CI/CD Cheatsheet

A comprehensive cheat sheet covering essential CI/CD concepts, tools, best practices, and workflows for modern DevOps.

CI/CD Fundamentals

Core Concepts

Continuous Integration (CI): Automating the integration of code changes from multiple contributors into a single software project. Key practices include frequent code commits, automated builds, and testing.

Continuous Delivery (CD): Extending CI to automatically prepare and release code changes to production or pre-production environments. Focuses on ensuring the software can be reliably released at any time.

Continuous Deployment (CD): Further automating CD to automatically deploy code changes to production without explicit approval. Requires robust monitoring and testing to ensure stability.

Pipeline: An automated workflow that defines the stages of the CI/CD process, including build, test, and deployment. Pipelines ensure consistency and repeatability.

Benefits of CI/CD

Faster Time to Market	Rapidly deliver new features and updates to users.
Reduced Risk	Automated testing and deployment minimizes errors.
Improved Quality	Continuous testing ensures code meets quality standards.
Increased Efficiency	Automated processes free up developer time.
Enhanced Collaboration	CI/CD fosters better communication and cooperation between development and operations teams.

Key Metrics

Deployment Frequency: How often code is deployed to production.

Lead Time for Changes: Time taken from code commit to code in production.

Mean Time to Recovery (MTTR): Average time to restore service after a failure.

Change Failure Rate: Percentage of deployments causing a failure.

CI/CD Tools

Build Automation

Jenkins: An open-source automation server widely used for CI/CD. Offers extensive plugins for various tools and platforms.

Maven: A build automation tool primarily used for Java projects. Manages dependencies and provides a standard build lifecycle.

Gradle: Another build automation tool, also popular for Java, Kotlin, and Android projects. Offers flexibility and performance improvements over Maven.

Make: A build automation tool used primarily for C/C++ projects.

Version Control

Git	Distributed version control system for tracking code changes. Platforms like GitHub, GitLab, and Bitbucket provide Git repositories.
GitHub Actions	CI/CD directly integrated into GitHub repositories.
GitLab CI/CD	CI/CD directly integrated into GitLab repositories.

Configuration Management

Ansible: An automation tool for configuration management, application deployment, and task automation. Uses a simple, human-readable language (YAML).

Chef: A configuration management tool that uses Ruby-based DSL (Domain Specific Language) to define infrastructure as code.

Puppet: Another configuration management tool that uses a declarative language to define the desired state of infrastructure.

Terraform: An infrastructure as code tool for building, changing, and versioning infrastructure safely and efficiently.

CI/CD Best Practices

General Practices

Automate Everything: Automate all repetitive tasks, including build, test, and deployment.

Version Control Everything: Keep all code, configurations, and scripts in version control.

Test Early and Often: Implement comprehensive testing throughout the CI/CD pipeline.

Monitor and Alert: Implement robust monitoring to detect and respond to issues quickly.

Small, Incremental Changes: Break down large changes into smaller, manageable increments.

Infrastructure as Code (IaC): Manage and provision infrastructure through code.

Testing Strategies

Unit Tests	Test individual components or functions in isolation.
Integration Tests	Test the interaction between different components or services.
End-to-End Tests	Test the entire application workflow from start to finish.
Performance Tests	Measure the performance and scalability of the application.
Security Tests	Identify and mitigate security vulnerabilities.

Pipeline Stages

Source: Code repository (e.g., Git)

Build: Compile code, package dependencies, and create artifacts.

Test: Run automated tests (unit, integration, end-to-end).

Package: Create deployable packages (e.g., Docker images).

Deploy: Deploy packages to staging or production environments.

Monitor: Track application performance and health.

CI/CD in the Cloud

Cloud Platforms

AWS: AWS offers a wide range of services for CI/CD, including CodePipeline, CodeBuild, CodeDeploy, and ECS/EKS for container orchestration.
Azure: Azure provides Azure DevOps for CI/CD, including Azure Pipelines, Azure Boards, and Azure Repos. It also offers AKS for container orchestration.
GCP: GCP offers Cloud Build for CI/CD, along with Google Kubernetes Engine (GKE) for container orchestration.

Containerization

Docker	A platform for building, shipping, and running applications in containers. Containers provide a consistent and isolated environment for applications.
Kubernetes	An open-source container orchestration platform for automating deployment, scaling, and management of containerized applications.
Serverless	Cloud functions like AWS Lambda, Azure Functions, and Google Cloud Functions enable event-driven CI/CD workflows.

Cloud-Native CI/CD

Leverage Managed Services: Utilize cloud provider's managed CI/CD services (e.g., AWS CodePipeline, Azure Pipelines, GCP Cloud Build) for simplified setup and maintenance.
Embrace Infrastructure as Code: Define and manage cloud infrastructure using tools like Terraform or CloudFormation for automated provisioning and configuration.
Automate Security Scans: Integrate security scanning tools into the CI/CD pipeline to identify vulnerabilities early in the development process.
Implement Blue/Green Deployments: Use blue/green deployments or canary releases to minimize downtime and risk during deployments.