V - Velocity Conference
D - Developers vs. Ops mismatch
L - Lean and Agile principles
C - Continuous Delivery (Jez Humble)
P - Phoenix Project (Gene Kim)
A - Adoption by enterprises
Use: "Very Dedicated Learners Can Practice Always".

Origins of DevOps

DevOps originated to solve the challenges of software development and operations working separately, which caused delays and inefficiencies. It focuses on improving collaboration, communication, and automation between teams. The main events in its origin are:

1. **Velocity Conference 2009**
   DevOps began with a talk titled *"10+ Deploys Per Day: Dev and Ops Cooperation at Flickr"* by John Allspaw and Paul Hammond. This showed how developers and operations could work together to achieve faster deployments. Later, Patrick Debois introduced the term "DevOps" during the first DevOpsDays in Belgium.
2. **The Problem of Dev and Ops**
   Developers aimed to release features quickly, while operations focused on stability. This mismatch created delays. DevOps solved this by encouraging teamwork and automating repetitive tasks.
3. **Lean and Agile Roots**
   - DevOps borrowed ideas from Lean manufacturing, which focused on reducing waste and improving efficiency.
   - Agile software development (2001) introduced iterative work and faster delivery. Agile's speed created the need for operations to adapt, which led to DevOps.
4. **Key Books and Thought Leaders**
   - Jez Humble's *"Continuous Delivery"* (2010) explained practices like automation and frequent delivery.
   - Gene Kim's *"The Phoenix Project"* (2013) showed how DevOps could transform IT in enterprises.
5. **Adoption by Enterprises**
   Initially used by startups, DevOps gained popularity in enterprises by 2012. Companies like IBM and ThoughtWorks helped large organizations adopt DevOps, while tools like Puppet and Chef enabled automation.

Roots of DevOps

DevOps is based on ideas from Lean manufacturing and Agile software development, created to solve inefficiencies in software delivery.

1. **Lean Manufacturing**
   - Lean started with Henry Ford's assembly lines in the 1910s to improve efficiency and reduce waste.
   - Toyota refined Lean in the 1930s with continuous improvement techniques.
   - Dr. W. Edwards Deming's *Plan-Do-Check-Act (PDCA)* cycle in the 1950s emphasized improving processes step-by-step.
2. **Agile Software Development**
   - Agile was introduced in 2001 to make software development faster and more flexible.
   - It focused on iterative development, quick customer feedback, and delivering small, working pieces of software.
3. **The Problem of Bottlenecks**

    - Agile's speed created delays because operations teams were too slow to test and deploy changes.
    - DevOps emerged to bring development and operations teams together, with shared goals and automation to remove these bottlenecks.
4. **Combining Lean and Agile**
    - DevOps uses Lean's ideas to reduce waste and Agile's iterative process to speed up software delivery.
    - It extends Agile principles by involving operations teams in the delivery process.

Rapid Development and the Need for Collaboration Between Dev and Ops

Agile development brought faster iterations and frequent code releases, but this created challenges for operations teams.

1. **Short Iterations in Agile**
    - Agile focuses on delivering small updates quickly, allowing constant feedback and improvements.
    - However, this speed put pressure on operations teams to test and deploy code faster.
2. **Challenges for Operations**
    - Operations teams were used to slower, planned releases, leading to delays in providing test environments or deploying updates.
    - These delays created bottlenecks and slowed down the entire process.
3. **Frequent Failures**
    - Developers lacked access to production-like environments, causing issues to appear only during deployment.
    - This lack of coordination led to frequent failures when releasing updates.
4. **DevOps as a Solution**
    - DevOps introduced better collaboration between development and operations.
    - Shared responsibility and automation tools (e.g., automated testing and environment setup) reduced delays and made delivery faster and smoother.

Conclusion
Agile's rapid development exposed the need for better coordination between Dev and Ops. DevOps bridges this gap by encouraging teamwork and using automation to streamline the process.

Dev vs. Ops: The Conflict and Solution

Development (Dev) and Operations (Ops) have historically worked separately, with different goals, causing delays and inefficiencies.

1. The Conflict

- **Development (Dev):**

    - Focuses on building new features quickly.
    - Prioritizes speed and innovation.
- **Operations (Ops):**
    - Focuses on maintaining system stability and uptime.
    - Prioritizes reliability and minimizing risks.
- **Opposing Goals:**
    - Dev wants rapid changes, while Ops wants to ensure stability.
    - Frequent updates from Dev often create risks for Ops.

2. Challenges

- **For Developers:**
    - Lack of understanding of production environments causes issues.
    - Deployment delays due to poor coordination with Ops.
- **For Operations:**
    - Frequent updates strain their ability to maintain stability.
    - Manual processes slow down deployments.

3. How DevOps Solves It

- **Collaboration:**
    - Dev and Ops work together throughout the delivery process.
- **Early Involvement (Shift Left):**
    - Ops is involved early in development to ensure compatibility.
- **Automation:**
    - Automated tools for deployment and testing reduce delays and errors.
- **Shared Goals:**
    - Both teams align on metrics like speed and system stability.

IBM Practices of DevOps (Simplified)

1. **Think**: Plan and set clear goals for application delivery.
2. **Code**: Write high-quality and secure code.
3. **Deliver**: Deploy software quickly and reliably.
4. **Run**: Ensure applications run smoothly after deployment.
5. **Manage**: Track and optimize resources and processes.
6. **Learn**: Use feedback to improve continuously.
7. **Culture**: Promote teamwork, trust, and shared responsibility.

T think
C code
D deliver
R run
M manage
L learn
C culture

Two Key Capabilities of DevOps (Simplified)

1. **Continuous Integration**:
    - Combine code changes frequently.
    - Use automated tests to find and fix issues early.
    - Helps teams work together and reduces risks.
2. **Continuous Delivery**:
    - Automate the delivery of tested code to environments ready for deployment.

    - Keeps software always ready for release.
    - Speeds up delivery and improves quality

Continuous Integration (CI)

- **Definition**: A practice where developers regularly integrate their code into a shared repository, ideally multiple times a day. Each integration triggers an automated build and testing process to detect integration issues early.
- **Purpose**: To identify and resolve defects as soon as possible, reducing the risk of major issues later in the development cycle.
- **Benefits**:
    - Early discovery of integration risks and bugs.
    - Improved collaboration among teams.
    - Faster feedback on code changes.

CI also supports Agile practices by ensuring that all components of a system (developed by different teams) work together seamlessly, enabling a smoother development process.

Continuous Delivery (CD)

- **Definition**: Extends CI by automating the delivery of integrated code to production-like environments for testing and verification. The goal is to ensure that the software is always in a deployable state.
- **Purpose**: To streamline the release process and make it predictable, allowing software to be deployed to production quickly and with minimal risk.
- **Benefits**:
    - Faster and more reliable deployments.
    - Reduced chances of errors during release.
    - Higher quality software delivered to customers sooner.

CD ensures that each change made to the codebase passes all stages of testing and is ready for production, allowing teams to release features and updates more frequently

Practices of Continuous Integration

1. **Use a Single Repository**
    - Keep all code in one place so everyone works from the same version.
2. **Automate Builds**
    - Automatically compile code to avoid manual errors.
3. **Self-Testing Builds**
    - Run tests during builds to ensure the system works properly.
4. **Commit Daily**
    - Developers should upload their work every day to catch problems early.
5. **Test Each Commit**

    - Every code change should trigger automated tests to find errors fast.
6. **Keep Builds Fast**
    - Make sure builds complete quickly to avoid slowing down development.
7. **Test in Production-Like Environments**
    - Test in environments similar to the real-world system to minimize surprises.
8. **Access Latest Build**
    - Make the newest version of the software easily available to everyone.
9. **Show Build Status**
    - Use tools to display the progress and results of builds for the team.
10. **Automate Deployment**

- Automatically move code to testing or staging environments to prepare for release

DevOps Culture

DevOps is primarily a **cultural movement** aimed at fostering trust, communication, and collaboration among teams involved in software delivery. It goes beyond processes and tools, requiring changes in how people work together and share responsibilities. Success in DevOps depends on breaking down silos, encouraging teamwork, and promoting a shared ownership of the application delivery process.

Three Areas of Cultural Inertia and How to Overcome Them

1. **Visibility**
    - **Challenge**: A lack of transparency between teams leads to mistrust and inefficiency. Teams don't fully understand what others have done with the artifacts being handed off.
    - **Solution**: Implement tools and dashboards that provide real-time visibility into workflows, ensuring all teams can see progress and dependencies.
2. **Effective Communication**
    - **Challenge**: Teams often rely on outdated methods like email or ticket-based communication, which can cause delays and misunderstandings.
    - **Solution**: Shift to real-time, peer-to-peer communication using tools like Slack or Yammer. Encourage live, direct discussions to foster collaboration and eliminate unnecessary bureaucratic steps.
3. **Common Measurements**
    - **Challenge**: Different teams (Dev, Ops, Test) often use varying metrics to measure success, which can lead to conflicting goals and hinder collaboration.
    - **Solution**: Establish shared metrics that align with the organization's overall goals. Ensure everyone is accountable for delivering business value, fostering a sense of shared responsibility