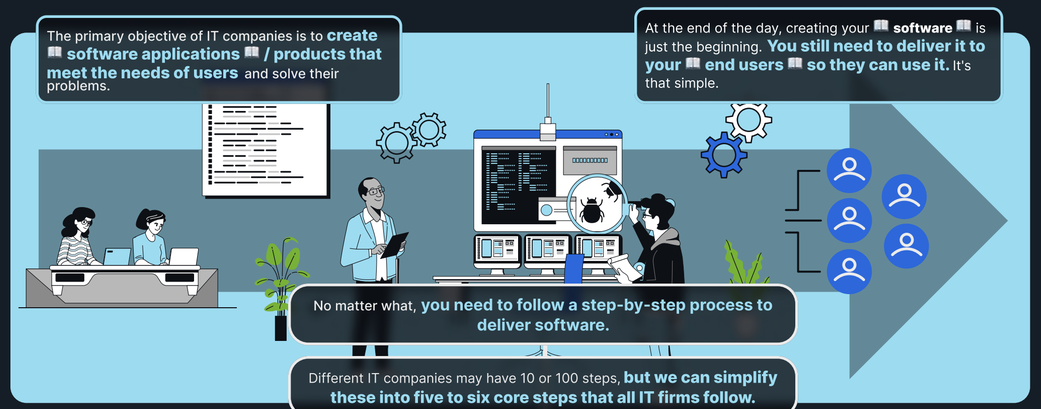
KodeKloud

DevOps for Absolute Beginners

DevOps for Absolute Beginners

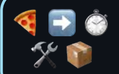


The primary objective of IT companies is to **create 📖 software applications 📖 / products that meet the needs of users** and solve their problems.

At the end of the day, creating your 📖 **software** 📖 is just the beginning. **You still need to deliver it to your 📖 end users 📖 so they can use it.** It's that simple.

No matter what, **you need to follow a step-by-step process to deliver software.**

Different IT companies may have 10 or 100 steps, **but we can simplify these into five to six core steps that all IT firms follow.**



Similar to ordering a pizza at Pizza Hut and receiving it after 15 minutes. **Both scenarios are similar at a high level, but delivering a software product takes much longer than 15 minutes.**

❗ **Everything starts with someone's brilliant idea to create something new or improve something existing.**

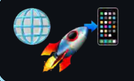
First, come **up with a good idea** for an application that solves a problem or enhances a workflow.

Next, **list the features it should have.**

Developers will then **write the code.**

After 📖 **coding,** 📖 testers will check the application to **ensure it works.**

Once the application is tested, you must 📖 **build and package** 📖 **your application** into a format that can run on the 📖 **server** 📖.



Finally, you need to **deploy it on a** 📖 **public server** 📖 so users can access it.

Is this the end?

No.

After delivering your software for the first time, you'll still need to address a few questions.

- ❓ **Are there any unexpected behaviors in the software that weren't intended?**
- ❓ **Are there any suggestions or 📖 performance 📖 improvement requests coming from end-users?**
- ❓ **Are users asking for any new ideas or features?**

🔁 **This is a never-ending cycle.**

You will always need to attend to these queries and provide solutions

This means you must continuously 📖 **monitor and operate** 📖 the application even after it is launched.

So, every time you need to introduce a new feature, fix something, or improve something, **the team will go through the same core steps we discussed previously.**

This is what we call as

📖 Software Development Life Cycle 📖 (SDLC):

1. BRAINSTORMING
2. REQUIREMENT GATHERING
3. CODING
4. TESTING
5. BUILD AND PACKAGE
6. DEPLOYING & RUNNING
7. MONITORING

This creates a continuous delivery process, making software development an ongoing cycle.

THIS CONTINUOUS DELIVERY **PROCESS SHOULD BE:**

FAST
Rapid delivery means the client gets new features and improvements quickly, keeping their business competitive.

FREQUENT
Regular updates give users new features, better performance, and quick fixes consistently, keeping their software up-to date.

HIGH-QUALITY
High-quality releases provide the end user with a reliable product, reducing disruptions and enhancing satisfaction.

Achieving this is challenging because of certain barriers in the way IT companies or teams have traditionally worked.

Introducing You to the Prime Challenge

Delivering software application has **two main parts** **Coding & Running**, with all other tasks(steps) revolving around and supporting them!

🚫 This can create a gap 🚫

CODING IT

📖DEVELOPERS TEAM📖
Developers code the application.

&

RUNNING IT

IT 📖OPERATIONS TEAM📖
The operations team runs the application.

This miscommunication can cause problems.

Dev Team
We wrote the code but can't run it ❌

Developers often **write code without thinking about how it will run** or be deployed.

Ops Team
We run the app but don't know how it works 🤷

The operations team **tries to run the application without fully understanding** it.

Developers might finish coding **but** The 📖 **deployment guide** 📖 for the operations team might not be good or detailed enough.

If the feature or bug fix deployment **gives deployment errors** **but** The **operations team** assumes it's a problem with the code and sends it back for improvements.

This back-and-forth can stretch the release period from days to weeks or even months. These barriers make it difficult to deliver software quickly, frequently, and with high quality.

DevOps was introduced to remove these type of obstacles and speed up the software delivery process while maintaining quality through a fully-automated, streamlined process.

Fundamentals of DevOps

The perfect starter course to launch yourself into the key concepts of the DevOps world!

MICHAEL FORRESTER

Principal Trainer at KodeKloud, DevOps Advocate, Certified AWS DevOps Engineer with 11 additional AWS certifications

PRACTICAL IMPLEMENTATION AND THE **ROLE OF DEVOPS ENGINEERS**

Creation of DevOps Role

- Collaborating development and operations tasks
- Required specific focus and expertise

Role Variations

- Developers or operations team taking on DevOps tasks
- Dedicated DevOps Engineer

This role ensures fast, high-quality delivery through key measures.

Keep in mind, **DevOps emphasizes cultural, philosophical, and process changes over specific roles!**

Want to Enter IT Industry Without a Software Engineering Background?

Many people want to join the IT industry because of its

- **Potential for innovation**
- **High paychecks**
- **And perks compared to other professions**

Which is very appealing and true.

Starting out as a junior or associate developer/software engineer is not easy.

Challenges Starting Out...

- Requires coding experience
- Familiarity with at least one 📖 **programming language** 📖
- Proficiency in 📖 **data structures and algorithms** 📖

You can still work in the IT industry without being an expert in coding.

Alternative Path

- Work in IT as a tech professional
- No need for deep coding expertise
- No need to be an experienced programmer.

DevOps Engineer role is ideal

Let's walk through the Software Development Life Cycle - SDLC

Key Areas of Focus

- What you need to know
- Where to give more focus
- High-level understanding needed

As a DevOps engineer, you don't need to focus on this part, but it's good to understand:

When an idea is proposed, it goes through several steps before becoming a fully functional part of the project.

DEFINING REQUIREMENTS:

- **Brainstorming and Documenting Needs:** Collect what users and stakeholders need, and turn these into clear requirements.
- **User Stories:** Create simple descriptions based on these needs, detailing how the system will help users.

MANAGING TASKS:

- **Project Management Tools:** Tools like JIRA, ClickUp, and Asana are used to organize and track these tasks and requirements.
- **Task Prioritization:** These tools help teams prioritize tasks, ensuring that the most critical requirements are addressed first.
- **Progress Tracking**: Regular updates and tracking ensure that the project stays on schedule .

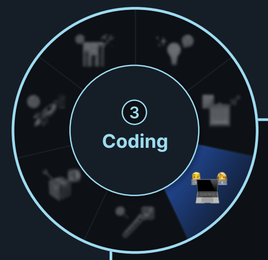
EXPLORE...

FEEDBACK LOOPS:
Regularly reviewing and incorporating feedback helps in improving the project adapting to any changes in requirements or priorities.

COMMUNICATION:
Alignment and updates.

PROJECT MILESTONES:
Identify key deliverables and deadlines in the project timeline.

STAKEHOLDER ENGAGEMENT:
Involve users and stakeholders early to gather accurate requirements.



Coding is the main responsibility of software engineers. It requires talent in coding and creating efficient software using best practices. However, **as a DevOps engineer, you are not responsible for coding the actual software** and do not need to be involved in this phase.

However, it is important to learn the following to gain a high-level understanding:

CHOOSING DEVELOPMENT METHODOLOGIES:

Selecting the right approach for structuring the project is essential for long-term success.

- **Waterfall:** Linear and step-by-step process where each phase must be completed before moving to the next.
- **Agile:** Flexible and iterative process allowing for regular feedback and continuous improvement.

STRUCTURING SOFTWARE APPLICATIONS:

Understanding how to structure your codebase for better maintenance and scalability.

- **Microservices:** Small, independent services that can be developed and deployed separately.
- **Monolithic:** All parts of the application are interconnected and deployed together.

MANAGING WORK WITH SCRUM:

A popular Agile framework for managing tasks and ensuring steady progress.

- **Scrum:** Uses short, regular intervals called 'sprints' to manage tasks.
- **Daily Stand-Ups:** Quick meetings to discuss progress and obstacles.

VERSION CONTROL:

Managing code changes and team collaboration.

- **Tracking Changes:** Tools like Git and SVN allow multiple people to work on the same code without conflicts.
- **Code Storage:** Central places like GitHub, GitLab and Bitbucket store and manage code.

EXPLORE...

INTRODUCTION TO DATABASES:
Learning how 📖 **data** 📖 is stored and managed in software applications.

SQL DATABASES:
Store structured data in tables (e.g., MySQL, PostgreSQL).

NOSQL DATABASES:
Store unstructured data flexibly (e.g., MongoDB, Cassandra).

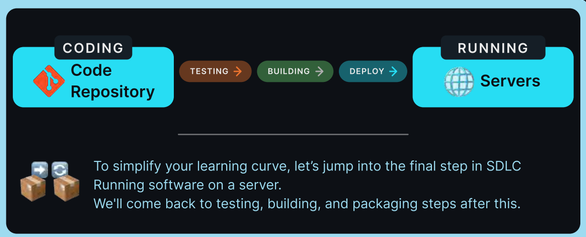
BASICS OF PROGRAM EXECUTION:
Code runs as binary (1s and 0s), the machine language. Compilers translate all code before running it, while interpreters execute it line by line.

GIT for Beginners
Learn Git with simple visualisations, animations and by solving lab challenges.

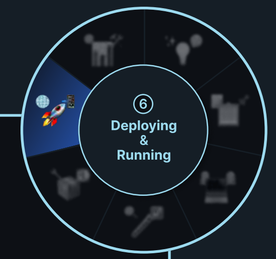
TAUGHT BY:
Lydia Hallie

Once coding is completed, only 50% of the work is done.

The next major task is to deliver the software to a public server and run it, so all end users can access and see it. This was seen as the final step in the delivery process.



To simplify your learning curve, let's jump into the final step in SDLC
Running software on a server.
We'll come back to testing, building, and packaging steps after this.



To run the coded software, so that end users can publicly access it, you need some kind of 📖 **infrastructure,** 📖 specifically server computers. These are special types of computers designed to handle such tasks.

In this area, you need to work on several important topics to become a DevOps engineer.

WHAT IS A SERVER?

A powerful computer that provides resources, data, and services to other computers (clients) over a network.

- Think of it like a waiter in a restaurant serving food and drinks to customers.
- When you visit a website, your device requests information from a server. The server sends back the data needed to display the webpage.

A SERVER CAN EXIST IN TWO MAIN 📖 ENVIRONMENTS 📖:

on-premises and in the cloud.

ON-PREMISE:
Your own servers and hardware located in your office.

CLOUD:
Using remote servers on the internet to store and manage data.

BENEFITS OF CLOUD INFRASTRUCTURE

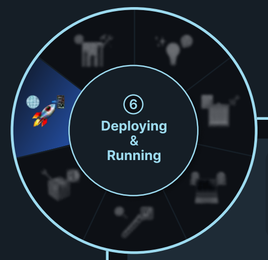
LOW COSTS:
No need to buy expensive hardware; pay for what you use.

NO MAINTENANCE:
The cloud provider handles all updates and fixes.

MANY SERVICES:
Offers security, load balancing, storage, and more.

SCALABILITY:
Easily increase or decrease resources as needed.

PAY-AS-YOU-GO:
Only pay for the resources you use, saving money.



CLOUD COMPUTING

Allows companies to use these 🕮 **cloud servers** 🕮 over the internet, making it easy to get more resources when needed without buying more hardware.

🕮 CLOUD PROVIDERS 🕮

AZURE
Microsoft's cloud service with many tools.

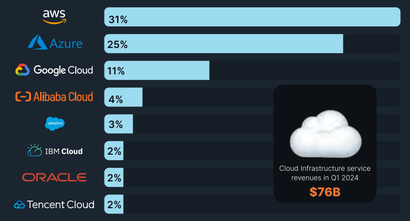
AWS
(Amazon Web Services)
A leading cloud provider with lots of services.

GCP
(Google Cloud Platform)
Google's cloud service, great for 🕮 **data** 🕮 and machine learning.

AMAZON MAINTAINS CLOUD LEAD AS MICROSOFT EDGES CLOSER

Worldwide market share of leading cloud infrastructure service providers in Q1 2024*

*Includes 🕮 **platform as a service (PaaS)** 🕮 and 🕮 **infrastructure as a service (IaaS)** 🕮 as well as hosted private cloud services
Source: Synergy Research Group



EXPLORE

SECURITY

Protecting data and systems.

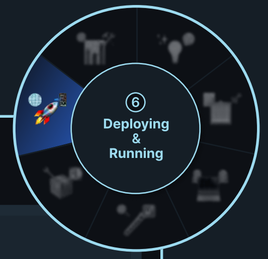
- **Security Best Practices:** Strong passwords and two-factor authentication enhance security.
- 🕮 **Encryption:** 🕮 Learn how data is protected using encryption techniques.
- 🕮 **Firewalls:** 🕮 Understand how firewalls block unauthorized access.

SERVERLESS COMPUTING

No need to manage servers; the cloud provider handles it all.

- **Easy deployment:** Simplified Setup
- **Automatic scaling:** Adjusts Resources as Needed
- **Pay only when code runs:** Cost-Effective

As a DevOps Practitioner, you don't need to be an expert in Security, Networking or Take over managing the whole infrastructure. These areas are typically handled by Specialized Individuals like System Administrators and Network/Security Engineers. However, having a basic understanding of the mentioned topics is essential.



To make these servers work efficiently, a good understanding of operating systems and networking is essential.

OPERATING SYSTEMS

These are the software that makes servers run, like Windows, MacOs or Linux. They manage resources and run applications.

Operating System (Linux)
Basics, file system, CLI, shell commands.

FILE SYSTEM
Files are stored in a structured manner for easy access and management.

CLI
Allows users to interact with the OS using text commands.

SHELL COMMANDS
Commands like ls, cd, and rm are used to manage files and directories.

FILE PERMISSIONS
Understand how to control access to files and directories.

NETWORKING

This enables communication between devices and servers, ensuring data can travel quickly and securely.
Networking Basics of LAN, WAN, IP Addresses, and protocols - 🕮 **TCP/IP** 🕮 , 🕮 **HTTP/HTTPS** 🕮, 🕮 **FTP** 🕮, 🕮 **SSH** 🕮.

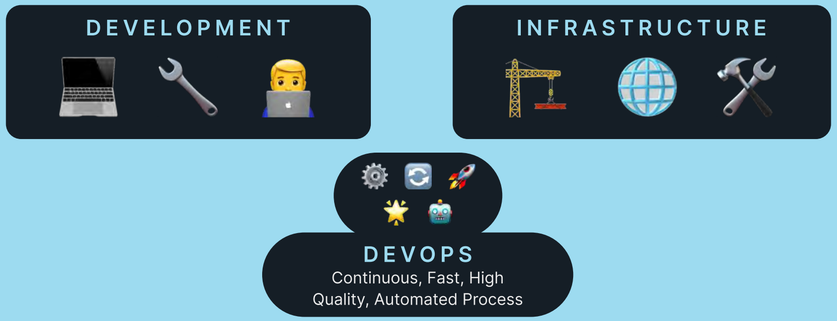
LAN (LOCAL AREA NETWORK)
Connects devices in a small area.

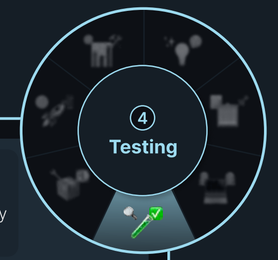
WAN (WIDE AREA NETWORK)
Connects devices over large distances.

IP ADDRESSES
Understanding IP addresses and their roles in communication.

The journey from coding to 📖 running software 📖 in a server, known as releasing software to end users, is where DevOps plays a crucial role.

This phase includes the majority of the tasks and responsibilities of a DevOps Engineer. So, you need to give more attention from here onwards.





After coding, the focus shifts to testing your software product. **As a DevOps engineer, you are not responsible for or directly involved in this work.** However, it is important to understand how the application is tested. Knowing how these tests work makes you a fully qualified DevOps engineer.

WHAT IS SOFTWARE TESTING?

Basics and importance of ensuring software quality.

SOFTWARE TESTING
Checking if the software works correctly and does what it's supposed to do.

IMPORTANCE
Finds and fixes problems (bugs) to make the software reliable and user-friendly.

BUG
An error or flaw in the software that causes it to behave unexpectedly or incorrectly.

MANUAL VS AUTOMATED TESTING

Differences and why automated testing is beneficial.

MANUAL TESTING
Humans check the software by following steps and reporting issues.

Slower and can miss problems because people make mistakes.

AUTOMATED TESTING
Computers run tests automatically using scripts.

Faster and more consistent in finding problems.

TYPES OF AUTOMATED TESTS

Different types of tests to check various aspects of the software.

UNIT TESTING
Tests individual parts of the software.

INTEGRATION TESTING
Tests how different parts of the software work together.

END-TO-END TESTING
Tests the entire application from start to finish.

POPULAR TESTING TOOLS

Tools like Selenium, JUnit, Cypress automate tests.

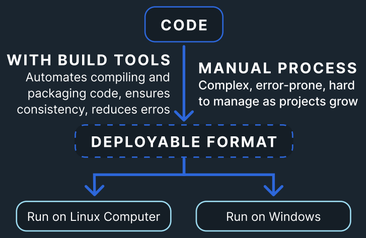
SELENIUM
Automates web browser testing.

JUNIT
Framework for unit testing in Java.

CYPRESS
End-to-end testing for web applications.

Now we move on to building the software application. This is where we convert the application into a deployable format, such as a JAR, WAR, Bundle or ZIP file.

To Become a DevOps engineer, there are some important things you need to know in this process.



CONVERTING CODE TO DEPLOYABLE FORMAT

This process turns written code into a program that can be executed on a computer.

DEPLOYABLE FORMAT

The final product is an application or program that can be installed and run on other machines.

THE BUILD PROCESS STEPS

COMPILATION

The source code is compiled into executable code. For example, Java code is compiled into bytecode. Translates human-readable code into a format that computers can execute.

PACKAGING

The compiled code is packaged into a format that can be easily deployed, such as a JAR or WAR file for Java applications. Prepares the software for deployment by bundling all necessary components together.

DIFFERENT BUILD TOOLS

Tools used for managing and automating the build process.

MAVEN
Helps manage and build Java projects.

GRADLE
A flexible tool that automates building, testing, and deployment.

NPM
Manages JavaScript projects and dependencies.

As software projects grow, they often rely on external libraries or tools to add features or functionality. Managing these dependencies manually can be time-consuming and error-prone. This is where package managers come in.

HOW PACKAGE MANAGERS WORK

Handle dependencies.

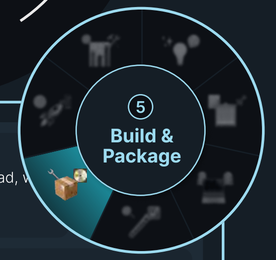
DEPENDENCIES

These are external libraries or tools that the software needs to work. Think of them as extra features or plugins.

PACKAGE MANAGERS

Tools that help install, update, and manage these dependencies. They ensure that all the necessary parts are in place for the software to run.

With the packaged software application bundle(WAR, Zip, etc.) ready, our journey doesn't stop here. **Instead of running our application on a server in the traditional way, we use the modern approach of containerizing the bundled application.**



Containerize the Software Application.

With DevOps, we don't just run our software directly on virtual machines like before. Instead, we use containers to run applications on servers. To do this, we package our application as a container image. **Docker is one of the most popular containerization technologies.**

As a DevOps engineer, there are several key concepts you need to know.

VIRTUAL MACHINES AND CONTAINERS

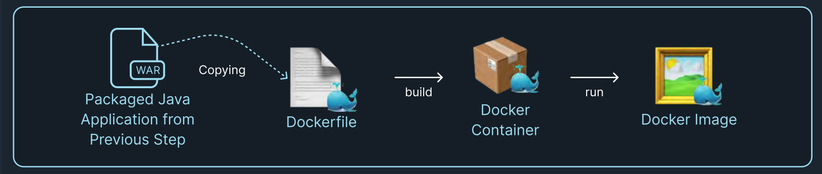
VIRTUAL MACHINE

Acts like a separate computer within your computer. Commonly used for running different operating systems on one machine.

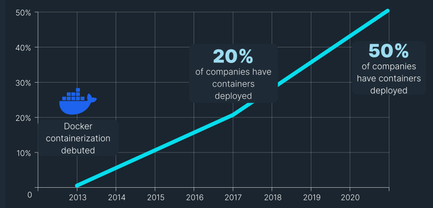
CONTAINER

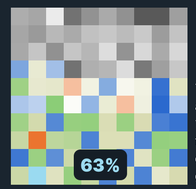
A lightweight way to run applications. Packages the software and its dependencies together.

Benefits over VMs: Faster, smaller, and easier to manage.



Containerization timeline

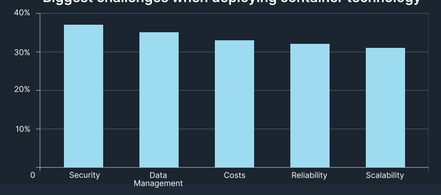


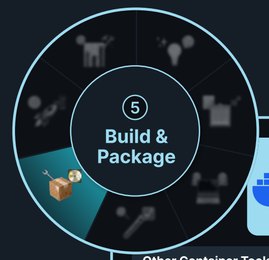


The majority of container adopters run more than 100 instances today, and the large-scale containerization ranks are growing.

Annual fees for container technologies in 2017*

Biggest challenges when deploying container technology





Docker is a tool that helps you easily **create** and **manage containers**. This ensures that the application works consistently on any computer.

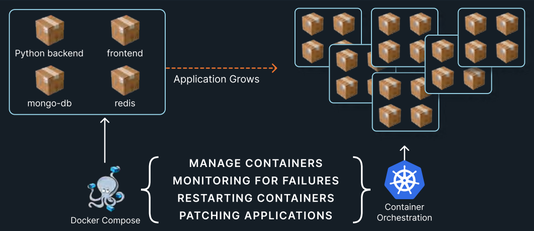
Other Container Tools

- **Podman:** A **daemonless container** engine for developing, managing, and running **OCI containers**.
- **Containerd**

podman containerd

Docker Training Course for the Absolute Beginner

TAUGHT BY:
Mumshad Mannambeth



Docker containers are used to run applications. **Docker Compose can manage a few containers, but for hundreds of containers, we use container orchestration tools like Kubernetes.**

Kubernetes(K8s) Can:

SCALE UP EFFORTLESSLY: Handle more traffic by adding containers automatically.

SEAMLESS UPDATES: Update your applications with zero downtime.

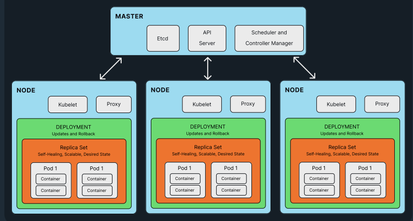
SELF-HEALING: Kubernetes automatically fixes and replaces failed containers.

EVEN TRAFFIC DISTRIBUTION: Keep performance optimal with smart load balancing.

ALWAYS ON WATCH: Kubernetes ensures your containers are always running smoothly.

AUTOMATIC RESTARTS: If something crashes, Kubernetes brings it back to life instantly.

To Become a DevOps Engineer, It's important to get familiar with its core objects, which are the building blocks for deploying and managing applications.



Push to Artifact Repository

After packaging into docker image, we should put it somewhere so everyone can access and use it. This is where we should use an **Artifact Repository** which contains Build artifacts like docker images.
DockerHub is one of the popular image repositories by Docker.
As a DevOps engineer, you should focus on the following:

Understanding Artifact Repositories

ARTIFACT REPOSITORY:
A place to store built and packaged software files(build artifacts). Like a digital library where you keep software components.

PURPOSE
Helps share and reuse these files easily. Makes it easy to use the same components in different projects.

OTHER POPULAR ARTIFACT REPOSITORIES

Other places to store Docker images:

AMAZON ECR
Amazon's storage service for Docker images.

GOOGLE GCR
Google's storage service for Docker images.

AZURE ACR
Microsoft's storage service for Docker images.

JFROG ARTIFACTORY
A tool for managing and storing software files.

GITHUB PACKAGES
GitHub's service for storing and sharing software packages.

HANDS-ON TUTORIAL

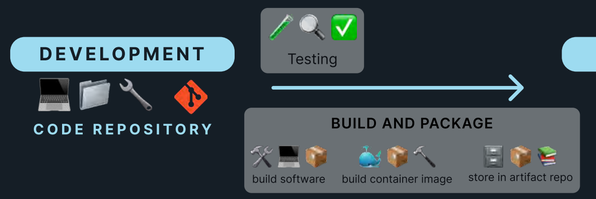
Kubernetes for the Absolute Beginners

TAUGHT BY:
Mumshad Mannambeth

🤖 🔄 🚀

Do you think Coding to 📖 Deploying Software 📖 to Server should be done manually?

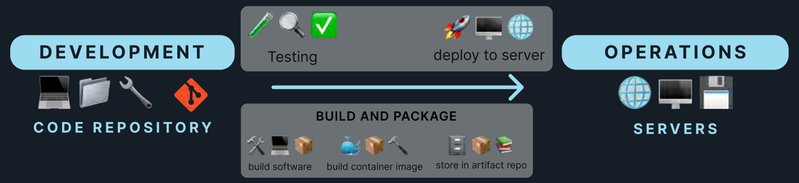
No, the purpose of DevOps is to automate this process, reducing manual intervention to make it more efficient, faster, and ensure continuous, high-quality delivery.



We should automate all the steps mentioned from running tests to pushing built container image to artifact repositories. This can be seen as a build automation pipeline. Also this process is part of the '**Continuous Integration**'. *To start this process, we should pull the code from the central repository, like GitHub.*

What happens next?

Since the testing and build steps are completed, the code needs to be pushed to the public server as we learnt earlier. This process, known as **Continuous Delivery/Deployment**, ensures that any bug fixes or new features are automatically deployed as part of the Continuous Delivery/Deployment cycle.



This flow is the heart of the **CI/CD Pipeline**, crucial for DevOps. **As a DevOps Engineer, you plan, create, and maintain this pipeline to continuously update software with bug fixes and new features.**

Jenkins is a popular tool to automate this process.

Jenkins

CI/CD

To excel as a DevOps engineer, you need to understand the CI/CD process

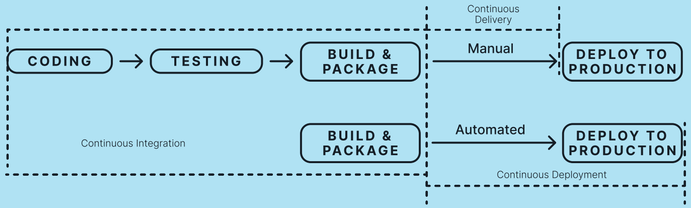
and be skilled in using an automation tool like Jenkins or GitHub Actions.

CI/CD

CI (CONTINUOUS INTEGRATION):
Frequently adding code changes into a shared repository. Developers regularly add their code changes, which are automatically tested to catch problems early.

CD (CONTINUOUS DELIVERY & DEPLOYMENT):
Ensures that code changes are automatically prepared for a release to production.

The DevOps lifecycle is an infinite loop, Because It represents the never-ending cycle of software features and improvements



WHAT'S THE DIFFERENCE BETWEEN CONTINUOUS DELIVERY AND CONTINUOUS DEPLOYMENT?

CONTINUOUS DELIVERY (CD):
A software development practice where code changes are automatically tested and prepared for a release to production. The deployment to production is a manual step. The team decides when to deploy.

- **Benefit:** Reduces the risk of deployment errors and allows for smoother releases by always having code in a deployable state.

"Continuous Delivery ensures your code is always ready to go live"

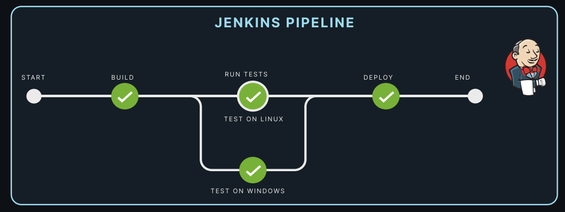
CONTINUOUS DEPLOYMENT (CD):
Extends Continuous Delivery by automatically deploying every code change that passes all stages of the production pipeline. The deployment to production is automatic without manual intervention.

- **Benefit:** Speeds up the deployment process, allowing for rapid delivery of new features and fixes to users.

"Continuous Deployment ensures your latest code changes are always live"

WHAT IS JENKINS?

Think of Jenkins as an assistant that helps developers by automatically running the steps we discussed earlier, from coding to deploying the application to a server, without any manual intervention. With these steps automated, it is like a **recipe for Jenkins to follow.** This is what we call a **Jenkins Pipeline**.



To be proficient in creating, maintaining, and troubleshooting Jenkins Pipelines, you'll need to learn how to define these pipelines using Groovy, the scripting language used by Jenkins. Mastering this skill will enable you to automate complex workflows and ensure smooth deployments.
To gain these essential skills, check out our course

Jenkins

TAUGHT BY:
Michael Levan

While Jenkins is one of the most popular tools for automating, there are many other CI/CD tools available, each with its own specific features:

- **GitLab CI , Travis CI, CircleCI, Github Actions , Azure DevOps**

Understanding how to automate the deployment of software to a server is crucial, but in real-world scenarios, companies rarely deploy software directly to the server that real end-users access. These servers, known as **Production servers or Production environments**, require careful handling.

We discussed running software on servers. Now, you need to know our software typically runs on at least three main servers for three purposes:

Development
Where developers create and modify software

Test
(Or Staging)
Where software is tested for functionality and performance.

Production
Where the software reaches users.

Deploying software across these environments can be automated with tools like Jenkins. However, manually creating and configuring these environments is complex, time-consuming, and prone to errors.

Thankfully, we can automate the creation and configuration of these servers too. One popular tool for creating (provisioning) infrastructure is **Terraform**.

However, setting up the servers doesn't stop at just creating them. To run our software applications, we need to configure the servers with the necessary dependencies, supporting libraries, and utilities. This configuration can be automated using tools like **Ansible, Chef, or Puppet.**

Infrastructure Creation with Terraform Scripts:

- You write Terraform scripts to define the infrastructure you need.
- You run these scripts, and Terraform creates the virtual machines, sets up networking, and creates databases as specified.

Configuration with Ansible Playbooks:

- After Terraform has created the infrastructure, you use Ansible to configure it.
- Ansible scripts called as 'playbooks' are run to install software on the virtual machines, configure settings, setting up users, and ensure everything is ready for use.

Terraform Basics Training Course

TAUGHT BY:
Vijin Palazhi

Learn Ansible Basics - Beginners Course

TAUGHT BY:
Mumshad Mannambeth

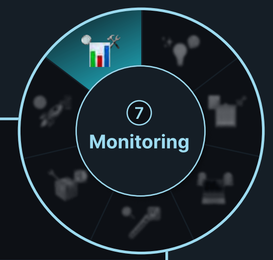
After creating infrastructure with Terraform, Ansible can be used to configure it, making them a powerful combination. Using both tools together offers numerous benefits.

EASE OF REPLICATION:
Easily replicate infrastructure and configurations in different environments.

DISASTER RECOVERY:
Quickly recover systems by re-running scripts to restore previous states.

CONSISTENCY:
Ensure consistent infrastructure and configurations, reducing errors.

AUTOMATION:
Automate tasks to save time and reduce manual work.



Once your cool software application is deployed and publicly accessible to end-users, it's important to remember that new issues can still arise.
These might include performance bottlenecks or new problems we call bugs (as you already know from the first part of this book).

Why Monitoring is Important

To ensure your application runs smoothly and provides a good experience for users, continuous monitoring is essential. Monitoring helps you detect and fix issues quickly before they affect your users.

What Needs to be Monitored

As a DevOps engineer, you should be familiar with setting up a mechanism to continuously monitor:

1. **Your Software Application:** Keep an eye on how the application performs, track user activities, and identify any errors.
2. **Kubernetes Cluster:** Monitor the health and performance of the Kubernetes cluster that runs your application.
3. **Infrastructure:** Keep track of the underlying servers, databases, and network that support your Kubernetes cluster.

You do not need to do this totally manually, there are tools to help you.

Prometheus is one of these tools that collects and stores metrics from your application and infrastructure. Metrics are numerical data that indicate how well your application and systems are performing.

You can set up Prometheus to collect data like response times, error rates, and resource usage (CPU, memory) from your application and servers.

This data is stored and can be queried to understand how your system is performing over time.

Grafana visualizes the data collected by Prometheus, turning raw metrics into interactive graphs and dashboards. You can create dashboards in Grafana to display key metrics from Prometheus. For example, you might have a dashboard that shows the average response time of your application, the number of errors per minute, and the current CPU usage of your servers. This helps you quickly identify any issues and understand the overall health of your system.

Another very popular monitoring toolchain is the ELK Stack.

ELK Stack (Elasticsearch, Logstash, Kibana):
You can use the ELK Stack to aggregate logs from various parts of your system. For example, if your application logs errors or important events, Logstash can collect these logs and send them to Elasticsearch. You can then use Kibana to search through the logs and create visualizations, such as a graph of error occurrences over time, helping you identify patterns and troubleshoot issues quickly.

Congratulations on reaching the final part of your journey to become a DevOps Engineer!

Remember, one of the key principles in DevOps is to prefer automation over manual tasks.

As a DevOps engineer, you'll work closely with developers and operations teams. You'll often need to automate repetitive tasks to save time and reduce errors. Some examples of tasks you might automate include:

ALERTING AND NOTIFICATIONS:
Notifying the team of issues or critical events.

SECURITY CHECKS:
Running automated scans to detect vulnerabilities

BACKUPS AND RESTORES:
Saving and restoring important data regularly.

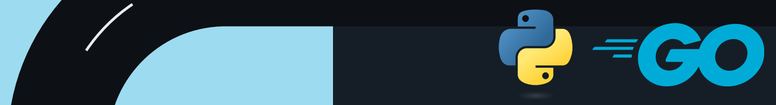
SOFTWARE UPDATES:
Keeping software and applications up to date.

To automate these tasks, you might need to **write small applications or scripts.** Having proper knowledge of a scripting language is crucial for a DevOps engineer.

Since you already know about Linux from the first part of this ebook, **learning shell scripting is a natural next step.**

Shell scripting allows you to execute a series of commands written in a script, automating repetitive tasks to save time and reduce errors. **Bash scripting is a type of shell scripting specific to the Bash shell, commonly used in Linux.**

Remember, in the first part of this book, we covered the **Linux CLI (Command Line Interface) as a mandatory topic.**



Additionally, learning more robust and powerful languages like Python or Golang can be very beneficial for your long-term career. These languages offer greater flexibility and capabilities for automating complex tasks and developing custom solutions.

While learning the basics in the first part of this book, you can also explore Python and Golang. These powerful programming languages are perfect for automating complex tasks. You can learn them in parallel, even though you won't be coding full applications as a DevOps engineer.

Python Basics

TAUGHT BY:
Lydia Hallie

Golang

TAUGHT BY:
Priyanka Yadav

Shell Scripts for Beginners

TAUGHT BY:
Mumshad Mannambeth & Vijin Palazhi

📖 Glossary of Terms 📖

BUILD & PACKAGE THE APPLICATION:
The process of transforming the source code into runnable applications and gathering them with all required resources and dependencies.

CLOUD PROVIDER:
A company that offers cloud computing services such as storage, processing power, and software applications over the internet.

CLOUD SERVER:
A virtual server hosted on the internet, allowing for easy scaling and management.

CODING:
The act of writing computer programs using programming languages.

DAEMONLESS CONTAINER:
A container that runs independently without needing a constantly running management program.

DATA:
Information processed or stored by a computer. This can include text, numbers, images, and more.

DATA STRUCTURES AND ALGORITHMS:
Techniques and methods used in programming to store, organize, and manipulate data efficiently.

DEPLOYMENT GUIDE:
A document or set of instructions detailing how to deploy software application on a specific environment or infrastructure.

DEPLOYING SOFTWARE APPLICATION:
The process of making a software application ready for use by installing, configuring, and launching it on the target environment.

DEVELOPERS / DEVELOPMENT TEAM:
Responsible for creating and maintaining software applications.

ENCRYPTION:
The process of converting data into a coded format to prevent unauthorized access.

END-USER:
The person or group who will ultimately use the software application.

ENVIRONMENT:
The setup of hardware, software, network resources, and configurations in which applications are deployed, tested, and run.

FIREWALL:
A security system that checks and controls data coming in and out of a network to keep it safe.

FTP:
A method to transfer files between computers over the internet.

HTTP:
The system that allows your web browser to load and display web pages from the internet.

HTTPS:
A secure version of HTTP that keeps your information safe while browsing.

INFRASTRUCTURE AS A SERVICE (IAAS):
A cloud computing service model that provides virtualized computing resources over the internet.

IT INFRASTRUCTURE / INFRASTRUCTURE:
The hardware, software, network resources, and services necessary for the operation and management of an enterprise IT environment.

MONITOR AND OPERATE:
The process of continuously observing the performance of a software application and managing its operations to ensure it runs smoothly.

OCI CONTAINERS:
Containers that follow the Open Container Initiative standards for runtime and image specifications to ensure compatibility and reliability.

OPERATIONS TEAM:
The team responsible for maintaining and managing the IT infrastructure and software applications

PERFORMANCE IN A SOFTWARE:
A measure of how well a software application functions, typically in terms of speed, responsiveness, and resource utilization.

PLATFORM AS A SERVICE (PAAS):
A cloud computing service model that provides a platform allowing customers to develop, run, and manage applications without dealing with the underlying infrastructure.

PROGRAMMING LANGUAGE:
A special language used to write instructions that a computer can follow to perform tasks and solve problems.

PUBLIC SERVER:
A server that is accessible over the internet and can be used by multiple clients or users.

RUNNING SOFTWARE:
The state of a software application when it is executing and performing its designed tasks.

SDLC (SOFTWARE DEVELOPMENT LIFE CYCLE):
A process used by the software industry to design, develop, and test high-quality software. It involves several stages: planning, design, development, testing, deployment, and maintenance.

SERVER/SERVER COMPUTER:
A computer or system that provides resources, data, services, or programs to other computers, known as clients, over a network.

SOFTWARE:
Programs and other operating information used by a computer.

SOFTWARE APPLICATION:
Also known as Application or Software, it is a program or group of programs designed for end-users to perform specific tasks.

SSH:
A secure way to access and control computers remotely over the internet.

TCP/IP:
The basic language that computers use to communicate over the internet.

DevOps for Absolute Beginners, 2024