Welcome!

Backend Engineering

October 26, 2023

by

Prachi Shah

Software Engineer,
Engineering Leader & Mentor,
Community Volunteer.

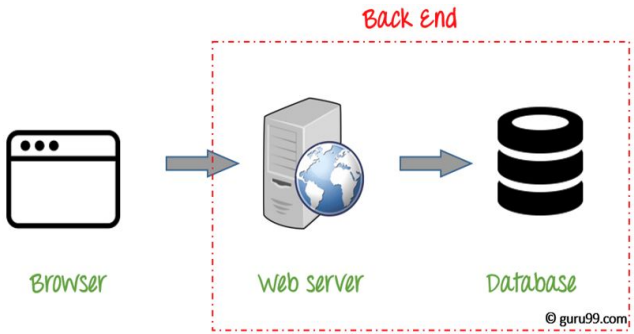
Topic: DevOps 101

- What is DevOps?
- CI/CD.
- Orchestration.
- Logging and Monitoring.
- Tools and tech stack.

Backend Engineering

- Design, build and maintain server-side web applications

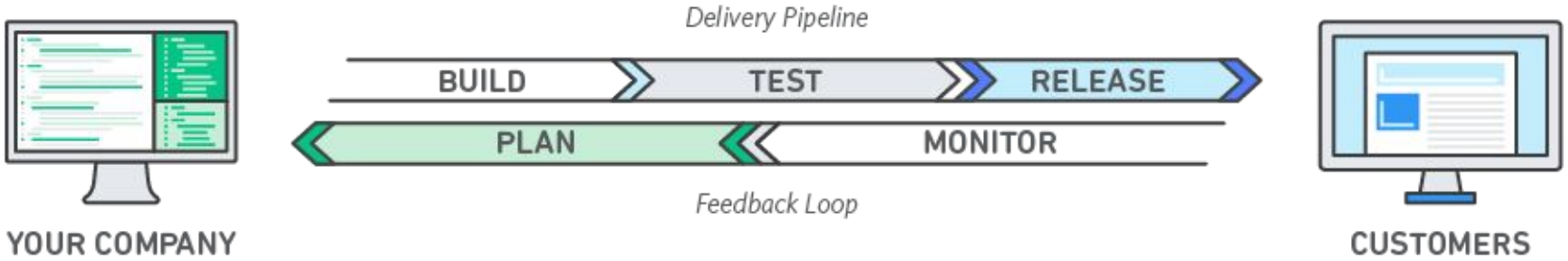
- Concepts: Client-server architecture, networking, APIs, web fundamentals, microservices, databases, security, operating systems, etc.



- Tech Stack: Java, PHP, .NET, C#, Ruby, Python, REST, AWS, Node, SQL, NoSQL, etc.

Developer Operations

• DevOps merges development and operations teams to shorten the system development lifecycle and provide high-quality software.
• Before DevOps, developers would write code and operations would deploy it. If an issue arose, blame games would start. DevOps seeks to end that by fostering collaboration.
• **Rapid Delivery:** Launch new features and rectify bugs at speed. Cater to your customers' needs and establish a competitive edge
• **Improved Collaboration**: Fosters ownership and accountability between developers and operations teams.
• **Security:** Ensures security of system and supports compliance needs.



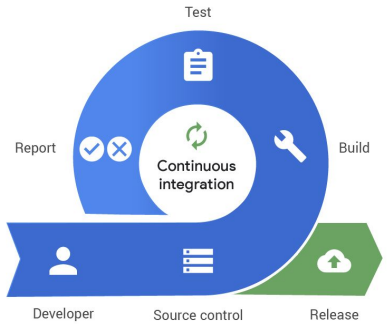
Continuous Integration

Continuous Integration (CI) is a software development practice where developers merge their code changes into a central repository. This is followed by automated testing and building, helping teams to identify and fix errors more quickly.

Principles:
- Frequent Code Commitment: Frequent merges into a central repository.
- Automated Testing: Automatic tests run after each code merge.
- Immediate Feedback: Developers are instantly informed about issues in their code.
- Build Automation: Automatic compilation of code into executable artifacts.
- Consistency: CI ensures the codebase remains stable.

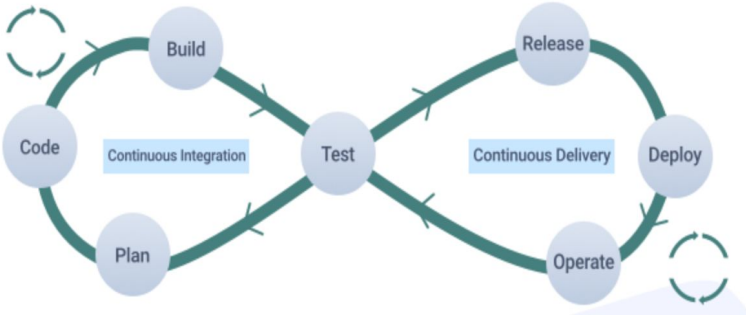
Tools:
- Version Control System: E.g., Git to track code changes.
- Build Server: E.g., Jenkins for automated builds and tests.

Examples:
- Open-Source Projects: Global collaboration and code integration.
- E-commerce Platforms: Consistent, functional, and secure updates.
- Mobile Apps: Smooth integration of new features without introducing bugs.



Workflow

- Develop: Code on local machines.
- Commit: Changes committed to the repository.
- Build CI: Automated processes run.
- Report: Feedback given to developers.
- Fix: Required changes are made and re-committed.



Benefits:

- Rapid Feedback: Issues are identified and fixed quickly.
- Reduced Bugs: Fewer bugs reach production due to automated tests.
- Enhanced Collaboration: Frequent code integration reduces issues and fosters collaboration.
- Time Efficiency: Faster development cycles, reduced time to market.

Challenges:

- Initial Setup: Complexity in setting up CI.
- Maintaining Test Suite: Ensuring tests remain up-to-date and valuable.
- Cultural Shift: Adapting to a CI mentality across the organization.

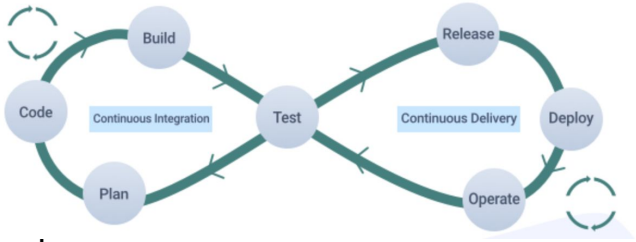
Continuous Delivery

Continuous Delivery (CD) is an approach where software is developed, tested, and released in short cycles to ensure that it can be reliably deployed at any time. The methodology aims for quicker, more frequent releases with a focus on automation and collaboration.

Principles:
- Automate Everything: Minimize manual errors by automating build, test, and release processes.
- Collaborate Across Teams: Foster inter-team collaboration to break down operational silos.
- Maintain a Repository: Manage code versions effectively.
- Stable Testing Environment: Reliably test every change.
- Consistency and Repeatability: Ensure a uniform, sustainable model for software releases.

Components:
- Version Control: Manage and track code changes.
- Continuous Integration: Frequent integration of code changes.
- Automated Testing: Ensure code quality through automated tests.
- Automated Deployments: Codebase should be ready for deployment at any time.
- Infrastructure as Code (IaC): Ensure technological consistency and repeatability.

Pipeline



- Build: Compile and package code.
- Test: Run automated tests for quality assurance.
- Deploy: Automatically deploy the application for further testing.
- Release: Manually trigger production deployment, ensuring control.

Benefits:

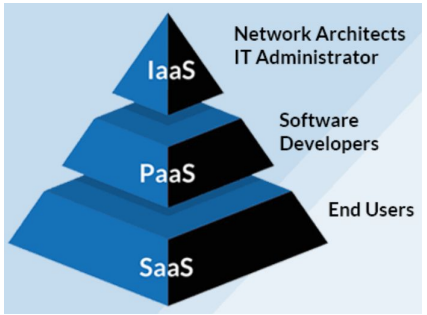
- Rapid Release: Faster, more reliable software rollouts.
- Lowered Risk: Small, incremental changes reduce bug and vulnerability risks.
- Higher Quality: Automated testing improves release quality.
- Enhanced Productivity: Automation speeds up the release process.
- Improved Customer Satisfaction: Quicker feature releases and bug fixes.

Challenges:

- Implementation Cost: Initial cost for tools and setup.
- Cultural Shift: Requires a change in team mentality towards collaboration.
- Management Complexity: Especially challenging in microservices architectures.
- Skill Requirement: Need for professionals well-versed in CD principles and tools.

Cloud Computing

Cloud Computing is the delivery of computing services, such as storage, databases, and software, over the Internet. It revolutionizes the way we access and manage resources.



Characteristics:
- On-Demand Self-Service: Automated access without human intervention.
- Broad Network Access: Accessible via standard network protocols.
- Resource Pooling: Dynamic allocation based on demand.
- Rapid Elasticity: Quick and automated scaling of resources.
- Measured Service: Automatic resource optimization.

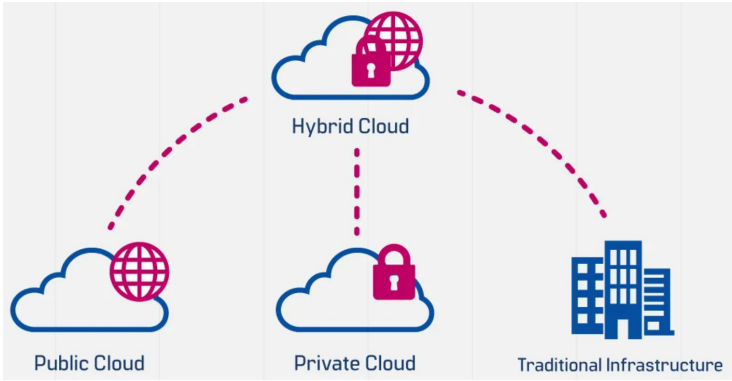
Service Models:
- Infrastructure as a Service (IaaS): Virtual computing resources. E.g., AWS, Google Cloud.
- Platform as a Service (PaaS): Platform for app development. E.g., Heroku, Google App Engine.
- Software as a Service (SaaS): Software accessed over the Internet. E.g., Gmail, Zoom.

Models

Deployment Models:
- Public Cloud: Open to anyone, services offered over the public Internet.
- Private Cloud: Exclusive to a single organization.
- Hybrid Cloud: Mix of public and private clouds.

Advantages:
- Cost-Efficient: Pay-as-you-go model, reducing capital expenditure.
- Scalability: Flexibility to scale resources.
- Data Backup: Reliable data backup and restoration.
- Automatic Updates: Managed by the provider.
- Remote Access: Accessible anytime, anywhere.

Challenges:
- Security: Concerns about data protection.
- Compliance: Adherence to data protection laws.
- Downtime: Possible service interruptions.
- Limited Control: Over data, services, and functionalities.



Infrastructure as a Code

Infrastructure as Code (IaC) is a pivotal DevOps practice that facilitates the management and provisioning of computing resources via machine-readable scripts, rather than manual configurations.

Concepts:

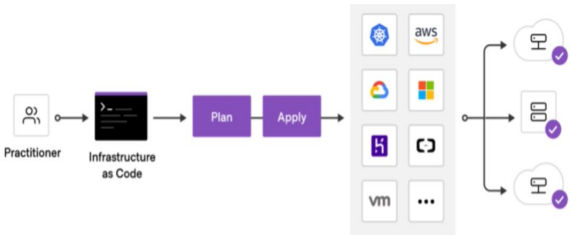
- Code-Based Management: Uses code for configuring and automating infrastructure.
- Automation: Streamlines setup, monitoring, and infrastructure management.
- Idempotency: Ensures repeatable results with the same configuration.
- Policy as Code: Manages policies and compliance via code.

Components:

- Configuration Files: Define desired states and settings.
- Automated Tools: Examples include Terraform, AWS CloudFormation, and Ansible.
- Base Images: Utilize pre-configured containers or VM images.

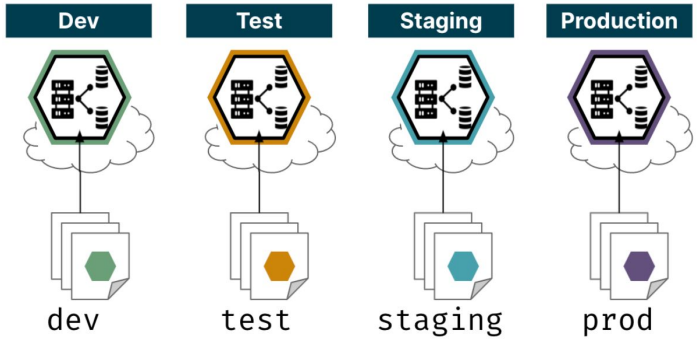
Principles:

- Source Code Control: Store IaC configurations in systems like Git.
- Versioning: Maintain different versions for rollback and reference.
- Modularity: Code should be reusable and modular.
- Continuous Integration: Automatically test and validate changes.



Advantages

- Consistency: Standardized, repeatable provisioning process.
- Scalability: Scale infrastructure effortlessly.
- Accountability: Track changes and audit easily.
- Efficiency: Minimizes manual effort in deployments.



Implementation Steps:

- Define: Specify infrastructure requirements in code.
- Version Control: Store files in a version control system.
- Automate: Use automated tools for deployment.
- Monitor: Keep track of infrastructure.
- Maintain: Update IaC according to organizational needs.

Tools:

- Terraform: Open-source IaC for data center infrastructure.
- Ansible: IT automation tool for system state.
- AWS CloudFormation: Declarative AWS infrastructure modeling.

Use Cases: Environment Setup, Auto-Scalability, Disaster Recovery.

Orchestration

Software Orchestration is the automated, coordinated management of tasks to optimize and streamline IT processes and workflows, primarily in software development and deployment.

Principles:
• Optimization and Streamlining: Aims to minimize manual intervention for efficiency and reduced errors.
• Automated Workflows: Uses tools to automate tasks such as code deployment, database management, and network configurations.
• Managed Execution: Ensures tasks follow a specific order, comply with policies, and manage dependencies.

Applications:
• Deployment Orchestration: Automates steps from code repository to production.
• Container Orchestration: Manages container lifecycles, especially in Docker.
• Workflow Orchestration: Automates and optimizes task sequences and workflows.
• Cloud Orchestration: Manages and automates cloud resources and services.

Tools: Kubernetes, Ansible, Terraform, Apache Airflow.

Monitoring

Software Application Monitoring (SAM) is the practice of observing the performance, health, and user experience of software applications to ensure optimal functionality and enhance user satisfaction.

Principles:

- Performance Monitoring: Focus on meeting or exceeding application performance benchmarks.
- Error Tracking: Identifying and logging application errors for debugging.
- User Experience: Monitoring user interactions to ensure a positive experience.
- Resource Utilization: Tracking resource usage for optimal allocation.

Components:

- Metrics Collection: Capture KPIs like response time, error rate, and throughput.
- Alerts: Configure and manage alerts for stakeholders.
- Data Visualization: Use dashboards and reports to visualize data.
- Log Management: Analyze and manage log files for insights.

Approaches:

- Real User Monitoring (RUM): Monitors actual user interactions.
- Synthetic Monitoring: Simulates user interactions via bots.
- Application Performance Monitoring (APM): Offers in-depth component monitoring.

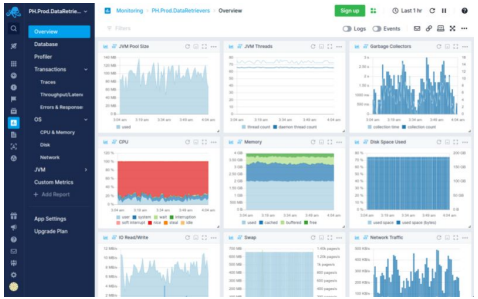
Benefits

- Enhanced User Satisfaction: A focus on a seamless user experience.
- Proactive Issue Resolution: Catch and fix issues before they affect users.
- Optimized Performance: Meet or exceed performance benchmarks.
- Data-Driven Decisions: Use monitoring data for informed planning.

Challenges:
- Complexity: Difficulties in managing and interpreting extensive data.
- Tool Management: Resource-intensive tool selection and implementation.
- Alert Fatigue: Need to prioritize alerts effectively.

Applications:
- E-Commerce Platforms: Focus on seamless transactions and high availability.
- Mobile Applications: Optimize performance across devices and networks.
- Cloud-Based Services: Manage resource usage and service availability.

Tools: Datadog, New Relic, AppDynamics, SolarWinds.

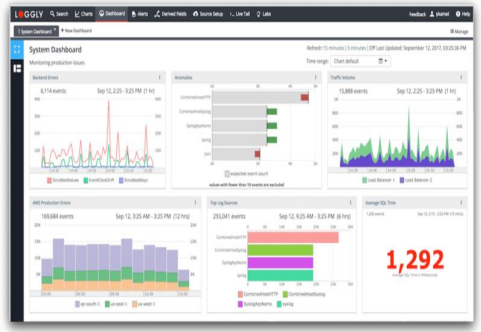


Logging

Software Application Logging is the practice of recording messages, events, and interactions during a software application runtime. Logs serve multiple purposes, including troubleshooting, auditing, and performance monitoring.

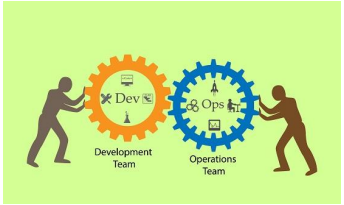
Aspects:
- Event Logging: Records specific events like user authentication and system errors.
- Debug Logging: Captures detailed info for troubleshooting issues and bugs.
- Audit Logging: Documents actions for compliance and security purposes.
- Performance Logging: Monitors system performance metrics such as response times.

Importance:
- Error Detection: Facilitates issue identification and resolution.
- Security: Helps detect security breaches or vulnerabilities.
- Compliance: Aids in maintaining regulatory compliance.
- Performance Analysis: Analyzes system performance and user behavior.



Tools

- Loggly: Cloud-based, facilitates centralized log management.
- Splunk: Provides extensive log aggregation, analysis, and visualization.
- ELK Stack: Open-source platform for logging, aggregating, and visualizing data.
- Graylog: Free, open-source tool for storing, searching, and analyzing logs.



Applications:
- Troubleshooting: For identifying and fixing software issues.
- Audit Trail: Maintains a record for audit purposes.
- Business Intelligence: Helps derive insights from user behavior and system usage.
- Security Monitoring: Tracks security events and potential vulnerabilities.

Challenges:
- Data Overload: Effective management of large volumes of log data.
- Storage Management: Efficient log storage is a challenge due to large data volumes.
- Log Security: Prevent unauthorized access and manipulation of log data.

Tech Stack

- Programing Languages: Bash, Shell scripting, Java, Ruby
- Monitoring: Splunk, StatsD, New Relic, SignalFx
- Automation: Chef, Kickstart, Puppet, Ansible
- Databases: MySQL, Oracle, BigQuery
- Application Servers: Apache, Httpd, Tomcat, nginx
- Cloud Providers: AWS, Google Cloud, Microsoft Azure
- Real Time analytics tools: Apache Storm, Spark, Shark
- Queues: RabbitMQ, Kafka, Kinesis, Redis
- Node Discovery: Zookeeper, Etcd
- Container Management: Docker, Kubernetes, OpenShift
- Build and CI/CD: Jenkins, GitLabs, JFrog, Maven, Gradle
- Version Control: git, GitHub, GitLab, Bitbucket
- Testing: Selenium, JUnit, TestNG
- Application Security: Okta, auth0, PagerDuty

Tech Stack