DevOps

- DevOps is a software development methodology that emphasizes collaboration, communication, and automation between software development (Dev) and IT operations (Ops) teams.

- Shorten the systems development life cycle and provide frequent delivery of high-quality software and services.

- DevOps is more than just a set of tools; it's a cultural shift that promotes shared responsibility and continuous feedback loops.

CI/CD

- In software engineering, CI/CD or CICD is the combined practices of continuous integration (CI) and continuous delivery (CD) or, less often, continuous deployment.

- CI (Continuous Integration)
  - Frequent merging of several small changes into a main branch.
  - Developers frequently merge code changes into a shared repository.
  - Automated builds and tests run on every commit.

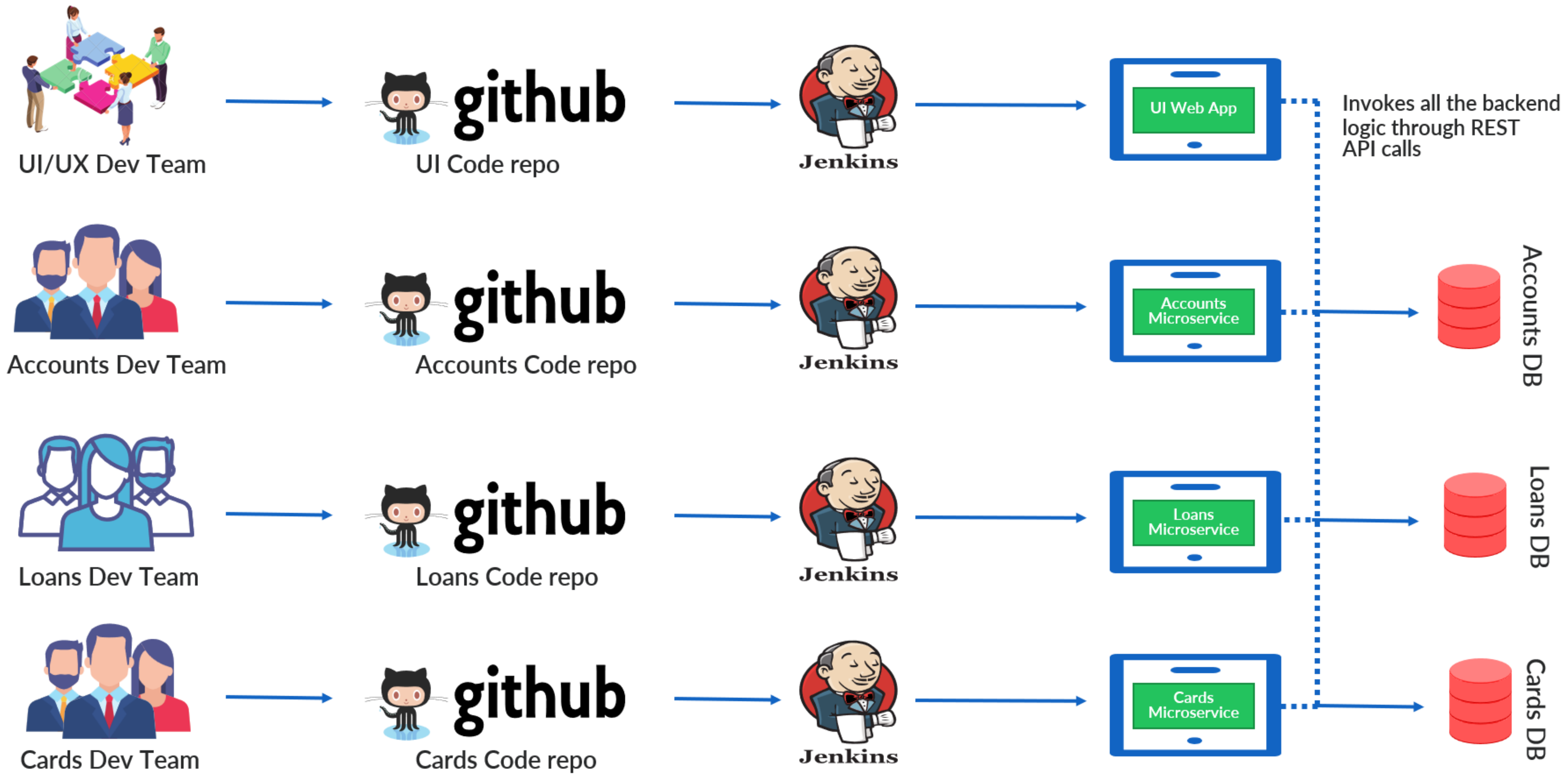
- CD (Continuous Delivery/Deployment)
  - Producing software in short cycles with high speed and frequency so that reliable software can be released at any time.
  - Ensures code is always deployable (manual trigger).

Motivation

- CI/CD bridges the gaps between development and operation activities and teams by enforcing automation in building, testing and deployment of applications.
- CI/CD services compile the incremental code changes made by developers, then link and package them into software deliverables.
- The aim is to increase early defect discovery, increase productivity, and provide faster release cycles.

Modern-day DevOps practices

- Continuous Development
- Continuous Testing
- Continuous Integration
- Continuous Deployment
- Continuous Monitoring


UI/UX Dev Team
github
UI Code repo
Jenkins
UI Web App
Invokes all the backend logic through REST API calls
Accounts Dev Team
github
Accounts Code repo
Jenkins
Accounts Microservice
Accounts DB
Loans Dev Team
github
Loans Code repo
Jenkins
Loans Microservice
Loans DB
Cards Dev Team
github
Cards Code repo
Jenkins
Cards Microservice
Cards DB

Why CI/CD Matters?

- Faster Time-to-Market (Rapid releases)
- Improved Code Quality (Automated testing catches bugs early)
- Reduced Manual Errors (Less human intervention)
- Better Collaboration (Dev & Ops work together)
- Increased Deployment Frequency (From months to minutes)

CI vs. CD vs. CD

Aspect	Continuous Integration (CI)	Continuous Delivery (CD)	Continuous Deployment (CD)
Definition	Frequent code merges & automated testing	Ensures code is always deployable	Automatically deploys to production
Deployment	No automatic deployment	Manual deployment trigger	Fully automated deployment
Goal	Detect bugs early	Ensure deployable code	Zero-touch production releases

CI/CD Pipeline Stages

- Code Commit (Git, SVN, Mercurial)
- Build (Compile code, resolve dependencies – Maven, Gradle, npm)
- Test (Unit tests, integration tests – JUnit, Selenium)
- Deploy to Staging (Test environment – Docker, Kubernetes)
- Post-Deployment Testing (Smoke tests, security scans)
- Release to Production (Automated or manual approval)

Popular CI/CD Tools

Category	Tools
CI Servers	Jenkins, GitHub Actions, GitLab CI, CircleCI
Build Tools	Maven, Gradle, npm, Make
Testing Tools	JUnit, Selenium, Cypress
Deployment	Docker, Kubernetes, Ansible, Terraform
Monitoring	Prometheus, Grafana, ELK Stack

Best Practices

- Automate Everything (Build, Test, Deploy)
- Use Version Control (Git with branching strategies like GitFlow)
- Run Tests in Parallel (Speed up feedback loop)
- Monitor Pipelines (Track failures & performance)
- Security Scanning (SAST, DAST in pipeline)
- Infrastructure as Code (IaC) (Terraform, Ansible)

Stages

- Code Commit (Version Control)
- Build Stage
- Test Stage (Automated Testing)
- Integration Tests
- Dockerize (Containerization – Optional)
- Deploy to Staging
- Approval Gate (Manual)
- Deploy to Production
- Post-Deployment Checks

Sample Pipeline – Git Action

```
name: Spring Boot CI/CD
on: [push]
jobs:
 build-and-test:
  runs-on: ubuntu-latest
  steps:
   - uses: actions/checkout@v4
   - name: Set up JDK 17
    uses: actions/setup-java@v3
    with:
     java-version: '17'
   - name: Build with Maven
    run: mvn clean install
   - name: Run Tests
    run: mvn test

 deploy-to-staging:
  needs: build-and-test
  runs-on: ubuntu-latest
  steps:
   - name: Deploy to Kubernetes
    run: kubectl apply -f k8s-deployment.yaml
```