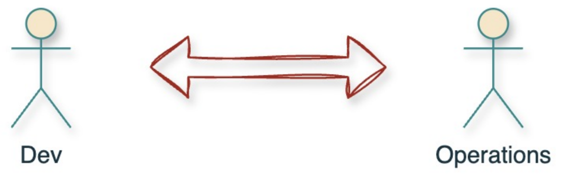
DevOps Fundamentals

On this page

What is DevOps?



- **Scenario**: Developers build features - they want to see new features in production. Operations want production to stable - very few changes. Developers and Operations work in silos => (delays, finger-pointing, no team spirit). Can we build and run software together?
- **DevOps**: A culture and set of practices that bring **Development** and **Operations** teams together to deliver software **faster**, **safer**, and **continuously**.

No Strict Definition

- **Amazon Web Services(AWS)**: "DevOps is the combination of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity"
- **The DevOps Handbook - Gene Kim , Patrick Debois, Et al.**: "DevOps is the outcome of applying the most trusted principles from the domain of physical manufacturing and leadership to the IT value stream. DevOps relies on bodies of knowledge from Lean, Theory of Constraints, the Toyota Production System, resilience engineering, learning organizations, safety culture, human factors, and many others. The result is world-class reliability, stability, and security at ever lower cost and effort; and accelerated flow and reliability through the technology value stream, including Product Management, Development, QA, IT Operations, and Infosec."

DevOps is about Collaboration

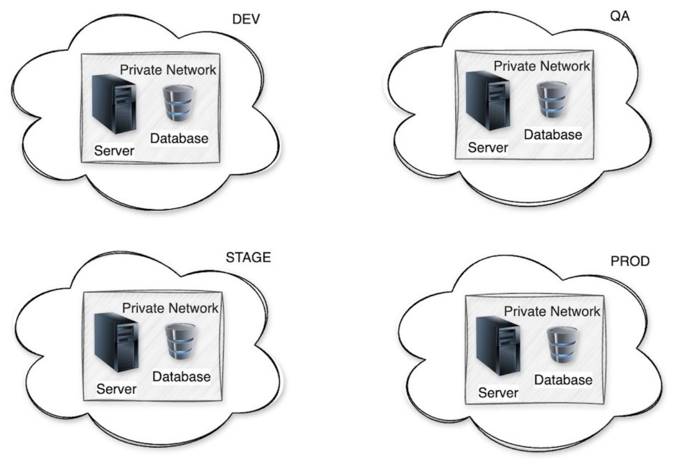
- **Dev + Ops = DevOps**: Teams work together across the software lifecycle
- **Shared Goals**: Build fast, release often, stay stable
- **Cultural Shift**: Break down walls between teams

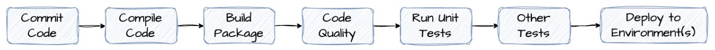
DevOps is about Quick Feedback

- **Business can't wait**: Releasing new features after months slows product innovation
- **Defects/Issues shouldn't wait**: Developers want to be notified about code quality issues or unit test failures immediately
- **Continuous Improvement**: Quick feedback loops help refine features and fix things faster

DevOps is about Automation

- **Manual = Slow + Error-Prone**: Repeating steps by hand increases chances of mistakes
- **Not Scalable**: What works for 1 server won't work for 100 or 1000 or 10,000 servers
- **Automation Brings**:
  - **Speed** – Do things quickly
  - **Reliability** – Same steps, every time (less chance of errors)





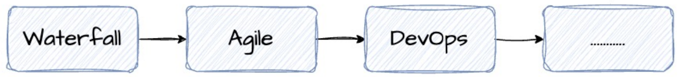
Key DevOps Practices

- **Version Control**: All code (infra + app) tracked in Git
- **CI (Continuous Integration)**: Compile code, build package, check code quality, run unit tests, .. immediately after code is committed to version control
- **CD (Continuous Delivery/Deployment)**: Automating deployment of code
- **Infrastructure as Code (IaC)**: Use code to create your resources
- **Standardization**: Use language neutral and platform neutral tools that can standardize your operations (Docker + Kubernetes)
- **Observability**: Understand what's happening inside a system (Metrics + Logs + Traces)

How Did We Evolve to DevOps?

1. Waterfall Era – Slow & Rigid

- **Waterfall**: One big team working in phases – requirements → design → build → test → deploy
- **Problems**:
  - Long release cycles (months or years)
  - No feedback until late stages
  - Developers and operations teams worked **separately** with conflicting priorities
- **Result**: Slow delivery and unhappy teams!



2. Agile Era – Fast but..

- **What Changed**:
  - Software started being built in **iterations** (2–4 weeks)
  - Agile brought **collaboration** (Business + Dev) and **customer feedback** into development
  - Developers focused on code quality and started writing automation tests - unit tests, ..
- **But Still a Problem**:
  - Dev teams moved fast, but **operations lagged behind**
  - Manual deployments and manual environment creation caused bottlenecks

3. Rise of DevOps – Deliver Fast & Operate Smoothly

- **Goal**: Bridge the gap between **development and operations**
- **Key Enablers**:
  - **Automation**: CI/CD, Containers, IaC
  - **Collaboration**: Shared responsibility
  - **Observability**: Fix issues early, improve continuously
- **Benefits**:
  - Faster delivery with fewer errors
  - Scalable, reliable systems
  - Happier teams and users

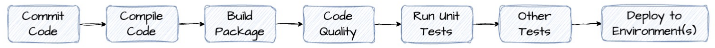
DevOps Is a Mindset Shift

- From **silos** to **shared ownership and responsibility**
- From **"throw code over the wall"** to **"build and run it together"**
- From **manual work** to **automated, continuous delivery**

What is Continuous Integration (CI) and Continuous Deployment (CD)?

What is Continuous Integration (CI)?

- **Scenario**: Developers work on the same code base. Without coordination, compilation issues, code quality issues or unit testing issues pop up late. Can we catch issues early?
- **Continuous Integration (CI)**: Developers frequently merge code into a shared repository. Automated builds and tests run on every change.
- **Goal**: Catch bugs early and integrate code often
- **Benefits**: Fewer integration issues, fewer defects, faster feedback



What is Continuous Deployment/Continuous Delivery?

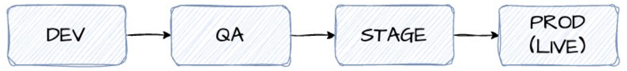
- **Scenario**: After code is tested, deployment still needs manual steps. This delays feedback and delivery. Can we automate deployments too?
- **Continuous Delivery (Deployment)**: Automatically deploy tested code to environments like Dev, QA, and even Production
- **Goal**: Deliver features to users quickly and safely
- **Benefits**: Faster releases, less manual work, consistent deployments

CI/CD Pipeline

- **Build - Test - Deploy**: Code is built, tested, and deployed automatically
- **Triggers**: Starts on every code change or pull request
- **Fast Feedback**: Issues are found and fixed early

How can you implement CI/CD?

- **General Tools**: GitHub Actions, Jenkins, Argo CD
- **Cloud**: AWS CodePipeline, AWS CodeBuild, AWS CodeDeploy, Azure DevOps, Google Cloud Build, Google Cloud Deploy, ..

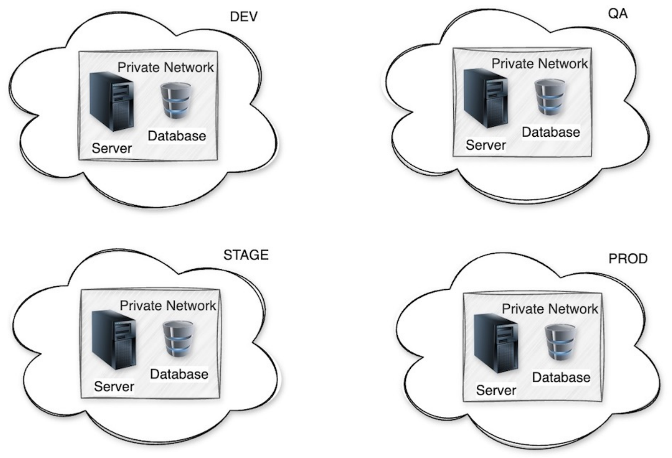


Continuous Deployment vs Continuous Delivery

- **Continuous Delivery**: Code is automatically tested & staged. A release manager clicks "Deploy" to go live.
- **Continuous Deployment**: Developer commits code → All tests pass → Goes live automatically
- **Key Difference**:
  - **Manual Gate**: Continuous Delivery has it, Continuous Deployment doesn't
  - **Risk**: CD (Deployment) needs higher confidence in automated testing and monitoring

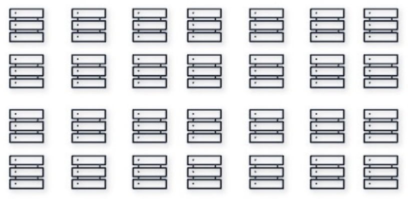
What is Infrastructure as Code?

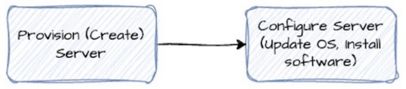
- **Scenario**: Imagine manually setting up VPCs, subnets, virtual machines, databases for Dev, QA, Staging, and Prod. It is time-consuming and error-prone.
- **IaC**: Automate infrastructure setup using code



What is Infrastructure as Code?

- **Definition**: Use code to define and manage cloud infrastructure
- **Goal**: Automate provisioning and configuration of resources
- **Benefits**:
  - Fewer manual errors
  - Faster deployments
  - Consistency across environments
  - Version-controlled changes




Provision (Create) Server
Configure Server (Update OS, Install software)

Two Parts of Infrastructure as Code

- **1. Infrastructure Provisioning**: Create cloud resources like VMs, storage, databases, networks (Tools: Terraform, Pulumi, ..)
- **2. Configuration Management**: Install software, packages, and runtime settings on provisioned resources (Tools: Ansible, Chef, Puppet, ..)
- **IaC = Provisioning + Configuration**
- Automate everything – from servers to software
- Repeatable and scalable infrastructure – just like code

Treat Infrastructure Like Code

- **Store in Git**: Just like application code
- **Test and Review**: Before deploying

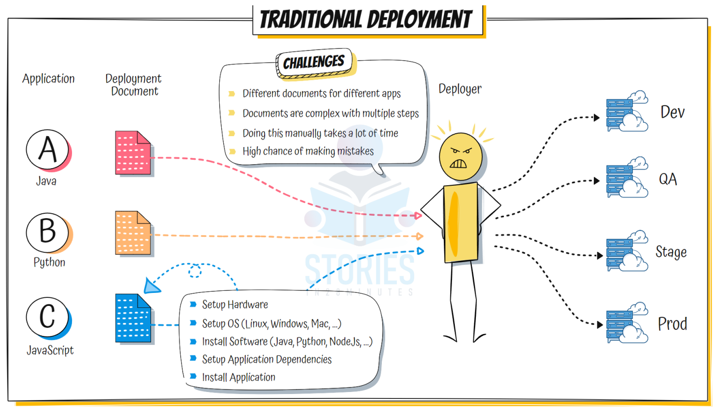
Cloud Managed Services for Infrastructure as Code

- **AWS**: AWS CloudFormation, AWS CDK, OpsWorks (Cloud based Chef, Puppet)
- **Azure**: ARM Templates, Bicep
- **Google Cloud**: Deployment Manager, Terraform (first-class support)

What is Standardization? How do containers and container orchestration help achieve it?

What is Standardization?

- **Scenario**: Imagine a scenario where you are using different processes for different applications built in different programming languages? Can we create a **consistent process** for all apps?
  - Can we have same process across different infrastructure as well? Can we create consistent process irrespective of where I'm deploying in local or on-premises or in the cloud?
- **Standardization**: Enabling **uniform processes and tools** for different types of applications across environments and infrastructure types!


TRADITIONAL DEPLOYMENT
Application
Deployment Document
CHALLENGES
Different documents for different apps
Documents are complex with multiple steps
Doing this manually takes a lot of time
High chance of making mistakes
Deployer
A
Java
B
Python
C
JavaScript
Setup Hardware
Setup OS (Linux, Windows, Mac, ...)
Install Software (Java, Python, NodeJs, ...)
Setup Application Dependencies
Install Application
Dev
QA
Stage
Prod

How Containers Enable Standardization

- **Self-Contained Units**: Containers package the **app + dependencies + config** together (Contain everything that an application needs to run!)
- **Same Image Everywhere**: Dev, QA, Prod – run the same container

Example: Containers Enable Standardization

- You build a Java app with specific versions of Java + libraries
- Package it in a Docker Container Image
- Run the exact same container image on your laptop, test server, and cloud
- **Result**: Works the same everywhere → standardized deployment
- **What's more**: You can use the same process for Python or NodeJs Applications as well. Build a container image and deploy where ever you want.

What is Container Orchestration?

- **Scenario**: You have dozens of containers running your app – across multiple servers. How do you scale them automatically? How do you find out if one of the containers is failing?
- **Kubernetes**: An open-source platform that **orchestrates** (manages) containers – it **automates deployment, scaling, and healing** of containerized applications.

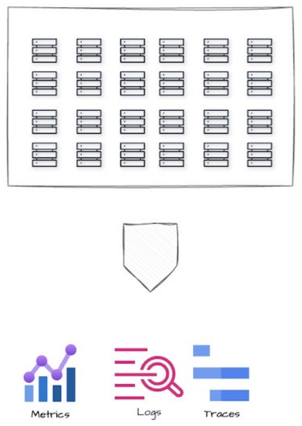
You can Request Kubernetes using a Declarative Approach

- **Run 10 instances of MicroserviceA V1 image all the time**
  - **Deployment and Scaling**: Kubernetes will deploy 10 containers
  - **Healing**: Restarts crashed containers
  - **AutoScaling**: You can increase and decrease instances automatically
- **Run Any Container Image**: Belonging to apps built in any programming language
- **Run Anywhere**: Works on local, on-prem, and all major clouds

What is Observability?

Why Observability?

- **Scenario**: Your app is running slowly or returning errors. But you do not know where or why!
- **Observability**: The ability to **understand what is happening inside your system** just by looking at **external outputs like logs, metrics, and traces**



3 Pillars of Observability

1. Logs

- **What They Are**: Text records of events (e.g., errors, warnings)
- **Use**: Helps answer *what happened*
- **Example**: "PaymentService failed: connection timeout"

2. Metrics

- **What They Are**: Numeric values tracked over time
- **Use**: Helps answer *how is the system performing*
- **Example**: "CPU usage = 80%", "Latency = 220ms"

3. Traces

- **What They Are**: The journey of a request through multiple services
- **Use**: Helps answer *where is the slowdown or failure*
- **Example**: Trace shows 2.5s delay in `OrderService` during checkout

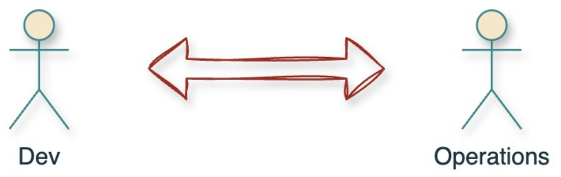
Observability vs Monitoring

- **Monitoring** = Alerts for known issues
- **Observability** = **Deep visibility** to explore unknown issues

Why OpenTelemetry?

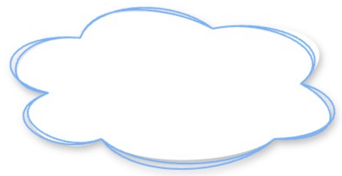
- **Challenge**: Different tools and formats for each pillar
- **Solution**: One standard for collecting telemetry (logs, metrics, and traces)
- **OpenTelemetry**:
  - Open-source, cloud-neutral project under CNCF
  - Collects logs, metrics, and traces using standard APIs and SDKs
  - Supported across different cloud platforms and almost all programming languages
- **Popular Observability Tools Integrating with OpenTelemetry**:
  - **Metrics**: Prometheus, Grafana (visualization), Amazon CloudWatch (AWS), Azure Monitor, Google Cloud Monitoring
  - **Logging**: ELK Stack (Elasticsearch, Logstash, Kibana), Amazon CloudWatch Logs, Azure Log Analytics, Google Cloud Logging
  - **Tracing**: Jaeger, Zipkin, Grafana Tempo, AWS X-Ray, Azure Application Insights, Google Cloud Trace

Can DevOps Be Done Without Cloud?



Can DevOps Be Done Without Cloud?

- **Yes – DevOps is Cloud-Friendly, Not Cloud-Dependent**: DevOps is a **culture + process**, not tied to **where** your infrastructure runs. You can do DevOps **with or without cloud**.



DevOps Without Cloud is Possible

- **On-Premises Servers**: Use your own data centers
- **Same Practices**: CI/CD, monitoring, IaC – still apply
- **Same Tools**: Jenkins, Git, Docker, Terraform work on-prem too
- **DevOps ≠ Cloud**: You can adopt DevOps on-prem or hybrid
- **HOWEVER Cloud Supercharges DevOps**: Easier infra + more automation

Cloud Makes DevOps Easy

- **On-Demand Infrastructure**: Dev teams get servers instantly
- **CI/CD Pipelines**: Implement CI/CD using cloud services
- **IaC (Infrastructure as Code)**: Create/manage cloud resources with code
- **Observability**: Observability through cloud services

Previous
Java Interview - Diagrams

Next
DevOps Interview - Diagrams