**DevOps Syllabus**

**Introduction**

* What is DevOps?
* What is SDLC (Software Development Lifecycle)?
* Why DevOps?
* DevOps Principles
* Waterfall vs Agile vs DevOps
* DevOps Tools Overview

**Python Programming**

* Installing Python on windows
* Creating Reusable Variables and Functions
* Working with Lists, Dictionaries, and Tuples
* Python Arithmetic Operators
* Installing and Using Python Libraries
* Control structure
* files
* Python Use Cases: Web Scraper, To-Do App

**Cloud and AWS Introduction Syllabus**

1. **Introduction to Cloud Computing**
   * Definition and benefits of cloud computing
   * Different cloud service models: IaaS, PaaS, and SaaS
   * Public, private, and hybrid cloud deployment models
   * Key cloud providers in the market (AWS, Azure, GCP)
   * Cloud security and compliance considerations
2. **Introduction to AWS (Amazon Web Services)**
   * Overview of AWS and its global infrastructure
   * Key AWS services: Compute, Storage, Database, Networking, Security
   * AWS Free Tier: Getting started with AWS services
   * Introduction to AWS Management Console and CLI
   * Overview of AWS pricing models and cost management
3. **AWS Compute Services**
   * Amazon EC2: Virtual machines on AWS
   * Elastic Load Balancing (ELB) and Auto Scaling
   * AWS Lambda: Serverless computing
   * AWS Elastic Beanstalk: Platform as a Service (PaaS) for deploying applications
   * Amazon Lightsail: Simplified cloud services for small applications
4. **AWS Storage Services**
   * Amazon S3: Object storage service
   * Amazon EBS: Block storage for EC2 instances
   * Amazon EFS: Managed file storage for EC2
   * Amazon Glacier: Archival storage and long-term backup
   * AWS Storage Gateway: Hybrid cloud storage solutions

**Version Control: Git**

* Introduction to Version Control Systems
* Centralized vs Distributed Version Control
* Git Advantages
* Installing Git
* Creating a Repository
* Adding Code and Creating Commits
* Creating a GitHub Account
* Pushing Code to GitHub
* Cloning a Repo from GitHub
* Forking a GitHub Repo and Working on It

**CI/CD with Jenkins**

* Jenkins Overview and Installation
* Setting Up Authentication
* Managing Plugins from Console
* Installing GitHub Plugin from Repository
* Adding Ant/Maven Support
* Configuring Email Notifications
* Continuous Deployments Using Jenkins
* Exploring Jenkins System Configuration
* Analyzing System Logs
* Understanding Stages of CI/CD
  + Continuous Download
  + Continuous Build
  + Continuous Deployment
  + Continuous Testing
  + Continuous Delivery

**Continuous Integration with Jenkins**

* Installing Git and Jenkins GitHub Plugin
* Installing Maven on Local Machine
* Configuring Jenkins for Java, Git, and Maven
* Creating and Running Jenkins Projects
* Troubleshooting Jenkins Projects
* Source Control Polling in Jenkins
* Build Triggers in Jenkins
* Install and Configure Tomcat for Staging
* Jenkins Build Pipeline
* Parallel Jenkins Builds
* Deploying to Production

**Distributed Builds in Jenkins**

* Introduction to Distributed Jenkins Builds
* Master-Slave Setup
* Installing Jenkins Master Node in the Cloud
* Concurrent Jenkins Builds
* Labeling Jenkins Builds
* Continuous Delivery with Jenkins
* Code as Pipeline
* CI/CD Using Jenkinsfile (Pipeline as Code)
  + Scripted Pipeline
  + Declarative Pipeline
  + Multi-Branch Pipeline

**Configuration Management: Ansible**

* Installing Ansible using RPM or Python PIP
* Inventory Management in Ansible
* Ansible Modules Overview
* Running Ad-hoc Commands
* Creating Ansible Playbooks
  + Variables and Loops
  + Conditional Execution
* Using Ansible Facts for Customization
* Creating and Using Ansible Roles
* Introduction to Ansible Galaxy
* Downloading Roles from Ansible Galaxy

**Modules and Ad-hoc Commands in Ansible**

* Working with Modules like Firewalld, URI, Get\_URL, APT/YUM, Service, User, etc.
* Writing Playbooks for Configuration Management (e.g., NFS, Apache, FTP)
* Playbook Notifications, Tags, and Handlers
* Exception Handling in Playbooks

**Roles in Ansible**

* Converting Playbooks to Roles
* Using Roles for Implementing Tomcat, Apache, etc.
* External Roles and Galaxy

**Build Automation: Maven**

* Understanding the Build Process
* Creating Maven Projects from Command Prompt
* Maven Dependencies and Plugins
* Maven Stages and Repositories
* Integrating Maven with Jenkins

**Containerization: Docker**

* Installing Docker
* Virtualization vs Containerization
* Creating and Using Containers
  + Starting Application Servers, Databases, and Operating Systems as Containers
  + Container vs VM
  + Managing Multiple Containers
  + Linking Containers
  + Docker Volumes and Networks
  + Docker Volumes for Reusability
* Getting a Shell Inside Containers (No Need for SSH)

**Container Images**

* Understanding Images and Layers
* Working with Docker Hub
* Image Tagging and Pushing to Docker Hub
* Building Images using Dockerfile and docker commit
* Extending Official Images

**Docker Compose**

* Introduction to Docker Compose and YAML Files
* Creating Multi-Container Environments with Docker Compose (Dev, QA, CI/CD)

**Docker Swarm**

* Container Orchestration with Docker Swarm
* Load Balancing, Scaling, and Handling Failover Scenarios with Swarm

**Container Orchestration: Kubernetes**

* Introduction to Kubernetes
* Kubernetes Architecture and Setup (Using Kubeadm and Play-with-K8s.com)
* Understanding Kubernetes Concepts (PODs, ReplicaSets, Deployments)
  + Replication Controllers and ReplicaSets using YAML
  + Load Balancing and High Availability
  + Scaling, Rolling Updates, and Rollbacks
  + Handling Failover Scenarios
* Kubernetes Networking Overview and Demo

**Infrastructure as Code: Vagrant**

* Introduction to Vagrant
* Installing Vagrant and Setting Up Vagrantfile
* Deploying Complete Environments with Vagrant
* Finalizing the Environment Setup

**Cloud Infrastructure: AWS**

* Environment Setup in AWS
* Cloud Deployment Scenarios in AWS
* Continuous Delivery in AWS
* Amazon Elastic Compute Cloud (EC2) and Simple Storage Service (S3)

**Linux Fundamentals**

* Linux Installation and Hard Disk Partition Details
* Linux Commands and Shell Commands
* User Administration
* Network Configuration
* Job Automation
* Disk Quota Management
* Package Management
* DNS, Process Management, and Installed Services
* Apache Web Server, MySQL Server, and MariaDB
* Log Server and Log File Management

**Infrastructure as Code: Terraform**

* Introduction to Terraform (by HashiCorp)
* Setting Up Projects with Terraform
* Terraform Configuration Files
* Authenticating with Providers (e.g., Azure)
* Terraform Commands: Init, Plan, Apply
* Managing Terraform State Files (Remote State, Output Retrieval)
* Using Terraform Modules and Registry—testing—selenium with python

**DevOps Overview**

**DevOps** is a culture and software development approach that emphasizes continuous integration, continuous delivery, and continuous deployment, along with continuous monitoring and feedback throughout the entire software development lifecycle (SDLC). This approach fosters collaboration between development (Dev) and operations (Ops) teams to automate and streamline the process of software delivery and infrastructure changes.

**Key Concepts of DevOps:**

1. **Continuous Integration (CI)**  
   Continuous Integration involves continuously developing code, combining it with the main codebase, and ensuring that the code is automatically tested. This process helps in detecting bugs early, allowing teams to fix them promptly and reduce time during the release process.
2. **Continuous Delivery (CD) - Manual Approach**  
   Continuous Delivery refers to automatically packaging code and delivering it to different environments, like Dev, QA, UAT, and Production. Although it involves automation, deployment to production is usually done manually after review.
3. **Continuous Deployment (CD) - Automated Approach**  
   Continuous Deployment takes continuous delivery a step further by automating the deployment process into live environments. Every code change passes through automated tests and is automatically deployed to production without manual intervention, ensuring rapid and safe deployments.

**DevOps Workflow Example (Jenkins Workflow):**

1. **Code Commit**  
   Developers commit code to a version control system like GitHub.
2. **Build**  
   The code is built using tools like Maven, producing artifacts such as WAR, EAR, or JAR files.
3. **Code Quality Check**  
   Code quality is assessed using tools like SonarQube, checking for critical issues, major bugs, and blockers.
4. **Release Package**  
   The build artifacts are packaged for release.
5. **Artifact Storage**  
   The release packages are uploaded to artifact repositories such as Nexus or Artifactory.
6. **Deployment**  
   The package is deployed to environments like Dev, QA, UAT, or Production using servers like Tomcat.
7. **Testing**  
   Automated or manual testing is conducted on the deployed application.

**SDLC (Software Development Life Cycle) Models:**

1. **Waterfall Model**  
   A traditional model where development and client interactions follow a linear approach, and issues are handled manually throughout the process.
2. **Agile Methodologies**  
   Agile focuses on collaboration between development, operations, and clients with continuous feedback and iteration, promoting a more flexible and efficient workflow.

**Key DevOps Tools:**

* **Version Control:** Git
* **Build Automation:** Maven
* **Application Servers:** Tomcat
* **CI/CD:** Jenkins
* **Code Quality:** SonarQube
* **Artifact Management:** Nexus
* **Configuration Management:** Ansible
* **Containerization:** Docker
* **Orchestration:** Kubernetes
* **Monitoring:** CloudWatch, New Relic, Datadog, Nagios, Grafana, Prometheus

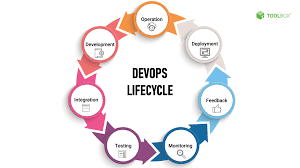
**Conclusion:**

DevOps is a transformative approach that automates the integration, delivery, and deployment of software, ensuring faster and more reliable releases. It involves a variety of tools and practices that improve collaboration between development, testing, and operations teams, ultimately enhancing efficiency and reducing time-to-market.

**What is DevOps?**

**DevOps** is a set of practices, tools, and cultural philosophies that integrate software development (Dev) and IT operations (Ops). It aims to shorten the development lifecycle, deliver high-quality software continuously, and improve collaboration between teams. DevOps focuses on automating repetitive tasks, implementing continuous integration/continuous delivery (CI/CD), and enhancing the overall efficiency of software development and deployment processes.

In essence, DevOps bridges the gap between development and operations by fostering collaboration, communication, and shared responsibility. It encourages a culture where software development, testing, and operations teams work closely together from the beginning of the development process through to production.



**DevOps Lifecycle & Key Practices**

The DevOps lifecycle includes continuous processes that support the entire software delivery pipeline. The main stages of the DevOps lifecycle are:

1. **Continuous Development**: This stage involves planning and coding. Tools like Git, SVN, and Mercurial are commonly used for version control, while agile methodologies drive planning and iteration.
2. **Continuous Integration**: In this stage, code is integrated into a shared repository multiple times a day. Automated testing is performed to catch bugs early, using tools like Jenkins, Travis CI, or CircleCI.
3. **Continuous Testing**: Automated tests are executed against the integrated codebase to ensure the software is functional and bug-free. Tools like Selenium, TestNG, and JUnit are popular for automated testing.
4. **Continuous Deployment**: This stage ensures the deployment of the application to production environments automatically. Tools like Docker, Kubernetes, Ansible, and Chef help manage deployments and configurations.
5. **Continuous Monitoring**: Monitoring and logging are essential to ensure that application performance meets expectations. Tools like Prometheus, Grafana, ELK Stack, and Splunk are used to monitor system health and alert the team to issues.
6. **Continuous Feedback**: Feedback is gathered from users and performance data to inform the development of future iterations, allowing continuous improvement.
7. **Continuous Operations**: This involves automating and optimizing routine operational tasks like infrastructure provisioning, scaling, and recovery using tools like Terraform, Ansible, or Puppet.

**Key Practices in DevOps**

* **Automation**: Automating repetitive tasks such as building, testing, deploying, and monitoring ensures consistency and speeds up the development pipeline.
* **Infrastructure as Code (IaC)**: Treating infrastructure configuration and management as code allows for automation and versioning. Tools like Terraform, Ansible, and CloudFormation are used for IaC.
* **Configuration Management**: Tools like Puppet, Chef, and Ansible automate the configuration of servers and ensure consistency across environments.
* **Continuous Integration/Continuous Delivery (CI/CD)**: CI/CD automates the integration and delivery of code to production environments, enabling frequent and reliable software releases.
* **Monitoring and Logging**: Continuous monitoring helps identify issues in real-time, enabling proactive resolution. Tools like Nagios, ELK Stack, and Prometheus help in monitoring and logging.
* **Collaboration and Communication**: DevOps promotes collaboration between development, operations, and business teams through regular communication, shared goals, and tools like Slack, Microsoft Teams, or Jira for issue tracking.

**Benefits of DevOps**

1. **Automation**: Automating the software delivery pipeline reduces manual errors, accelerates releases, and frees up teams to focus on higher-value tasks.
2. **Collaboration**: By fostering collaboration between development, operations, and other stakeholders, DevOps reduces silos and aligns teams towards common goals.
3. **Continuous Integration/Continuous Delivery (CI/CD)**: CI/CD enables frequent, reliable releases, improving the software's time-to-market and overall quality.
4. **Faster Time to Market**: The DevOps approach shortens the software development lifecycle, enabling faster and more frequent releases.
5. **Improved Reliability and Stability**: Automation and monitoring reduce downtime and ensure that software is always in a deployable state.
6. **Enhanced Security**: Automated testing and security integration (DevSecOps) ensure that security is built into every stage of the software lifecycle.
7. **Cost Efficiency**: By automating processes and improving resource utilization, DevOps reduces costs associated with manual interventions, downtimes, and inefficient processes.

**Overview of DevOps Tools & Ecosystem**

The DevOps ecosystem consists of various tools categorized by function, such as:

1. **Version Control**:
   * **Git**, **SVN**, **Mercurial**
2. **CI/CD**:
   * **Jenkins**, **Travis CI**, **CircleCI**
3. **Containerization and Orchestration**:
   * **Docker**, **Kubernetes**, **OpenShift**
4. **Configuration Management and IaC**:
   * **Ansible**, **Puppet**, **Chef**, **Terraform**
5. **Monitoring and Logging**:
   * **Nagios**, **Prometheus**, **Grafana**, **ELK Stack (Elasticsearch, Logstash, Kibana)**, **Splunk**
6. **Collaboration**:
   * **Slack**, **Microsoft Teams**, **Jira**
7. **Cloud Platforms**:
   * **AWS**, **Azure**, **Google Cloud Platform (GCP)**
8. **Security (DevSecOps)**:
   * **Aqua Security**, **Twistlock**, **Prowler**
9. **Testing**:
   * **Selenium**, **TestNG**, **JUnit**

The choice of tools often depends on the specific requirements of the organization, and tools are frequently combined to form a complete DevOps toolchain.

**DevOps Cultural Transformation & Best Practices**

DevOps is not just about tools and processes but also about fostering a culture of collaboration and continuous improvement. Key aspects of DevOps cultural transformation include:

* **Cross-functional Collaboration**: Breaking down silos between development, operations, QA, and other teams to promote a shared responsibility for delivering high-quality software.
* **Agile and Lean Principles**: Applying agile and lean methodologies to DevOps practices helps improve responsiveness and focus on customer needs.
* **Continuous Learning**: Encouraging a culture of continuous learning and improvement through regular feedback, post-mortems, and knowledge sharing.
* **Blameless Culture**: Promoting a blameless culture ensures that failures are seen as learning opportunities, not as a source of punishment.
* **Automation and Measurement**: Automating repetitive tasks and measuring key metrics such as deployment frequency, lead time, and MTTR (Mean Time to Recovery) helps drive continuous improvement.
* **Security Integration (DevSecOps)**: Embedding security into the development process from the beginning ensures that security is not an afterthought but a continuous concern.

**Conclusion**

Adopting DevOps practices leads to faster, more reliable software delivery, improved collaboration between teams, and a culture of continuous improvement. By leveraging automation, CI/CD, and a strong toolchain, organizations can achieve significant benefits in terms of speed, quality, and efficiency.