Available online www.ejaet.com

European Journal of Advances in Engineering and Technology, 2021, 8(3):48-58

Research Article ISSN: 2394 - 658X

Cloud Computing: Strategies for Data Storage, Containerization, CI/CD, and Testing in Cloud Environments

Maheswara Reddy Basireddy

Maheswarreddy.basireddy@gmail.com

ABSTRACT

Because cloud computing offers cost-effectiveness, scalability, and on-demand access to computer resources, it has completely changed how businesses handle these resources. The data storage services provided by the main cloud platforms, containerization technologies for application deployment, continuous integration and deployment (CI/CD) procedures, and testing and maintenance techniques in cloud environments are just a few of the aspects of cloud computing strategies that are examined in this research paper. The first section of the article evaluates and discusses the capabilities, use cases, and selection criteria of the database and storage services offered by Microsoft Azure, Amazon Web Services (AWS), and Google Cloud Platform (GCP). After that, it explores containerization technologies like Docker and Kubernetes, which make it easier to package, launch, and manage cloud applications. In addition, the study looks at best practices, technologies, and ideas related to continuous integration and delivery (CI/CD), highlighting the value of automating software delivery pipelines for accelerated time-to-market, enhanced teamwork, and dependable deployments. It also looks at different approaches, techniques, tools, logging, monitoring, scalability issues, and maintenance procedures specific to cloud-based apps.

Keywords: Performance Testing, Load Testing, Python, AI, Automation, Locust, pytest-benchmark, JMeter, Sentry, New Relic, Monitoring, Issue Detection, Test Planning, Test Execution, Test Analysis, Performance Optimization, Scalability, Reliability, Bottleneck Identification, Intelligent Testing, Machine Learning, Cloud Computing, Distributed Testing.

---

1. INTRODUCTION

A. Background and Motivation

The way businesses handle application deployment and computing resources has been completely transformed by cloud computing. Businesses may obtain on-demand computing power, storage, and services via the internet by utilizing the cloud, eliminating the need for substantial upfront expenditures in hardware and infrastructure. Organizations can now expand their resources dynamically, cut operating expenses, and adapt more quickly to shifting business demands thanks to this paradigm change.

The explosion of big data and the Internet of Things (IoT), the need for affordable and scalable solutions, and the growing need for digital transformation have all contributed to the acceptance of cloud computing. The need for efficient methods and best practices to handle data storage, launch apps, set up continuous integration and deployment (CI/CD) pipelines, and guarantee reliable testing and maintenance procedures is increasing as more companies move their workloads and applications to the cloud.

B. Objectives and Scope

With an emphasis on data storage services, containerization technologies, continuous integration/continuous development processes, and testing/maintenance methodologies in cloud settings, this research study attempts to offer a thorough examination of cloud computing tactics. The following are the goals of this study:

- Analyze the database and data storage capabilities offered by the main cloud computing platforms, such as Microsoft Azure, Google Cloud Platform, and Amazon Web capabilities (AWS).
- Examine containerization technologies and how they help with cloud application deployment and management, such as Docker and Kubernetes.
- Investigate CI/CD principles, instruments, and optimal methodologies to mechanize software supply chains and guarantee dependable and effective application implementation.
- Examine testing approaches, techniques, and resources for cloud-based apps together with logging, monitoring, scaling, and maintenance factors.
- Give a real-world case study or implementation example to illustrate how the technology and concepts being presented may be used.

The database and storage services provided by AWS, GCP, and Azure, as well as the deployment and administration of containerized apps via CI/CD pipelines and testing/maintenance procedures in cloud environments, are all included in the purview of this research.

C. Organization of the Paper

The structure of the paper is as follows: Cloud platforms and data storage services, such as those provided by AWS, GCP, and Azure, are covered in Section 2. With an emphasis on Docker and Kubernetes, Section 3 examines containerization technologies. Section 4 looks at best practices, tools, and concepts related to CI/CD. The study is finally concluded in Section 5, which includes a review of the results, limits, and future work.

2. CLOUD PLATFORMS AND DATA STORAGE SERVICES

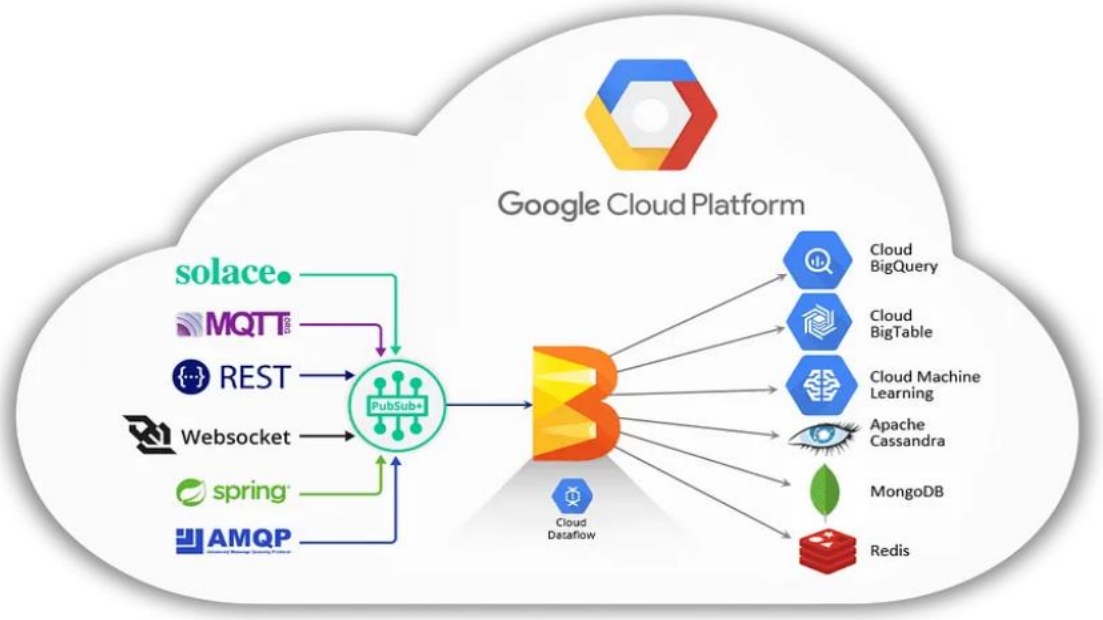
A. Amazon Web Services (AWS)

1. AWS Storage Services (S3, EBS, EFS, etc.), To satisfy different needs for data management and storage, Amazon Web options (AWS) provides a full range of storage options. Simple Storage Service (S3) is one of the most popular services; it offers highly scalable, durable, and available object storage. Large volumes of unstructured data, including pictures, videos, backups, and static website content,

may be stored and retrieved using S3. Elastic Block Store (EBS), which offers permanent block-level storage volumes for use with Amazon Elastic Compute Cloud (EC2) instances, is another significant storage solution. Applications requiring frequent read and write operations can benefit from high-performance and low-latency storage provided by EBS volumes, which can be network-attached to EC2 instances as storage. Elastic File System (EFS), a fully managed, scalable file storage solution that allows easy interaction with AWS computing resources, is another product that AWS offers. EFS is made for applications like content management systems, web servers, and large data analysis that need shared access to file data.

2. AWS Database Services (RDS, DynamoDB, etc.), AWS offers a variety of database services in addition to storage services to meet various needs for data retrieval and storage. The managed relational database service known as AWS Relational Database Service (RDS) is compatible with several database engines, such as MySQL, PostgreSQL, Oracle, SQL Server, and Amazon Aurora, which is AWS's own database. RDS makes database management chores like scalability, provisioning, patching, and backups easier, freeing up developers to concentrate on creating applications. AWS offers Amazon DynamoDB, a fully managed serverless NoSQL database solution that enables smooth scalability and fast, predictable performance, to meet your needs for NoSQL databases. Applications like gaming, IoT, and mobile apps that need low-latency data access can benefit from DynamoDB's ability to manage large workloads.

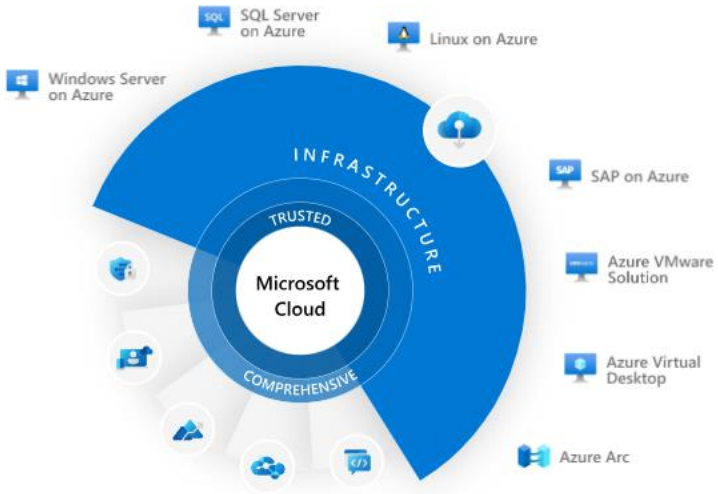
B. Google Cloud Platform (GCP)



1. GCP Storage Services (Cloud Storage, Persistent Disk, etc.), To accommodate a range of data storage requirements, Google Cloud Platform (GCP) provides a number of storage services. Website content, backups, and big data analytics can all be safely and affordably stored using Google Cloud Storage, an extremely scalable and resilient object storage solution. Google Compute Engine (GCE) virtual machine instances may utilize high-performance persistent disks thanks to GCP's Persistent Disk block storage solution. High-performance and low-latency data access is made possible by the ability to connect Persistent Disks to GCE instances as network-attached storage. Other storage services provided by GCP include Cloud Bigtable, a NoSQL big data analytics database service intended for workloads requiring low latency and high throughput, and Cloud Filestore, a fully managed file storage solution for corporate applications.
2. GCP Database Services (Cloud SQL, Cloud Bigtable, etc.), In order to accommodate different needs for data storage and retrieval, GCP offers a variety of database services. Relative database services like Cloud SQL are completely managed and support well-known database engines including MySQL,

PostgreSQL, and SQL Server. By making database maintenance duties simpler using Cloud SQL, developers can concentrate on creating applications. Cloud Bigtable, a fully managed NoSQL big data analytics database service intended for low-latency and high-throughput applications, is provided by GCP as a solution for NoSQL database needs. Applications like IoT, financial data analysis, and user analytics that need real-time data access can benefit from Cloud Bigtable.

C. Microsoft Azure



1. Azure Storage Services (Blob Storage, File Storage, etc.), To meet a variety of data storage needs, Microsoft Azure provides a number of storage services. A highly scalable object storage option for unstructured data, including pictures, videos, backups, and logs, is Azure Blob Storage. Blob storage offers safe data transmission, robustness, and high availability. You can build and access file shares from Windows, Linux, and macOS with Azure File Storage, a fully managed file sharing solution. It offers a recognizable Server Message Block (SMB) protocol interface for easy compatibility with current tools and applications. Azure provides additional storage services as well, including Data Lake Storage for big data analytics applications, Archive Storage for affordable long-term data preservation, and Disk Storage for permanent block-level storage.
2. Azure Database Services (SQL Database, Cosmos DB, etc.), A variety of database services are offered by Microsoft Azure to satisfy different demands for data storage and retrieval. With Azure SQL Database, developers can create and launch apps without worrying about infrastructure administration. Azure SQL Database is a fully managed relational database service that supports Microsoft SQL Server. Azure provides Cosmos DB, a globally distributed multi-model database service that supports key-value, document, graph, and column-family data models, to meet NoSQL database needs. Cosmos DB is appropriate for globally dispersed applications and real-time data processing because it has low latency, high availability, and automated indexing.

D. Comparison and Selection Criteria

Some things to think about while choosing a cloud platform and related database and storage services are as follows:

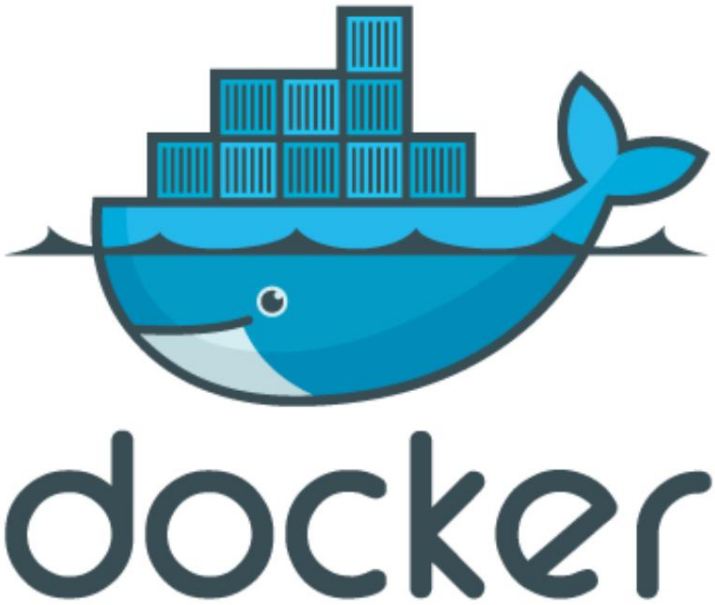
- **Performance and scalability**: Assess the scalability and performance needs of your applications and select services that can effectively manage your workloads and adjust to your needs as they arise.
- **Data availability and durability**: Evaluate the assurances offered by each provider to make sure your data is available when you need it and is safe from outages.
- **Cost:** Evaluate the various cloud service providers' and providers' pricing models and cost structures, accounting for compute resources, data transport, and storage use.

- **Integration and compatibility:** Take into account how well your current tools and apps integrate with the cloud platform and services of your choice.
- **Security and compliance:** To make sure that your data and apps satisfy the security and regulatory standards of your firm, assess the security features, data encryption choices, and compliance certifications provided by each cloud provider.
- **Lock-in of vendors:** Examine the ease of data and application migration between cloud providers and on-premises systems, as well as the possible hazards associated with vendor lock-in.

You are able to choose the cloud platform and services that best fit your needs by taking these variables into account and ensuring that they are in line with the specifications of your particular project.

3. CONTAINERIZATION AND DEPLOYMENT

A. Docker



1. Docker Containers, Within the cloud computing environment, Docker is an open-source containerization technology that has become widely used. Code, runtime, system tools, libraries, settings, and other components required to run an application are all included in Docker containers, which are small, standalone executable packages. Because these containers are sealed off from both the host operating system and other containers, consistent and repeatable application behavior is guaranteed in a variety of settings.

   Among the advantages of utilizing Docker containers are:

- Portability: Docker containers solve the "it works on my machine" issue by enabling consistent operation across a variety of platforms and contexts, including development, testing, and production.
- Isolation: Every Docker container operates in a separate environment, which minimizes resource usage and avoids conflicts between apps.
- Lightweight and effective: Because containers share the kernel of the host operating system and use fewer resources than traditional virtual machines, they are more effective and lightweight than virtual machines.
- Scalability: By spinning up more container instances, Docker containers may be readily expanded horizontally, allowing for effective load balancing and resource allocation.
- Consistency: Applications operate consistently across many environments thanks to Docker containers, which isolate the complete application environment and minimize compatibility and deployment problems.

2. Docker Compose, A tool called Docker Compose makes it easier to define and manage Docker applications with many containers. Known as a Docker Compose file, it enables developers to specify

and set up every service needed for an application in one file. The services, dependencies, networking setups, and other configurations needed to operate the program are listed in this file.

By offering a number of advantages, Docker Compose makes containerized application deployment and administration easier:

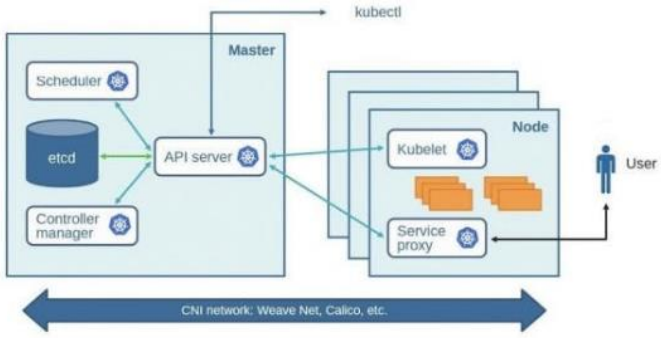
- **Simplified configuration**: Compose files allow developers to define and configure all services and their dependencies in a single location, making it easier to manage and maintain complex applications.
- **One-line deployment:** With a single command, Docker Compose can spin up all the required services and their dependencies, simplifying the deployment process.
- **Service orchestration:** Docker Compose orchestrates the startup and shutdown of services in the correct order, ensuring that dependencies are met and services are started or stopped gracefully.
- **Environment consistency:** By defining the entire application stack in a Compose file, developers can ensure consistent environments across different deployment targets, such as development, staging, and production.

3. Docker Swarm, A built-in clustering and orchestration tool for Docker containers is called Docker Swarm. It makes it possible for developers to set up and maintain a cluster of Docker nodes, which facilitates the deployment and administration of containerized applications on several hosts. Features like load balancing, service discovery, scalability, and high availability are offered by Docker Swarm.

Among the main advantages of utilizing Docker Swarm are:

- **Scalability**: By adding or deleting nodes from the cluster, Docker Swarm enables developers to grow their applications horizontally, facilitating effective load balancing and resource usage.
- **High availability:** Docker Swarm provides fault tolerance and failover capabilities by distributing containers over many nodes, guaranteeing that applications continue to function even in the event of a node failure.
- **Service discovery and load balancing:** Docker Swarm comes with built-in features for both service discovery and load balancing, which let containers interact with one another and split traffic among several instances.
- **Declarative configuration:** By enabling developers to use declarative configuration files to specify the intended state of their applications, Docker Swarm streamlines deployment and administration.
- **Docker Swarm:** Docker Swarm's smooth integration with other Docker tools, such Docker Compose, facilitates the management and deployment of multi-container applications in a production setting.

B. Kubernetes



1. Kubernetes Architecture, Deploying, scaling, and maintaining containerized applications in cloud settings is now done de facto using Kubernetes, an open-source container orchestration framework. The Kubernetes architecture is made up of a number of parts that combine to offer a stable and expandable container management platform.

The following are the key elements of the Kubernetes architecture:

- The Control Plane, sometimes known as the cluster's brain, is in charge of scheduling and resource allocation in addition to maintaining the cluster's state.

a) **kube-apiserver:** The main module that acts as the control plane's front end and provides access to the Kubernetes API.
b) **etcd:** A distributed key-value store that keeps track of the status and configuration information for the cluster.
c) **Kube-scheduler:** In charge of planning and allocating pods, or collections of containers, to the proper nodes in accordance with resource limitations and requirements.
d) **kube-controller-manager:** This program manages the replication, node, and service account controllers, among other controllers that control the cluster's state.
e) **Cloud-controller-manager:** Coordinates with cloud providers' APIs and oversees resources unique to the cloud.

- **Worker Nodes**: The physical or virtual computers that operate the containerized applications are known as worker nodes.

a) **kubelet**: An agent that manages pods and makes sure they function as intended, running on every node.
b) **Kube-proxy**: A network proxy that makes it easier for pods and outside services to communicate with one another.
c) **Container Runtime**: The program, such as Docker or containerd, that operates and maintains containers on the node.

- **Kubectl**: The command-line utility for managing the resources of the Kubernetes cluster and interacting with it.

2. Kubernetes Objects (Pods, Services, Deployments, etc.), A containerized application's numerous components may be managed and represented using a variety of objects and abstractions offered by Kubernetes. Among these items are:

- **Pods** are the simplest and most fundamental units in Kubernetes; they stand for one or more containers that share a network or storage system. Pods are not meant to last a long time; they are transient.
- **Services**: Services give a group of pods load balancing capabilities and a reliable network endpoint. They separate the frontend clients from the backend pods by acting as an abstraction layer.
- **Deployments**: Including the quantity of replicas, update tactics, and other configuration specifics, deployments specify the intended state of an application. They oversee the pod lifespan and make sure the intended state is upheld.
- **ConfigMaps and Secrets:** These variables may be injected into pods as environment variables or mounted volumes, and they hold configuration data and confidential information, respectively.
- Containers within of pods are stored in volumes, while persistent storage is stored in volumes that can last longer than individual pods or nodes.
- **Namespaces:** To improve organization and resource management, namespaces are utilized to divide and isolate resources inside a Kubernetes cluster.

These objects offer a strong and adaptable method for managing and orchestrating containerized applications in Kubernetes, in conjunction with other structures like labels, annotations, and selectors.

3. Kubernetes Cluster Management, Numerous responsibilities, including providing nodes, scaling resources, keeping an eye on the cluster's health, and applying changes or settings, are involved in managing a Kubernetes cluster.

   Key components of Kubernetes cluster management include the following:

- **Cluster Setup & Provisioning**: Kubernetes may be implemented on a range of platforms, such as on-premises infrastructure, public clouds (AWS, GCP, Azure), or hybrid environments. Provisioning the control plane and worker nodes, setting up secure communication channels, and configuring networking are all part of the setup procedure.
- **Resource management and scaling:** Kubernetes comes with built-in tools for managing resources both vertically (by changing the resource limitations for pods) and horizontally (by adding or deleting nodes). Based on workload needs, autoscaling features can automatically modify resource allocation.
- **Monitoring and Logging:** To guarantee optimal operation and spot possible problems, it's essential to keep an eye on the performance and health of a Kubernetes cluster. Kubernetes interfaces with

Elasticsearch/Fluentd/Kibana (EFK) stack, Prometheus, and Grafana, among other monitoring and logging technologies.

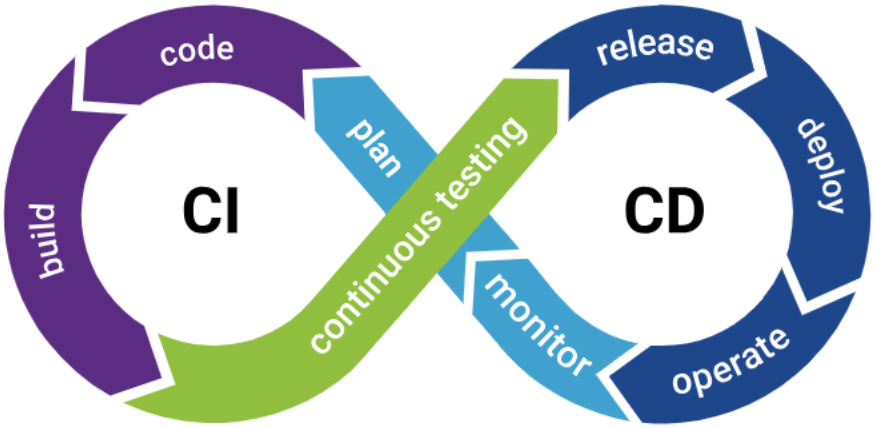
- **Updates and Upgrades:** Kubernetes releases new features and enhancements on a regular basis. Cluster administrators must carefully plan and carry out updates, adhering to best practices, in order to reduce downtime and guarantee application compatibility.
- **Security and Access Control:** Role-based access control (RBAC), network policies, and secrets management are just a few of the strong security features that Kubernetes offers. To protect the cluster and applications, administrators need to make sure these elements are configured correctly.
- **Backup and Disaster Recovery:** To safeguard application data and maintain business continuity, a backup and disaster recovery plan must be put into place. A variety of backup and restoration techniques, such as etcd backups, persistent volume backups, and snapshots, are supported by Kubernetes.

Utilizing the platform's built-in tools, integrating third-party solutions, and adhering to best practices are all necessary for effective Kubernetes cluster administration in order to guarantee the dependable and effective functioning of containerized applications in a cloud environment.

4. CONTINUOUS INTEGRATION AND DEPLOYMENT (CI/CD)

A. CI/CD Concepts and Benefits

In order to expedite software delivery, enhance teamwork, and guarantee high-quality releases, current software development approaches like as continuous integration (CI) and continuous deployment/delivery (CD) are crucial. Cloud computing environments, where applications are routinely deployed and updated to suit quickly changing business demands, are particularly important contexts for CI/CD techniques.



Continuous Integration (CI)

Software developers use a process called continuous integration (CI) to regularly merge code changes from several developers into a single repository. An automated build process is started whenever a developer contributes changes to the code. It then compiles the code, runs unit tests, and looks for integration problems. By detecting and fixing problems early in the development cycle, continuous integration (CI) lowers the chance of introducing defects and facilitates problem identification and resolution.

Continuous Deployment/Delivery (CD):

Building upon Continuous Integration, Continuous Deployment (CD) and Continuous Delivery (CD) are closely related ideas. The process of automatically sending each code update to a production environment once it passes automated tests and inspections is known as continuous deployment. However, manual approval is usually needed prior to the actual deployment. Continuous Delivery, on the other hand, entails automatically creating, testing, and preparing code modifications for deployment to production.

The benefits of adopting CI/CD practices include:

- Faster time-to-market: New features and upgrades may be released to clients more rapidly, giving businesses a competitive edge. This is made possible by automating the development, test, and deployment processes.

- Better software quality is ensured by frequent code integration, thorough automated testing, and early problem detection and resolution. These measures also lower the likelihood of new defects being introduced into the product.
- Enhanced productivity and teamwork: Continuous Integration/Continuous Delivery (CI/CD) encourages developers to work together more often by allowing them to incorporate changes to their code without worrying about disrupting the application.
- Decreased risk and downtime: By minimizing the possibility of releasing flawed code into production, automated testing and rollback procedures help to cut down on risk as well as potential commercial consequences.
- Deployments that are uniform and reproducible across many settings are made possible by CI/CD pipelines, which also lower the possibility of environmental variations and make troubleshooting easier.

B. CI/CD Tools

In cloud settings, CI/CD pipelines may be implemented and managed using a variety of tools and platforms. Among the widely used CI/CD tools are:

1. **Jenkins**

   Jenkins is an open-source automation server that works with many technology stacks and programming languages to provide CI/CD pipelines. It offers a web interface for development, test, and deployment process configuration and management. Jenkins is quite versatile and flexible to various project requirements thanks to its extensive selection of plugins and integrations. Applications can be seamlessly deployed to cloud environments because to its easy integration with cloud platforms like AWS, GCP, and Azure.
2. **CI Travis**

   A popular code repository like GitHub may be easily integrated with Travis CI, a cloud-based continuous integration and delivery technology. It offers a quick and easy method for building, testing, and launching apps straight from GitHub sources. Travis CI can be set up to deploy apps to AWS, GCP, and Azure, among other cloud platforms. It also supports a multitude of programming languages.
3. **Actions on GitHub**

   A CI/CD tool called GitHub Actions is built right into GitHub repositories. Developers may design unique processes for creating, evaluating, and releasing apps straight from their GitHub repositories. GitHub Actions is a potent tool for automating software delivery pipelines since it supports a large number of programming languages and is simple to integrate with different cloud platforms and services.

C. CI/CD Pipelines and Best Practices

The steps of a CI/CD pipeline usually include source control management, code development, testing, deployment, and monitoring. The following are some recommended practices for creating and putting into use efficient CI/CD pipelines:

1. Version control: To efficiently manage source code and track changes, use a powerful version control system like Git.
2. Automated testing: To detect problems early and guarantee software quality, implement thorough automated testing at several levels (unit, integration, and end-to-end).
3. Continuous integration (CI): Create a process that, upon each new commit being submitted to the repository, automatically builds and tests the code.
4. Strategies for deployment: Use deployment techniques like rolling updates, canary releases, and blue/green deployments to cut down on downtime and lower the chance of bringing issues into production systems.
5. Environment parity: To prevent disparities in the environments and guarantee seamless deployments, keep development, staging, and production environments uniform.
6. Monitoring and logging: To track application performance, identify problems, and facilitate efficient troubleshooting, provide strong monitoring and logging capabilities.

7. Feedback loops: To guarantee process improvement and alignment with company goals, create feedback loops including the development, operations, and business teams.
8. Automated rollbacks: In the event of deployment problems or malfunctions, put automated rollback procedures in place to promptly roll back to a prior stable version.
9. Code as Infrastructure: Adopt Infrastructure as Code (IaC) techniques to ensure repeatable and consistent deployments by defining and managing cloud infrastructure using declarative code.
10. Security and compliance: To make sure that applications adhere to legal and security standards, incorporate compliance validations and security checks into the CI/CD pipeline.

Through adherence to these recommended practices, enterprises may optimize their software delivery procedures, enhance teamwork, and provide superior apps to cloud environments with more efficiency and dependability.

5. CONCLUSION

A. Summary of Findings

The way businesses handle software distribution, application deployment, and computing resources has changed dramatically as a result of cloud computing. Organizations may gain from scalability, cost-effectiveness, and agility in reacting to quickly changing business demands by utilizing cloud platforms.

This study examined several methods and best practices for CI/CD pipeline implementation, application deployment, data storage management, and guaranteeing reliable testing and maintenance procedures in cloud settings. Important conclusions and revelations from this study include:

- **Cloud platforms and data storage services**: To fulfill a variety of data storage and retrieval needs, major cloud providers such as AWS, GCP, and Azure provide a wide range of storage and database services. Evaluating aspects like pricing, performance, scalability, and project needs is necessary to choose the right services.
- **Containerization and Deployment:** The packaging, deployment, and management of programs in cloud settings have been completely transformed by containerization technologies such as Docker and Kubernetes. While Kubernetes offers a reliable and scalable framework for coordinating and managing containerized applications, Docker streamlines the construction and administration of containers.
- **Continuous Integration and Deployment (CI/CD):** Implementing CI/CD techniques is crucial to guaranteeing high-quality releases, optimizing software delivery, and enhancing teamwork. Build, test, and deployment procedures may be automated with the help of tools like Travis CI, GitHub Actions, and Jenkins. Automated testing, deployment plans, and feedback loops are other best practices that help ensure dependable and efficient application delivery.
- **Testing and Maintenance in Cloud Environments:** To guarantee the quality and dependability of cloud-based systems, testing techniques, tools, and strategies are essential. Applications must be kept up to date with the newest security patches and feature upgrades in order to maintain performance and discover problems. This requires effective monitoring, logging, scalability testing, and maintenance procedures.

Organizations can harness the power of cloud computing while efficiently managing data storage, deploying apps, automating software delivery pipelines, and guaranteeing strong testing and maintenance procedures in cloud environments by implementing the strategies and best practices covered in this research paper.

B. Limitations and Future Work

- Scope: The main cloud platforms (AWS, GCP, and Azure) and the services that go along with them were the main focus of this study. But there are further cloud service providers and cutting-edge technologies (such edge computing and serverless computing) that deserve further investigation.
- Applications Native to the Cloud: The study addressed deployment tactics and containerization, but it might have gone into greater detail about the fundamentals of cloud-native application design, microservices architectures, and associated patterns and practices.
- Hybrid and multi-cloud environments: As businesses implement hybrid and multi-cloud strategies, they must investigate the best methods for coordinating and administering applications across various cloud providers and on-premises infrastructure.

Ongoing study and investigation will be necessary to keep up with new advances, handle new issues, and maximize the acceptance and use of cloud technologies across a variety of businesses and areas as cloud computing continues to expand and become more widespread.

REFERENCES

[1]. Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., ... & Zaharia, M. (2010). A view of cloud computing. Communications of the ACM, 53(4), 50-58.

[2]. Sharma, S., & Kalra, S. (2020). Cloud computing: A practical approach. Manning Publications.

[3]. Humble, J., & Farley, D. (2010). Continuous delivery: Reliable software releases through build, test, and deployment automation. Addison-Wesley Professional.

[4]. Fowler, M., & Lewis, J. (2014). Microservices: a definition of this new architectural term. https://martinfowler.com/articles/microservices.html

[5]. Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. Linux Journal, 2014(239), 2.

[6]. Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). Borg, omega, and kubernetes. Communications of the ACM, 59(5), 50-57.

[7]. Duvall, P. M., Matyas, S., & Glover, A. (2007). Continuous integration: improving software quality and reducing risk. Pearson Education.

[8]. Parnin, C., Helbling, C., Atlee, C., Boughton, H., Ghattas, M., Glover, A., ... & Williams, L. (2017). The continuous integration context: software engineering solution practices discovered from qualitative studies. In Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track (pp. 421-430). IEEE Press.

[9]. Lyu, M. R. (1996). Software fault tolerance. John Wiley & Sons, Inc..

[10]. Jiang, Z. M., Hassan, A. E., Hamann, G., & Flora, P. (2008). An automated approach for abstracting execution logs to execution events. Journal of Software Maintenance and Evolution: Research and Practice, 20(4), 249-267.