# DevOps Quick Facts

# **DevOps**

## **What is it?**

DevOps can be called out as a concept. The fundamental philosophy of DevOps is that developers, operations staff, and support staff work closely together regularly.

It involves collaboration between development and operations throughout a product or service's development and delivery life cycle.

We work together in development and operations in the entire lifecycle, from design to delivery to prod support.

DevOps concept brings hope to lean and agile development and delivery, an understanding that a product or service of software application is only partially done once it has been delivered to a user successfully while meeting their expectations of user experience, availability, and performance.

## **Benefits**

* It improved deployment frequency.
* Faster resolution of problems.
* Improved productivity.
* Higher employee engagement.
* Faster delivery of features.
* More stable operating environments.
* Improved communication and collaboration.
* More time to innovate.

## **Goals**

* Improve deployment frequency.
* Achieve faster time to market.
* Lower failure rate of new releases.
* Shorten lead time between fixes.
* Improve mean time to recovery.

## **Basic Principles**

1. **Communication:** Strong shared communication that can bind and act as a bonding agent between developers and operations, as well as other stakeholders in the entire process of SDLC.
2. **Collaboration:** In a distributed environment, collaboration is vital. Building relationships across teams and using natural partnerships will help to put products or services into production faster. It will also help troubleshoot issues and resolve them faster.
3. **Culture: A culture that encourages learning, understanding, and working together between all stakeholders involved (**business, development, operations, security, testing, performance engineering, and other teams who are engaged in delivery)

## **Working Principles**

1. Eliminating “It’s not my job”. It does not necessarily mean that you will be helping everyone with every task, but a culture where we take collective ownership and assist when and if you can or point to the right person/team who can.
2. DevOps is a team sport—approachability and the ability to build genuine team connections.
3. Trust in teammate’s performance.
4. Think of everything in terms of repeatable tasks that can be automated.
5. Recognizes the interdependence of stakeholders and helps them improve productivity and delivery with a lower rate of failure.
6. Improve deployment frequency with a lower failure rate and faster recovery.
7. Support teams to enable continuous workflow by complementing agile development and delivery.

## **Values**

* Increased collaboration.
* Decreased silos.
* Shared responsibility.
* Autonomous teams.
* Improving quality.
* Valuing feedback.
* Increasing automation.

## **Components**

* Development
* Testing
* CI/CD
* Deployment
* Monitoring
* Communication

## **Elements of DevOps**

The Most Important Elements of DevOps are people, processes, and technology.

People involve culture and change.

The process involves adopting the process that makes the most sense for the people and the business.

## **Key Concepts**

### Automated Delivery Pipeline

### The Pipeline

The pipeline is a sequence of stages outlining the required actions from the initial check-in of new code into version control to the final deployment into a production environment. Assuming we have three deployment environments (Development, Test, and Production), then a typical delivery pipeline would have the following steps:

1. The developer pushes code to source control.
2. A build is triggered to compile the source code and run tests to ensure everything is in order.
3. An artifact is created, given a unique version number, and published to an artifact repository.
4. Deploy the latest artifact to the Development environment at a defined schedule. The schedule could be every hour, three times a day, after the successful publishing of an artifact, or whatever suits you.
5. Deploy to the Test environment the artifact last deployed to the previous environment at a defined schedule. This schedule can be different from the Development environment schedule.
6. Run smoke and regression tests.
7. Deploy to the Production environment the artifact last deployed to the previous environment at a defined schedule. This schedule can be different from the schedule of the other environments.
8. Testing.

Each step needs to be executed successfully before the next step is triggered.

### Automation

Automation is a crucial part of the delivery pipeline. Once a developer pushes new code to source control, every step should be automatically triggered and executed. To achieve automation, the following points need to be in place:

* Source code version control system.
* Continuous integration server that supports automatic triggers and scheduling.
* Automated compiling and testing using a build tool.
* Source code change can trigger continuous builds.
* Versioning of artifacts.
* Database scripts need to be part of the source code and version controlled.

### Configuration Management

To achieve configuration management automation, externalize the configuration from the source code into a configuration file (e.g., a properties file for Java or an App.config file for .NET applications).

There are two approaches to deciding what config files to use when:

1. **At Run Time:** When the application starts, it will determine the environment it’s running and load the appropriate configuration file. Hence, this approach requires a separate configuration file for each environment.
2. **At Deployment Time:** The second approach will instead write the configuration file at deployment time. The deployment script will write the configuration file with the appropriate values depending on our deploying environment.

### Regular Integration

A regular schedule for integrating the application or applications is essential in achieving a feedback cycle.

The primary blocker for good integration cycles is manual processes. Hence, automation is critical to achieving regular and continuous integration. Sample manual steps that need to be automated are:

* **Building the application**
* **Deploying the application**
* **Testing**
* **Automated monitoring and Reporting**
* **Troubleshooting**

### Automated Monitoring & Health Checks

DevOps involves operational duties. Alerts and monitoring are essential, where problems can be captured and reported before they are noticed by users or impact the user experience.

A few basic essential checks are:

* Periodically checking that all applications and services are **reachable and responsive.**
* Alerts and monitoring set up for infrastructure involving servers and networks (CPU utilization, memory consumption, and disk storage are examples in the case of servers).

### Infrastructure as Code

The best practices of infrastructure as code:

* Use **version control.**
* **Use good naming conventions and organization.**
* **Reuse scripts and code as much as possible.**
* **Automate the running of scripts using tools like Jenkins.**
* Test your code using automated tests.

## Best Practices

A few best practices in DevOps

* Active Stakeholder Participation
* Automated Testing
* Integrated Configuration Management
* Integrated Change Management
* Continuous Integration
* Integrated Deployment Planning
* Continuous Deployment
* Production Support
* Application Monitoring
* Automated Dashboards

## Possible Roles

Seven possible roles and related skill sets to adopt a successful DevOps approach.

### DevOps Champion

The role of the DevOps Champion is to anchor the DevOps philosophy within the company. They advocate for the operational and developmental gains that stem from DevOps' enhanced agility. Serving as agents of change, they secure a commitment from developers and operations staff, pinpoint critical roles for effective DevOps practices, and guarantee that IT professionals are skilled and authorized to implement these changes. Moreover, they spearhead the shift toward a resilient organizational mindset that encourages learning from rapid failure to foster continuous improvement.

### Release Coordinator

The Release Coordinator, a release engineer or product stability manager, is tasked with the governance and synchronizing the product's journey from development to production. Unlike traditional project managers, they delve into the intricacies of coordinating, integrating, and streamlining the development, testing, and deployment process to facilitate uninterrupted delivery. They aim to construct and uphold a seamless end-to-end delivery system for applications.

### Automation Architect

The Automation Architect is pivotal in DevOps. Also referred to as integration specialists, they formulate and execute strategies for steady deployments while ensuring system availability in both production and staging environments. They comprehensively automate processes across various DevOps tools and cloud services, incorporating efficiency and lean methods into core DevOps practices, especially in organizations across multiple locations.

### Software Developer/Tester

In a DevOps setting, the software developer evolves to include testing within their purview, expanding their responsibilities significantly. They are charged with converting new demands into executable code and managing unit testing, deployment, and continuous monitoring. This integrated role surpasses the traditional boundaries of coding to specification and promotes a collaborative approach known as DevTestOps, emphasizing the inclusion of testing within the development cycle. This necessitates a shift towards automated testing to maintain agility and quality.

### Experience Assurance (XA) Specialist

The transition to DevOps transforms the quality assurance domain, giving rise to the Experience Assurance (XA) Specialist. This role transcends traditional QA testing by focusing on the user's experience with new features and services, highlighting the importance of user experience testing alongside functional testing.

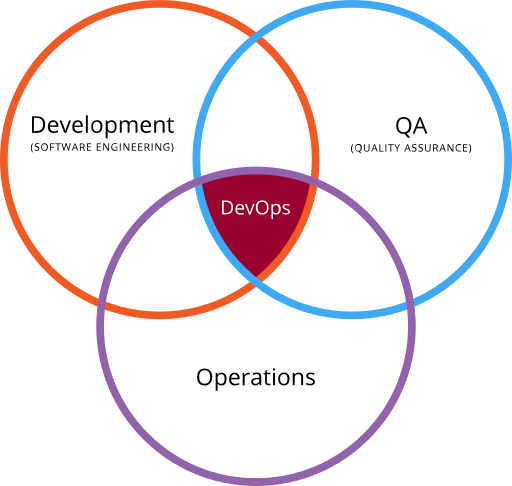
### Security Engineer

In a DevOps culture, security is a fundamental aspect of the development process. Security Engineers collaborate closely with developers from the outset, integrating security measures early to embed security within the product, moving away from the traditional approach where system security is addressed only at the final stages.

### Versatile IT Operative

This role represents a modern take on traditional IT operations or systems administration. Instead of merely ensuring server uptime, these professionals now play a multifaceted role, including managing applications to prevent service disruptions. Their focus has shifted from enforcing rigid controls and extensive QA in pre-production environments to embracing more frequent releases and active involvement in post-production activities, reflecting DevOps's more collaborative and integrated approach.

The fast-paced DevOps environment requires a new working culture where operations or admin experts get involved throughout development.



(Borrowed from Jenkins site.)

coup 
DevOps life cycle processes 
unn 

# DevOps Subtopics

DevOps is a broad field encompassing various practices, methodologies, and tools to unify software development (Dev) and operation (Ops). The main subtopics under DevOps include:

### 1. Version Control

1. Managing source code is like keeping all document drafts so you can see the history of changes, who made them, and when.
2. Branching strategies: Think of this as working on different book chapters independently without affecting the main text until they're ready to be added.
3. Collaboration techniques: This involves methods for multiple people to work on the same set of files without overwriting each other's changes.

### 2. Continuous Integration (CI) and Continuous Deployment (CD)

1. Automated testing means running tests automatically to catch bugs whenever someone changes the code.
2. Build automation: This is the process of scripting or automating a wide variety of tasks that software developers do in their day-to-day work, like compiling computer source code into binary code.
3. Deployment strategies: These are plans to move code tested to a live environment where everyone can use it.
4. Pipeline design and management: Setting up a series of automated processes so that code can go from development to deployment with minimal human intervention.

### 3. Configuration Management

1. Infrastructure as code: Writing code to manage configurations, enabling you to treat your servers and other infrastructure like software.
2. Automated provisioning: This is the automatic setup of an IT environment, like creating servers and installing and configuring software on them.
3. Configuration consistency: Ensuring that all environments, from development to production, are configured similarly.

### 4. Containerization and Orchestration

1. Container creation and management: Packing up software with everything needed to run it (like code, runtime, and libraries) so it can run anywhere.
2. Service orchestration: This is like conducting an orchestra, but instead of musicians, you're managing services that run across multiple containers.
3. Scaling and networking of containers: Adjusting the number of containers based on demand and enabling them to talk to each other.

### 5. Monitoring and Logging

1. Application and infrastructure monitoring: Keep an eye on your apps and servers to ensure they work correctly and efficiently.
2. Log aggregation and analysis: Collecting logs (records of events) from different sources to determine what's happening in your system.
3. Alerting and incident response: Notifying the right people when something goes wrong and having a plan to fix it.

### 6. Infrastructure as Code (IaC)

1. Templating and provisioning infrastructure: Creating templates for infrastructure setup so you can recreate it quickly, like a recipe for setting up a computer network.
2. Version control for infrastructure: Tracking changes in the infrastructure setup just like you would with source code.
3. Infrastructure automation and scaling: Automatically adjust the size or power of your infrastructure based on the workload.

### 7. Cloud Computing

1. Cloud services and management: Using and managing resources like servers and databases over the internet.
2. Hybrid and multi-cloud strategies: Mixing different types of clouds (private, public) and cloud providers to balance cost, performance, and security.
3. Cloud-native applications: Apps designed to take full advantage of cloud computing features.

### 8. Security (DevSecOps)

1. Integrating security into the DevOps process: Make sure security is a part of the app lifecycle, not an afterthought.
2. Automated security testing: Using tools to check for vulnerabilities in your code automatically.
3. Compliance and vulnerability management: Making sure your applications meet legal and security standards and managing weaknesses that could be exploited.

### 9. Automation

1. Scripting and task automation: Writing scripts to perform routine tasks automatically to save time and reduce errors.
2. Workflow automation: Using technology to streamline processes in the software delivery cycle.
3. Process automation: Using technology to manage, monitor, and automate daily tasks without human intervention.

### 10. Performance Tuning

1. Application performance optimization: Making your applications run faster and more efficiently.
2. System performance optimization: Tuning your servers and other infrastructure to their best.

### 11. Collaboration and Communication

1. Team collaboration techniques: Ways for teams to work together effectively, like standup meetings or pair programming.
2. Effective communication in DevOps culture: Sharing information openly and frequently to ensure everyone is aligned.
3. Tools for team coordination: Software that helps manage projects, track issues, and facilitate continuous communication.

### 12. Testing

1. Automated testing strategies: Setting up tests that run automatically whenever you make changes to the code to ensure nothing breaks.
2. Integration and system testing: Ensure all your application pieces work together correctly.
3. Performance and security testing: Testing how well your application performs under stress and ensuring it's secure from threats.

### 13. Release Management

1. Managing release cycles: Planning and overseeing releasing new software versions.
2. Rollback strategies: Plan to revert to a previous version if a new release causes problems.
3. Change management: Processes to handle software development changes and ensure they're made smoothly without disrupting service.

### 14. Site Reliability Engineering (SRE)

1. Reliability and uptime management: Ensuring services are available when users need them.
2. Incident management and postmortems: Handling problems when they occur and analyzing them afterward to avoid recurrence.
3. Service level objectives (SLOs) and indicators (SLIs): Setting and measuring service availability and performance goals.

### 15. Project and Process Management

1. Agile methodologies: Approaches to software development that emphasize incremental delivery, collaboration, and flexibility.
2. Lean practices: Techniques that focus on delivering value to the customer.
3. Kanban and Scrum methodologies: Systems for managing work emphasizing just-in-time delivery and adapting to change.

Grasping these concepts will provide a strong foundation for anyone starting in DevOps. They form the essential toolkit for managing modern software development and operational challenges.