1 Introduction to DevOps

UNIT - I

Syllabus

What is DevOps? Role of DevOps Engineer, Developer responsibility, Introduction to Continuous Integration and Continuous Delivery Policies, DevOps Culture: Dilution of barriers in IT departments, Process automation, Agile Practices, Reason for adopting DevOps, What and Who Are Involved in DevOps? Changing the Coordination, Introduction to DevOps pipeline phases , Defining the Development Pipeline, Centralizing the Building Server, Monitoring Best Practices, Best Practices for Operations.

1.1 What is DevOps?

- DevOps is the combination of Development (Dev) and operations (Ops) which is an umbrella term that describes the operation of a team collaborating throughout an entire programming production process - from the design through the development stages. It's a combination of tools and philosophies that increase a team's capability to produce results at high efficiency.
- Devops allows a single team to handle the entire application lifecycle, from development to testing, deployment, and operations. DevOps helps you to reduce the disconnection between software developers, quality assurance (QA) engineers, and system administrators.
- Devops helps organizations to increase the speed of delivering the software or applications and services. It also allows organizations to serve their customers better and compete more strongly in the market. DevOps has become one of the most valuable business disciplines for enterprises or organizations. With the help of DevOps, quality, and speed of the application delivery has improved to a great extent.

Why DevOps?

We need to understand why we need DevOps over the other methods.

- The development (Dev) and operations (Ops) team worked in complete isolation.

- Without the use of DevOps, the team members are spending a large amount of time on designing, testing, and deploying instead of building the project.
- After the design-build, the testing and deployment are performed respectively. That's why they consumed more time than actual build cycles.
- Manual code deployment leads to human errors in production.
- Coding and operation teams have their separate timelines and are not in synch, causing further delays.

1.2 Role of DevOps Engineer

- DevOps culture is introduced to build better communication, improved collaboration, and agile relation between the software development team and Operations team. Typically, the role of a DevOps engineer is not as easy as it appears. It involves looking into seamless integration among the teams, successfully and continuously deploying the code.
- A DevOps engineer introduces processes, tools, and methodologies to balance needs throughout the software development life cycle, from coding and deployment, to maintenance and updates.
- DevOps engineer is one of the most challenging roles and often organizations find it difficult to find an efficient DevOps engineer. A DevOps engineer must have a strong passion for scripting and coding, has expertise in handling deployment automation, infrastructure automation and ability to handle the version control.
- DevOps engineers must to have a knowledge about server less computing such as version control, testing, integration and deployment methods. A DevOps engineer is subject to face continuous challenges when it comes to server less integration, deployment, technology and incident management. They need to have excellent complex solving skills, must have the ability to think out of the box, and curious.
- It's important to understand that a DevOps engineer is formed out of the growing needs of the business to get a better hold of the cloud infrastructure in a hybrid environment. Organizations implementing DevOps skills yield better advantages such as spend relatively spend less time on configuration management, deploy application faster and frequently.
- DevOps Engineer is responsible for handling the IT infrastructure as per the business needs of the code which can be deployed in a hybrid multi-tenant environment which needs continuous monitoring of the performance. DevOps engineer must be aware of the development tools which write the new code or enhance the existing code.

- In DevOps, there is more scope for frequent changes in the code, which includes continuous automating, and deployment. It's not expected to write the code right from scratch but choosing the right combination of coding, how to integrate several elements of SQL data is important as a part of DevOps engineer role
- A DevOps engineer has to associate with the team to handle the challenges arising in the coding or scripting part which includes libraries and software development kits to run the software on various OS and for deployment.
- DevOps engineer has to handle code which has to fit across multi-tenant environments including cloud. Hence a DevOps engineer role is more of a cross-functional role which manages and handles software that's built and deployed across challenging applications.

1.3 Developer Responsibility

DevOps engineers need to be able to multitask, demonstrate flexibility, and deal with many different situations at a time. Specifically, a DevOps engineer's responsibilities include:

- **Documentation** : Writes specifications and documentation for the server-side features.
- **Systems analysis** : Analyzes the technology currently being used and develops plans and processes for improvement and expansion. The DevOps engineer provides support for urgent analytic needs.
- **Development** : Develops, codes, builds, installs, configures, and maintains IT solutions.
- **Project planning** : Participates in project planning meetings to share their knowledge of system options, risk, impact, and costs vs. benefits. In addition, DevOps engineers communicate operational requirements and development forecasts.
- **Testing** : Tests code, processes, and deployments to identify ways to streamline and minimize errors.
- **Deployment** : Uses configuration management software to automatically deploy updates and fixes into the production environment.
- **Maintenance and troubleshooting** : Performs routine application maintenance to ensure the production environment runs smoothly. Develops maintenance requirements and procedures.
- **Performance management** : Recommends performance enhancements by performing gap analysis, identifying alternative solutions, and assisting with modifications.

1.4 Introduction to Continues Integration and Continues Delivery Policies

Continuous integration (CI) and continuous delivery (CD) are the processes that are used to build, package, and deploy your application. Basically, it lays out some practices to follow in order for the code you write to more quickly and safely get to your users and ultimately generate value.

1.4.1 Continuous Integration

- **Continuous integration** is the process of automating the integration of code changes from multiple contributors in a software project. This extends beyond development teams to the rest of the organization.
- For example, product teams coordinate when to sequentially launch features and fixes and which team members will be responsible. It's a primary DevOps best practice, allowing developers to frequently merge code changes into a central repository where builds and tests then run. Automated tools are used to assert the new code's correctness before integration.

A. The importance of continuous integration

- In order to understand the importance of CI, it's helpful to first discuss some pain points that often arise due to the absence of CI. Without CI, developers must manually coordinate and communicate when they are contributing code to the end product. This coordination extends beyond the development teams to operations and the rest of the organization. Product teams must coordinate when to sequentially launch features and fixes and which team members will be responsible.
- The communication overhead of a non-CI environment can become a complex and entangled synchronization chore, which adds unnecessary bureaucratic cost to projects. This causes slower code releases with higher rates of failure, as it requires developers to be sensitive and thoughtful towards the integrations. These risks grow exponentially as the engineering team and codebase sizes increase.
- Without a robust CI pipeline, a disconnect between the engineering team and the rest of the organization can form. Communication between product and engineering can be cumbersome. Engineering becomes a black box which the rest of the team inputs requirements and features and maybe gets expected results back. It will make it harder for engineering to estimate time of delivery on requests because the time to integrate new changes becomes an unknown risk.

B. Getting started with continuous integration

- The foundational dependency of CI is a version control system (VCS. Some popular VCSs are Git, Mercurial, and Subversion.
- Once version control is in place, finding a version control hosting platform is the next move. Most modern version control hosting tools have support and features built in for CI. Some popular version control hosting platforms are Bitbucket, Github, and Gitlab.
- After version control has been established on the project, integration approval steps should be added. The most valuable integration approval step to have in place is automated tests. Adding automated tests to a project can have an initial cost overhead. A testing framework has to be installed, then test code and test cases must be written by developers.
- Some ideas for other, less expensive CI approval mechanisms to add are syntax checkers, code style formatters, or dependency vulnerability scans.

1.4.2 Continues Delivery

- **Continuous delivery** is an organizational methodology that brings together engineering and non-engineering teams like design, product, and marketing to deliver a product. Environments without CD encourage "over the wall" behavior where developers focus on the QA team as the primary user experience. It means the "trunk" branch of your repository is in a "deployable" state at all times.
- Continuous deployment allows code changes to be automatically deployed into production when they are made, either hidden behind a feature flag, deployed to a small percentage of customers, and/or easily rolled back. This gives teams greater flexibility to respond to changing markets and customer demands since teams can react to customer feedback and rapidly deploy and validate new features. They can also easily rollback features, allowing teams not to be hampered by breaking the build.
- Continuous deployment, is embraced and promoted by such large organizations as Netflix but not commonly adopted (or required) in most smaller companies. This is because continuously deploying new features into a production environment requires a high degree of confidence that new code has been thoroughly tested and can be deployed safely (e.g. behind a feature toggle). So, unless your organization deploys many times a day it may not be worth investing in the processes that support this approach.

1.5 DevOps Culture : Dilution of Barriers in IT Departments

- DevOps is an agile approach to organizational change that seeks to bridge traditional, siloed divides between teams and establish new processes that facilitate greater collaboration.
- While DevOps is made possible by new tools and agile engineering practices, these are not enough to gain the benefits of DevOps. Without the right mindset, rituals, and culture, it's hard to realize the full promise of DevOps.

1.5.1 What is DevOps Culture?

- At its essence, a DevOps culture involves closer collaboration and a shared responsibility between development and operations for the products they create and maintain. This helps companies align their people, processes, and tools toward a more unified customer focus.
- It involves cultivating multidisciplinary teams who take accountability for the entire lifecycle of a product. DevOps teams work autonomously and embrace a software engineering culture, workflow, and toolset that elevates operational requirements to the same level of importance as architecture, design and development. The understanding that developers who build it, also run it, brings developers closer to the user, with a greater understanding of user requirements and needs. With operations teams more involved in the development process, they can add maintenance requirements and customer needs for a better product.
- At the heart of DevOps culture is **increased transparency, communication, and collaboration** between teams that traditionally worked in siloes. But there are important cultural shifts that need to happen to bring these teams closer together. DevOps is an organizational culture shift that emphasizes continuous learning and continuous improvement, especially through team autonomy, fast feedback, high empathy and trust, and cross-team collaboration.
- DevOps entails **shared responsibilities.** Development and operations staff should both be responsible for the success or failure of a product. Developers are expected to do more than just build and hand off to operations — they are expected to share the responsibility of overseeing a product through the entire course of its lifetime, adopting a "you build it, you run it" mentality. They test and operate software and collaborate more with QA and IT Ops. When they understand the challenges faced by operations, they are more likely to simplify deployment and maintenance. Likewise, when operations understand the system's business goals, they can work with developers to help define the operational needs of a system and adopt automation tools.

- **Autonomous teams** are another important aspect of DevOps. For the development and operations teams to collaborate effectively, they need to make decisions and implement changes without a cumbersome and lengthy approval process. This involves handing over trust to teams and establishing an environment where there is no fear of failure. These teams should have the right processes and tools in place to make decisions faster and easier, for each level of risk to the customer.
- For example, a typical development workflow might require engagement from several contributors on different teams to deploy code changes. The developer makes a code change and pushes it to a source control repository, a build engineer builds and deploys the code to a test environment, a product owner updates the status of the work in an issue tracking tool, etc. An autonomous team will take advantage of tools that automate these processes, so that pushing new code triggers the building and deployment of a new feature into a test environment and the issue tracking tool is updated automatically.
- A DevOps team culture values fast **feedback** that can help with continuous improvement of a unified development and operations team. In an environment where the development and operations teams are in isolated silos, feedback about the performance and stability of application software in production is often slow to make it back to the development team, if it makes it at all.
- DevOps ensures that developers get the fast feedback they need to rapidly iterate and improve on application code by requiring collaboration between operations folks in designing and implementing application monitoring and reporting strategies. For example, any sufficiently capable continuous integration tool will enable automated build and testing of new code pushes and provide the developer with immediate feedback on the quality of their code.
- **Automation** is essential to DevOps culture, since it allows great collaboration and frees up resources. Automating and integrating the processes between software development and IT teams helps them to build, test, and release software faster and more reliably.

1.5.2 Benefits of DevOps

- The value of DevOps is big. Nearly all (99%) of respondents said DevOps has had a positive impact on their organization. Teams that practice DevOps ship better work faster, streamline incident responses, and improve collaboration and communication across teams.

- Teams who fully embrace DevOps practices work smarter and faster, and deliver better quality to their customers. The increased use of automation and cross-functional collaboration reduces complexity and errors, which in turn improves the Mean Time to Recovery (MTTR) when incidents and outages occur.
- **Collaboration and trust :** Building a culture of shared responsibility, transparency, and faster feedback is the foundation of every high-performing DevOps team. In fact, collaboration and problem-solving ranked as the most important elements of a successful DevOps culture,
- **Release faster and work smarter :** Speed is everything. Teams that practice DevOps release deliverables more frequently, with higher quality and stability
- **Accelerate time-to-resolution :** The team with the fastest feedback loop is the team that thrives. Full transparency and seamless communication enable DevOps teams to minimize downtime and resolve issues faster.
- **Better manage unplanned work :** Unplanned work is a reality that every team faces-a reality that most often impacts team productivity. With established processes and clear prioritization, development and operations teams can better manage unplanned work while continuing to focus on planned work. Transitioning and prioritizing unplanned work across different teams and systems is inefficient and distracts from work at hand. However, through raised visibility and proactive retrospection, teams can better anticipate and share unplanned work

1.6 Process Automation

- **Process automation** uses technology to automate complex business processes. It typically has three functions: automating processes, centralizing information, and reducing the requirement for input from people. It is designed to remove bottlenecks, reduce errors and loss of data, all while increasing transparency, communication across departments, and speed of processing.
- Business process automation (BPA) is the use of software to automate repeatable, multistep business transactions. In contrast to other types of automation, BPA solutions tend to be complex, connected to multiple enterprise information technology (IT) systems, and tailored specifically to the needs of an organization

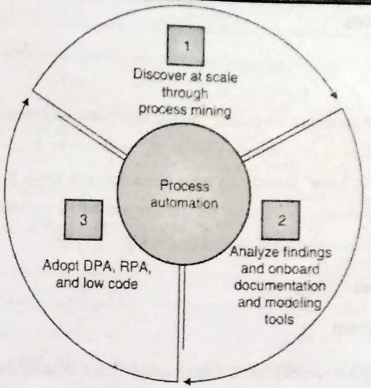


Fig. 1.6.1

- Think about a car wash. The process, which is fully automated, looks something like this :
  - The customer chooses which wash type they want
  - System requests payment from the customer
  - Takes payment from the customer
  - Approves transaction and advises customer to drive into the carwash
  - Identifies when the car is in the right position using a sensor and advises the driver to stop
  - Uses a range of sensors to identify the car's height and size
  - Runs the pre-selected and paid program. This includes a number of variables, including rinsing, washing with soap and brushes, wax application and blow-drying the car
  - When complete, advises the customer to exit the carwash.
- While a very simple automation, this process is one that most people are familiar with, and seamlessly integrates a digital transaction and input from the customer and turns it into a mechanical series of automations using software, hardware, and communication to an accounting system.
- Process automation streamlines a system by removing human inputs, which decreases errors, increases speed of delivery, boosts quality, minimizes costs, and simplifies the business process. It incorporates software tools, people, and processes to create a completely automated workflow.

1.7 Agile Practices

A. What is Agile?

Agile is an iterative approach to project management and software development that helps teams deliver value to their customers faster and with fewer headaches. Instead of betting everything on a "big bang" launch, an agile team delivers work in small, but consumable, increments. Requirements, plans, and results are evaluated continuously so teams have a natural mechanism for responding to change quickly.

B. Agile Best Practices

1. Iterative Development

Through agile iterative development, bigger projects are broken down into smaller chunks and continuous tests are done in repetitive cycles. Through this practice, agile teams get a perspective on new features that need to be added to the final product or service and contribute towards more flexible product development.

2. Daily Meetings

Regular meetings are key to agile implementation. These meetings should be short and concise, with each member of the team explicitly stating the progress of tasks and what needs to be done. This practice is a great way to monitor the performance of the team and check if there are any obstacles in the way of product development.

3. Using Professional Tools

Using project management tools for the implementation of agile methodology helps the team to better structure their workflows and improve team collaboration. For proper documentation and meetings management, professional project management software can greatly reduce the effort it takes to manage your tasks otherwise.

C. Agile Best Practices : Scrum Project Management

Scrum is considered to be a dominant agile framework, with stats showing that 58% of the organizations implement this for their products' development and 18% of organizations use it in combination with other frameworks.

Some agile best practices for Scrum implementation are :

1. Creating Product Backlog and Product Vision Together

A product backlog is an ordered list of items that are required to be added to product development. A good practice for scrum implementation is to create the product backlog and product vision together so that both the development team and stakeholders are on the same page. This ensures mutual understanding and helps in aligning the vision in a better way.

2. Use Burn down Charts for Sprints

A daily burn down chart is a great way of monitoring the progress of Sprints. Burn down charts graphically show the work that has been done and the total work remaining against time. It's a useful tool to inform the team about project scope and make them aware of scope creep that might occur. These charts also help in identifying the risks associated with undelivered work.

3. Setting communication guidelines for teams

Uninterrupted communication is key for the Scrum framework and can become a bottleneck if not tackled efficiently. An effective way to ensure seamless communication is to formulate a communication strategy with all the essential guidelines for teams. This particular practice can really come in handy for remote teams as it will make team goals transparent.

4. Practicing Stand-Ups

Also known as the 'Daily Scrum', stand-ups are short meetings held with the team members on a daily basis. These meetings are typically for a maximum of 15 minutes to keep their duration short. Practicing Stand-ups for product or project development are a great way to monitor the progress of work and helps in keeping everyone in the loop with the project updates. These meetings also assist the team in tracking the dos and don'ts of product development.

1.8 Reason for Adopting DevOps

- DevOps, a portmanteau of "Development" and "Operations", is a set of tools, processes, and practices that combine software development and IT operations to improve software delivery.
- Due to the tendency for development and operations teams to work in silos, DevOps acts as the go-between to improve collaboration efficiency.

- Following are main reasons for adopting DevOps
  1. Efficient Development and Deployment Planning
  2. Continuous Improvement and Software Delivery
  3. Improves Software Security
  4. Improves Customer Experiences
  5. Better Collaboration Among Teams
  6. DevOps Practices Create More Time to Innovate
  7. DevOps Enhances Decision-Making
  8. DevOps Encourages Higher Trust and Better Collaboration

1. Efficient Development and Deployment Planning

- One of the biggest headaches of any IT Manager is managing cross-functional teams to develop and deploy software in good time. DevOps practices help you plan in advance how both teams will work cohesively to deliver products, meet customer needs and stay competitive.
- DevOps methods such as Scrum and Kanban provide practices that define how teams work together. For example, Scrum practices include working in sprints, holding Scrum meetings, and designating time boxes.

2. Continuous Improvement and Software Delivery

Adopting DevOps practices improves the quality of your software when deploying new features and allows you to make changes rapidly. Continuous Integration and Continuous Deployment (CI/CD) is the practice where you make incremental changes to your code and quickly merge to the source code. This DevOps practice delivers your software to the market faster while addressing customer issues rapidly.

3. Improves Software Security

A subset of DevOps is DevSecOps which combines development, security, and operations. Adopting DevOps practices allows you to build security into your software continuously. The IT security team is involved in the software development cycle from the start rather than at the deployment stage. Outdated security practices integrate infrastructure security features independently. With DevOps, software security is a collaborative practice and is considered the foundation before any product development.

4. Improves Customer Experiences

A major benefit to adopting DevOps practices is it lowers the failure rates of new features while improving recovery time. The continuous deployment, testing, and feedback loop ensure faster service delivery and happier customers. By automating the software pipeline, the development teams focus on creating superior products while the operations team can improve business delivery.

5. Better Collaboration Among Teams

DevOps practices dismantle silos between teams and improve collaboration. By adopting methods such as Scrum and Kanban, teams collaborate more efficiently, friction among colleagues is dealt with quickly, and communication flows seamlessly across the organisation. Since DevOps discourages hierarchies among teams, everyone is responsible for software quality and speedy delivery. This approach reduces cases of low-effort contributors and blame games.

6. DevOps Practices Create More Time to Innovate

- DevOps automates repetitive tasks, improves service delivery, and reduces the software development cycle. The methodology frees up time for teams to develop new products and better features. Unfortunately, inefficient teamwork leads to time wasted fixing errors rather than innovating. Building better products and services leads to a competitive advantage.
- Companies that innovate continuously are also better able to react to market changes and take advantage of new opportunities.

7. DevOps Enhances Decision-Making

- DevOps helps you leverage data to make better decisions and removes hierarchies that require layers of approvals. In addition, DevOps practices reduce time spent between design and execution. Scrum and the Kanban approach address blockers as they come, manage workflows efficiently, and tackle projects in sprints.
- Fast decision-making based on accurate data is a competitive advantage. Slow decision-making delays go-to-market plans and throttles the implementation of new ideas.

8. DevOps Encourages Higher Trust and Better Collaboration

- Due to improved collaboration across development and IT operations, teams trust each other more and produce higher quality work. Higher trust leads to improved morale and better productivity.

- Conversely, teams working in silos mistrust each other's new suggestions and ideas. A culture of trust and collaboration creates transparency in workflows and gives visibility into projects. This approach leads to better development and deployment planning. The net effect is a collaborative environment that works in unity, solves problems faster, and delivers superior products to the market.

A. How to adopt DevOps

Adopting DevOps first requires a commitment to evaluating and possibly changing or removing any teams, tools, or processes your organization currently uses. It means building the necessary infrastructure to give teams the autonomy to build, deploy, and manage their products without having to rely too heavily on external teams.

- **DevOps culture :** A DevOps culture is where teams embrace new ways of working that involve greater collaboration and communication. It's an alignment of people, processes, and tools toward a more unified customer focus. Multidisciplinary teams take accountability for the entire lifecycle of a product.
- **Continuous learning :** Organizations that do DevOps well are places where experimentation and some amount of risk-taking are encouraged. Where thinking outside the box is the norm, and failure is understood to be a natural part of learning and improving.
- **Agile :** Agile methodologies are immensely popular in the software industry since they empower teams to be inherently flexible, well-organized, and capable of responding to change. DevOps is a cultural shift that fosters collaboration between those who build and maintain software. When used together, agile and DevOps result in high efficiency and reliability.

B. DevOps practices

- **Continuous integration :** Continuous integration is the practice of automating the integration of code changes into a software project. It allows developers to frequently merge code changes into a central repository where builds and tests are executed. This helps DevOps teams address bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates.

- **Continuous delivery :** Continuous delivery expands upon continuous integration by automatically deploying code changes to a testing/production environment. It follows a continuous delivery pipeline, where automated builds, tests, and deployments are orchestrated as one release workflow.
- **Situational awareness :** It is vital for every member of the organization to have access to the data they need to do their job as effectively and quickly as possible. Team members need to be alerted of failures in the deployment pipeline — whether systemic or due to failed tests - and receive timely updates on the health and performance of applications running in production. Metrics, logs, traces, monitoring, and alerts are all essential sources of feedback teams need to inform their work.
- **Automation :** Automation is one of the most important DevOps practices because it enables teams to move much more quickly through the process of developing and deploying high-quality software. With automation the simple act of pushing code changes to a source code repository can trigger a build, test, and deployment process that significantly reduces the time these steps take.
- **Infrastructure as Code :** Whether your organization has an on-premise data center or is completely in the cloud, having the ability to quickly and consistently provision, configure, and manage infrastructure is key to successful DevOps adoption. Infrastructure as Code (IaC) goes beyond simply scripting infrastructure configuration to treating your infrastructure definitions as actual code : using source control, code reviews, tests, etc.
- **Microservices :** Microservices is an architectural technique where an application is built as a collection of smaller services that can be deployed and operated independently from each other. Each service has its own processes and communicates with other services through an interface. This separation of concerns and decoupled independent function allows for DevOps practices like continuous delivery and continuous integration.
- **Monitoring :** DevOps teams monitor the entire development lifecycle — from planning, development, integration and testing, deployment, and operations. This allows teams to respond to any degradation in the customer experience, quickly and automatically. More importantly, it allows teams to "shift left" to earlier stages in development and minimize broken production changes.

1.9 What and Who Are Involved in DevOps?

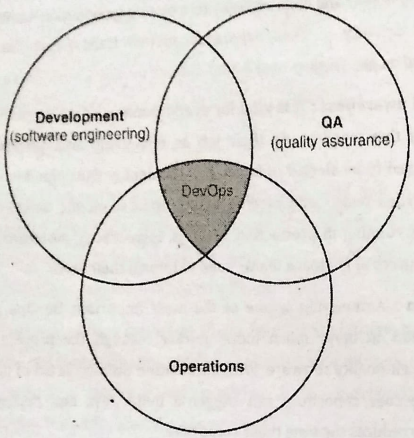


Fig. 1.9.1

- DevOps' advent has transformed the software development landscape, bringing cross-functional teams of developers, operations, and QA to seamlessly collaborate and deliver quality in an automated continuous delivery environment. In a DevOps-driven continuous delivery environment, developers and operations teams jointly think about how a feature will respond in a production environment, resulting in reduced errors, faster time to market, better quality, and efficiencies. With Quality Engineering and Quality Assurance going hand in hand, QA teams are happier now as quality is not just their job, but it turns into DevOps Team responsibilities.
- DevOps teams comprise professionals from development, quality, security, and the operations segment. As the core responsibility of the team would be on the person who owns the DevOps team, a senior person from the organization would be an ideal person to lead the team, referred to as a DevOps Evangelist. The DevOps evangelist will ensure that the responsibilities of DevOps processes are assigned to the right people. The smallest DevOps team should comprise the following people; A software developer/tester, automation engineer/automation expert, quality assurance professional, security engineer, and release manager. The granularity of the team ultimately depends on the size of the organization.
- Here are a few common roles in a DevOps team.

1. DevOps Engineer

A DevOps engineer is responsible for designing the right infrastructure required for teams to continuously build and deliver products. The engineer identifies project requirements and KPIs and customizes the tool stack. He is well versed with automation tools and security technologies. Right from the build, test, deployment, and monitoring of a product, the engineer integrates all resources and functions required at every stage of the product lifecycle while protecting the cloud architecture from hacking attacks. In addition, the engineer is involved in team composition, project activities, defining and setting the processes for CI/CD pipelines and external interfaces.

2. Release Manager

When it comes to the DevOps team structure, the release manager holds one of the most demanding and stressful roles. The release manager is responsible for the entire release lifecycle, right from planning, scheduling, automating, and managing continuous delivery environments. Release managers are mostly Ops-focused wherein they design an automation pipeline for a smooth progression of code to production, monitor feedback, reports, and plan the next release, working in an endless loop.

3. DevOps/CloudOps architect

The responsibility of a DevOps architect is to analyse existing software development processes and create an optimized DevOps CI/CD pipeline to rapidly build and deliver software. The architect analyses existing processes and implement best practices to streamline and automate processes using the right tools and technologies. In addition, he monitors and manages technical operations, collaborates with dev and ops, and offers support when required. He also acts as a leader as required.

4. Security and Compliance Engineer

The Security and Compliance Engineer (SCE) is responsible for the overall security of the DevOps environment. The SCE closely works with the development teams to design and integrate security into the CI/CD pipeline, ensuring data integrity and security are not compromised at every stage of the product lifecycle. In addition, the SCE ensures that the products being developed are adhering to governing regulations and compliance standards.

5. Software Developer/Tester

While a regular software developer writes the code to build a product, the DevOps software developer/tester is involved across the product lifecycle. Responsibilities of DevOps developers include tasks such as updating the code, adding new features, and resolving bugs while ensuring that the application meets business objectives. In addition, the developer runs unit tests, pushes the code to production, and monitors its performance.

6. DevOps evangelist

For an organization to fully leverage DevOps, it should go through a complete cultural shift. A DevOps evangelist is the one who acts as this change agent, inspiring, educating, and motivating people across the organization to embark on the DevOps journey. The evangelist removes silos between different teams, brings them onto a common platform, determines the roles and responsibilities of DevOps members, and ensures everyone is trained on the job they are assigned. Overall, the evangelist leads the DevOps journey, ensuring that a cultural shift is happening across the organization, everyone is aware of their roles and responsibilities, finds ways to optimize processes and ensures that best practices are implemented in an end-to-end product development lifecycle.

1.10 Changing the Coordination

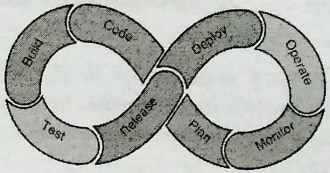
It's best to understand DevOps as a business drive to improve communication and collaboration among development and operations teams, in order to increase the speed and quality of software deployment. It's a new way of working that has profound implications for teams and the organizations they work for.

- The culture encourages cross-functional collaboration and shared responsibilities and avoids silos between Dev, Ops and QA.
- The culture encourages learning from failures and cooperation between departments.
- Communication flows fluidly across the end-to-end cross-functional team using collaboration tools where appropriate (for example Slack, HipChat, Yammer).
- The DevOps system is created by an expert team, and reviewed by a coalition of stakeholders including Dev, Ops and QA.
- Changes to end-to-end DevOps workflows are led by an expert team, and reviewed by a coalition of stakeholders including Dev, Ops and QA.

- DevOps system changes follow a phased process to ensure the changes do not disturb the current DevOps operation. Examples of implementation phases include : proof of concept (POC) phase in a test environment, limited production and deployment to all live environments.
- Key performance indicators (KPIs) are set and monitored by the entire team to validate the performance of the end-to-end DevOps pipeline, always. KPIs include the time for a new change to be deployed, the frequency of deliveries and the number of times changes fail to pass the tests for any stage in the DevOps pipeline.

1.11 Introduction to DevOps Pipeline Phases

- DevOps is visualized as an infinite loop comprising the steps : plan, code, build, test, release, deploy, operate, monitor, then back to plan, and so on.
- All of these components of the DevOps lifecycle are necessary to take the maximum leverage of the DevOps methodology.



DevOps lifecycle diagram

Fig. 1.11.1 : DevOps Lifecycle Diagram

Following are some important components in devops lifecycle.

1) Continuous Development :

With continuous development, every change to your software is may be integrated, tested, and verified. This helps to ensure that your team is able to release a safe, secure, reliable, and high-quality product on time. This practice spans the planning and coding phases of the DevOps lifecycle. Version-control mechanisms might be involved.

2) Continuous Integration :

This software engineering practice develops software by frequently integrating its components. It helps to ensure that changes in the source code do not break the build or cause other problems. The key goals of continuous integration are to find and address bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates.

3) Continuous Testing :

Continuous testing in DevOps is a type of software testing that involves testing at every stage of the development life cycle. The goal of continuous testing is to evaluate the quality of the software as part of a continuous delivery process, by testing early and often.

4) Continuous Deployment :

Continuous deployment is a strategy in software development where code changes to an application are released automatically into the production environment. This automation is driven by a series of predefined tests. Once new updates pass those tests, the system pushes the updates directly to the software's users.

5) Continuous Monitoring :

During this phase, developers collect data, monitor each function, and spot errors like low memory or server connection are broken. For example, when users log in, they should access their account, and a failure to do so means there's a problem with your application.

6) Continuous Feedback :

Continuous feedback is like a progress report. In this DevOps stage, the software automatically sends out information about performance and issues experienced by the end-user. It's also an opportunity for customers to share their experiences and provide feedback.

7) Continuous Operations :

It involves automating the application's release and all these updates that help you keep cycles short and give developers and provide more time to focus on developing.

1.12 Defining the Development Pipeline

- A DevOps pipeline is a set of tools and automated processes utilized by the software engineering team to compile, build, and deploy code. Building an effective DevOps pipeline enables companies to rapidly develop, test, and deploy new code on an ongoing basis.
- One of the key objectives of a DevOps pipeline is to automate the software delivery process, eliminating the need for manual changes through every step of the pipeline. Manual work is time-consuming and introduces the potential for human error, often pushing back deployments.
- The shift from manual changes to automated changes not only results in fewer errors, but also allows developers to push out higher-quality code faster than ever before.

A. Components of a DevOps Pipeline

There are several different approaches and tools that organizations can use to create a customized DevOps pipeline. Common pipeline components facilitate continuous delivery to ensure that code moves seamlessly from one stage to the next, automating the entire process and minimizing manual work.

- **Continuous integration and continuous delivery (CI/CD)** – CI/CD, which allows for the rapid integration of new code, is one of the cornerstones of DevOps pipelines.CI allows for the rapid integration of small chunks of new code from multiple developers into a shared repository. CI also allows you to automatically test the code for errors to identify bugs early on, making them easier to fix. CD is an extension of CI, enabling developers to perform additional tests such as UI tests, which helps ensure bug-free deployment. CD also helps the DevOps team deliver bug fixes, increase the frequency of new feature releases, and automate the entire software release. These features reduce the overall time and cost of a project.
- **Continuous Testing (CT)** – CT allows companies to perform automated testing at every stage of the development process. A CT strategy allows for quick evaluations of the release risks of code integrations. Tests begin to run automatically once code is integrated.
- **Continuous Deployment** – Continuous deployment is often confused with continuous delivery, but there's a substantial difference between the two. At the Continuous deployment stage, the entire release cycle is automated and code updates go directly to the end user without manual interventions. The downside to these automated deployments is that if bugs have not been detected along the way, they will be released and can cause the app to fail. Continuous deployments are only recommended for minor code updates. In the worst-case scenario, you can roll back the changes. The upside is that continuous deployment enables frequent deployments in a single day.
- **Continuous Monitoring** – Continuous monitoring enables rapid detection of compliance issues and security risks, empowering SecOps (a word that blends of "security" and "operations") teams with real-time information from across a company's IT infrastructure as well as supporting critical security processes like threat intelligence, forensics, root cause analysis, and incident response.

- **Continuous Feedback** – Once code is successfully deployed, continuous feedback shows the impact of the release on end users. By automating feedback, the company gets insights and information on how users are reacting to the new build. If critical issues are discovered, development teams will get notified and can immediately start working on bug fixes.
- **Continuous Operations** – The goal of continuous operations is to reduce or eliminate the need for planned downtime, which results in minimal interruption to the end users. Setting up continuous operations is a costly endeavor, but it may be worth the extra cost considering its advantages.

B. How to Build a DevOps Pipeline

Companies use a diverse set of tools and approaches to build unique and effective DevOps pipelines customized for the needs of their organization. Common steps include establishing a CI/CD tool, sourcing a control environment, setting up a build server, setting up build automation tools for testing, and deploying to production.

Step 1 : Establish CI/CD Tool

For companies just getting started building a DevOps pipeline, the first order of business is to pick a CI/CD tool. Each company has different needs and requirements, so not every tool will be right for every situation. While there are many CI/CD tools available, Jenkins is one of the most commonly used tools. Jenkins features hundreds of community-contributed plugins and tools, so it can easily be customized to work well for many different applications.

Step 2 : Source a Control Environment

Companies working with large development teams need a dedicated place to store and share the ever-changing code, avoid merge conflicts, and easily create different versions of the app or software. Source control management tools such as Git allow for effective collaboration with team members from anywhere in the world, storing code from each developer in a separate shared repository. BitBucket and GitLab are two other popular source control management tools.

Step 3 : Set up a Build Server

- Setting up a build server, which is also called a continuous integration (CI) server is a crucial next step before your project can make its way down the rest of the pipeline. A build server is a centralized, stable and reliable environment for building distributed development projects. Build servers retrieve integrated code from source code repositories, acts as an integration point for all developers, and provides an untainted environment to make sure that the code actually works.

- Like other tools mentioned above, companies can choose from many build servers, each with different features. Jenkins is one of the most popular solutions for creating builds. Other solutions include TeamCity, Travis-CI, and go.cd.

Step 4 : Setup or Build Automation Tools for Testing

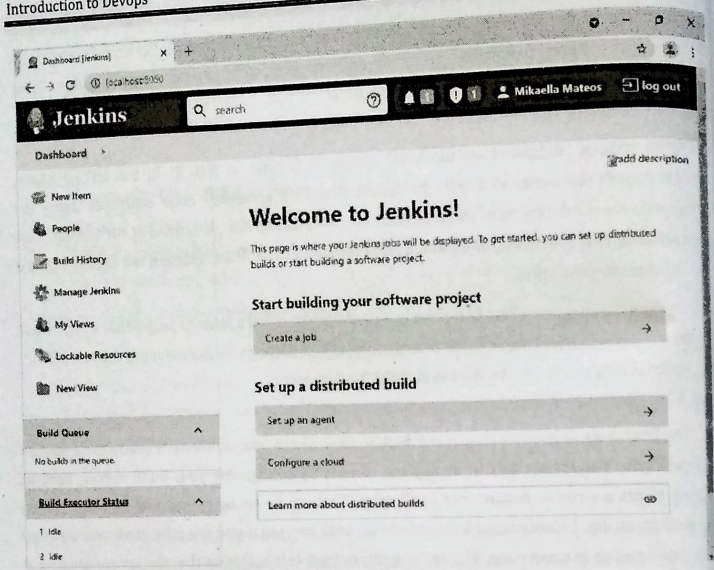
- Once code is configured on the build server, it is time to test it! In the testing phase, developers run automated tests to ensure that only error-free code continues down the pipeline to the deployment stage. Several automated tests are performed at the testing stage including unit, functional, integration, and regression tests. Most tests are run through CI and run one after the other.
- To start running tests, Test Complete is a good option. It has a Jenkins plugin that enables you to run tests in a Jenkins Pipeline project with added features such as storing the test history and enabling you to see the results directly from Jenkins.

Step 5 : Deploy to Production

The final stage of your pipeline is the deploy stage where your software is ready to be pushed to production. The easiest way to deploy the code is by configuring your build server, such as Jenkins, to run a script to deploy your application. This can be set up to run automatically or you can do it manually. Automatic deployment should only be used if you are fully confident that bad code won't end up in production. You can link this to your test builds, so that the script only runs if all the tests have been passed.

1.13 Centralizing the Building Server

- Jenkins is one of the most popular and flexible open-source orchestration tools out there. No matter if you working on a legacy monolith or micro-services, on-premise or in the cloud, Jenkins can help.
- You can install Jenkins in numerous ways, each of them fitting a different platform and use case. the easiest way to install Jenkins is with a package manager.
- Once you have installed Jenkins, you can access it right from your web browser. On Windows, you likely use Chocolatey, on MacOS Homebrew, and on Linux it depends on which distribution of Linux you are using.
- Open your browser and navigate to localhost:8080, and you see Jenkins starting or waiting for an initial password.
- Jenkins is now installed and running on Windows, and you are ready to start building your CI/CD pipeline.



1.14 Monitoring Best Practices

- Monitoring is the process of gathering and analyzing data related to a critical system, service or application's performance. Monitoring helps ensure systems and services are running as intended and helps teams keep a pulse on the performance and availability of any internal or external application, system, or service. If a disruption or outage occurs, teams are immediately aware via the monitoring system and can take action towards a resolution immediately.
- Monitoring is proactive approach of collecting and analyzing information can help developers, sysadmins and security teams in many ways such as detecting issues in their code via "application-level logging," identifying anomalies in network traffic via "infrastructure logs like AWS/Azure," and detect as well as prevent security incidents by using advanced Security Information and Event Management capabilities. You can address these challenges by employing these logging and monitoring best practices.

1. **Define your need to log and monitor**
   - Determining why the organization wants a logging solution will help define what you need to log. The following are some of the reasons an organization might want such as solution :
     - Compliance requirements
     - Local laws and regulations
     - Incident response requirements
   - Discussing these factors with your organization's security governance team, legal department, and other stakeholders will help define the goals for logging and monitoring.
2. **List what needs to be logged and how it needs to be monitored**
   - Based on your goals, determine what metadata needs to be captured and what events need to be logged. Some examples of metadata and events to be logged. Infrastructure administrators and security teams should collaborate to build an effective logging and monitoring program that collects traditional operational metrics and can analyze them to mitigate attacks.
   - Alerts on certain events, such as multiple failed login attempts or weekly notifications on commands executed on a server, can be set up to monitor these events. It is also important to work with application teams to understand what the different attributes of a log entry mean. Once you have a baseline for normal operations, you can configure correlation rules, aggregations, thresholds, and alerts to be triggered for any anomaly based on the security risk profile for the application
3. **Identify assets and events that need to be monitored**

   Log data is a huge volume of datasets that impact performance and costs. When determining what data you should monitor start by not leaving anything out. You need to identify which systems/applications should be monitored and what level of monitoring is required. You should also classify your data and systems according to the organization's statutory, regulatory, or contractual requirements. Keep in mind that this classification may differ from your security system classification or your business data classification.
4. **Determine the right solution for logging and monitoring**
   - There are a many solutions—both commercial products and open source projects—to choose from when you want to build a scalable and resilient logging and monitoring program.

- Choosing the right technologies for a logging and monitoring architecture can be overwhelming. A few key points that you need to keep in mind are :
  - Automate as much of the monitoring process as possible
  - Constantly tune your alerts and log sources as threats evolve
  - Ensure that log and alerts are generated in a standardized format

5. Design logging and monitoring systems with security in mind

- A logging and monitoring program by itself is an asset to the organization because it looks into organization wide activities and may contain sensitive information. Here are few points to consider to secure it :
  - Redact/mask/anonymize sensitive information from event logs beforehand, to prevent sensitive information from being logged in plain text (e.g., PHI/PII information)
  - Enforce role-based access controls
  - Perform log integrity checks to ensure that logs are not tampered with
  - Apply encryption at rest and transit
  - Follow the principle of least privilege when configuring log sources
  - Sanitize logs before storing and processing
  - Include capabilities for high availability and redundancy

6. Adopt organization wide logging and monitoring policies

Work with security teams to enforce companywide policies and procedures that define logging requirements in detail for all systems. This ensures consistency and that protocols and procedures are followed in logging. Policies with a strong mandate and corporate backing ensure that logging and monitoring practices are followed.

7. Establish active monitoring, alerting and incident response plan

- Without strong logging mechanisms, an organization is truly in the dark before, during and after any incident. Attacks on sophisticated systems are often carried out for months or even years. The following steps are vital to prevent such a scenario :
  - Establish an incident response plan and rehearse it at regular intervals
  - Trigger alerts in an adequate amount of time
  - Take active automated actions on the alerts

1.15 Best Practices for Implementing DevOps

Agile project management

Agile is an iterative approach to project management and software development that helps teams deliver value to their customers faster and with fewer headaches. Agile teams focus on delivering work in smaller increments, instead of waiting for a single massive release date. Requirements, plans, and results are evaluated continuously, allowing teams to respond to feedback and pivot as necessary.

Shift left with CI/CD

When teams "shift left", they bring testing into their code development processes early. Instead of sending multiple changes to a separate test or QA team, a variety of tests are performed throughout the coding process so that developers can fix bugs or improve code quality while they work on the relevant section of the codebase. The practice of continuous integration and continuous delivery (CI/CD), and deployment underpins the ability to shift left.

Build with the right tools

A DevOps tool chain requires the right tools for each phase of the DevOps lifecycle, with key capabilities to improve software quality and speed of delivery

Implement automation

Continuous integration and delivery allows developers to merge code regularly into the main repository. Instead of manually checking code, CI/CD automates this process, from batching in a specified window to frequent commits. In addition to CI/CD, automated testing is essential to successful DevOps practices. Automated tests might include end-to-end testing, unit tests, integration tests, and performance tests.

Monitor the DevOps pipeline and applications

It's important to monitor the DevOps pipeline so a broken build or failed test doesn't cause unnecessary delays. Automation improves the speed of development tremendously, but if there is a failure in an automated process and nobody knows about it, you're better off doing the work manually. it's important to monitor production applications in order to identify failures or performance deficiencies, before you hear about them from your customers.

Observability

- As the industry moved away from monolithic, on-premise systems and applications to cloud-native, microservice-based applications, monitoring is now considerably more complex. As a result, there is an increasing interest in observability.

- It is often said that the three pillars of observability are **logs, traces,** and **metrics.** Logs are generated by most systems components and applications and consist of time-series data about the functioning of the system or application. **Traces** track the flow of logic within the application. **Metrics** include CPU/RAM reservation or usage, disk space, network connectivity, and more. Observability simply means using all three of these sources of information in aggregate to make discoveries and predictions about the functioning of a complex system, which would otherwise be difficult to achieve.

Gather continuous feedback

Continuous feedback ensures team members have all the information needed to do their jobs on a timely basis. From the development perspective this entails that the team is alerted to any pipeline failures immediately. It also means that clear, thorough code test results are made available to the developers as quickly as possible. From the product management perspective the team is made aware of any production failures or performance deficiencies, or reported bugs. Continuous feedback is one of the elements of DevOps that makes it possible to have both.

Change the culture

DevOps requires collaboration, transparency, trust, and empathy. If your organization is one of the rare ones where these qualities are already established, it should be fairly easy for your teams to adopt DevOps practices. If not, some effort will be required to develop these qualities. The most common organizational structures are siloed, meaning different teams have separate areas of ownership and responsibility and there is minimal cross-team communication or collaboration. For DevOps to succeed, these barriers must be eliminated by adopting the "you build it, you run it" practice.

REVIEW QUESTIONS

Q. 1 What is Devops?

Q. 2 Write and Explain the role of devops engineer.

Q. 3 What are the responsibilities of developer Responsibility?

Q. 4 What do you mean by continuous Integration and continuos Delivery? Explain it

Q. 5 What is Devops culture?

Q. 6 What are the benefits of Devops?

Q. 7 Write and explain Process automation in detail.

Q. 8 Explain Agile process in detail?

Q. 9 Write and explain reasons for adopting the devops.

Q. 10 What and who are involved in devops?

Q. 11 What is Devops Pipeline? Explain phases of devops pipeline.

Q. 12 Explain Development pipeline in detail.

Q. 13 What is centralizing the building server.

Q. 14 What do you mean by monitoring best practices.

Q. 15 What are the best practices for implementing devops.

□□□

2 Microservices Architecture and Cloud Native Development

UNIT - II

Syllabus

Monolithic applications, Introduction to microservice architecture, Implementing a microservices Architecture, Pros and Cons of a microservice Architecture, Characteristics of microservice architecture, Monolithic applications and microservices compared, microservices best practices, Deployment strategies, Introduction to cloud computing, cloud computing deployment models, service models, why to use cloud, Principle of container based application design, Introduction to Docker, Serverless computing, orchestration, Difference between orchestration and automation.

2.1 Monolithic Applications

- The monolithic architecture style is like a large container in which all the software components of an application are integrated into one large package.
- If all the functionalities of a project exist in a single codebase, then that application is known as a **monolithic application.** We all must have designed a monolithic application in our lives in which we were given a problem statement and were asked to design a system with various functionalities. We design our application in various layers like presentation, service, and persistence and then deploy that codebase as a single jar/war file. This is nothing but a monolithic application, where "mono" represents the single codebase containing all the required functionalities. For example, a traditional application will have a front end, API, services, load balancer, and database. If you build everything together and deploy it on the server, that's called a monolithic architecture, where services tightly couple together.
- Let's imagine that you are 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, along with some backend services for checking credit, maintaining inventory and shipping orders.

- The application is deployed as a single monolithic application. For example, a Java web application consists of a single WAR file that runs on a web container such as Tomcat.

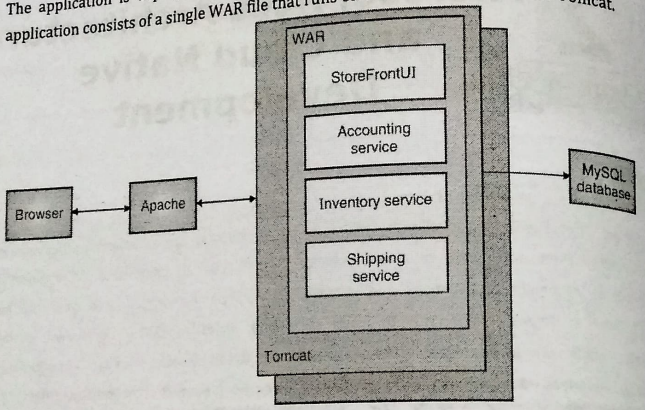


Fig. 2.1.1

Advantages of monolithic applications :

- **Easier to develop**. Working with a single executable is simple. So, for straightforwar applications or the beginning of a development project, a monolithic architecture is easie However, as development progresses and complexities arise, monolithic environments ca become a drawback.
- **Easier to test**. Due to the nature of the application, you can simply launch the app and tes the user interface with a given tool. Using one executable means there's only one applicatio you need to set up for logging, monitoring, and testing.
- **Easier to deploy**. There's much less complexity when working with a single executable. T deploy to other systems, you need to copy the packaged application to a different server an run it.
- **Less complex and lower overhead**. Microservices can become complex quickly. However, a monolithic architecture, you can avoid additional costs associated with interservic communication, service discovery and registration, load balancing, distributed logging distributed performance monitoring and management, and data management.

Disadvantages of Monolithic applications :

- It becomes too large with time and hence, difficult to manage.

- We need to redeploy the whole application, even for a small change.
- As the size of the application increases, its start-up and deployment time also increases.
- For any new developer joining the project, it is very difficult to understand the logic of a large Monolithic application even if his responsibility is related to a single functionality.
- Even if a single part of the application is facing a large load/traffic, we need to deploy the instances of the entire application in multiple servers. It is very inefficient and takes up more resources unnecessarily. Hence, horizontal scaling is not feasible in monolithic applications.
- It is very difficult to adopt any new technology which is well suited for a particular functionality as it affects the entire application, both in terms of time and cost.
- It is not very reliable, as a single bug in any module can bring down the entire monolithic application..

2.2 Introduction to Microservice Architecture

- The objective of the microservice architecture style is to completely decouple application components from one another such that they can be created, deployed, scaled, and maintained independently
- In a monolithic application, all processes rely heavily on each other and operate as a single service. In such an architecture, an increase in the demand for the bandwidth of any one process would mean that the complete architecture needs to be scaled up.

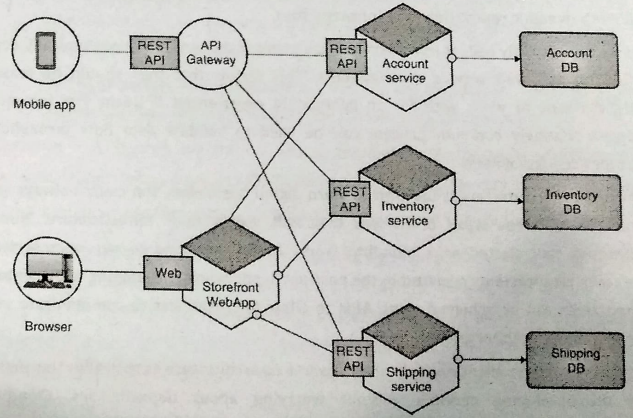


Fig. 2.2.1 : The Architecture of an Ecommerce Application

- Since all the code in a monolithic application is deployed together on the same base, adding or enhancing features becomes a complicated process, especially as the code base expands in size and complexity. Additionally, monolithic applications might be susceptible to failure. This is because tightly coupled, essentially interdependent processes are easily affected if a single process goes down.
- All this puts constraints on experimentation and can make it difficult for enterprises to stay fluid and responsive; this potentially puts them at a disadvantage in a highly dynamic, customer-centric market.
- Microservices allow large applications to be split into smaller pieces that operate independently. Each 'piece' has its responsibilities and can carry them out regardless of what the other components are doing. A microservices-based application summons the collective services of these pieces to fulfill user requests.
- The services in a microservices architecture 'talk' to each other using lightweight application programming interfaces (APIs) that connect to detailed interfaces. These services are created to perform specific business functions, such as monetary transactions, invoice creation, and data processing. Each service carries out a single operation. As they run independently, the services can be deployed, updated, and scaled according to the demand for their specific functions.
- The implementation of a microservices architecture results in the creation of business systems that are flexible and scalable. However, shifting from monolith to microservices requires a dynamic renovation of IT infrastructure.
- This isn't necessarily bad, as microservices use many of the same solutions typically deployed in RESTful and web service environments. This means that they should be reasonably straightforward to work with for an adequately experienced IT team. For instance, API testing–a relatively common process–can be used to validate data flow throughout the microservices deployment.
- Microservices architecture is ideal for modern digital businesses that cannot always account for all the different types of devices that will access their infrastructure. Numerous applications that started as a monolith were slowly revamped to use microservices as unforeseen requirements surfaced in the post-pandemic world. Revamping larger enterprise environments can be achieved using APIs to allow microservices to communicate with an older monolithic architecture.
- Containers are an excellent example of microservices architecture as they allow businesses to focus on developing services without worrying about dependencies. Cloud-native applications are commonly built as microservices by leveraging containers.

2.3 Implementing a Microservices Architecture

- As Microservices bring a fundamental shift in the way applications are designed and developed, it's helpful to explore how Microservices are implemented.
- The foundation of a comprehensive Microservice solution is the ability to define all microservices within clear boundaries and aligning them with business requirements. A key element of Microservice design is understanding that each microservice should be dedicated to one specific function or feature.

Consider the following aspects while architecting a Microservice :

- **Communication –** Microservices architecture is unique in a way. It is a collection of numerous services, each running its functions and interacts with each other. This requires them to communicate with each other at the right time, and hence communication is important.
- Microservices can communicate in two ways : **Synchronous and Asynchronous**. To understand how they work, just follow the formation of these two words :
  - **Synchronous Communication** works in a coordinated and patterned way. For example, a customer added items to the shopping cart (Shopping Cart is a microservice), this will lead to creating a chain of dependencies and various services are being called one after the another like Applying Discounts, Payment, and Inventory. This can often lead to speed issues and errors in case any of the services become unavailable.
  - **Asynchronous Communication** works independently. The services are independent of each other and handle communication separately.

Integration – A smooth integration among Microservices ensure a seamless end-user experience. Microservices support API, Message, and File Transfer way of integration style.

- **The API style of Integration** is synchronous and triggers subsequent responses.
- **Messaging Style** works via a common messaging system. The messaging system is used to exchange data or pass on messages in a pre-defined format. This type of integration can handle high-volume spikes. **File Transfer Integration** works by exporting source applications to a file system in CSV format and then getting them processed by the target system. This type of integration works when loads of data need transfer and execution at pre-defined times.

- **Deployment** - Choosing the right deployment strategies to help organizations to attai higher agility, flexibility, scalability, and efficiency. There are different Microservice deployment strategies that teams can use - **Multiple Service Instance Per Host, Servic Instance Per Host, Service Instance Per Container, and Server-Less Deployment.**

2.4 Pros and Cons of a Microservice Architecture

a. Microservice Architecture Pros

- **Improved continuous integration/continuous delivery.** Because the architectur decouples services, DevOps teams can scale complex applications in a mor straightforward manner. Instead of one extensive application, teams can work o application pieces to ensure a better development pipeline.
- **Better testing.** With smaller services, it's easier to test and monitor applicatio performance and components. This simplifies visibility and enables faster testing c specific application parts.
- **Easier deployment.** Instead of deploying one large application, you can deploy specifi services to support your application. Furthermore, you can deploy service independently without affecting downstream operations.
- **Improved distributed team functionality.** With independent services, teams can ow parts of the development effort. Each team can develop, deploy, and scale service independently of other teams.
- **Easier to understand.** Microservice size is relative to the project and overall applicatio However, loose coupling and service separation make microservices and the entir application architecture easier to understand. A developer can focus on one service o see how different independent services affect the application overall.
- **Faster performance.** Microservices are decoupled and independent. Therefore, DevOp teams can better control application performance, so applications can start faster an run more efficiently. This improved performance makes developers more productive an speeds deployments.
- **Improved fault isolation.** With independent, isolated services, you can find applicatio faults faster than with one monolithic executable. For example, you can isolate a memor leak to one service instead of an entire application. Additionally, you can adjust tha service while allowing other microservices to support the application. A single memor leak could take down the entire application in a monolithic architecture.

- **Less code and stack lock-in.** You're not limited to one codebase with microservices. This eliminates any long-term commitments to a technology stack. You can keep parts of the application on one platform while designing a new service on a different stack.

b. Microservices cons

- **Additional complexity.** Developers need to become accustomed to working with the complexity of a distributed system.
- **Transition challenges.** If an organization primarily uses monolithic applications, then it's more difficult for teams to develop distributed applications. Teams need to do an inventory of exiting tools and development practices before moving to microservices. Remember, it's often a massive undertaking to refactor an application built on monolithic architecture.
- **Complicated testing.** Because teams no longer work with one executable, they have more services and pieces of an application to test.
- **Interservice communication needs.** Unlike a single monolithic application, DevOps teams must ensure microservices can talk to one another. Developers need to deploy an interservice communication mechanism to support the app.
- **Careful deployments.** If an application spans multiple services, it will require careful coordination between teams to deploy it properly. Further, deploying and managing a system in production that's composed of different microservice types introduces complexity.
- **Increased resource consumption.** Rather than a single application instance, DevOps teams must provision resources — memory, CPU, and disk — to accommodate each cluster or service requirement.

2.5 Characteristics of Microservice Architecture

The following are the Characteristics of microservices :

Decoupling : Services within applications are primarily decoupled. So the application as a whole can be easily built, altered, deployed, and scaled.

Componentization : Microservices are treated as independent or autonomous components that can be easily replaced and upgraded.

Business capabilities : Microservices are simple and focus on a single business capability.

- **Autonomy** : Development teams can work independently of each other, thus increasing speed and agility for the business.
- **Continuous delivery** : Allows frequent releases of software through systematic automation of software creation, testing, approval, and deployment.
- **Responsibility** : Microservices do not treat applications as projects. Instead, they regard these applications as products for which they are solely responsible.
- **Decentralized governance** : The emphasis is on leveraging the right tool for the right requirement. This means there is no technology pattern or standardized pattern. The development team has the freedom to choose the best tools to solve the problems.
- **Business agility** : Microservices support the agile development methodology. Any new feature can be quickly developed and discarded.

2.6 Comparison of Monolithic Applications and Microservices

Sr. No.	Microservices architecture	Monolithic architecture
1.	Build as a suite of small services.	Build as a single logical executable.
2.	Requirement changes can be applied to each service independently.	Requirement changes involve building and deploying a new version of the entire application.
3.	Each service can be scaled independently.	Entire application has to be scaled in case a bottleneck is identified in one part.
4.	Each service can be developed in different programming languages.	Typically, the entire applications is developed in one programming language or framework.
5.	Smaller code base is easier to maintain and manage.	Large code base is intimidating to a new development team.
6.	Simple deployment as each service can be deployed individually with minimum downtime.	Complex deployment with maintenance windows and scheduled downtime.

Sr. No.	Microservices architecture	Monolithic architecture
7.	Microservices architecture is loosely coupled.	Monolithic architecture is primarily tightly coupled.
8.	Changes done in a single data model does not affect other microservices.	Any changes in the data model affect the entire database.

2.7 Microservices Best Practices

By following ten basic microservices best practices, you can achieve an efficient microservices ecosystem devoid of unnecessary architectural complexities. These best practices will help you create a robust, easy-to-manage, scalable, and secure system of intercommunicating microservices.

1. **The Single Responsibility Principle**

Just like with code, where a class should have only a single reason to change, microservices should be modeled in a similar fashion. Building bloated services which are subject to change for more than one business context is a bad practice.

Example : Let's say you are building microservices for ordering a pizza. You can consider building the following components based on the functionality each supports like Inventory Service, Order Service, Payments Service, User Profile Service, Delivery Notification Service, etc. Inventory Service would only have APIs that fetch or update the inventory of pizza types or toppings, and likewise others would carry the APIs for their functionality.

2. **Have a separate data store(s) for your microservice**

It defeats the purpose of having microservices if you are using a monolithic database that all your microservices share. Any change or downtime to that database would then impact all the microservices that use the database. Choose the right database for your microservice needs, customize the infrastructure and storage to the data that it maintains, and let it be exclusive to your microservice. Ideally, any other microservice that needs access to that data would only access it through the APIs that the microservice with write access has exposed.

Use asynchronous communication to achieve loose coupling

To avoid building a mesh of tightly coupled components, consider using asynchronous communication between microservices.

a. Make calls to your dependencies asynchronously, example below.

Example : Let's say you have a Service A that calls Service B. Once Service B returns response, Service A returns success to the caller. If the caller is not interested in Service B output, then Service A can asynchronously invoke Service B and instantly respond with success to the caller.

b. An even better option is to use events for communicating between microservices. You microservice would publish an event to a message bus either indicating a state change or failure and whichever microservice is interested in that event, would pick it up and process it

Example : In the pizza order system above, sending a notification to the customer once the order is captured, or status messages as the order gets fulfilled and delivered, can happe using asynchronous communication. A notification service can listen to an event that an orde has been submitted and process the notification to the customer.

4. Fail fast by using a circuit breaker to achieve fault tolerance

- If your microservice is dependent on another system to provide a response, and tha system takes forever to respond, your overall response SLAs will be impacted. To avo this scenario and quickly respond, one simple microservices best practice you can follo is to use a circuit breaker to timeout the external call and return a default response or a error.
- The Circuit Breaker pattern is explained in the references below. This will isolate th failing services that your service is dependent on without causing cascade failure keeping your microservice in good health. You can choose to use popular products lik Hystrix that Netflix developed.
- This is better than using the HTTP CONNECT_TIMEOUT and READ_TIMEOUT settings a it does not spin up additional threads beyond what's been configured.

5. Proxy your microservice requests through an API Gateway

- Instead of every microservice in the system performing the functions of AP authentication, request / response logging, and throttling, having an API gateway doin these for you upfront will add a lot of value. Clients calling your microservices wi connect to the API Gateway instead of directly calling your service.
- This way you will avoid making all those additional calls from your microservice and th internal URLs of your service would be hidden, giving you the flexibility to redirect th traffic from the API Gateway to a newer version of your service.

- This is even more necessary when a third party is accessing your service, as you can throttle the incoming traffic and reject unauthorized requests from the API gateway before they reach your microservice. You can also choose to have a separate API gateway that accepts traffic from external networks.

6. Ensure your API changes are backwards compatible

- You can safely introduce changes to your API and release them fast as long as they don't break existing callers. One possible option is to notify your callers , have them provide a sign off for your changes by doing integration testing.
- However, this is expensive, as all the dependencies need to line up in an environment and it will slow you down with a lot of coordination . A better option is to adopt contract testing for your APIs. The consumers of your APIs provide contracts on their expected response from your API.
- You as a provider would integrate those contract tests as part of your builds and these will safeguard against breaking changes. The consumer can test against the stubs that you publish as part of the consumer builds. This way you can go to production faster with independently testing your contract changes.

7. Version your microservices for breaking changes

- It's not always possible to make backwards compatible changes. When you are making a breaking change, expose a new version of your endpoint while continuing to support older versions. Consumers can choose to use the new version at their convenience.
- However, having too many versions of your API can create a nightmare for those maintaining the code. Hence, have a disciplined approach to deprecate older versions by working with your clients or internally rerouting the traffic to the newer versions.

8. Have dedicated infrastructure hosting your microservice

You can have the best designed microservice meeting all the checks, but with a bad design of the hosting platform it would still behave poorly. Isolate your microservice infrastructure from other components to get fault isolation and best performance. It is also important to isolate the infrastructure of the components that your microservice depends on.

Example : In the pizza order example above, let's say the inventory microservice uses an inventory database. It is not only important for the Inventory Service to have dedicated host machines, but also the inventory database needs to have dedicated host machines.

9. Create a separate release train

Your microservice needs to have its own separate release vehicle which is not tied to othe components within your organization. This way you are not stepping on each other's toes an wasting time coordinating with multiple teams.

10. Create Organizational Efficiencies

- While microservices give you the freedom to develop and release independently, certai standards need to be followed for cross cutting concerns so that every team doesn't spen time creating unique solutions for these.
- This is very important in a distributed architecture such as microservices, where you need t be able to connect all the pieces of the puzzle to see a holistic picture. Hence, enterpris solutions are necessary for API security, log aggregation, monitoring, API documentatio secrets management, config management, distributed tracing, etc.

2.8 Deployment Strategies

- A deployment strategy is a way to change or upgrade an application. The aim is to make th change without downtime in a way that the user barely notices the improvements
- The following sections will explain six deployment strategies.
  - **Recreate** : Version A is terminated then version B is rolled out.
  - **Ramped** (also known as rolling-update or incremental): Version B is slowly rolled ou and replacing version A.
  - **Blue/Green** : Version B is released alongside version A, then the traffic is switched t version B.
  - **Canary** : Version B is released to a subset of users, then proceed to a full rollout.
  - **A/B testing** : Version B is released to a subset of users under specific condition.
  - **Shadow** : Version B receives real-world traffic alongside version A and doesn't impac the response.
- There are multiple ways to deploy a new version of an application and it really depends o the needs and budget. When releasing to development/staging environments, a recreate o ramped deployment is usually a good choice. When it comes to production, a ramped o blue/green deployment is usually a good fit, but proper testing of the new platform i necessary.

- Blue/green and shadow strategies have more impact on the budget as it requires double resource capacity. If the application lacks in tests or if there is little confidence about the impact/stability of the software, then a canary, a/ b testing or shadow release can be used. If your business requires testing of a new feature amongst a specific pool of users that can be filtered depending on some parameters like geolocation, language, operating system or browser features, then you may want to use the a/b testing technique. a shadow release is complex and requires extra work to mock egress traffic which is mandatory when calling external dependencies with mutable actions (email, bank, etc.). However, this technique can be useful when migrating to a new database technology and use shadow traffic to monitor system performance under load.

A. Recreate

- In this strategy, the deployment is done when the application on each instance in the deployment group is stopped, the latest application revision is installed, and the new version of the application is started and validated. You can use a load balancer so that each instance is deregistered during its deployment and then restored to service after the deployment is complete. This technique implies downtime of the service that depends on both shutdown and boot duration of the application.
- Use this strategy if 1) your application service is not business, mission, or revenue-critical, or 2) your deployment is to a lower environment, during off-hours, or with a service that is not in use.

Pros :

- Easy to setup.
- Application state entirely renewed.

Cons :

- High impact on the user, expect downtime that depends on both shutdown and boot duration of the application.

B. Ramped

The ramped deployment strategy consists of slowly rolling out a version of an application by replacing instances one after the other until all the instances are rolled out. It usually follows the following process: with a pool of version A behind a load balancer, one instance of version B is deployed. When the service is ready to accept traffic, the instance is added to the pool. Then, one instance of version A is removed from the pool and shut down.

Pros :

- Easy to set up.
- Version is slowly released across instances.
- Convenient for stateful applications that can handle rebalancing of the data.

Cons :

- Rollout/rollback can take time.
- Supporting multiple APIs is hard.
- No control over traffic.

C. Blue/Green

- Blue-green deployment is a deployment strategy that utilizes two identical environments, a "blue" (aka staging) and a "green" (aka production) environment with different versions of an application or service.
- Quality assurance and user acceptance testing are typically done within the blue environment that hosts new versions or changes. User traffic is shifted from the green environment to the blue environment once new changes have been testing and accepted within the blue environment. You can then switch to the new environment once the deployment is successful.

Pros :

- Instant rollout/rollback.
- Avoid versioning issue, the entire application state is changed in one go.

Cons :

- Expensive as it requires double the resources. Cost is a drawback to blue-green deployments. Replicating a production environment can be complex and expensive especially when working with microservices.
- Proper test of the entire platform should be done before releasing to production.
- Handling stateful applications can be hard.

D. Canary

- A canary deployment consists of gradually shifting production traffic from version A to version B. Usually the traffic is split based on weight. For example, 90 percent of the requests go to version A, 10 percent go to version B.
- This technique is mostly used when the tests are lacking or not reliable or if there is little confidence about the stability of the new release on the platform. A canary release is the lowest risk-prone, compared to all other deployment strategies, because of this control.

Pros :

- Canary deployments allow organizations to test in production with real users and use cases and compare different service versions side by side. It's cheaper than a blue-green deployment because it does not require two production environments. And finally, it is fast and safe to trigger a rollback to a previous version of an application.

Con :

- Drawbacks to canary deployments involve testing in production and the implementations needed. Scripting a canary release can be complex: manual verification or testing can take time, and the required monitoring and instrumentation for testing in production may involve additional research.

E. A/B testing

- A/B testing deployments consists of routing a subset of users to a new functionality under specific conditions. It is usually a technique for making business decisions based on statistics, rather than a deployment strategy. However, it is related and can be implemented by adding extra functionality to a canary deployment so we will briefly discuss it here.
- This technique is widely used to test conversion of a given feature and only roll-out the version that converts the most.
- Here is a list of conditions that can be used to distribute traffic amongst the versions :
  - By browser cookie
  - Query parameters
  - Geolocalisation
  - Technology support: browser version, screen size, operating system, etc.
  - Language

Pros :

- Several versions run in parallel.
- Full control over the traffic distribution.

Cons :

- Requires intelligent load balancer.
- Hard to troubleshoot errors for a given session, distributed tracing becomes mandatory.

F. Shadow

- A shadow deployment consists of releasing version B alongside version A, fork version A's incoming requests and send them to version B as well without impacting production traffic. This is particularly useful to test production load on a new feature. A rollout of the application is triggered when stability and performance meet the requirements.
- This technique is fairly complex to setup and needs special requirements, especially with egress traffic. For example, given a shopping cart platform, if you want to shadow test the payment service you can end-up having customers paying twice for their order. In this case, you can solve it by creating a mocking service that replicates the response from the provider.

Pros :

- Performance testing of the application with production traffic.
- No impact on the user.
- No rollout until the stability and performance of the application meet the requirements.

Cons :

- Expensive as it requires double the resources.
- Not a true user test and can be misleading.
- Complex to setup.
- Requires mocking service for certain cases.

2.9 Introduction to cloud computing

- Cloud computing is the delivery of computing services-including servers, storage, databases networking, software, analytics, and intelligence-over the Internet ("the cloud") to offer faster innovation, flexible resources, and economies of scale. You typically pay only for cloud services you use, helping you lower your operating costs, run your infrastructure more efficiently, and scale as your business needs change.
- Following as the five essential characteristics of cloud computing :
  - **On-demand self-service** : Cloud resources can be accessed or provisioned without human interaction. With this model, consumers can gain immediate access to cloud services upon signup. Organizations can also create mechanisms for allowing employees customers, or partners to access internal cloud services on demand according to predetermined logics without needing to go through IT services.

  - **Broad network access** : Users can access cloud services and resources through any device and in any networked location provided that they have permission.
  - **Resource pooling** : Cloud provider resources are shared by multiple tenants while keeping the data of individual clients hidden from other clients.
  - **Rapid elasticity** : Unlike on-premise hardware and software, cloud computing resources can be rapidly increased, decreased, or otherwise modified based on the cloud user's changing needs.
  - **Measured service** : Usage of cloud resources is metered so that businesses and other cloud users need only pay for the resources they use in any given billing cycle.

2.10 Cloud Computing Deployment Models

There are three main models for cloud computing. Each model represents a different part of the cloud computing stack.

1. Infrastructure as a Service (IaaS)
2. Platform as a Service (PaaS)
3. Software as a Service (SaaS)

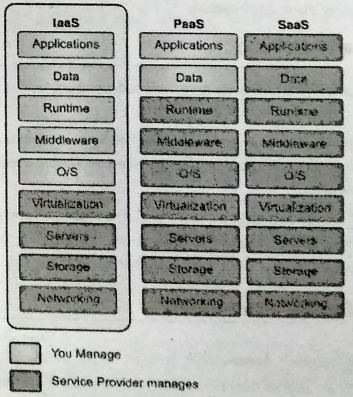


Fig. 2.10.1 : Cloud computing models

2.10.1 Infrastructure as a Service (IaaS)

- IaaS, contains the basic building blocks for cloud IT and typically provide access to networking features, computers (virtual or on dedicated hardware), and data storage space.
- IaaS means a cloud service provider manages the infrastructure for you the actual servers network, virtualization, and data storage through an internet connection.
- The user has access through an API or dashboard, and essentially rents the infrastructure The user manages things like the operating system, apps, and middleware while the provide takes care of any hardware, networking, hard drives, data storage, and servers; and has the responsibility of taking care of outages, repairs, and hardware issues.

2.10.2 Platform as a Service (PaaS)

- PaaS means the hardware and an application-software platform are provided and manage by an outside cloud service provider, but the user handles the apps running on top of th platform and the data the app relies on.
- Platforms as a service remove the need for organizations to manage the underlyin infrastructure (usually hardware and operating systems) and allow you to focus on th deployment and management of your applications.
- This helps you be more efficient as you don't need to worry about resource procuremen capacity planning, software maintenance, patching, or any of the other undifferentiated heav lifting involved in running your application.

2.10.3 Software as a Service (SaaS)

- Software as a Service provides you with a completed product that is run and managed by th service provider.
- With a SaaS offering you do not have to think about how the service is maintained or how th underlying infrastructure is managed; you only need to think about how you will use thi particular piece of software.
- A common example of a SaaS application is web-based email where you can send and receiv email without having to manage feature additions to the email product or maintaining th servers and operating systems that the email program is running on.

2.11 Cloud Computing Service Models

- The cloud deployment models denote the specific type of cloud environment based o ownership, size, and access. It also describes the cloud's nature and purpose. Most business use cloud infrastructure to reduce capital expenditure and control operating costs.

- A cloud deployment model signifies a specific cloud environment based on who controls security, who has permissions, and whether resources are shared or dedicated. Cloud deployment models describe how cloud services are made available to users. The four cloud computing deployment models are as follows:
- Different types of cloud computing deployment models are :

1. Public cloud
2. Private cloud
3. Hybrid cloud
4. Community cloud
5. Multi-cloud

1. Public cloud :

- This type of cloud is open to the public. Public cloud deployment models are ideal for organizations with fluctuating and growing demands. It is a perfect choice for businesses with low-security concerns. As a result, you pay a cloud service provider for networking, computing, virtualization, and storage accessible via the public internet.
- It is also an excellent delivery model for development and testing teams. Its quick and simple configuration and deployment make it ideal for test environments. Some of the largest public cloud providers include Alibaba Cloud, Amazon Web Services (AWS), Google Cloud, IBM Cloud, and Microsoft Azure.

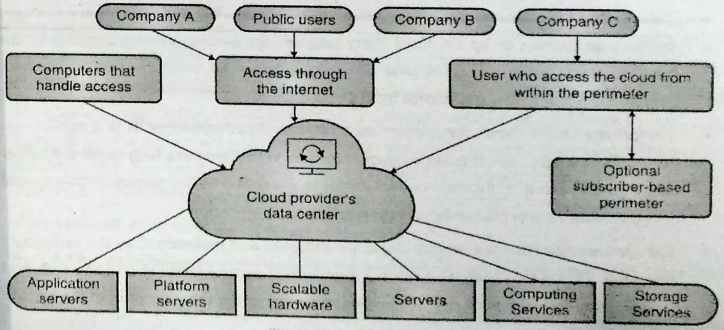


Fig. 2.11.1 : Public Cloud

Benefits of Public Cloud

- **Minimal Investment –** There is no initial investment because it is a pay-per-use service. Making it ideal for businesses that require immediate access to resources.
- **No Hardware Set-up –** No hardware installation is required because the cloud service providers fully fund the entire infrastructure
- **No Infrastructure Management –** Does not require an in-house team to utilize the public cloud

Limitations of Public Cloud

- **Data Security and Privacy Concerns –** Since it is accessible to all, it does not fully protec against cyber-attacks and could lead to vulnerabilities
- **Reliability Issues –** Since the same server network is open to a wide range of users, it ca lead to malfunction and outages
- **Service/License Limitation –** While there are many resources that you can exchange wit tenants, there is a cap on usage
- **Less control -** You are not in control of the systems that host your business applications. I the unlikely event that a public cloud platform fails, you do not have access to ensur continuity as would be the case with a traditional server room or data center environment.

2. Private Cloud

- This is almost similar features as the public cloud, but the data and services are manage by the organization or by the third party only for the customer's organization. In thi type of cloud, major control is over the infrastructure so security-related issues ar minimized which makes it different from a public cloud.
- The private cloud allows for greater control over cloud resources. It is a one-on-on setting for a single user. It is not essential to share your hardware with anyone. Anothe name for this cloud is "internal cloud." And it refers to the ability to access systems an services within a specific border or organization.
- The systems that run on a private cloud are designed and maintained by the company' own staff. This means that the company that runs a private cloud must have technic staff on hand to assist with any issues that come up during the operation of the privat cloud.

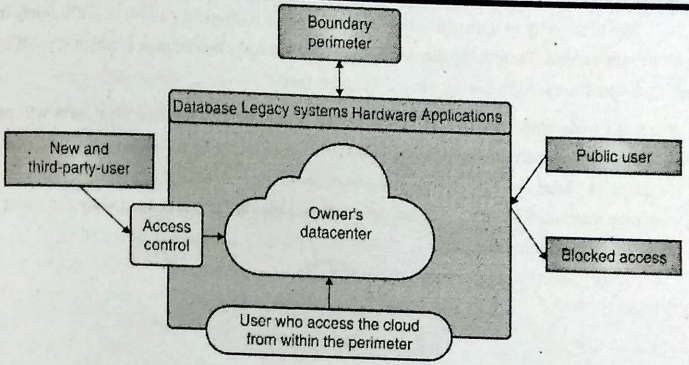


Fig. 2.11.2 : Private Cloud

enefits of Private Cloud

Increased control. Less people will have access to the administration and configuration of the back end infrastructure that powers your private cloud, which gives you more control.

Customization. If there is a business case for a new feature, you can have it developed and deployed in house, giving you more options than a publicly available cloud.

Highly secure. You can incorporate as many security services as you want in order to secure your cloud. Two-Factor Authentication is far more secure when combined with security best practices such as complex passwords and mandatory password changes.

isadvantages of Private Cloud

Learning curve. To take advantage of being able to customize your private cloud, you need the right technical skills. Developers, cyber security experts, and DevOps professionals are all roles that you need to fill in order to effectively develop a solution on your private cloud.

Cost. All but the largest companies in the world can afford to set up their own private cloud infrastructure. The hardware costs alone are prohibitively expensive for most companies. There's also the costs of keeping skilled staff and other infrastructure costs.

Hybrid Cloud

- As the name suggests Hybrid, is the combination of both private and public cloud. The decision to choose a type of cloud i.e. private or public usually depends on various parameters like the sensitivity of data and applications, industry certifications and required standards, regulations, and many more.

- Due to security or data protection concerns, some businesses cannot operate solely in public cloud. To address this issue, they may opt for a hybrid cloud, which combines requirements with the benefits of a public cloud.
- Let's understand the hybrid model better. A company that has critical data will pre storing on a private cloud, while less sensitive data can be stored on a public cloud. T hybrid cloud is also frequently used for 'cloud bursting'. It means, suppose organization runs an application on-premises, but due to heavy load, they can burst t the public cloud.

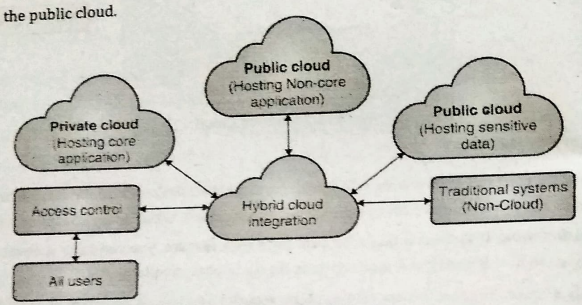


Fig. 2.11.3 : Hybrid Cloud

Benefits of Hybrid Cloud

- **Cost-Effectiveness** – The overall cost of a hybrid solution decreases since it majorly uses public cloud to store data
- **Security** – Since data is properly segmented, the chances of data theft from attacker significantly reduced
- **Flexibility** – With higher levels of flexibility, businesses can create custom solutions tha their exact requirements

Limitations of Hybrid Cloud

- **Complexity** – It is complex setting up a hybrid cloud since it needs to integrate two or m cloud architectures
- **Specific Use Case** – This model makes more sense for organizations that have multiple cases or need to separate critical and sensitive data

4. Community Cloud

- The community cloud operates in a way that is similar to the public cloud. There's just one difference – it allows access to only a specific set of users who share common objectives and use cases.
- A community cloud is basically a multi-tenant hosting platform that usually involves similar industries and complimentary businesses with shared goals all using the same hardware. By sharing the infrastructure between multiple companies, community cloud installations are able to save their members money. Data is still segmented and kept private, except in areas where shared access is agreed upon and configured.

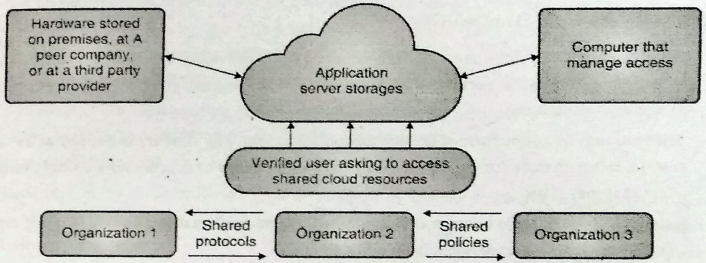


Fig. 2.11.5 : Community Cloud

Benefits of Community Cloud

- **Cost Savings.** As we touched on before, the main benefit of using this kind of setup is that there are cost savings. This is because all of the users that access the community cloud will share the costs to create an equitable experience.
- **Security between tenants.** If the security policies are aligned and if everyone follows the same standards then the community cloud model is very secure.
- **Enhanced collaboration.** When there is a shared goal then having everyone on the same platform creates more opportunities to work together towards the same objectives.

Disadvantages of Community Cloud

- **Technical requirements.** A community cloud has to agree upon a set of standards and then coordinate across that cloud. This means that each stakeholder must have their own technical resources available to enforce the policies.
- **Data isolation.** Security and segmentation is difficult to maintain.

- **Rarity.** This model is not widely used, yet, so there are not too many resources available fo people to learn from or well known examples.

5. **Multi-cloud**
   - Multi-cloud" means multiple public clouds. Multi-cloud is a model of cloud computin where an organization utilizes a combination of clouds—which can be two or mor public clouds.
   - A company that uses a multi-cloud deployment incorporates multiple public clouds fro more than one cloud provider. multi-cloud deployment improves the high availability o your services.

Advantages of a multi-cloud model :

- Flexibility to choose cloud services from different cloud providers based on the combinatio of pricing, performance, security and compliance requirements, geographical location tha best suits the business;
- Ability to rapidly adopt "best-of-breed" technologies from any vendor, as needed or as the emerge, rather than limiting customers to whatever offerings or functionality a single vendo offers at a given time;
- Reduced vulnerability to outages and unplanned downtime (because an outage on one clou won't necessarily impact services from other clouds);
- Reduced exposure to the licensing, security, compatibility and other issues that can resu from "shadow IT" - users independently signing up for cloud services that an organizatio using just one cloud might not offer.

2.12 Why to use cloud

Cloud computing is a big shift from the traditional way businesses think about IT resources. Here are seven common reasons organizations are turning to cloud computing services :

1. **Cost :** Cloud computing eliminates the capital expense of buying hardware and software an setting up and running on-site data centers—the racks of servers, the round-the-cloc electricity for power and cooling, and the IT experts for managing the infrastructure. It add up fast.
2. **Speed :** Most cloud computing services are provided self service and on demand, so even vas amounts of computing resources can be provisioned in minutes, typically with just a few mouse clicks, giving businesses a lot of flexibility and taking the pressure off capacit planning.

3. **Global scale :** The benefits of cloud computing services include the ability to scale elastically. In cloud speak, that means delivering the right amount of IT resources—for example, more or less computing power, storage, bandwidth—right when they're needed, and from the right geographic location.
4. **Productivity :** On-site datacenters typically require a lot of "racking and stacking"—hardware setup, software patching, and other time-consuming IT management chores. Cloud computing removes the need for many of these tasks, so IT teams can spend time on achieving more important business goals.
5. **Performance :** The biggest cloud computing services run on a worldwide network of secure datacenters, which are regularly upgraded to the latest generation of fast and efficient computing hardware. This offers several benefits over a single corporate datacenter, including reduced network latency for applications and greater economies of scale.
6. **Reliability :** Cloud computing makes data backup, disaster recovery, and business continuity easier and less expensive because data can be mirrored at multiple redundant sites on the cloud provider's network.
7. **Security :** Many cloud providers offer a broad set of policies, technologies, and controls that strengthen your security posture overall, helping protect your data, apps, and infrastructure from potential threats.

2.13 Principle of Container based Application Design

Following these principles would ensure that the resulting containers behave like a good loud-native citizen in most container orchestration engines, allowing them to be scheduled, caled, and monitored in an automated fashion.

Single Containment Principle : Each container addresses a single concern and does it well.

Self-Containment Principle : A container relies only on the presence of the Linux kernel. Additional libraries are added when the container is built.

Image Immutability Principle : Containerized applications are meant to be immutable, and once built are not expected to change between different environments.

High Observability Principle : Every container must implement all necessary APIs to help the platform observe and manage the application in the best way possible.

Lifecycle Conformance Principle : A container must have a way to read events coming from the platform and conform by reacting to those events.

6. **Process Disposability Principle** : Containerized applications must be as ephemeral a possible and ready to be replaced by another container instance at any point in time.
7. **Runtime Confinement Principle** : Every container must declare its resource requirement and restrict resource use to the requirements indicated.

1. Single Containment Principle

- Each container addresses a single concern and does it well. This principle is more like th single responsibility principle (SRP), which advises that a class should have only on responsibility.
- If your containerized microservice needs to address multiple concerns, it can us patterns such as sidecar and init-containers to combine multiple containers into a sing deployment unit (pod), where each container still handles a single concern.

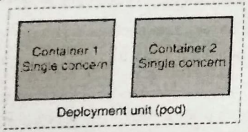


Fig. 2.13.1 : Single Concern Principle

2. Self-Containment Principle (S-CP)

- This principle dictates that a container should contain everything it needs at build time. container should rely only on the presence of the Linux kernel and have any addition libraries added to it at the time the container is built.
- The only exceptions are things such as configurations, which vary between differe environments and must be provided at runtime; for example, through Kubernet ConfigMap.

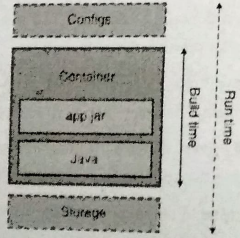


Fig. 2.13.2 : Self-Containment Principle

- Some applications are composed of multiple containerized components. For example, a containerized web application may also require a database container. This principle does not suggest merging both containers. Instead, it suggests that the database container contains everything needed to run the database, and the web application container contains everything needed to run the web application, such as the webserver. At runtime, the web application container will depend on and access the database container as needed.

Image Immutability Principle

- Containerized applications are meant to be immutable, and once built are not expected to change between different environments.
- This implies the use of an external means of storing the runtime data and relying on externalized configurations that vary across environments, rather than creating or modifying containers per environment. Any change in the containerized application should result in building a new container image and reusing it across all environments.

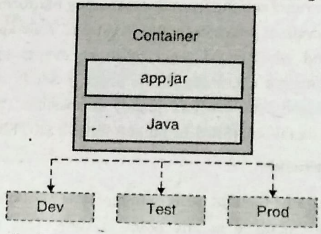


Fig. 2.13.3 : Image Immutability Principle

High Observability Principle

- Containers provide a unified way for packaging and running applications by treating them like a black box. Every container must implement all necessary APIs to help the platform observe and manage the application in the best way possible.

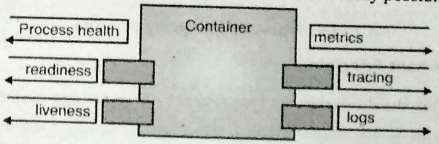


Fig. 2.13.4 : High Observability Principle

- Your containerized application must provide APIs for the different kinds of hea checks—liveness and readiness. The application should log important events into s standard error (STDERR) and standard output (STDOUT) for log aggregation by to such as Fluentd and Logstash and integrate with tracing and metrics-gathering librar such as Open Tracing, Prometheus, and others.

5. Lifecycle Conformance Principle

- A container should have a way to read the events coming from the platform and confo by reacting to those events.

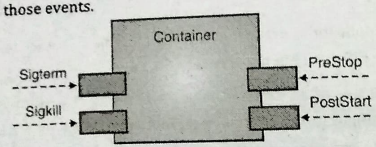


Fig. 2.13.5 : Lifecycle Conformance Principle

- There are all kind of events coming from the managing platform that are intended to h you manage the life cycle of your container. It is up to your application to decide wh events to handle and whether to react to those events or not. For example, a application that requires a clean shutdown process needs to catch signal: termin (SIGTERM) messages and shut down as quickly as possible. This is to avoid the force shutdown through a signal: kill (SIGKILL) that follows a SIGTERM.

6. Process Disposability Principle

- Containerized applications need to be as ephemeral as possible and ready to be replac by another container instance at any point in time. This means that containeriz applications must keep their state externalized or distributed and redundant. It a means the application should be quick in starting up and shutting down, and even ready for a sudden, complete hardware failure.

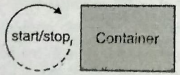


Fig.2.13.6 : Process Disposability Principle

7. Runtime Confinement Principle

- Every container should declare its resource requirements and pass that information the platform. It is also important that the application stay confined to the indicat resource requirements.

- There source requirements is declared in terms of CPU, memory, networking, disk influence on how the platform performs scheduling, auto-scaling, capacity management, and the general service-level agreements (SLAs) of the container.

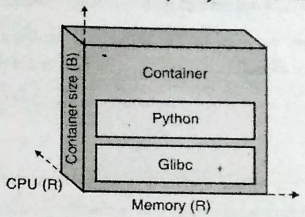


Fig. 2.13.7 : Runtime Confinement Principle

- In addition to passing the resource requirements of the container, it is also important that the application stay confined to the indicated resource requirements.

2.14 Introduction to Docker

Docker is a containerisation platform – it is a toolkit that allows you to build, deploy and manage containerised applications. There are alternative containerisation platforms, such as podman, however, Docker is the leading player in this space.

- Docker is an open source platform, free to download. There is also Docker Inc, the company that sells the commercial version of Docker. Docker comes with a command line interface (CLI), using which you can do all of the operations that the platform provides.
- Docker is a Linux-based, open-source containerization platform that developers use to build, run, and package applications for deployment using containers.

A. Difference between Docker Containers and Virtual Machines

1. Docker Containers

- Docker Containers contain binaries, libraries, and configuration files along with the application itself.
- They don't contain a guest OS for each container and rely on the underlying OS kernel, which makes the containers lightweight.
- Containers share resources with other containers in the same host OS and provide OS-level process isolation.

2. Virtual Machines

- Virtual Machines (VMs) run on Hypervisors, which allow multiple Virtual Machines to run on a single machine along with its own operating system.
- Each VM has its own copy of an operating system along with the application and necessary binaries, which makes it significantly larger and it requires more resources.
- They provide Hardware-level process isolation and are slow to boot.

B. Docker terminology

- **Images :** The blueprints of our application which form the basis of containers. These contain all of the configuration settings that define the isolated environment.
- **Containers :** Are instances of a Docker image and are what run the actual application.
- **Docker Daemon :** That background service running on the host that listens to API calls (via the Docker client), manages images and building, running and distributing containers. The Deamon is the process that runs in the operating system which the client talks to – playing the role of the broker.
- **Docker Client :** The command line tool that allows the user to interact with the daemon. There are other forms of clients too.
- **Docker Hub :** A registry of Docker images containing all available Docker images. A user can have their own registry, from which they can pull images.
- **Base images** are images that have no parent image – they don't build on or derive from another image, usually images that represent an operating system (e.g. Ubuntu, busybox).
- **Child images** are images that build on base images and add additional functionality. most images you're likely to make will be child images.
- **Official images** are images that are officially maintained and supported by the people at Docker. These are typically one word long. Examples include python, ubuntu, and hello-world.
- **User images** are images created and shared by people who use Docker. They usually build on base images and add functionality. Typically these are formatted as user/image-name.

C. Benefits of Docker in the SDLC

There are numerous benefits that Docker enables across an application architecture. These are some of the benefits that Docker brings across multiple stages of the software development lifecycle (SDLC) :

- **Build.** Docker allows development teams to save time, effort, and money by *dockerizing* their applications into single or multiple modules. By taking the initial effort to create an image tailored for an application, a build cycle can avoid the recurring challenge of having multiple versions of dependencies that may cause problems in production.
- **Testing.** With Docker, you can independently test each containerized application (or its components) without impacting other components of the application. This also enables a secured framework by omitting tightly coupled dependencies and enabling superior fault tolerance.
- **Deploy & maintain.** Docker helps reduce the friction between teams by ensuring consistent versions of libraries and packages are used at every stage of the development process. Besides, deploying an already tested container eliminates the introduction of bugs into the build process, thereby enabling an efficient migration to production.

2.15 Serverless computing

Serverless computing is an architecture where code execution is fully managed by a cloud provider, instead of the traditional method of developing applications and deploying them on servers.

Serverless is a cloud computing application development and execution model that enables developers to build and run application code without provisioning or managing servers or backend infrastructure.

Serverless lets developers put all their focus into writing the best front-end application code and business logic they can. All developers need to do is write their application code and deploy it to containers managed by a cloud service provider. The cloud provider handles the rest, provisioning the cloud infrastructure required to run the code and scaling the infrastructure up and down on demand as needed. The cloud provider is also responsible for all routine infrastructure management and maintenance such as operating system updates and patches, security management, capacity planning, system monitoring and more.

Serverless computing offerings typically fall into two groups, Backend-as-a-Service (BaaS) and Function-as-a-Service (FaaS).

- BaaS gives developers access to a variety of third-party services and apps. For instance, cloud-provider may offer authentication services, extra encryption, cloud-accessibi databases, and high-fidelity usage data. With BaaS, serverless functions are usually calle through application programming interfaces (APIs).
- More commonly, when developers refer to serverless, they're talking about a FaaS mode Under FaaS, developers still write custom server-side logic, but it's run in containers full managed by a cloud services provider.
- Today every leading cloud service provider offers a serverless platform including Amazo Web Services (AWS Lambda), Microsoft Azure (Azure Functions), Google Cloud (Google Clou Functions) and IBM Cloud (IBM Cloud Code Engine).
- Knative is an open source community project which adds components for deploying, runnin and managing serverless apps on Kubernetes. The Knative serverless environment lets yo deploy code to a Kubernetes platform, like Red Hat OpenShift. With Knative, you create service by packaging your code as a container image and handing it to the system. Your cod only runs when it needs to, with Knative starting and stopping instances automatically Knative can run in any cloud platform that runs Kubernetes.

A. Advantages of serverless computing

- **Lower costs** - Serverless computing is generally very cost-effective, as traditional clou providers of backend services (server allocation) often result in the user paying fo unused space or idle CPU time. organizations only pay for the compute resources the actually use in a very granular fashion, rather than buying physical hardware or rentin cloud instances that mostly sit idle.
- **Simplified scalability** - developers can focus on the business goals of the code the write, rather than on infrastructural questions Developers using serverless architectur don't have to worry about policies to scale up their code. The serverless vendor handle all of the scaling on demand.
- **Simplified backend code** - With serverless, developers can create simple functions tha independently perform a single purpose, like making an API call.
- **Quicker turnaround** - Serverless architecture can significantly cut time to market Instead of needing a complicated deploy process to roll out bug fixes and new features developers can add and modify code on a piecemeal basis.

- **Develop in any language** - Serverless is a polyglot environment, enabling developers to code in any language or framework—Java, Python, JavaScript, node.js—with which they're comfortable.
- **Streamlined development/DevOps cycles** - Serverless simplifies deployment and, in a larger sense, simplifies DevOps because developers don't spend time defining infrastructure required to integrate, test, deliver and deploy code builds into production.

B. Serverless use cases /Application

- Serverless architecture is ideal for asynchronous, stateless apps that can be started instantaneously. Likewise, serverless is a good fit for use cases that see infrequent, unpredictable surges in demand.
- Think of a task like batch processing of incoming image files, which might run infrequently but also must be ready when a large batch of images arrives all at once. Or a task like watching for incoming changes to a database and then applying a series of functions, such as checking the changes against quality standards, or automatically translating them.
- Serverless apps are also a good fit for use cases that involve incoming data streams, chat bots, scheduled tasks, or business logic.
- Some other common serverless use cases are back-end APIs and web apps, business process automation, severless websites, and integration across multiple systems.

2.16 Orchestration

Cloud orchestration can be used to provision or deploy servers, assign storage capacity, create virtual machines, and manage networking, among other tasks. There are many different orchestration tools that can help you with cloud orchestration.

You can use orchestration to automate IT processes such as server provisioning, incident management, cloud orchestration, database management, application orchestration, and many other tasks and workflows.

Orchestration by definition is the automated configuration, coordination, and management of computer systems and software.

In enterprise IT, orchestrating a process requires :

1. Knowing and understanding the many steps involved.

2. Tracking each step across a variety of environments: applications, mobile devices, databases, for instance.

- The goal of orchestration is to streamline and optimize frequent, repeatable proces orchestration can be used to optimize the process in order to eliminate redundancies.
- The main use cases for orchestration include:
  - Speedier software development
  - Batch processing daily transactions
  - Managing many servers and applications
  - Data analytics

Benefits of IT Orchestration

To simplify and manage cloud services deployments, enterprises turn to orchestration to IT orchestration offers several benefits, such as :

- **Unifies automation:** Many companies unify their workflow by automating some of t processes. It's not the same as a fully unified automation platform as cloud orchestra provides. Using cloud orchestration, you can centralize your automation into one place. result, the process is more cost-effective, faster, and easier to change or alter in the future.
- **Streamlines optimization:** Cloud orchestration combines several tasks into an effic workflow. Unlike automation, orchestration performs these individual tasks at the same ti
- **Better visibility and control:** For many organizations, managing volumes of vir machines can be a problem. It leads to excessive waste of financial resources & ma platform management a challenge.
- **Facilitates business agility:** Digital enablement is a hot topic among businesses today achieve it, IT companies must adapt and take advantage of emerging opportuni Orchestration enables rapid flexibility & digital transformation.
- **Saves cost:** By eliminating manual processes, orchestration reduces human labor& incre productivity. Organizations can also channel the extra time into more productive operatio

2.17 Difference between Orchestration and Automation

- Enterprises face a burning concern to increase productivity, grow revenues, & lower operat costs. With the increasing workloads of these businesses, there's a need for more autom processes. At first glance, automation and orchestration look similar, but they are not. You hav merge automation and orchestration to improve productivity, workflow, and reduce IT costs.

- Automation refers to automating a single process or a small number of related tasks (e.g., deploying an app). Orchestration refers to managing multiple automated tasks to create a dynamic workflow (e.g., deploying an app, connecting it to a network, and integrating it with other systems).
- Technically, **automation is a subset of orchestration** as you cannot orchestrate manual, non-automated tasks.
- Automation is a simple "if this, then that" process, orchestration has many moving parts and requires advanced logic that can :
  - Make decisions based on an output from an automated task.
  - Branch out into different steps and actions.
  - Adapt to changing circumstances and conditions.
  - Coordinate multiple tasks at the same time.
- While automation can be seen as a first step for starting, orchestration tools can be a further step. Orchestration maintains the coordination of numerous automated tasks. By using an orchestration tool, numerous automated tasks can be managed easily. Therefore, if you are a new starter, automation will be best for your business. If you have many automated tasks and struggle to manage them, then, orchestration can help you to coordinate them.
- Automation enables you to set up a single process to run without human involvement. Instead of a staff member performing a task by hand, a computer can follow a script and reliably perform a task quickly and as many times as necessary.
- In automation, the sysadmin has already made all task-related decisions. A computer only executes a "recipe" of instructions and fulfills a single goal.
- Automation makes time-intensive processes more efficient and reliable. A business can automate various repetitive tasks both on-premises and in the cloud, including :
  - Provisioning a virtual server when there is a spike in traffic or usage.
  - Transferring customer data from a CRM.
  - Launching a web or app server.
  - Sending a PDF to a customer after a user submits an online form.
  - Integrating a software API.
  - Deploying security measures on an endpoint if an Intrusion Detection System (IDS) discovers a threat.
  - Sending info to a third-party app (e.g., a ticketing system).

- Changing a line of code in all JSON or YAML files.
- Sending an automatic email based on different stages of a marketing funnel.
- Adding a record to a database at the start of a batch job.

- Below are some special key differences between the two for you to understand.

Feature	Orchestration	Automation
Concept	It can automate a set of multiple tasks at once. It automates arrangement, management and coordination of computer systems and services for executing a larger workforce.	It sets up single and exclusive tasks like web server launching and configuring, service termination etc. on its own.
Resource Utilization	It enumerates resources, Identity, and Access Management (IAM) roles, etc. for achieving accurate results, while surging the speed of operations. It utilizes the resources more efficiently.	Cloud Automation employs resources independently for automating tasks.
Performance	It focuses on ensuring comprehensive performance of all undergoing automated tasks in a definite order concerning one another and within a workflow. It also optimizes coding and thereby preventing errors.	It involves huge amounts of coding and is conducted in a well- defined sequence under strict policies and security guidelines.
Role of Personnel	Cloud Orchestration requires less intervention from personnel.	Engineers must finish all the manual tasks to deliver a new environment.
Monitoring and Alerting	It only requires monitoring and alerting for its workflows.	Cloud Automation can send the data to a third party reporting services.
Policy Decisions	It handles all permissions and security of automation tasks.	It does not typically implement any policy decisions which fall outside of OS- level ACLs.

REVIEW QUESATIONS

Q. 1 Explain monolithic applications.

Q. 2 Explain Microservice Architecture.

Q. 3 Explain how to implement microservice architecture.

Q. 4 What are the pros and cons of microservice architecture?

Q. 5 Explain characteristics of microservice architecture.

Q. 6 Write the difference between Monolithic and microservice applications.

Q. 7 What are the best practices in microservice?

Q. 8 Explain deployment strategies in microservice.

Q. 9 Write a short note on cloud computing.

Q. 10 Which are the cloud computiong deployment models?

Q. 11 Explain cloud computing service models.

Q. 12 Why to use Cloud?

Q. 13 Explain the principle of container based application design.

Q. 14 What is Docker ? Explain it.

Q. 15 What do you mean by Serverless computing?

Q. 16 What is Orchestration ? Explain it.

Q. 17 What is the difference between Orchestration and automation?

□□□