DevSecOps

ALI AHMED THAWERANI

DevSecOps Building Blocks

LECTURE 2

The First way

Selecting Which Value Stream to Start With

- Identify the optimal value stream to begin DevOps transformation efforts.
  - Focus on manageable, high-impact areas.
  - Minimize risks and maximize success.
- Importance of Value Stream Selection
  - Dictates transformation complexity.
  - Influences team organization and involvement.
  - Determines impact on overall goals.
- Key Factors for Value Stream Selection
  - Strategic alignment with organizational goals.
  - Feasibility and potential impact.
  - Level of resistance from involved teams.

Selecting Which Value Stream to Start With

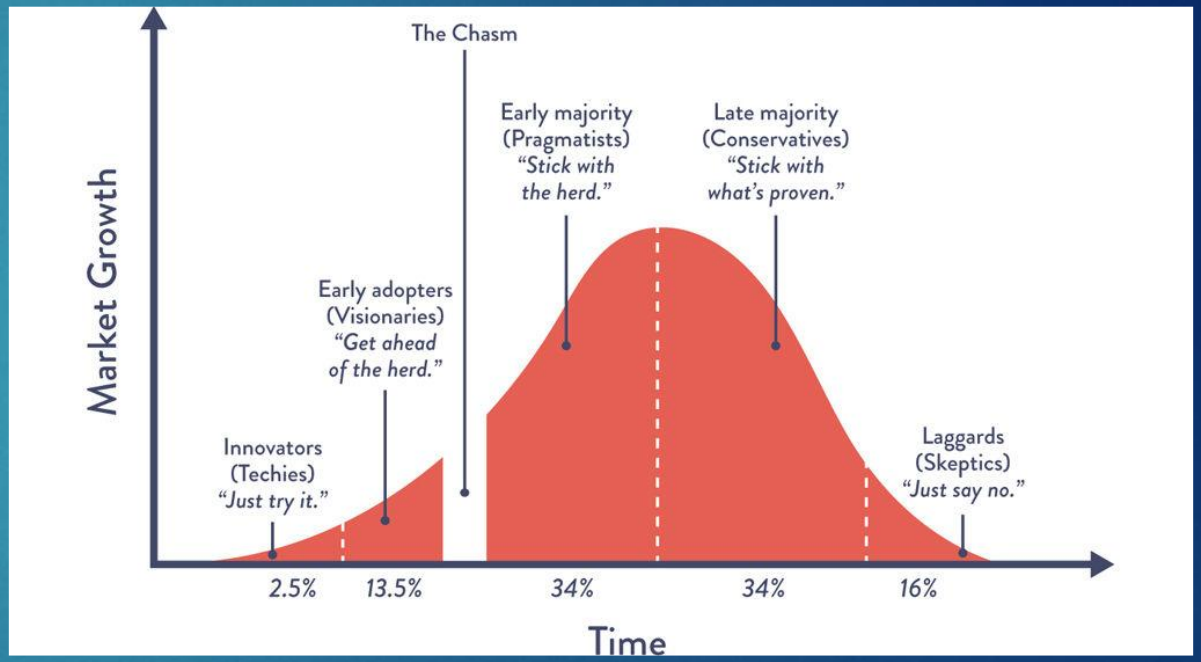
- Greenfield vs. Brownfield Projects
  - Greenfield: New software projects with minimal constraints.
  - Brownfield: Existing projects with legacy challenges.
- Systems of Engagement vs. Systems of Record
  - Systems of Engagement: Customer-facing, fast-paced changes.
  - Systems of Record: Backend systems, slower changes.
  - Key Insight: High-performing organizations excel in both areas.

Evaluating Risk and Reward

- Framework: Assess risk vs. reward for transformation efforts.
  - Business impact.
  - Technical challenges.
  - Team readiness.
- The Role of Leadership Support
  - Secure executive buy-in.
  - Align transformation goals with organizational strategy.
  - Empower teams to experiment and innovate.

Selecting Which Value Stream to Start With

- Starting with Sympathetic Teams
  - Early adopters drive momentum.
  - Builds credibility with successful outcomes.
  - Example: Innovators and early adopters in the technology adoption lifecycle.
- Importance of Early Wins
  - Focus on small, impactful projects.
  - Broadcast successes organization-wide.



Creating a Dedicated Transformation Team

- A team focused solely on transformation initiatives.
  - Allocate full-time members.
  - Provide resources and autonomy.
  - Protect from daily operational pressures.
- Incremental vs. Big-Bang Approach
  - Incremental: Focused, low-risk steps.
  - Big-Bang: Organization-wide change.
- Recommendation: Start small, scale gradually.

Managing Resistance

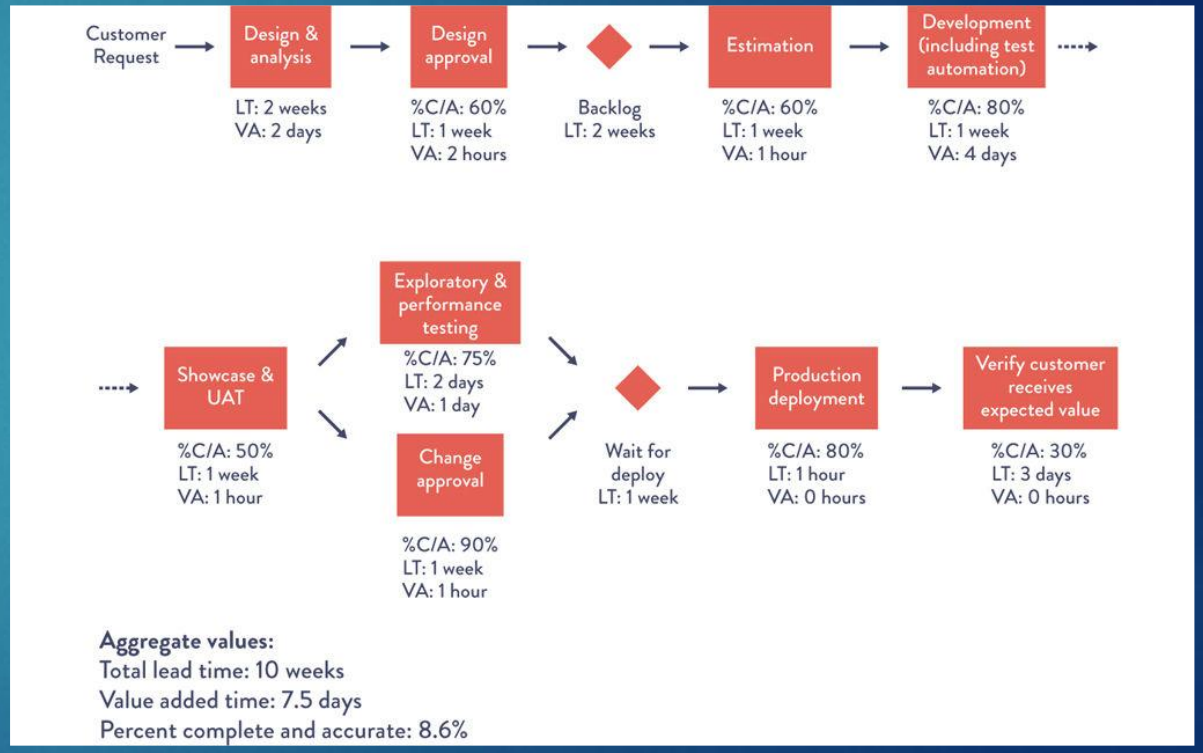
- Resistance from conservative groups.
- Fear of disruption.
  - Engage key influencers.
  - Provide training and support.
- Balancing Speed and Stability
  - Implement automated testing.
  - Enforce standardized deployment practices.
- Metrics for Measuring Success
  - Deployment frequency.
  - Lead time for changes.
  - Incident resolution time.

Understanding the Work in Our Value Stream, Making It Visible

- Learn to map, visualize, and analyze the work in value streams to identify inefficiencies.
  - Increase visibility of work across teams.
  - Identify and eliminate bottlenecks.
  - Enhance collaboration and efficiency.
- What is a Value Stream?
  - The series of steps required to deliver a product or service to a customer.
    - Development activities.
    - Operational workflows.
    - Customer delivery steps.

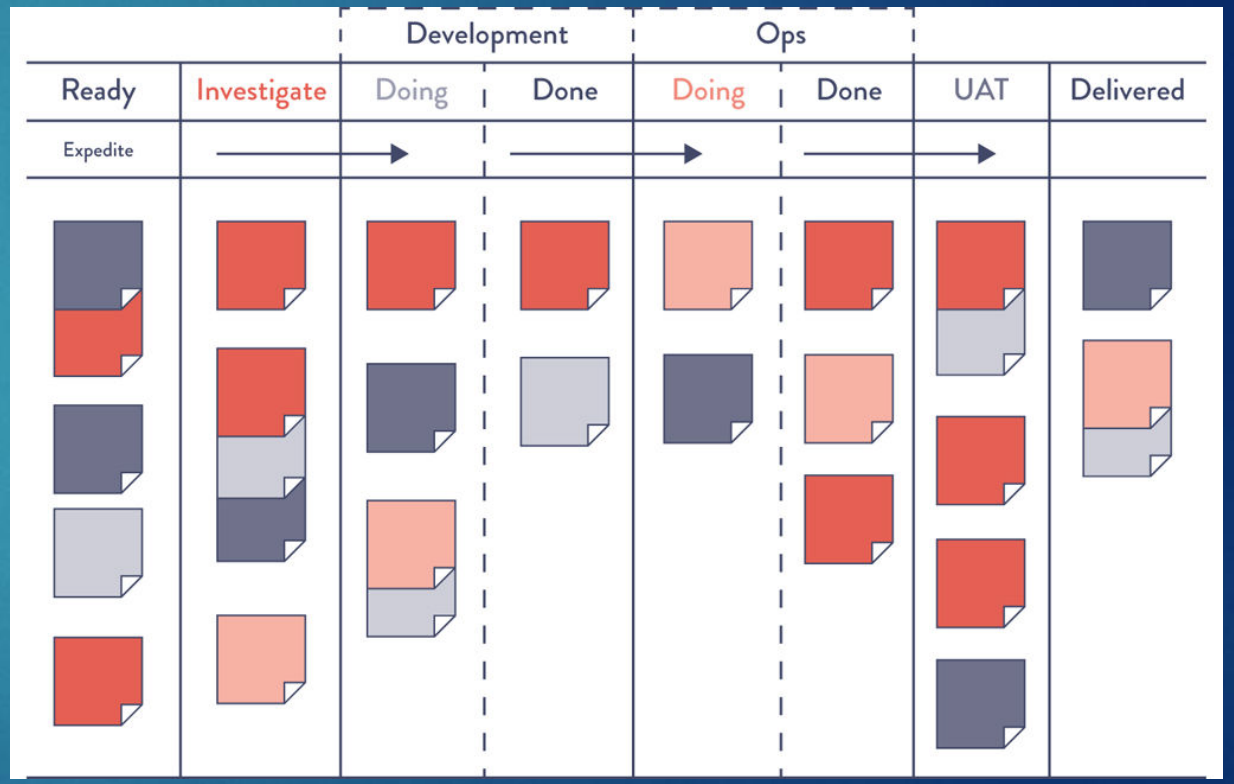
Making Work Visible

- Importance of Making Work Visible
  - Identifies delays and inefficiencies.
  - Enhances communication between teams.
  - Enables data-driven decision-making.
- Mapping the Value Stream
  - Identify the product or service.
  - Map all activities from concept to delivery.
  - Capture cycle times and delays.



Making Work Visible

- Identifying Bottlenecks
- A stage in the value stream where work piles up, slowing overall progress.
  - High work-in-progress (WIP).
  - Long wait times.
- Visualizing Work
  - Kanban boards for task tracking.
  - Workflow management software (e.g., Trello, Jira).
  - Gantt charts for project timelines.



Making Work Visible

- Work In Progress (WIP) Limits
- Restricting the number of tasks in progress to enhance focus and efficiency.
  - Reduces context switching.
  - Improves team productivity.
- Types of Waste in Value Streams
  - Overproduction.
  - Waiting.
  - Rework and defects.
  - Motion (unnecessary steps).

Making Work Visible

- Managing WIP with Kanban
  - Set WIP limits for each column.
  - Example: Limit of three cards for testing.
- Benefits of Limiting WIP
  - Makes it easier to see and address problems.
  - Encourages finishing work before starting new tasks.
- Reducing Batch Sizes
  - Perform work in small batch sizes.
  - Strive for single-piece flow to reduce lead times and increase quality.

Making Work Visible

- Large vs. Small Batch Sizes
  - Large batch sizes result in high WIP and variability.
  - Small batch sizes lead to smoother flow and better quality.
- Example: Newsletter Mailing Simulation
  - Large batch strategy: Sequentially perform each operation on all items.
  - Small batch strategy: Complete all steps for one item before starting the next.

Making Work Visible

- Reducing Batch Sizes
  - Small batch sizes result in less WIP, faster lead times, faster error detection, and less rework.
- Large Batch vs. Small Batch
  - Large batch: High WIP, long lead times, poor quality.
  - Small batch: Faster completion, quicker error detection, less rework.
- Continuous Deployment
  - Equivalent to single-piece flow in technology.
  - Each change committed to version control is integrated, tested, and deployed into production.

Making Work Visible

- Reducing the Number of Handoffs
  - Multiple operations required to move code from version control to production.
  - Each handoff requires communication and coordination, leading to delays.
- Mitigating Handoff Problems
  - Automate significant portions of the work.
  - Build platforms and reorganize teams for self-service builds, testing, and deployments.
- Identifying and Elevating Constraints
  - Continually identify system constraints and improve work capacity.
  - Follow Dr. Goldratt's "five focusing steps."

Making Work Visible

- Feedback Loops
- Mechanisms for providing timely feedback at every stage of the value stream.
  - Automated test results.
  - Deployment success metrics.
- Role of Leadership in Visibility
  - Promote transparency in workflows.
  - Encourage cross-functional collaboration.
  - Prioritize continuous improvement.

Making Work Visible

- Metrics for Evaluating the Value Stream
  - Lead time.
  - Cycle time.
  - Deployment frequency.
  - Change failure rate.
- Automation to Enhance Visibility
  - Jenkins for CI/CD.
  - ELK Stack for log aggregation.
  - Prometheus for monitoring.

Making Work Visible

- Common Challenges in Visualization
  - Lack of standardized workflows.
  - Resistance to transparency.
  - Overwhelming data volume.
- Solutions:
  - Use consistent mapping techniques.
  - Provide training for teams.

Making Work Visible

- Continuous Improvement in Visibility
  - Conduct regular retrospectives.
  - Update value stream maps frequently.
  - Incorporate new tools and technologies.
- Future Trends in Value Stream Visualization
  - AI-driven workflow optimization.
  - Real-time data visualization.
  - Increased focus on developer experience.

Accelerate Feedback

- Learn how to accelerate feedback loops to improve quality and reduce time-to-market.
  - Enhance visibility of feedback across teams.
  - Automate feedback mechanisms.
  - Foster a culture of continuous learning and improvement.
- Why Accelerate Feedback?
  - Detect and fix issues early.
  - Improve collaboration between teams.
  - Enhance customer satisfaction.

Accelerate Feedback

- Feedback Loops in DevOps
- Continuous cycles of information flow to provide insights on performance and issues.
  - Development feedback.
  - Operational feedback.
  - Customer feedback.
- Development Feedback
  - Unit tests.
  - Code reviews.
  - CI/CD pipelines.
    - Run tests automatically after every commit.
    - Provide actionable feedback to developers.

Accelerate Feedback

- Operational Feedback
  - Monitoring and telemetry.
  - Incident reports.
  - Logs and alerts.
    - Centralize monitoring dashboards.
    - Automate alerting systems.
  - Tools: Prometheus, Grafana.
- Customer Feedback
  - Surveys and reviews.
  - User behavior analytics.
  - Support tickets.
    - Regularly analyze customer feedback.
    - Use feedback to prioritize features and fixes.

Accelerate Feedback

- Automating Feedback Mechanisms
  - Integrate feedback into CI/CD pipelines.
  - Automate metrics collection and visualization.
  - Use AI and ML for trend analysis.
- Shortening Feedback Cycles
  - Use small batch sizes for work.
  - Increase deployment frequency.
  - Conduct daily stand-ups for immediate feedback.

Accelerate Feedback

- Continuous Testing for Feedback
  - Unit Testing: Focus on individual components.
  - Integration Testing: Ensure components work together.
  - End-to-End Testing: Validate the entire system.
  - Tools: Selenium, JUnit.
- Monitoring for Real-Time Feedback
  - System uptime.
  - Error rates.
  - Latency and response times.
  - Tools: Datadog, New Relic.

Accelerate Feedback

- Feedback During Incident Management
  - Detect the issue.
  - Analyze root causes.
  - Apply fixes and capture lessons learned.
- Feedback in Retrospectives
  - Reflect on past performance to improve future workflows.
  - Focus on actionable insights.
  - Involve cross-functional teams.

Accelerate Feedback

- Feedback-Driven Decision Making
  - Collect relevant data.
  - Analyze trends and patterns.
  - Make informed decisions.
- Overcoming Barriers to Feedback
  - Resistance to transparency.
  - Overwhelming data volume.
  - Lack of actionable insights.
    - Foster a culture of openness.
    - Use tools for data visualization.

Accelerate Feedback

- Continuous Improvement Through Feedback
  - Regularly review processes.
  - Update tools and workflows based on feedback.
  - Encourage a growth mindset across teams.
- Leadership's Role in Accelerating Feedback
  - Promote a culture of learning.
  - Provide resources for automation.
  - Act on feedback promptly.
- Metrics to Measure Feedback Effectiveness
  - Time to detect and resolve issues.
  - Frequency of customer feedback integration.
  - Number of improvements based on feedback.

Optimize for Resilience

- Learn how to design systems and practices that enhance resilience in the face of failures.
  - Build fault-tolerant systems.
  - Recover quickly from incidents.
  - Foster a culture of resilience.
- What is Resilience?
  - The ability of a system to maintain functionality despite failures or adverse conditions
    - Fault tolerance.
    - Rapid recovery.
    - Continuous improvement.

Optimize for Resilience

- Importance of Resilience in DevOps
  - Reduces downtime and customer impact.
  - Increases trust and confidence in systems.
  - Enables faster recovery from failures.
- Designing for Failure
  - Assume failures will happen.
  - Build redundancy into systems.
  - Use distributed architectures.

Optimize for Resilience

- Redundancy and Fault Tolerance
  - Use load balancers to distribute traffic.
  - Implement failover mechanisms.
  - Store backups in multiple regions.
- Monitoring and Observability
  - Monitor critical metrics (e.g., uptime, error rates).
  - Use observability tools for deeper insights.
  - Set up alerts for anomalies.
  - Tools: Prometheus, Grafana.

Optimize for Resilience

- Incident Response and Recovery
  - Detect and analyze incidents promptly.
  - Contain and mitigate impacts.
  - Recover systems to a known good state.
- Automating Recovery Processes
  - Use automated scripts for failover.
  - Schedule regular disaster recovery drills.
  - Implement self-healing systems.
  - Tools: Kubernetes, AWS Lambda.

Optimize for Resilience

- Chaos Engineering
  - Experimenting on systems to identify weaknesses.
    - Simulate failures in production.
    - Measure system response and recovery.
- Resilience in CI/CD Pipelines
  - Ensure pipeline reliability through automated testing.
  - Use canary deployments to test in production.
  - Implement rollback mechanisms for failed deployments.

Optimize for Resilience

- Building a Resilient Culture
  - Encourage blameless post-mortems.
  - Foster collaboration across teams.
  - Reward proactive problem-solving.
- Measuring Resilience
  - Mean Time to Recovery (MTTR).
  - Failure rates.
  - System uptime.
  - Tools: Datadog, ELK Stack.

Optimize for Resilience

- Continuous Improvement in Resilience
  - Regularly review incidents and lessons learned.
  - Update processes based on feedback.
  - Invest in training and tools.
- Overcoming Challenges in Resilience
  - Resistance to chaos engineering.
  - Lack of automation.
  - Limited cross-team communication.
- Solutions:
  - Demonstrate the value of resilience practices.
  - Gradually introduce automation tools.

Optimize for Resilience

- Resilience in Cloud-Native Architectures
  - Auto-scaling for demand fluctuations.
  - Distributed data storage.
  - Container orchestration for failover.
  - Tools: Kubernetes, Docker Swarm.
- Leadership's Role in Resilience
  - Prioritize resilience in planning.
  - Allocate resources for training and tools.
  - Lead by example during incidents.

Shifting Security Left

Key Practices for Shifting Left

- Integrating security into the software delivery process instead of treating it as a separate downstream phase.
- Improves continuous delivery and overall delivery performance.
- Security Reviews:
  - Conducted for all major features without slowing down development.
- Integration:
  - Embed security across the entire software delivery lifecycle, from development to operations.
- Collaboration:
  - Involve infosec experts in application design and software demonstrations.

Automation in Security

- Automated Testing:
  - Ensure security features are included in the automated test suite.
- Developer Tools:
  - Provide preapproved libraries, packages, toolchains, and processes.
- Goal:
  - Make it easy for developers to implement security best practices.

Shifting Responsibilities

- From Inspection to Enablement:
  - Infosec teams empower developers to build security into the software.
- Benefits:
  - Reduces significant architectural problems and rework.
  - Aligns with frequent deployment cycles.
- Challenge:
  - Security teams can't perform reviews for every deployment in high-frequency environments.
- Solution:
  - Involve infosec professionals throughout development to improve communication and workflows.

Developer Empowerment

- Key Actions:
  - Provide tools, training, and support for developers to make secure choices.
  - Create easy-to-consume resources for security best practices.
- Outcome:
  - Improves delivery performance and security outcomes.

Benefits of Building Security In

- Efficiency Gains:
  - High performers spend 50% less time remediating security issues than low performers.
- Proactive Security:
  - Building security in daily reduces time spent addressing issues later.
- Enhanced Collaboration:
  - Improves communication between developers and infosec teams.
- Key Outcomes:
  - Faster deployments with fewer bottlenecks.
  - Better alignment of security and compliance with DevOps principles.

High-Performance Security Practices

- Traits of High Performers:
  - Seamless integration of infosec in daily work.
  - Strong collaboration between infosec and development teams.
  - Continuous delivery with minimal security disruptions.

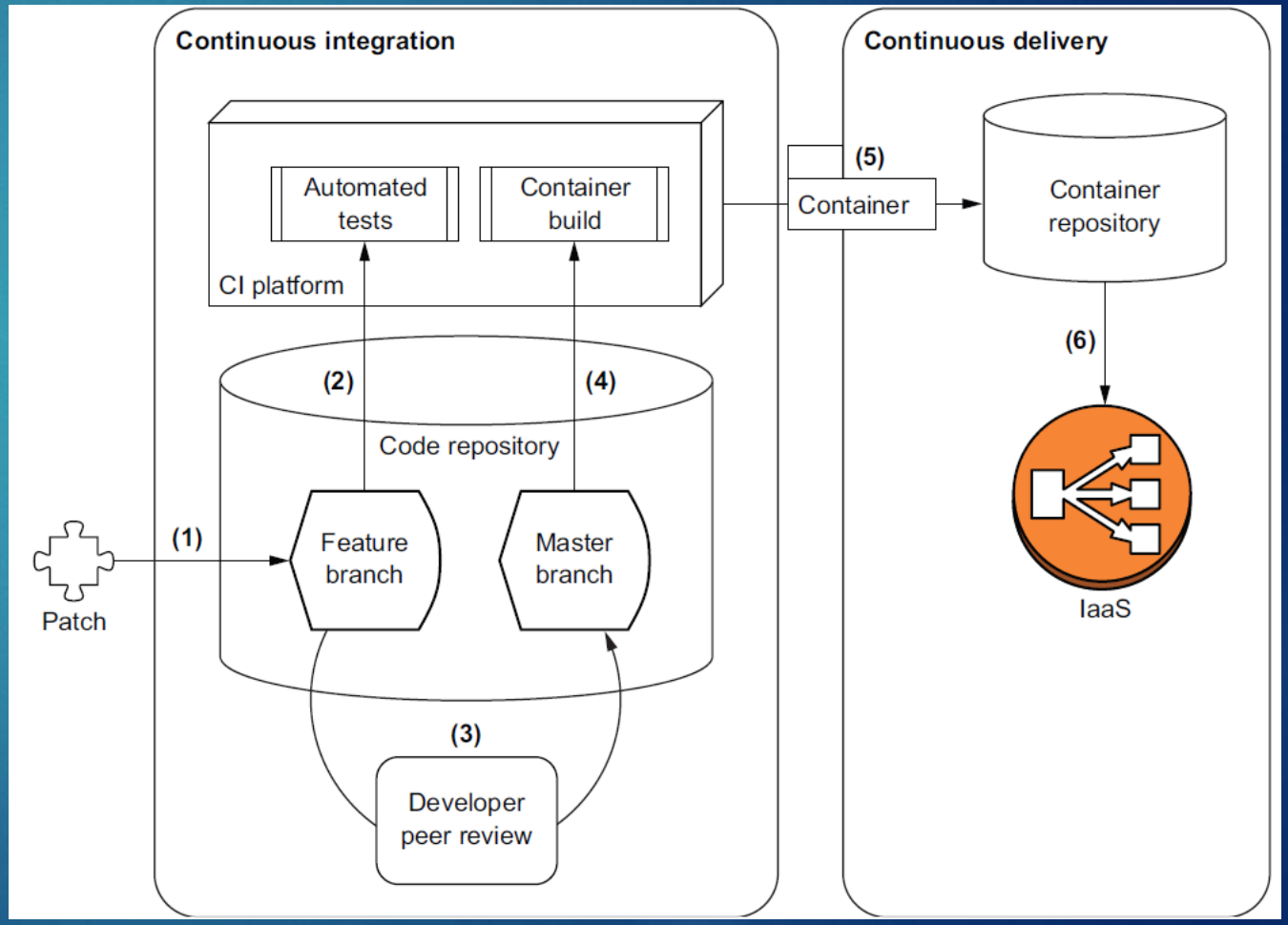
CI / CD pipeline

CI/CD Pipeline

- Continuous Integration (CI): Automating the integration of code changes.
- Continuous Delivery (CD): Automating the release process.
- Continuous Deployment: Fully automating the deployment to production.
- Key Tools in DevSecOps
  - Version Control Systems: GitHub, GitLab, Bitbucket.
  - CI/CD Tools: Jenkins, Travis CI, GitLab CI/CD.
  - Automation Tools: Ansible, Docker, Kubernetes.

Implementation Roadmap

- Developer submits a patch to a feature branch.
- Automated tests run against the application.
- Peer review merges the patch into the master branch.
- The application is packaged into a container.
- The container is published to a registry.
- Production infrastructure retrieves and deploys the container.

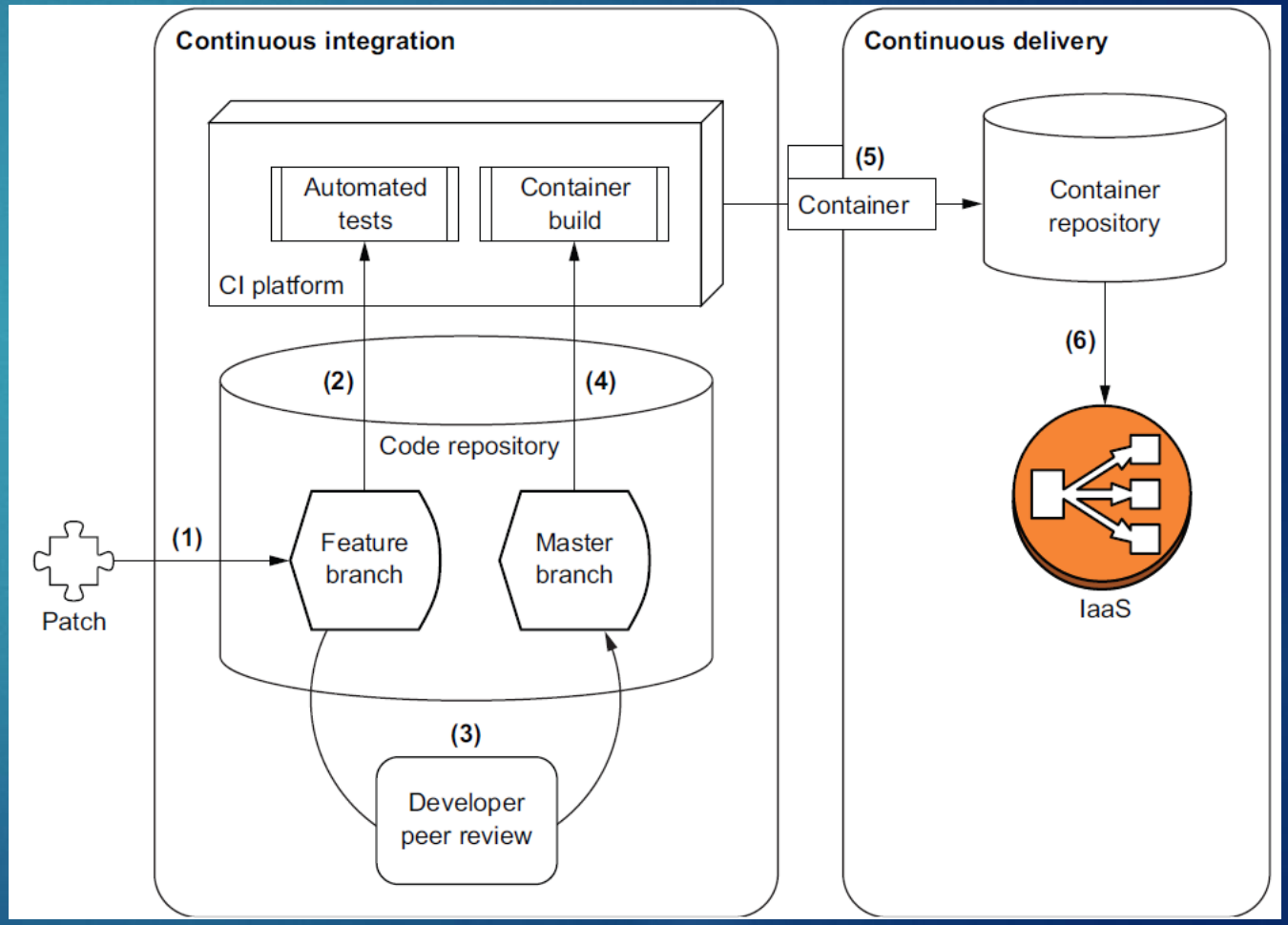


Building a DevOps Pipeline

- Creating a functional pipeline requires seamless integration of key components.
  - Source Code Repository
  - CI Platform
  - Container Repository
  - IaaS Provider

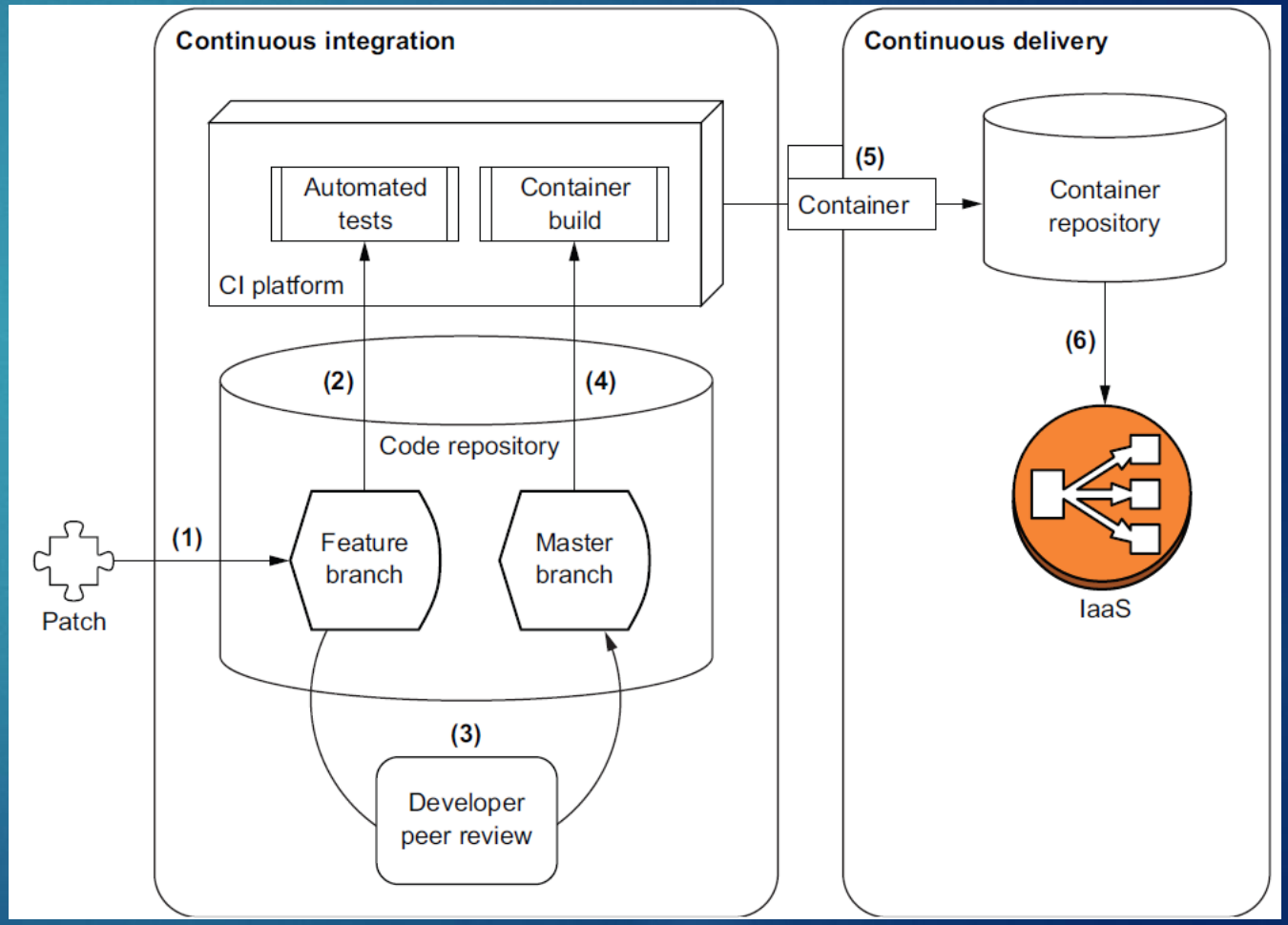
Code Repository

- A central repository for source code management.
- Example Tool: GitHub
  - Utilizes webhooks for triggering actions.
- Process:
  - Code is pushed to the repository.
  - Webhooks notify the CI platform of changes.



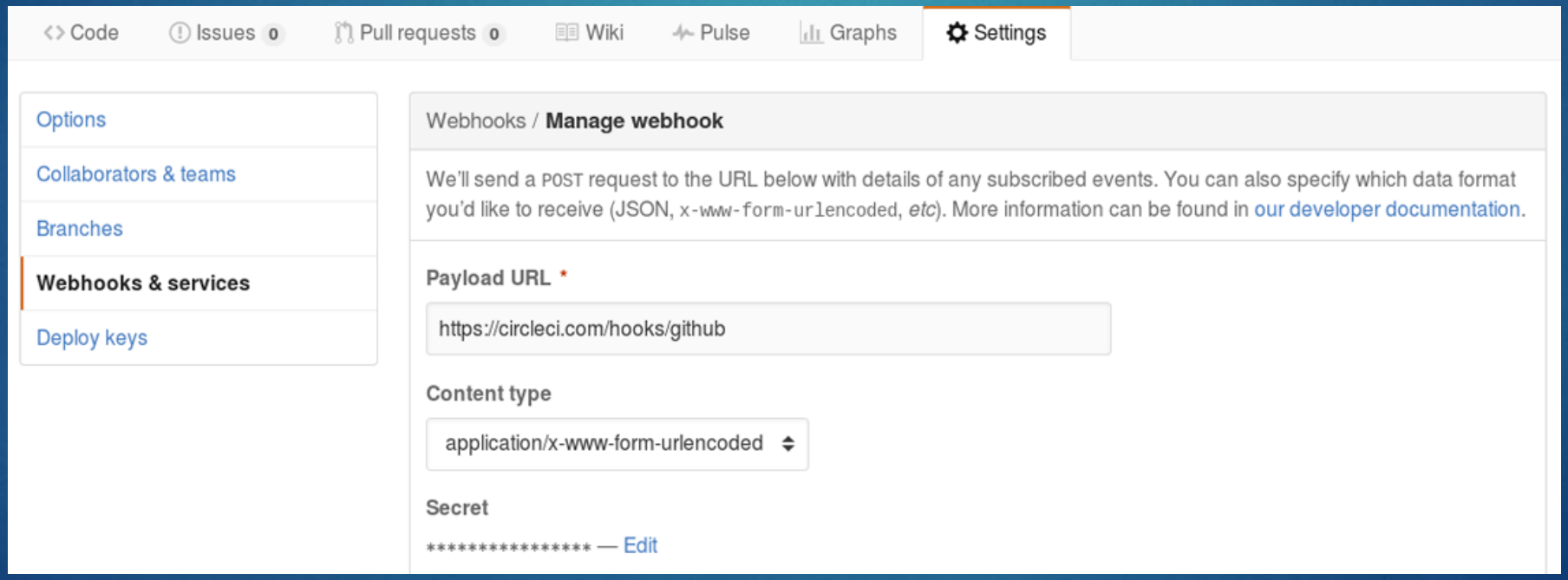
Continuous Integration (CI)

- A workflow to integrate and test code changes frequently.
- Process:
  - Code submission triggers automated unit and integration tests.
  - Peer review ensures quality.
- Example Tool: CircleCI
  - Other tools: Jenkins
  - Automates testing and container builds.



Circle CI

- Circle CI is a CI platform
- It can integrate with Git using web hooks



Circle CI

- Circle CI configuration file needs to be stored in the repo

Listing 2.1 `config.yml` **configures CircleCI for the application**

```
version: 2
jobs:
 build:
  working_directory:
➥/go/src/github.com/Securing-DevOps/invoicer-chapter2
```

Configures a working directory to build the Docker container of the application

```
 docker:
  - image: circleci/golang:1.8
 steps:
  - checkout
  - setup_remote_docker
```

Declares the environment the job will run on

Circle CI


```
- run:
    name: Setup environment
    command: |
      gb="/src/github.com/${CIRCLE_PROJECT_USERNAME}";
      if [ ${CIRCLE_PROJECT_USERNAME} == 'Securing-DevOps' ]; then
        dr="securingdevops"
      else
        dr=$DOCKER_USER
      fi
      cat >> $BASH_ENV << EOF
      export GOPATH_HEAD="$(echo ${GOPATH}|cut -d ':' -f 1)"
      export GOPATH_BASE="$(echo ${GOPATH}|cut -d ':' -f 1)${gb}"
      export DOCKER_REPO="$dr"
      EOF

- run: mkdir -p "${GOPATH_BASE}"
- run: mkdir -p "${GOPATH_HEAD}/bin"

- run:
    name: Testing application
    command: |
        go test \
        github.com/${CIRCLE_PROJECT_USERNAME}/${CIRCLE_PROJECT_REPONAME}
```

Environment variables needed to build the application

Runs the unit tests of the application

Circle CI

If changes are applied to the master branch, builds the Docker container of the application

Builds the application binary

Logs into the Docker Hub service

```
- deploy:
    command: |
     if [ "${CIRCLE_BRANCH}" == "master" ]; then
      docker login -u ${DOCKER_USER} -p ${DOCKER_PASS};
      go install --ldflags '-extldflags "-static"' \
      github.com/${CIRCLE_PROJECT_USERNAME}/${CIRCLE_PROJECT_REPONAME};
      mkdir bin;
      cp "$GOPATH_HEAD/bin/${CIRCLE_PROJECT_REPONAME}" bin/invoicer;
      docker build -t ${DOCKER_REPO}/${CIRCLE_PROJECT_REPONAME} .;
      docker images --no-trunc | awk '/^app/ {print $3}' | \
        sudo tee $CIRCLE_ARTIFACTS/docker-image-shasum256.txt;
      docker push ${DOCKER_REPO}/${CIRCLE_PROJECT_REPONAME};
     fi
```

Builds a container of the application using a Dockerfile

Pushes the container to Docker Hub

Creates a Git feature branch

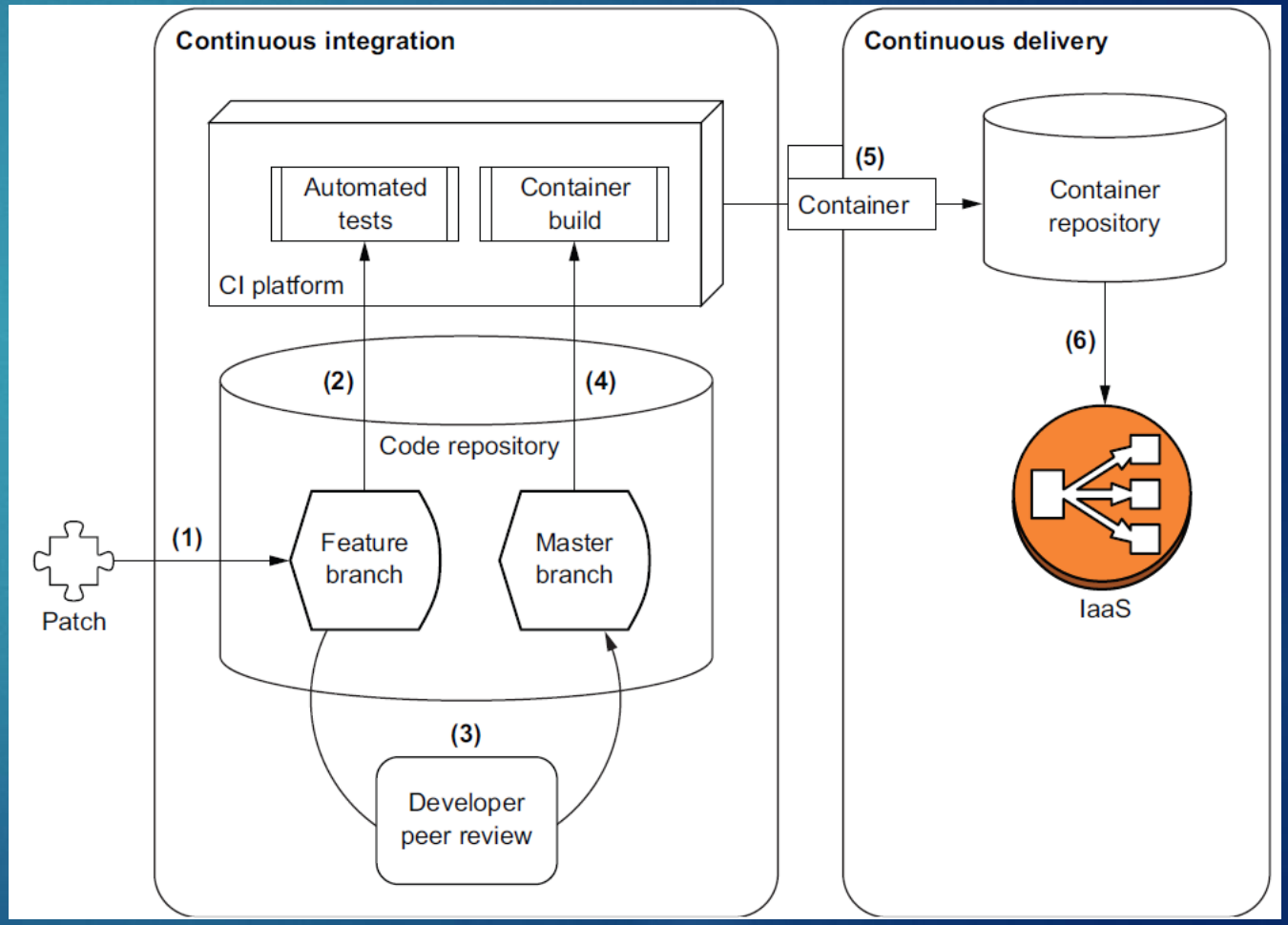
Adds config.yml to the branch

Pushes changes to the code repository

```
$ git checkout -b featbr1
$ git add .circleci/config.yml
$ git commit -m "initial circleci conf"
$ git push origin featbr1
```

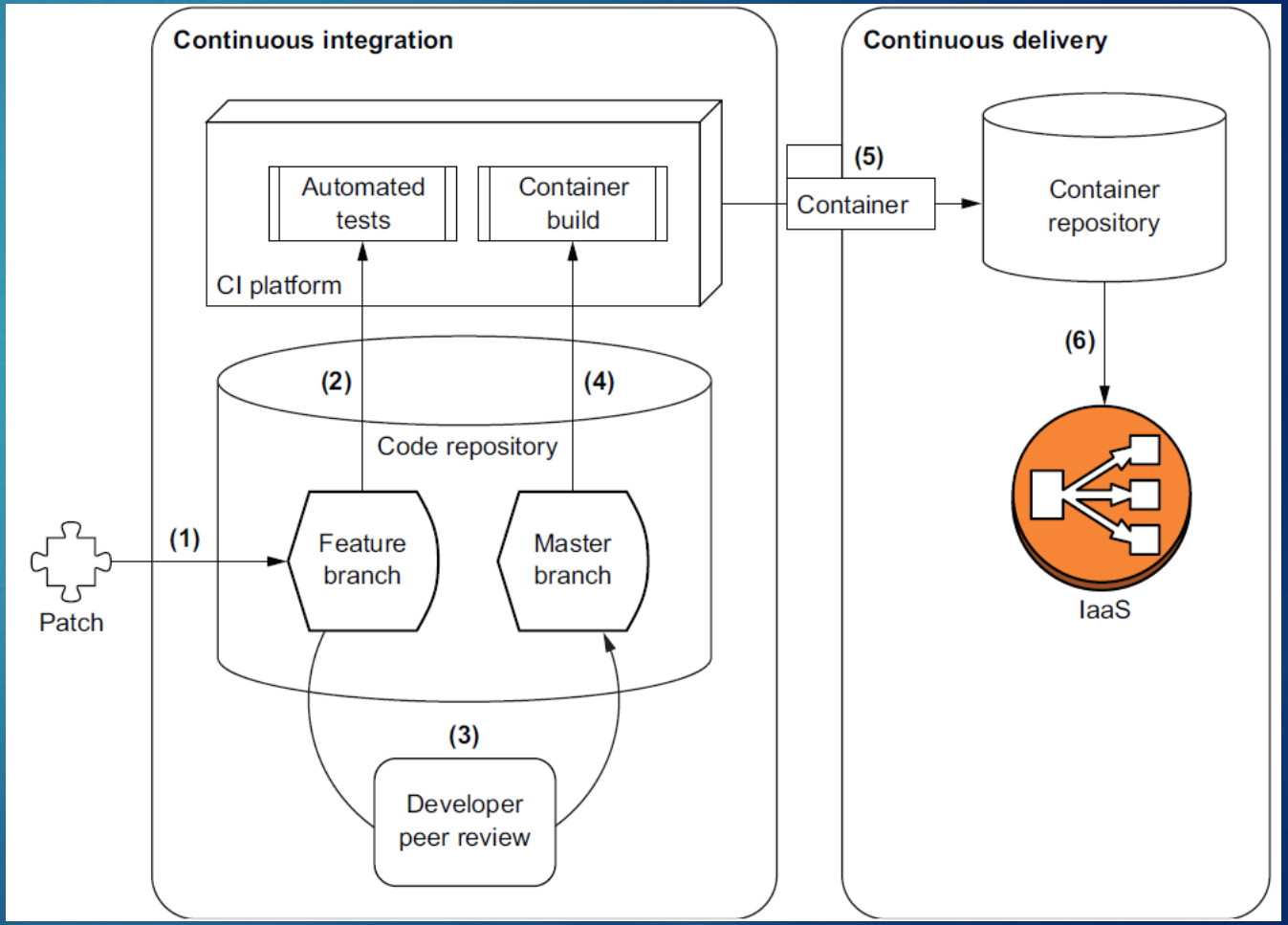
Continuous Delivery (CD)

- Automates the deployment of code to production.
- Key Actions:
  - Retrieve the latest code version.
  - Package and deploy to staging environments.
  - Promote to production after testing.



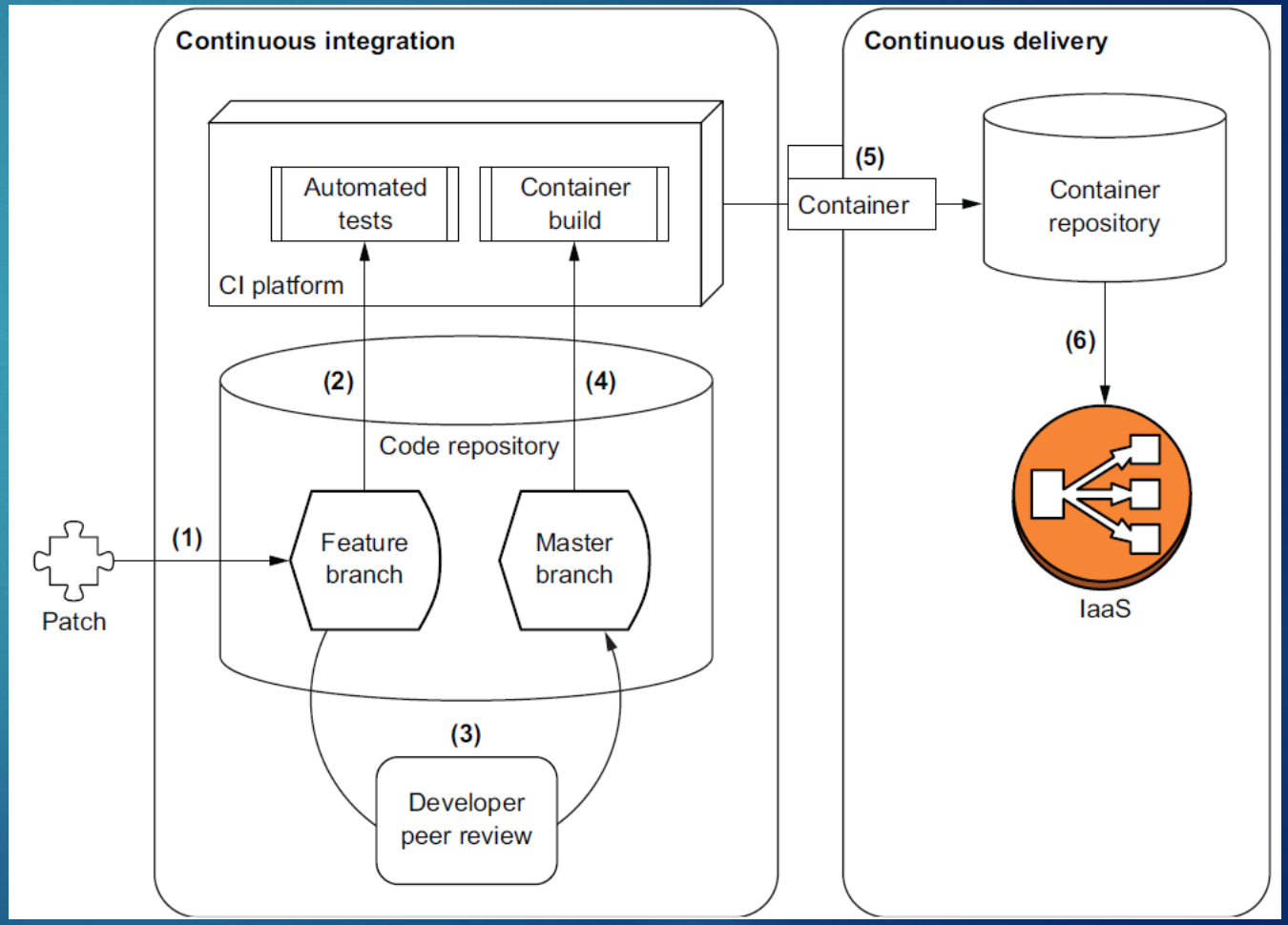
Containers and Docker

- Containers encapsulate applications and their dependencies for consistent deployment.
- Key Benefits:
  - Isolation of dependencies.
  - Simplified deployments.
- Example Tool: Docker
  - Builds and manages containers.
  - Dockerfile specifies container configurations.



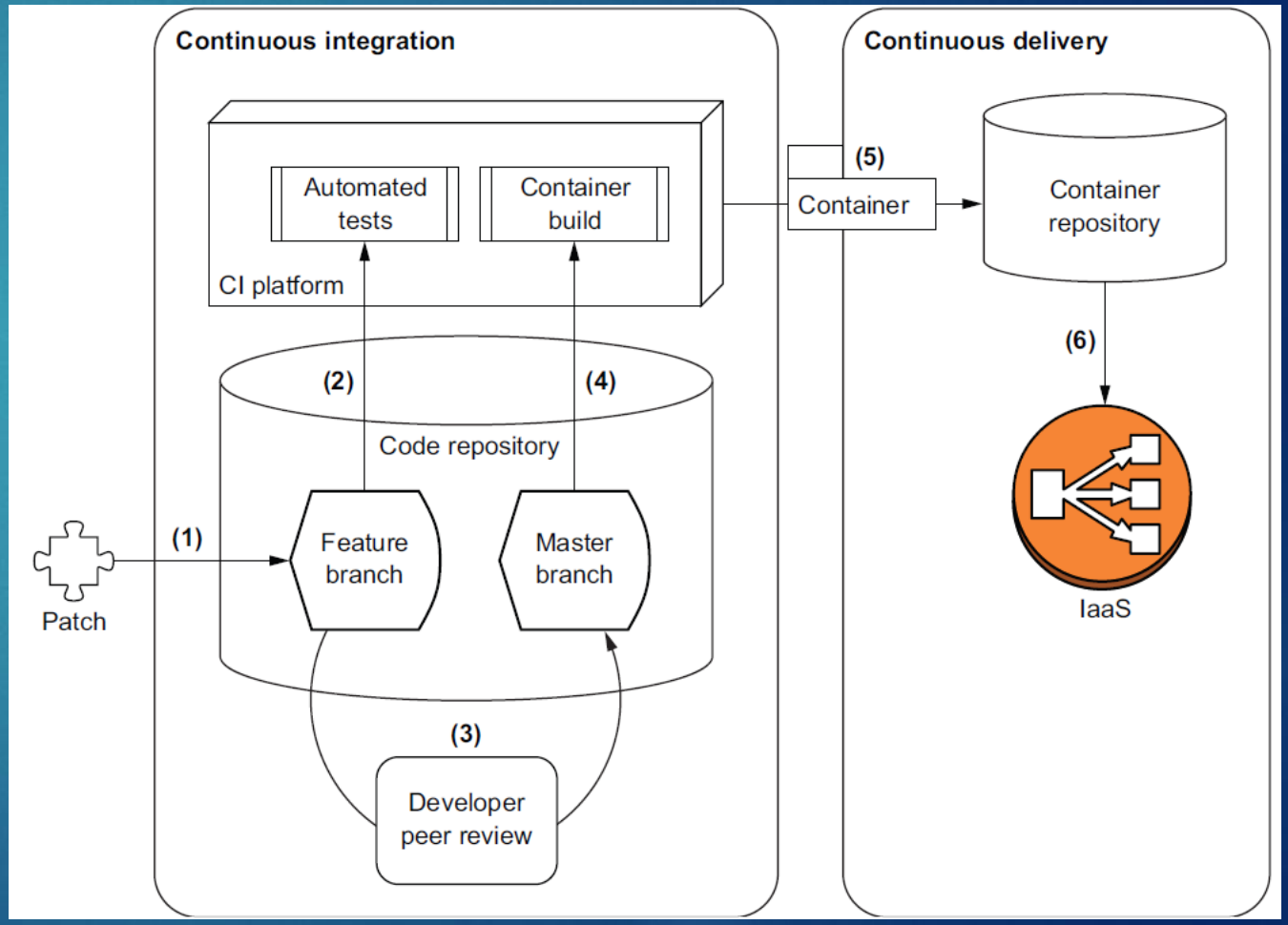
Container Repository

- A storage location for container images.
- Example Tool: Docker Hub
  - Hosts container images.
  - Integrates with CI platforms for automated publishing.
- Process:
  - Build container using Docker.
  - Push container to the repository.



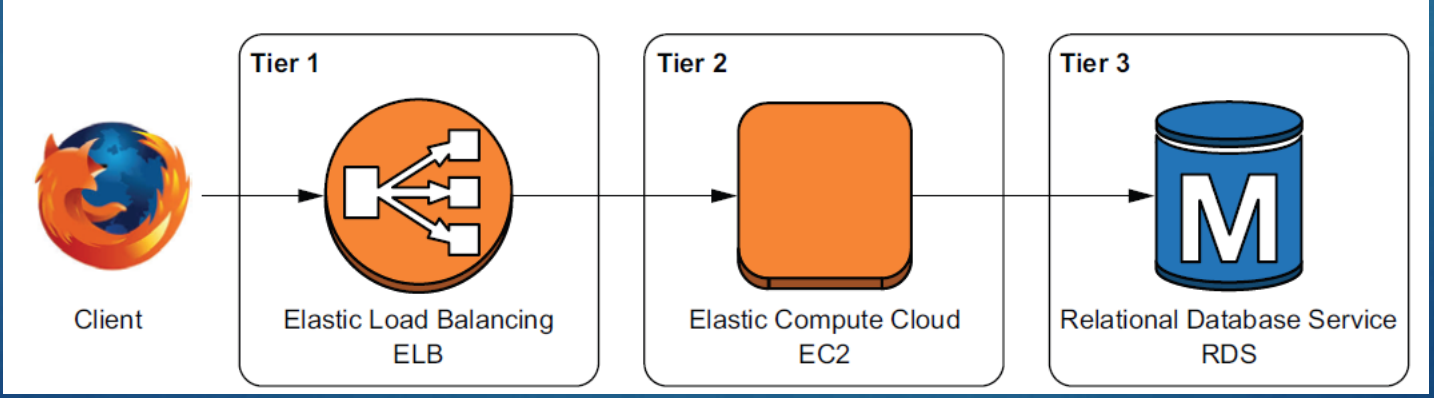
Infrastructure as a Service (IaaS)

- Definition: Cloud-managed virtualized infrastructure.
- Example Tool: AWS
  - Manages infrastructure components like VMs and databases.
  - Elastic Beanstalk simplifies container hosting.



Production Infrastructure

- Example Configuration:
  - Three-tier architecture:
    - Load balancer (Elastic Load Balancer).
    - Compute nodes (Elastic Compute Cloud).
    - Database backend (Relational Database Service).



Security Considerations

- Webhooks: Use low-privilege accounts for CI platforms.
- Credentials Management:
  - Store secrets securely.
  - Avoid hardcoding sensitive information.
- Infrastructure Security: Ensure network segmentation with security groups.