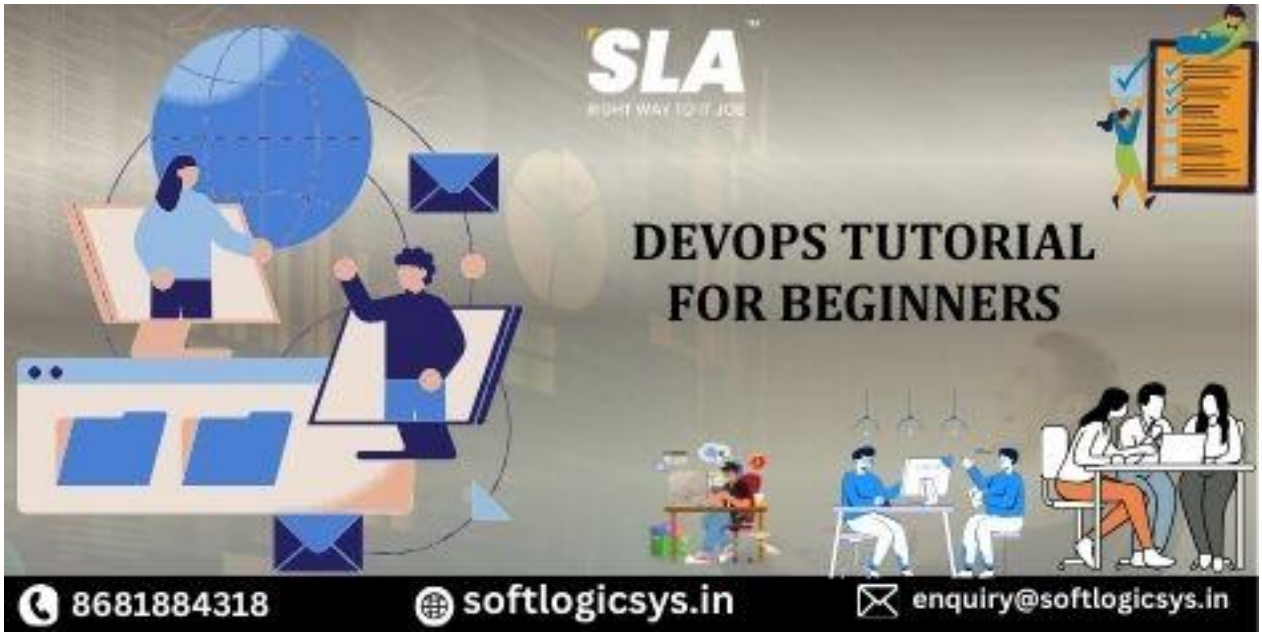

DevOps Tutorial for Beginners

Introduction

Battling the confusing DevOps toolchain or unclear about how Development and Operations teams collaborate? Feeling overwhelmed by continuous integration? Well, simplify DevOps for a beginner in this tutorial. We will explain in detail the philosophy involved and the main practices that include CI/CD and automation so that you can bridge the gap between coding and deployment to production. Ready for building modern software pipelines? Review the full **DevOps course syllabus** now!

Why Students or Freshers Learn DevOps?

For students and freshers, mastering DevOps is a route to sought-after tech careers:

- **Critical Industry Movement:** Practically every large company uses DevOps to deliver software faster and more reliably.
- **High Earning Potential:** DevOps engineers are one of the highest-paid in view of how critical their jobs are.

- **Automation Focus:** You learn automation tools like Ansible, Terraform, Jenkins, which are essential in modern IT infrastructure.
- **Cross-Functional Skill:** Develop expertise in coding (development) and infrastructure (operations) to be versatile.
- **Cloud Native Demand:** DevOps is essential when operating with cloud platforms: AWS, Azure, GCP.

Land your dream job! Download our essential **DevOps Interview Questions and Answers** guide now!

Step-by-Step DevOps Tutorial for Beginners

The following tutorial will practically outline, in a step-by-step manner, how a beginner can understand and implement the core concepts of DevOps, only focusing on the essential tools and workflows needed for a modern software delivery pipeline.

Step 1: Setup and Installation of Core Tools

DevOps operates around tools for automatization. In terms of immediate need, a beginner will do well by learning Git, Docker, and a basic CI/CD environment.

1.1. Git (Version Control System)

Git is non-negotiable for collaborative development and tracking code changes.

1. **Installation:** Download Git from an official site for your OS and follow the installation steps (Windows, macOS, or Linux).
2. **Configuration:** Open your terminal and set your global user name and email.

git config –global user.name “Your Name”

git config –global user.email “your.email@example.com”

3. **Basic Workflow:**

- **Initialize a Repository:** *git init*
- **Stage Changes:** *git add*. (stages all changes in the current directory)
- **Commit Changes:** *git commit -m “Initial commit message”*

1.2. Docker (Containerization)

It wraps your applications and their dependencies into a deployable container that runs exactly the same in different environments, whether it’s local, testing, or production.

1. **Installation:** Download and install Docker Desktop for your operating system. Docker Desktop contains the Docker Engine, Docker CLI, and Docker Compose.
2. **Verification:** Open terminal and execute:

docker –version

3. **First Consider (Hello World):**

docker run hello-world

This command pulls the hello-world image from **Docker Hub** and runs it in a container.

Step 2: Containerizing an Application

The first practical DevOps skill is to take a simple application and put it into a container using a Dockerfile.

2.1. Developing a Simple Application in Python

In the new directory, create two files: *app.py* and *requirements.txt*.

requirements.txt (Defines dependencies)

```
flask
```

app.py (A simple web server using Flask)

```
from flask import Flask
import os
app = Flask(__name__)
@app.route('/')
def hello():
    # Environment variable for dynamic greeting
    name = os.environ.get('NAME', 'World')
    return f'Hello, {name}! This is a containerized Python app.'
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

2.2. Creating the Dockerfile

The Dockerfile contains instructions for building the image.

Dockerfile

```
# Stage 1: Define the base image (a lightweight Python environment)
FROM python:3.10-slim
```

 # Set the working directory inside the container

WORKDIR /app

 # Copy the requirements file and install Python dependencies

 # This step is often cached, speeding up builds

COPY requirements.txt .

RUN pip install –no-cache-dir -r requirements.txt

 # Copy the application code into the container

COPY . .

 # Expose the port the app runs on

EXPOSE 5000

 # Set an environment variable (used by app.py)

ENV NAME=DevOps_Learner

 # Command to run the application when the container starts

CMD [“python”, “app.py”]

2.3. Container Build and Run

1. **Build the Image:** The. specifies the current directory as the build context. The -t tags the image with a name.

docker build -t flask-devops-app:v1 .

2. **Run the Container:**
    - **-d:** Runs the container in detached mode (background).
    - **-p 8080:5000:** Port mapping, allows mapping of internal container port 5000 to the outside host at port 8080.

docker run -d -p 8080:5000 –name devops-web-app flask-devops-app:v1

3. **Test:** Open your web browser or use curl: http://localhost:8080
4. **Stop and Remove:**

docker stop devops-web-app

docker rm devops-web-app

Step 3: Continuous Integration

Continuous Integration (CI) is when you automatically create and test your code every time someone commits changes to the shared repository. We're going to simulate that using GitHub Actions.

3.1. Version Control and Remote Repository

1. **Create GitHub Repo:** Create a new repository on GitHub.
2. **Link Local to Remote:**

git remote add origin <your_github_repo_url>

git branch -M main

git push -u origin main

3.2. Configure GitHub Actions Workflow

Create a directory `*.github/workflows*` and add a YAML file named `*python_ci.yml*`.

```
.github/workflows/python_ci.yml

name: Python CI Build

# Trigger this workflow on every push to the main branch

on:

  push:

    branches: [ main ]

  pull_request:

    branches: [ main ]

jobs:

  build:

    runs-on: ubuntu-latest # The environment (runner) to execute the job

    steps:

    – uses: actions/checkout@v4 # Step 1: Checkout the code

    – name: Set up Python 3.10 # Step 2: Set up the Python environment

      uses: actions/setup-python@v5

      with:

        python-version: ‘3.10’
```

```
  – name: Install dependencies # Step 3: Install dependencies
    run: |
      python -m pip install –upgrade pip
      pip install -r requirements.txt
  – name: Run unit tests # Step 4: Execute tests (replace 'test.py' with your actual test command)
    run: |
      # In a real project, this would run tests like: pytest
      echo "Tests passed successfully (Simulated)"
  – name: Build Docker Image # Step 5: Build the container image
    run: |
      docker build . –file Dockerfile –tag my-ci-image:latest
```

How it works: When you push code, GitHub notices this file, provisions an Ubuntu runner, and runs these steps: checking out code, setting up Python, installing dependencies, running tests, and finally **building the Docker image**.

Step 4: Continuous Delivery

Continuous Delivery (CD) expands upon that by automatically deploying the application, once successfully tested and built into an image, to either a staging or production environment.

4.1. The Push to Registry

Instead of deploying directly, the image is pushed to a centralized container registry such as Docker Hub or Amazon ECR. It is at this stage that the output of your CI job, which is the image, becomes the input for your CD job.

Add to your *python_ci.yml* job (Step 3.2):

 # … previous steps (Setup Python, Install dependencies, Run tests)

– name: Log in to Docker Hub # Requires storing Docker Hub credentials as GitHub secrets

uses: docker/login-action@v3

with:

username: ${{ secrets.DOCKER_USERNAME }}

password: ${{ secrets.DOCKER_PASSWORD }}

– name: Build and push Docker image # Tag and push the image

id: docker_build

uses: docker/build-push-action@v5

with:

context: .

push: true

tags: ${{ secrets.DOCKER_USERNAME }}/flask-devops-app:latest

Note: You need to set up DOCKER_USERNAME and DOCKER_PASSWORD in your GitHub repository secrets.

4.2 Simplified CD: Deploy to Cloud/Server

True CD would require a target environment such as AWS EC2 or Kubernetes. In this beginner's tutorial, we are going to simulate the CD step that would involve an IaC tool such as Terraform or a configuration management tool such as Ansible.

Concept (Post-push): A second CD pipeline job would be triggered by the image push. This job would typically use Ansible to:

1. SSH into the target server, for example a DigitalOcean droplet.
2. Pull the newly pushed Docker image.
3. Stop the old container.
4. Start a new container using the most recent image.

The whole automated loop—Code, Commit, Build, Test, Push Image, Deploy—is the CI/CD Pipeline.

Step 5: Infrastructure as Code (IaC) with Terraform

Infrastructure as Code is the management or provisioning of computing infrastructure through configuration files rather than a manual process. The leading tool to do this is Terraform.

5.1. Installation

Download and install the Terraform CLI from HashiCorp.

5.2. A Simple Terraform Configuration

Terraform uses HashiCorp Configuration Language – HCL. Here is a generic structure for defining a resource.

main.tf – Define a cloud provider and a resource, such as Virtual Private Cloud.

```
# Define the required cloud provider (e.g., AWS)
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
  }
}
# Provider configuration
provider "aws" {
  region = "us-east-1"
}
# Define a resource to be created (e.g., a simple VPC)
resource "aws_vpc" "devops_vpc" {
  cidr_block = "10.0.0.0/16"
```

```
  tags = {
    Name = "DevOps_Beginner_VPC"
  }
}
```

5.3. Basic Terraform Workflow

1. **Initialize:** Downloads required provider plugins.

terraform init

2. **Plan:** Generates an execution plan, displaying explicitly what resources will be created, modified, or destroyed before making any changes.

terraform plan

3. **Apply:** This is where the planned activities are executed and the infrastructure is built on the cloud.

terraform apply

4. **Destroy:** Removes all resources managed by this config. Use with caution!

terraform destroy

Proficiency as a DevOps Engineer requires deeperening your skills in the following:

- **Advanced CI/CD:** Check out Jenkins, GitLab CI, or cloud-specific offerings such as AWS CodePipeline or Azure DevOps.

- **Configuration Management:** Learn how to use Ansible for configuring and managing operating systems and applications inside the infrastructure provisioned with Terraform.
- **Orchestration:** Mastering Kubernetes to run containerized applications at scale.
- **Monitoring and Logging:** implement observability tools such as Prometheus, Grafana, and ELK Stack.

Ready to debug pipeline failures and automate complex infrastructure? Download our **DevOps Challenges and Solutions** guide to test your skills and advance your career!

Real Time Examples for DevOps Tutorial for Learners

These examples demonstrate core practices and tools utilized by DevOps teams in efficiently and reliably delivering software:

Automated Continuous Integration/Continuous Deployment (CI/CD) for a Microservice

- **Goal:** To have automatic building, testing, and deployment of a small service, say a login API, on every developer commit.
- **Tools:** Git for version control, Jenkins or GitHub Actions for the CI/CD orchestrator, and Docker for containerization.

** **Process:***

1. *Developer pushes code to the `main` branch (**Git**).*
2. *The pipeline is triggered (**Jenkins**).*
3. *The code is built, dependencies are installed, and unit tests run.*
4. *A **Docker image** is created and pushed to a registry (e.g., Docker Hub).*

5. *The new image is automatically deployed to a staging environment (CD).*

** **Impact:** Reduces manual errors, ensures rapid feedback on code quality, and speeds up time-to-market.*

Infrastructure Provisioning with Terraform-IaC

- **Goal:** The whole cloud infrastructure, including all servers, networks, and load balancers, which an application requires in test and production environments, should be reliably and repeatedly established.
- **Tools:** Terraform, Infrastructure as Code, and Cloud Provider (AWS, Azure, GCP).

** **Process:***

1. *Infrastructure requirements are written in **Terraform** configuration files.*

2. *The engineer runs `terraform plan` to review changes.*

3. *The engineer runs `terraform apply`, and Terraform interacts with the cloud provider's API to provision the resources exactly as defined.*

** **Impact:** Eliminates configuration drift, ensures environments are identical (testing replicates production), and allows infrastructure to be version-controlled.*

Configuration Management for Server Patching (Ansible)

- **Goal:** Manage hundreds of virtual servers and enforce their state to have the proper software installed on them, for example, the correct version of Python installed in them, along with security patches.
- **Tools:** Ansible (Configuration Management) and SSH.
- **Process:**

    - A developer creates an Ansible Playbook-a YAML file that describes a desired state, such as “Python 3.10 should be installed.”
    - Ansible uses SSH to log into the target servers and run the configured tasks in the playbook.
    - Ansible is idempotent: it only alters the servers if they are not in the desired state.
- **Impact:** Ensures security compliance, reduces manual server maintenance, and guarantees consistency across the entire fleet.

Ready to begin automating? Here’s a curated list of **DevOps project ideas** to get you started building your portfolio.

FAQs About DevOps Tutorial for Beginners

1. How do I start learning DevOps?

To learn DevOps, the focus should be on three core areas:

1. **Infrastructure:** Linux, Cloud basics – AWS/Azure/GCP;
2. **Automation Tools:** Git, Docker, Kubernetes; and
3. **CI/CD practices:** Jenkins or GitHub Actions.

Create small projects in which to immediately apply concepts.

2. What are the 7 C’s of DevOps?

The 7 C’s represent the values and principles that guide the DevOps culture: Culture, Collaboration, Continuous Improvement, Continuous Integration, Continuous Delivery, Continuous Deployment, and Continuous Monitoring. These put a focus on communication and automation across the entire software lifecycle.

3. Is SQL needed for DevOps?

This isn’t a primary skill; however, basic SQL knowledge is helpful for DevOps. You may need it to manage application databases, monitor application health by querying log data, or set up database deployments in the CI/CD pipeline.

4. What is DevOps’ job salary?

DevOps Salaries in India are very competitive because of the high demand. The average compensation for a senior DevOps Engineer in the US ranges between $120,000 and $160,000+ every year. In India, it is in the range of ₹10 LPA to ₹25 LPA for experienced engineers, depending on the city and the skill set.

5. Can I learn DevOps in 3 months?

You can master the basics of Git, Docker basics, Linux, and a general overview of CI/CD in 3 months. Job readiness, defined as deep expertise in Cloud, Kubernetes, and IaC-Terraform, requires 6 to 12 months of focused, project-based learning.

6. Will AI replace DevOps?

While AI will not replace the DevOps engineer altogether, it certainly complements their role. AI will be automating more repetitive tasks of monitoring, basic troubleshooting, and resource optimization, thus freeing engineers to work at higher levels of system architecture, security, and complexity management.

7. What are the 7 phases of DevOps?

The 7 phases make up the DevOps loop or infinity loop. These are: 1. Plan, 2. Code, 3. Build, 4. Test, 5. Release, 6. Deploy, and 7. Monitor/Operate. It emphasizes continual feedback and collaboration within the entire lifecycle of software development.

8. Is DevOps job stressful?

DevOps is stressful, in particular during big deployments, security incidents, or on-call duties because of production outage management. However, the focus is to decrease long-term stress caused by manual and repetitive tasks or unexpected failures by emphasizing automation and stability in infrastructure.

9. What is SDLC in DevOps?

SDLC in DevOps is agile and an ongoing cycle. Unlike traditional sequential models, DevOps integrates Operations practices throughout each phase of the SDLC, driving speed, iteration, and focus on reliable delivery of software.

10. What is the salary of DevOps engineer in TCS?

The **salary of a DevOps Engineer in TCS** ranges from ₹6 LPA for a fresher to ₹15 LPA or more for a highly experienced professional. The exact amount heavily depends on years of experience, specific cloud certifications, and city location.

Conclusion

You've gained the basics and necessary tools for DevOps, including Git and Docker, along with how the logic works behind CI/CD pipelines. With such knowledge, you are right at the heart of modern software delivery, which is highly automated and collaborative. Your career will be future-proof if you stick to the DevOps mindset. Ready to master advanced cloud platforms and orchestration tools such as Kubernetes? Enroll for our detailed **DevOps course in Chennai**!