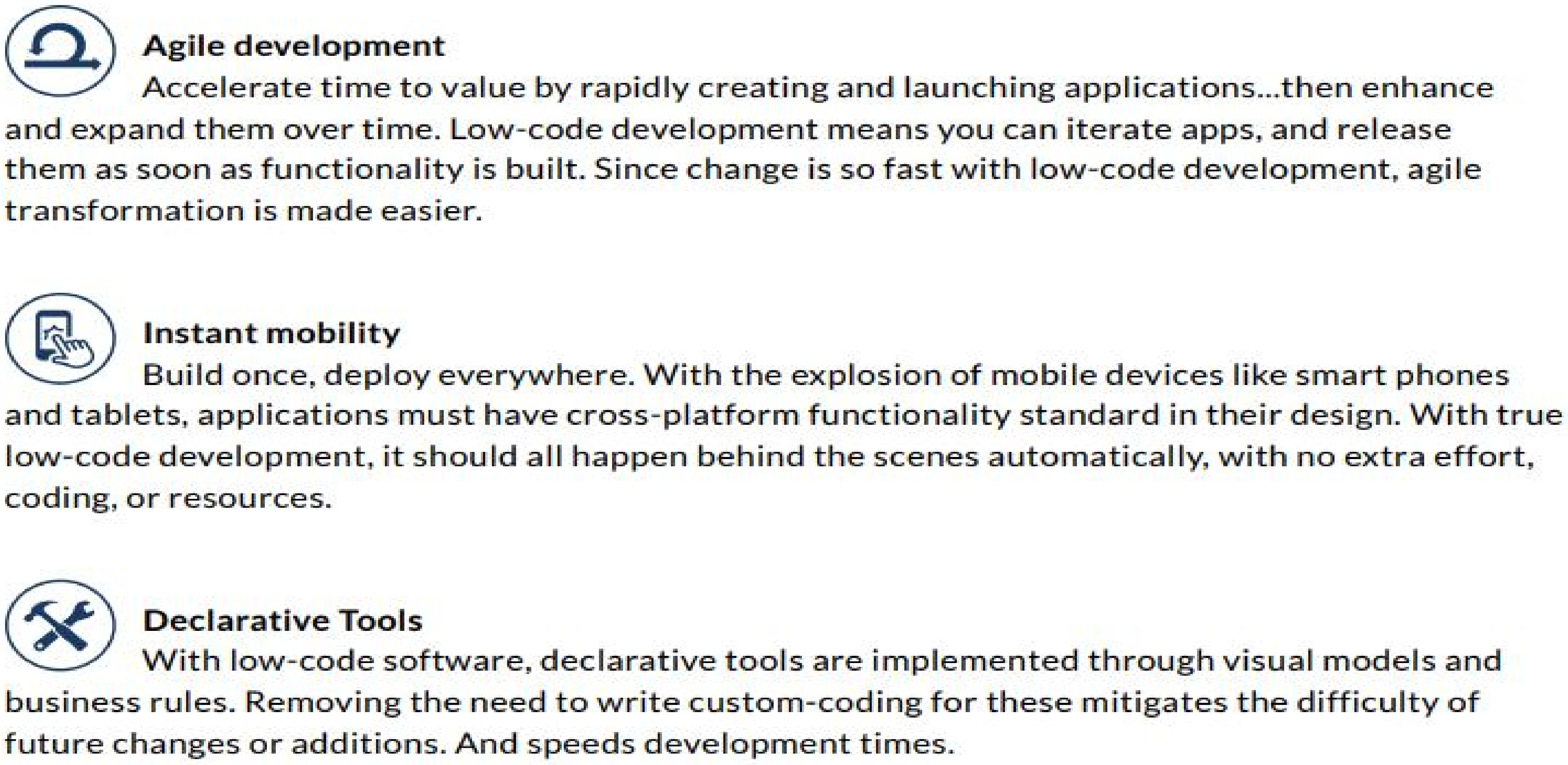
Drag-and-drop interfaces

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:

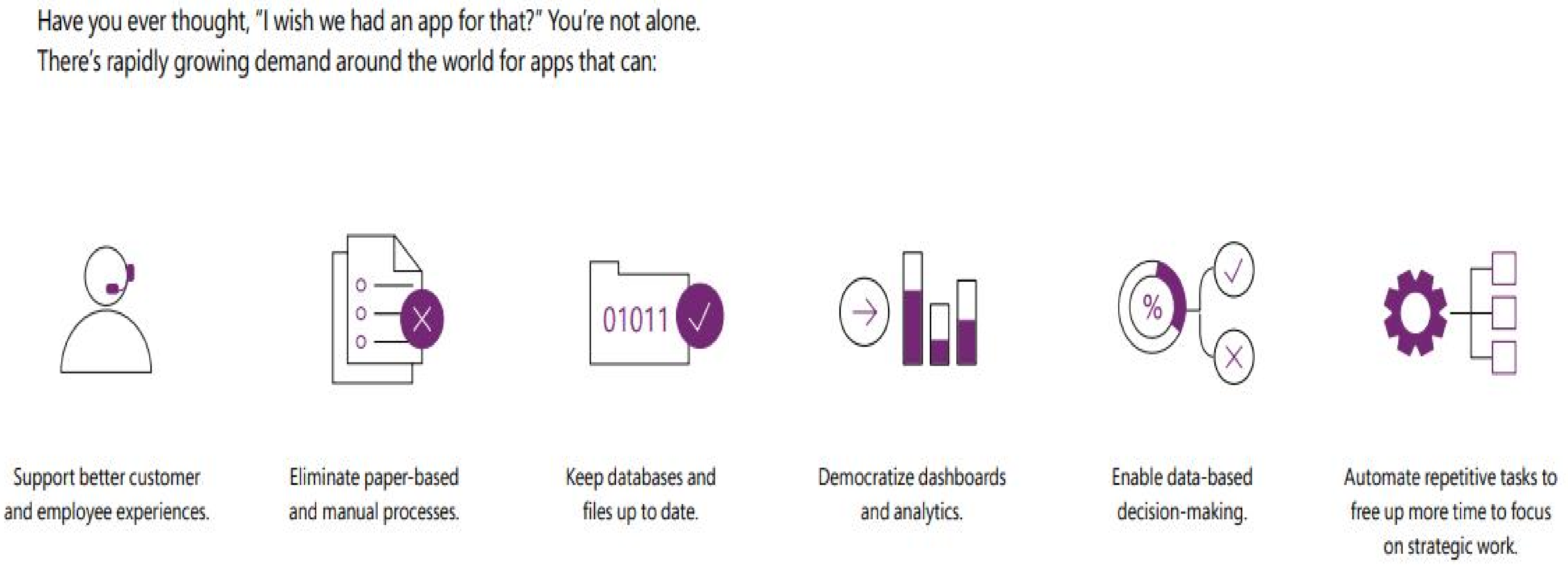
Developing Low Code Apps

Chandan Ravandur N

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

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?

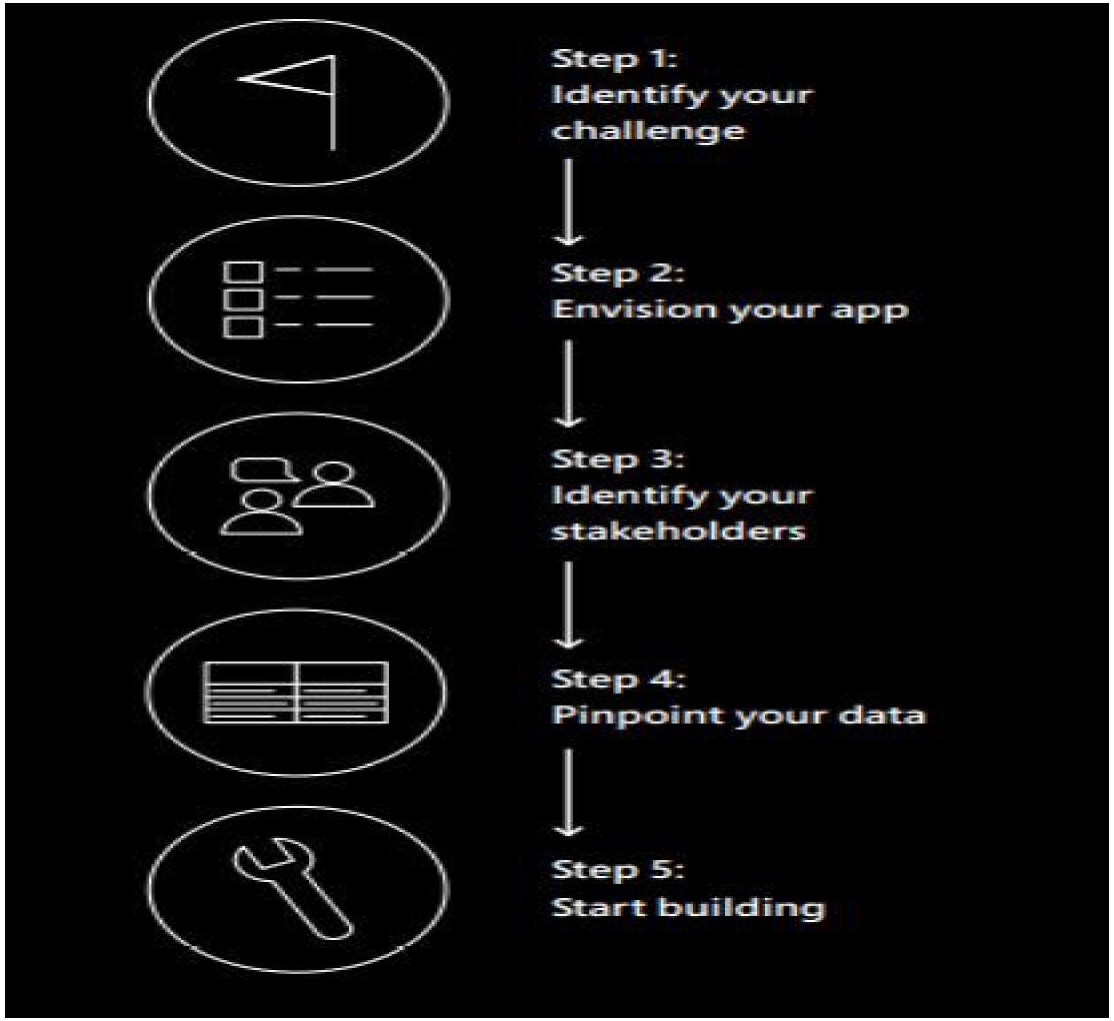
- With LCAP, you can
  - ✓ rapidly automate processes
  - ✓ build custom apps to meet your business needs
  - ✓ easy to start small with prebuilt templates and in-app guidance to deliver quick wins
  - ✓ if needed, pro developers can extend Power Apps for fully custom solutions

- 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

Step 2:
Envision your app

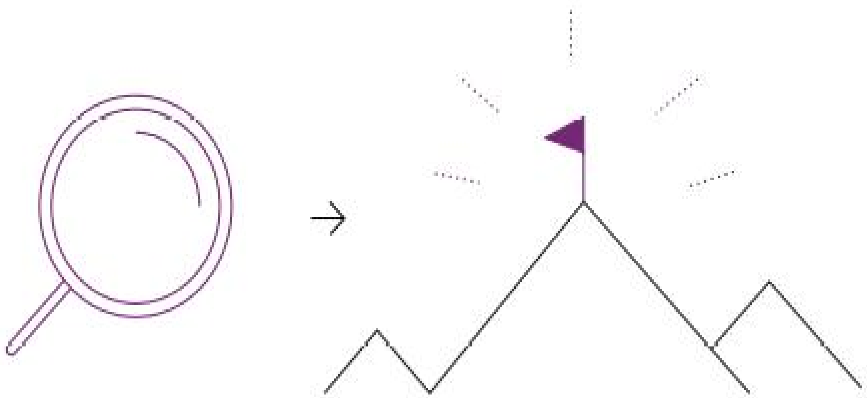
Step 3:
Identify your stakeholders

Step 4:
Pinpoint your data

Step 5:
Start building

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.

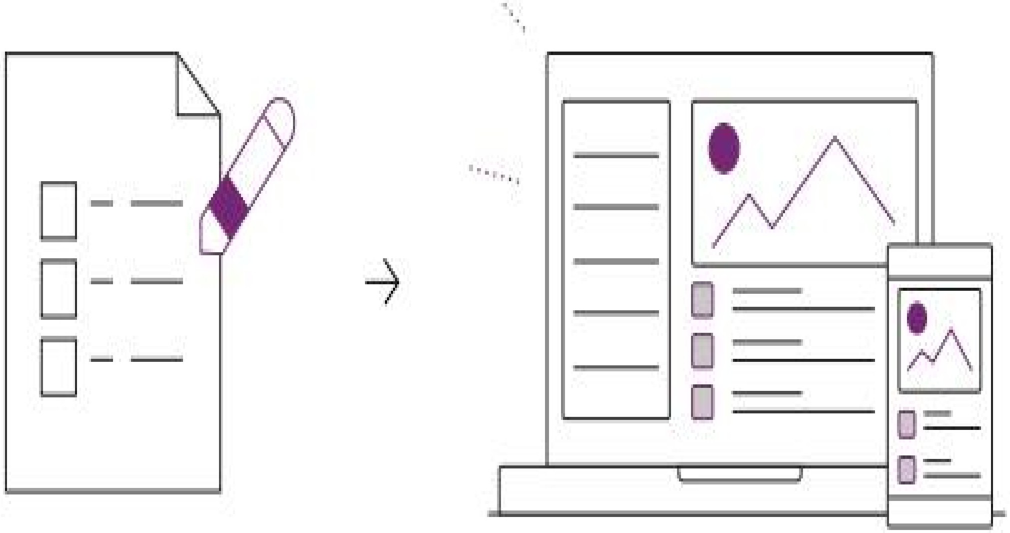


Common uses for Power Apps

- ✓ 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

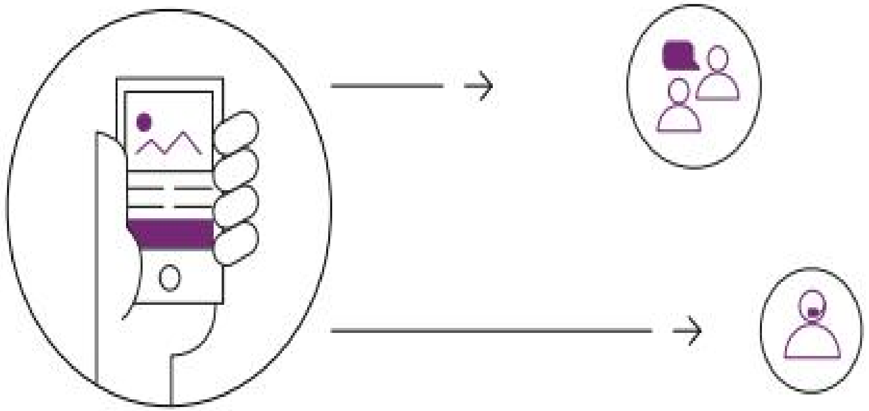
Take time to define your vision for the app. It doesn't have to be elaborate—just a few simple bullets about what you want it to do, who will use it, and what the experience will be like, along with a mockup of how the app could be structured. Getting a clear vision in your mind before you start will 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.

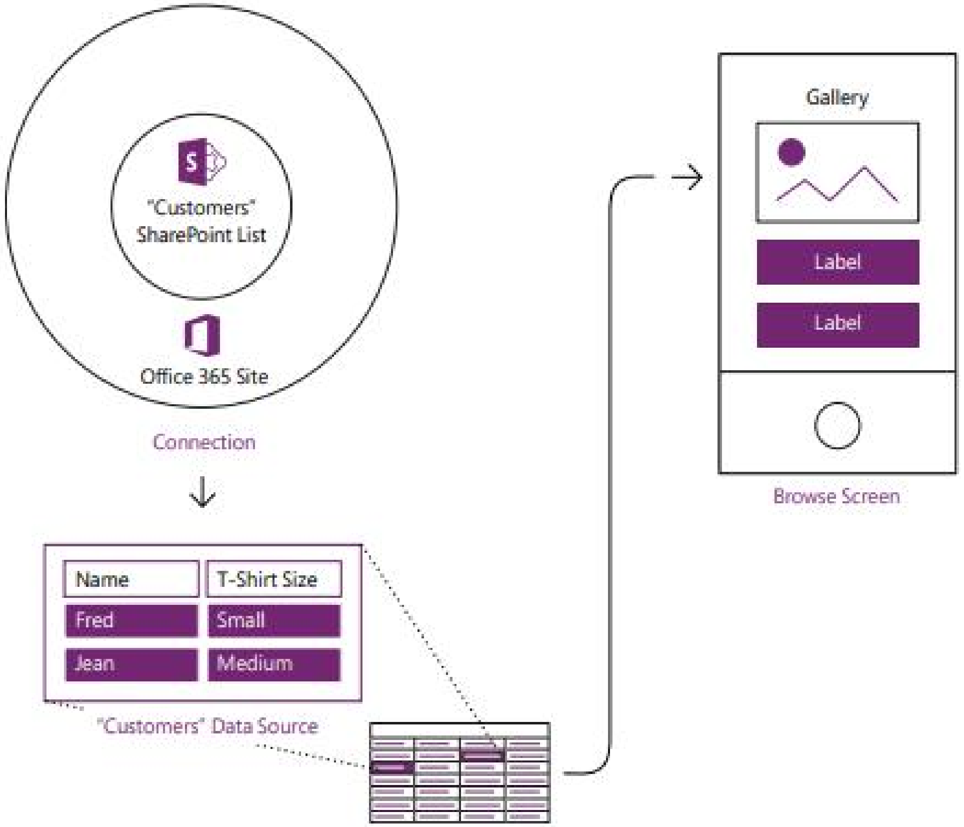
Step 4: Pinpoint your data

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

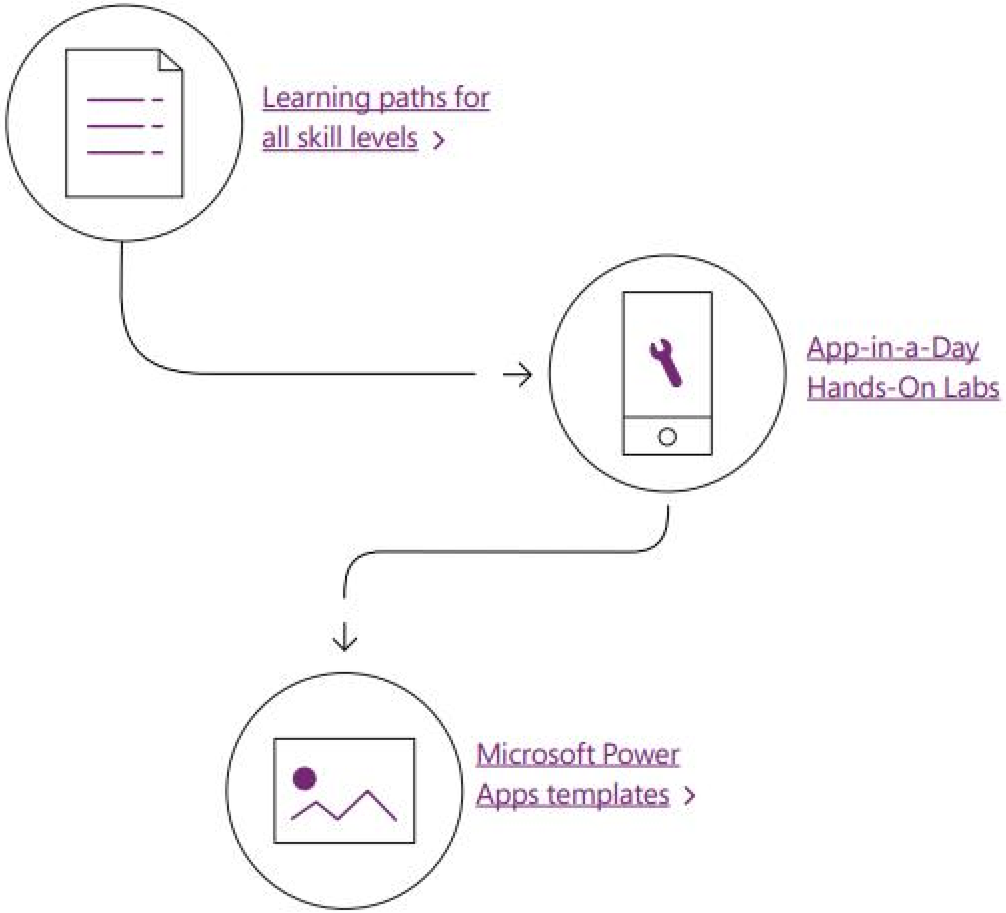
consider whether the data might be sensitive or proprietary and work with the appropriate people in your organization to keep it secure and compliant.

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.



Learning paths for all skill levels >

App-in-a-Day Hands-On Labs

Microsoft Power Apps templates >

Reference:
The DIY Guide to Building Your First Business App
Turn bright ideas into brilliant apps
by Microsoft

Thank You!

In our next session: