DevOps Tools for Azure and AWS

1. Version Control System (VCS)

- **Azure**: Azure Repos (Git)
- **AWS**: AWS CodeCommit
- **Common Tool**: Git, GitHub, GitLab, Bitbucket

2. CI/CD (Continuous Integration/Continuous Deployment)

- **Azure**: Azure DevOps (Pipelines), GitHub Actions
- **AWS**: AWS CodePipeline, AWS CodeBuild, AWS CodeDeploy
- **Common Tools**: Jenkins, GitLab CI/CD, CircleCI

3. Configuration Management

- **Azure**: Azure Automation, Desired State Configuration (DSC)
- **AWS**: AWS Systems Manager, AWS OpsWorks
- **Common Tools**: Ansible, Chef, Puppet, SaltStack

4. Infrastructure as Code (IaC)

- **Azure**: Azure Resource Manager (ARM) Templates, Bicep
- **AWS**: AWS CloudFormation
- **Common Tools**: Terraform, Pulumi

5. Containerization & Orchestration

- **Azure**: Azure Kubernetes Service (AKS), Azure Container Instances (ACI)
- **AWS**: Amazon Elastic Kubernetes Service (EKS), Amazon Elastic Container Service (ECS)
- **Common Tools**: Docker, Kubernetes, Helm

6. Monitoring & Logging

- **Azure**: Azure Monitor, Log Analytics, Application Insights
- **AWS**: AWS CloudWatch, AWS X-Ray
- **Common Tools**: Prometheus, Grafana, ELK Stack (Elasticsearch, Logstash, Kibana), Splunk, Datadog

7. Security & Compliance

- **Azure**: Azure Security Center, Azure Policy, Key Vault
- **AWS**: AWS Security Hub, AWS Identity and Access Management (IAM), AWS Secrets Manager
- **Common Tools**: HashiCorp Vault, SonarQube, Aqua Security, Snyk

8. Artifact Repository & Dependency Management

- **Azure**: Azure Artifacts
- **AWS**: AWS CodeArtifact
- **Common Tools**: JFrog Artifactory, Nexus Repository

9. Cloud Networking & Load Balancing

- **Azure**: Azure Load Balancer, Azure Front Door, Traffic Manager
- **AWS**: AWS Elastic Load Balancer (ELB), Route 53

10. Serverless & Function-as-a-Service (FaaS)

- **Azure**: Azure Functions
- **AWS**: AWS Lambda
- **Common Tools**: Serverless Framework

11. Service Mesh

- **Azure**: Open Service Mesh (OSM), Istio
- **AWS**: AWS App Mesh
- **Common Tools**: Istio, Linkerd

12. Database Management & DevOps

- **Azure**: Azure SQL Database, Cosmos DB
- **AWS**: AWS RDS, DynamoDB
- **Common Tools**: Flyway, Liquibase, dbt

13. Messaging & Event-Driven Architecture

- **Azure**: Azure Service Bus, Event Grid, Event Hub
- **AWS**: AWS SNS, AWS SQS, AWS EventBridge
- **Common Tools**: RabbitMQ, Apache Kafka

14. FinOps & Cost Management

- **Azure**: Azure Cost Management + Billing
- **AWS**: AWS Cost Explorer, AWS Budgets
- **Common Tools**: CloudHealth, Kubecost, Spot.io

15. Collaboration & Documentation

- **Azure**: Azure DevOps Boards, Wiki
- **AWS**: AWS WorkDocs, AWS Chime
- **Common Tools**: Confluence, Notion, Jira, Slack, Microsoft Teams

Step-by-Step Guide to Full Stack Project Development with DevOps (Hands-on Practice)

1. Planning & Requirement Analysis

Example: Building a Todo App

- Define project goals: Create a web-based Todo App with user authentication and real-time task updates.
- Choose tech stack:
  - **Frontend:** React.js
  - **Backend:** Node.js with Express
  - **Database:** MongoDB (NoSQL)
  - **DevOps Tools:** GitHub, Docker, Kubernetes, Jenkins, AWS/Azure
- Set up a roadmap: Define milestones for development, testing, and deployment.

2. Version Control & Repository Setup

Hands-on Practice

- Create a GitHub repository:
- git init
- git remote add origin https://github.com/yourusername/todo-app.git
- Create branches for different features:
- git checkout -b frontend
- git checkout -b backend

3. Frontend Development

Hands-on Practice

- Set up React.js project:
- npx create-react-app todo-frontend
- cd todo-frontend
- npm start
- Install dependencies:
- npm install axios react-router-dom
- Implement UI and connect API endpoints.

4. Backend Development

Hands-on Practice

- Set up Express server:
- mkdir todo-backend && cd todo-backend
- npm init -y
- npm install express mongoose cors dotenv
- Create server.js file and define routes.

5. Database Management

Hands-on Practice

- Set up MongoDB using Docker:
- docker run -d -p 27017:27017 --name mongo-db mongo
- Create schema and models in models/todo.js.

6. Infrastructure as Code (IaC)

Hands-on Practice

- Write Terraform script to deploy resources in AWS:
- resource "aws_instance" "web" {
- ami = "ami-12345678"
- instance_type = "t2.micro"
- }
- Apply changes:
- terraform init
- terraform apply

7. CI/CD Pipeline Setup

Hands-on Practice

- Create a Jenkinsfile for automation:
- pipeline {
- agent any
- Stages {
- stage('Build') { steps { sh 'npm install' } }
- stage('Test') { steps { sh 'npm test' } }
- stage('Deploy') { steps { sh 'docker build -t todo-app .' } }
- }
- }
- Push Jenkinsfile to repository.

8. Containerization & Orchestration

Hands-on Practice

- Create a Dockerfile for backend:
- FROM node:14
- WORKDIR /app
- COPY . .
- RUN npm install

- CMD ["node", "server.js"]
- Build and run container:
- docker build -t todo-backend .
- docker run -d -p 5000:5000 todo-backend

9. Monitoring & Logging

Hands-on Practice

- Deploy Prometheus and Grafana for monitoring:
- docker run -d -p 3000:3000 grafana/grafana

10. Security & Compliance

Hands-on Practice

- Use environment variables for sensitive data:
- export DB_PASSWORD=mysecurepassword
- Store secrets securely in AWS Secrets Manager.

11. Deployment & Scaling

Hands-on Practice

- Deploy on AWS using ECS:
- aws ecs create-cluster --cluster-name todo-cluster

12. Cost Optimization & FinOps

Hands-on Practice

- Set up AWS Cost Explorer to track expenses.

13. Documentation & Collaboration

Hands-on Practice

- Maintain documentation in Notion or Confluence.
- Use Jira for tracking issues.

By following these steps with hands-on examples, you can build, deploy, and manage a full-stack application with DevOps practices.

Here are your **DevOps Engineer Notes** with your question included:

A DevOps Engineer's work doesn't just start after the development is complete—it runs in parallel with development to ensure smooth integration, deployment, and maintenance. Here's how it works:

📌 **DevOps Engineer Notes**

❓ **When does a DevOps Engineer's work start?**

✅ **DevOps starts alongside development**, not after coding is completed.

1️⃣ Planning & Setup (Before Coding Starts)

- Set up **Git repositories** (GitHub, GitLab, Bitbucket) and branching strategy.
- Configure **CI/CD pipelines** for automatic testing and deployment.
- Choose **cloud services (AWS, Azure, GCP)** and infrastructure tools (**Docker, Kubernetes, Terraform**).
- Define security policies, monitoring, and logging from the beginning.

👉 **Example:**

- Create a **GitHub repo** with a **GitHub Actions pipeline** for CI/CD before development starts.

2️⃣ During Development (Continuous Integration & Testing)

- Automate **builds and testing** (JUnit, Jest, Selenium, PyTest).
- Use **Docker** for containerized development to ensure consistency.
- Monitor code quality using **SonarQube, ESLint, or Prettier**.
- Implement **feature flagging** for controlled feature releases.

👉 **Example:**

- When a developer **pushes code**, a **CI/CD pipeline runs unit tests** and deploys it to a **staging server**.

3️⃣ Deployment (After Development)

- Deploy applications on **AWS (EC2, Lambda), Azure (App Service), or Kubernetes clusters**.
- Use **Terraform or Ansible** for Infrastructure as Code (IaC).
- Implement **auto-scaling, load balancing, and blue-green deployments**.

👉 **Example:**

- Deploy the backend **on AWS EC2** and frontend **on S3 with CloudFront** for better performance.

4️⃣ Post-Deployment (Monitoring & Maintenance)

- Monitor application performance using **CloudWatch, Prometheus, Grafana**.
- Set up **alerts for failures and auto-recovery mechanisms**.
- Improve security by **patching vulnerabilities and managing IAM roles**.
- Automate **backup and disaster recovery strategies**.

👉 **Example:**

- If the API **starts failing**, an **alert is triggered** and logs help **identify the issue** quickly.

🚀 **Conclusion: DevOps is Continuous!**

✅ **DevOps starts when development starts**—not just after coding.
✅ **It ensures smooth development, deployment, and post-production maintenance.**
✅ **Automates workflows to improve efficiency and reliability.**

As a **DevOps Engineer**, your work **begins on Day 1 and never stops!** 🔥

🚀 **Real-Time DevOps Workflow with a Full-Stack Project**

💡 **Project: Full-Stack Todo App Deployment**

We'll take your **Full-Stack Todo App** (React + Node.js + MongoDB) and go through every **DevOps step** from development to deployment.

🛠️ **1. Initial Setup (Before Development)**

💼 **DevOps Role**: Set up repositories, CI/CD pipelines, and cloud infrastructure **before development starts**.

🔹 **Step 1: Set Up Version Control & Branching Strategy**

- **Create a GitHub Repository** for code collaboration.
- **Branching Strategy**:
  - main → Stable production branch
  - dev → Active development branch
  - Feature branches → For new features

✅ **Example:**

```
git init
git remote add origin https://github.com/your-repo/todo-app.git
git checkout -b dev  # Switch to dev branch
```

🔹 **Step 2: Set Up CI/CD Pipeline**

- Use **GitHub Actions** to automate testing & deployment.
- Create .github/workflows/main.yml for CI/CD.

✅ **Example: CI/CD Workflow (GitHub Actions)**

```
name: CI/CD Pipeline

on: [push, pull_request]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
  - uses: actions/checkout@v2
  - name: Install Dependencies
      run: npm install
  - name: Run Tests
      run: npm test
```

```
  - name: Deploy to Staging
    if: github.ref == 'refs/heads/dev'
    run: echo "Deploying to staging..."
```

👷 2. During Development (Continuous Integration & Testing)

💼 **DevOps Role**: Automate builds, run tests, and ensure secure coding practices.

🔹 **Step 3: Containerize the Application (Docker)**

- Ensure developers **work in the same environment** using Docker.
- Create a Dockerfile for backend & frontend.

✅ **Example: Backend Dockerfile**

```
FROM node:18
WORKDIR /app
COPY package.json .
RUN npm install
COPY . .
CMD ["node", "server.js"]
EXPOSE 5000
```

✅ **Example: Frontend Dockerfile**

```
FROM node:18
WORKDIR /app
COPY package.json .
RUN npm install
COPY . .
RUN npm run build
CMD ["npx", "serve", "-s", "build"]
EXPOSE 3000
```

🔹 **Step 4: Automate Testing**

- Use **Jest for frontend** and **Mocha/Chai for backend**.
- Ensure **unit tests run in the CI/CD pipeline**.

✅ **Example: Backend Test (Mocha/Chai)**

javascript

```
const request = require('supertest');
const app = require('../server');

describe('GET /todos', () => {
  it('should return all todos', async () => {
    const res = await request(app).get('/todos');
  expect(res.status).toBe(200);
 });
});
```

🚀 **3. Deployment (After Development)**

💼 **DevOps Role**: Deploy app on **AWS (EC2, S3, RDS)** or **Azure (App Service, CosmosDB)**.

🔹 **Step 5: Deploy Backend to AWS (EC2)**

1⃣ **Create an EC2 instance**
2⃣ **Install Node.js & MongoDB**
3⃣ **Pull the code & run it**

✅ **Example: SSH into EC2 & Deploy**

```
ssh -i my-key.pem ubuntu@ec2-ip-address
sudo apt update && sudo apt install -y nodejs npm
git clone https://github.com/your-repo/todo-app.git
cd todo-app
npm install
node server.js
```

🔹 **Step 6: Deploy Frontend to AWS S3**

1⃣ **Build the React app**
2⃣ **Upload to S3 bucket**
3⃣ **Enable CloudFront for global access**

✅ **Example: Deploy React App to S3**

```
npm run build
aws s3 cp build s3://my-todo-app-bucket --recursive
```

📊 **4. Post-Deployment (Monitoring & Scaling)**

💼 **DevOps Role**: Ensure uptime, logging, monitoring, and auto-scaling.

🔹 **Step 7: Set Up Monitoring (CloudWatch & Prometheus)**

- Use **AWS CloudWatch** for logs & alerts.
- Use **Prometheus & Grafana** for monitoring metrics.

✅ **Example: Monitor Logs with CloudWatch**

```
aws logs create-log-group --log-group-name "/todo-app/backend"
aws logs create-log-stream --log-group-name "/todo-app/backend" --log-stream-name "server-logs"
```

🔹 **Step 8: Implement Auto-Scaling**

- Use **AWS Auto Scaling Groups** for EC2 instances.
- Configure **Kubernetes (EKS) for dynamic scaling**.

✅ **Example: Kubernetes Deployment (EKS)**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: todo-app
spec:
  replicas: 3
  template:
    spec:
      containers:
        - name: backend
          image: my-todo-app-backend
```

🎯 **Conclusion: What Did We Achieve?**

✅ **Full DevOps Lifecycle**: From coding to deployment & monitoring.
✅ **CI/CD Pipeline**: Automated builds, tests & deployments.
✅ **Cloud Deployment**: Hosted on AWS EC2, S3 & Kubernetes.
✅ **Scaling & Monitoring**: Auto-scaling & logging for stability.

💥 **Now you can deploy your own Full-Stack App like a pro!** 🚀 🔥