CI CD - Cheat Sheet

1. Was ist CI/CD?

- **Continuous Integration (CI):** Der Prozess, bei dem Entwickler ihre Codeänderungen regelmäßig (mehrmals täglich) in ein gemeinsames Repository integrieren. CI stellt sicher, dass der Code durch automatisierte Tests und Builds validiert wird, um Integrationsprobleme frühzeitig zu erkennen.
- **Continuous Delivery (CD):** Der Prozess, bei dem der Code nach der erfolgreichen Integration und den Tests automatisch in eine Produktionsumgebung überführt wird. Dies beinhaltet auch das Continuous Deployment, bei dem der Code ohne manuelle Eingriffe in die Produktion geht.

2. Wesentliche CI/CD Konzepte

- **Automatisierte Builds:** Automatisierte Prozesse, um den Quellcode zu kompilieren, zu testen und in eine lauffähige Software zu verwandeln.
- **Automatisierte Tests:** Unit-Tests, Integrationstests und UI-Tests, die nach jedem Build automatisch ausgeführt werden, um die Codequalität sicherzustellen.
- **Deployment:** Die Bereitstellung der Software in verschiedenen Umgebungen (z. B. Staging, Produktion) mit möglichst wenig manuellen Eingriffen.

3. Build-Pipelines

Eine **Build-Pipeline** ist eine Serie von Automatisierungsschritten, die den Code von der Integration bis zur Bereitstellung in die Produktion durchläuft. Diese Schritte umfassen üblicherweise:

1. **Code Commit:** Entwickler checken Code in ein Versionskontrollsystem ein (z. B. Git).
2. **Build:** Der Code wird in eine ausführbare Software umgewandelt.
3. **Test:** Automatisierte Tests werden durchgeführt, um Fehler zu erkennen.
4. **Deploy:** Der Code wird in eine Staging- oder Produktionsumgebung überführt.
5. **Monitor:** Überwachung der Anwendung nach der Bereitstellung, um Fehler in der Produktion zu erkennen.

4. Tools für CI/CD

- **Jenkins:**
  - **Beschreibung:** Jenkins ist ein weit verbreitetes Open-Source-Tool zur Automatisierung von CI/CD-Pipelines. Es unterstützt viele Plugins für verschiedene Programmiersprachen und Build-Tools.

    - **Funktionen:** Erstellen und Ausführen von Pipelines, Integration mit Versionskontrollsystemen und Benachrichtigungen.
    - **Vorteile:** Flexibilität, umfangreiche Plugin-Unterstützung, breite Community.
- **GitHub Actions:**
    - **Beschreibung:** GitHub Actions ermöglicht die Automatisierung von Workflows direkt in GitHub-Repositories. Es wird häufig für CI/CD verwendet und ist in GitHub integriert.
    - **Funktionen:** Erstellen von Workflows, Testen und Bereitstellen von Anwendungen, automatische Releases.
    - **Vorteile:** Keine separate Infrastruktur nötig, nahtlose GitHub-Integration.
- **Bamboo:**
    - **Beschreibung:** Bamboo von Atlassian ist ein kommerzielles CI/CD-Tool, das gut mit anderen Atlassian-Produkten wie Jira und Bitbucket integriert ist.
    - **Funktionen:** Erstellen von Build-Plänen, paralleles Testen, kontinuierliche Bereitstellung.
    - **Vorteile:** Starke Integration mit Jira, erweiterte Berichterstattung.
- **Maven:**
    - **Beschreibung:** Maven ist ein Build-Management-Tool für Java-Projekte. Es vereinfacht den Build-Prozess, indem es die Abhängigkeitsverwaltung und das Projektmanagement übernimmt.
    - **Funktionen:** Erstellen von Java-Projekten, Abhängigkeitsmanagement, Ausführung von Tests.
    - **Vorteile:** Standardisiert den Build-Prozess, große Community, einfache Integration in CI-Tools.
- **Docker:**
    - **Beschreibung:** Docker ist eine Plattform zur Erstellung, Bereitstellung und Ausführung von Anwendungen in Containern. Diese Container beinhalten alles, was für die Ausführung einer Anwendung notwendig ist, wodurch Konsistenz über verschiedene Umgebungen hinweg gewährleistet wird.
    - **Funktionen:** Containerisierung von Anwendungen, Portabilität zwischen Entwicklungs- und Produktionsumgebungen.
    - **Vorteile:** Isolation, Skalierbarkeit, schnellere Bereitstellung.
- **Helm:**
    - **Beschreibung:** Helm ist ein Paketmanager für Kubernetes, der das Management von Kubernetes-Anwendungen vereinfacht. Es hilft, komplexe Kubernetes-Deployments zu organisieren und zu versionieren.
    - **Funktionen:** Erstellen und Verwalten von Kubernetes-Deployments als Helm-Charts.
    - **Vorteile:** Wiederverwendbarkeit von Kubernetes-Konfigurationen, einfache Updates und Rollbacks.

5. Beispiel für eine typische CI/CD-Pipeline

1. **Code Commit (GitHub oder Bitbucket):** Entwickler pushen ihren Code.
2. **Jenkins/GitHub Actions:** Die CI/CD-Plattform startet den Build-Prozess automatisch.
3. **Build (Maven + Docker):** Der Code wird gebaut, und ein Docker-Image wird erzeugt.
4. **Tests (JUnit, Selenium):** Unit-Tests und Integrationstests werden ausgeführt.
5. **Deployment (Docker + Helm):** Das Docker-Image wird in einem Kubernetes-Cluster bereitgestellt.
6. **Monitoring:** Tools wie Prometheus oder Grafana überwachen die Produktion.

6. Best Practices für CI/CD

- **Automatisierung:** Minimieren Sie manuelle Schritte, um Fehler zu reduzieren und die Konsistenz zu gewährleisten.
- **Kontinuierliche Verbesserung:** Analysieren und optimieren Sie regelmäßig die Pipeline.
- **Fehler schnell erkennen:** Verwenden Sie Monitoring und Benachrichtigungen, um Probleme frühzeitig zu identifizieren.
- **Sicherheit:** Integrieren Sie Sicherheitsprüfungen und Scans in die Pipeline.