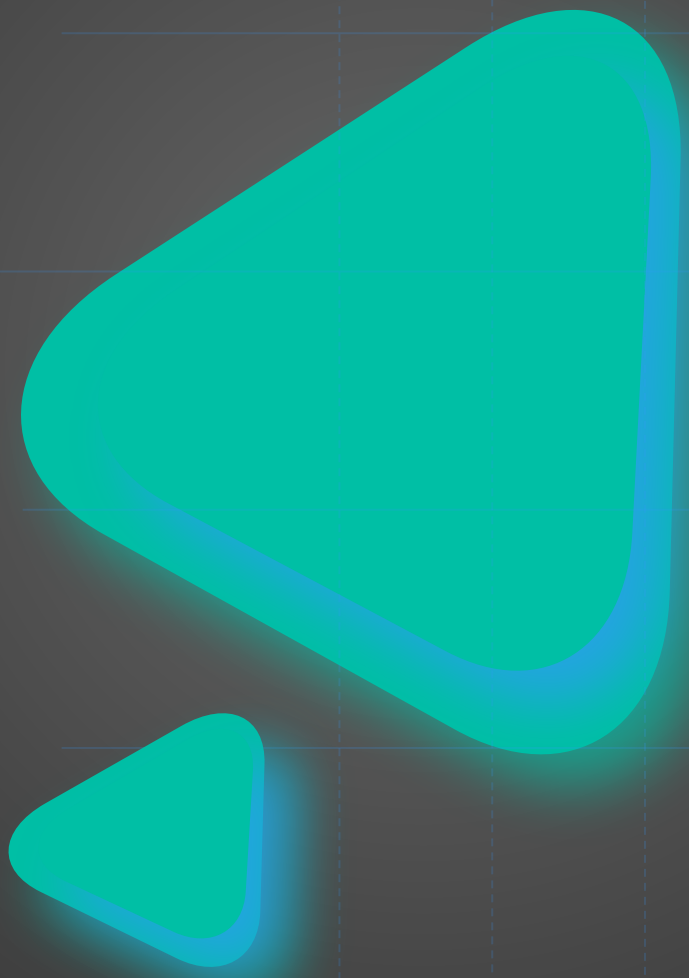
CLOUD NATIVE EVOLUTION



JASKARAN SINGH

pepper

Table of Contents

Introduction

In a relatively short span of time, we've evolved from the traditional approach of software development, in which the resources needed to run applications were deployed on-premise, to the **cloud native age**, in which the applications running in the cloud are purpose built and optimized to take full advantage of cloud benefits such as elasticity and resiliency.

It seems that everyone in the industry is talking about cloud native — including the role of distributed systems, microservices, containers, serverless computing, and other emerging technologies and architectures. While many organizations are somewhere in the process of defining and optimizing their cloud approach, there's still a fair amount of confusion among IT practitioners and decision makers about what cloud native really means and what the ultimate goals and expectations should be for an organization leveraging modern technologies and practices. The biggest question of all may be…

"How can my organization be successful with a cloud native approach and what impact will this successful technology approach have on my business objectives?"

This ebook aims to answer this fundamental question by relating the technical and cultural initiatives essential to the cloud native journey back to how they drive positive outcomes for both IT and the business. We'll also sprinkle in some best practices for extracting maximum value out of your cloud environment. Before we go into details about cloud native computing, we should first understand the goals of an enterprise application, how can we go about achieving those goals, and how do we evaluate performance metrics against those goals.

CHAPTER 1: Goals of an Enterprise Application

Every enterprise starts out by determining a set of goals to achieve while servicing its customers. We want applications to be available 24/7, be perpetually upgraded with new state-of-the-art features, and provide personalized experiences. We can summarize these goals into five pillars of an enterprise application.

1. Business Continuity

The Business Continuity pillar outlines how a business will continue operating during an unplanned disruption in service and contains contingencies for business processes, assets, human resources, and business partners — every aspect of the business that might be affected. The business continuity goal has the following design principles:

Availability: Described as the percentage of time when the service will be able to respond to the request i.e., system uptime

Durability: Refers to the ongoing existence of the object or resource. Note that it does not mean you can access the object or resource, only that it continues to exist.

Reliability: Closely related to availability, however, a system can be 'available' but not be working properly. Reliability is the probability that a system will work as designed.

Resiliency: Typically described as the ability of a system to self-heal after damage, failure, load, or attack. Note that this does not mean that it will be available continuously during the event, only that it will self-recover.

Establish recovery metrics — recovery time objective (RTO) and recovery point objective (RPO). RTO is the maximum acceptable time an application can be unavailable after an incident. RPO is the maximum duration of data loss that is acceptable during a disaster.

2. Security

The security pillar describes how to leverage cloud technologies to protect data, systems, and assets in a way that can improve your security posture. There are seven design principles for security in the cloud that a well architected enterprise solution should implement:

Implement a strong identity foundation: Implement the principle of least privilege and enforce separation of duties with appropriate authorization for each interaction with your resources. Centralize identity management and aim to eliminate reliance on long-term static credentials.

Enable traceability: Monitor, alert, and audit actions and changes to your environment in real time. Integrate log and metric collection with systems to automatically investigate and act.

Apply security at all layers: Apply a defense in depth approach with multiple security controls. Apply to all layers (for example, edge of network, VPC, load balancing, every instance and compute service, operating system, application, and code).

Automate security best practices: Automated software-based security mechanisms improve your ability to securely scale more rapidly and cost effectively. Create secure architectures, including the implementation of controls that are defined and managed as code in version-controlled templates.

Protect data in transit and at rest: Classify your data into sensitivity levels and use mechanisms, such as encryption, tokenization, and access control where appropriate.

Keep people away from data: Use mechanisms and tools to reduce or eliminate the need for direct access or manual processing of data. This reduces the risk of mishandling or modification and human error when handling sensitive data.

Prepare for security events: Prepare for an incident by having incident management and investigation policy and processes that align to your organizational requirements. Run incident response simulations and use tools with automation to increase your speed for detection, investigation, and recovery.

3. Agility

Agility is the ability of an organization to rapidly adapt to market and environmental changes in productive and cost-effective ways. To stay competitive, every business wants to make sure their systems are state-of-the-art and keeping up with the times. Here are some of the goals that a business strives to achieve:

Quicker time-to-market: Cloud computing allows companies to significantly decrease the time it takes to provision and deprovision IT infrastructure, speeding delivery of IT projects that are critical to revenue growth or cost reduction. While a physical server could take days or weeks to procure and provision, a cloud server takes minutes. Faster time to market means faster time to revenue.

Faster innovation: Cloud computing allows companies to support an increased pace of product development and marketing programs that better align IT infrastructure and management costs with the goals and objectives of the business.

Make frequent, small, reversible changes: Design workloads to allow components to be updated regularly to increase the flow of beneficial changes into your workload. Make changes in small increments that can be reversed if they fail to aid in the identification and resolution of issues introduced to your environment (without affecting customers when possible).

Anticipate failure: Perform “pre-mortem” exercises to identify potential sources of failure so that they can be removed or mitigated. Test your failure scenarios and validate your understanding of their impact. Test your response procedures to ensure they are effective and that teams are familiar with their execution. Set up regular game days to test workload and team responses to simulated events.

Learn from all operational failures: Drive improvement through lessons learned from all operational events and failures. Share what is learned across teams and through the entire organization.

4. Scalability and Performance Efficiency

Performance efficiency is the ability of your workload to scale to meet the demands placed on it by users in an efficient manner. The Performance Efficiency pillar includes the ability to use computing resources efficiently to meet system requirements, and to maintain that efficiency as demand changes and technologies evolve. Scalability is the ability of a system to handle increased load. An important consideration in achieving performance efficiency is to consider how your application scales and to implement PaaS offerings that have built-in scaling operations. Services can scale automatically to match demand to accommodate workload. They will scale out to ensure capacity during workload peaks and scaling will return to normal automatically when the peak drops. Here are some of the goals:

Become data-driven: Embrace a data-driven culture to deliver timely insights to everyone in your organization across all your data. To harness this culture, get the best performance from your analytics solution across all your data, ensure data has the security and privacy needed for your business environment, and make sure you have tools that enable everyone in your organization to gain insights from your data.

Perform load testing to set limits: Load testing helps ensure that your applications can scale and do not go down during peak traffic. Load test each application to understand how it performs at various scales.

Understand billing for metered resources: Your business requirements will determine the tradeoffs between cost and level of performance efficiency. Azure doesn't directly bill based on the resource cost. Charges for a resource, such as a virtual machine, are calculated by using one or more meters. Meters are used to track a resource's usage over time. These meters are then used to calculate the bill.

Monitor and optimize: Lack of monitoring new services and the health of current workloads are major inhibitors in workload quality. The overall monitoring strategy should take into consideration not only scalability, but resiliency (infrastructure, application, and dependent services) and application performance as well. For purposes of scalability, looking at the metrics would allow you to provision resources dynamically and scale with demand.

Use serverless architectures: Serverless architectures remove the need for you to run and maintain physical servers for traditional compute activities. For example, serverless storage services can act as static websites (removing the need for web servers) and event services can host code. This removes the operational burden of managing physical servers, and can lower transactional costs because managed services operate at cloud scale

5. Cost Optimization

The Cost Optimization pillar includes the ability to run systems to deliver business value at the lowest price point. Here are some of the goals that organizations strive to meet:

Keep within the cost constraints: Every design choice has cost implications. Before choosing an architectural pattern consider the budget constraints set by the company. As part of design, identify acceptable boundaries on scale, redundancy, and performance against cost. After estimating the initial cost, set budgets and alerts at different scopes to measure the cost. One of cost drivers can be unrestricted resources. These resources typically need to scale and consume more cost to meet demand.

Aim for scalable costs: A key benefit of the cloud is the ability to scale dynamically. The workload cost should scale linearly with demand. You can save cost through automatic scaling. Choose smaller instances for a highly variable workload and scale out to get the required level of performance, rather than up. This choice will enable you to make your cost calculations and estimates granular.

Pay for consumption: Adopt a leasing model instead of owning infrastructure. Azure offers many SaaS and PaaS resources that simplify overall architecture. The cost of hardware, software, development, operations, security, and data center space included in the pricing model. Also, choose to pay-as-you-go over fixed pricing. That way, as a consumer, you are charged for only what you use.

Right resources, right size: Choose the right resources that are aligned with business goals and can handle the performance of the workload. An inappropriate or misconfigured service can impact cost. For example, building a multi-region service when the service levels don't require high-availability or geo-redundancy will increase cost without any reasonable business justification. Certain infrastructure resources are delivered as fix-sized building blocks. Ensure that these blocks are adequately sized to meet capacity demand, deliver expected performance without wasting resources.

Monitor and optimize: Treat cost monitoring and optimization as a process, rather than a point-in-time activity. Conduct regular cost reviews and measure and forecast the capacity needs so that you can provision resources dynamically and scale with demand. Review the cost management recommendations and act.

CHAPTER 2: How We Got Here

Every enterprise in today's modern era tries to meet the goals we just discussed. Your software needs to be up, **running 24/7**. You need to be able to **release frequently** to give your users the instant gratification they seek. The **mobility and always-connected state** of your users drives a need for your software to be responsive to larger and more fluctuating request volumes than ever before. And connected devices ("things") form a distributed data fabric of unprecedented size that requires new storage and processing approaches. These needs, along with the availability of new platforms on which you can run the software, have led directly to the emergence of a new architectural style for software – Cloud native software.

In the pre-cloud era software was developed as a *waterfall approach,* software took *years to create*, with *infrequent releases* of new features. *Development, quality assurance, and operations were distinct groups*, often operating in silos. Applications were deployed on infrastructure that consisted of physical hardware which was usually located in on-premises or in data centers. With the advent of server virtualization, virtual servers were deployed to consolidate operations on fewer, larger physical servers. Operational tasks were highly manual, limiting the scale at which systems could operate. Over time, provisioning and maintaining the physical and virtual infrastructure were made less onerous by configuration management tools. When it came to planning how a system would handle increases in load, many organizations *intentionally provisioned workloads to be oversized* to meet business requirements. This might make sense in on-premises environments because it ensured capacity during peak usage. Capacity reflects resource availability (CPU and memory). This was a major consideration for processes that would be in place for several years.

The **cloud model** is a departure from previous data center strategies since the model provides a **pool of resources** that can be consumed by users as services as opposed to dedicating infrastructure to each individual application. In the case of public cloud, these services can take the shape of end- user applications, back-office platforms, or virtual servers – all hosted over the internet with simple billing models that charge the user just what they use.

During the initial years of the cloud technology adoption, organizations tried to follow a simple *"lift and shift"* approach which involved merely migrating applications from the data centers to the cloud without making any changes to application architecture. Multi-tier applications were simply migrated from local physical servers to a virtualized cloud environment.

"This progression toward the cloud native era highlights the fact that a simple migration or "lift and shift" of applications to the cloud is not enough to exploit the true benefits of the cloud."

When cloud computing initially arose, many IT departments tried to merely transfer their systems to the cloud without making any changes to application architecture. Multi-tier applications were simply migrated from local physical servers to a virtualized cloud environment. Over time, cloud system engineers have made many innovative improvements in cloud platforms and infrastructures. These initiatives have spawned several engineering trends that continue to this day. One such trend is the move from monolithic systems toward **microservice architectures,** which has enabled the cloud native movement we see exploding today.

CHAPTER 3: What are cloud native apps?

Within a short time, cloud native has become a driving trend in the software industry. Cloud native computing is a new way to think about building large, complex systems, an approach that takes full advantage of modern software development practices, technologies, and cloud infrastructure. The approach changes the way you design, implement, deploy, and operationalize systems.

Consider the ***Cloud Native Computing Foundation (CNCF)***, a consortium of over 300 major corporations with a charter to make cloud native computing ubiquitous across technology and cloud stacks. The CNCF provides an official definition:

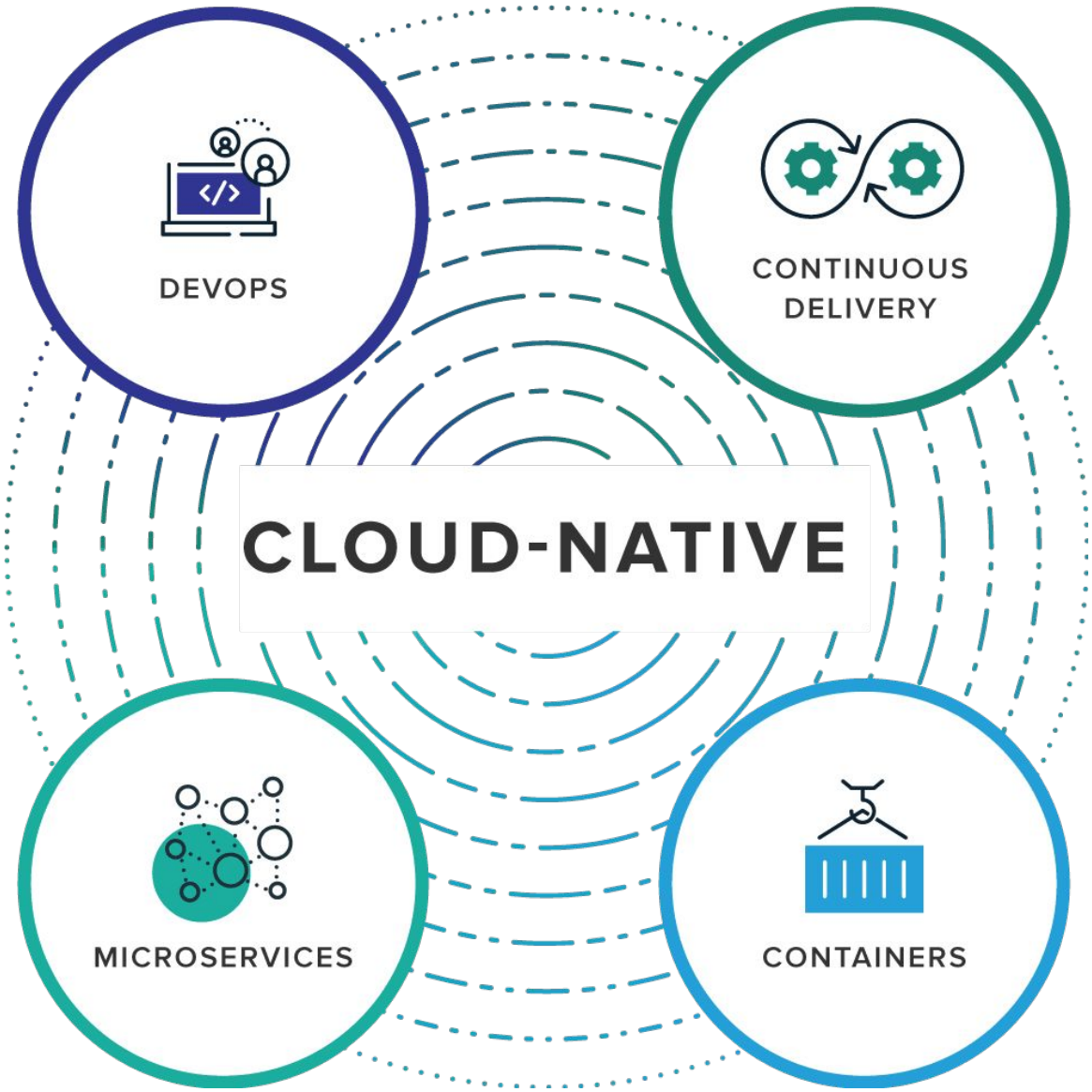
- Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.

- These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil.

Cloud native is about speed, agility, and scalability

As you can see, Netflix, Uber, and WeChat expose systems that consist of hundreds of independent microservices. This architectural style enables them to rapidly respond to market conditions. They can instantaneously update small areas of a live, complex application, and individually scale those areas as needed. The speed and agility of cloud native come about from several factors.

Cloud native is an approach on how you design & implement, deploy, and operationalize systems.



Architecture

A cloud native app is designed and architected specifically to exist in the cloud. A widely accepted methodology for constructing cloud-based applications is the **Twelve-Factor Application**. It describes a set of principles and practices that developers follow to construct applications optimized for modern cloud environments. Special attention is given to portability across environments and declarative automation.

While applicable to any web-based application, many practitioners consider Twelve-Factor as a solid foundation for building cloud native apps. Systems built upon these principles can deploy and scale rapidly and add features to react quickly to market changes.

12 Factor Methodology

Code base: A single code base for each microservice, stored in its own repository. Tracked with version control, it can deploy to multiple environments (QA, Staging, Production).

Dependencies: Each microservice isolates and packages its own dependencies, embracing changes without impacting the entire system.

Configurations: Configuration information is moved out of the microservice and externalized through a configuration management tool outside of the code. The same deployment can propagate across environments with the correct configuration applied.

Backing services: Ancillary resources (data stores, caches, message brokers) should be exposed via an addressable URL. Doing so decouples the resource from the application, enabling it to be interchangeable.

Build, release, run: Each release must enforce a strict separation across the build, release, and run stages. Each should be tagged with a unique ID and support the ability to roll back. Modern CI/CD systems help fulfill this principle.

Processes: Each microservice should execute in its own process, isolated from other running services. Externalize required state to a backing service such as a distributed cache or data store.

Port binding: Each microservice should be self-contained with its interfaces and functionality exposed on its own port. Doing so provides isolation from other microservices.

Concurrency: Services scale out across many small identical processes (copies) as opposed to scaling-up a single large instance on the most powerful machine available.

Disposability: Service instances should be disposable, favoring fast startups to increase scalability opportunities and graceful shutdowns to leave the system in a correct state. Docker containers along with an orchestrator inherently satisfy this requirement.

Dev/prod parity: Keep environments across the application lifecycle as similar as possible, avoiding costly shortcuts. Here, the adoption of containers can greatly contribute by promoting the same execution environment.

Logging: Treat logs generated by microservices as event streams. Process them with an event aggregator and propagate the data, to data mining/log management tools like Azure Monitor or Splunk and eventually long-term archival.

Admin processes: Run administrative/management tasks as one-off processes. Tasks can include data cleanup and pulling analytics for a report. Tools executing these tasks should be invoked from the production environment, but separately from the application.

Microservices

A central tenet of a cloud native approach is the concept of leveraging distributed systems. This includes either transforming monolithic applications into decomposed services or developing distributed systems from scratch. Either way, the idea is to enable continuous delivery and deployment of large, complex applications.

A microservices architecture is the approach of choice for breaking down applications into their smallest reasonable services. The main idea behind a micro-service architecture is that applications are simpler to build, modify, and maintain when broken down into smaller pieces that work seamlessly together.

Built as a distributed set of small, independent services that interact through a shared fabric, microservices share the following 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 using standard communication protocols such as HTTP/HTTPS, WebSockets, or AMQP.

They compose together to form an application.

Because each microservice runs in its own container, service, or function, developers can build, manage, and scale each service independently. When implemented well, a microservices architecture has the potential to:

Accelerate the velocity and increase the scale of software deployment. Each microservice has an autonomous lifecycle and can evolve independently and deploy frequently. You don't have to wait for a quarterly release to deploy a new feature or update.

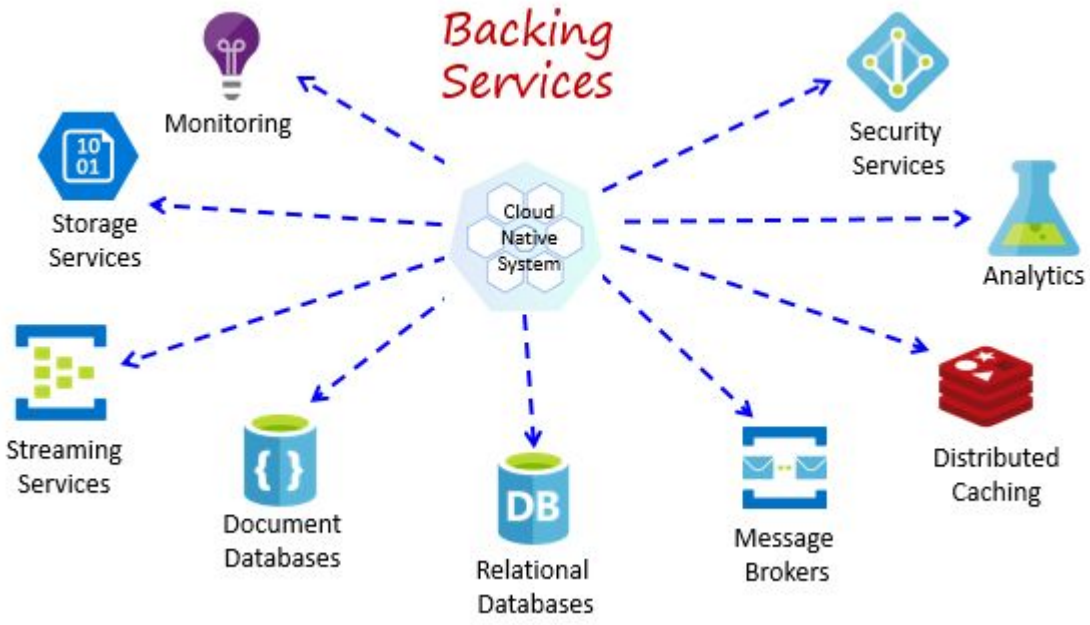
Improve application scalability and optimize scaling decisions. Each microservice can scale independently. Instead of scaling the entire application as a single unit, you scale out only those services that require more processing power or network bandwidth. This fine-grained approach to scaling provides for greater control of your system and helps to reduce overall costs as you scale portions of your system, not everything.

Improve fault isolation. You can update a small area of a complex application with less risk of disrupting the entire system.

Increase business agility. Enable smaller and more agile teams

Backing Services

Cloud-native systems depend upon many different ancillary resources, such as data stores, message brokers, monitoring, and identity services. These services are known as backing services.



You could host your own backing services, but then you would be responsible for licensing, provisioning, and managing those resources.

Cloud providers offer a rich assortment of managed backing services. Instead of owning the service, you simply consume it. The provider operates the resource at scale and bears the responsibility for performance, security, and maintenance. Monitoring, redundancy, and availability are built into the service.

A best practice is to treat a backing service as an attached resource, dynamically bound to a microservice with information (a URL and credentials) stored in an external configuration. With this pattern, a backing service can be attached and detached without code changes. You might promote a microservice from QA to a staging environment. You update the microservice configuration to point to the backing services in staging and inject the settings into your container through an environment variable.

Containers

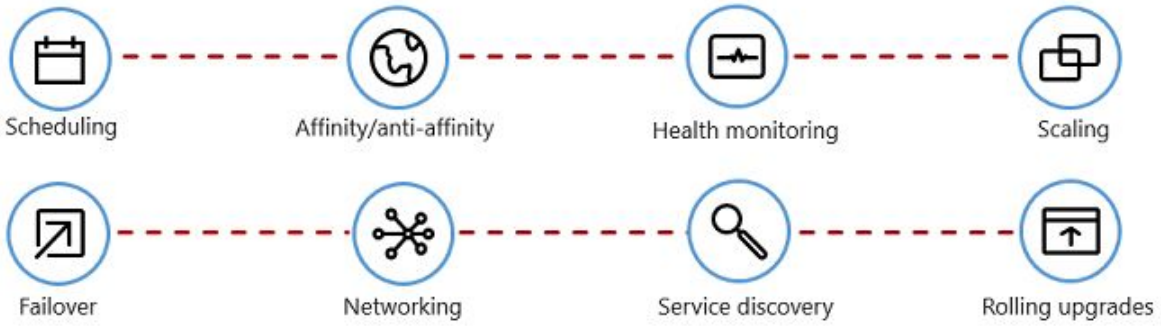
Containers are a great enabler of cloud native software. A container is a standard unit of software that packages up code and all its dependencies, so the application runs quickly and reliably from one computing environment to another. A Docker Container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings. Containers are gaining popularity, and here are some of the reasons:

Packaging and portability: Containers provide a packaging mechanism. This allows the building of a system to be separated from the deployment of those systems. Containerized software will always run the same, regardless of the infrastructure. Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging. Finally, deployments become more atomic.

Efficiency: Containers can be lighter weight than full systems, leading to increased resource utilization. By sharing a kernel and allowing for much more fluid overcommit, containers can make it easier to maximize the use of compute resources.

Container Orchestration

While tools such as Docker create images and run containers, you also need tools to manage them. Container management is done with a special software program called a container orchestrator. When operating at scale, container orchestration is essential.



DevOps

DevOps is a tightly woven collaboration among software developers and IT operations. For cloud native DevOps teams, the difference from conventional teams is that these apps are built to exploit the adaptability and resiliency of cloud computing technologies. The goal is to continuously and progressively deliver high-quality software that meets customer requirements and expectations. On the best teams, DevOps supports a culture in which software is frequently built, tested, and released with a high degree of consistency.

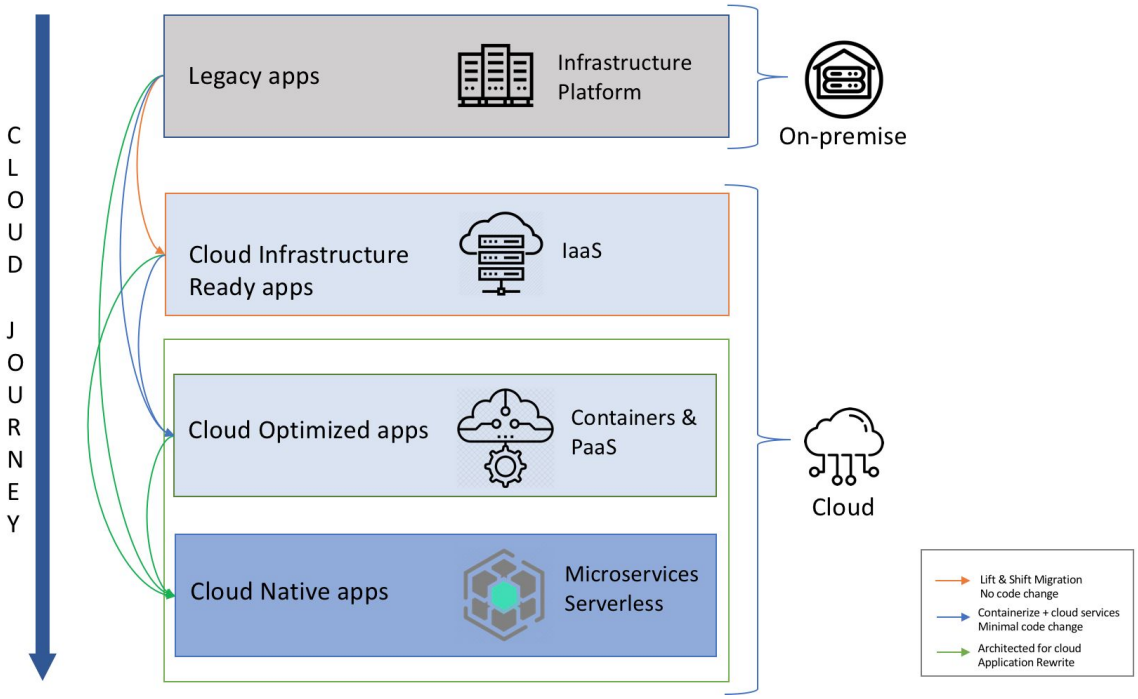
Infrastructure as Code, or IaC is a widely accepted practice to automate platform provisioning and application deployment. You essentially apply software engineering practices such as testing and versioning to your DevOps practices. Your infrastructure and deployments are automated, consistent, and repeatable. Teams who implement IaC can deliver stable environments rapidly and at scale. Teams avoid manual configuration of environments and enforce consistency by representing the desired state of their environments via code. Infrastructure deployments with IaC are repeatable and prevent runtime issues caused by configuration drift or missing dependencies. DevOps teams can work together with a unified set of practices and tools to deliver applications and their supporting infrastructure rapidly, reliably, and at scale.

CHAPTER 4: From Legacy Apps to Cloud Native

It Doesn't Happen Overnight

Cloud native is more than migrating existing applications to the cloud or flipping a switch to turn on new cloud services. It's also more than developing your applications from scratch in the cloud. It's about how you develop and run applications, not just where those applications are deployed.

Which is why becoming cloud native doesn't happen the instant you leverage cloud services or infrastructure. Rather, it requires fundamental changes to processes, architecture, culture, and measurement. These changes take time and commitment, and the cycle is iterative and constant.



There isn't a single, one-size-fits-all strategy for migrating legacy applications to the cloud. The right migration strategy for you will depend on your organization's needs and priorities, and the kind of applications you are migrating. Not all applications warrant the investment of moving to a platform as a service (PaaS) model or developing a cloud native application model. In many cases, you can take a phased or incremental approach to invest in moving your assets to the cloud, based on your business needs.

For modern applications with the best long-term agility and value for the organization, you might benefit from investing in cloud native application architectures. However, for applications that are existing or legacy assets, the key is to spend minimal time and money (no rearchitecting or code changes) while moving them to the cloud, to realize significant benefits. The above diagram shows the possible paths you can take when you move existing applications to the cloud in incremental phases.

Cloud Infrastructure-Ready Applications: In this migration approach, you simply migrate or re-host your current on-premises applications to an infrastructure as a service (IaaS) platform. Your apps have almost the same composition as before, but now you deploy them to VMs in the cloud. This simple type of migration is typically known in the industry as "Lift & Shift."

Advantages

- No re-architecting, no new code
- Least effort for quick migration
- Basic availability guarantees

After moving to the cloud, it is easier to modernize even more

Challenges

- Smaller cloud value, other than shift in operating expense or closing data centers
- Little is managed. No OS or middleware patching.

Cloud Optimized Applications

At this level and still without rearchitecting or altering significant code, you can gain additional benefits from running your app in the cloud with modern technologies like containers and additional cloud-managed services. You improve the agility of your applications to ship faster by refining your enterprise development operations (DevOps) processes. You achieve this by using technologies like Containers, which is based on Docker Engine. Containers remove the friction that is caused by application dependencies when you deploy in multiple stages. In this maturity model, you can deploy containers on IaaS or PaaS while using additional cloud-managed services related to databases, cache as a service, monitoring, and continuous integration/continuous deployment (CI/CD) pipelines.

Advantages

- No re-architecting, no new code
- Minimal code/config changes Containers offer small incremental effort over VMs
- Improved deployment and DevOps agility to release because of containers
- Increased density and lower deployment costs
- Portability of apps and dependencies
- Flexibility of host targets: PaaS approaches or IaaS

Challenges

- Containerizing is an additional step in the learning curve for developers and IT Operations
- DevOps and CI/CD pipelines is usually ‘a must’ for this approach. If not currently present in the culture of the organization, it might be an additional challenge

Cloud Native Applications

This migration approach typically is driven by business need and targets modernizing your mission-critical applications. At this level, you use PaaS services to move your apps to PaaS computing platforms. You implement cloud native applications and microservices architecture to evolve applications with long-term agility, and to scale to new limits. This type of modernization usually requires architecting specifically for the cloud. New code often must be written, especially when you move to cloud native application and microservice-based models. This approach can help you gain benefits that are difficult to achieve in your monolithic and on-premises application environment.

Advantages

- Architect for the cloud, you get the best benefits from the cloud, but new code is needed
- Microservices cloud native approaches
- Modern mission-critical applications, cloud-resilient, hyper-scalable
- Fully managed services
- Optimized for scale
- Optimized for autonomous agility by subsystem
- Built on deployment
- DevOps

Challenges

- Requires re-architecture for cloud native apps and microservice architectures and usually requires significant code refactoring or rewriting when modernizing (increased time and budget)

CHAPTER 5: Benefits of Cloud Computing

Speed to market

The cloud allows organizations to decrease the time it takes to provision IT infrastructure, speeding delivery of IT projects that are critical to revenue growth or cost reduction. While a physical server could take days or weeks to procure and provision, a cloud server takes minutes. Faster time to market means faster time to revenue.

Cost savings

With traditional on-premises enterprise solutions, IT architects build dedicated, hard-wired infrastructure that often is over-sized to accommodate worst-case capacity estimates — but with little capability for scaling to meet future increase in demand. This results in excessive and wasteful over-capacity. But the cloud methodology allows you to be only responsible for what you use. **Pay-as-you-go** cloud computing system allows users to **scale, modify, and provision computing resources,** including but not limited to software, storage, and other platforms. Fees charged are based on used services, versus an entire infrastructure. Hence, paying only for the IT capacity you consume generates major savings. By using cloud infrastructure, you don't have to spend huge amounts of money on purchasing and maintaining equipment. This drastically **reduces capex costs and moves the technology spend to Operating Expenses**. You don't have to invest in hardware, facilities, utilities, or building out a large data center to grow your business. You do not even need large IT teams to handle your cloud data center operations, as you can enjoy the expertise of your cloud provider's staff.

Data security

One of the major concerns of every business, regardless of size and industry, is the security of its data. Data breaches and other cybercrimes can devastate a company's revenue, customer loyalty and brand positioning. Cloud offers many advanced security features that guarantee that data is securely stored and handled. Cloud storage providers implement baseline protections for their platforms and the data they process, such authentication, access control, and encryption. From there, most enterprises supplement these protections with added security measures of their own to bolster cloud data protection and tighten access to sensitive information in the cloud.

Scalability

Different companies have different IT needs — a large enterprise of 1000+ employees won't have the same IT requirements as a start-up. Using cloud is a great solution because it enables enterprise to efficiently — and quickly — scale up/down their IT departments, according to business demands. Cloud based solutions are ideal for businesses with growing or fluctuating bandwidth demands. If your business demands increase, you can easily increase your cloud capacity without having to invest in physical infrastructure. This level of agility can give businesses using cloud computing a real advantage over competitors. This scalability minimizes the risks associated with in-house operational issues and maintenance. You have high-performance resources at your disposal with professional solutions and zero up-front investment. Scalability is probably the greatest advantage of the cloud.

Disaster recovery

One of the factors that contributes to the success of a business is control. Unfortunately, no matter how in control your organization may be when it comes to its own processes, there will always be things that are completely out of your control, and in today's market, even a small amount of unproductive downtime can have a resoundingly negative effect. Downtime in your services leads to lost productivity, revenue, and brand reputation. But while there may be no way for you to prevent or even anticipate the disasters that could potentially harm your organization, there is something you can do to help speed your recovery. Cloud-based services provide quick data recovery for all kinds of emergency scenarios, from natural disasters to power outages.

Competitive edge

One of the factors that contributes to the success of a business is control. Unfortunately, no matter how in control your organization may be when it comes to its own processes, there will always be things that are completely out of your control, and in today's market, even a small amount of unproductive downtime can have a resoundingly negative effect. Downtime in your services leads to lost productivity, revenue, and brand reputation. But while there may be no way for you to prevent or even anticipate the disasters that could potentially harm your organization, there is something you can do to help speed your recovery. Cloud-based services provide quick data recovery for all kinds of emergency scenarios, from natural disasters to power outages.

Jaskaran Singh is a technology and strategy leader specializing in propelling world-class high technology initiatives that meet diverse business operations and industry needs. With over 20 years of experience across various industries, Jaskaran has designed large and complex data systems for some of the globally most recognized companies and brands. He has proven expertise in directing successful enterprise level Big Data, Data Science, AI/ML, Business Intelligence/Analytics digital platforms.

Currently, Jaskaran serves as the CTO of Pepper, Inc. He is responsible for defining and executing Pepper's product strategy, and leads a global team involved in designing, developing, testing, deploying, and supporting Pepper's software stack. He is a big proponent of ongoing continuous improvements of processes, practices, and capabilities. His prior work experience include Jack Henry & Associates, Active Network and JCPenney

Jaskaran earned his master's degree in computer science from Baylor University and holds various technology certifications. When not working, Jaskaran spends most of his time reading, cooking, and gardening. An admitted sports fanatic, he feeds his addiction to football by watching Dallas Cowboys games on Sunday afternoons.

Pepper is full stack platform provider to Investment Managers. Pepper enables clients by delivering and managing mission-critical data on a flexible, and secure infrastructure platform. Pepper solves enterprise complexity by seamlessly and comprehensively integrating data with the speed and scale necessary to translate real insights into profitable actions for Asset Managers. With Pepper, Asset Managers are enabled to make the best decisions they can.

Pepper has consistently believed in the power of Cloud Native applications.