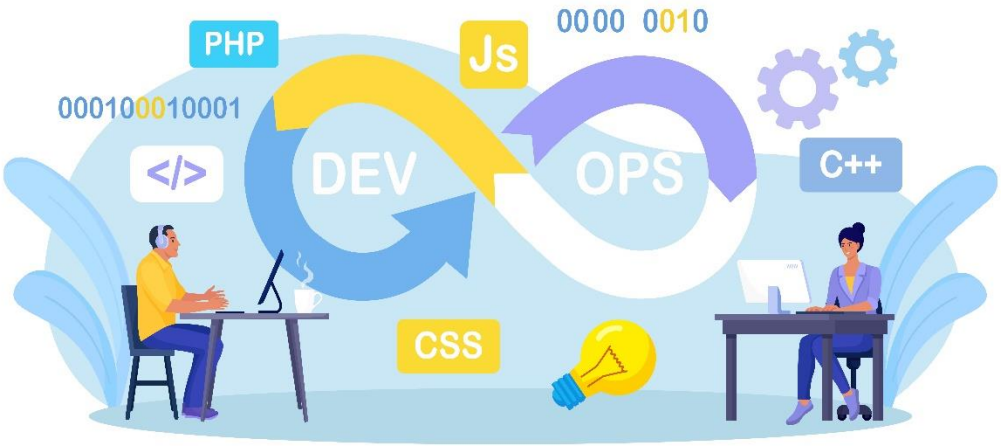
Exploring DevOps:

An Introduction for Beginners



NAISIT
Authored by: Tiny Content
ISBN: 978-1-927041-22-2

Philip Rolfe - philip.rolfe@georgebrown.ca

Contents

Chapter 1: Introduction to DevOps, Bash & CLI

Introduction

Today, we exist in the era of computers, the internet, smart applications, etc. Hence, it has become common to encounter jargon like Linux Bash, DevOps, CLI, SecOps, etc., thrown at us. What do these terms mean, and how can they help the industry and business perform better? This chapter takes us through DevOps, CLI, Bash, and Linux concepts and simplifies them for everyone to understand their significance.

What Is DevOps?

The expanded form of DevOps is Development and Operations. The DevOps approach represents a collaborative effort between the organization's application development and the IT operations teams. DevOps is not a technology but a philosophy promoting better communication between different teams in an organization.

DevOps can also be described as adopting automation, iterative software development, and programmable infrastructure deployment and maintenance. So, it is also defined as one of the techniques IT teams use to execute IT projects that satisfy business requirements.

DevOps is also a philosophy that covers cultural changes, including building trust and collaboration between system administrators and developers to align technological projects to business requirements.

DevOps environments use multiple common methodologies, like

a. CI/CD (Continuous integration and Continuous deployment/delivery) tools emphasizing task automation
b. Systems/tools supporting real-time monitoring, configuration management, incident management, and collaboration platforms
c. Concurrent implementation of cloud computing, microservices, and containers.

The best feature of DevOps is its adaptability because it can coexist with Agile, ITIL, Lean, Sigma, and other strategies.

Why DevOps?

While businesses face challenges, some problems are common to multiple departments, such as delayed releases, inefficient software that falls short of expectations, and other IT issues limiting business growth. DevOps can come to the rescue because it does not have lengthy wait times and manual processes. The shorter cycle times between requirements and live software can prevent shifting requirements and ensure customers get the desired products.

(Image Source: Freepik.com)

DevOps eliminates communication and priority issues between IT specializations. For example, development teams must understand production equipment and test codes in realistic conditions to create effective software solutions. Usually, organizations have development and operations teams working in different silos without proper communication channels. As a result, developers consider their jobs over when their code delivers functionality, and the operations teams are saddled with fixing problems that arise during the operation process.

DevOps culture removes this distinction because of increased collaboration between development and operations teams. The problems are resolved immediately during development by making small and reversible changes. Therefore, the teams understand the changes, significantly simplifying the incident management process.

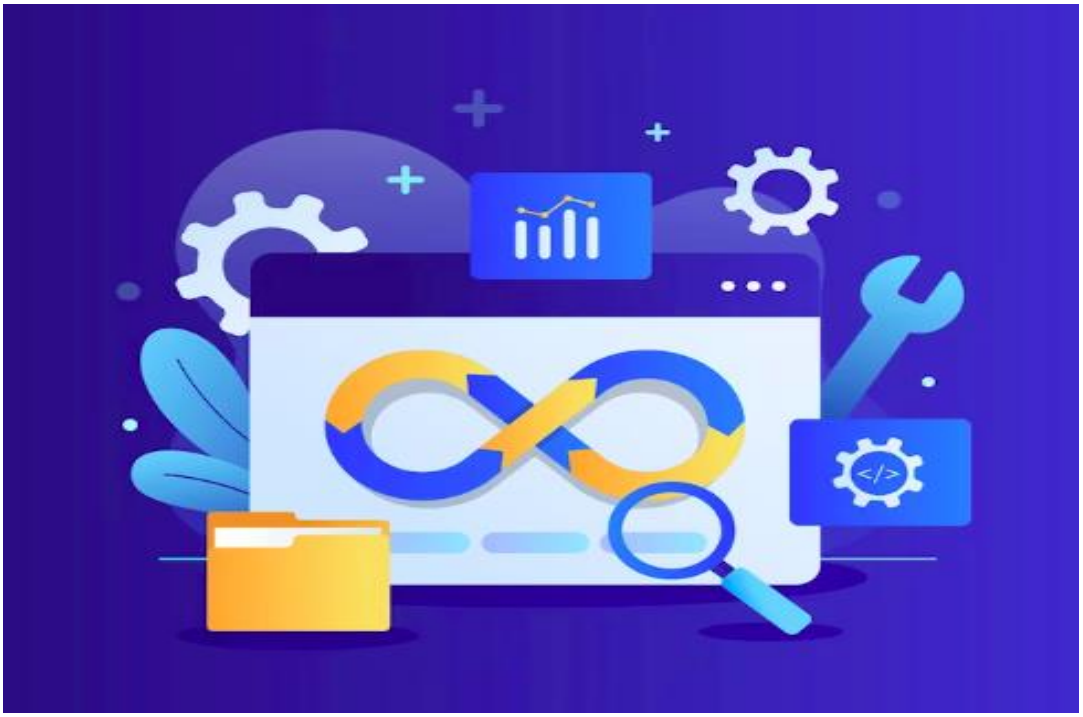
Thus, it expedites the process from the idea to the live software stage, allowing businesses to capitalize on market opportunities. Therefore, companies get a competitive advantage because of DevOps.

Future of DevOps

DevOps is a recent development that has been in existence since 2009. Patrick Debois coined this term to address the shortcomings of Agile software development methodologies.

DevOps has a bright future in various industrial sectors like IT, telecommunications, healthcare, hospitality, banking, insurance, manufacturing, and inventory management.

Similarly, DevOps has tremendous demand in countries like India, the US, Canada, the UK, Australia, Singapore, Dubai, etc. DevOps professionals have excellent employment scope because companies worldwide are committed to continuously delivering high-quality code with accuracy, speed, stability, and improved security.



(Image Source: Freepik.com)

What is a Build Pipeline?

By now, you know what DevOps is and how it works. The advantage of DevOps is that it collaborates the working of two siloed departments, development and operations. When these teams work together, the code-checking and deployment process is expedited. At the same time, new features and fixes are consistently fed into the cycle. Therefore, the entire process, from planning to deployment and monitoring, is known as a DevOps Build Pipeline. Let us discuss its components and how to build a DevOps Build Pipeline.

Software engineering teams use the DevOps pipeline, a set of tools/automated processes to compile, build, and deploy code. This process allows companies to develop, test, and deploy new code continuously. The objective is to eliminate the manual process, introduce automation, and reduce the possibilities of human error.

Here are the components of a DevOps Pipeline

- **Continuous Integration and Continuous Delivery (CI/CD)**

  CI integrates small pieces of new code built by multiple developers into a shared repository. Simultaneously, it allows testing the codes for errors and identifying bugs. CD is a natural extension of CI that enables developers to intensify their testing and ensure bug-free deployment. DevOps teams use CD to deliver bug fixes, expedite new releases, and automate the software release with the objective of reducing the project's cost and implementation.

- **Continuous Testing (CT)**

  CT enables companies to carry out automated testing at each stage of the development process. It allows the quick evaluation of code integration risks. Once integrated, the tests run automatically.

- **Continuous Deployment**

  The continuous deployment stage begins after the automation of the release cycle, where the code updates are directly sent to the end user without any manual intervention. It is recommended for minor updates alone because the bugs can be released, and the app could fail if they have not been detected in time. However, it allows for rolling back the changes. The benefit of continuous deployment is that it enables frequent deployments in a single day.

- **Continuous Monitoring**

It allows quick detection of compliance and security risks to empower SecOps teams by providing real-time info from multiple sources, like the company's IT infrastructure and other critical security processes, including incident response. Forensics, threat intelligence, and root cause analysis.

- **Continuous Feedback**

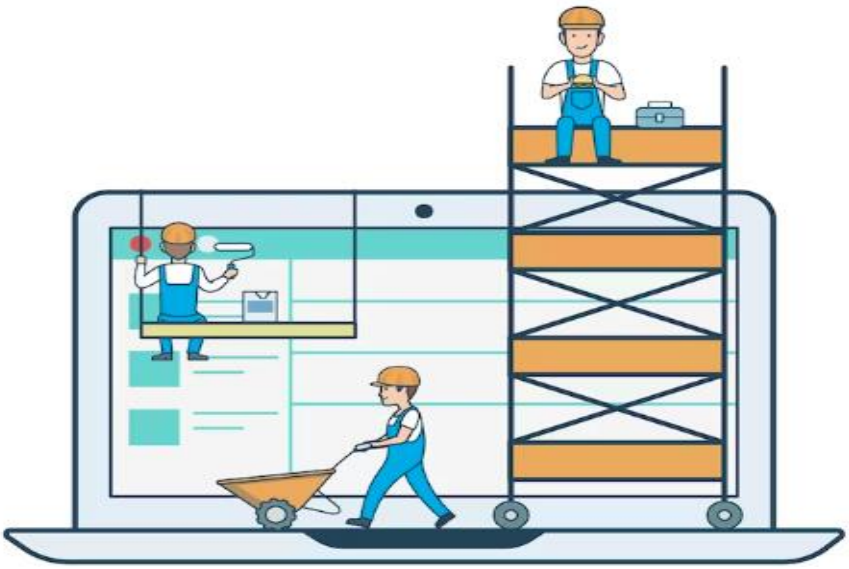
  It shows the impact of the software release on end users after successful deployment. Automating the entire process gives insights into how users react to the new build. It notifies the development teams of discovering critical issues and works towards fixing the bugs.

- **Continuous Operations**

  The objective is to eliminate planned downtime, resulting in minimum interruption to end users. Though expensive, it is worth the cost because of its advantages.

Let us now discuss how to create a DevOps Build Pipeline.

Companies build a DevOps Pipeline in various ways depending on their requirements. We shall discuss the common steps used in building a DevOps Pipeline.



(Image Source: Freepik.com)

- **Establishing CI/CD Tool**

  CI/CD tools vary according to the situation because each company has varying requirements. Selecting a suitable tool is crucial for customizing the pipeline for different applications. One of the commonly used tools is Jenkins.

- **Sourcing a Control Environment**

  Since companies working with large DevOps teams require dedicated space for storing and sharing codes, merging conflicts, and creating different software versions, sourcing a controlled environment becomes essential. GitLab and BitBucket are popular source control management tools.

- **Setting Up a Build Server**

  The next step involves setting up a build server, a stable, centralized, and reliable environment, for building distributed development projects. Build servers ensure continuous integration by retrieving integrated code from the repositories, acting as integration points for developers, and providing an environment to ensure the codes work. Popular build servers include Jenkins, Travis-CI, go.cd, and TeamCity.

- **Building Automation Tools for testing**

  Testing the code after its configuration is essential. Developers run automated tests, like unit, functional, integration, and regression tests, to ensure that error-free codes move down the pipeline to the development stage. TestComplete is an excellent option for running tests because the Jenkins plugin has features like storage for test history.

- **Deploy to production**

  The deployment stage is the final stage of the DevOps Build Pipeline. Using configuration tools like Jenkins allows you to run scripts for easy deployment. Ideally, a manual setup is better, but if you are confident, you can set up automatic deployment. It ensures that the script runs only if it passes all tests.

The CLI and Bash Commands 101

A computer is an excellent tool, but it can work only if you communicate with it in the language it understands. Since the standard user does not know computer language, there is a need for a user interface to serve as a medium between the user and the

computer. Therefore, CLI (Comman-Line Interface) is a text-based user interface that helps users run software programs, manage files, and interact with the computer.

Bash is the Unix CLI, the terminal, the shell, or the command line. It is a common language that helps users interact with the computer, manage files and use the machine more efficiently than a Graphical User Interface (GUI).

Here are some commonly used Bash Commands and their explanation that should help you understand how to interact with the machine.

For better understanding, we shall break the Bash commands into three categories.

a. **Basic User** - The Basics that one must know
b. **Intermediate User** - A comparatively higher level than the Basic User
c. **Advanced User** - Experts

The Basics	
Navigating File Systems	Almost all modern file systems have directories and sub-directories (directories within another). You can traverse backward from the child directory to the parent directory. For example, Windows has drives C:\, A:\, etc. Unix has a single root directory known as \
pwd / ls / cd	When you work on computers, you always work within a directory. ● pwd - Print the user's working directory. ● ls - List the contents of the directory. ● cd - Change to a different directory
; / && / &	Commands constitute whatever we type into the command line. They are used to execute machine code stored on the computer. ● ; - The semicolon - is used to run one command right after the other. ● && - This command says that the command to the right will not run if the command on the left fails. ● & - It is used when executing lengthy commands. This command & allows you to enter another command while the existing command is still running.
Getting Help	The next set of commands can help you use them appropriately.
-h or - - help	This command brings out the Help menu for the command.

man	Typing man before any command brings out the manual for the command.
Viewing and Editing Files	These commands help users view and edit files.
head / tail / cat / less	● head - Outputs the initial few lines of the file. If you wish to specify the line number, add -n to head. ● tail - Outputs the last few lines of a file. If you wish to specify the line number from where to view data, you can suffix -n to tail. ● cat - The cat command concentrates the specific list of files and sends them to the terminal. You can use this command with a single file or multiple files. ● less - This command allows you to view a specific file quickly.
nano / nedit	nano - is a simple command-line text editor for beginners. nedit - is a graphical editor and opens an X Window to allow point-and-click editing.
Other common editors	include emacs, vu, gedit, and vim.
Creating and Deleting Files and Directories	These commands can help create, modify, and delete files and directories.
touch	This command can modify file timestamps and quickly create an empty file.
mkdir / rm / rmdir	● mkdir - create new empty directories ● rm - removes files, but you must be careful because the files become non-recoverable. ● -i - Adding this command after rm gives you an "Are you sure" prompt. It is a safety measure to prevent the deletion of important files. ● rmdir - Removes empty directories
Moving/copying files, making links,	These commands help move, rename, and copy files. You can use them to create a hard/soft link to the file.
mv / cp / ln	● mv - moves or renames a file ● cp - copies a file ● ln - creates a hard link to the file ● ln -s - creates a soft link to the file
Directory trees, disk usage, and processes	These commands help create directory trees, determine disk usage and processes.
mkdir -p / tree	● mkdir -p - Adding -p to mkdir allows to make all directories in the path if they do not exist. ● tree - helps better visualization of a directory's structure. ● Adding - - prune to tree hides empty directories in the path.
df / du / ps	● df - shows the space occupied by files for the disks ● du - shows the space taken up by a directory ● ps - shows the user's currently-running processes

Miscellaneous commands	Here are some commonly used commands that can benefit beginners.
passwd / logout / exit	● passwd - change your current password, but it will ask you for the current password for verification. ● logout - exits specific shell users have logged into ● exit - exits all kinds of shells
clear / *	● clear - moves the current terminal line to the top. It helps clear your workspace. ● * - it helps you look for your files

Intermediate	
Disk, Memory, and Processor usage	As you gain mastery, you can use these commands at an intermediate level.
ncdu	ncdu - An improved version of the du command. It provides a navigable overview of file space usage.
top / htop	● top - displays currently running processes, their owners, and memory usage, etc. ● htop - an improved version of top where you can restrict the displayed processes to specific owners by name
REPLs and Software Versions	Read - Evaluate - Print - Loop - Used for particular programming languages For example, you can open Python REPL using the python command. The q command enables you to quit the operation.
Environment Variables and Aliases	● Environment variables are persistent variables created and used within the bash shell. ● You define them with an = sign and use them with a $ sign ● The command printenv allows you to see all currently-defined environment variables. ● Other environment variable commands include export and unset. ● Aliases are similar to environment variables but used in a different manner. It replaces long commands with short ones. An unalias command removes an alias.
Basic Scripting	● bash - These scripts end in .sh, and allow automating complicated processes. It can contain numerous normal shell commands.
Config Files / .bashrc	● bashrc - The ~/ .bashrc file is the primary configuration file for a bash shell. Commands made in the ~/ .bashrc file will run every time you log in.
Types of Shells	● If you have a username, log into your Login shell. ● Interactive shells accept commands. ● Bash is one kind of shell. Other commonly used shell

	commands include zsh, csh, and fish.
Finding Things,	These commands help search for files related to a specific command. ● whereis - This command returns the binary source code's location and page for the specific command. ● which - Returns the location of the binary alone. It is useful for finding the command's original version if hidden by an alias. ● whatis - Prints the one-line description of the command from its 'man' page. ● locate - helps find files from anywhere in the system ● find - Compared to locate, find is an inferior version as it was written for the initial version of Unix in 1971. In contrast, locate was added to GNU in 1994. However, find has more features than locate.
Downloading things	These commands help download content from the internet. ● ping - This command is used to check the strength of your internet connection. It attempts to open a communication line with your network host. ● wget - enables you to download files from the internet. ● curl - is similar to the wget command. The difference is that curl can send data, whereas wget can only receive data. While wget downloads files recursively, curl cannot. ● apt - Helps install, upgrade, and delete software on your system. ● gunzip - used for unzipping .tar.gz files ● tar - this command extracts a .tar file to a directory of files. ● gzip - This command can zip .tar files.
Redirecting Input and Output	● | - used for redirecting the first command's output to the input of the second command. ● > - redirects output from stdout to a specific location. ● < - gets input from specific locations rather than stdin. ● echo - writing text to stdout by default ● printf - an improved version of echo, as it allows formatting and escape sequences. ● 0 / 1 / 2 - Standard input, output, and error streams ● tee - send output to stdout and multiple locations.

Advanced - Expert users	
Superuser - the only person who can create users, install software, etc.,	● sudo - Run a command as another user with sudo -u username. However, you will need the user password. If -u is not provided, the superuser is the default user with unlimited permissions. It is also called as root. ● whoami - check your user name ● su - allows you to become another user temporarily. Type exit to switch back.
File Permissions	● ls -l - This command allows viewing file permissions.

	• chmod - Allows modifying file permissions by setting access bits. • chown - change users who own a file • chgrp - change the group who owns a file
User and Group Management	• users - shows all users who have logged in • /etc/passwd - see all users, even those who have not logged in. Please do not modify this file because it can corrupt user accounts. • useradd - add a user • userdel - delete a user • groups - show all groups where the current user is a member • /etc/group - see all groups in the system. Please do not modify this file. • groupadd - add a group • groupdel - delete a group • groupmod - change a group's name, password, or ID number
Text Processing	• uniq - print unique lines • sort - sort lines alphabetically or numerically • diff - report the lines that differ between two files • cmp - report the bytes that differ between two files • cut - good for CSV processing as it is used for cutting a line into sections • sed - replace one string with another in a file
Pattern Matching	• grep - find lines of a file matching some pattern • awk - pattern matching language built around reading/manipulating delimited data files • sed - Like awk, it is a Turing-complete language
Copying Files	• ssh - shows how Unix-based machines connect over a network. • scp - copy a file to another machine • rsync - a file-copying tool
Long Running Processes	• nohup - ensures the command will not hang if the shell is closed or the network fails. • yes - continually outputs until it is killed • ps - shows a list of a current user's processes • cron - an easy way of automating regular and scheduled tasks
Miscellaneous	• pushd - maintains a directory stack, adds the new directory to the left-hand side of the list • popd - return to the most recent directory • xdg-open - opens a file with a default app • xargs - vectorizes commands

Installing Course Dependencies

Let us see how to install Linux on Windows 10.

Installing Linux on Windows requires the following steps.

- Partition a hard drive in Windows 10
    a. Open the Windows Search Bar and find the search icon at the bottom corner.
    b. Type DISKMGMT.MSC and press Enter.
    c. Shrink the volume by right-clicking on the main hard drive.
    d. Set around 20GB for Linux.
    e. Click 'Shrink.'
- Make a Linux Bootable USB
    a. Download a Linux distro in ISO format using freely downloadable options like Fedora, Ubuntu, or Mint.
    b. Insert the USB into your computer and format it.
    c. Download Rufus.
    d. Open Rufus and select your USB drive.
    e. Click Select under Boot Selection and choose the downloaded ISO file. Please do not change any other settings.
    f. Click Start. Choose ISO if you get a pop-up message.
    g. Rufus will mount the ISO file on your drive. It will take some time.
- Install Linux from the USB
    a. Insert the Linux USB into your computer.
    b. Click Start.
    c. Hold the Shift Key while clicking Restart to navigate to the Windows Recovery Environment.
    d. Select Use a Device and choose your USB device on the next screen.
    e. The computer will boot Linux.
    f. Select Install Linux.
    g. You need to fill in the details asked for.
    h. Reboot when prompted.

1.6 Lab Assignment

Since we have discussed DevOps and other terminologies, let us test what we have learned. Here are ten multiple-choice questions (MCQs) on DevOps with four options. Provide the correct answer to each question.

- **What, according to you, is the primary objective of DevOps?**
    A) Marketing and Sales
    B) Integrating Development and Operations

C) Network Security
D) Software Development alone

Answer - B

- **What does the future of DevOps primarily revolve around?**
  A) Focusing on individual tasks
  B) Slowing down processes
  C) Disconnecting Development and Operations
  D) Increasing automation and collaboration

  Answer - D

- **Why do you feel DevOps is considered essential in the industry today?**
  A) It deals with hardware issues.
  B) It focuses on development alone.
  C) It accelerates the development process and enhances collaboration.
  D) None of the above.

  Answer - C

- **What is a Build Pipeline in DevOps?**
  A) Type of a programming language
  B) A marketing strategy
  C) A sequence of processing stages for software development
  D) A tool for network security

  Answer - C

- **Which language is commonly used for scripting in DevOps?**
  A) HTML
  B) C++
  C) COBOL
  D) Bash

  Answer - D

- **Which of the following options is NOT a benefit of DevOps?**
  A) Decreased Efficiency
  B) Enhanced Security
  C) Improved Collaboration
  D) Rapid Development

Answer - A

- **What is the primary function of CLI in DevOps?**
    A) For sending emails
    B) Handling Sales calls
    C) Playing video games
    D) Operating System Control and Scripting

    Answer - D

- **What is the full form of the acronym CLI in DevOps?**
    A) Customer Line Interface
    B) Continuous Language Interface
    C) Command Line Interface
    D) Constant Learning Interface

    Answer - C

- **Why are Bash commands essential in DevOps?**
    A) They assist in controlling and scripting the system.
    B) They help send emails.
    C) They help make phone calls.
    D) They are necessary for marketing.

    Answer - A

- **What do you think is the primary focus of DevOps in software development and delivery?**
    A) Only Sales
    B) Only Coding
    C) Only Testing
    D) A complete lifecycle from development to deployment

    Answer - D

1.7 Conclusion

We have discussed terminologies like DevOps, Linux, Bash, and CLI in this chapter. We trust the concepts have been explained clearly for everyone to understand them properly. All these concepts help simplify industrial development and processes while ensuring proper collaboration between different departments, such as development and operations. It shows that effective communication between an industry's various departments is essential to its overall development.

We will discuss more such topics in our forthcoming chapters. Please keep reading the book to enrich your knowledge.

1.8 References

a. Courtemanche, M., Mell, E., Gillis, A. S., & Riley, C. (2021, December 22). *What Is DevOps? The Ultimate Guide*. IT Operations; TechTarget. https://www.techtarget.com/searchitoperations/definition/DevOps
b. mijacobs. (n.d.). *What is DevOps?* Microsoft.Com. Retrieved August 11, 2023, from https://learn.microsoft.com/en-us/devops/what-is-devops
c. *How to Build a DevOps Pipeline: A Beginner's Guide*. (2021, March 2). PagerDuty. https://www.pagerduty.com/resources/learn/what-is-a-pipeline-in-devops-and-how-to-build/
d. Loshin, P., & Gillis, A. S. (2021, December 3). *command-line interface (CLI)*. SearchWindowsServer; TechTarget. https://www.techtarget.com/searchwindowsserver/definition/command-line-interface-CLI
e. *101 Bash Commands and Tips for Beginners to Experts*. (n.d.). Awwsmm.Com. Retrieved August 12, 2023, from https://www.awwsmm.com//blog/101-bash-commands-and-tips-for-beginners-to-experts
f. *How to Install Linux on Windows 10*. (2019, October 2). HelloTech How. https://www.hellotech.com/guide/for/how-to-install-linux-on-windows-10
g. Image source, Freepik.com

Chapter 2 - Understanding Git, GitHub, and BASH

Introduction

We have discussed the concept of DevOps, BASH, and CLI in the first chapter. Let us proceed further with more advanced concepts like Git and GitHub in this chapter. We shall also touch upon more BASH commands and understand the Version Control System concept. Let's plunge into the topic without wasting time and see what Version Control System is.

What is a Version Control System

Version Control System (VCS) is also known as source control. It involves tracking and managing the various changes carried out to software code. VCS constitute software tools that help software teams to understand the changes made to the source code over time. VCS tracks the modifications done in the source code and records the changes made to the files, making it an integral part of DevOps.

Let us now see why VCS is critical to DevOps.

The earlier chapter discussed that DevOps requires excellent coordination and collaboration between the development and operations teams. Development teams can be located at multiple locations, each contributing to developing specific features or functions. It requires constantly modifying the source code by adding or removing specific code components.

A VCS is handy because it enables the developer team to communicate and manage all the changes, they make to the source code from different locations. Each contributor contributing to the changes has a separate branch. Unless experts analyze them properly, the changes are not merged into the source code. Therefore, VCS helps organize the source code and improve productivity by smoothening the development process.

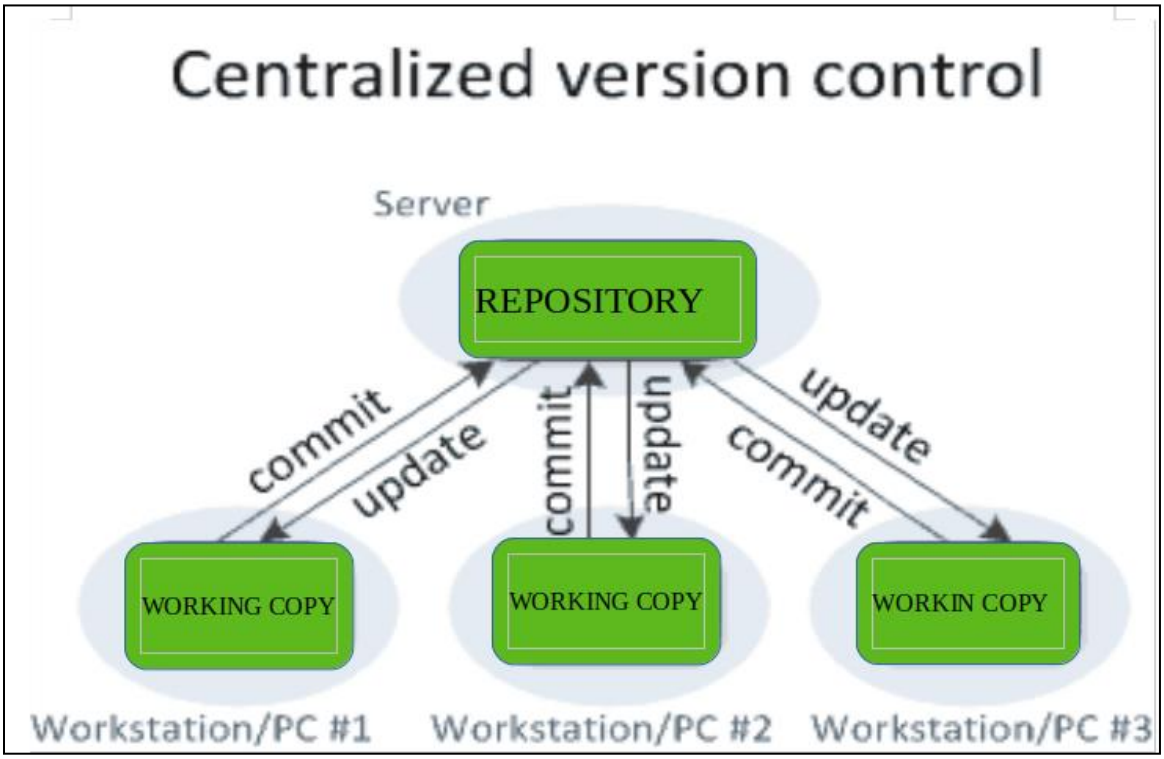
A VCS tracks the modifications and maintains each record meticulously by taking snapshots. It helps development teams who add new functions because it allows them to return to the original position, review the changes, and continue with the older version if the updated version fails. Alternatively, it enables them to fix their mistakes while minimizing the disruption to other team members.

Most software development teams consider the VCS as a crucial repository of invaluable knowledge because it maintains detailed records of the problem domain that developers collect and refine through careful effort. Hence, VCS protects the source code from catastrophic and casual degradation of human error and unintended consequences.

Different Types of Version Control Systems - VCS Terminologies

Depending on their functionality, VCSs are classified into three primary types.

a. ***Local Version Control Systems (LVCS)*** is the most straightforward format with a database to manage all file changes. Revision Control System (RCS), the earliest VCS implementation, is a commonly used VCS tool. It maintains the patch sets in a unique disk format. Software teams can add all the patches and recreate how a particular file was at any time.
b. ***Centralized Version Control Systems (CVCS)*** have a single global repository that every user must commit to reflect its changes. Updating the repository enables users to view their changes. The principle is that 'You commit, and They update.'



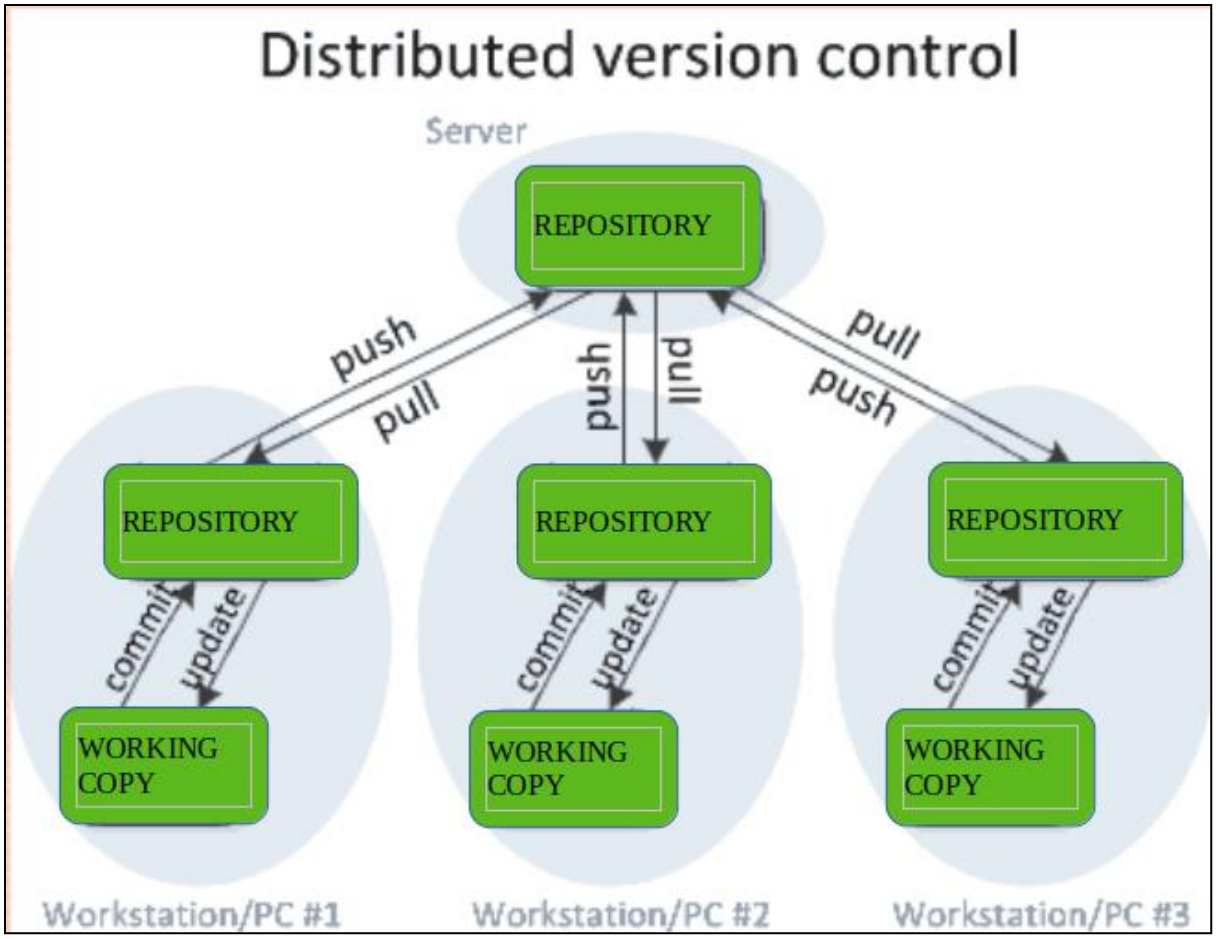
Source - CVCS

The most significant benefit of CVCS is that it allows better collaboration between developers and provides insight into what others are doing on the project. Hence, it enables better control. At the same time, its drawback is that if it breaks down during this period, it becomes impossible to collaborate and save the versioned changes. Therefore, developers lose everything if the database becomes corrupted and proper backups are not maintained.

This drawback led to the development of the Distributed Version Control System (DVCS).

c. ***Distributed Version Control System (DVCS)*** has multiple repositories instead of a single one, as seen in CVCS. Thus, each user has their individual repository and working copy. It implies that others will not have access to the changes a particular user makes. The changes will reflect in their local repository, and the user must push them through to be visible on the central repository. Similarly, the user cannot view the changes others make. They must pull those changes from the central to their local repository.

   Therefore, the principle is, 'You commit, you push, they pull, and they update.' The most glaring examples of DVCS are Mercurial and Git. The most significant benefit is overcoming the problem of a single point of failure.



Source - DVCS

Purpose of Version Control System

The question arises as to why a VCS is required. Let us discuss the purpose of having a VCS.

- ***Multiple users*** - A VCS allows multiple users to work simultaneously on any project. It enables each user to work on and edit their file copies. They can decide when to share their changes with other team members.
- ***Multiple machines*** - A single user can work simultaneously on multiple computers. Hence, a VCS is an excellent tool for users who work in isolation.
- ***Integration*** - A VCS enables convenient integration of work performed simultaneously by multiple team members. However, manual intervention becomes necessary when two users make conflicting edits on the same file line.

- ***Access to historical versions*** - VCS allows users to access historical versions of their projects. It insures against data loss due to computer crashes. It also enables users to roll back to a previous version and undo their mistakes without losing the work. VCS lets users know who, when, and why any person edited a part of a file.

Uses of Version Control System

A VCS has the following uses.

- ***As a repository*** - Since the VCS contains all the changes and the historical snapshots of a project, it is helpful as a database of changes.
- ***As a copy of work*** - VCS allows users to edit their copies without affecting the work of others. Finally, they can commit the changes to the repository. Hence, it has utility as a personal copy of all files in a project.
- ***Working in a group*** - VCS allows users to work in a group without changing the main code because any change can inconvenience other users. VCS lets you adapt to other users' changes, merge different requests, and test the functionalities before putting it live.
- ***As a project management tool*** - VCS lets developers plan projects, track and manage them to achieve their objectives by collaborating effectively with others and accelerating the projects to meet the specified constraints. It is one of the best project management and tracking tools.

Advantages of VCS

While each VCS can have different advantages, let us discuss the general benefits of a VCS.

- ***Data Integrity*** - VCS gives the entire software development team a single source of truth. It ensures the teams work with the same data set and access it from a common location.
- ***Improves visibility*** - VCS allows users to view every change made to the file, who made it, and when. Therefore, it is excellent for GRC (governance, risk, and compliance) or regulatory needs. It allows other users to see who is working on the specific file. Besides, users can revert to previous file versions to fix errors.
- ***Enables concurrent development*** - Multiple developers can work simultaneously on the same files without worrying about overwriting other team

members' work or duplicating their efforts. Hence, a good VCS enables concurrent development.

- ***Better team collaboration*** - Since VCS offers complete visibility to data, it enables better team collaboration as others know the changes you make and vice versa. It also enables team leaders to monitor and control the changes efficiently.
- ***Supports automation*** - VCS allows teams to achieve their goals by automating tasks like testing and deployment. Today, CI with automated builds and code reviews has become an SOP in software development.

Role of VCS in Build Pipelines

VCS is the link between development and deployment. Hence, it is a critical component with a significant role in CI/CD pipelines. VCS is where the CI begins in a CI/CD pipeline. It creates the source of the record and organizes it for the build.

VCS allows multiple users to view the files in a repository and edit the source codes. It keeps track of each change made to the code, like who made the change and when. Thus, software development teams can trigger automated build and test processes to provide immediate feedback on the changes.

Issues can arise during development. A VCS lets users go back into history and identify the commits that caused the problem. They can use the revision history to know what changed exactly, who did it, and when. Therefore, it is the first step towards implementing continuous integration and building a CI/CD pipeline.

CVCS vs. DVCS

We have seen the CVCS and DVCS concepts earlier in this chapter. Let us understand the difference between the two VCS terminologies.
This table should simplify things.

Parameters	CVCS	DVCS
Working	The user gets the local copy of the source code from the	Every user has a local branch and a complete history on it.

	server edits them and commits the changes to the central source on the server.	The client pushes the changes to the branch, which are then pushed to the server repository.
Branches	The developer can face merge conflicts while working on branches.	Developers find it easy to work on branches because of fewer merge conflicts.
Learning Curve	Compared to DVCS, CVC systems are easier to learn.	DVCS can be challenging to beginners because they need to remember multiple commands.
Speed	Compared to DVCS, CVCS is slower because every command must communicate with the server.	Working on DVCS is easier because the users deal with the local copy without communicating with the server every time.
Offline Access	CVCS does not provide offline access to users.	Things are simpler in a DVCS because users can download the entire repository on their local machines and work offline.
Backup	Developers cannot work if the CVCS servers are down.	Developers can work on local copies even if the DVCS servers are down.

Project Management and Tracking Tools

Let us discuss some popular project management and tracking tools.

Git

Git was created in 2005 by Linus Torvalds and, since then, maintained by Junio Hamano. It is a popular version control system used for the following.

a. Tracking code changes
b. Tracking users who made the changes.
c. Coding collaboration

Git is a DVCS that helps users.

a. Manage multiple projects with repositories.
b. Clone projects to allow users to work on a local copy.
c. Use Staging and Committing to control and track changes.
d. Allow users to work on different versions and parts of a specific project using the Branch and Merge features.
e. Pull the latest versions of the project to a local copy.
f. Push local updates to the main project.

Source : https://www.devopsschool.com/

How do you work with Git?

a. Initialize Git on a folder to make it a repository.
b. Git creates a hidden folder and keeps track of the changes in that folder.
c. Any change to a file modifies it.
d. If you wish to stage a file, you must select the modified files.
e. Since the staged files are committed, Git stores a permanent snapshot in its records.
f. Git allows users to view the entire history of every commit.
g. It allows users to revert to a previous commit.
h. While Git tracks the changes made in a commit, it does not store a separate copy of each file.

Why is Git popular?

a. Git is an effective tool because over 70% of developers use it.
b. It lets developers work together from anywhere.
c. Git allows developers to view the complete history of the project they work on.
d. It enables developers to access and revert to earlier project versions.

How is Git different from GitHub?

GitHub is not the same as Git, but it makes the tools that use Git. Microsoft owns GitHub since 2018 and is the largest host of source code globally.

Let us now discuss how to install Git and get started.

How to install Git

Git is available free at https://www.git-scm.com/. Users can download the application and install it on their systems.

a. Open up the Command Shell.
b. Windows users can use Git bash because it is included in Git for Windows.
c. Mac and Linux users must use the built-in terminal.
d. Check whether you have installed Git correctly. It should show as 'git version X.Y.
e. Configure Git and change the username and email address according to your preferences. It is necessary for registering to GitHub later on.
f. Create a Git folder using BASH commands ***mkdir*** and ***cd***.
g. On navigating to the correct folder, initiate Git using the 'init' command.

   Your first Git repository is ready.

Add files to Git.

You can add or create new files using text editors like HTML and save them to the new folder. Use the ***ls*** command to list the files. Check the Git status to see if it is part of the repository.

Git has not added the file to the repository but is aware of it. The files can be in the tracked or untracked state. Initially, the files are in an untracked state, but you must add them to the staging environment for tracking.

Git Staging Environment

The concepts of Staging Environment and Commit are the core functions of Git. Adding files to the staging environment is essential before committing them to the repository. You can add multiple files to the Git Staging Environment. Use the command ***git add -A***.

Git Commit

Once you have moved the files to the Git Staging Environment, you can move to Commit. It helps track your changes as you work. Please note to include a message whenever you commit files to Git. It simplifies things for other users because they can see what has changed and when.

Use the ***commit*** command to perform a commit. Adding ***-m*** to ***commit*** allows you to add a message.

However, you can commit small changes to the files without staging them. The ***-a*** command automatically stages every changed and tracked file.

The ***log*** command allows users to view the history of commits for a repository.

Git Help

You can use Git help if you forget the commands or options. For example, git command ***-help*** lets you see the available options for a specific command. Similarly, ***git help - -all*** lets you see all the commands.

GitHub

a. Sign up for a GitHub account at GitHub.
b. On signing up, you must create a new repository on GitHub. You can choose to keep it Private or Public. Click on 'Create Repository.'
c. Push the local Git Repository to GitHub.
d. Besides being a host for Git content, GitHub is a good code editor.

GitHub Commands

a. ***pull*** - The pull command allows users to pull the recent changes made to the file to your local copy.
b. ***fetch*** - The fetch command allows you to get the entire change history of a tracked repository.
c. ***merge*** - The merge command combines the current working branch with a specific branch or the master.
d. The pull command is a combination of fetch and merge.

e. ***push*** - Similarly, the push command allows users to push changes to GitHub.

   You can create a new branch on GitHub and pull/push changes.

GitHub Flow

The GitHub flow works well with Git and GitHub. It simplifies things for teams to experiment freely and make deployments. Here is how it works.

a. Create a new branch on GitHub.
b. Make the changes and add your comments.
c. Open a Pull request and review it.
d. Deploy
e. Merge

Hosting Pages on GitHub

a. Create a new repository.
b. Push your local repository to GitHub Pages.
c. Now, you check out your GitHub Page.

We have learned the basics of Git and GitHub. Are you game for a small MCQ exercise on Git and GitHub?

Chapter Questions:

1. What is Git?
   a. A version control system
   b. A text editor
   c. A hosting service for Git repositories
   d. A programming language

2. What is GitHub?
   a. A programming language
   b. A version control system
   c. A hosting service for Git repositories
   d. A text editor

3. What is the purpose of Git?
   a. To host Git repositories

b. To write programs in a specific language
c. To edit text files
d. To track changes made to files over time.

4. What is the purpose of using GitHub?
   a. To edit text files
   b. To host Git repositories
   c. To track changes made to files over time.
   d. To write programs in a specific language

5. Is it possible to use Git for non-source code files?
   a. Yes,
   b. No
   c. It depends on the file type
   d. It depends on the OS

6. How do you install Git?
   a. By purchasing it online
   b. By downloading from the official website.
   c. By requesting your ISP
   d. By writing a program to download it automatically.

7. How do you configure Git?
   a. By setting your username and email address using an online form
   b. By setting your username and email address using a programming language.
   c. By setting your username and email address using the command line.
   d. By setting your username and email address using a text editor.

8. What is a repository in Git?
   a. A single file associated with a project with its revision history.
   b. A single file associated with a project without any history of revisions.
   c. A collection of files and folders associated with a project, including each file's revision history.
   d. A collection of files and folders associated with a project without any history of revisions.

9. What is the difference between Git and GitHub?
   a. Git and GitHub are the same.
   b. Git and GitHub are different tools used for different purposes.
   c. Git is the tool, and GitHub is the service for projects that use Git.
   d. GitHub is the tool, and Git is the service for projects that use GitHub.

10. What does GitHub offer?
    a. GitHub offers tools to ship better code through command line features, threaded discussions, pull requests, and code reviews.

b. GitHub offers tools to ship better code through threaded discussions only.
c. GitHub offers tools to ship better code through command-line features alone.
d. GitHub offers tools to ship better code through pull requests only.

Chapter Questions Answers:

1. A
2. C
3. D
4. B
5. A
6. B
7. C
8. C
9. C
10. A

Conclusion

We have discussed version control systems, their uses, purpose, and significance in CI/CD build pipelines. We have also seen the different types of version control systems and their differences. This chapter explained the concepts of Git and GitHub in detail. That should set the ground for sterner tests, like Version Control for DevOps.

References

1. Atlassian. (n.d.). *What is version control*? Atlassian. Retrieved August 19, 2023, from https://www.atlassian.com/git/tutorials/what-is-version-control
2. S. (2019, July 11). *Version Control Systems*. GeeksforGeeks. https://www.geeksforgeeks.org/version-control-systems/
3. *Introduction to Git and GitHub*. (n.d.). W3schools.Com. Retrieved August 19, 2023, from https://www.w3schools.com/git/git_intro.asp?remote=github

4. *What Is Version Control and How Does It Work?* (n.d.). Perforce Software. Retrieved August 20, 2023, from https://www.perforce.com/blog/vcs/what-is-version-control
5. *What is Version Control?* (n.d.). JetBrains. Retrieved August 20, 2023, from https://www.jetbrains.com/teamcity/ci-cd-guide/concepts/version-control/
6. *Difference between Centralized Version Control and Distributed Version Control*. (n.d.). Tutorialspoint.Com. Retrieved August 20, 2023, from https://www.tutorialspoint.com/difference-between-centralized-version-control-and-distributed-version-control#

Chapter 3 Version Control for DevOps

Introduction

Here is a brief recap before understanding Version Control for DevOps. We have discussed DevOps and Version Control Systems in the first two chapters. DevOps has simplified business processes by ensuring proper collaboration between software development and operations teams. Version Control Systems enable multiple software development teams to carry out changes simultaneously and revert to earlier versions of the applications to ensure they work correctly.

We have seen the concept of Build Pipelines and understood BASH commands. At the same time, we have gone through Git and understood the difference between Git and GitHub. We explore the topic further and understand how version control systems benefit DevOps.

This chapter discusses Version Control for DevOps. So, understanding the different application architectures becomes essential. Let us see what an application architecture is and what are the various types of application architectures.

Before understanding the various application architectures, let us clarify what an application architecture is.

What is an application architecture?

An application architecture is akin to a structural map providing guidelines for assembling software applications. It defines how applications interact with each other to satisfy client requirements. An application architecture comprises software modules, systems, components, and their various interactions.

Front-end application architecture focuses on user experience, whereas back-end application architecture constitutes the systems that power the application. Software

developers use application architecture to ensure applications remain functional, reliable, and scalable. They are critical because they help organizations implement software development to deliver valuable products while saving time and money.

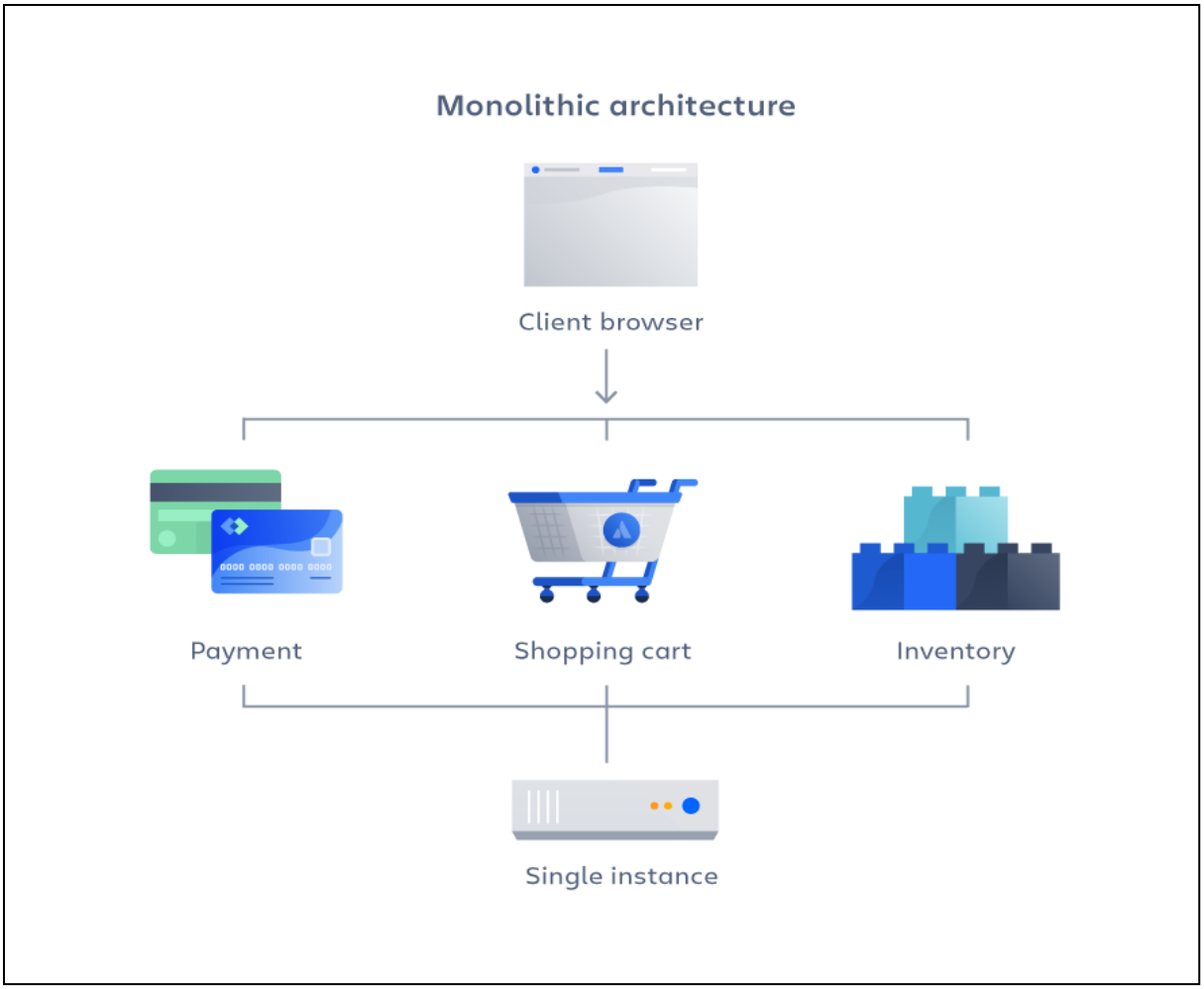
Different Types of Application Architecture

Application Architecture can be of various types, depending on the service relationships. The most common types of application architecture include monoliths, microservices, N-tier, event-driven, and service-oriented architecture.

- Monolith Architecture

  The name monolith signifies a single application stack architecture containing all the functionality. It is tightly coupled in how they are developed and delivered and in the interaction between the services.

  Since it comprises a single application stack, updating or scaling one aspect of the monolith application impacts the entire application and its underlying infrastructure. Therefore, the whole application must be released again, even if there is a slight change to the code. It implies that there will be fewer updates and releases.



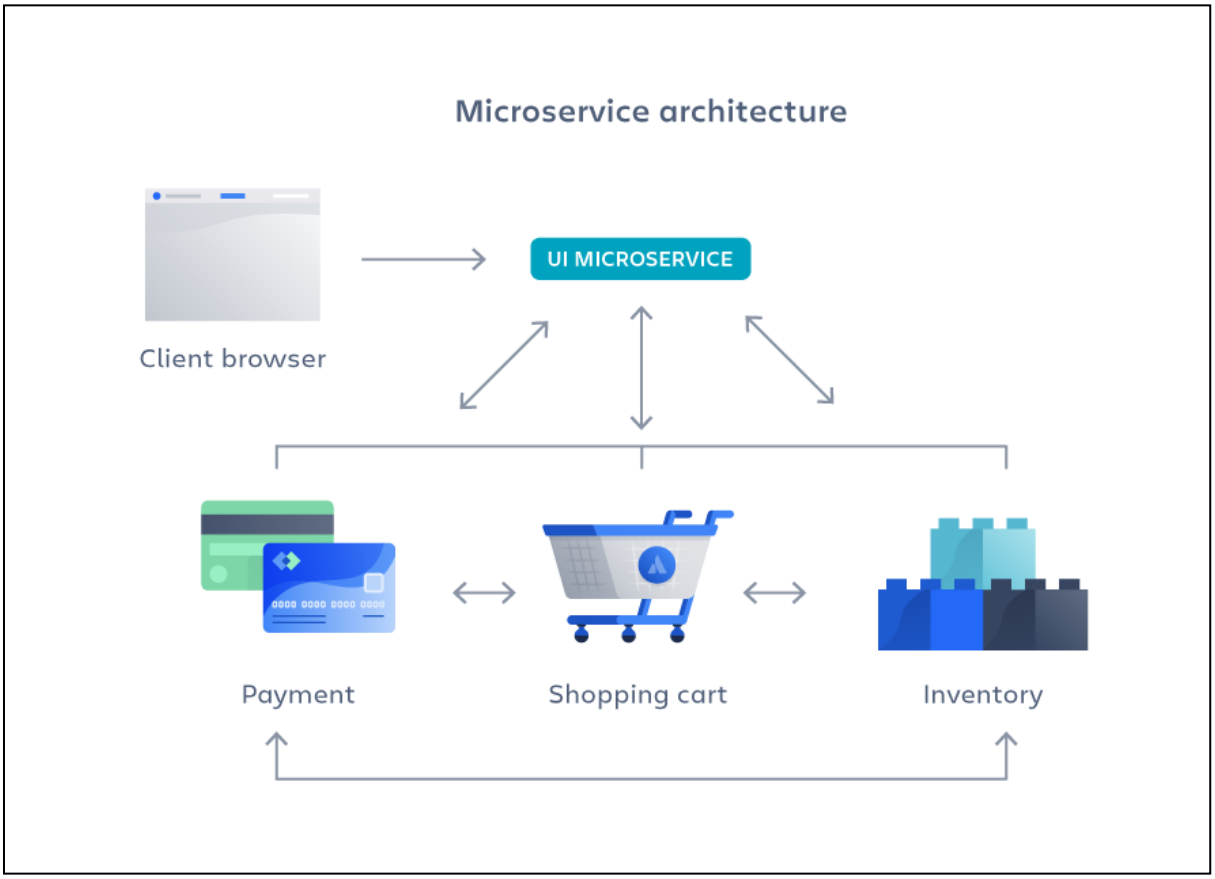
Source - [Atlassian]

- Microservices Architecture

As the name suggests, microservices comprise several processes or components as the applications are broken down into small components. Each component acts as a microservice.

Since the microservices are loosely coupled and distributed, a slight change in the application code in one microservice does not impact the other microservices. The most significant advantage is that it allows updating or scaling individual services without requiring complicated infrastructure.

The fault tolerance levels of a microservices architecture are high, and it offers dynamic stability. Developing multiple microservices concurrently is possible. Since they are deployed individually, any changes to the app do not require its rebuilding or redeploying. Therefore, multiple developers can work simultaneously. Hence, it saves time and makes releasing new features with higher frequency possible.



Source - Atlassian

- N-tier Architecture

  N-tier or layered architecture is generally used for building enterprise and on-premise apps. Usually, you associate legacy apps with this traditional architecture. A layered architecture can have multiple (n numbers) layers. Most often, there are three layers, each with its individual responsibility.

The layers are arranged horizontally, one over the other, in an N-tier architecture. These layers help perform logical functions and manage dependencies. The unique feature of a layered architecture is that any layer can call into the layer immediately below or other layers below it but not into the layers placed above it.

- Event-driven Architecture

  Traditionally, application architectures are request-driven models. Sometimes, they can work depending on the happening of a specific event. Under such circumstances, they are known as event-driven architecture. An event can be a significant change in the system software or hardware, with the source being internal or external inputs.

  The core of an event-driven architecture is the capture, processing, communication, and persistence of events. The benefit is that it requires minimal coupling, making it ideal for modern, distributed application architectures.

- Service-oriented Architecture:

  Service-oriented architecture is an advanced, well-established software design like microservices architecture. It structures applications into discrete, reusable services that use an enterprise service bus (ESB) for communication.

  This architecture comprises individual architecture, each organized around specific business processes. These services adhere to a communication protocol such as Apache Thrift, ActiveMQ, or SOAP. Together, these services integrate through an ESB and are used by front-end applications to provide value to the customer.

Monolith Vs. Microservice Architecture

So, we have seen various types of architecture. Shall we now understand the difference between monolith and microservices architecture?

Here are the key differences between monolith and microservices architecture from the development team angle in DevOps.

Parameters	Monolith Architecture	Microservices Architecture
Structure	It comprises a client-side UI, a database, and a server-side application. They are built on a single code base.	It is a distributed architecture, with each component accomplishing a single feature. They communicate with an API instead of exchanging data within the same code base.
Development process	It does not require much upfront planning. Adding code modules is easy, but the application can become complex and challenging to update over a period.	It requires thorough planning before implementation. Since there is better coordination, maintenance becomes easy. Making changes and finding bugs is quicker. Using code becomes easier with time.
Deployment	The deployment of	Since each microservice

	monolithic architecture is easier because developers install the entire code base and its dependencies in a single environment.	is an independent software package, deploying a multiservice architecture is more challenging. Developers use containers for packaging the code and dependencies to make each platform independent.
Modifications	Since monolithic architecture involves tight coupling of codes, a slight change in one component adversely impacts other software functions. The system requires retesting and redeployment after carrying out the modifications.	Microservice architecture is more flexible because developers change specific functions and not all the services. Hence, it does not affect the system's continuity.
Scaling	Since monolithic architecture has a single code base, the application requires scaling as the requirements change. It can result in resource wastage.	The microservice architecture supports distributed systems. Therefore, independent scaling is possible based on the predicted demands.
Debugging	Debugging monolithic architecture is easy	Debugging microservice architecture is complex

	because the developer traces data movement or examines code behavior within the same environment.	because it comprises multiple loosely packed individual services. Several developers could be responsible for different microservices. It can take more time because it involves coordinating with other team members.

Here are the primary differences between monolithic and microservices architecture from the operations team angle in DevOps.

Parameters	**Monolithic Architecture**	**Microservices Architecture**
Innovation	It does not allow rebuilding certain parts of the code base with new technology. Thus, it limits the organization's ability to innovate and introduce new business capabilities.	Developers can rebuild the independent software components with new software technologies. Hence, innovation becomes easier because of the loose coupling in microservices architecture.
Risks	Monolithic architecture is more vulnerable to bugs because updating software is challenging.	Microservice architecture is comparatively safer because the other applications remain

	Besides, a monolithic architecture experiences higher chances of failure because a minor error in the code base can cause the entire application to fail.	operational if one microservice fails. Developers find it more convenient to identify bugs and mitigate risks without impacting other microservices.
Time to market	Monolithic architecture is more rigid. Hence, the software development effort increases exponentially as the code becomes more complex. Developers need more time to manage cross-reference files and build new features.	Microservice architecture offers more flexibility because it allows developers to focus on smaller chunks of code. They need not understand how other microservices work when creating a specific microservice. They use APIs that are quicker to learn.
Total cost of ownership	Scaling monolithic applications requires upgrading memory and processing power for the entire application, making it more expensive. Maintaining a monolithic architecture is expensive.	Scaling microservice applications are easy because they allow horizontal scaling by adding resources on demand. Since microservices architecture runs independently of specific hardware, the

		maintenance costs are fewer.

What is 3-tier architecture?

A 3-tier architecture is an established software application architecture using three logical and physical computing tiers for organizing applications. The three tiers are a) the presentation tier or the user interface, b) the application or data processing tier, and c) the data tier where the data is stored and managed.

The advantage of 3-tier architecture is that each tier has its own infrastructure and can be developed simultaneously by multiple development teams. Besides, updating or scaling one tier does not impact the other two.

The 3-tier architecture was the prevailing architecture for client-server applications for decades. However, today, they serve as targets for modernization, using cloud-native technologies like microservices and containers, and for migration to the cloud.

Tier Vs. Layer

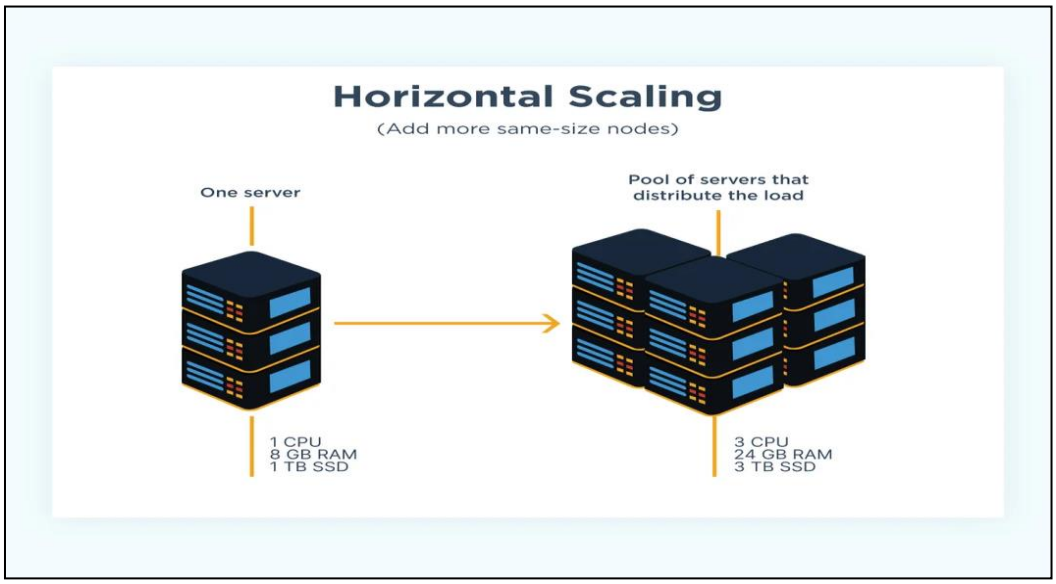
While layer and tier are used interchangeably, they have different concepts. A layer is a functional software division, whereas a tier constitutes the functional division of software running on infrastructure separate from other divisions. For example, the contacts app on your smartphone is a single-tier application with three layers. Understanding the difference between layer and tier is crucial because the tier offers more benefits than layers.

Vertical & Horizontal Scaling

Technologies keep developing with time, and systems must adapt to changing environments. Therefore, systems need to grow, and this ability is known as scaling. Scaling represents a system's elasticity, and it is possible to scale up, down, or out, depending on the circumstances.

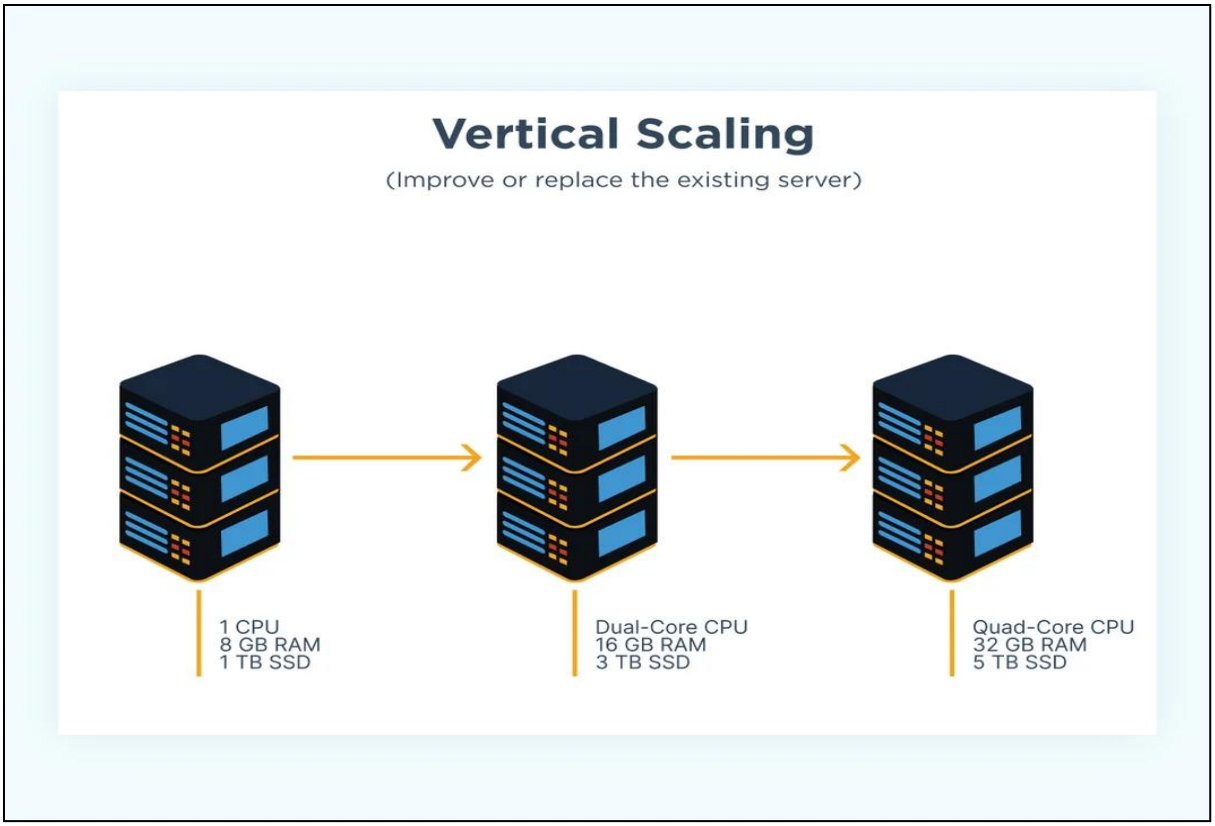
Scaling can be horizontal or vertical. Let us understand the concepts better before discussing the differences.

- Horizontal Scaling involves adding more nodes to your system/infrastructure to cope with the additional demands. It is also known as scaling out. For example, if a server does not have the capacity to handle additional traffic, adding more servers is the ideal solution.



Source - CloudZero

- Vertical Scaling is also referred to as scaling up. It involves adding more resources to a system to help meet additional demands. For example, upgrading the CPU's processing power adds more power to the machine.



Source - CloudZero

Difference Between Horizontal and Vertical Scaling

Parameters	Horizontal Scaling	Vertical Scaling
Description	It involves increasing or decreasing the number of nodes in a system/cluster to handle an increase or decrease in the workload.	It involves enhancing or reducing the system's power to handle an increased or reduced workload.
Example	Adding or reducing VMs in a cluster of VMs	Adding or reducing CPU RAM of an existing VM

Execution Type	Scale-In or Scale-Out	Scale up or Scale down
Distribution of workload	The workload is distributed over several nodes.	The entire workload is distributed over a single node.
Concurrency	It distributes the work across multiple machines to reduce individual workloads.	It depends on multithreading the existing machine to handle multiple requests simultaneously.
Required Architecture	It requires a distributed architecture.	It is possible in any architecture.
Implementation Time	It is more time-consuming and requires more expertise.	It requires less effort, expertise, and time.
Maintenance and Complexity	It is highly complex and requires more maintenance.	It is less complicated and requires less maintenance.
Load Balancing	It requires active distribution of the workload across multiple nodes.	Load balancing is not required because a single machine handles the workload.
Configuration	It requires modifying sequential pieces of logic to distribute the workload across multiple machines.	Since the same code can run on the machine with higher specs, there is no need to change the code.
Downtime	It does not need any	It requires downtime.

	downtime.	
Resiliency	The failure resilience is low because other machines in the cluster provide the necessary backup.	The failure resilience is high because of the single source of failure.
Networking	It requires quick inter-machine communication.	Inter-machine communication is slower.
Performance	It ensures higher performance because of the increased number of endpoints.	It results in lower performance because there is a limit to how much you can upgrade a machine.
Limitations	You can add as many machines as you can.	There are limits to how much a single machine can handle.
Costs	The costs are high initially but optimized with time.	The costs are low initially, but it is less cost-effective.

Introduction to Various Text Editors

Writing code requires using text editors. Let us introduce the five best text editors in use in 2023. These text editors allow the crafting of beautiful code. Besides, some of them are available for free.

- Sublime Text - An excellent feature-rich text editor

Its sleek interface and features like split editing and distraction-free writing mode are some of the best reasons to use Sublime Text.

This text editor offers an exciting range of keyboard shortcuts allowing you to open files, hide sidebars, duplicate lines, move to a specific line number, and open the spell checker.

Though it is not available for free, the exciting features make it worth buying.

- Visual Studio Code - Microsoft

  Microsoft Visual Studio Code is popular because it is highly customizable, has source control tools, and offers many integrations.

  Visual Studio Code has its own terminal and debugger. Besides, it supports linting and integrates well with all source control tools. It is one of the best IDEs for Python developers.

  This text editor offers multiple packages and free extensions.

- Espresso - A productive text editor

  Espresso is a good text editor because it offers frag-and-drop content facilities and a well-designed interface. However, it has drawbacks as it does not have Windows or Linux versions.

  However, Mac users will enjoy Espresso because it is powerful and smooth. It is ideally built for real-time editing because it allows you to see the various sections of code you work on. It has three columns comprising your navigator, code editor, and files.

- Brackets - Best for web designers and front-end developers.

  Brackets is a free text editor offering exciting features like Adobe Photoshop integration and real-time visualization. Since it is an open-source editor, it qualifies as one of the best for designing websites.

  Brackets, developed by Adobe, offers real-time visualization of the website you are working on as it displays the changes reflected in real-time. Hence, front-end developers relish using Brackets.

- Notepad++ - Full-featured and fast

  Notepad++ is popular because of its autosave feature and tabbed document interface. However, it has limitations because it is available only on Windows OS.

  Notepad++ is a free, open-source text editor allowing anyone to contribute content. Programmers love Notepad++ because it lets them see code quickly and paste one-off snippets from FTP clients without loading their IDE.

  The autosave function lets you save your files temporarily before you allocate a specific location for saving them.

We discussed popular text editors. We move on to the final part of this chapter to discuss the concept of the tech stack and its significance.

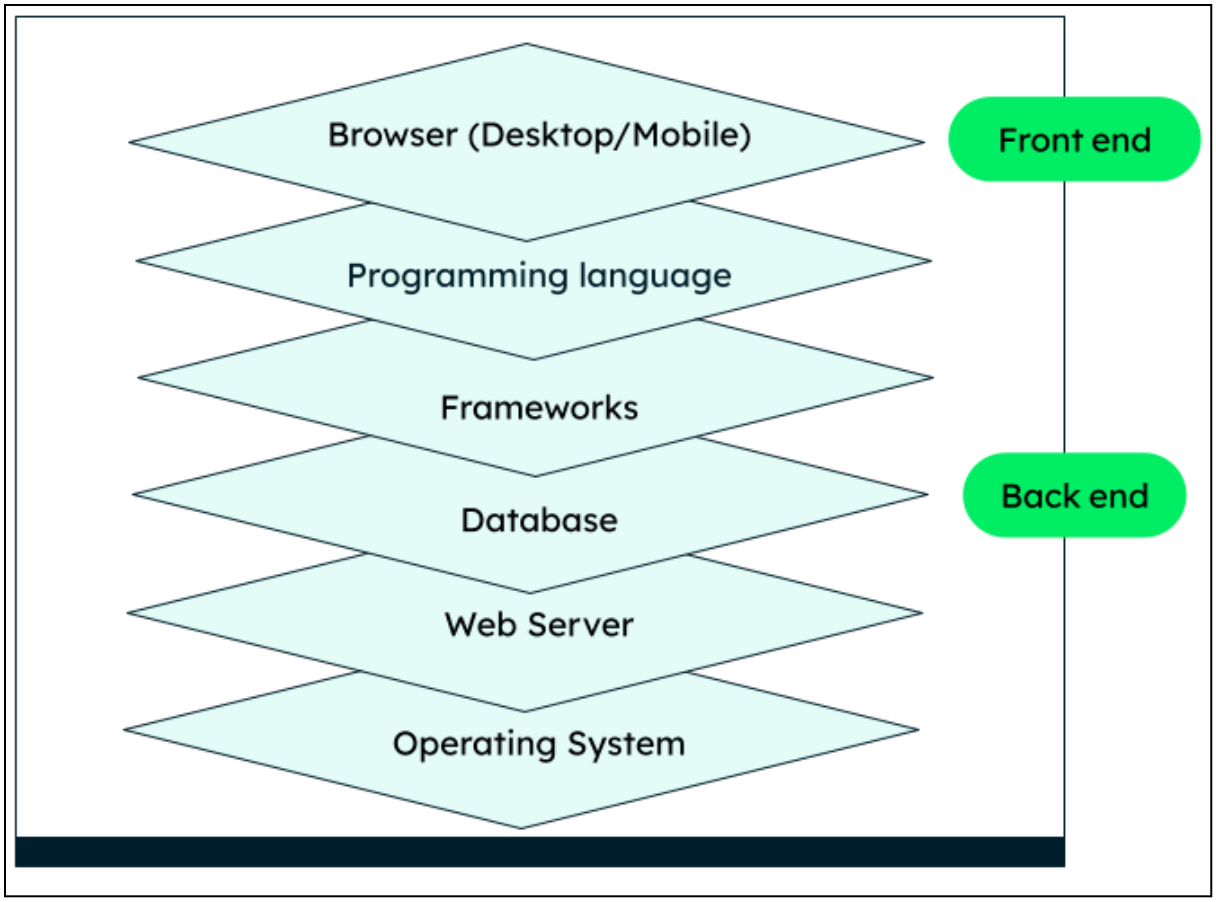
Tech Stack Behind Reliable Applications

Let us begin by learning what a tech stack is before discussing the tech stack behind reliable applications.

A tech stack is a set of technologies used for developing applications. It includes programming languages, databases, frameworks, APIs, and front-end and back-end tools.

Here are the main components of a tech stack.

- Servers - Web servers are critical components of a tech stack as they accept requests (HTTP) from clients (web browsers), pass them to the database seeking information, and handle the response accordingly. Examples of web servers include Nginx and Apache https.
- Databases - Databases like MySQL and MongoDB are essential components of a tech stack because they serve as data repositories where you store information. Modern-day platforms enhance database functionalities by providing features like querying, aggregation, and transformation.
- Frameworks - They provide structure to the application. Since they have common utilities, developers do not have to write logic from scratch. Popular web frameworks include Django, Rails, Spring, Laravel, Express, and .NET.
- Runtime Environment - This software runs the application. Runtime provides cross-platform compatibility. Some examples of runtime environments are JRE, CLR, Node.js, etc.
- Programming Languages - Some top programming languages include Java, Python, C#, PHP, JavaScript, Scala, Ruby, and TypeScript. They take care of the critical business logic your applications need.
- User Interface - Depending on the project's styling requirements, users can use markups like HTML, CSS stylesheets or frameworks like Tailwind or Bootstrap.



Source - Mongodb.com

Benefits of a Tech Stack

- It streamlines the development process and improves developer efficiency and productivity. Besides, it saves the time and effort of setting up and configuring the different application components.
- It allows developers to concentrate on perfecting code and building innovative features instead of dealing with integration issues.
- It ensures consistency across the project and offers a standardized approach to development. In addition, it offers guidance and provides the best practices for coding, architecture, and deployment.
- It improves the overall software quality and promotes code reuse and maintainability.

- Tech stacks are scalable. Hence, they can handle increased traffic, user interactions, and massive data volumes without disruptions or significant architectural modifications.
- Its seamless working allows developers to leverage optimizations specific to the chosen technologies.
- It reduces the risk of handling significant technical issues and security vulnerabilities.
- It reduces licensing costs and encourages teams to build feature-rich solutions without additional expenses.

Chapter Questions:

We have discussed Version Control and its connection with DevOps. Let us now test what you have understood from this chapter. Here are some multiple-choice questions connected with this topic.

1. What is horizontal scaling?
    a. Adding more power to existing systems
    b. Adding more nodes or instances to the system
    c. Moving to a new machine with more power
    d. Upgrading the capacity of a single machine

2. Which of the following databases can be easily scaled horizontally?
    a. MySQL
    b. Cassandra
    c. Google Cloud Spanner
    d. Amazon RDS
3. What is the main advantage of horizontal scaling?
    a. It is easy to implement.
    b. It is harder to upgrade and may involve downtime.
    c. It is easier to upgrade.
    d. It can help distribute the load and improve the system's overall performance.

4. What is vertical scaling?
    a. Adding more power to existing systems
    b. Moving to a new machine with more power
    c. Upgrading the capacity of a single machine
    d. Adding more nodes or instances to the system

5. Which databases can be easily scaled vertically?
    a. Amazon RDS
    b. MySQL
    c. Cassandra
    d. Google Cloud Spanner

6. What is the main advantage of vertical scaling?
    a. It is harder to upgrade and may involve downtime.
    b. It can help to distribute the load and improve the system's overall performance.
    c. It is easier to implement.
    d. It is easier to upgrade.

7. What is a text editor?
    a. A program that allows you to edit images.
    b. A program that allows you to edit audio files.
    c. A program that allows you to edit videos.
    d. A program that allows you to edit text files.

8. What is the main advantage of using a text editor?
    a. It allows you to format and style text easily.
    b. It allows you to manipulate audio files easily.
    c. It allows you to manipulate videos easily.
    d. It allows you to manipulate images easily.

9. Which of the following is a popular text editor?
    a. Adobe Photoshop

b. Microsoft Word
c. Audacity Answer
d. Adobe Premier Pro

10. Which of the following is not a text editor feature?
    a. Image manipulation
    b. Syntax highlighting
    c. Spell checking
    d. Auto-completion

Chapter Questions Answers:

1. B
2. B and C
3. D
4. A, B, and C
5. A and B
6. C and D
7. D
8. A
9. B
10. A

Conclusion

This chapter discussed critical aspects like application architecture, text editors, and tech stacks. They are integral to DevOps. It explained the significance of version Control for DevOps. We trust the concepts are clear now, enabling us to proceed towards more challenging topics in the forthcoming chapters.

References

1. Indeed Editorial Team. (n.d.). *What Is Application Architecture? (With 11 Types and Tips)*. Indeed.Com. Retrieved August 21, 2023, from https://www.indeed.com/career-advice/career-development/what-is-application-architecture
2. *What is Three-Tier Architecture*? (n.d.). Ibm.Com. Retrieved August 21, 2023, from https://www.ibm.com/topics/three-tier-architecture
3. *What is an application architecture?* (n.d.). Redhat.Com. Retrieved August 21, 2023, from https://www.redhat.com/en/topics/cloud-native-apps/what-is-an-application-architecture
4. *Monolithic vs Microservices - Difference Between Software Development Architectures- AWS*. (n.d.). Amazon Web Services, Inc. Retrieved August 23, 2023, from https://aws.amazon.com/compare/the-difference-between-monolithic-and-microservices-architecture/
5. CloudZero Team. (n.d.). *Horizontal Vs. Vertical Scaling: How Do They Compare?* Cloudzero.Com. Retrieved August 23, 2023, from https://www.cloudzero.com/blog/horizontal-vs-vertical-scaling
6. Athow, D. (2022, January 7). *Best text editors of 2023*. TechRadar; TechRadar pro. https://www.techradar.com/best/best-text-editors
7. *What Is A Technology Stack? Tech Stacks Explained*. (n.d.). MongoDB. Retrieved August 23, 2023, from https://www.mongodb.com/basics/technology-stack

Chapter 4 - Developer Operations Landscape

In this chapter, you'll learn about the roles and responsibilities that make up a DevOps team. You'll also explore the tools and techniques that help DevOps professionals do their job effectively.

DevOps isn't just about processes and tools; it's a culture promoting innovation and continuous improvement. This chapter helps you delve into the heart of DevOps culture that encompasses the core principles guiding DevOps professionals. You will also unveil the DevOps lifecycle that draws energy from these qualities and principles.

DevOps Roles and Responsibilities

In DevOps, roles and responsibilities are like the gears of a well-oiled machine, each with a separate existence yet all working together harmoniously toward a common goal. So, what are the key roles and responsibilities define a DevOps team? Here’s a close look.

- **DevOps Evangelist**
  - **Responsibilities:** The DevOps evangelist is the team's linchpin responsible for the seamless integration of development and operations.
  - **Key Tasks:** They automate and streamline processes, manage infrastructure as code, and oversee continuous integration and continuous delivery (CI/CD) pipelines.
- **Software Developer**
  - **Responsibilities:** Developers play a pivotal role in creating code that can be easily integrated and deployed.
  - **Key Tasks:** They ensure that code is maintainable, testable, and compatible with the CI/CD process.
- **System Administrator (SysAdmin)**

- **Responsibilities:** SysAdmins focus on automating server provisioning, ensuring system stability, and collaborating closely with developers.

- **Key Tasks:** SysAdmin responsibilities include managing infrastructure as code, developing and Implementing software features using Agile Methodology, and enabling DevOps solutions development and deployment paradigms.

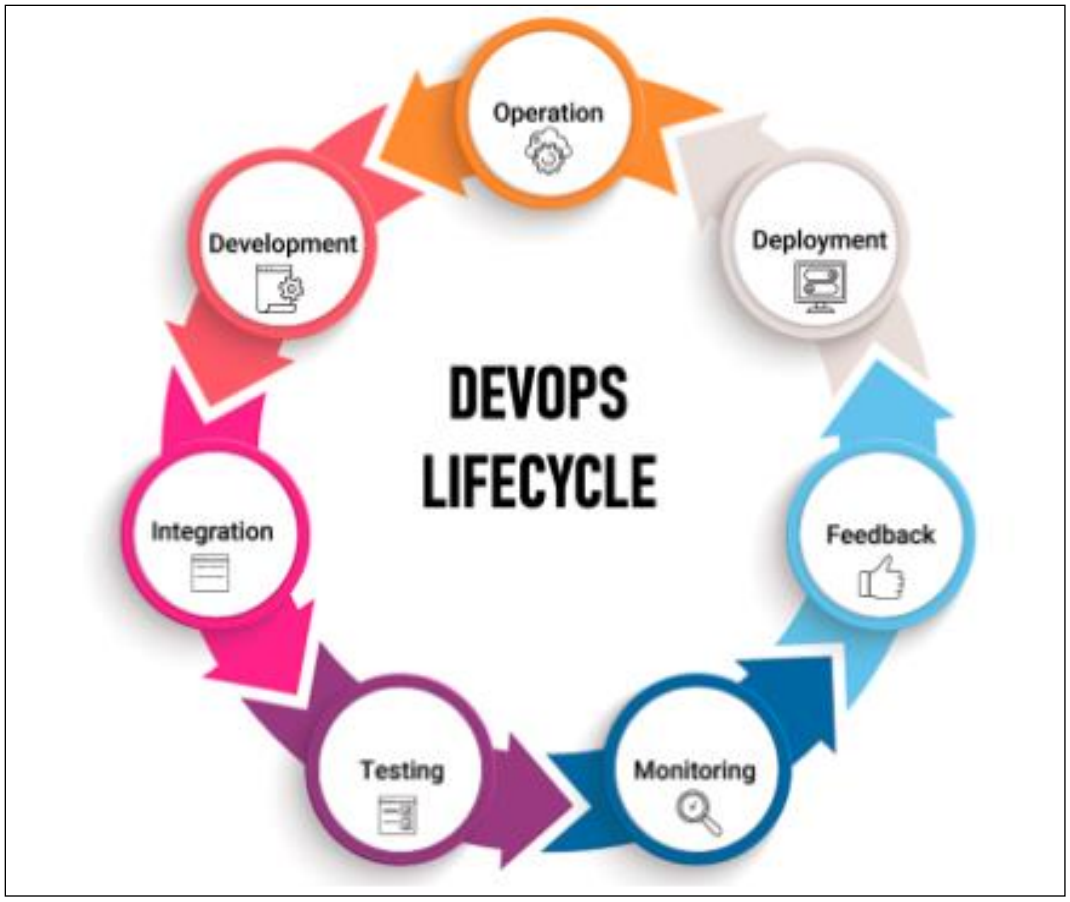
- **Quality Assurance (QA) Engineer**
  - **Responsibilities:** QA Engineers uphold the quality standards throughout the software development lifecycle.
  - **Key Tasks:** They automate testing procedures, identify bugs early, and work alongside developers to maintain code quality.

- **Release Manager**
  - **Responsibilities:** Release managers oversee the deployment process, ensuring smooth and error-free releases.
  - **Key Tasks:** They coordinate between development and operations teams, plan release schedules, and mitigate risks during deployment.

- **Security Specialist**
  - **Responsibilities:** Security specialists ensure the software is secure throughout development and deployment.
  - **Key Tasks:** They conduct security audits, implement best practices, and collaborate with the team to address vulnerabilities.

DevOps vs. Traditional IT Roles

So, what makes DevOps so strong? The true strength of DevOps lies in its emphasis on collaboration and communication among these roles. DevOps promotes a unified approach, unlike traditional IT setups, where departments often work in isolation.



(Image Source: [Spiceworks.com)](Spiceworks.com)

Tools and Techniques in the DevOps Sphere

DevOps relies on tools and techniques that are vital in modern software development. Below are the prominent ones:

- **Version Control:** Various version control systems, such as Git, are at the heart of DevOps. They allow teams to track changes to code, collaborate seamlessly, and manage multiple versions of a project.

- **Automation:** Automation tools, such as Jenkins and Ansible, are the workhorses of DevOps. They automate repetitive tasks like building, testing, and deployment, reducing manual errors and speeding up the development lifecycle.

- **Monitoring:** Monitoring tools, including Prometheus and Nagios, provide real-time insights into system performance. They help identify issues proactively, allowing teams to address them before they impact users. Monitoring is crucial for maintaining system reliability and ensuring a positive user experience.

System Admin Responsibilities as DevOps

As a System Admin transitioning to a DevOps role, there are several responsibilities that you will need to take on. Below are some of them:

- **Infrastructure as Code:** DevOps System Admins automate infrastructure provisioning and configuration using tools like Terraform, CloudFormation, and Ansible.

- **Continuous Integration/Delivery:** DevOps System Admins set up and maintain CI/CD pipelines, automating application build, testing, and deployment stages using tools like Jenkins, GitLab CI, and CircleCI.

- **Monitoring and Logging:** They monitor application and infrastructure performance, analyzing metrics and logs with tools like Prometheus, Grafana, and ELK Stack.

- **Collaboration:** DevOps System Admins collaborate with developers, operations teams, and stakeholders to meet project requirements and ensure seamless deployment.

- **Security:** They also ensure security through practices like least privilege, encryption, and network segmentation.

Skills System Admin Can Leverage to Transition as a DevOps Evangelist

Transitioning from a System Admin to a DevOps Evangelist requires a change in mindset and skillset. Here are the skills you'll need for this.

- **Scripting and Automation:** Use Bash, Python, and PowerShell skills to automate tasks and work with infrastructure as code.

- **Cloud Computing:** Transition from on-premises expertise to cloud platforms like AWS, Azure, and Google Cloud for scalable infrastructure.

- **Containerization:** Shift from virtualization knowledge to Docker and Kubernetes for scalable and portable application deployment.

- **Monitoring and Logging**: Gain experience with tools like Nagios, Zabbix, and ELK Stack, as well as cloud-native tools like CloudWatch, Azure Monitor, and Stackdriver.

- **Problem Solving:** Utilize troubleshooting and problem-solving abilities to address complex issues and find rapid solutions in a DevOps role.

DevOps Culture and Lifecycle

The DevOps lifecycle consists of several stages of continuous software development, including integration, testing, deployment, and monitoring. What exactly are these? Read on for details:

DevOps Culture

The following qualities are integral to a DevOps culture, and members who imbibe the spirit can make the best DevOps team:

- **Collaboration:** Foster closer cooperation and shared responsibility between development and operations teams.

- **Transparency:** Promote increased transparency, communication, and collaboration among traditionally siloed teams.

- **Continuous Learning:** Emphasize ongoing learning and improvement through team autonomy, fast feedback, empathy, trust, and cross-team collaboration.

DevOps Lifecycle

The following are the different stages in the life cycle of DevOps, encompassing everything from the initial conceptual phase to software deployment and beyond:

- **Planning:** Define requirements, create user stories, prioritize tasks, set project goals, and create a roadmap.

- **Development:** Write code, build, test features, and integrate changes into a shared repository.

- **Testing:** Test for functionality, performance, and security, including automated and manual tests.

- **Deployment:** Deploy to production manually or through automation, monitor for errors, and address performance issues.

- **Monitoring:** Collect and analyze metrics and logs, identify issues, improve performance, create alerts, and respond to incidents.

Core Principles of DevOps

Ready to get into the DevOps core principles? DevOps methodology emphasizes collaboration and communication between the development and operations teams, enhancing software delivery speed and quality. The following processes and concepts encapsulate these qualities:

- **Automation:** Automation is a fundamental aspect, streamlining repetitive tasks like testing, building, and deploying software, thus reducing errors and boosting efficiency.

- **Collaboration:** The principle of collaboration extends to all stakeholders, fostering a shared commitment to a common objective among development and operations teams.

- **Continuous Integration/Delivery:** Continuous Integration and Continuous Delivery are core practices that ensure rapid and reliable testing and deployment of software changes through dedicated pipelines.

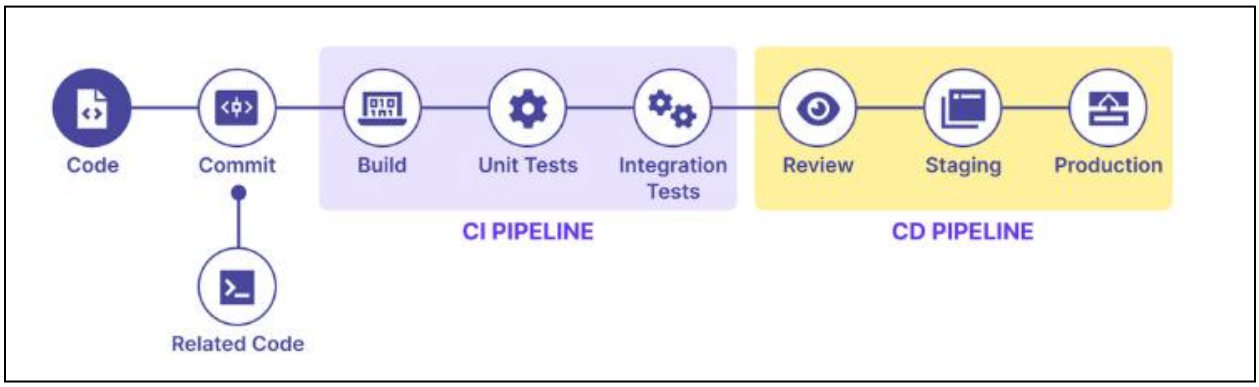
- **Infrastructure as Code:** Infrastructure as Code is essential in DevOps, automating infrastructure provisioning and configuration.

- **Monitoring and Logging:** DevOps also relies on monitoring and logging tools to collect and analyze metrics and logs, pinpointing issues and enhancing system performance.

Introduction to CI/CD Tools

Continuous Integration and Continuous Delivery (CI/CD) are two essential processes in DevOps that help teams deliver software faster and more reliably. CI helps automatically build, test, and integrate code changes into a single repository, while CD automatically

deploys code changes to production or staging environments. CI/CD tools automate processes, making it easier for teams to release software frequently and confidently.

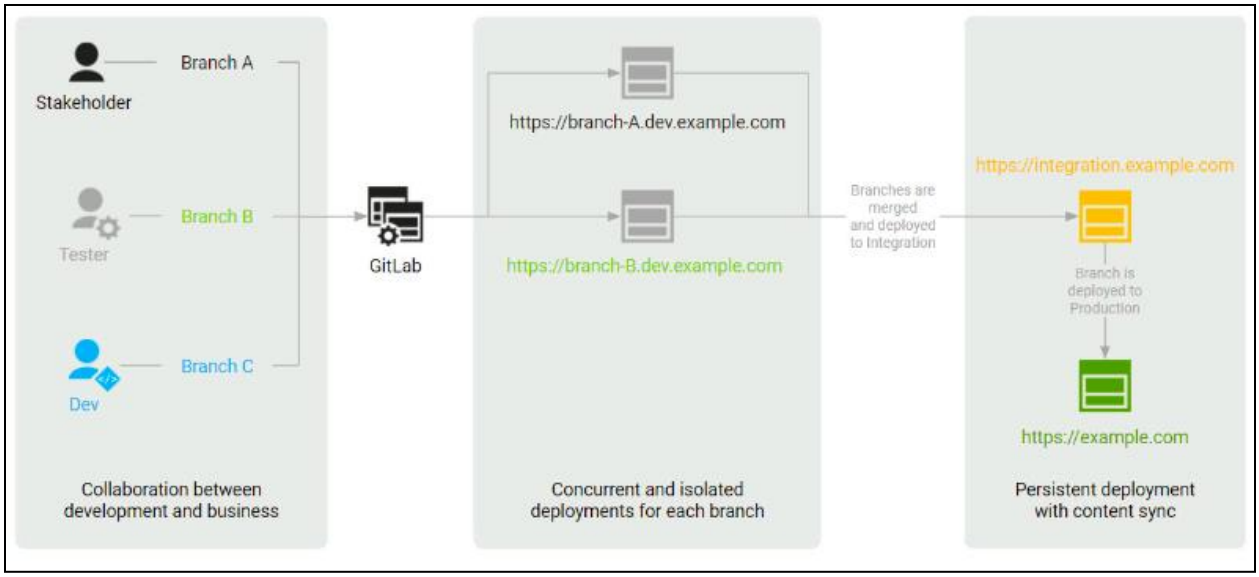


(Image Source: [Katalon.com)](Katalon.com)

Some popular CI/CD tools include:

- Jenkins
- Travis CI
- GitLab CI

These tools encompass a variety of capabilities, including automated testing, deployment, monitoring, and seamless integration with other DevOps solutions, such as version control systems and containerization platforms. How GitLabCI works is explained in the image below.



(Image Source: [Magnolia-cms.com)](Magnolia-cms.com)

CI/CD tools yield numerous benefits, such as accelerated time-to-market, enhanced software quality, and a diminished risk of errors and system downtime. You can delve deeper into CI/CD concepts in the subsequent chapter.

Chapter Questions:

1. What is the primary responsibility of a DevOps Evangelist?

A. The critical role of a DevOps Evangelist is to ensure seamless integration of development and operations.

2. Which DevOps principle emphasizes reducing errors and increasing efficiency through automation?

A. Automation is emphasized by the DevOps principle, which helps reduce errors and increase efficiency.

3. Name one key role that traditionally transitions to a DevOps role and mention a responsibility they may take on in DevOps.

A. A System Administrator (SysAdmin) can transition to a DevOps role and take on responsibilities like automating server provisioning and ensuring system stability.

4. What is the main objective of a Continuous Integration (CI) process in DevOps?

A. The main objective of CI is to automatically build, test, and integrate code changes into a shared repository to ensure early identification of issues.

5. Why is collaboration between development and operations teams considered a core principle of DevOps?

A. Collaboration ensures that everyone works toward a common goal, fostering a culture of shared responsibility and faster, more reliable software delivery.

References

1. Anand, B. (2023, August 24). 35 Best DevOps Tools and Technologies in 2023. Knowledgehut. https://www.knowledgehut.com/blog/devops/top-devops-tools
2. Anand, B. (2023, July 14). DevOps roles and responsibilities: Building an effective team. Knowledgehut. https://www.knowledgehut.com/blog/devops/devops-roles-and-responsibilities
3. Hall, T. (n.d.). What is DevOps culture? Atlassian. https://www.atlassian.com/devops/what-is-devops/devops-culture
4. Bravo, M. (2020, March 18). From sysadmin to DevOps. Red Hat. https://www.redhat.com/sysadmin/sysadmin-devops
5. Hincapie, (2019, March 25). M. DASA DevOps Fundamentals FR 1.3.0 - eBook-English - (en-US). Scribd. https://www.scribd.com/document/435468859/DASA-DevOps-Fundamentals-FR-1-3-0-eBook-English-en-US
6. Duggal, N. (2023, August 9). DevOps engineer job description: Skills, Roles and Responsibilities. Simplilearn. https://www.simplilearn.com/devops-engineer-job-description-article
7. Fallon, A. (2022, April 7). DevOps administrator: The role and responsibilities. IT Operations; TechTarget.

https://www.techtarget.com/searchitoperations/feature/DevOps-administrator-The-role-and-responsibilities

8. Jain, A. (2023, August 7). DevOps Lifecycle: Definition, Phases. Knowledgehut. https://www.knowledgehut.com/blog/devops/devops-lifecycle
9. Simmons, L. (2023, May 16). What is DevOps? Becoming a DevOps engineer. Forbes Advisor. https://www.forbes.com/advisor/education/what-is-development-operations/
10. Sugandhi, A. (2023, August 4). What is DevOps Engineer? Skills, Roles & Responsibilities. Knowledgehut. https://www.knowledgehut.com/blog/devops/what-is-devops-engineer
11. Red Hat. (2022, May 11). What is CI/CD? https://www.redhat.com/en/topics/devops/what-is-ci-cd
12. DevOps Stack Exchange. (n.d.). What is the difference between a Sysadmin and a DevOps Engineer? https://devops.stackexchange.com/questions/157/what-is-the-difference-between-sysadmin-and-devops-engineer
13. AWS. (N.d.). What is DevOps? https://aws.amazon.com/devops/what-is-devops/
14. Image from www.spiceworks.com
15. Image from https://katalon.com
16. Image from https://docs.magnolia-cms.com/

Chapter 5 - Preparing and Hosting Automation Software Builds

Armed with the knowledge of DevOps, Version Control, and Scaling, it is time to think about preparing and hosting automation software builds. This chapter introduces you to the basics of web servers and web hosting. The discussion will also encompass the tools required for front-end and back-end development. Finally, the chapter will conclude with an introduction to Build Tools, which will help carry on with the discussion in the subsequent chapter.

Introduction to Web Server and Web Hosting

As a web development aspirant, you frequently hear terms like web server and web hosting. The below sections will help you gather a clear idea of what web servers and web hosting are.

Web Server

A web server comprises software and hardware and uses protocols, such as HTTP (Hypertext Transfer Protocol), SMTP (Simple Mail Transfer Protocol), and FTP (File Transfer Protocol), to respond to requests clients make on the World Wide Web.

Difference Between Web Server Hardware and Software

The difference between web server hardware and software is that the former connects users to the internet and allows data exchange between connected devices. On the other hand, web server software controls how users access the hosted files. All information systems hosting websites must have web server software.

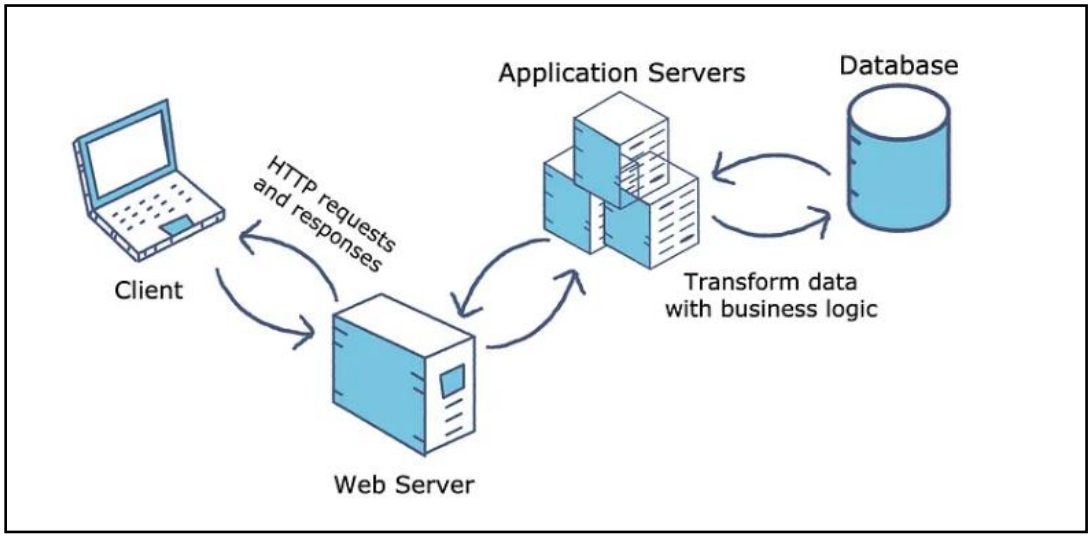
How Do Web Servers Work?

With the basic knowledge of a web server, here is a close look at how it works.

A web server software's primary job is to search the world wide web for websites using their domain names and ensure the delivery of their contents to the requesting user. The web server hardware is an information system that stores web server software and other website-related files like JavaScript files and HTML documents.

For instance, when you intend to visit Google, you type www.google.com on the web browser search bar. Entering this URL starts a request over the internet, which passes

through several layers, one or more of which could be web servers. The web browser translates the URL through DNS or searches in its cache to bring it to the web server. If the information is not found readily, it sends a request to the IP of the web server, which is already configured as per the DNS records. The server responds by sending the browser to the requested page through HTTP.



(Image Source: DevGenius)

The request opens Google's home page, allowing you to browse further and search for your desired content. This process happens instantly and is available 24/7. The server responds with an error message if something goes wrong or the requested page does not exist.

Though the Google homepage looks like a single webpage, it combines multiple resources. Web servers act as intermediaries between the front and *back* ends, serving resources like HTML, CSS, JavaScript files, etc.

Different Types of Web Servers - Static vs. Dynamic

A web server can serve either static or dynamic content. Static content is shown as it is, whereas dynamic content can be updated or altered. The below points will help you understand the difference between static and dynamic web servers.

- A static web server sends hosted files to a browser as they are. It consists of an information system and HTTP software.
- Dynamic web servers consist of web servers and software applications that connect with database servers and application servers.

● While a static server sends the hosted files directly to the browser, dynamic servers can be used for updating the hosted files before sending them to the browser.

Common Web Server Software Available on the Market

Here are some common varieties of web server software on the market.

● ***Microsoft Internet Information Services*** - Microsoft has developed this server for Microsoft platforms. Though it is not open source, it is used extensively.

● ***Apache HTTP Server*** - This web server developed by Apache Software Foundation is free and open-source. It is compatible with most operating systems like MacOS, Windows, Solaris, Linux, and Unix. However, it needs the Apache license.

● ***Sun Java System Web Server*** - This free web server from Sun Microsystems runs on Linux, Unix, and Windows. It can handle medium to large websites.

● ***Nginx*** - Administrators love using this open-source web server because of its scalability and light resource utilization. Its event-driven architecture helps it handle multiple sessions concurrently. Nginx also serves as a proxy server and load balancer.

● ***Lighttpd*** - This free, fast, and secure web server comes with the FreeBSD operating software. The best aspect is that it consumes less CPU power.

Common Use Cases of Web Servers

A web server is specifically used for handling requests on the internet through HTTP and HTTPS protocols. Hence, they are also known as HTTP servers. A web server is essential to make websites or apps that should connect to the internet.

Here are some common tasks that web servers handle:

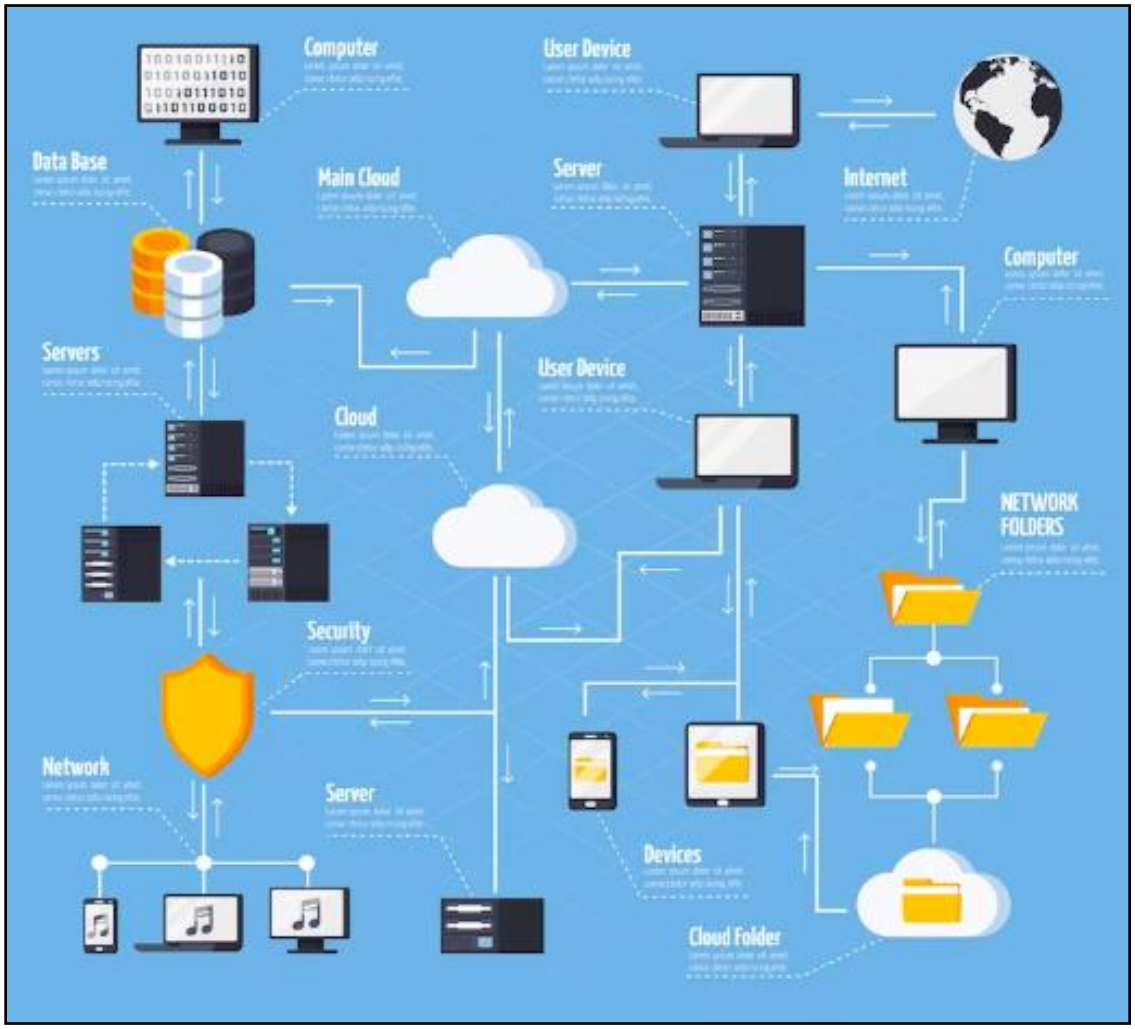
- Serving JavaScript, CSS, and HTML files
- Handling HTTP error messaging
- Serving images and videos
- Handling user requests
- Processing and serving dynamic content
- Directing URL matching and rewriting
- Enabling browser caching for static content
- Compressing content for optimized speed and data usage

Web Hosting

Now that you know what a web server is, it will be easy to understand web hosting and associated concepts. While a web server comprises software and hardware, web hosting

is a service and process through which files and applications are stored on a web server. The hosted files are publicly available at all times.

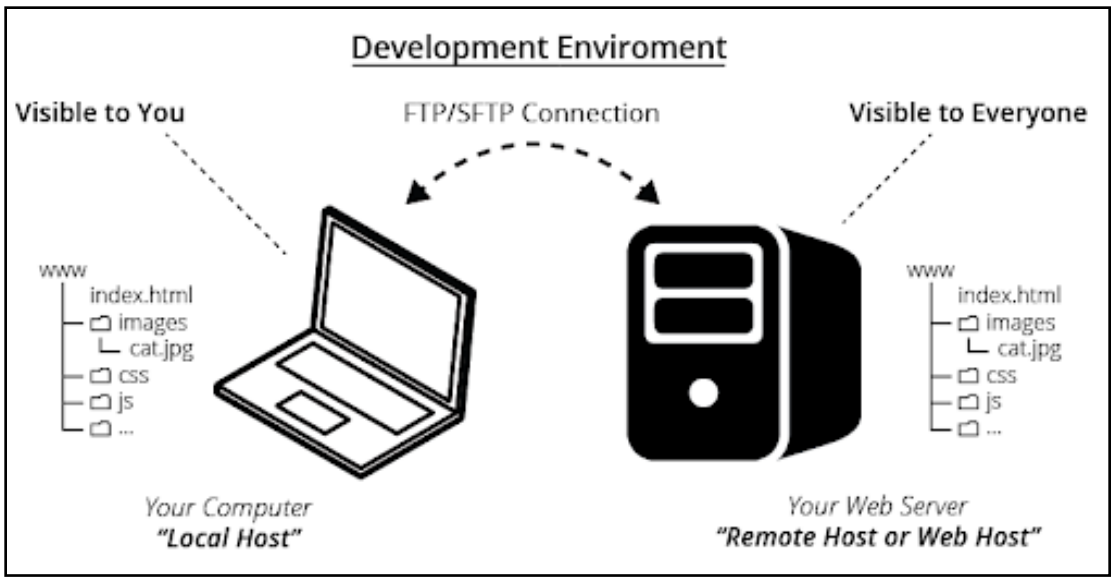
Web hosting requires information systems capable of offering 24x7 uptime to serve multiple client requests simultaneously. Your PC can be set up as a web server, and you can run applications and websites from it. It is the simplest example of web hosting. However, you must keep the PC running full-time because web servers require 100% uptime.



(Image Source: Freepik.com)

How to Use Web Hosting?

The two ways of performing web hosting are explained below:



(Image Source: Mobile-dev)

Hosting on Your Local Computer

The following are the requirements for hosting websites on your local computer:

- First, you will need to set up your web server.
- It requires an internet connection available 24/7.
- You must ensure that your ISP supports this type of usage and can handle large traffic requests.
- Besides, you must know how to handle network hardware and have website administration skills like installing, configuring, and managing a web server, and handling virus protection, patch management, and firewalls.

Using a Hosting Service Provider to Host Websites

Using a hosting service provider is better because it ensures quick website launching and efficient file hosting. Here are the advantages of using a hosting service provider:

- With hosting service providers maintaining web hosting, organizations find it convenient to focus on developing the website or application.

(Image Source: Freepik.com)

- Hosting service providers provide control panels for managing applications using GUI for enhanced efficiency.
- They assure 99.9% uptime with continuous FTP and HTTPS support.
- It is cost-effective because hosting services can offer better hardware to handle traffic requirements.
- Besides, hosting services offer additional support, enabling easy communication and resolution.
- Organizations can also benefit from services like CDN, image hosting, business emails, DDoS mitigation, website backups, and database hosting.

Different Types of Hosting

Below are the three most common types of web hosting:

Shared Hosting

Here, the hosting service provider provides web hosting for multiple clients on a single information system, with each client sharing the CPU, storage space, memory, and web server software. Therefore, it reduces the cost factor significantly.

However, the drawback is that it compromises security because one affected website can render other sites on the server vulnerable. Besides, unusually high traffic to one hosted site can slow down other websites and deplete their resources.

Shared hosting is feasible for personal websites, non-transactional business websites, and blogs.

Virtual Private Server (VPS)

Considering the drawbacks of shared hosting, organizations prefer to have a dedicated virtual server. Such a hosting service is called Virtual Private Server (VPS) hosting. Generally, you share hardware resources but with fewer co-tenants. Hence, it reduces the chances of security breaches and website crashes.

VPS hosting allows organizations to enjoy complete control over operating systems, CMS, and other software. Hence, VPS is ideal for hosting custom web applications or web-based SaaS (Software-as-a-Service). These features make VPS hosting more expensive but simultaneously more secure than shared hosting.

Cloud-based VPS hosting is a better form of VPS hosting because it allows more effortless scalability, offers more storage capacity and bandwidth, and makes the systems more resilient to hardware issues or natural/human-made disasters.

Dedicated Hosting

While VPS hosting is safer and better than shared hosting, it has limitations. Dedicated hosting is ideal because it gives you exclusive control over your web server software. Since you are the only entity using this hosting service, your site runs faster than those using other hosting services. It is also less vulnerable to cyberattacks from security issues of other websites.

However, dedicated hosting has drawbacks, such as being expensive. Secondly, you must pay additional fees to the service provider to manage the service if you do not have the talent to do it yourself. Scaling is also an issue because the server needs more RAM and storage when needed.

Therefore, dedicated hosting is only recommended when the security considerations and performance justify the additional costs and inherent drawbacks.

Tools Required for Front-end Development and Back-end Development.

Creating a website requires developers to use different front-end and back-end tools.

Front-end tools are the programming languages that decide the look and feel of your website. Besides, these tools ensure user-friendly features like mobile/desktop views, drop-down menus, etc.

On the other hand, back-end tools are the languages, servers, databases, etc., essential for writing the website software's code that will control the system's deeper functions. They ensure simplified operations and that various codes communicate with each other appropriately.

Here are some of the most commonly used front-end and back-end tools for web development.

Ten Commonly Used Front-end Web Development Tools:	
Chrome DevTools	● Chrome DevTools allows editing web pages directly from Google Chrome and diagnosing problems quickly to help build better websites. ● Editing HTML and CSS code or debugging JavaScript code is easy in a real-time environment using this tool. ● Additional advantages include: ▪ **Timeline -** Recognizes run-time problems ▪ **Device Mode -** Tests website responsiveness ▪ **Sources Panel -** Debugs JavaScript using breakpoints ▪ **Network Panel -** Views and debugs network activity
Visual Studio Code	● This Microsoft open-source code editor has features like smart code completion, syntax highlighting, an inbuilt debugger, easier deployment capabilities, and inbuilt Git commands.

	• It has different types of free-to-use plugins for almost every development segment. • It is available for MacOS, Windows, and Linux. • It provides inbuilt support for TypeScript, Node.js, JavaScript, and languages like Python, PHP, Java, and C++.
HTML5 Boilerplate	• It is a simple and helpful tool allowing developers to create robust, fast, and adaptable apps and websites. • Integrating with other front-end frameworks is easy. • Some top organizations using this tool include Microsoft, NASA, and Nike.
Git	• Git, a popular version control system, allows for conveniently managing source codes, tracking changes, or rolling back codes. • It smoothens collaboration among developers by reducing risks like code conflict. • Git is free, open-source, and secure.
Sass	• Sass is an extensively used CSS extension language. It extends CSS's functionality, like inline imports, easier nesting, inheritance, and variable creation. • The top organizations using Sass include Bourbon, Susy, and Compass.
TypeScript	• This programming language forms a syntactical superset of JavaScript. • It can be executed on any browser or JavaScript engine. • It is more structured and concise,

	has better documentation, upholds interfaces, and provides better tooling support with IntelliSense.
AngularJS	● This open-source front-end framework extends static HTML to dynamic HTML, allowing you to build dynamic websites. ● Creating client-side apps using JavaScript is easy. ● Additional features include Data Binding, Controller, Reusable Components, and Directives.
Npm (Node Package Manager)	● Npm serves as a package manager for Node JS. ● Installing and publishing packages is easy using the command ‘npm init.' ● This tool helps discover and install packages of reusable code in programs.
jQuery	● This cross-platform JavaScript library helps with event handling, animation, DOM manipulation, CSS manipulation, Ajax/JSON support, etc. ● jQuery has a massive development community, robust chaining capabilities, better documentation, and other advantages.
Grunt	● Grunt is an excellent front-end tool for talk automation. ● It efficiently automates repetitive tasks like linting, unit testing, and compilation. ● Popular organizations that use Grunt include X (earlier Twitter), Mozilla, Adobe, and Walmart.

Ten Commonly Used Back-end Development Tools	
JavaScript	● JavaScript is a lightweight language integral to almost all web development projects. ● Knowledge of the core technologies or, at least, working knowledge of JavaScript is essential for web developers. ● It is helpful for front-end and back-end development.
C#	● C# is an object-oriented programming language deployed by Microsoft's asp.net and others. ● C# allows for quick coding. ● It is suitable for most web applications.
Python	● Python is a powerful programming language capable of handling simple and complicated jobs. ● It is compatible with most trending technologies and is used by most platforms. ● It is one of the easiest languages to learn and works well with HTML, CSS, and front-end tools.
Java	● Java is among the most widely used programming languages. ● Most applications and platform development projects use Java. ● Java, an object-oriented programming language, has syntax like C and C++.
PHP	● PHP is the most commonly used back-end language for many popular CMS tools, like WordPress. ● It helps extend your website and create plugins. ● PHP is flexible and does not compromise security.

MongoDB	● MongoDB, an open-source database, is gaining popularity among web app developers. ● It offers JSON-like schemas and documents for storing and retrieving data. ● It is lightweight and scalable.
Oracle	● It is a popular database offering on-site and cloud-based web application support and development. ● It is extensible with databases like MongoDB. ● Oracle offers easy migration services for onboarding applications.
MySQL	● MySQL is the most famous database management system that is open-source and used in back-end development projects. ● It is reliable and supports various features.
Apache	● It powers more than 50% of the websites in the world. ● Apache is customizable and can be modified to suit individual needs. ● It is compatible with almost all operating systems.
Microsoft IIS	● Microsoft IIS is an extensible software for web servers in Windows operating systems. ● It supports various HTTP protocols.

Introduction to Build Tools

The discussions so far have covered DevOps, version control systems, and the different types of front and back-end development tools. It's time to get acquainted with build tools and look at the most commonly used ones.

What are Build Tools?

Build Tools are programs that automate the executable application-building process from the source code. The process involves compiling, linking, and packaging the code in an executable form. The automation continues with running the tests and deploying the code to production systems.

Below are details about the commonly used build tools for DevOps.

SBT - Scala-oriented Build Tool
● SBT is meant for programming languages like Scala and Java and is specifically targeted to work on projects involving them. ● SBT offers a unique interactive mode to expedite the building process. ● It can compile Scala code, package archive artifacts, execute tests, and support other build operations.
Advantages
● It is ideal for small and simple projects. ● It is generally identified with Scala open-source products. ● It provides excellent integration when using IntelliJ IDEA IDE for development. ● Most tasks are usage-ready.

Terraform
● Terraform is dedicated to building, changing, and versioning infrastructure. ● Managing existing service providers with in-house solutions is easy with Terraform. ● Terraform generates execution plans and builds the described infrastructure. ● It identifies configurational changes and creates incremental execution plans for development. ● It can manage and include low-level components.
Advantages
● Terraform offers a custom HCL to create templates and simplify documenting and commenting on your code.

- It allows users to write data scripts separately as you do on the server locally.

Gradle
● Gradle is an easy way to build projects because it focuses on eliminating XML as part of the build script generation. ● Instead, Gradle uses a DSL (domain-specific language) based on Groovy that can be run on the JVM. ● Gradle lets you define the core parts of the build file and the specific steps called TASK. Hence, it is extensible, making it convenient to define customizable tasks.
Advantages
● Adding dependencies and applying plugins are easily manageable. ● Gradle offers an entirely automated build process that allows multi-module builds. ● Gradle is more comfortable to use than Apache Maven or ANT. ● It integrates well with IDEs like IntelliJ IDEA and Eclipse.

Apache Maven
● Besides being a build tool, Apache Maven can serve as a project management and comprehension tool for software. ● It is founded on the Project Object Model (POM) and can manage a project from a central piece of information. ● Apache Maven is an XML-based build file but outlines rigid standards for itself. ● Apache Maven is a two-in-one tool: a powerful build tool and dependency manager.
Advantages
● It enables configuring the entire project by glancing through one crucial file, pom.xml. ● The Dependency Manager reduces the burden of updating the dependencies. ● You can quickly build a Cucumber project or a TestNG project. ● It supports any platform for the build process and all unit testing and logging requirements.

CruiseControl
● CruiseControl fits into the Continuous Integration and Builds Space of the DevOps Tools. ● CruiseControl is an extensible framework for creating a customized continuous build process. ● Features like email notification and instant messaging make it a popular tool. ● It has an interactive web UI to provide all the necessary details about the previous builds of the current project.
Advantages
● It is a straightforward tool so that anyone from the user group can pick the corresponding tasks from their standpoint. ● It provides solid community support. ● CruiseControl is a pioneer in automated builds and CI. ● CruiseControl can be easily customized to suit individual requirements. ● The interactive dashboard allows control of all the information about the builds.

Apache Buildr
● Apache Buildr is ideal for Java-based applications. ● It supports other programming languages like Groovy and Scala. ● Apache Buildr is an intelligent tool that does not require developers to provide much info. Besides, it is fast and reliable.
Advantages
● Apache Buildr is easy to use with various projects. ● It can be used to build small and large projects. ● Preconfigured tasks keep the build scripts on the DRY side, easy to maintain. ● It supports APT source code generation and Javadoc. ● Managing upgrades to newer versions is easier.

CMake

● CMake is a cross-platform, free, open-source software that uses the compiler-independent method to build software processes. ● It works in conjunction with build environments like Visual Studio and Xcode. ● It supports directory hierarchies and applications that rely on multiple libraries.
Advantages
● CMake enables cross-platform discovery of system libraries. ● It provides automatic discovery and configuration. ● CMake is easier to use than its predecessor, Make. ● It delivers better performance than Make.

Chapter Questions:

Having discussed various aspects of preparing and hosting automation software builds, it's time to test what you've learned in this chapter through this simple MCQ test.

1. What is a web server?
    a. An information system that runs websites
    b. A process of using a server to host websites.
    c. A way to get a website hosted to enable everyone to access it on the internet.
    d. A way to prevent malicious actors and malware from accessing data.
2. What is web hosting?
    a. A way to get websites hosted to enable everyone to access them on the internet.
    b. computer that runs websites
    c. A way to prevent malicious actors and malware from accessing data.
    d. A process of using a server to host websites.
3. What is the primary purpose of a web server?
    a. To get a website hosted so everyone can access it on the web
    b. To prevent malicious actors and malware from accessing data
    c. To store website files and share them over the internet
    d. To provide good security services
4. What is the primary purpose of web hosting?
    a. To store website files and share them over the internet
    b. To get a website hosted so everyone can access it on the web
    c. To provide good security services
    d. To prevent malicious actors and malware from accessing data
5. What is a front-end development tool?
    a. A tool used to read information and write information onto a server

    b. A software application that helps developers easily build attractive website layouts and apps
    c. A tool used to write APIs and develop information architecture
    d. A programming language employed for writing software for deeper functions of a website
6. What are some examples of front-end development tools?
    a. Django, Flask, Express
    b. Oracle, MySQL, MongoDB
    c. HTML5 Boilerplate, Chrome DevTools, Sass
    d. PHP, Python, Ruby
7. What is a back-end development tool?
    a. A programming language to write the software for deeper functions of a website
    b. A software application that helps developers build attractive website layouts and apps
    c. A tool used to read information and write information onto a server
    d. A tool used to write APIs and develop information architecture
8. What are some examples of back-end development tools?
    a. PHP, Python, Ruby
    b. HTML5 Boilerplate, Chrome DevTools, Sass
    c. Oracle, MySQL, MongoDB
    d. Django, Flask, Express
9. What is a build tool?
    a. A tool used to develop information architecture
    b. A tool used to read information and write information onto a server
    c. A tool used to prevent malicious actors and malware from accessing data
    d. A tool used to manage and organize builds
10. What are some examples of build tools?
    a. Django, Flask, Express
    b. Oracle, MySQL, MongoDB
    c. ANT, Maven, Gradle
    d. HTML5 Boilerplate Chrome DevTools, Sass

Chapter Questions Answers:

1. A
2. D
3. C
4. B
5. B
6. C
7. A
8. A
9. D
10. C

Final Words

Hope that web-building concepts are clear now. This chapter discussed the basics of web servers and web hosting. It also explained the concepts of front-end and back-end development tools while listing some of the most commonly used tools. Simultaneously, it examined what build tools are and discussed several widely used ones today. This chapter should provide the perfect base for more challenging assignments awaiting you in the following pages.

References

1. Gillis, A. (2020, July). *Web Server.* WhatIs. TechTarget. https://www.techtarget.com/whatis/definition/Web-server
2. W3schools. (n. d.). *Introduction to Web Hosting.* https://www.w3schools.in/web-hosting/introduction
3. Tran, T. (2022, June 28). *Introduction to Web Servers*. DigitalOcean. https://www.digitalocean.com/community/conceptual-articles/introduction-to-web-servers
4. IBM. (n.d.). *What is Web Hosting?* https://www.ibm.com/topics/web-hosting
5. Madhur. (2021, March 27). *10 Best Tools For Front-End Web Development*. GeeksforGeeks. https://www.geeksforgeeks.org/10-best-tools-for-front-end-web-development/
6. Smith, M. (2023, February 7). *The Best Back-End Tools for Web Developers to Use in 2023*. HubSpot. https://blog.hubspot.com/website/backend-tools
7. Savaram, R. (2023, September 01). *Top 12 Open Source DevOps Build Tools.* Mindmajix. https://mindmajix.com/12-open-source-devops-build-tools

Chapter 6 - Continuous Integration / Continuous Delivery

This chapter will help you understand the CI and CD concepts in detail and also introduce you to Jenkins and popular infrastructure deployment tools.

There are a handful of processes that go into the making of software without hitches. Here's a peek behind the curtain to understand Continuous Integration and Continuous Delivery (CI/CD) concepts, the dynamic duo making modern development tick.

What Is Continuous Integration / Continuous Delivery?

Continuous Integration, or CI, is a software development practice where code changes made by multiple developers are frequently combined into a shared repository. Thus, CI is like a watchman for smooth code teamwork. Developers often combine their code changes to test them to dodge last-minute confusion before launch.

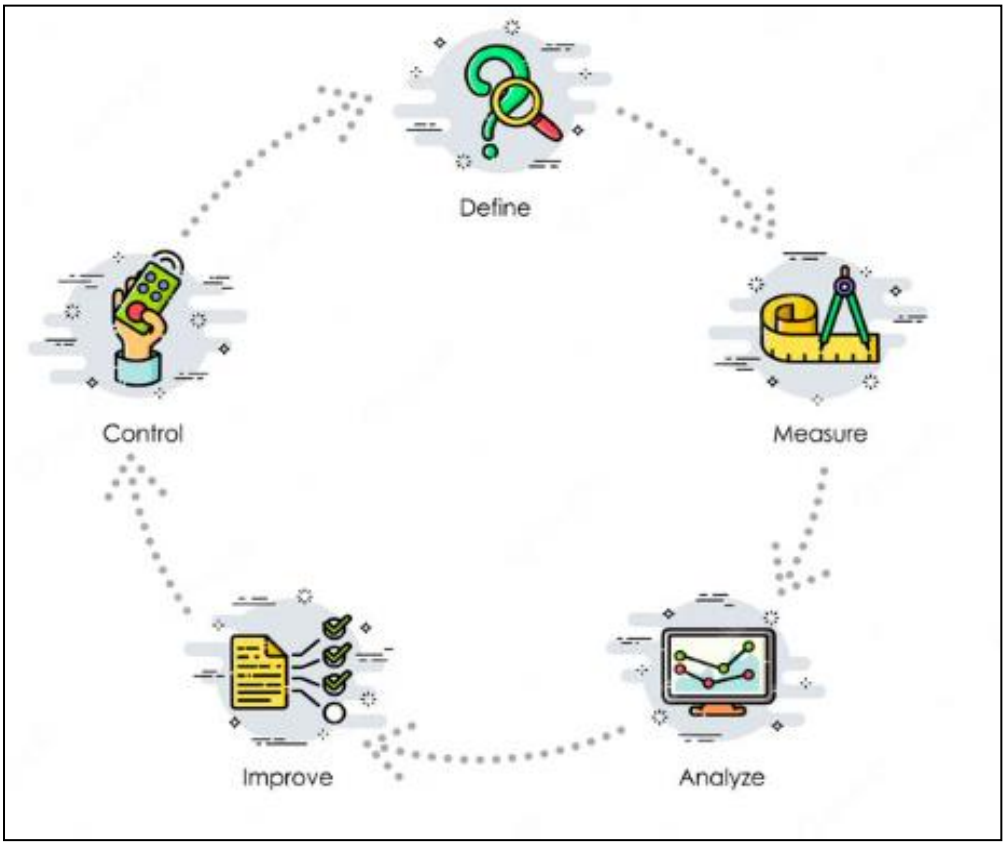
On the other hand, Continuous Delivery, or CD, is an extension of CI where validated code changes are automatically delivered to production environments. Once code changes are ready, they zoom into the real world.

The CI/CD Workflow Explained: How to Integrate and Validate Source Code Changes

The integration and validation of source code changes are vital in ensuring reliable software delivery. The below sections provide a dive into the heart of these processes to see how it all works.

Breakdown of the CI/CD Pipeline Stages: Integrate, Build, Test, and Deploy

The CI/CD pipeline consists of several interconnected stages, each serving a distinct purpose.



(Image Source: Freepik.com)

1. **Integrate:** Integration involves merging code changes from different developers and branches into a shared repository to ensure that the codebase remains cohesive and functional.
2. **Build:** Next comes the building stage. The code is compiled, transformed, and prepared for testing and deployment. Building is about creating a reliable and consistent foundation for the upcoming phases.
3. **Test:** Then comes the testing stage. Testing is where the real quality assurance takes place. Various tests, such as unit, integration, functional, and performance tests, are automated.

4. **Deploy:** The deployment stage is about releasing the tested code changes into production or other environments. It can be a complex process involving various considerations, such as release orchestration and gradual rollouts.

Rationale Behind Each Stage and the Benefits of Automation

Each stage in the CI/CD pipeline serves a critical purpose. Here's a quick look at the benefits of each one.

- Integration prevents code fragmentation, enabling developers to collaborate effectively.
- Building ensures that the software is deployable and free from errors.
- Automated testing provides rapid code quality and functionality feedback, catching bugs early in the development process.
- Deployment automation reduces human errors, accelerates release cycles, and enables the rapid delivery of features to end-users.

Version Control and Collaborative Development

Version control systems like Git are indispensable in orchestrating the intricate process described above. They provide a structured environment for collaborative coding, enabling developers to work concurrently without disrupting each other's progress.

Furthermore, branching strategies, pull requests, and code reviews enhance this collaborative experience. These practices ensure that code changes are thoroughly reviewed, discussed, and validated before being integrated into the main codebase.

Continuous Integration: Glue of the Workflow

At the heart of all this lies CI, as Continuous Integration promotes the frequent integration of code changes into the main codebase. In a nutshell, continuous integration mitigates

risks, encourages regular collaboration among developers, and serves as the bedrock for achieving seamless delivery. CI shines as a beacon of efficiency, reliability, and cooperation to help the development process progress from code creation to deployment.

Introduction to Jenkins: How to Install and Configure

Jenkins is an efficient helper in automating the whole software development process, including putting code together and deploying it. Here's more information on it.

What is Jenkins?

Jenkins is a widely used CI/CD tool that helps automate and streamline the process of building, testing, and deploying software. It is famous for its flexibility, open-source nature, and the ability to extend its functionality through various plugins.

How to Install Jenkins?

You can easily install Jenkins on different platforms, including Windows, Linux, and Docker. Here's how:

Windows

1. Download the Jenkins installer for Windows from the official website.
2. Run the installer and follow the steps shown.

```
// Navigate to the downloaded installer location
cd path\to\downloaded\installer
// Execute the installer
.\jenkins.msi
```

Linux

1. Open a terminal on your Linux system.
2. Use package managers like 'apt' or 'yum' to install Jenkins.

```
# For Ubuntu
sudo apt update
sudo apt install jenkins
```

```
# For CentOS
sudo yum update
sudo yum install jenkins
```

3. Start the Jenkins service and set it to start on boot.

```
# For CentOS
sudo yum update
sudo yum install jenkins
```

Docker

1. Install Docker on your system if you haven't already

```
# For Ubuntu
sudo apt update
sudo apt install docker.io
```

2. Pull the Jenkins Docker image using the command provided on the official Jenkins Docker Hub page.

```
docker pull jenkins/jenkins:lts
```

3. Run a Docker container using the pulled image, exposing port 8080.

```
docker pull jenkins/jenkins:lts
```

After installing it on your operating system, you can open it by visiting http://localhost:8080 on your web browser.

Configuring Jenkins and Setting It Up

After installing Jenkins, you need to perform some initial setup and configuration. Here is what you need to do:

Access Jenkins Web Interface

- Open Jenkins at http://localhost:8080/

- Retrieve the initial admin password from the Jenkins installation directory and enter it.

```
cat /var/lib/jenkins/secrets/initialAdminPassword
```

- You'll be prompted to install either the recommended plugins or your own selection. Install recommended plugins or select specific plugins based on your needs.

```
java -jar jenkins-cli.jar -s http://localhost:8080/ install-plugin [plugin-name]
```

- You may be instructed through a wizard to set up an Admin User Account, but if you want to automate user creation later, you could create an Admin User account with a username and password.

```
Jenkins.instance.securityRealm.createAccount('new_user', 'password')
```

- Place this in a ***.groovy*** file and load it into the Jenkins script console.
- Configure the URL for Jenkins and complete the setup.

```
<jenkins.model.JenkinsLocationConfiguration>
  <adminAddress>admin@sampledomain.com</adminAddress>
  <jenkinsUrl>http://sample_jenkins_url/</jenkinsUrl>
</jenkins.model.JenkinsLocationConfiguration>
```

Creating Jenkins Jobs

Jenkins jobs are the building blocks of your CI/CD pipeline. They define specific tasks that need to be executed. Here's how you can create one:

- Log in to your Jenkins dashboard.
  First, open a web browser and navigate to the Jenkins dashboard, which is http://localhost:8080

```
# Assuming you're running Jenkins locally
curl http://localhost:8080
```

- Click on "New Item" to create a new job.

```
# If you're using Jenkins CLI, you could use
jenkins create-job <sample_job_name>
```

- Enter a name for your job and select the type of job (e.g., Freestyle or a pipleine or sample project).

```
<!—This is a sample XML configuration for using Job DSL -->
```

```
<project>
  <description></description>
  <keepDependencies>false</keepDependencies>
  <!-- Other configurations here -->
</project>
```

- Configure the job details, such as defining the source code repository, build triggers, and build steps.

```
// A sample Jenkins file for pipeline
pipeline {
    agent any
    stages {
        stage('Checkout') {
            steps {
                checkout scm
            }
        }
        // more required steps here
    }
}
```

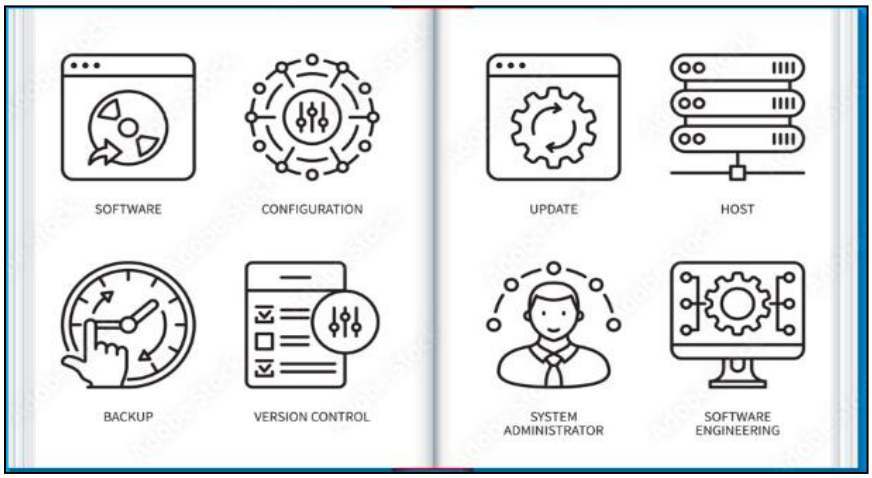
- Save your job configuration.

```
# Save your job configuration by using the command below
jenkins reload-job <job_name>
```

Integration with Version Control

Jenkins can be configured to monitor repositories for changes. Upon detecting new commits, it triggers relevant jobs, initiating the CI/CD pipeline. This integration ensures

that every code change goes through the automated pipeline, maintaining the integrity of the software.



(Image Source: Freepik.com)

Infrastructure Environment Deployment Made Easy

When looking to deploy your infrastructure, you have various powerful tools available. Below is a brief examination of the world of IaC (Infrastructure as Code), which helps configure your infrastructure using code.

Simplifying Deployment with IaC

Have you ever wished for a way to deploy your infrastructure consistently like you do with code? That's where IaC comes in. With IaC, you can treat your infrastructure configuration like lines of code, allowing you to automate deployment processes and ensure consistency.

IaC provides you with the ability to define your infrastructure using code. You can easily replicate setups, reduce human errors, and roll out changes faster. Besides, by using IaC, you can adapt to the evolving needs of your projects without breaking a sweat.

Most Popular Infrastructure-as-Code Tools

Navigating the world of automation is made simpler with the right tools. The below information shows how Terraform, Ansible, and CloudFormation revolutionize the way you deploy infrastructure:

- **Terraform**
  It's a popular open-source IaC tool with much flexibility and supports significant cloud platforms such as AWS and Azure. Its declarative approach lets you define your setup using easy-to-understand code.

- **Ansible**
  This agentless automation tool lets you define your desired state and automate repetitive tasks. Ansible's simplicity and flexibility streamline the deployment process from server setup to software installations.

- **CloudFormation**
  If you're an AWS user, CloudFormation is your go-to tool. It allows you to model and provision AWS resources using JSON or YAML templates.

Exercise: Build Your Own Jenkins Job

Now that you're familiar with CI/CD and Jenkins, here's a simple exercise to try putting the knowledge into practice. Create a simple Jenkins job that performs the following tasks:

- Retrieves code from a sample Git repository.
- Compiles the code.
- Executes a basic automated test.
- Deploys the code (to a mock environment)

This exercise will help you gain hands-on experience with Jenkins and understand how the CI/CD pipeline works in practice:

Exercice : Jenkins as a CI-CD Environnent

Objective: This exercise aims to demonstrate the essential features of Jenkins in a Continuous Integration and Continuous Deployment (CI-CD) environment. By the end of this exercise, you will be able to set up a simple Jenkins pipeline, build code, and deploy an application.

Prerequisites:

- Jenkins installed and running
- A sample code repository (e.g., a GitHub repository with some Python or Java code)
- Basic understanding of source control and versioning

Steps:

1. Open a web browser and navigate to your Jenkins dashboard after log in to Jenkins
2. Create a new job by going to 'New Item' > Enter a job name > Select 'Freestyle project' > Click 'OK'.
3. Configure source control under 'Source Code Management', select your source control system (e.g., Git) and enter the URL of your code repository.
4. Set build triggers under 'Build Triggers', select how you want to trigger this build (e.g., 'Poll SCM' for automated builds or manual triggering).
5. Add build step by going to 'Build', add build steps based on your project. For a Java project, you might add 'Invoke top-level Maven targets' or for Python, you could add a shell command to run your scripts.
6. Add Post-build actions where you can add actions like 'Archive the artifacts', or 'Deploy to environment' depending on what you need in your CI/CD pipeline.
7. Save and build by clicking 'Save,' and then click 'Build Now' to see your pipeline in action.

8. Review the build and deploy status once the build is complete, check the console output for any errors, and ensure your application is deployed as expected.

Verification

- Ensure the build is successful and all post-build actions are completed as configured.
- Check if the application is deployed successfully in the environment you configured.

Chapter Questions:

Let’s answer the following quick quiz and see how much you learn.

Q1: Name one popular CI/CD tool.

Q2: What does IaC stand for?

Q3: What is the purpose of the Build stage in CI/CD?

Q4: What's the benefit of Infrastructure as Code (IaC)?

Q5: What's the role of version control systems like Git?

Chapter Questions Answers:

A1: Jenkins.

A2: Infrastructure as Code.

A3: The Build stage compiles and prepares code before testing and deployment.

A4: IaC treats infrastructure configuration as code, enabling automated deployment processes and ensuring consistency.

A5: Version control systems like Git provide a structured environment for collaborative coding, allowing developers to work concurrently without conflicts.

References

1. Pittet, S.Atlassian. (n.d.). Continuous integration vs. delivery vs. deployment. Atlassian. https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment
2. Atlassian. (n.d.). Source code management. https://www.atlassian.com/git/tutorials/source-code-management
3. Despa, V. (2020, January 24). How to install and configure Jenkins on Windows 10. Coralogix. https://coralogix.com/blog/how-to-install-and-configure-jenkins-on-windows-10/
4. Hamilton, T. (2023, July 1). How to download & install Jenkins on Windows. Guru99. https://www.guru99.com/download-install-jenkins.html
5. Kelley, K. (2023, January 13). Managing your CI/CD continuous improvement with pipelines. Simplilearn. https://www.simplilearn.com/tutorials/jenkins-tutorial/ci-cd-pipeline
6. Scaled Agile Framework. (2023, January 6). Continuous Integration. https://scaledagileframework.com/continuous-integration/
7. Pialoux, F. (n.d.). Best Infrastructure as Code Tools (IaC): The Top 11 for 2023. Bluelight. Retrieved August 30, 2023, from https://bluelight.co/blog/best-infrastructure-as-code-tools
8. Powell, R. (2023, April 28). Canary vs blue-green deployment to reduce downtime. CircleCI. https://circleci.com/blog/canary-vs-blue-green-downtime/
9. Johanna. (2018, September 3). Using multiple environments to improve your development workflow. Deploybot. https://deploybot.com/blog/using-multiple-environments-to-improve-your-development-workflow
10. Sacolick, I. (2022, April 15). What is CI/CD? Continuous integration and continuous delivery explained. InfoWorld. https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html
11. Singh, G. (2023, August 15). Infrastructure as code tools to boost your productivity in 2023. Xenonstack. https://www.xenonstack.com/blog/infrastructure-as-code-tools

12. Son, B. (2019, September 5). Building CI/CD pipelines with Jenkins. Opensource. https://opensource.com/article/19/9/intro-building-cicd-pipelines-jenkins

13. Microsoft. What is infrastructure as code (IaC)? (2022, November 29). https://learn.microsoft.com/en-us/devops/deliver/what-is-infrastructure-as-code

Chapter 7 - Preparing Data Storage and Cloud Databases

This chapter provides insights into databases and storage, focusing on SQL, NoSQL, and cloud databases.

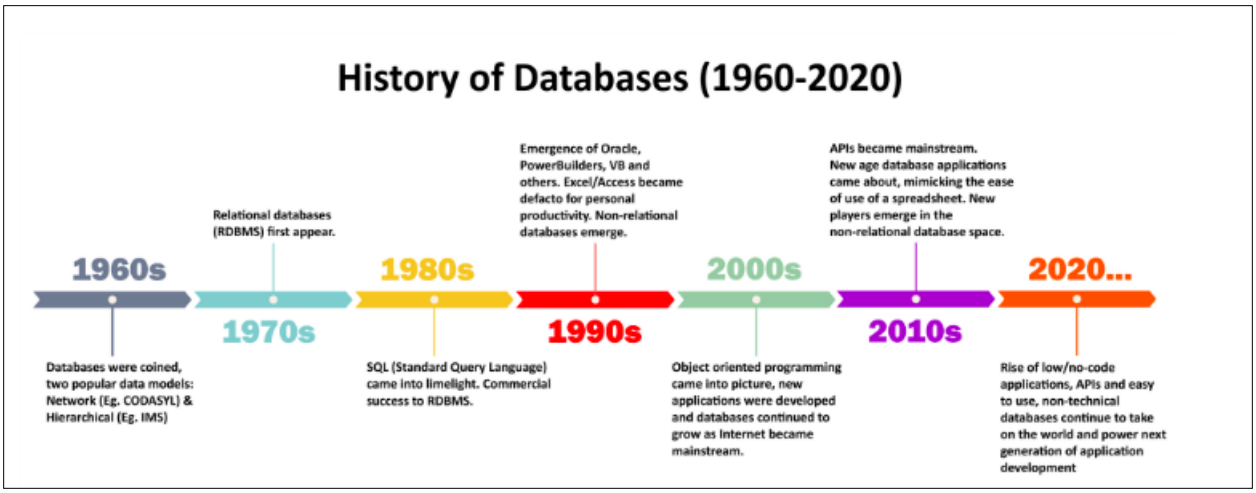
Selecting the right technology for data storage implies using the proper tool for a specific task. It can make the difference between a smooth, efficient process and one that is cumbersome and tedious. This chapter delves into the realm of databases, exploring two primary types, SQL and NoSQL, while also touching upon cloud solutions.

Introduction to Database Technologies

A database is fundamentally an organized data collection designed for easy storage, retrieval, and utilization. Databases serve as the superheroes of data management, offering an orderly repository for your information. Below is a brief passage that traces their evolution.

The Evolution of Databases

The journey of database technology has progressed significantly from the early days of conventional file systems. In the 1960s, the first Database Management System (DBMS) was introduced, the model of which was based on the concept of data hierarchy. Subsequently, flat-file systems emerged, storing data in singular files or tables. However, these systems presented unreliable security and high maintenance costs. Today, a variety of database technologies exist. Keep reading to know more.



(Image Source: [Stackby.com)](Stackby.com)

Diverse Database Types and Their Applications

Below are the different types of databases, each most suitable for specific purposes.

- **Relational Databases (SQL):** These databases employ SQL (Structured Query Language) to manage data like tables with rows and columns. They shine in intricate applications like those used in banking and finance.

- **NoSQL Databases:** NoSQL databases handle unstructured data without adhering to the traditional table format. They excel in domains like social media and online stores, encompassing document databases, graph databases, and even wide-column stores.

- **NewSQL Databases:** These databases blend aspects of relational and NoSQL databases, aiming for the flexibility offered by NoSQL while retaining key features of relational databases.

- **Graph Databases:** They are designed for managing highly interconnected data, such as social networks and recommendation engines.

- **In-memory Databases:** These databases store data in memory for lightning-fast performance, ideal for real-time analytics and trading.
- **Distributed Databases:** Distributed database systems are a type of database system that stores data across multiple physical locations or nodes in a network. This architecture is used to improve data availability, scalability, and fault tolerance in large-scale applications.

So, how do you decide which one to opt for? Read in the below sections.

Choosing the Right Database

Having clarity about your purpose and what you aim to achieve will help you choose the correct database type, as each type has different features and is most suitable for specific purposes, as listed below.

- Relational databases are the choice for intricate queries and transactions.
- NoSQL databases excel in scalability and availability.
- NewSQL databases strike a balance between scalability and ACID compliance.
- Graph databases are your ally when dealing with intricate relationships between data.
- In-memory databases provide speed for high-performance, low-latency requirements.
- Distributed systems provide high availability, scalability, data partitioning and fault tolerance.

SQL vs. NoSQL for Database Development and Management

SQL and NoSQL stand out from the others in the world of databases. SQL databases are structured like tables with fixed rules, while NoSQL databases can take on various forms. SQL adheres to specific rules for data interaction, whereas NoSQL is more flexible, employing diverse data models.

Pros and Cons of SQL Databases

Below are the pros and cons you generally come across with SQL databases:

Pros:

- **Strong data consistency:** SQL databases ensure that data remains accurate and reliable, which is vital for critical applications like banking systems.
- **Excellent for complex queries:** SQL databases can efficiently handle intricate queries, making them suitable for business intelligence and reporting.
- **Proven reliability:** SQL databases have a long track record of stability and performance, trusted by many industries.

Cons:

- **Limited flexibility:** SQL databases are less adaptable to changes in data structure, making them less suitable for unstructured or rapidly evolving data.
- **Scaling challenges:** Expanding SQL databases to handle high traffic can be complex and costly, often requiring significant hardware upgrades.
- **Licensing costs:** Some SQL database systems have expensive licensing fees, which can financially burden smaller businesses.

Pros and Cons of NoSQL Databases

NoSQL databases have the following benefits and drawbacks:

Pros:

- **Flexible data handling:** NoSQL databases can store diverse data types, making them ideal for applications with varying data structures.
- **Horizontal scalability:** NoSQL databases can effortlessly expand to handle increased data and user loads by adding more servers.

- **Real-time data processing:** NoSQL databases excel at handling dynamic, real-time data crucial for modern web and mobile applications.

Cons:

- **Inconsistent data:** The flexibility of NoSQL databases can lead to data inconsistencies if not appropriately managed, potentially affecting data quality.
- **Learning curve:** Developing and managing NoSQL databases may require new skills and tools, especially for teams accustomed to SQL databases.
- **Limited support for complex queries:** NoSQL databases may struggle with complex queries that SQL databases handle effortlessly, making them less suitable for specific analytical tasks.

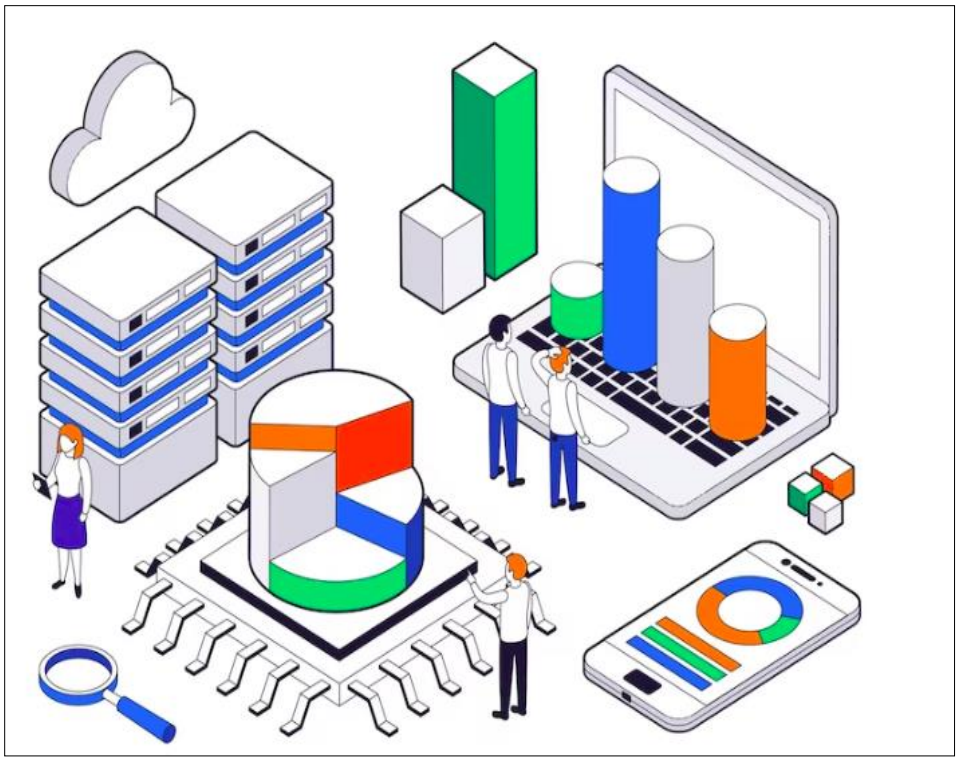
Picking the Right Database for Your Needs

Uncertain about which database to choose? Here's a straightforward rule to remember:

- Opt for SQL if you require a structured and well-organized database, particularly for dependable data storage.
- Select NoSQL when speed, scalability, and reliability are essential for web applications.

Storing Data in the Cloud

The cloud serves as an online repository for your data, accessible from anywhere with internet connectivity. This convenience is made possible by Internet technology. The below sections explore the advantages of cloud databases and critical players in the cloud industry.



(Image Source:Freepik.com)

The Advantages of Cloud Databases

Cloud databases act like digital genies, fulfilling the needs of businesses regardless of their size. Here's why they are advantageous:

- **Scalability:** Your database can expand or contract effortlessly in the cloud. Whether you're a small startup or a large corporation, you can adjust your resources as required. It's like a magical backpack that adjusts to accommodate its contents.
- **Cost-efficiency:** Say goodbye to the expense of buying hardware and hiring technical experts. With cloud databases, you only pay for the resources you use. It's like paying for the train until the place you get down, not for its entire journey.
- **Data Resilience:** Though chances of cyber-risks cannot be entirely ruled out, in the cloud, your data possesses the resilience of a phoenix—it can rise from the ashes. Cloud providers take extra precautions to safeguard your data and enable recovery from unforeseen mishaps. It's like having a backup expert at your side.

- **Accessibility:** The cloud system makes it possible for you to gain access to your data anywhere with an internet connection. Think of it as carrying a pocket-sized library holding all your vital information.

Major Cloud Players

The following giants are significant names in the cloud database industry.

1. **Amazon:** Amazon offers Amazon RDS for various data types and DynamoDB for high-speed databases.
2. **Microsoft:** Microsoft provides services like Azure SQL Database and Cosmos DB to cater to your database needs.
3. **Google:** Google offers Google Cloud SQL and Google Cloud Spanner, providing flexible and powerful database solutions.

Why Choose Cloud Storage?

Cloud storage offers numerous benefits, some of which are listed below:

- Scalability to match your needs, saving costs.
- No investment is required for expensive hardware or hiring technical experts.
- Data safety and resilience, enabling easy recovery from accidents.

With cloud databases, you're not merely storing data; instead, you're harnessing the potential of the digital world to enhance your business's efficiency, cost-effectiveness, and resilience. It's like having your digital toolkit at your fingertips.

Moving to the Cloud: Key Considerations

If you're contemplating a move to the cloud, keep these tips in mind:

- **Prioritize Security:** Safeguard your valuable data in the cloud by adhering to security best practices and encryption standards to protect it from unauthorized access.

- **Plan Ahead:** Transferring significant data volumes may take time, and during the transition, disruptions might occur. Plan your migration meticulously to minimize any potential hiccups.

Training Exercise

Exercise Title: "Database Decision-Making Challenge"
Description: Imagine you are building a new social media platform. Choose whether to use SQL or NoSQL for your user data storage and justify your choice based on the unique requirements of your social media applications. Consider factors like scalability, data consistency, and query complexity in your decision.

Chapter Questions:

1. What's the main difference between SQL and NoSQL databases?
2. When would you choose a relational database (SQL) over a non-relational database (NoSQL)?
3. What do scalability and elasticity mean for cloud databases?
4. What should you consider when migrating a database to the cloud?

Chapter Questions Answers:

1. SQL databases use structured schemas and tables, while NoSQL databases have flexible, schema-less data models.
2. You'd choose SQL for applications with complex queries, data consistency, and multi-row transactions.
3. Scalability and elasticity for cloud databases mean that they can adjust to changes in demand, making them cost-effective.

4. Consider data security, compliance, data transfer, and potential latency issues when moving data to the cloud.

References

1. Priya, U. (2023, July 1). Pros and Cons of Using SQL vs NoSQL Databases. Coding Ninjas. https://www.codingninjas.com/studio/library/pros-and-cons-of-using-sql-vs-nosql-databases
2. Gill, N. (2022, October 20). Types of Databases with Benefits and Use Cases. Xenonstack. https://www.xenonstack.com/blog/databases
3. Management Information Systems (MIS). What are the current trends and innovations in database management systems for MIS? Linkedin. https://www.linkedin.com/advice/0/what-current-trends-innovations-database
4. Smallcombe, M. (2023, January 12). SQL vs NoSQL: 5 critical differences. Integrate. https://www.integrate.io/blog/the-sql-vs-nosql-difference/
5. Coursera. (2023, June 16). SQL vs. NoSQL: The differences explained + when to use each. https://www.coursera.org/articles/nosql-vs-sql
6. Google Cloud. (n.d.). What is a cloud database? https://cloud.google.com/learn/what-is-a-cloud-database
7. Redhat. What is cloud infrastructure? (2019, May 18). https://www.redhat.com/en/topics/cloud-computing/what-is-cloud-infrastructure
8. Google Cloud. (n.d.). What is Cloud Storage? https://cloud.google.com/learn/what-is-cloud-storage
9. Yehuda, Y. (2022, September 15). Top 7 cloud databases. DBmaestro. https://www.dbmaestro.com/blog/database-automation/top-7-cloud-databases
10. Upwork. (2023, August 21). What Is Database Technology? 2023 Guide to Database Tech. https://www.upwork.com/resources/data-base-technology

Chapter 8 - Working with Cloud Infrastructure

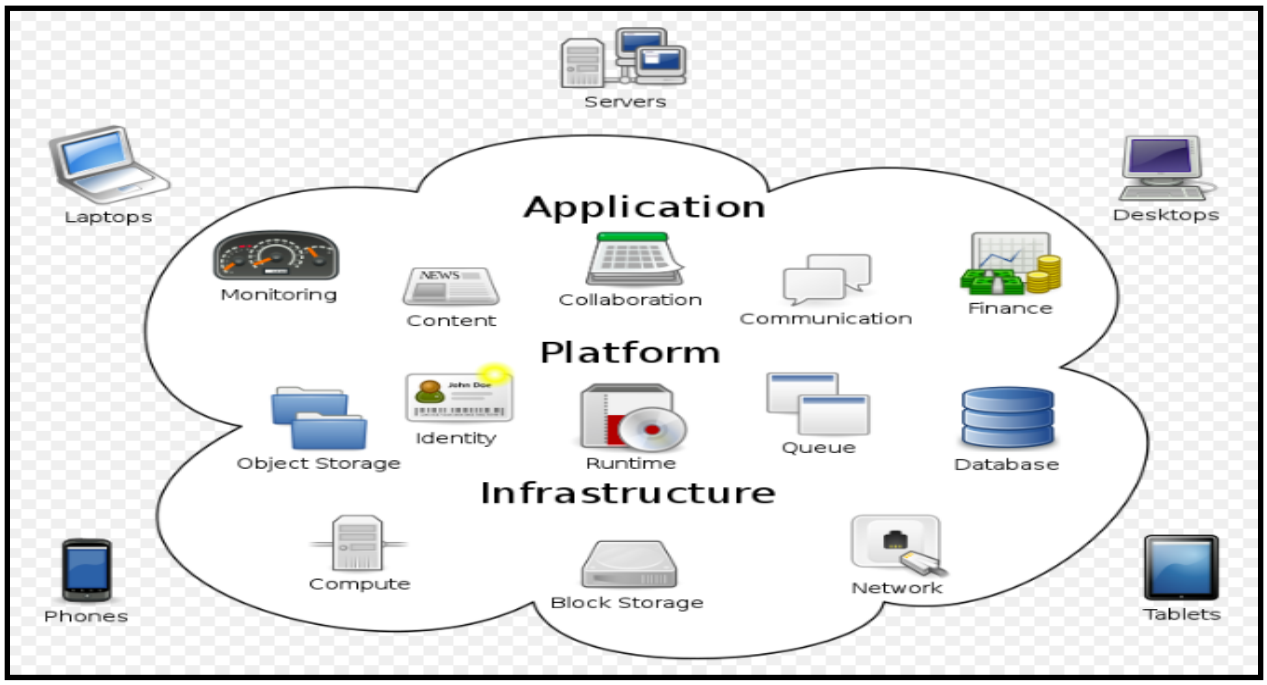
This chapter will take you through cloud computing, its types, and why it is crucial for DevOps.

You might have heard the term ‘cloud computing.’ But what is it? Cloud computing is one of today's cutting-edge technologies, making things much better and faster. The sections below will explore cloud computing, its different types, its relevance to DevOps, and more.

What is Cloud Computing?

Cloud computing encompasses all concepts that involve accessing computing resources and services remotely using the Internet. It is helpful because it makes things faster, more flexible, and cheaper. You don't have to manage the physical stuff yourself, and you only pay for what you use. The below sections will delve deeper into the subject.

Source: Wikipedia



The Fundamental Concept of Cloud Computing

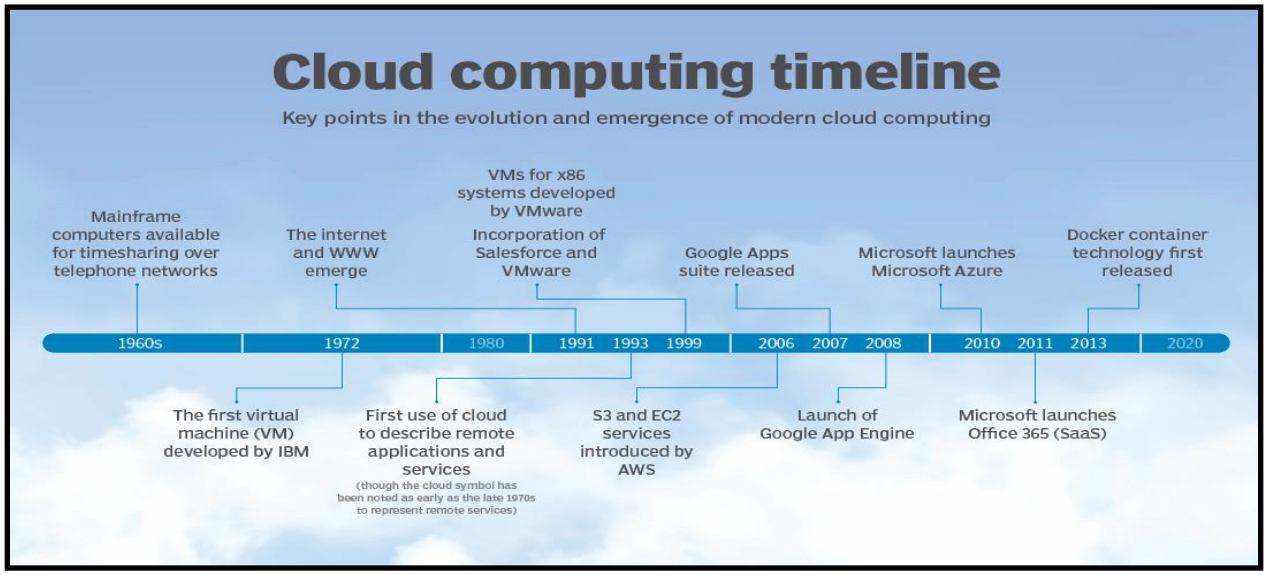
Cloud computing allows you to access and utilize servers, storage, applications, and other computing resources over the Internet on a pay-as-you-go basis. This method enables flexibility, scalability, and cost-efficiency in IT infrastructure management.

On-Demand Access to Computing Resources

With cloud computing, you can get computing infrastructure whenever needed. You only pay for what you use and can have a little or a lot of it. Regardless of the type of resource you receive, you don't need anything except a computer terminal and the Internet to receive the services.

Historical Evolution of Cloud Computing

People have been utilizing the cloud computing concept for a long time. However, the most advanced methodologies associated with its use, as seen currently, emerged in the 2000s. Cloud computing evolved from basic grid computing in the 1960s to virtualization and client-server models in the 1970s.



Source: [Techtarget]

The 2000s saw the emergence of utility and on-demand computing, paving the way for today's cloud services with scalable infrastructure and widespread adoption across industries.

Different Types of Cloud Computing Models

Cloud computing implies accessing computing infrastructure resources and services such as IT servers, storage, databases, networking, software, and much more remotely. Cloud services are divided into three types, each serving a different need.

- **IaaS (Infrastructure as a Service)**
  - IaaS provides access to computing resources like virtual machines, storage, and networking.
  - You can control and set up your resources per your requirements.
  - Prominent IaaS providers are Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform.

- **PaaS (Platform as a Service)**
  - PaaS helps you create and run the applications you need by accessing the necessary development platforms and tools remotely.
  - It includes tools for developing, testing, deploying, and renovating applications.
  - Popular PaaS providers are Heroku, Google App Engine, and Microsoft Azure App Service.

- **SaaS (Software as a Service)**
  - SaaS offers software applications hosted remotely and managed by a service provider.
  - Anyone can access such applications through a web browser or a mobile interface.

- Examples of SaaS providers are Salesforce, Dropbox, and Microsoft Office 365.

- **Unified Communications-as-a-Service (UCaaS)**
  - Amid the current crisis, there is a notable surge in the adoption of Unified Communications-as-a-Service (UCaaS). This service model offers users across the globe communications continuity and remote collaboration capabilities through cloud-based networks.
  - UCaaS stands out for its ability to deliver advanced security and reliability, creating a secure and virtualized cloud environment where remote workers can seamlessly conduct their tasks

Thus, cloud computing appears in different forms to meet various needs. IaaS provides you with infrastructure resources, PaaS is the arena for platforms to develop apps, and SaaS lets you use software applications remotely.

Source: Softweb Solution

Why the Cloud Matters for DevOps

The cloud is vital for DevOps due to multiple reasons. The below points show why it's essential for DevOps.

Scalability: Picture the cloud as a versatile toolbox. It can conjure up more tools as needed and stow them away when you finish your work. This feature is a boon for developers because they can swiftly access everything they need.

Collaboration: Envision the cloud as a massive chat room where developers, operations experts, and other team members converse and collaborate, resulting in superior products.

Cost Savings: The cloud is also a money-saver. Instead of purchasing and maintaining costly hardware and software, you can utilize cloud resources to the extent you need them and pay only for what you use. It's akin to borrowing rather than buying and can yield substantial long-term savings.

Faster Time-to-Market: Thanks to the cloud, you can swiftly develop, test, and launch applications. Its speed is to that of traditional systems what a race car is to a tortoise. Speed is invaluable because it enables you to unveil innovations sooner than your competitors.

Popular Examples of Cloud Computing

Cloud computing is a technological paradigm that lets people tap into valuable resources over the Internet. Here are some everyday examples.

Internet Email

Think Gmail or Outlook. They stash your emails on their information storage systems, not yours, allowing you to check your email from anywhere with internet access.

Online Document Sharing

Google Docs or Microsoft 365 lets you make, tweak, and share documents with others online. It's convenient for users who collaborate on projects from different locations.

Cloud Storage

Dropbox, Gmail, and Facebook let you save and access your data from anywhere. It is ideal for sharing and storing files, including important documents, images, and videos.

Streaming Apps

Netflix and others use cloud power to handle big crowds watching movies and videos online. It is helpful for businesses to broadcast high-quality videos to many viewers simultaneously.

Big Data, Backups, and More

Cloud facilities also help with big data crunching, secure data storage, and quick software testing. These services greatly benefit businesses dealing with tons of data and having speedy software needs.

Serverless Computing and CloudFormation

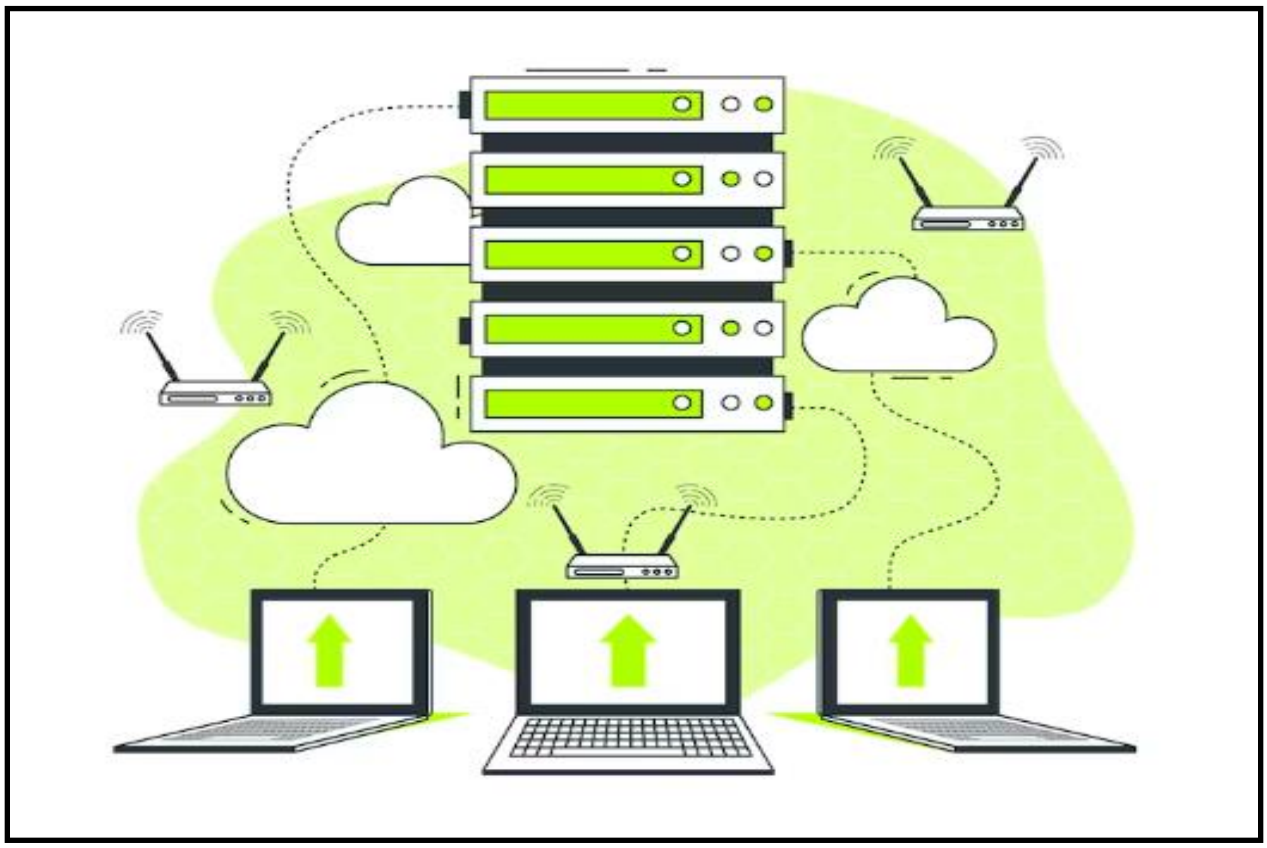
Serverless Computing offers back-end services remotely over the cloud and charges the user on an as-used basis. It's called serverless, as the user never has to bother about the existence of the remote servers while using the resources. The payment will be per usage and not dependent on the number of servers used or bandwidth. Serverless computing

simplifies cloud operations by letting the cloud provider handle infrastructure and resource allocation, including upscaling, automatically, reducing the workload and costs.

Key Benefits of Serverless Computing

Serverless computing offers various benefits, as listed below:

1. **Cost Savings:** Serverless computing can save you money by only charging for the resources you use, reducing idle time costs.
2. **Automatic Scaling:** It can automatically adjust resources to meet demand, ensuring your application performs well under varying workloads.
3. **Better Operational Focus:** You don't need to worry about managing servers and infrastructure, allowing you to focus on writing code that serves customers.



Source: Freepik

Harnessing the Power of Serverless Functions

Serverless functions are dynamic tools in the arsenal of application development, providing an amalgamation of simplicity and efficiency that's hard to beat.

Embracing Infrastructure as Code (IaC)

Infrastructure as Code (IaC) revolutionizes infrastructure management, replacing the tedious manual processes with a streamlined approach rooted in code.

AWS CloudFormation: Your IaC Wingman

Enter AWS CloudFormation, the champion of Infrastructure as Code tools. It's a game-changer that equips developers with the ability to define and orchestrate cloud resources through meticulously crafted templates.

The Advantages of IaC with AWS CloudFormation

Below are some of the advantages provided by IaC, particularly with CloudFormation:

Lightning-Fast Provisioning: IaC turbocharges the resource provisioning journey, slashing wait times and significantly enhancing efficiency.

Simplified Resource Management: The complexity of managing cloud resources dissipates with every change meticulously tracked through code.

Team Collaboration Redefined: Collaboration among teams gets a significant boost when everyone operates from a standard code-based template.

A Real-World Example: Crafting an Amazon S3 Bucket

It’s time for a glimpse of the natural world with a practical example of leveraging AWS CloudFormation to forge an Amazon S3 bucket:

- **Defining the Bucket's Core Attributes**

Within the confines of the CloudFormation template, you can specify crucial attributes for your S3 bucket. Whether it's the bucket's moniker, geographical placement, or the fine-grained access controls, you're in the driver's seat.

- **Version Control and Seamless Deployment**

  Templates can be elegantly controlled through versioning, providing a clear lineage of modifications. Whether you opt for the command-line prowess of the AWS CLI (Command Line Interface) or the more friendly AWS Management Console, the deployment process is a breeze.

This powerful synergy of serverless functions, Infrastructure as Code, AWS CloudFormation, and S3 bucket creation exemplifies the modern era of agile, efficient, and collaborative application development.

Exercise

Now, you can assess your proficiency in the above topics with a brief task. Picture yourself as a member of a team engaged in a fresh software venture, and you are tasked with determining the most suitable cloud service model (IaaS, PaaS, or SaaS) for the project's specific requirements.

Drawing from your knowledge, provide a concise summary outlining your selection and its rationale.

Chapter Questions:

Q1. What are the three primary categories of cloud computing services and their respective offerings?

A) IaaS offers application platforms, PaaS handles remote software, and SaaS delivers computing resources.

B) IaaS furnishes computing resources, PaaS provides a platform for applications, and SaaS is for remote software applications.

C) IaaS is for data storage, PaaS focuses on networking, and SaaS takes care of the hardware.
D) IaaS handles machine learning, PaaS is for big data, and SaaS provides website hosting.

Q2. How does cloud technology contribute to cost reduction within business operations?
A) It enables businesses to buy more hardware.
B) It encourages businesses to invest in data centers.
C) It allows businesses to pay for actual usage, reducing hardware costs and facilitating long-term savings.
D) It eliminates the need for software licenses.

Q3. What are the principal advantages of using cloud infrastructure in the DevOps sphere?
A) It offers only enhanced scalability.
B) It provides enhanced scalability, streamlined collaboration, and expedited application development and deployment.
C) It exclusively focuses on hardware virtualization.
D) It allows DevOps teams to skip the software testing phase.

Q4. What are everyday instances of cloud computing services that simplify daily lives?
A) Using a VPN, configuring firewalls, and setting up a local server.
B) Gmail, Google Docs, and Netflix, which allow for remote access and efficient resource sharing.
C) Windows OS, computer mouse, and keyboard.
D) Physical storage devices like USB drives and external hard disks.

Q5. What does the "Shared Responsibility Model" entail in cloud-based DevOps pipelines?
A) Only cloud providers are responsible for security, and customers need not worry about it.
B) Cloud providers secure the infrastructure, while customers are responsible for protecting their built components.
C) Both cloud providers and customers share equal responsibility for all aspects of the application, from code to infrastructure.
D) Customers are solely responsible for all security aspects, from the cloud infrastructure to the application code.

Chapter Questions Answers:

1. B
2. C
3. B
4. B
5. B

References

1. Chai, W., & Bigelow, S. (2022, November). What is a public cloud? Everything you need to know. TechTarget. https://www.techtarget.com/searchcloudcomputing/definition/cloud-computing
2. ThinkIT. (n.d.). Cloud computing examples. https://www.thinkitsolutions.com/cloud-computing-examples/
3. Courtemanche, M., Mell, E., Gillis, A. & Riley, C. (2021, December 22). What is DevOps? The ultimate guide. IT Operations; TechTarget. https://www.techtarget.com/searchitoperations/definition/DevOps
4. Daley, S. (2023, March 27). 25 cloud computing examples that keep the world at our fingertips. Built In. https://builtin.com/cloud-computing/cloud-computing-examples
5. Frankenfield, J. (2023, April 5). What is cloud computing? Pros and cons of different types of services. Investopedia. https://www.investopedia.com/terms/c/cloud-computing.asp
6. Linthicum, D. (n.d.). DevOps dictates new approach to cloud development. TechBeacon. https://techbeacon.com/app-dev-testing/devops-dictates-new-approach-cloud-development
7. Digital Cloud Training. (2023, May 10). 9 common uses of cloud computing. https://digitalcloud.training/9-common-uses-of-cloud-computing/

8. Peterson, R. (2023, September 1). What is Cloud Computing? Definition, Explain with Examples. Guru99. https://www.guru99.com/what-is-cloud-computing-with-example.html
9. Ranger, S. (2022, February 25). What is cloud computing? Everything you need to know about the cloud explained. ZDNET. https://www.zdnet.com/article/what-is-cloud-computing-everything-you-need-to-know-about-the-cloud/
10. Simplilearn. (2023, August 28). What is cloud computing and the top cloud technologies to look out. https://www.simplilearn.com/cloud-technologies-article
11. Singh, G. (2023, February 3). AWS serverless computing, benefits, architecture and use-cases. Xenonstack. https://www.xenonstack.com/blog/aws-serverless-computing/
12. Watts, S. (2019, April 15). The role of cloud in DevOps. BMC Blogs. https://www.bmc.com/blogs/devops-cloud/
13. Amazon. (N.d.). Serverless on AWS. https://aws.amazon.com/serverless/
14. Amazon. (N.d.). Understanding serverless architectures. https://docs.aws.amazon.com/whitepapers/latest/optimizing-enterprise-economics-with-serverless/understanding-serverless-architectures.html
15. https://www.softwebsolutions.com/resources/cloud-computing-service-models.html

Chapter 9 - Automating DevOps and Quality Assurance

This chapter will help you explore the Automating DevOps & Quality Assurance world. It involves teaching computers to help build software faster and better while keeping it safe.

Introduction

Automation is inevitable if one must derive the optimum output of DevOps. Similarly, quality assurance is vital for the results to be effective and efficient. DevOps quality requires recognizing code security as one of its most significant factors. The following sections explain how automation enhances software development, ensures security, and maintains code quality. Get ready to unlock the power of automation in your software projects.

Automation in DevOps

Automation is critical in streamlining DevOps processes, enabling teams to automate repetitive and manual tasks throughout the software development lifecycle.

DevOps automation offers several benefits, including:

- **Faster Deployments:** Automating repetitive tasks speeds up project delivery.
- **Reduced Errors:** Automation minimizes miscommunication-related errors, enhancing system reliability.
- **Improved Focus:** Developers can stay focused on the project as automation eliminates the need to manually trigger DevOps tasks.

Ensuring Security in Cloud Services

Cloud-based services have gained popularity in modern development but bring challenges and security concerns. Cloud security encompasses measures to safeguard

cloud infrastructure, applications, and data. Developers working with cloud services must know cloud security concepts and be well-versed in associated safe practices.

(Image Source: Freepik.com)

Here are the best practices for securing cloud-based DevOps pipelines:

- **Access Security**: Ensure robust access security with mechanisms like Multi-Factor Authentication (MFA). It limits access only to authorized personnel.
- **Access Privilege Management**: Regularly update user access privileges, granting only necessary permissions (for example, following the principle of least privilege.)
- **Data Encryption**: Encrypt data at rest and during transit to prevent unauthorized access.
- **Continuous Monitoring**: Continuously monitor cloud infrastructure for anomalies and potential security breaches.

- **Incident Response Plans**: Establish response plans for security breach incidents, encompassing data backup and disaster recovery for business continuity.
- **Compliance and Risk Management**: Stay compliant with cloud providers' specific requirements and maintain up-to-date security protocols for risk management.

DevOps Security Best Practices (SAST/DAST)

Static Application Security Testing (SAST) is like an X-ray for your software. It examines your code from the inside before it becomes a working program. This method is called "white-box testing." It's an excellent method to find hidden vulnerabilities.

Dynamic Application Security Testing (DAST) is like a security guard patrolling your software from the outside. It tests a running program, just like a "black box." It checks for weaknesses that threat actors could exploit.

Complementary Security

SAST and DAST aren't rivals; they're partners. They team up to form tools like Interactive Application Security Testing (IAST) to give your software a complete health check. IAST combines the strengths of both SAST and DAST.

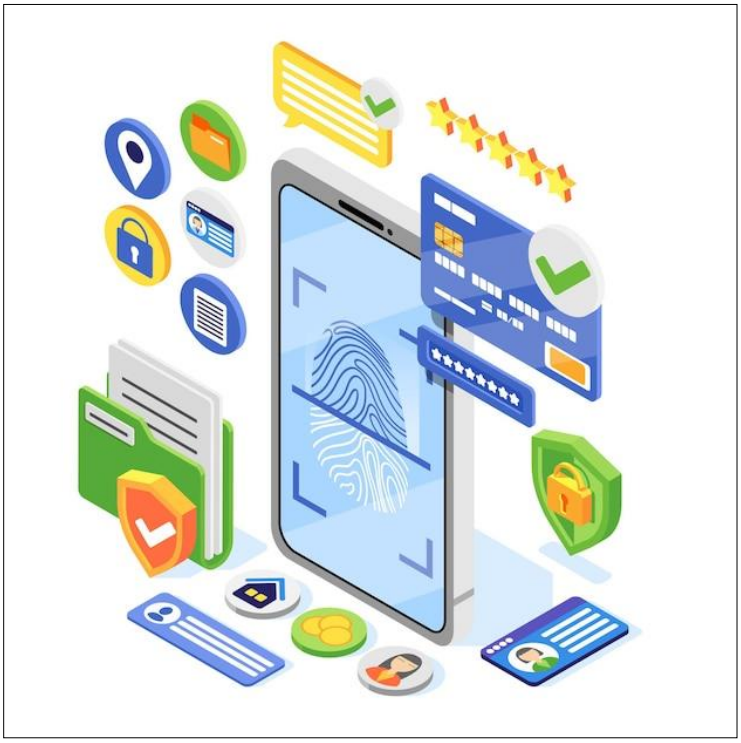
Benefits of SAST/DAST

SAST helps developers learn about security as they work. It offers real-time guidance, making finding and fixing vulnerabilities quickly easier. On the other hand, DAST goes beyond development and tests in the live environment. It simulates attacks, including the ones users have never encountered before.

SAST and DAST are vital for DevOps security. They detect issues early, reduce risks, and help developers by giving instant feedback. By using both, you'll build a robust defence system against security threats throughout your development cycle.

Authentication and Access Control

Meticulous access control management in DevOps environments remains paramount to safeguarding sensitive data and resources. Technologies like asymmetric cryptography and SSH play a vital role in ensuring secure access control, as explained below.



(Image Source: Freepik.com)

- **Asymmetric Cryptography**

  Asymmetric cryptography, often called public-key cryptography, is a robust encryption and decryption method. It finds application in software programs to establish secure connections across potentially insecure networks, such as internet browsers, or when validating digital signatures.

  The primary advantage of asymmetric cryptography lies in increased data security. This encryption process stands out for its ability to keep private keys confidential, reducing the risk of malicious actors intercepting them during transmission.

- **SSH and Access Tokens**

Secure Shell (SSH) is a protocol that leverages cryptography to facilitate secure communications between two untrusted hosts operating over an insecure network. It is a commonly employed solution for remote server access and network devices.

In parallel, access tokens serve as valuable credentials, enabling secure access to specific resources. Access tokens are pivotal in managing access control in DevOps settings, ensuring that only authorized users or applications can access designated resources. And that, too are given for a specific time limit, after that, the tokens get expire.

Effective Strategies for Access Control Management in DevOps Environments

Following are some of the best practices in access control management you can follow to ensure access control security:

- **Principle of Least Privilege:** Grant users only the lowest level of access essential for their job responsibilities, minimizing potential security pitfalls.
- **Implement MFA:** Elevate security by mandating two or more authentication methods for accessing sensitive data and resources.
- **Leverage Access Tokens:** Embrace access tokens as an efficient means to verify users and applications, allowing tailored access to specific resources.
- **Regular Policy Review and Audits:** Continuously assess and update access control policies to ensure their relevance and effectiveness.
- **Automation for Access Control Management:** Employ automation tools to streamline access control policies, promoting its consistent application across all environments.

Introduction to BDD and TDD

Behavior Driven Development (BDD) is an approach that emphasizes testing software behavior from an end-user's perspective. It seeks to ensure that software functions correctly and aligns with user expectations and requirements.

On the other hand, Test Driven Development (TDD) is a methodology that centers on testing smaller code units in isolation. Developers write tests before the actual code to ensure the code performs as intended.

Understanding BDD and TDD

While Test Driven Development concentrates on testing smaller code units in isolation, Behavior-Driven Development extends TDD by focusing on testing from an end-user's perspective. Both methods involve creating tests that help preemptively detect bugs.

Enhancing Code Quality and Reliability

The strengths of BDD and TDD in enhancing code quality and reliability are listed below:

- **Benefits of BDD and TDD:** Both BDD and TDD significantly enhance code quality and reliability by reducing the likelihood of coding errors.
- **Collaborative Nature of BDD:** BDD encourages collaboration among product managers, developers, and test engineers to generate tangible examples of desired functionality.
- **TDD for Solo Developers:** TDD empowers individual developers to write tests before code, ensuring the desired functionality and quality.

Practical Application of BDD and TDD

Both approaches incorporate tests into automated frameworks to proactively identify and prevent bugs. Their practical implications are as follows:

- **BDD in Action:** BDD involves cross-functional teams collaborating to create precise examples of desired functionality.
- **Step-by-Step TDD:** Developers create tests ahead of code in TDD, systematically ensuring that the code meets expectations.

Source Code Quality Check

Maintaining top-notch source code is vital for a robust and sustainable codebase. Employing automated code quality checks serves as a sentinel, detecting common issues and errors that enhance code quality and ease maintenance. The below section explains why a source code quality check is inevitable.

(Image Source: Freepik.com)

Significance of a High-Quality Source Code

High-quality source code bears immense significance due to the following reasons:

1. **Enhanced Maintenance:** It simplifies codebase upkeep and facilitates improvements.
2. **Error Mitigation:** It lowers the likelihood of bugs and errors, bolstering reliability.
3. **Optimized Performance:** It elevates software performance, resulting in smoother operation.
4. **Clarity and Comprehension:** It renders code more readable and understandable, promoting collaboration.

Tools and Strategies for Automated Code Quality Checks

Widely used tools and strategies for automated code quality assessments include:

1. **Linting:** This technique scrutinizes code to unveil potential errors and bugs.
2. **Code Reviews:** Involving peer review, this practice identifies issues and recommends enhancements.
3. **Continuous Integration:** It automates building, testing, and deployment, ensuring code integrity.

The Benefits of Automated Checks

Automated checks play a crucial role in ensuring a stable and maintainable codebase by:

1. **Enforcing Consistency:** They maintain code consistency and adherence to best practices.
2. **Detecting Issues Early:** Problems are unearthed early in development, saving time and resources.
3. **Driving Sustainable Progress:** Codebase enhancement and upkeep are facilitated for long periods.

Automated code quality assessment processes are indispensable in contemporary software development, as they empower developers to produce code of the highest calibre.

Chapter Questions:

Q1. What is the primary advantage of using both Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST) in DevOps?

A) It reduces manual code review efforts.
B) It helps in detecting security issues early and in live environments.
C) It only focuses on real-time testing.
D) It eliminates the need for compliance checks.

Q2. How does Behavior-Driven Development (BDD) differ from Test-Driven Development (TDD)?

A) BDD emphasizes automation, while TDD does not.
B) BDD focuses on testing smaller code units in isolation.
C) BDD focuses on testing from an end-user's perspective.
D) TDD concentrates on application functionality only.

Q3. Name one benefit of maintaining high-quality source code in software development.

A) High-quality source code guarantees no future bugs.
B) High-quality source code simplifies codebase upkeep and enhances reliability.
C) High-quality source code replaces the need for testing.
D) High-quality source code ensures immediate market success.

Q4. What is the purpose of automation in DevOps processes?

A) To make developers' lives more complicated.
B) To streamline tasks and speed up project delivery.
C) To completely replace human input in development.
D) To focus solely on coding standards.

Q5. What does the "Shared Responsibility Model" entail in cloud-based DevOps pipelines?

A) The cloud provider is responsible for all security aspects.
B) Both the customer and cloud provider share equal responsibilities.
C) The customer is solely responsible for all aspects of security.
D) The cloud provider secures the infrastructure while customers protect their built components.

Chapter Questions Answers:

1. B
2. C
3. B
4. B
5. D

References

1. Baig, A. (2023, May 24). Cloud data security: Challenges and best practices. Data Science Central; TechTarget. https://www.datasciencecentral.com/cloud-data-security-challenges-and-best-practices/
2. Brush, K., Rosencrance, L., & Cobb, M. (2021, September). Asymmetric Cryptography (Public key cryptography). Security; TechTarget. https://www.techtarget.com/searchsecurity/definition/asymmetric-cryptography
3. Ekran. (2023, March 17). Cloud infrastructure security: 7 best practices to Secure Your Sensitive Data. https://www.ekransystem.com/en/blog/cloud-infrastructure-security
4. Hackerone. (n.d.). Cloud security: Challenges, solutions, and best practices. https://www.hackerone.com/knowledge-center/cloud-security-challenges-solutions-and-best-practices
5. Cloudian. (n.d.). Data Protection in the Cloud. https://cloudian.com/guides/data-protection/data-protection-in-the-cloud-challenges-and-best-practices/amp/
6. Fernandes, C. (2022, August 24). DevOps automation: Best practices and benefits. Sumo Logic. https://www.sumologic.com/blog/devops-automation-best-practices-benefits/
7. Stevens, P. (2019, March 07). Understanding the differences between BDD & TDD. Cucumber. https://cucumber.io/blog/bdd/bdd-vs-tdd/
8. Martin, M. (2023, September 2). 15 BEST Code Review Tools for code quality analysis [2023]. Guru99. https://www.guru99.com/code-review-tools.html

9. Mehta, M. (2020, November 3). What is asymmetric encryption & how does it work? InfoSec Insights. https://sectigostore.com/blog/what-is-asymmetric-encryption-how-does-it-work/
10. West Agile Labs. (n.d.). Best practices and benefits of automating DevOps. https://www.westagilelabs.com/blog/best-practices-and-benefits-of-automating-devops/
11. Schmitt, J. (2023, May 10). SAST vs DAST: What they are and when to use them. CircleCI. https://circleci.com/blog/sast-vs-dast-when-to-use-them/
12. Owasp. (n.d.). Source Code Analysis Tools. https://owasp.org/www-community/Source_Code_Analysis_Tools
13. Imperva. (n.d.). SAST, DAST, IAST and RASP. https://www.imperva.com/learn/application-security/sast-iast-dast/
14. Katalon. (n.d.). TDD vs BDD: Full comparison. https://katalon.com/resources-center/blog/tdd-vs-bdd
15. Dowling, L. (2022, May 04). The 10 best code quality tools. LinearB. https://linearb.io/blog/the-best-code-quality-tools
16. Unadkat, J. (2023, June 15). TDD vs BDD vs ATDD: Key differences. BrowserStack. https://www.browserstack.com/guide/tdd-vs-bdd-vs-atdd
17. NetApp. (n.d.). What is DevOps? https://www.netapp.com/devops-solutions/what-is-devops/
18. Redhat. (2019, January 08). What is DevOps automation? https://www.redhat.com/en/topics/automation/what-is-devops-automation
19. Microfocus. (n.d.). What is Static Application Security Testing (SAST)? https://www.microfocus.com/en-us/what-is/sast
20. CISA. (2018, July). SECURING HIGH VALUE ASSETS. https://www.cisa.gov/sites/default/files/publications/Securing%20High%20Value%20Assets_Version%201.1_July%202018_508c.pdf
21. Softwaretestinghelp. (2023, June 26). TDD Vs BDD – Analyze The Differences With Examples. https://www.softwaretestinghelp.com/tdd-vs-bdd/
22. Softwaretestinghelp. (2023, July 24). 12 BEST Code Quality Tools For Error Free Coding In 2023. https://www.softwaretestinghelp.com/code-quality-tools/

Chapter 10 - Understanding DevOps Assembly Lines

DevOps is a crucial part of modern software creation. This chapter will help you explore a new and powerful way of handling DevOps called 'DevOps Assembly Lines.'

Introduction

DevOps Assembly Lines are like supercharged puzzle-solving machines that help developers build software faster and better. It can be considered a stage higher than CI. While CI oversees a development team, an Assembly Line helps integrate many teams for a common goal. The following sections provide insight into CI/CD pipelines and Assembly Lines. You will also find a comparison of these two concepts, the advantages of Assembly Lines, and when to use CI pipelines and when Assembly Lines.

(Image Source: Freepik.com)

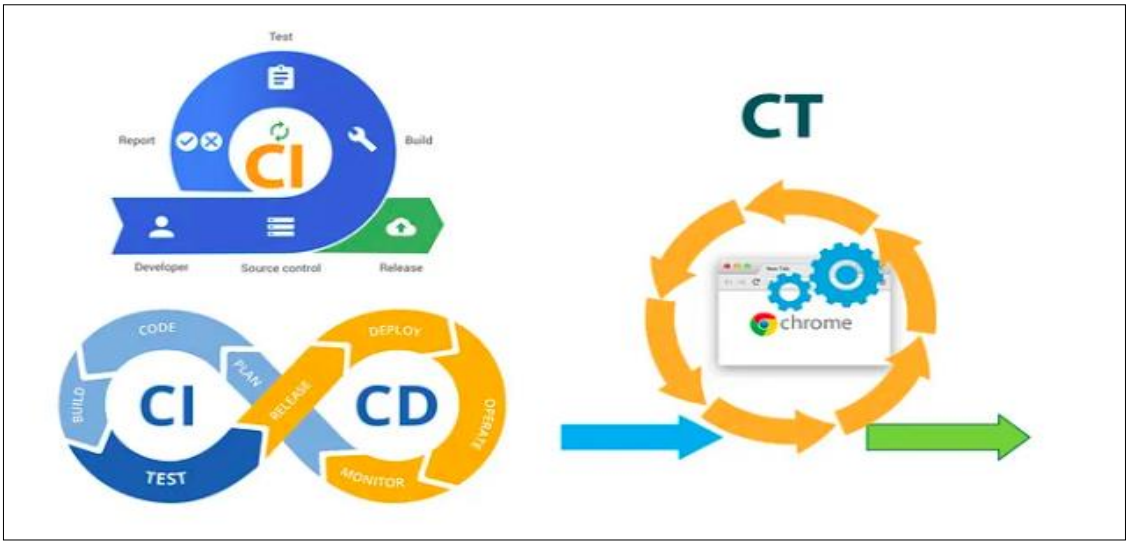
What is a CI/CD Pipeline?

A CI/CD pipeline is like a handy conveyor belt for software creation. It's a set of automatic steps that make creating, testing, and releasing software easy. It helps professionals in a software development team to make things better constantly.

Key Constituents of a Typical CI/CD Pipeline

Here are the main constituent parts of a CI/CD Pipeline:

- **Source Control Management:** This component is the central repository where all your code is meticulously stored and organized.
- **Continuous Integration:** Like a chef's workstation, it functions as a hub where all your code seamlessly converges and harmonizes.
- **Continuous Delivery:** This phase ensures that your software is impeccably prepared and almost ready for deployment, granting you one final opportunity to inspect it before unveiling it to the audience.
- **Continuous Deployment:** In this stage, your software is promptly delivered to customers without further scrutiny.
- **Monitoring and Feedback:** This step observes your software in the real-world environment, offering insights into its performance and user satisfaction.



(Image Source: Pagerduty & Techahead)

The CI/CD Pipeline Workflow

A typical CI/CD pipeline follows these steps:

1. **Code Commit:** Developers upload their code changes to a particular niche called a Version Control System (VCS), much like saving your work.
2. **Build:** The pipeline takes the code and turns it into something that can run or be used.
3. **Automated Testing:** The pipeline runs automatic tests to ensure the code changes won't cause problems or disrupt what already works.
4. **Code Integration:** The pipeline combines the new and existing code without complicating things.
5. **Artifact Deployment:** Finally, the pipeline puts the new version where it needs to go, like a stage.

Limitations of CI/CD Pipelines

CI/CD pipelines are not free from limitations, as evident from the below statements:

- Traditional CI/CD pipelines may struggle with complex applications and diverse environments.
- Implementing CI/CD pipelines can pose challenges, such as integrating them into existing projects, ensuring security and compliance, handling different versions, and scaling pipelines.
- Challenges during CI/CD implementation also include a need for more expertise, inadequate training, and resistance from team members and project stakeholders.

A more comprehensive DevOps approach often becomes necessary to address these limitations of traditional CI/CD pipelines, which takes you to the concept of DevOps Assembly Lines.

What are DevOps Assembly Lines?

DevOps Assembly Lines are like all-in-one automation systems combining different teams and tasks. You may think of it as a supercomputer that coordinates everyone's work.

Assembly Lines can handle complex projects and multiple teams that may be challenging for a standalone CI system. They help integrate several verticals, including CI, infrastructure, and security. Assembly Lines use straightforward language to plan and automate work, making things smoother and reducing human effort.

Key Components of a DevOps Assembly Line

The vital components of DevOps Assembly Lines are designed to make things easier and more efficient for you and your teams. Here's what they include:

- **Infrastructure as Code (IaC):** This paradigm helps you to set up and manage your digital infrastructure automatically—no more manual work for operations teams.
- **Automated Testing and Deployment:** Your code gets tested and deployed automatically, ensuring it is ready for the production sphere.
- **Security and Compliance Checks:** These steps are built into the process, ensuring your software stays safe and follows the rules. This section is usually called DevSecOps.

DevOps Assembly Lines aim to eliminate extra work and create smooth workflows with these constituent parts.

Advantages of Using DevOps Assembly Lines

DevOps Assembly Lines are designed to simplify and automate various tasks in your development process. They bring several benefits to your DevOps practices:

- **Improved Scalability, Consistency, and Traceability:** You can quickly expand and manage your projects while ensuring uniformity and keeping track of changes.
- **Enhanced Collaboration:** Your development, operations, and security teams can collaborate seamlessly, promoting better teamwork.

- **Accurate Continuous Deployment:** DevOps Assembly Lines ensure precise and reliable deployment processes with compatibility across different systems.
- **Powerful Visibility:** You gain in-depth insight into your workflows, helping you understand and manage complex processes effectively.
- **Efficient Onboarding and Scaling:** With an "as-code" mindset, you can swiftly bring new team members on board and scale your operations.

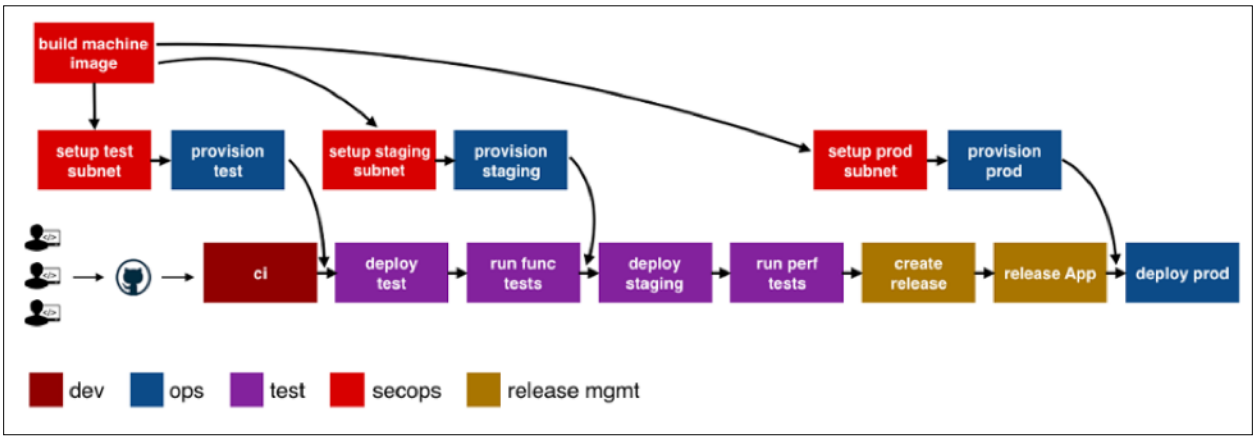
CI Pipelines vs. DevOps Assembly Lines: A Side-by-Side Comparison

Here's a straightforward comparison to help you understand the key differences between CI pipelines and Assembly Lines:

- **CI Pipelines**
  - **Focus**: CI pipelines focus on code integration and basic automation, ensuring that code changes come together smoothly.
  - **Scope**: They are typically centred around developer-centric tasks, making them well-suited for smaller, code-centric workflows.
  - **Structure**: CI pipelines consist of a series of automated steps, each tailored to specific phases of the software development cycle.
  - **Visual**: Think of CI pipelines as individual boxes in an Assembly Line, each dedicated to different tasks.

- **Assembly Lines**
  - **Comprehensive Approach**: DevOps Assembly Lines take a more comprehensive approach, handling code, infrastructure, and security.
  - **Complexity**: They are like an intricate web of multiple pipelines, managing various activities needed for an application's development and deployment.

- **Interconnected**: Imagine an intricate network of several CI pipelines, all interconnected to create a more streamlined and holistic process.
- **Management**: Managing Assembly Lines can be challenging due to their complexity. Many organizations use a managed services portal to keep things running smoothly across their DevOps Assembly Lines.



(Image Source: [Devops.com)](Devops.com)

In short, CI pipelines are a more efficient, modernized version of traditional CI. On the other hand, DevOps Assembly Lines offer a more comprehensive approach, covering code, infrastructure, and security.

When to Use CI Pipelines

You can make good use of CI/CD pipelines in different situations, such as:

- When dealing with straightforward applications or projects with only a few interconnections.
- In the beginning stages of developing your software.

These pipelines help automate your software's delivery, from the source code to the final product. They also ensure your technology setup stays consistent and dependable across various settings. Moreover, they are excellent at spotting and resolving problems early during development, lowering the chances of encountering errors and conflicts.

When to Choose DevOps Assembly Lines

DevOps Assembly Lines can prove a better choice if you're working on a big, complex project or a business-level application that needs to grow as your user base does. They're also perfect for projects with strict safety and compliance rules.

DevOps Assembly Lines are like your personal team of superheroes in that they take care of numerous diverse tasks that different groups in your software development project need to do. Here's how they help:

- **Developers:** They make sure your code fits together smoothly, like the pieces of a jigsaw puzzle. Thus, when building an extensive application, everything stays neat.
- **Operators:** They manage the behind-the-scenes activities, like what occurs behind the stage in a big show. They ensure everything is set up and working smoothly so your application can run without a hitch.
- **Testing Team:** Think of them as the quality control team. They have scripts that check if your application works as intended. If something is amiss, they'll let you know.
- **SecOps (Security Operations):** They're like the security guards of your application. They ensure no sneaky bugs or malicious actors get in and cause trouble.
- **Continuous Integration (CI):** They are like the conductor of an orchestra. It makes sure all the different teams work together in harmony.

The best part is that DevOps Assembly Lines put all these teams and tools on autopilot. They ensure the smooth flow of every process so you can deliver your application to your users without any hiccups.

Training Exercise

Imagine you are part of a software development team working on a complex project. Take a moment to outline the key steps you would follow in implementing a DevOps Assembly Line for your project. Consider how it helps streamline your workflow, enhance collaboration, and address challenges your team might face. Sketch a basic diagram to illustrate the flow of your Assembly Line.

Chapter Questions:

Question 1: What is the primary purpose of a CI/CD pipeline?

A. To automate social media posts

B. To manage employee payroll

C. To automate software creation, testing, and release

D. To assist in customer support

Question 2: In which scenarios would you prefer DevOps Assembly Lines over CI Pipelines?

A. For writing marketing content

B. For small, short-term projects

C. For large, complex projects or applications with strict security and compliance requirements

D. For maintaining physical hardware

Question 3: Name one advantage of using DevOps Assembly Lines for software development.

A. To lower the operational costs

B. To enhance collaboration among development, operations, and security teams

C. To improve the quality of office coffee

D. To handle customer support more efficiently

Question 4: What is the main benefit of incorporating "Infrastructure as Code" (IaC) in DevOps Assembly Lines?

A. To streamline and automate the digital infrastructure's setup and management

B. To increase sales and revenue

C. To enhance brand image

D. To improve the UI/UX of the software

Chapter Questions Answers:

1. C
2. C
3. B
4. A

1.6 References

1. Anastasov, M. (2022, July 15). CI/CD pipeline: A gentle introduction. Semaphore. https://semaphoreci.com/blog/cicd-pipeline
2. Active Batch. (2023, August 25). Creating A CI/CD pipeline for faster, more reliable delivery. https://www.advsyscon.com/blog/ci-cd-pipeline/
3. Ganguly, S. (2022, July 19). 7 CI/CD Challenges & their Must-Know Solutions. BrowserStack. https://www.browserstack.com/guide/ci-cd-challenges-and-solutions
4. Gill, N. (2023, May 10). DevOps assembly lines and Continuous Integration Pipelines. Xenonstack. https://www.xenonstack.com/blog/devops-assembly-line
5. Hamilton, T. (2023, July 22). CI/CD pipeline: Learn with example. Guru99. https://www.guru99.com/ci-cd-pipeline.html
6. MSys Technologies. (2023, October 9). Making DevOps sensible with assembly lines. https://www.msystechnologies.com/blog/making-devops-sensible-with-assembly-lines/

7. Longbottom, C. (2021, April 22). The pros and cons of CI/CD pipelines. Software Quality; TechTarget. https://www.techtarget.com/searchsoftwarequality/tip/The-pros-and-cons-of-CI-CD-pipelines
8. Roper, J. (2023, September 4). CI/CD pipeline: Everything you need to know. Spacelift. https://spacelift.io/blog/ci-cd-pipeline
9. Sacolick, I. (2022, April 15). What is CI/CD? Continuous integration and continuous delivery explained. InfoWorld. https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html
10. Sahasrabudh, M. (2017, September 8). Differentiating between CI pipelines and DevOps assembly lines. DevOps. https://devops.com/differentiating-ci-pipelines-devops-assembly-lines/
11. Redhat. (2022, May 11). What is a CI/CD pipeline? https://www.redhat.com/en/topics/devops/what-cicd-pipeline
12. GitLab. (n.d.). What is a CI/CD pipeline? https://about.gitlab.com/topics/ci-cd/cicd-pipeline/
13. GitLab. (n.d.). What is CI/CD? https://about.gitlab.com/topics/ci-cd/
14. Synopsys. (n.d.). CI CD. https://www.synopsys.com/glossary/what-is-cicd.html
15. Image Credit: https://www.pagerduty.com/resources/learn/what-is-continuous-integration/; https://www.techaheadcorp.com/blog/how-ci-cd-save-app-development-time/

Chapter 11 - Commonly Used DevOps Tech Explained

This chapter explores innovative DevOps technologies like remote server automation, infrastructure automation, containerization, and more.

Introduction

Previous chapters discussed the concept of DevOps and the role automation plays in enhancing the overall efficiencies of systems. Besides, you learned cloud computing aspects like version control, scaling, and hosting automation software builds. You also know the different front and back-end development tools and the basics of web servers and hosting. This chapter discusses the concept of remote server automation, infrastructure configuration, provisioning and monitoring tools, and application containerization.

Remote Server Automation

You have seen how DevOps perfectly coordinates development and operations teams to ensure the project's success. Automation expedites the process and eliminates manual intervention at all stages. Consequently, it improves the accuracy and efficiency of the project. Remote Server Automation is an extension of the automation process. It is a crucial aspect of software development involving automating tasks and processes on remote servers.

Remote Server Automation includes tasks like deployment, monitoring, maintenance, and troubleshooting of various applications and services on remote servers. The process streamlines workflows, reduces human errors, and improves overall efficiency. Developers can automate repetitive tasks using Remote Server Automation, ensuring consistent server configurations and quicker issue resolution. It enables end-users to benefit from reliable services and applications.

The Connection Between Remote Server Automation and DevOps

Remote server automation enables repetitive and manual DevOps tasks to be conducted with no human interaction. You can apply automation throughout the DevOps lifecycle, spanning design and development, software deployment, release, and monitoring. The objective of DevOps automation is to streamline the DevOps lifecycle by reducing manual workload.

Automation eliminates the requirement of large teams, reduces human errors significantly, increases team productivity, and creates a fast-moving DevOps lifecycle. Remote server automation facilitates CI and CD, resulting in smoother and more frequent deployments with less time-to-market, which helps teams gain an edge over their business competitors.

Remote Server Automation provides better consistency across repetitive tasks by configuring an automation tool and eliminating the threat of human error. It increases the team's speed from code integration to application deployment, which ensures better stability and reliability due to improved collaboration.

Introduction to Infrastructure Configuration, Provisioning, and Monitoring Tools

Remote server automation removes the manual error component from any project. However, monitoring the overall health and performance of back-end components in cloud, on-premise, and hybrid environments is critical to the entire process. That brings infrastructure configuration, provisioning, and monitoring tools to the picture.

Consider an example where one of the packages you use in your team's application develops a significant security flaw. Naturally, you rush to apply a patch program to address the issue because protecting your customers' data is paramount. A couple of days later, your team finishes testing a new auto-complete feature. While deploying the feature, you find that it does not work correctly. So, did you miss something crucial? Yes,

you should have updated the package on staging servers, leading to non-synchronization between staging and production.

But do you know such instances are preventable if you have good infrastructure configuration practices? The below section explores the concept of infrastructure configuration.

What Is Infrastructure Configuration?

Provisioning and configuring the infrastructure is crucial before deploying any application. Provisioning entails setting up the servers, network equipment, and other infrastructure. You can also call it spinning up a virtual machine. Besides server provisioning, there are other types, including network, user, and service provisioning. Once you have provisioned the infrastructure, you can configure it.

Infrastructure configuration involves customizing provisioned resources. The tasks include installing dependencies on a server, setting up logging, updating to a specific Linux distribution, and creating database configuration files.

While provisioning is the initial step in a process, infrastructure configuration is continuous because of requirements like updating the software and changing passwords. The infrastructure needs continuous scaling as the organization grows. Thus, every change to the infrastructure qualifies as infrastructure configuration.

Issues with Infrastructure Configuration

While infrastructure configuration is a critical component of every process, it has its challenges, especially with manual configurations. They include cost, scalability issues, poor visibility, and configuration drift. However, technologies have improved significantly to overcome these issues. As a result, modern infrastructure configuration is automated.

Initially, shell scripts were used for configuring servers. Subsequently, tools like ‘csshx’ allowed the passing of commands to multiple servers simultaneously. Eventually, it led to Infrastructure as Code (IaC).

IaC is the process of defining infrastructure in configuration files stored and tracked in version control. It led to applying the best development practices to infrastructure, like

- Configuration files must be version-controlled.
- They must be the source of truth for the infrastructure state.
- Any change to configuration files must be tested before deployment.
- Provisioning and configuration processes must be automated.

Benefits of Infrastructure as Code

IaC is an automated process and, hence, more beneficial than manual configurations in the following ways:

- IaC is faster, making it easier to automate repetitive tasks.
- IaC leads to reliable configurations. Hence, it is more consistent.
- IaC offers more visibility and makes it easy to tell when and where the changes are made.
- IaC is cost-effective because it reduces human labour hours spent configuring and troubleshooting infrastructure.

Infrastructure Configuration, Provisioning, and Monitoring Tools

Infrastructure configuration, provisioning, and monitoring tools are critical to managing and maintaining a tech stack because they help DevOps engineers track the health and performance of back-end components in the cloud, on-premises, and hybrid environments.

- Terraform and CloudFormation are two examples of infrastructure provisioning or orchestration tools used to provision the infrastructure, such as servers, databases, load balancers, and networking configuration.

- Ansible, Puppet, and Saltstack are configuration management tools used to manage the configurations of these components.

- Infrastructure monitoring tracks the performance, availability, and resource utilization of hosts, containers, and other back-end components. Generally, engineers install agents (software) on their hosts to collect infrastructure metrics from them and send the collected data to a monitoring platform for analysis and visualization.

The 20 Top DevOps Tools for Infrastructure Automation

You can classify the top 20 DevOps tools for infrastructure automation into seven categories as below.

Infrastructure Provisioning	● **Terraform** - Supports all public and private cloud infrastructure. ● **Pulumi** - Supports multiple programming languages like Python, Go, C#, and JavaScript
Configuration Management	● **Ansible** - Also works as an orchestration tool and for cloud provisioning. ● **Chef** – A developer-centric ruby-based configuration management tool ● **Puppet** – A Ruby-based configuration tool developed for system administrators ● **Saltstack** - Supports remote execution of commands

	• **Helm** - An ideal configuration and package manager for Kubernetes
Continuous Integration/Deployment	• **Jenkins** - A Java-based CI tool for faster application delivery • **GitHub Actions** - For setting up CI pipelines • **Kubernetes Operators** - One of the best container orchestration tools
Image Management	• **Packer** - Supports private cloud and public cloud VM (Virtual Machine) image management • **Docker** - Good for creating isolated environments for applications called containers • **Podman** - An open-source container management tool
Infrastructure Development	• **Vagrant** - Excellent for configuring virtual machines • **Minikube** - Good for developing and testing Kubernetes
Config/Secret Management	• **Hashicorp Consul** - Used for service discovery processes • **etcd** - A key component of Kubernetes architecture • **Hashicorp Vault** - An open-source tool for storing and retrieving secret data
Logging and Monitoring	• **Prometheus** - An open-source monitoring system built for modern application monitoring • **Alert Manager** - Manages alerting setups for monitoring metrics

	• **Sensu** - An open-source Ruby-based monitoring tool built for cloud environments.

Virtualization Challenges and Containerization

Virtualization is the growing use of virtual machines (VMs) in the cloud. While virtualization has its benefits, its unique challenges, including the following, make these environments somewhat inefficient.

- **Inconsistent Environment** – It deploys apps and packages to virtual environments.
- **Dependency on OS** - Deployed apps run on compatible OS alone.
- **Isolation Level** – It cannot provide instant sandbox above OS level.
- **Compute Consumption Granularity** – It cannot deploy multiple replicated applications.
- **Patching Images in a Production-Grade Environment** - Blue-green and canary deployments are inflexible at the cluster level and challenging to manage across multiple regions.

Containerization is the best solution to overcome these virtualization challenges because it is more flexible and granular. It has a higher efficiency and is considered a natural evolution of virtualization.

What is Application Containerization?

People working in the cloud computing environment must have heard the term ‘containerization.’ It is considered one of the most preferred processes for modernizing legacy systems by creating new, scalable, cloud-native applications.

Containerization is a form of OS virtualization that allows running applications in isolated user spaces called containers. An application container presents a fully packaged and portable computing environment. Some salient features of containerization are:

- An application container has everything an app requires to run. Everything, including its configuration files, libraries, binaries, and dependencies, is encapsulated and isolated in a container.

- It is similar to a lightweight virtual machine because containerizing an app abstracts the container from the host OS with limited access to underlying resources.

- Containerized applications can run on different infrastructures, like within a VM, in the cloud, or on 'bare metal’ servers, without refactoring it for each environment.

Containerization is beneficial because there are fewer overheads during startup. Secondly, there is no need to set up a separate guest OS for each app because they share a single OS kernel. Therefore, software developers find application containerization highly useful for packaging individual microservices.

What Does Containerization Do?

In a traditional environment, software developers develop code in specific computing environments. However, errors and bugs can creep in when transferring it to a new location. In contrast, containerization allows quicker and safer options for creating and deploying apps.

Containerization allows bundling of the supplication codes with its related configuration files, dependencies, and libraries. The single package can be abstracted away from its host OS, allowing it to stand alone and be portable. Thus, it can run on any platform or cloud without any issues.

In simple words, containerization allows software developers to write apps once and run them everywhere. This portability is critical to the development process and vendor compatibility. Other benefits include ease of management, security, and fault isolation.

Containers encapsulate applications as an executable software package, including its application code and related config files, dependencies, and libraries required for its functioning. These apps are isolated because they do not bundle within an OS. Instead, they have an open-source runtime engine on the host's OS. The primary advantage of containerization is that isolating it from the OS reduces the chances of malicious code in one container invading the host OS. Besides, isolation makes them portable and easily transportable from one platform to another.

Advantages of Containerization

Containerization has numerous advantages, which can be summarized as below:

- **Portability** - The executable software package is not dependent on the host OS. Hence, it is portable and can run consistently across multiple platforms.

- **Speed** - Since the containers share the host machine's OS kernel, they do not have extra overheads. Therefore, they can boost the starting time, reduce server and licensing costs, and drive higher server efficiencies.

- **Scalability** - Containerization allows adding new functions, updates, and features without affecting original applications. Therefore, they are highly scalable with minimum resource usage.

- **Agility** - Application containers originally started with the Docker engine and simple developer tools that worked for Windows and Linux OS. Today, they have shifted to the highly advanced Open Container Initiative management, allowing DevOps tools for rapid app development.

- **Efficiency** - Developers can share app layers across containers because the software in these environments shares the machine's OS kernel. Their inherent smaller capacity and minimal startup times allow developers to run multiple containers on the same machine as one VM. Thus, it increases overall efficiency and reduces server and licensing costs.

- **Fault Isolation Ability** - Since containerization works in isolation, it can operate independently of others. It means that the failure of one component in a container does not affect the operation of the others.

- **Security** - Isolation prevents malicious codes from affecting other containerized apps or the host OS. It also helps developers share additional features without risk.

- **Ease of Management** - The container orchestration platform helps automate containerized workloads and services' installation, management, and scaling.

- **Continuity** – Since isolation allows different containers to run independently, developers can rectify the errors in a container without causing downtime in the others, which ensures continuity.

- **Developer-friendliness** - Since developers can use a single environment for production and development, containers are developer-friendly.

Disadvantages of Containerization

However, containerization also has a few drawbacks, like the following:

- **Security Management Risks** - Containers have multiple layers compared to traditional VMs. Therefore, they are more prone to security risks. Containers require multi-level security, including securing the application, registry, host OS, and Docker daemon.

- **Orchestration** - VMs can work with a single orchestrator, whereas containers require selecting from multiple orchestrators like Mesos, Kubernetes, or Swarm.

- **Data Storage** - Storage in virtual machines is straightforward, but not so in containers. Data storage in containers requires moving the data from the container to the host system. The significant drawback is that the data can disappear forever when the container shuts down.

- **Monitoring** - Container performance and security issues require continuous monitoring. However, external monitoring services, analytics, and essential monitoring tools are available.

Despite the drawbacks, the benefits of application containerization make it one of the most sought-after services for DevOps. Therefore, developers must prudently decide whether they need containers, depending on the specific cloud requirements.

Containerization vs. Virtualization

With the knowledge of containerization and how it is better than virtualization, you can compare the two concepts and see how containerization benefits business enterprises.

Containerization and virtualization are similar in many respects because both these environments use comparable properties at the technical level. However, they deliver different outcomes. Here are a few prominent differences between the two technological concepts.

Parameters	Virtualization	Containerization
Isolation	Virtualization results in a fully isolated VM instance (a VM unit) and OS.	Containerization isolates the host OS machine and containers from one another.
Deployment	Each virtual machine has a hypervisor, a program that manages the virtual machine on the physical device.	In containerization, Docker is used to deploy individual containers. Alternatively, Kubernetes is used to orchestrate multiple containers across multiple systems.
Different OS	Virtualization can host multiple operating systems, each with its kernel.	Containerization allows running all containers via user mode on one operating system.
Persistent Virtual Storage	Virtualization assigns a VHD (virtual hard disk) to each virtual machine. In the case of multiple servers, it assigns a server	In the case of containerization, the local hard disk is used for storage for single nodes and SMB for shared

	message block (SMB).	storage across multiple nodes.
Guest Support	Virtualization allows multiple users to use a range of OS on the same server or machine.	Containerization relies on the host OS. Therefore, a Linux container cannot run on Windows.
Virtualized Networking	Virtualization uses VNAs (Virtual Network Adaptors) to facilitate networking through a master NIC (Network Interface Card).	The VNA is divided into multiple isolated views for lightweight network virtualization.
Virtual Load Balancing	Failover clusters are used to run VMs with load-balancing support.	Containerization maximizes resource utilization because it uses orchestration via Kubernetes or Docker to start or stop containers.

Benefits and Drawbacks of Virtualization

Since you have understood the difference between virtualization and containerization, it is also good to learn the benefits and drawbacks of virtualization.

Benefits of Virtualization	**Drawbacks of Virtualization**
● Efficient resource utilization through multi-tenant support on hardware	● Higher upfront costs result in an extended ROI. The recovery can take many years.

● Enables easier multi-cloud migrations because of enhanced cloud portability. ● High availability by spooling virtualized resources instantly and decommissioning once complete. ● Quick deployment, as the OS and the dependencies are already loaded on the hypervisor. ● Better business continuity because of easy virtual recovery through duplication and backups.	● Multiple virtualized instances can take a long time to scale. ● The risk of data breaches is high in public cloud virtual instances, with high possibilities of data or kernel leaks. ● Lower performance levels because hypervisor technology comes with performance overheads. ● It can create more management burdens for the IT department if not adequately monitored because virtual servers containing virtualized instances can sprawl endlessly.

Chapter Questions:

1. What is the primary purpose of remote server automation?
   a) To manually control servers from a remote location
   b) To reduce the need for physical server maintenance
   c) To automate tasks and processes on remote servers
   d) To increase the number of servers in a network

2. Which of the following is a popular remote server automation tool?
   a) Adobe Photoshop
   b) Ansible

c) Microsoft Word

d) Google Chrome

3. What is the key benefit of automating server tasks?

a) It reduces errors and increases efficiency.

b) It allows servers to be located in more places.

c) It makes servers larger and more powerful.

d) It increases the need for IT staff.

4. Which of the following is a popular tool used for infrastructure provisioning?

a) Adobe Photoshop

b) Terraform

c) Google Chrome

d) Microsoft Word

5. What is the primary purpose of infrastructure configuration management?

a) To manually control servers from a remote location

b) To reduce the need for physical server maintenance

c) To increase the number of servers in a network

d) To identify, record, control, report, audit, and verify service assets and configuration items

6. What is the key benefit of automating infrastructure tasks?

a) It increases the need for IT staff.

b) It reduces errors and increases efficiency.

c) It allows servers to be located in more places.

d) It makes servers larger and more powerful.

7. What is the primary difference between virtualization and containerization?

a) Virtualization is used for applications requiring a lot of storage, whereas containerization is used for applications requiring less storage.

b) Virtualization is used for applications with high workload requirements, whereas containerization is used for applications that need to be nimble.

c) Virtualization produces a virtualized environment for the entire operating system, whereas containerization virtualizes the parts needed for an application to operate.
d) Virtualization is used for applications requiring a dedicated, isolated environment, whereas containerization is used for applications sharing hardware resources with other environments.

8. Which of the following is crucial when deciding between virtualization and containerization?
a) IT Infrastructure
b) Workload characteristics
c) Storage requirements
d) All of the above.

9. Can virtualization and containerization be used together?
a) Yes, but only in certain situations
b) Yes, the container's host OS can be powered by a VM.
c) No, they are mutually exclusive technologies.
d) No, they cannot be used together under any circumstances.

10. What is the key benefit of using both virtualization and containerization together?
a) It allows for creating a dedicated, isolated environment for containers while sharing hardware resources with other VM environments.
b) It increases the number of servers in a network.
c) It reduces the need for physical server maintenance.
d) It allows servers to be located in more places.

Answers to Multiple Choice Questions

1. c
2. b
3. a
4. b

5. d
6. b
7. c
8. d
9. b
10. a

Final Words

You have understood that automation is crucial to DevOps because it reduces human error and increases efficiency and how remote server automation plays a critical role in the DevOps cycle. Besides, concepts like infrastructure configuration, provisioning, monitoring, application containerization, and its positive and negative aspects have also been covered. Finally, this chapter also explored virtualization and highlighted its differences with containerization. Thus, as this chapter ends, you have covered most techniques/technologies associated with DevOps.

References

1. Reintech. (n.d.). Remote Server Automation. https://reintech.io/terms/category/remote-server-automation
2. Codecademy. (n.d.). Infrastructure Configuration. https://www.codecademy.com/article/infrastructure-configuration
3. Wilson, B. (2023, July 22). 20 Devops Tools For Infrastructure Automation and Monitoring DevopsCube. https://devopscube.com/devops-tools-for-infrastructure-automation/
4. Veritas. (n.d.). What is Containerization? What are the Benefits? https://www.veritas.com/information-center/containerization

5. Baeldung. (2020, July 12). *Virtualization vs Containerization*. Baeldung on Computer Science. https://www.baeldung.com/cs/virtualization-vs-containerization

Chapter 12 - Transitioning into the Role of DevOps

This chapter will look at important facts about the entry into DevOps and help outline the skills and expertise needed to excel in this field.

Looking to get into DevOps but need to figure out how? Don’t worry. Here are all the essential insights, tips, and tricks to help you get started.

Getting a Foot in the DevOps Door

As a DevOps engineer, you hold a pivotal position, uniquely poised to make a far-reaching impact from the top-down and bottom-up, ushering in a new era of collaboration and efficiency. The information below gives you an idea of what to consider and understand when you are at the door to be a DevOps engineer.

- **Embarking on the DevOps Journey**

  At its core, DevOps represents a convergence of practices and a culture shift, acting as the connective tissue between software development (Dev) and IT operations (Ops). As a DevOps engineer, your mission is clear: to streamline processes, enhance efficiency, and catalyze collaboration, all with the ultimate aim of influencing project success.

- **Evaluating Your Skill Set**

  Your journey begins with introspection. Take the time to assess your current skills and knowledge through self-reflection. Don't underestimate the power of transferable skills from your existing role. These hidden gems could prove invaluable in your transition to DevOps.

- **Discovering Your Entry Points**

Setting out on this adventure involves discovering entry points and seizing opportunities that beckon. Keep your eyes peeled for entry-level positions and avenues that align with your aspirations.

Networking and mentorship can be your allies in navigating the DevOps landscape, opening doors that might have otherwise remained closed.

Focusing on the Mindset of an Engineer

How to get into the right mindset for DevOps? In engineering, a particular attitude plays a decisive role in success. This mindset combines technical prowess, creative problem-solving capabilities, and adept communication. Here's how.

- **The DevOps Way**

  The DevOps mindset represents a collaborative spirit in software development, stressing continuous improvement and automation. It champions dismantling barriers between development and operations teams, fostering a culture of teamwork and dialogue.

  Moreover, it cultivates resilience, the ability to adapt swiftly to changes without significant disruptions or complications. It's akin to a well-tuned orchestra, where everyone plays in harmony.

- **Mastering Problem-Solving and Critical Thinking**

  Engineers are like puzzle solvers when confronting intricate technical challenges. They employ problem-solving and critical thinking, breaking down issues into manageable bits. Engineers assess problems, propose solutions, and weigh the merits of each. Learning from hiccups and missteps is as important as formal learning.

- **The Power of Communication and Collaboration**

  Practical communication skills are engineers' secret weapons. They enable them to convey intricate technical ideas to those unfamiliar with technical jargon.

Engineers thrive in cross-functional teams, wherein collaboration reigns supreme. Tools and practices like agile development methodologies serve as bridges, connecting minds and skills to create synergistic results.

The engineer's mindset encompasses a blend of technical mastery, creative problem-solving, and adept communication. It embraces the DevOps philosophy, fosters problem-solving and critical thinking, and champions effective communication and collaboration. It's a mindset that fuels innovation, overcomes obstacles, and turns ideas into reality. Are you ready to don the engineer's hat and get started?

Market Demand for a DevOps Engineer

The market is as hot as ever for anyone who wants to get into DevOps. Here are some statistics to show how in-demand DevOps is.

- **Industry Trends**

  More and more organizations from diverse sectors are beginning to use DevOps. They love it because it helps them release new software quickly. The DevOps market is growing fast!

  By 2028, the DevOps market will be worth around $25.5 billion, a significant rise from $10.4 billion in 2023.

  The prominent players in the market are flourishing, thanks to DevOps. Amazon, Netflix, and Etsy have shown how amazing DevOps can be. Amazon releases code swiftly, Netflix has hardly any downtime, and Etsy deploys code a whopping 50 times daily. It's like magic!

- **Job Opportunities**

  Numerous organizations and businesses want skilled people in DevOps. Below are some numbers that demonstrate it.

- Did you know? There are more than 17,000 DevOps job ads in the USA—furthermore, the salaries for DevOps range from $96,600 to $122,000 on average.
- A quarter of software developers wish they knew DevOps, and nearly a third of recruiters are struggling to find DevOps experts. If you're thinking about a career change, this could be perfect!

Must-Have Skills to Penetrate the DevOps Market

DevOps blends software development and IT operations to streamline the software development process, ensuring speedy, high-quality, and continuous delivery.

As the DevOps market expands, the demand for proficient professionals is rising. Here are the crucial skills you need to thrive in the DevOps landscape:

- **Technical Skills**

  The following are some of the essential technical skills you require to become a DevOps Engineer:

  - **Scripting:** Scripting is critical in DevOps, enabling task automation, system monitoring, data collection, and resource management. Beginning with user-friendly languages like PowerShell or Python is a smart choice due to their versatility and widespread use in DevOps.
  - **Automation:** Automation is the backbone of DevOps, reducing manual errors, boosting efficiency, and accelerating software development. Widely adopted automation tools such as Jenkins, Ansible, and Terraform are indispensable in the DevOps toolkit.
  - **Infrastructure as Code (IaC):** IaC involves managing and provisioning infrastructure through code, enhancing accuracy, efficiency, and consistency. Tools like Chef, Puppet, and Ansible are invaluable for implementing IaC practices.

- **Tools and Technologies**

  If you aim for DevOps, you should start with the following tools and technologies.

  - **Docker:** Docker is a containerization platform that streamlines application packaging and ensures consistency, scalability, and portability.
  - **Kubernetes:** Kubernetes is an open-source software that enhances availability and scalability by automatically handling how applications run and grow.
  - **Jenkins:** Jenkins, an open-source automation server, empowers developers to build, test, and deploy software, significantly enhancing efficiency, error reduction, and the speed of the software development process.

- **Soft Skills**

Apart from the technical expertise, here are some indispensable soft skills for DevOps.

- **Teamwork:** DevOps thrives on collaboration, demanding effective teamwork, open communication, and receptiveness to feedback.
- **Adaptability:** In the dynamic DevOps realm, adaptability is vital. Quick learning, flexibility, and a willingness to embrace new approaches are essential.
- **Problem-solving:** DevOps revolves around problem-solving. Being adept at identifying issues, troubleshooting, and generating innovative solutions is paramount.

Additional Resources and Certification Opportunities

If you're keen on expanding your knowledge of DevOps and gaining valuable certifications, numerous resources are at your disposal. Here's a compilation of top-notch options to consider:

- **Online Learning and Tutorials**

  Here are some of the prominent online sources of learning for DevOps:

  - DevOps courses offered through Coursera
  - Continuous Delivery & DevOps Course by Udemy
  - DevOps online training courses by LinkedIn Learning

- **DevOps Certification Pathways**

  Below are some programs and certificates to help you become a qualified DevOps professional.

  - AWS Certified DevOps Engineer
  - Docker Certified Associate
  - Simplilearn's DevOps Engineer Masters Program
  - Professional Certificate in DevOps (edX)

  A DevOps certification offers several advantages, such as broadening job prospects, potentially higher earnings, and formal recognition of your proficiency in the field.

- **User Groups and Communities**

  Participating in DevOps-focused communities and forums can facilitate networking and knowledge-sharing with peers. Notable options include the DevOps Institute Community and DevOpsDays events.

These resources and certification pathways will serve as your guiding lights on your journey toward mastering DevOps, offering a spectrum of opportunities for growth and development. Take a look at the supplementary material provided below to get started now.

Chapter Questions:

[1] What is one essential skill emphasized for success in DevOps besides technical prowess?

a) Mastery of a single programming language
b) Expertise in server maintenance
c) Effective communication and collaboration
d) Advanced knowledge of operating systems

[2] What is the primary role of automation in DevOps?

a) To handle customer support queries efficiently.
b) To manage the company's finances and budgeting.
c) To reduce manual errors and accelerate software development.
d) To design the user interface of software applications.

[3] Which DevOps practice emphasizes the need for frequent but small updates?

a) Continuous Integration
b) Agile Programming
c) Continuous Deployment
d) Infrastructure as Code

[4] Which tools are commonly used for containerizing cloud-based applications in DevOps?

a) Jenkins
b) Docker
c) Git
d) JIRA

[5] In the realm of DevOps, why is ongoing or continuous feedback crucial?

a) To confirm that the software remains ready for release.

b) To refine the software in line with user requirements and desires.

c) To perpetually assess code for potential risks.

d) To promote consistent upskilling for DevOps experts.

Chapter QUetions Answers:

1. c) Effective communication and collaboration
2. c) To reduce manual errors and accelerate software development.
3. c) Continuous Deployment
4. b) Docker
5. b) To refine the software per user requirements and needs.

Supplementary Material

Books:

For a deeper understanding of DevOps principles, a wealth of books and blogs await your exploration. Dive into renowned works like:

- “The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win” by Gene Kim, Kevin Behr, George Spafford
- "Continuous Delivery" by Jez Humble and David Farley
- "The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations" by Gene Kim, Jez Humble, Patrick Debois, and John Willis.

Blogs:

If you don't want to invest in books, here are blogs that you should go for:

- DevOps.com
- The New Stack
- DZone DevOps

YouTube:

Some famous YouTube channels and courses:

[1] YouTube Channel: KodeKloud

URL: https://www.youtube.com/@KodeKloud

[2] YouTube Channel: GUVI

URL: https://www.youtube.com/@Guvisharing

[3] YouTube Channel: Edureka

URL: https://www.youtube.com/watch?v=hQcFE0RD0cQ

References

1. Abukar, M. (2021, November 30). The pathway to DevOps. Medium. https://moabukar.medium.com/my-pathway-to-devops-22f9f0cc950
2. Adam, J. (2023, July 11). DevOps stats and facts -- all the numbers you might ever need on DevOps in 2022. K&C. https://kruschecompany.com/devops-statistics-and-facts/
3. Casey, K. (2017, September 11). DevOps Jobs: 6 eye-opening statistics. The Enterprisers Project. https://enterprisersproject.com/article/2017/9/devops-jobs-6-eye-opening-statistics
4. Vailshery, L. (2023, April 28). DevOps - statistics & facts. Statista. https://www.statista.com/topics/9369/devops/
5. WeCloudData. (n.d.). DevOps engineer job market. https://wecloudata.com/career-guides/devops-engineer/devops-engineer-job-market/
6. DevOps market. (2023 March). MarketsandMarkets. https://www.marketsandmarkets.com/Market-Reports/devops-market-824.html
7. DevOps market. (2022 January). Allied Market Research. https://www.alliedmarketresearch.com/devops-market
8. Mathur, G. (2023, September 5). DevOps Mindset: Implementation Guide. Knowledgehut. https://www.knowledgehut.com/blog/devops/devops-mindset

9. Nana. (2022, October 13). From Zero to DevOps Engineer - DevOps Roadmap for YOUR specific background. DEV. https://dev.to/techworld_with_nana/from-zero-to-devops-engineer-devops-roadmap-for-your-specific-background-4h8n

10. Reddit.(2014). Getting your foot in the door... https://www.reddit.com/r/devops/comments/2cq7dl/getting_your_foot_in_the_door/

11. Rosencrance, L. (2023, April 19). 11 best free DevOps certifications and training courses in 2023. Whatis. TechTarget. https://www.techtarget.com/whatis/feature/9-best-free-DevOps-certifications-and-training-courses

12. Schaub, W. (2019, May 9). 5 essential values for the DevOps mindset. Opensource.com. https://opensource.com/article/19/5/values-devops-mindset

13. Model Thinkers. (n.d.). DevOps Mindset. https://modelthinkers.com/mental-model/devops-mindset

14. COPADO. (2022, July 12). The DevOps mindset: How to adopt it within your organization. https://www.copado.com/devops-hub/blog/the-devops-mindset-how-to-adopt-it-within-your-organization

15. Canlas, J. (n.d.). Your path to becoming a DevOps Engineer in 2023: A comprehensive guide. Instatus. https://instatus.com/blog/become-a-devops-engineer

16. Udemy. (2023, April). DevOps MasterClass: Terraform Jenkins Kubernetes Ansible. https://www.udemy.com/course/devops-training/