innovate achieve lead

BITS Pilani
Pilani Campus

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

By Prof A R Rahman
CSIS Group WILP

innovate achieve lead

BITS Pilani
Pilani Campus

Course Name: Introduction to DevOps
Course Code : CSI ZG514/SE ZG514

By Prof A R Rahman
CSIS Group WILP

CS – 4 DevOps Dimensions

Coverage

- DevOps – People
- Team structure in a DevOps
- Transformation to Enterprise DevOps culture
- Building competencies, Full Stack Developers
- Self-organized teams, Intrinsic Motivation
- Technology in DevOps (Infrastructure as code, Delivery Pipeline, Release Management)
- Tools/technology as enablers for DevOps

DevOps – People

- Addresses the people aspect of adopting DevOps, including creating the necessary culture.

- An organization may adopt the most efficient processes or automated tools possible, but they're useless without the people who eventually must execute those processes and use those tools.

DevOps– People

- DevOps CULTURE
- (1) Identifying business objectives
  - Creating a culture is getting everyone headed in the same direction and working toward the same goal
  - Identifying common business objectives for the team and the organization as a whole.
  - identify goals that it wants to achieve; then it can develop a common set of milestones toward those goals for different teams of stakeholders to use
  - Collaboration and communication across stakeholders

DevOps– People

- (2) Create an environment of sharing

  - leaders of the organization to work with their teams to create an environment and culture of collaboration and sharing.

  - Leaders must remove any self-imposed barriers to cooperation.

  - Typical measurements reward operations teams for uptime and stability, and reward developers for new features delivered, but they pit these groups against each other.

DevOps– People

- (2) Create an environment of sharing
  - The leaders of the organization should further encourage collaboration by improving visibility.
  - Establishing a common set of collaboration tools is essential, especially when teams are geographically distributed and can't work together in person.
  - Giving all stakeholders visibility into a project's goals and status is crucial for building a DevOps culture based on trust and collaboration.

DevOps– People

- DEVOPS TEAM

  - The arguments for and against having a separate DevOps team are as old as the concept itself.

  - DevOps liaison teams, which resolve any conflicts and promote collaboration.

  - Such a team may be an existing tools group or process group, or it may be a new team staffed by representatives of all teams that have a stake in the application being delivered.

  - DevOps team, your most important goal is to ensure that it functions as a Center of Excellence that facilitates collaboration without adding a new layer of bureaucracy or becoming the team that owns addressing all DevOps related problems

DevOps– People

Key competencies to succeed

- Culture – Shared organizational assumptions to resolve problems
- Leadership
  - visioning, strategic management, flexibility, and the ability to inspire others to innovate and perform
  - individuals with advanced communications skills, the knowledge of diverse cultures, and people who behave collaboratively when working in teams

DevOps– People

Key competencies to succeed

- Resource/management
  - effective management of available resources
  - collaboration across multi-functional teams is at the heart of DevOps, delivering an accountable business outcome
- Subject matter experts (SME’s) in different areas like – Automation, Build & Release management, CI/CD, Containers, Security, Cloud (AWS/GCP/Azure)
- DevOps engineers who are members from different department (development, QA, Operations)

Create the ideal DevOps team structure

- A solid DevOps platform needs a solid DevOps team structure to achieve maximum efficiency.
- Building a robust DevOps team structure is not merely about assembling a group of individuals with technical skills; it's about fostering a DevOps culture that emphasizes collaboration, continuous improvement, and efficiency across the product lifecycle.
- As organizations look to streamline their software development process, understanding the roles of a DevOps engineer, the development team, and how automation tools can enhance productivity is vital.
- This holistic view helps in shaping a DevOps structure that aligns with the overarching goals of reliable software delivery and a productive work environment.

Create the ideal DevOps team structure

- Several factors come into play when it comes to team structure:
- Existing silos: Are there product sets/teams that work independently?
- Technical leadership: Are group managers set up to achieve DevOps goals?
- Changing roles: Ops tasks have bled into dev roles, security teams are working with everyone, and technology is changing. Expect to regularly re-evaluate everything.
- Continuous improvement: A DevOps team will never be a “one and done.” Iteration will be required.

Types of silos

- The division of Dev and Ops into separate teams often leads to challenges in the deployment process.

- However, embracing a DevOps culture where common tools are integrated can bridge these gaps.

- A few DevOps scenarios in great detail, but we'll discuss just a few of the silos specifically and how they impact an organization.

Dev and ops are completely separate

- Skelton refers to this as a classic “throw it over the wall” team structure and, as implied, it’s not the most effective DevOps strategy.

- Both teams work in their bubbles and lack visibility into the workflow of the other team.

- This complete separation lacks collaboration, visibility, and understanding – vital components of what effective DevOps should be.

- What happens is essentially blame-shifting: “We don’t know what they are doing over there, we did our part and now it’s up to them to complete it,” and so on.

DevOps middleman

- In this team structure, there are still separate dev and ops teams, but there is now a “DevOps” team that sits between, as a facilitator of sorts.

- This is not necessarily a bad thing and Skelton stresses that this arrangement has some use cases.

- For example, if this is a temporary solution with the goal being to make dev and ops more cohesive in the future, it could be a good interim strategy.

Ops stands alone

- In this scenario, dev and DevOps are melded together while ops remains siloed.
- Organizations like this still see ops as something that supports the initiatives for software development, not something with value in itself.
- Organizations like this suffer from basic operational mistakes and could be much more successful if they understand the value ops brings to the table.

What can DevOps team leadership do?

- To break down DevOps team silos requires leadership at all levels.
- Start by asking each group to surface the major areas of friction and then identify leaders in each group – dev, ops, security, test.
- Each leader should work individually and together on all of the friction points.
- The importance of communication can’t be overstated: Teams need to hear regular feedback about all aspects of their roles.
- It might also be helpful to insert “champions” into struggling groups; they can model behaviors and language that facilitate communication and collaboration.

DevOps roles are blurring

- Technology advances from multicloud to microservices and containers also play a role when it comes to defining the right DevOps team structure.

- In a 2020 Global DevSecOps Survey, 83% of respondents said their teams are releasing code more quickly but they also told us their roles were changing, dramatically in some cases.

- Devs today are creating, monitoring, and maintaining infrastructures, roles that were traditionally the province of ops pros.

DevOps roles are blurring

- Ops are spending more time managing cloud services, while security team members are working on cross-functional teams with dev and ops more than ever before.

- Obviously, the software development lifecycle today is full of moving parts, meaning that defining the right structure for a DevOps team will remain fluid and in need of regular re-evaluation.

Remember to iterate

- Iteration is one of best practices and it’s something we need practice a lot when it comes to our own DevOps team structure.

- Dev teams are organized into stages because there would be separate products at any company and require their own autonomy.

- We should also have other functional DevOps groups besides “Dev” that manage other aspects of our product.

Remember to iterate

- We have a reliability group that manages uptime and reliability.
- A quality department, and a distribution team, just to name a few.
- The way that we make all these pieces fit together is through our commitment to transparency and our visibility through the entire SDLC.
- But we also tweak (i.e. iterate on) this structure regularly to make everything work.
- The bottom line: Plan to build your DevOps team, and then re-think it, and re-think it some more.
- The benefits in faster code releases and happier team members will make it worthwhile.

Remember to iterate

- The journey to optimizing a DevOps team structure is iterative, reflecting the continual advancements in DevOps processes and tools.

- Each element plays a crucial role in the team's success.

- By breaking down traditional silos and integrating roles within DevOps teams, organizations can foster a more cohesive and efficient environment.

- Ultimately, the key to sustained improvement lies in regularly re-evaluating and refining the DevOps structure to keep pace with the fast-evolving demands of software production and deployment.

- This commitment not only speeds up the software development process but also builds a more resilient and responsive organization.

How to lead a successful DevOps transformation (without burning out your teams)

- DevOps transformation has become essential for organizational survival.
- Yet despite the clear benefits, many transformation efforts fail to deliver expected results.
- Engineering leaders find themselves caught between executive demands for faster delivery and teams resistant to yet another disruptive change.
- The difference between successful transformations and failed experiments often comes down to leadership approach.
- This guide provides a practical blueprint for engineering directors and DevOps leaders who need to drive meaningful transformation without sacrificing team well-being.

What is DevOps transformation (and why it's harder than it sounds)?

- DevOps transformation is the systematic evolution of how you build, deploy, and operate software. It's not about installing new tools.

- It's about changing mindsets, collaboration models, and delivery rhythms.

- True transformation requires rewiring how your development and operations teams collaborate, breaking down traditional silos that separate those who build software from those who run it.

- This fundamentally shifts responsibilities, processes, and most critically, culture.

What makes DevOps transformation particularly challenging?

- Cultural resistance exceeds technical hurdles.
- Engineers comfortable with established workflows naturally resist changes that disrupt their routines.
- Mid-transformation chaos feels unavoidable.
- As you migrate from legacy to modern practices, you'll temporarily run parallel systems that can increase complexity.
- Leadership alignment is difficult but necessary.
- When executives and engineering leaders differ on transformation goals, teams receive mixed signals that hinder progress.

What makes DevOps transformation particularly challenging?

- Measuring success isn't straightforward.

- Unlike pure technical projects, DevOps transformation success indicators blend technical metrics with cultural and business outcomes.

- The essence of DevOps extends far beyond continuous integration and deployment tools.

- It represents a fundamental shift in how teams collaborate, share responsibility, and deliver value.

Why DevOps transformation is a business imperative (not just an IT project)

- DevOps transformation directly impacts business survival, not just technical efficiency.

- Companies that master DevOps practices consistently outperform their competitors across critical business metrics:

- Revenue impact: Companies with mature DevOps practices generate more revenue growth than their competitors.

- This stems from releasing features faster, responding to market changes quicker, and reducing lost revenue from outages.

Why DevOps transformation is a business imperative (not just an IT project)

- **Customer satisfaction:** When you deploy smaller changes more frequently, you reduce risk while delivering value continuously.

- This translates to more responsive product development and fewer customer-impacting issues.

- **Innovation capacity:** By automating routine work and reducing toil, DevOps practices free your engineers to focus on strategic innovations rather than maintenance and firefighting.

- **Talent acquisition and retention:** Engineers increasingly evaluate potential employers based on their technical practices.

- Organizations with modern DevOps approaches attract and retain top talent more effectively.

Why DevOps transformation is a business imperative (not just an IT project)

- Competitive resilience: Market demands change rapidly.
- DevOps-mature organizations can pivot quickly, respond to competitive threats, and capitalize on emerging opportunities faster than competitors.

Consider two contrasting scenarios:

- Company A: Deploys monthly with numerous quality issues, spends 60% of engineering time on maintenance, and loses key talent due to frustration with rigid processes.

- Company B: Deploys daily with minimal issues, spends 80% of time on innovation, and attracts top engineers who value their modern practices.

- Which organization will dominate the market by 2026?

Consider two contrasting scenarios:

- The business case becomes clear: DevOps transformation isn't about engineering preferences—it's about creating the technical capabilities necessary for business growth, innovation, and competitive advantage.

- For engineering leaders in particular, successful DevOps transformation represents one of the highest-impact contributions you can make to organizational success.

- It directly connects technical excellence to business outcomes in a measurable, undeniable way.

The 10 essential steps to a high-impact DevOps transformation

- DevOps transformation isn't a linear journey; it's iterative, courageous, and requires strategic leadership.

- These ten steps provide a framework that balances technical changes with the equally critical people and process components.

- 1. Baseline current workflows, bottlenecks, and team sentiment

- Before making changes, establish a clear picture of your current state.

- This requires both quantitative and qualitative assessment:

The 10 essential steps to a high-impact DevOps transformation

- Measure your current DORA metrics to establish a performance baseline across deployment frequency, lead time, change failure rate, and time to restore.

- Map your value streams to visualize how work flows from idea to production, identifying handoffs, wait states, and approval gates that create bottlenecks.

- Conduct anonymous developer surveys to understand team pain points, perceived barriers, and cultural readiness for change.

The 10 essential steps to a high-impact DevOps transformation

- Inventory your existing toolchain to identify fragmentation, manual steps, and integration challenges.

- This baseline serves two critical purposes: it identifies your highest-impact improvement opportunities and provides clear before/after metrics to demonstrate transformation success.

The 10 essential steps to a high-impact DevOps transformation

- 2. Define a vision that ties engineering goals to business outcomes

- Effective DevOps transformation needs a compelling vision that connects technical practices to business value. Your vision should:
- Articulate how technical improvements drive specific business outcomes like faster time-to-market, improved customer satisfaction, or increased innovation capacity.
- Set ambitious but achievable goals with clear timelines and success metrics.
- Establish a "north star" for decision-making that guides prioritization when you face competing options.
- Use language that resonates with both technical teams and executives to ensure alignment across organizational layers.
- Your vision becomes the foundation for communication, prioritization, and measuring progress throughout the transformation journey.

The 10 essential steps to a high-impact DevOps transformation

- 3. Build cross-functional squads focused on product delivery

- Traditional department boundaries between development, operations, QA, and security create handoffs that slow delivery and diffuse responsibility. Restructure into cross-functional squads that:
- **Align team composition with product or service boundaries** rather than technical specialties.
- **Embed operations, security, and quality engineers** directly within development teams.
- **Grant end-to-end ownership** from development through production support.
- **Establish shared metrics and goals** that reinforce collective responsibility for outcomes.

- This structural reorganization breaks down silos, accelerates decision-making, and creates shared ownership for both speed and stability.

The 10 essential steps to a high-impact DevOps transformation

- 4. Prioritize psychological safety before tooling changes

- Technical changes fail without a foundation of psychological safety: the confidence that team members can take risks without fear of blame. Before implementing new tools or processes:
- Establish blameless postmortems that focus on system improvements rather than individual errors.
- Recognize and reward learning from failure to encourage experimentation.
- Create forums for honest feedback where teams can voice concerns without repercussion.
- Model vulnerability as a leader by acknowledging mistakes and demonstrating a growth mindset.

- Psychological safety enables the transparency, experimentation, and continuous learning essential for DevOps success.

The 10 essential steps to a high-impact DevOps transformation

- 5. Establish CI/CD pipelines and testing gates

- With the foundational cultural elements in place, begin building the technical infrastructure that enables rapid, reliable delivery:
- Implement continuous integration with automated build verification testing to detect issues early.
- Develop deployment pipelines that standardize the path to production across teams.
- Automate security scanning and compliance checks to shift security left without slowing delivery.
- Create progressive testing strategies that balance speed with quality assurance.

- Rather than viewing CI/CD as a purely technical implementation, position it as enabling infrastructure that supports your teams' need for speed, feedback, and quality.

The 10 essential steps to a high-impact DevOps transformation

- 6. Automate infrastructure provisioning and rollback plans

- Infrastructure automation creates consistency, minimizes human error, and enables rapid recovery when issues emerge:
- Implement infrastructure as code (IaC) to manage environments through version-controlled configuration rather than manual processes.
- Establish golden paths for provisioning that incorporate security and compliance requirements by default.
- Create automated rollback mechanisms for rapid recovery when deployments fail.
- Standardize environments across development, testing, and production to eliminate "works on my machine" problems.

- Automating infrastructure changes creates reproducibility and safety nets that give teams confidence to deploy more frequently.

The 10 essential steps to a high-impact DevOps transformation

- 7. Choose an integrated DevOps toolchain — not a "Frankenstack“

- Tool fragmentation creates friction that slows delivery and frustrates teams. Build a cohesive toolchain that:
- Prioritizes integration capabilities over individual feature richness.
- Minimizes context switching by leveraging tools that work well together.
- Balances standardization with flexibility to meet unique team needs.
- Considers the entire development lifecycle from planning through monitoring.

- Your toolchain choices should reduce cognitive load for developers rather than adding complexity through poor integrations.

The 10 essential steps to a high-impact DevOps transformation

- 8. Measure everything: DORA, engagement, lead time for learning

- Create a comprehensive measurement framework that balances delivery performance with team health:
- Track DORA metrics to measure technical performance improvements.
- Monitor team engagement through regular pulse surveys and 1:1 conversations.
- Measure "lead time for learning" (how quickly you gather feedback on new features).
- Connect technical metrics to business outcomes to demonstrate transformation value.

- Effective metrics serve as navigation instruments that guide your transformation journey and demonstrate progress to stakeholders.

The 10 essential steps to a high-impact DevOps transformation

- 9. Launch a transformation pilot with clear success criteria

- Rather than attempting a big-bang transformation, start with a focused pilot:
- Select a team with the right mix of challenges and readiness to demonstrate transformation value.
- Define clear success criteria that blend technical, cultural, and business outcomes.
- Time-box the pilot to maintain focus and urgency.
- Provide additional support and resources to ensure pilot success.

- A well-executed pilot creates momentum, proves concepts, and builds organizational confidence in your transformation approach.

The 10 essential steps to a high-impact DevOps transformation

- 10. Commit to quarterly retrospectives and public wins

- Transformation is iterative, requiring regular reflection and visible success stories:
- Conduct quarterly retrospectives to evaluate progress, capture learnings, and adjust your approach.
- Celebrate and publicize wins to build momentum and demonstrate value.
- Share lessons learned openly across the organization to accelerate adoption.
- Adjust roadmaps based on feedback to address emerging challenges and opportunities.

- This commitment to reflection and adaptation prevents your transformation from becoming rigid or disconnected from evolving needs.

Building Competencies: Full Stack Developers in DevOps

- In today's rapidly evolving software landscape, organizations demand professionals who can seamlessly bridge development and operations.

- Full Stack Developers with DevOps competencies are uniquely positioned to deliver end-to-end solutions — from coding the application to automating deployments, ensuring scalability, reliability, and continuous delivery.

Building Competencies: Full Stack Developers in DevOps

- 2. Core Competencies Required
- a) Technical Skills
- Frontend Development: HTML5, CSS3, JavaScript frameworks (React, Angular, Vue).
- Backend Development: Node.js, Python (Flask/Django), Java (Spring Boot), PHP.
- Databases: Relational (MySQL, PostgreSQL) and NoSQL (MongoDB, Redis).
- Version Control: Git/GitHub/GitLab for collaboration.

Building Competencies: Full Stack Developers in DevOps

- b) DevOps Skills
- CI/CD Pipelines: Jenkins, GitHub Actions, GitLab CI for automated testing and deployment.
- Containerization: Docker for application packaging.
- Orchestration: Kubernetes or Docker Compose for scaling and resilience.
- Infrastructure as Code (IaC): Terraform, Ansible, or CloudFormation.
- Cloud Platforms: AWS, Azure, GCP for hybrid and multi-cloud deployments.
- Monitoring & Logging: Prometheus, Grafana, ELK Stack for observability.

Building Competencies: Full Stack Developers in DevOps

- c) Soft Skills
- **Collaboration:** Agile/Scrum methodology participation.
- **Problem Solving:** Debugging across the stack and resolving infrastructure bottlenecks.
- **Adaptability:** Keeping up with evolving DevOps tools and cloud-native practices.

Building Competencies: Full Stack Developers in DevOps

- 3. Learning Roadmap
- Foundations: Core programming + database skills.
- System Design & APIs: RESTful and GraphQL design.
- DevOps Fundamentals: CI/CD pipelines, Git workflows.
- Cloud Native Development: Deploying full-stack apps on AWS/Azure/GCP.
- Advanced DevOps: Microservices, container orchestration, and serverless computing.
- Security (DevSecOps): Secure coding, vulnerability scanning, compliance checks.

Building Competencies: Full Stack Developers in DevOps

- 4. Role in the Industry

- **Accelerating Delivery:** Ability to develop, test, and deploy quickly with automation.

- **Bridging Silos:** Connecting development and IT operations seamlessly.

- **Scaling Systems:** Designing fault-tolerant, cloud-ready applications.

- **Career Growth:** High demand in startups, enterprises, and cloud service providers.

Building Competencies: Full Stack Developers in DevOps

- A Full Stack Developer with DevOps competencies is not just a coder, but an architect of resilient, automated, and scalable digital solutions.
- By mastering both development and operations, such professionals drive digital transformation, faster innovation, and reliable software delivery.

Self-Organized Teams in DevOps

- 1. DevOps emphasizes collaboration, automation, and continuous delivery.
- At the heart of this culture are self-organized teams — groups empowered to make decisions, manage workflows, and deliver business value with minimal managerial intervention.

- 2. Key Characteristics of Self-Organized Teams
- Autonomy: Teams decide how to achieve goals instead of being micromanaged.
- Shared Responsibility: Development, testing, deployment, and operations are everyone's responsibility.
- Cross-Functionality: Each team has diverse skills — developers, testers, operations engineers, and sometimes business analysts — enabling end-to-end delivery.
- Continuous Learning: Teams embrace experimentation, feedback, and adaptation.
- Accountability: Members hold themselves accountable for outcomes, not just tasks.

Self-Organized Teams in DevOps

- 3. Benefits in a DevOps Environment
- Faster Delivery: Teams own the entire software lifecycle, reducing handoff delays.
- Improved Quality: Continuous integration, testing, and monitoring reduce defects.
- Higher Innovation: Teams can experiment and adapt processes/tools without waiting for approval.
- Employee Engagement: Autonomy increases motivation and ownership.
- Resilience: Teams can adapt quickly to failures, incidents, or changing business priorities.

Self-Organized Teams in DevOps

- 4. Practices that Enable Self-Organization
- Clear Goals & Vision: Teams must understand business objectives.
- Automation: CI/CD, Infrastructure as Code (IaC), and monitoring tools reduce manual overhead.
- Agile Practices: Scrum or Kanban help teams self-manage work.
- Feedback Loops: Daily standups, retrospectives, and monitoring dashboards support rapid learning.
- Trust & Empowerment: Leadership trusts teams to make decisions and gives them authority to act.

Self-Organized Teams in DevOps

- 5. Example Scenario
- A **self-organized DevOps team** working on an e-commerce platform:
- **Developers** push code into GitHub.

- **CI/CD pipelines** (Jenkins/GitHub Actions) automatically build, test, and deploy to a staging environment.

- **Operations engineers** monitor performance using Prometheus and Grafana.

- The **team collectively reviews incidents** and implements fixes without needing managerial approval.

- This approach shortens the release cycle, improves reliability, and boosts customer satisfaction.

Self-Organized Teams in DevOps

- In DevOps, self-organized teams are the backbone of continuous delivery and innovation.
- By blending cross-functional expertise, automation, and autonomy, these teams deliver high-quality software faster while remaining adaptable to dynamic business needs.

Traditional Teams vs. Self-Organized Teams in DevOps

Aspect	Traditional Teams	Self-Organized DevOps Teams
Decision Making	Manager-driven, top-down	Team-driven, decentralized
Work Distribution	Assigned by manager	Shared ownership and collaborative task selection
Skills	Specialized (e.g., only dev, only ops)	Cross-functional (dev, ops, QA, monitoring)
Responsibility	Individuals responsible for tasks	Team collectively responsible for outcomes
Adaptability	Slower to react to changes	Highly adaptive; continuous improvement
Innovation	Limited, dependent on approvals	High; empowered to experiment and iterate
Feedback Loops	Long cycles (monthly/quarterly reviews)	Short, continuous (standups, retrospectives, metrics)
Delivery Speed	Slower due to handoffs and silos	Faster with CI/CD and automation
Motivation & Ownership	Lower; task-focused mindset	Higher; goal-oriented and value-driven

Intrinsic Motivation in DevOps

- In a DevOps culture, people are as important as processes and tools.
- While automation and CI/CD pipelines streamline technical tasks, the real success of DevOps comes from the intrinsic motivation of team members — their inner drive to learn, collaborate, and deliver value beyond external rewards.
- 2. What is Intrinsic Motivation?
- Intrinsic motivation refers to the internal desire to perform a task because it is personally rewarding, not because of external pressures like salary, deadlines, or recognition.
- In DevOps, this means individuals take ownership of quality, reliability, and innovation because they care about the outcome.

Intrinsic Motivation in DevOps

- 3. Key Drivers of Intrinsic Motivation in DevOps
- Autonomy – The freedom to make decisions about tools, methods, and workflows.
- Mastery – The opportunity to continuously improve technical and problem-solving skills.
- Purpose – A clear understanding of how their work impacts the business, customers, and society.
- Collaboration – Working in cross-functional teams where knowledge-sharing and peer learning are valued.
- Trust & Empowerment – Teams feel trusted by leadership to experiment, take risks, and recover from failures.

Intrinsic Motivation in DevOps

- 4. Benefits of Intrinsic Motivation in DevOps
- Higher Quality Work: Motivated teams naturally pay attention to detail and proactively fix issues.
- Innovation: Passion for solving problems leads to creative approaches and new solutions.
- Resilience: Intrinsically motivated members see failures as learning opportunities.
- Employee Retention: Engaged individuals stay longer and contribute more deeply.
- Continuous Improvement: Teams drive adoption of new tools, practices, and automation without external enforcement.

Intrinsic Motivation in DevOps

- 5. Example Scenario
- In a healthcare DevOps team:
- Developers implement automated testing not just because management demands it, but because they want to ensure patient data safety and system reliability.

- Operations engineers proactively monitor system performance and fine-tune resources because they want to make sure the hospital can access life-saving data without delay.

- This sense of purpose and responsibility fuels continuous improvement.

- Intrinsic motivation is the heartbeat of DevOps culture. By fostering autonomy, mastery, and purpose, organizations can empower teams to deliver high-quality software faster, innovate continuously, and build sustainable, resilient systems.

Technology in DevOps

- 2. Infrastructure as Code (IaC)

- Definition: IaC is the practice of managing and provisioning infrastructure (servers, networks, storage) using code instead of manual processes.

- Tools: Terraform, Ansible, AWS CloudFormation, Puppet, Chef.

- Benefits:
  - Consistency across environments (Dev, QA, Prod).
  - Version-controlled infrastructure, enabling rollbacks.
  - Scalability through automation (auto-scaling clusters, VMs, containers).

- Example: Deploying a Kubernetes cluster automatically with Terraform scripts.

Technology in DevOps

- 3. Delivery Pipeline
- Definition: A delivery pipeline automates the software build, test, and deployment process, ensuring that code changes move smoothly from development to production.
- Stages:
  - Source Code Management (Git/GitHub/GitLab).
  - Continuous Integration (CI): Automated builds and unit testing (Jenkins, GitHub Actions, GitLab CI/CD).
  - Continuous Delivery (CD): Automated deployment to staging/production environments.
- Benefits:
  - Faster release cycles.
  - Early detection of defects.
  - Reduced manual intervention.
- Example: A Jenkins pipeline that builds a Docker image, runs unit/integration tests, and deploys to AWS EC2.

Technology in DevOps

- 4. Release Management
- **Definition:** Release Management in DevOps involves planning, scheduling, and controlling software delivery into production environments.
- Approach in DevOps:
  - **Continuous Release:** Frequent small updates rather than big-bang releases.
  - **Blue-Green Deployment:** Two environments (blue and green) are used to minimize downtime.
  - **Canary Releases:** Gradually releasing updates to a subset of users before full rollout.
- Benefits:
  - Minimized risks during deployment.
  - Improved stability with automated rollbacks.
  - Better alignment with business needs.
- **Example:** Using Kubernetes + Helm charts for rolling updates with zero downtime.

Technology in DevOps

- Technologies like IaC, Delivery Pipelines, and Automated Release Management form the backbone of DevOps.

- They ensure reliable, repeatable, and scalable deployments, enabling organizations to deliver high-quality software faster and with greater confidence.

Tools and Technologies as Enablers for DevOps

- DevOps is built on a culture of collaboration, automation, and continuous improvement.
- However, the real enablers of DevOps success are the tools and technologies that make it possible to achieve speed, reliability, and scalability.
- These tools span the entire software development lifecycle (SDLC), from code integration to monitoring in production.

Key Categories of DevOps Tools

- a) Version Control
- Purpose: Manage code, track changes, enable collaboration.
- Examples: Git, GitHub, GitLab, Bitbucket.
- Enabler Role: Facilitates team collaboration and code traceability.

Key Categories of DevOps Tools

- b) Continuous Integration & Continuous Delivery (CI/CD)
- Purpose: Automate build, test, and deployment pipelines.
- Examples: Jenkins, GitLab CI/CD, GitHub Actions, CircleCI, Bamboo.
- Enabler Role: Ensures faster, error-free deployments with minimal manual intervention.

Key Categories of DevOps Tools

- c) Infrastructure as Code (IaC) & Configuration Management
- Purpose: Automate provisioning and configuration of infrastructure.
- Examples: Terraform, Ansible, Puppet, Chef, AWS CloudFormation.
- Enabler Role: Provides consistency, scalability, and version-controlled infrastructure.

Key Categories of DevOps Tools

- d) Containerization & Orchestration
- Purpose: Package applications with dependencies and run them anywhere.
- Examples: Docker, Kubernetes, OpenShift, Docker Compose.
- Enabler Role: Supports microservices, portability, and high availability.

Key Categories of DevOps Tools

- e) Monitoring & Logging
- Purpose: Ensure system reliability and observability.
- Examples: Prometheus, Grafana, ELK Stack (Elasticsearch, Logstash, Kibana), Splunk.
- Enabler Role: Enables proactive issue detection, performance optimization, and feedback loops.

Key Categories of DevOps Tools

- f) Collaboration & Communication
- Purpose: Align development, operations, and business teams.
- Examples: Slack, Microsoft Teams, Jira, Confluence, Trello.
- Enabler Role: Promotes transparency, agility, and effective communication.

Key Categories of DevOps Tools

- 3. Benefits of Tools as DevOps Enablers
- Automation: Eliminates repetitive manual tasks.
- Consistency: Ensures uniform environments across Dev, QA, and Prod.
- Scalability: Rapidly scale applications in cloud or hybrid environments.
- Reliability: Continuous monitoring improves uptime and user experience.
- Faster Delivery: Shorter release cycles enable quicker time-to-market.

Key Categories of DevOps Tools

- 4. Example Scenario
- A fintech company uses:
- GitHub for source control,
- Jenkins for CI/CD,
- Docker & Kubernetes for microservices deployment,
- Terraform for infrastructure automation,
- Prometheus & Grafana for monitoring.
- This integrated toolchain enables them to release updates multiple times a day, reduce downtime, and ensure compliance with financial regulations.

- DevOps tools are not just add-ons; they are the backbone of the DevOps ecosystem.

- When strategically chosen and integrated, they empower organizations to achieve continuous delivery, agility, and business value.

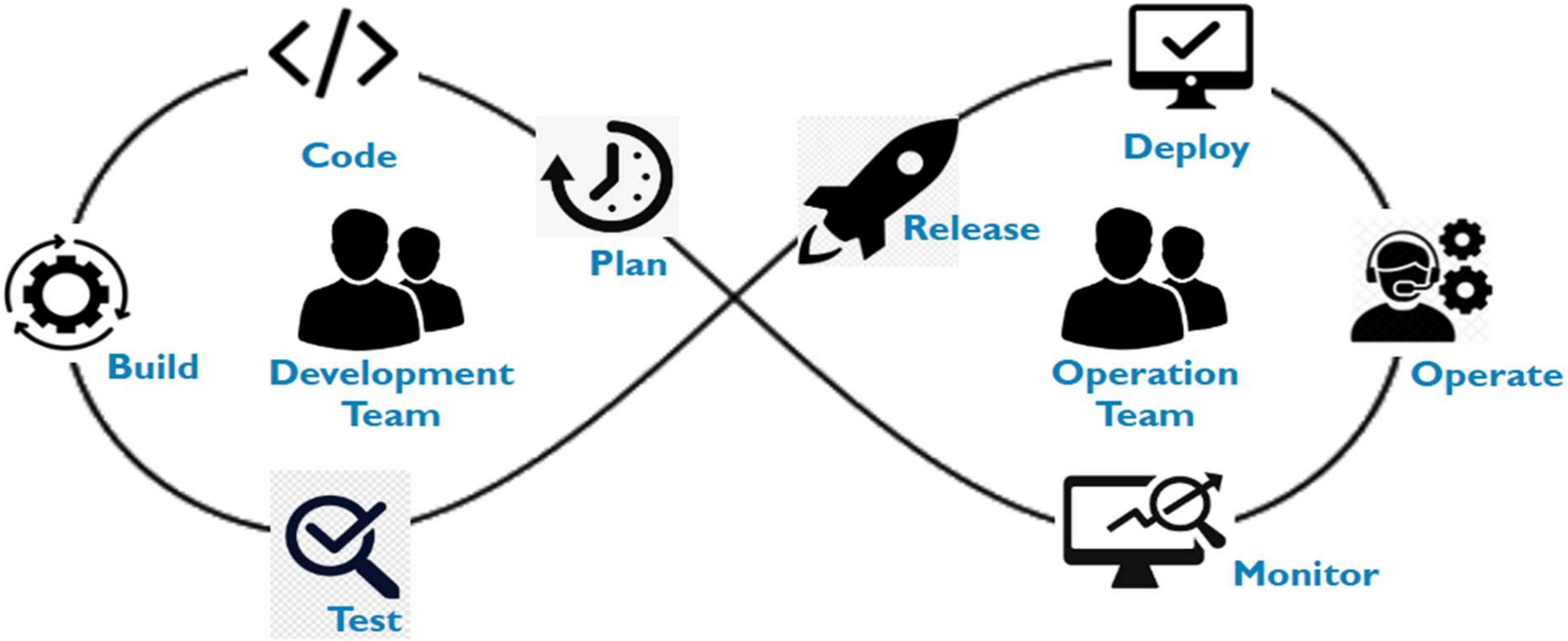
Development with DevOps

Streamlined Deliveries

Team Work in Collaboration

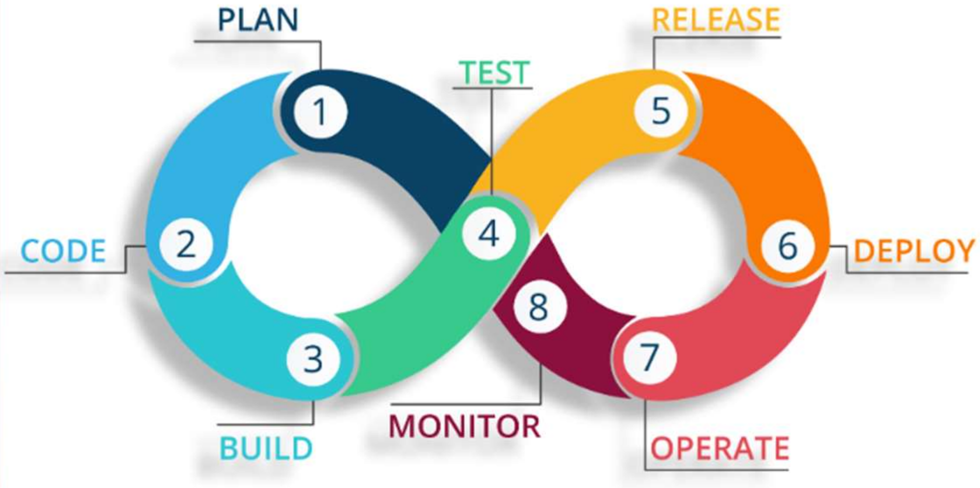
Continuous Monitoring and Feedback

DevOps in Real Life



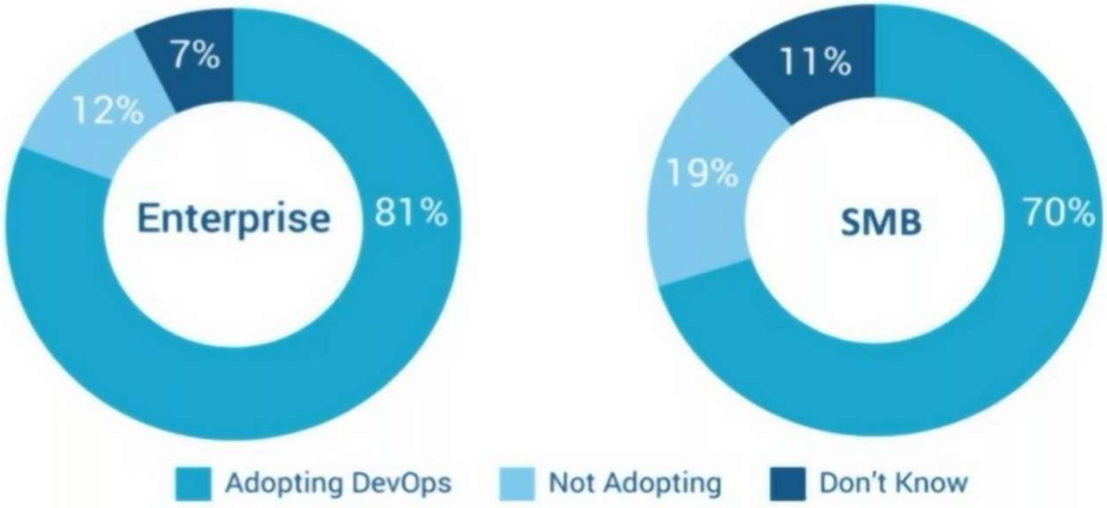
DevOps in Real Life

- Integrates developers and operations teams
- Improves collaboration and productivity by:
  - ✓ Automating infrastructure
  - ✓ Automating workflows
  - ✓ Continuously measuring application performance



DevOps Current Scenario

“Considering the changing pace of IT landscape, almost all the companies require fast paced development environment”



DevOps Lifecycle - Plan

"First stage of DevOps cycle, where you Plan, Track, Visualize and Summarize your Project before working/starting it."

1
2
3
4
5
6
7
PLAN

Planning Tools

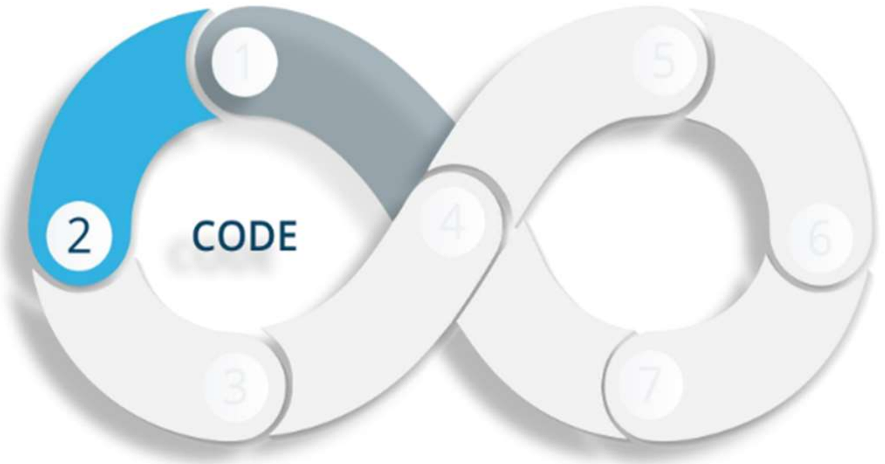
JIRA

Trello

Tricentis

DevOps Lifecycle - Code

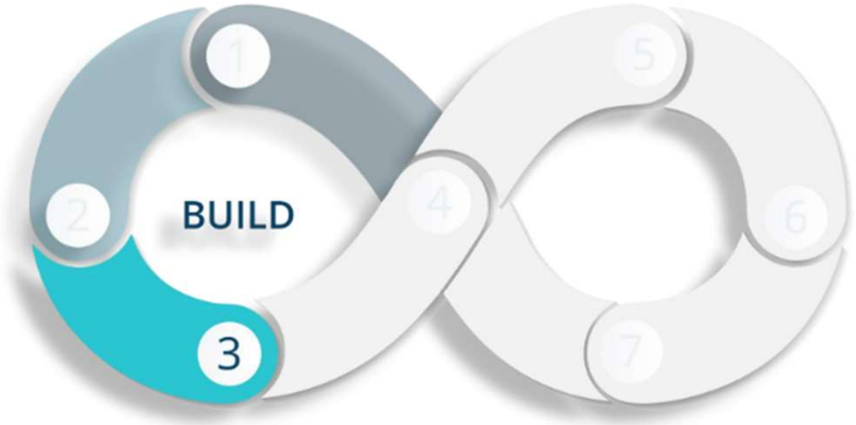
"Second stage of DevOps cycle, where the developers write their code"



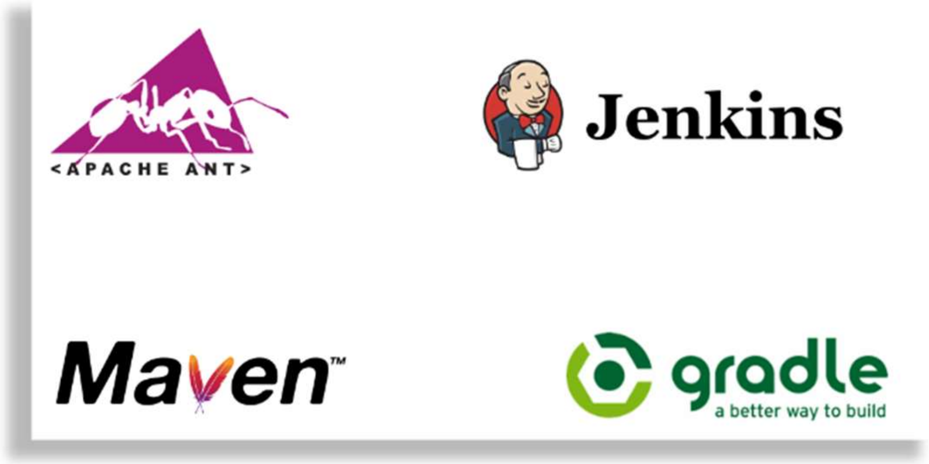
Coding Tools
eclipse
git
SUBVERSION

DevOps Lifecycle - Build

“Build is a pre-release version and is identified by a build number, rather than by a release number”

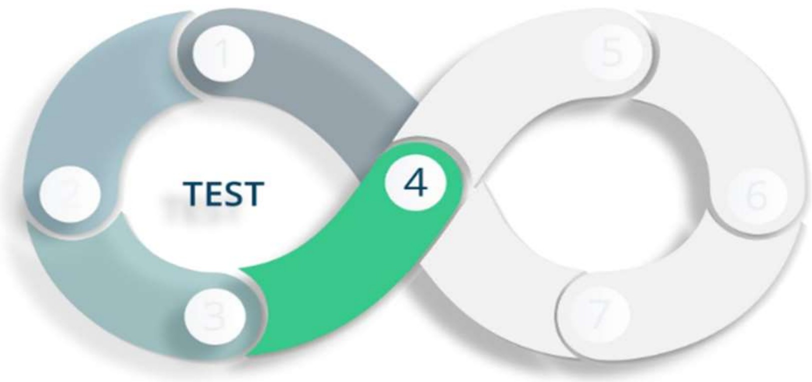


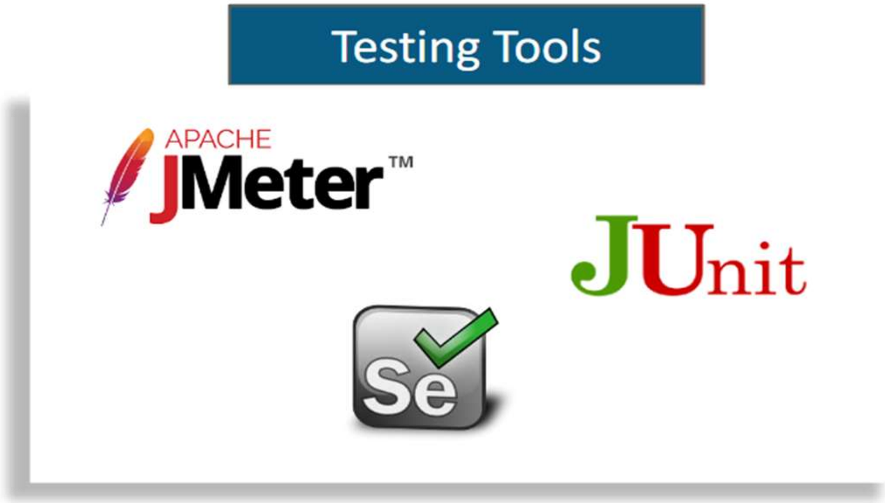
Building Tools



DevOps Lifecycle - Test

“Process of executing automated tests as part of the software delivery pipeline in order to obtain feedback on the business risks associated with a software release as rapidly as possible”

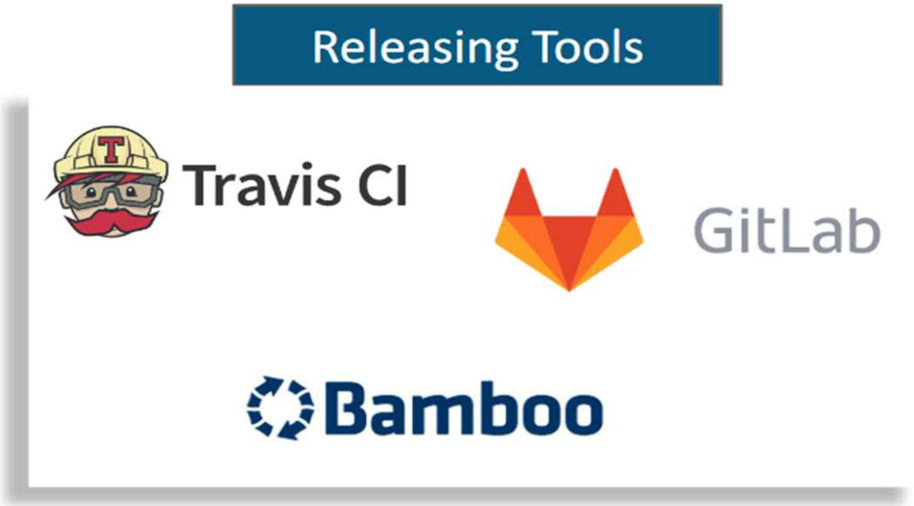




DevOps Lifecycle - Release

"This phase helps to integrate code into a shared repository using which, you can detect and locate errors quickly and easily"

1
2
3
4
5
RELEASE
6
7



DevOps Lifecycle - Deploy

"Manage and maintain development and deployment of software systems and servers in any computational environment"

1
2
3
4
5
DEPLOY
6
7

Deploying Tools

aws
ANSIBLE
CHEF

DevOps Lifecycle - Operate

"This phase is to keep the system upgraded with the latest update"

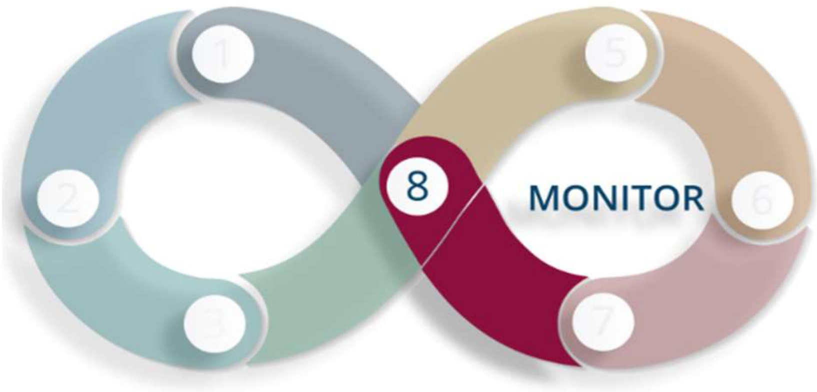
1
2
3
4
5
6
7
OPERATE

Operating Tools

aws
ANSIBLE
CHEF

DevOps Lifecycle - Monitor

"It ensures that the application is performing as desired and the environment is stable. It quickly determines when a service is unavailable and understand the underlying causes"



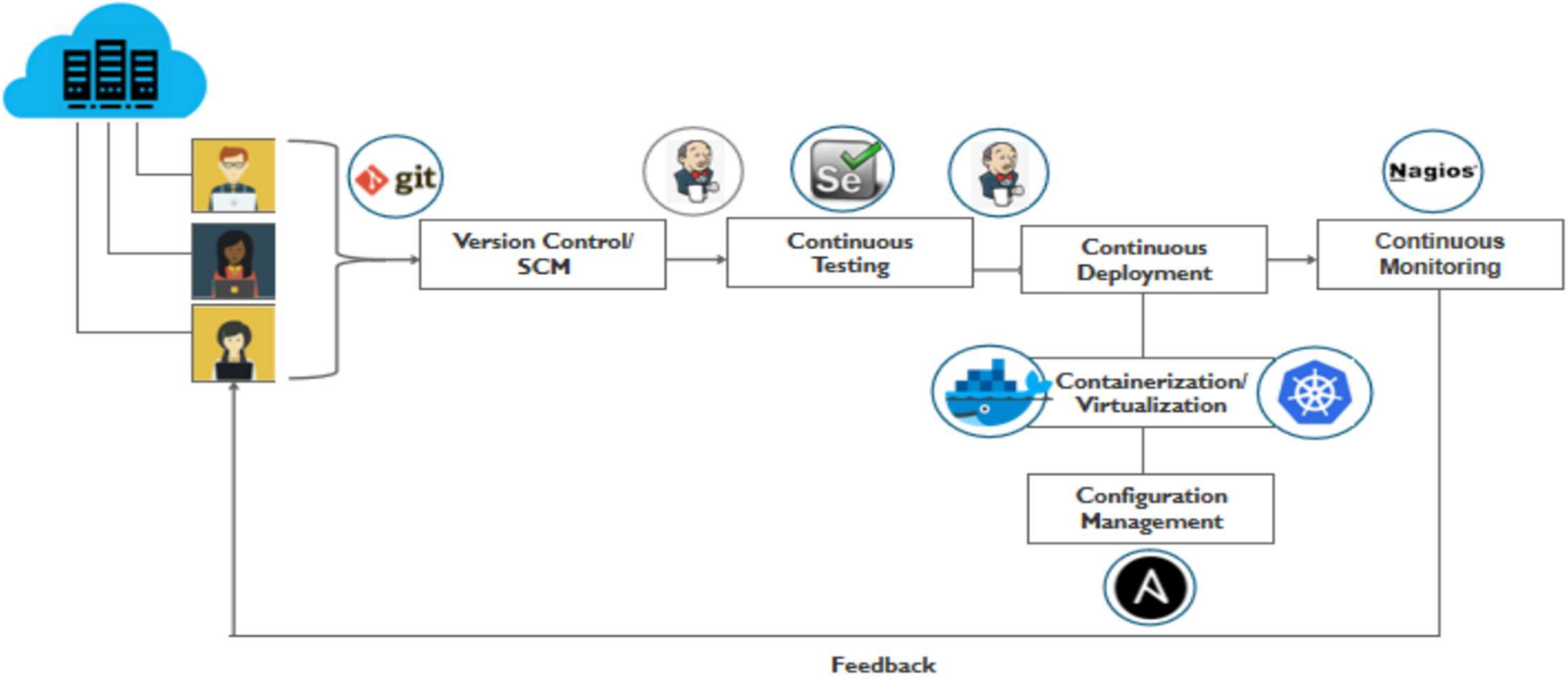
Monitoring Tools

Nagios®

sensu

splunk>

DevOps Stages



DevOps Stages

Continuous Integration (CI):

- ✓Continuous integration is a DevOps software development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests are run.
- ✓ The key goals of continuous integration are to find and address bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates.

Continuous Delivery (CD):

- ✓CD ensures that code is always in a state ready for deployment, streamlining release processes and reducing risks.
- ✓With continuous delivery, code changes are automatically built, tested, and prepared for a release to production.
- ✓Continuous delivery expands upon continuous integration by deploying all code changes to a testing environment and/or a production environment after the build stage.

Continuous Deployment (CD):

- ✓CD automates the release to production, increasing deployment frequency while maintaining high standards of reliability.

QUESTIONS AND DISCUSSION

THANK YOU