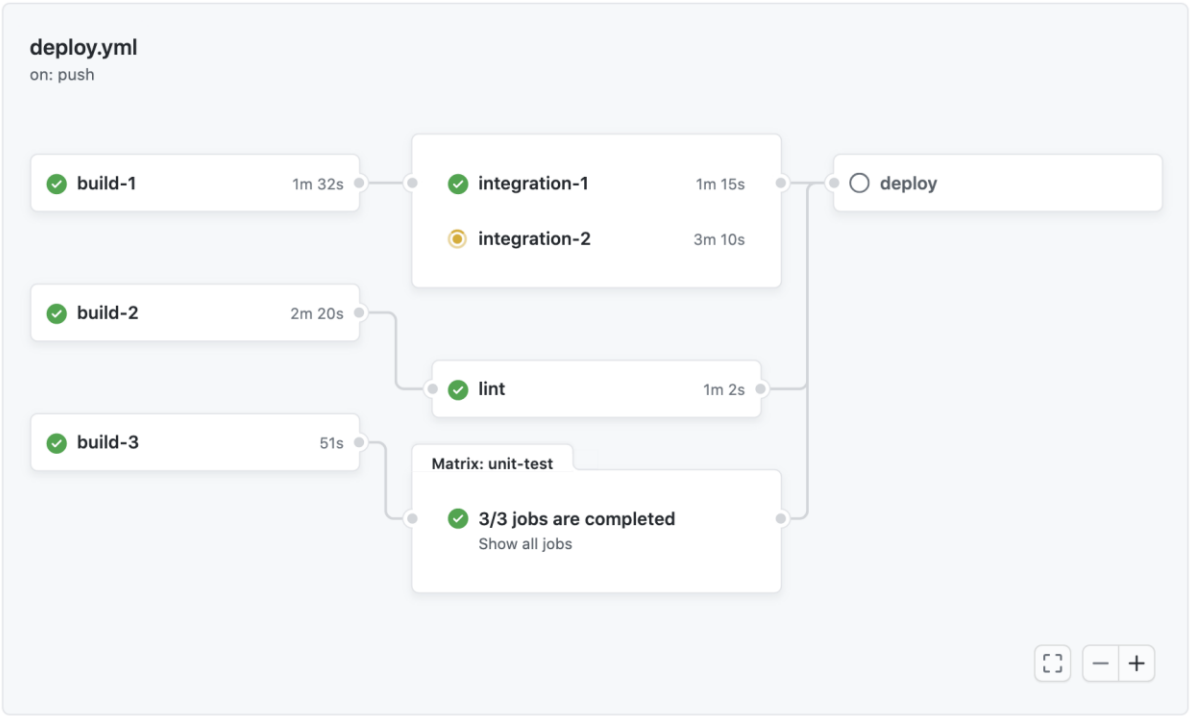
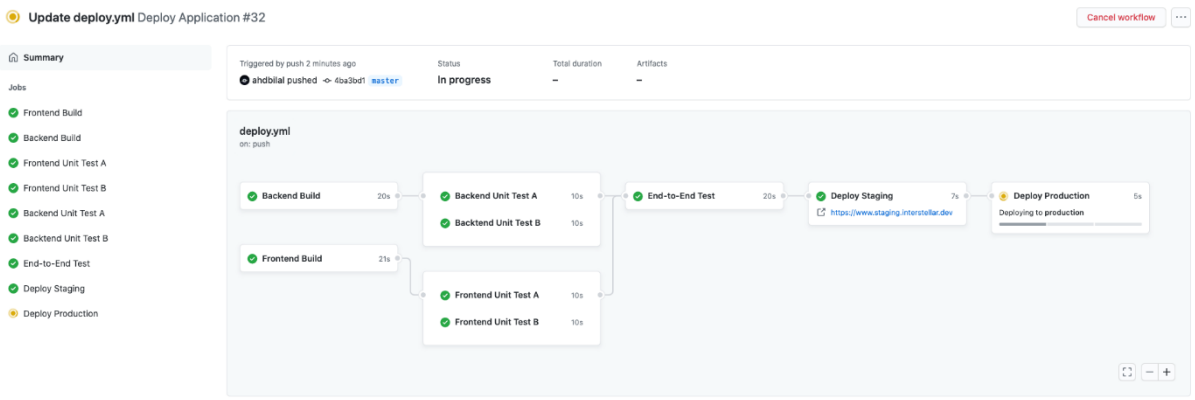
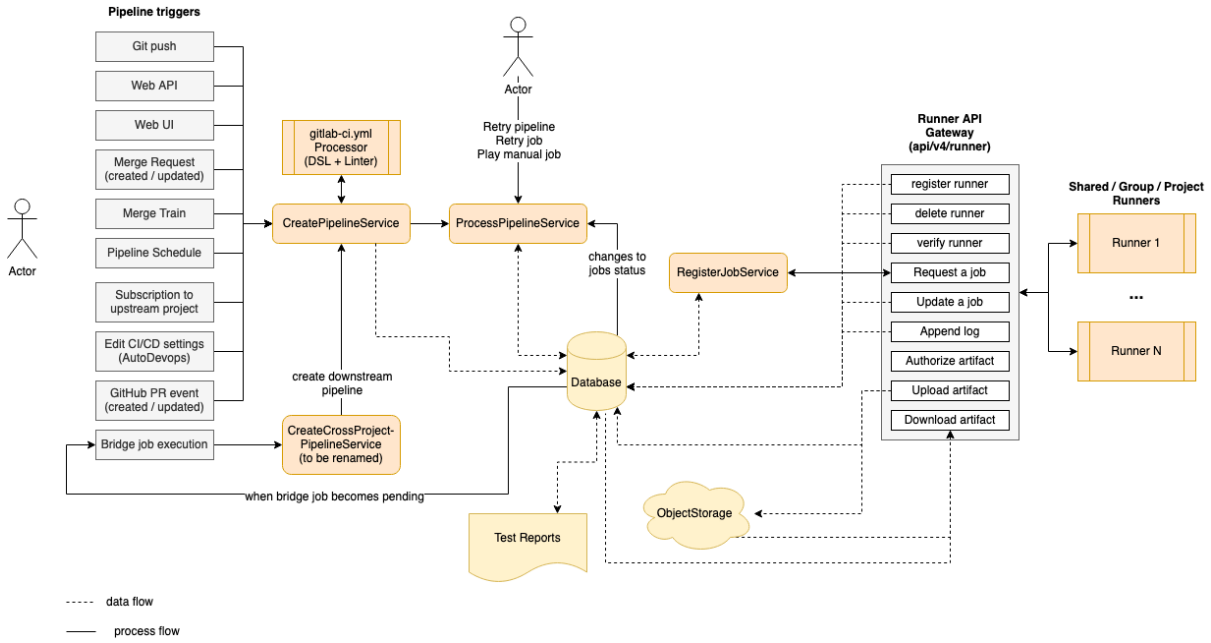
Module 5: CI/CD Automation in the Cloud — Detailed Notes

1. GitHub Actions, GitLab CI, Azure DevOps Pipelines







1.1 GitHub Actions

GitHub Actions enables **workflow automation** using YAML files stored inside .github/workflows/.

Key Features:

- Event-driven (push, PR, schedule, tag)
- Built-in marketplace actions
- Supports container images and VMs
- Matrix builds for parallel testing

Workflow Example:

```
name: CI Build

on:

  push:

    branches: ["main"]

jobs:

  build:

    runs-on: ubuntu-latest

    steps:

      - uses: actions/checkout@v3

      - name: Build app
```

run: echo "Building application"

1.2 GitLab CI/CD

GitLab uses a .gitlab-ci.yml file at repo root.

Key Features:

- Auto DevOps
- Container registry integration
- Runner agents for execution
- Supports Docker, Kubernetes, VMs

Pipeline Example:

```
stages:
  - build
  - test
  - deploy

build:
  stage: build
  script:
    - docker build -t myapp .

test:
  stage: test
  script:
    - pytest

deploy:
  stage: deploy
  script:
    - kubectl apply -f k8s/
```

1.3 Azure DevOps Pipelines

Azure Pipelines support:

- YAML pipelines
- Classic GUI pipelines
- Multi-stage deployments

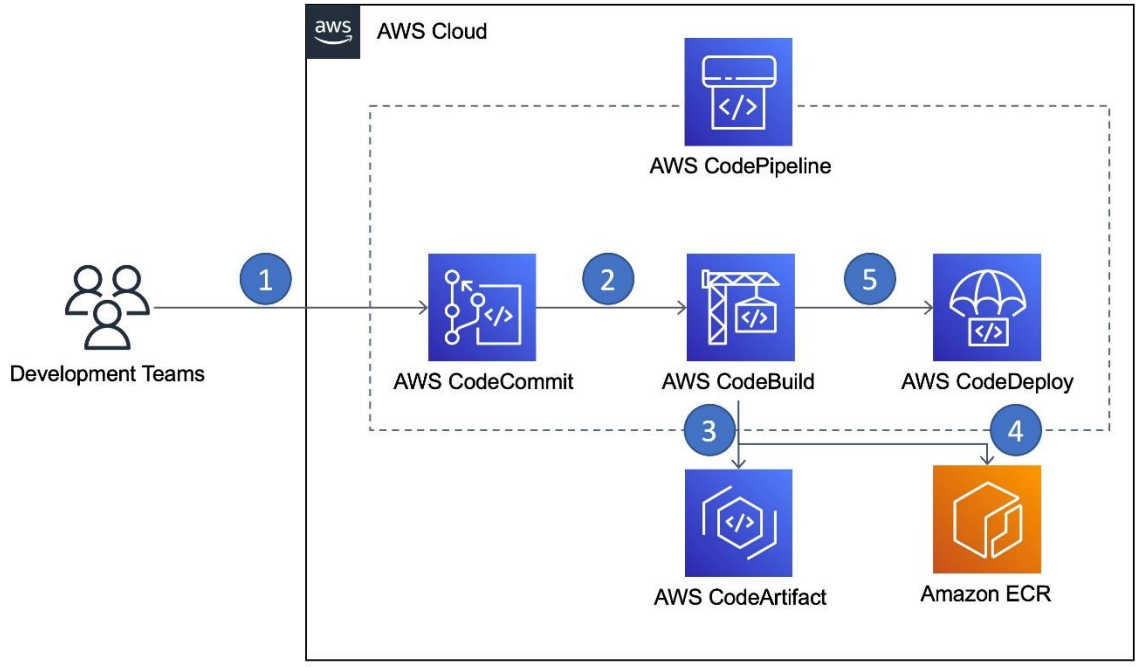
Pipeline Example:

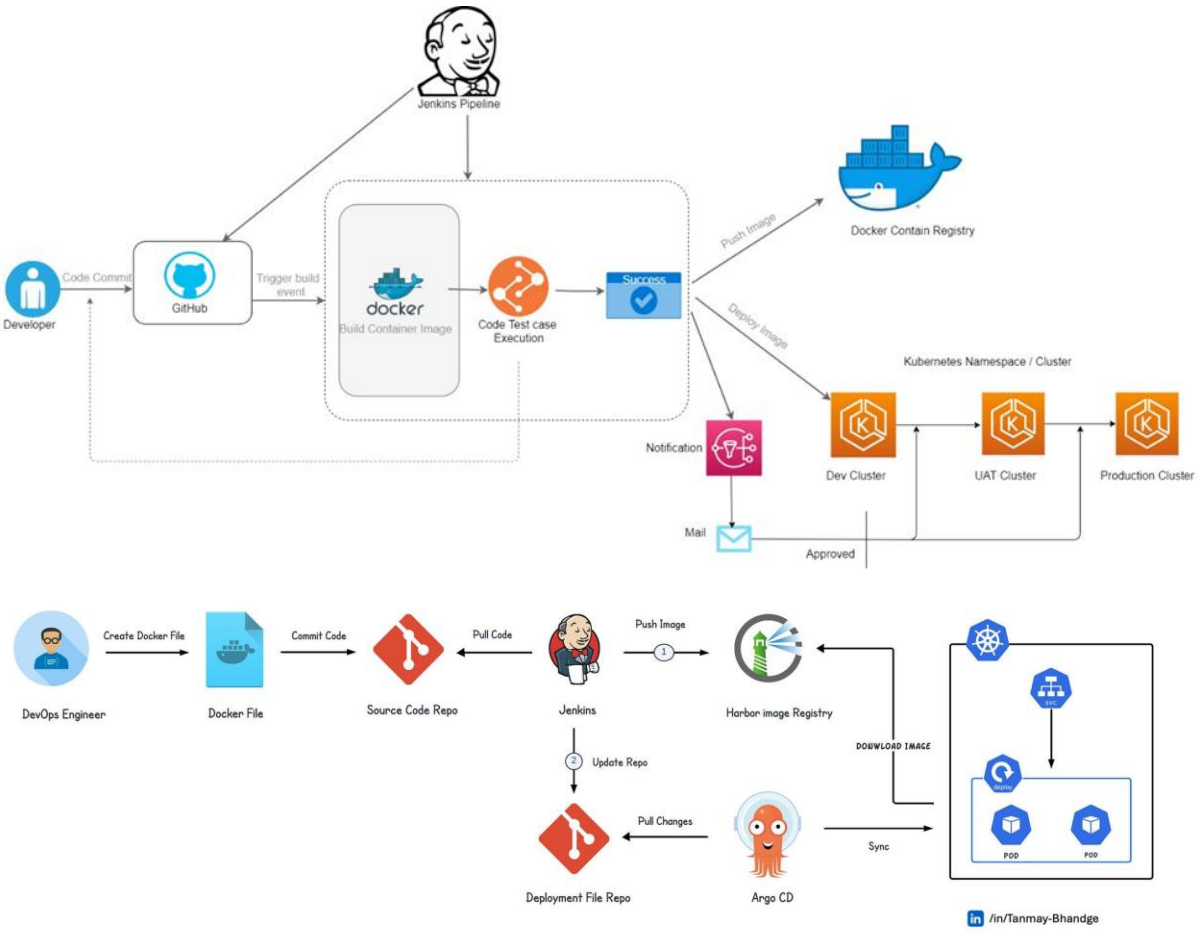
```
trigger:
 - main
pool:
 vmImage: ubuntu-latest


steps:
 - checkout: self
 - task: Docker@2
   inputs:
     command: build
     Dockerfile: Dockerfile
```

---

2. Build Pipelines for Containerized Workloads





2.1 Common Steps in Container Build Pipelines

1. Clone repository
2. Install or download tools
3. Build Docker image
4. Tag and version the image
5. Run unit tests
6. Push image to registry
7. Deploy to Kubernetes (optional)

2.2 Example: Container Build (Github Actions)

```
jobs:
  build-docker:
    runs-on: ubuntu-latest
    steps:
```

```
    - uses: actions/checkout@v3

    - name: Build Docker Image
      run: docker build -t myapp:${{ github.sha }} .

    - name: Login to DockerHub
      run: echo "${{ secrets.DOCKER_PASSWORD }}" | docker login -u ${{ secrets.DOCKER_USERNAME }} --password-stdin

    - name: Push Image
      run: docker push myapp:${{ github.sha }}
```

2.3 Example: Build + Deploy to Kubernetes (GitLab)

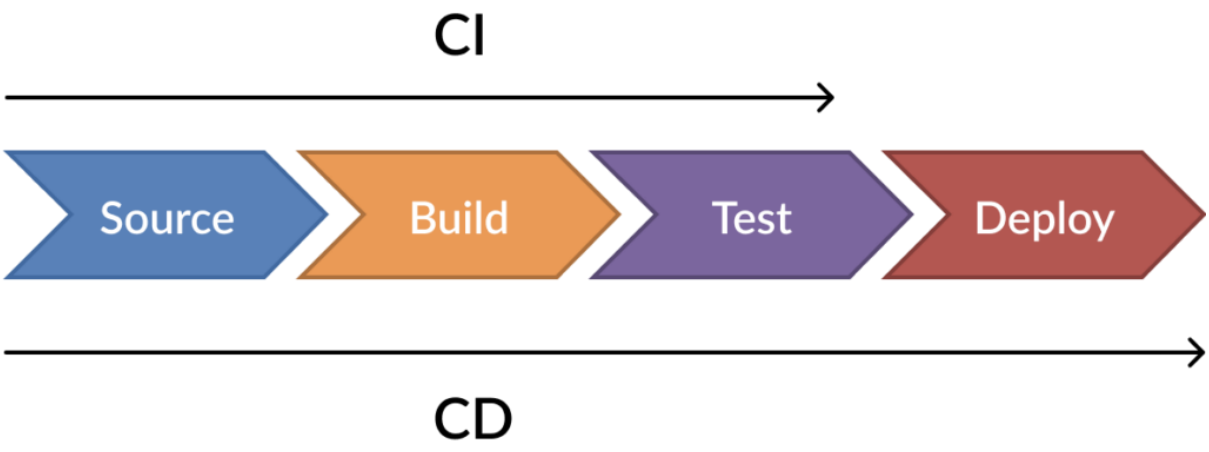
```
deploy:
  stage: deploy
  script:
    - kubectl config set-cluster mycluster ...
    - kubectl apply -f k8s/
  environment:
    name: prod
```

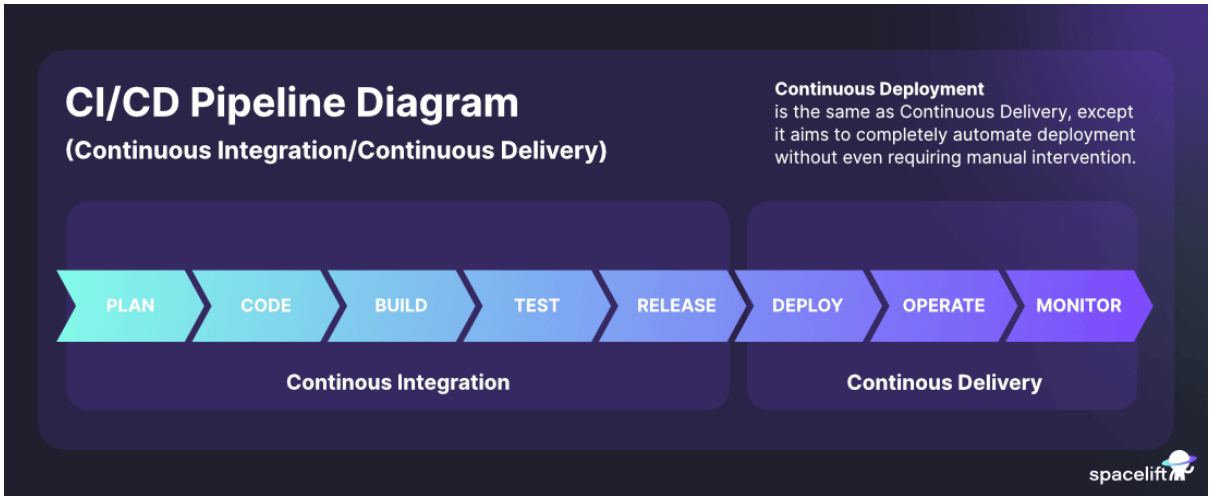
2.4 Azure DevOps Full Build Example

```
steps:
  - task: Docker@2
    inputs:
      command: buildAndPush
      containerRegistry: dockerConnection
      repository: myapp
      Dockerfile: Dockerfile
```

---

3. CI/CD Best Practices in Multi-Stage Delivery





3.1 Multi-Stage Pipeline Structure

Typical stages:

1. **Source** (commit/push)
2. **Build** (Docker build)
3. **Test** (unit + integration)
4. **Security Scan** (Trivy, Snyk, SonarQube)
5. **Artifact Storage** (registry)
6. **Staging Deployment**
7. **Manual/Automated Approval**
8. **Production Deployment**
9. **Monitoring Feedback Loop**

3.2 Best Practices

Best Practice	Benefit
Use multi-stage Docker builds	Smaller images, faster pipelines
Shift-left testing	Early detection of issues
Use versioned images (SHA/tag)	Repeatable builds
Use environments like dev/qa/prod	Better control
Use Infrastructure as Code in pipelines	Automated environment creation
Use deployment gates	Controlled production rollout
GitOps model for deployments	Full traceability

3.3 Artifact Versioning Strategy

Tag images like:

- myapp:v1.2.4
- myapp:2024.03.25
- myapp:main-${GITHUB_SHA}

Avoid using:
latest

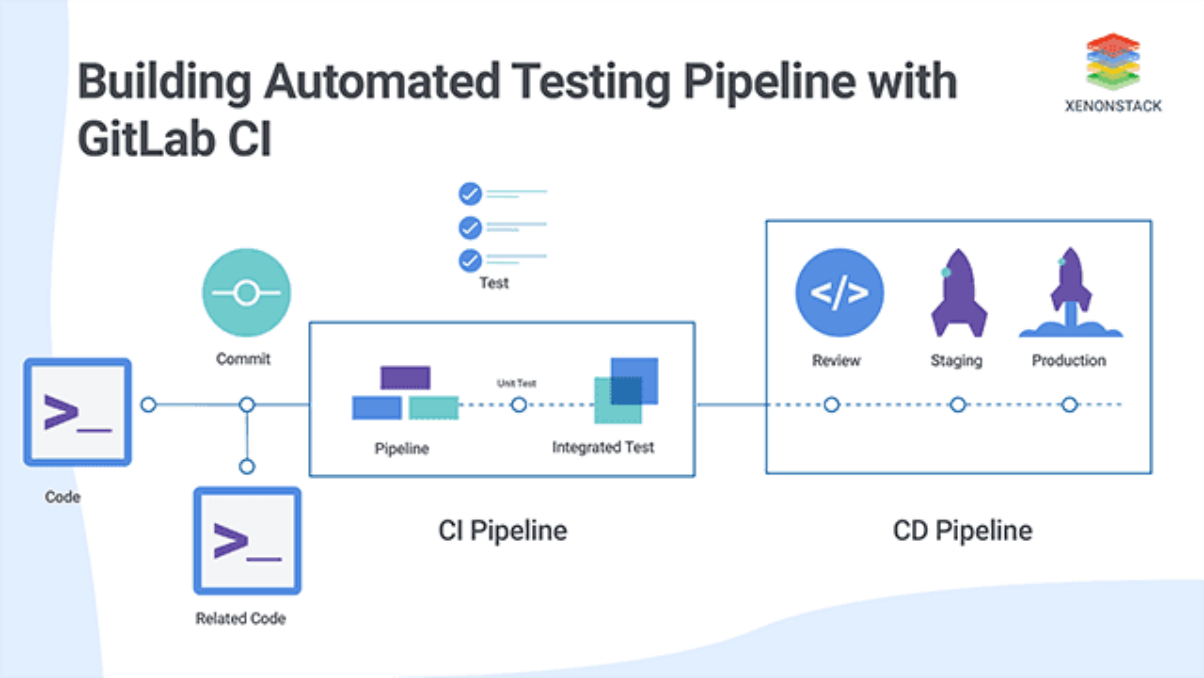
3.4 Canary & Blue-Green Deployment Support

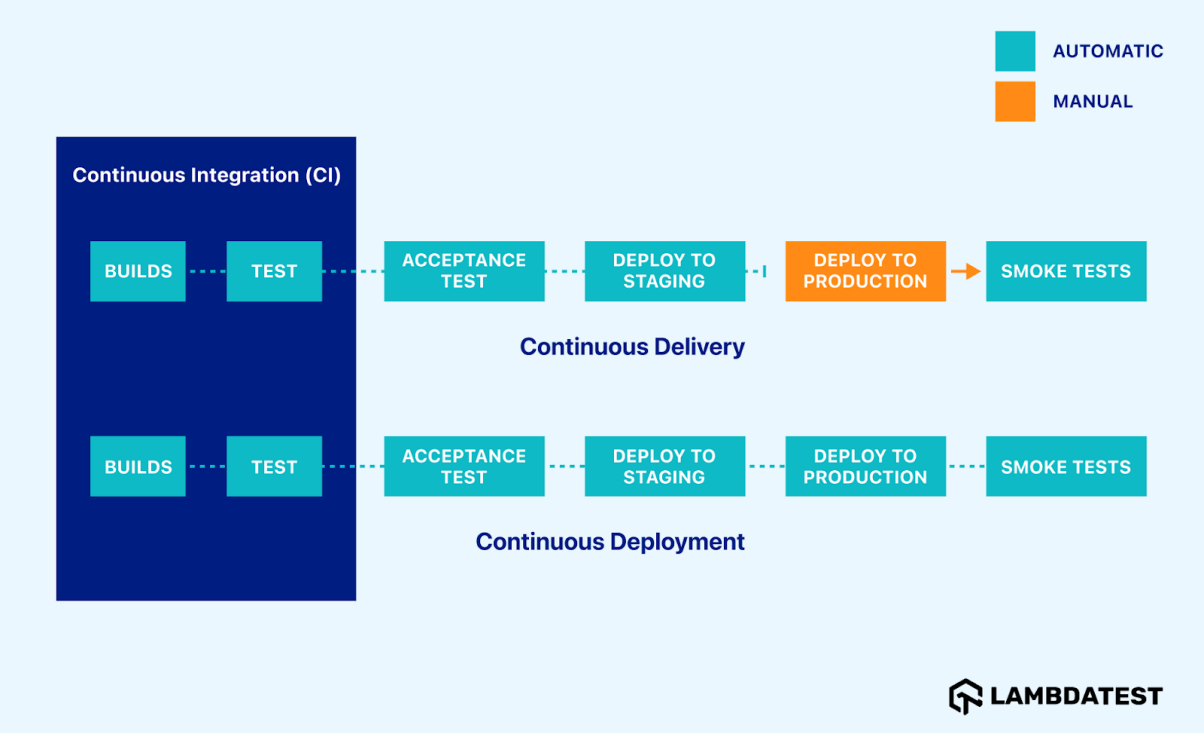
Modern pipelines support:

- Canary release (1% → 5% → 100%)
- Blue-green deployments
- Rolling updates

---

4. Automated Testing & Linting in Pipelines





4.1 Types of Automated Tests

Test Type	Purpose
Unit Tests	Validate individual functions
Integration Tests	Validate modules/components
API/Contract Tests	Validate external services
E2E Tests	Validate full flow
Load/Performance Tests	Validate scaling
Security Tests	Vulnerability scanning
Linting	Coding standard enforcement

4.2 Example: Automated Testing (GitHub Actions)

```
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Install Packages
        run: npm install
      - name: Run Tests
        run: npm test
```

4.3 Linting Example

Code Lint (JavaScript)

```
- name: Run ESLint
  run: npm run lint
```

Dockerfile Lint

```
- name: Lint Dockerfiles
  uses: hadolint/hadolint-action@v3.1.0
```

YAML Lint

```
- name: Lint YAML
```

run: yamllint .

4.4 Security & Quality Gates

Common tools:

- SonarQube (code quality)
- Trivy (vulnerability scanning)
- Snyk (dependencies)
- Checkov (IaC scanning)
- OWASP ZAP (DAST)
- Semgrep (SAST)

4.5 Test Coverage Reporting

GitHub Actions Example:

- name: Upload Coverage

uses: codecov/codecov-action@v3

Module 5 Summary

Topic	**Summary**
CI/CD Platforms	GitHub Actions, GitLab CI, Azure DevOps
Container Build Pipelines	Docker build → test → push → deploy
Multi-Stage Deliveries	Build, test, scan, stage, prod, monitor
Automated Testing/Linting	Unit tests, security scans, code quality