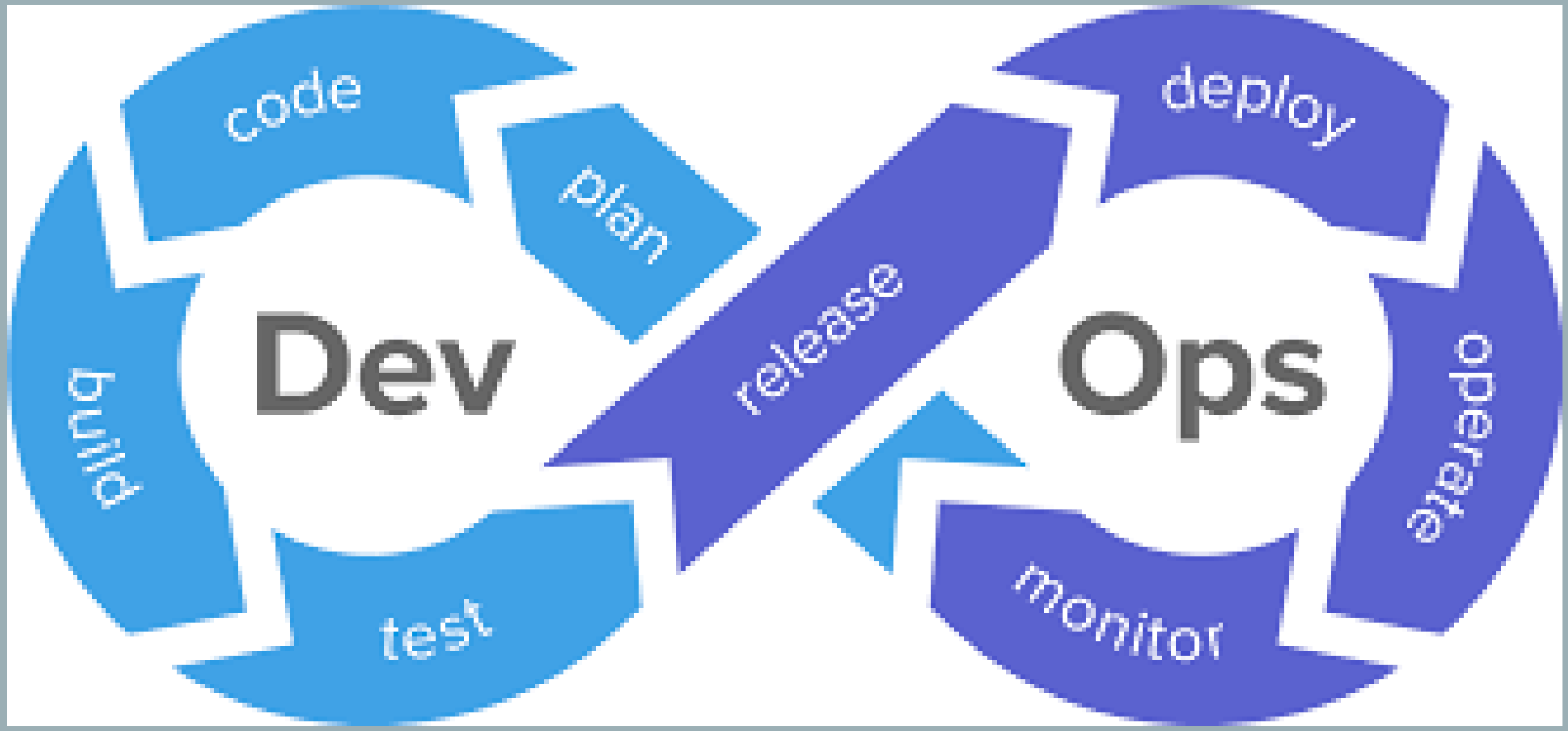
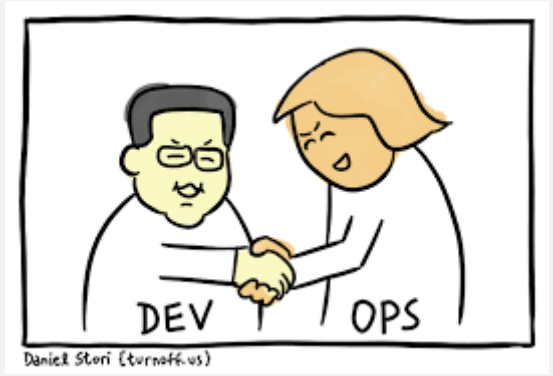
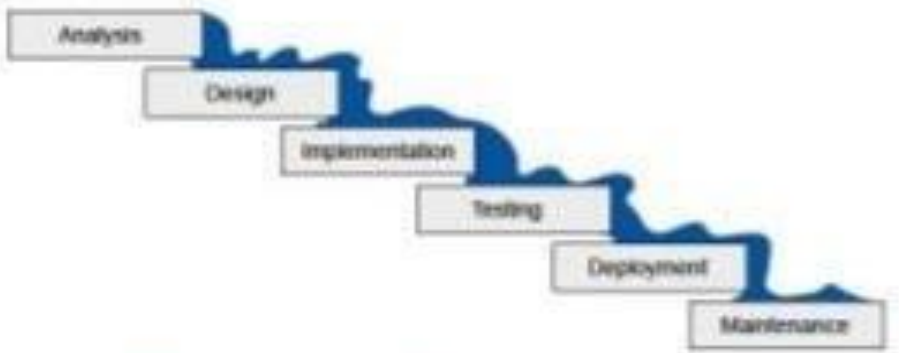

code
plan
build
test
Dev
release
deploy
operate
monitor
Ops

WHY DEVOPS



Waterfall



Agile

DevOps

WHAT IS DEVOPS

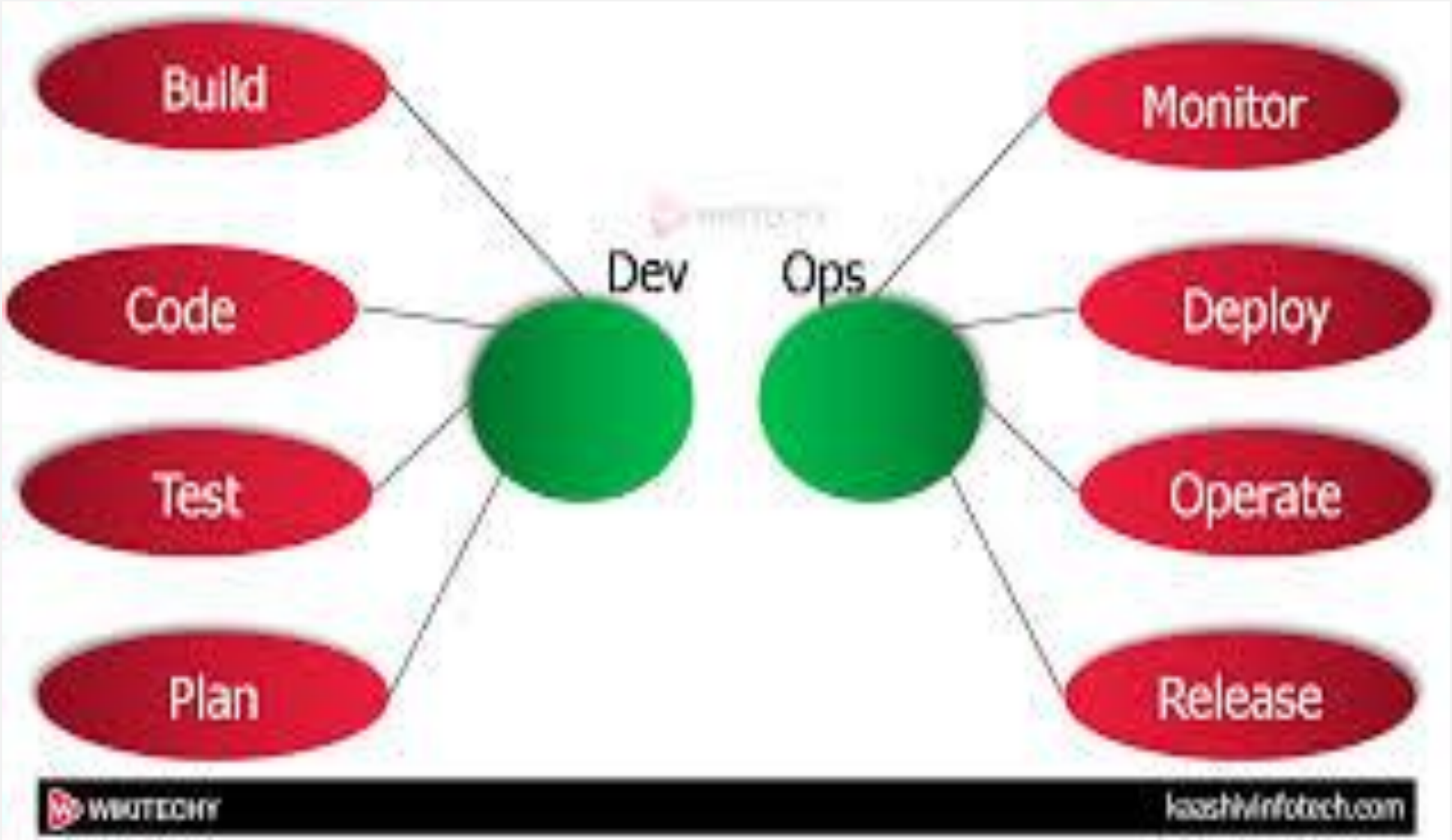
- DevOps is a set of practices that works to automate and integrate the processes between software development and IT teams, so they can build, test, and release software faster and more reliably.

WHAT IS DEVOPS

- The term DevOps was formed by combining the words “development” and “operations” and signifies a cultural shift that bridges the gap between development and operation teams, which historically functioned in isolation

DEVOPS PROCESS

- Because of the continuous nature of DevOps, practitioners use the infinity loop to show how the phases of the DevOps lifecycle relate to each other. Despite appearing to flow sequentially, the loop symbolizes the need for constant collaboration and iterative improvement throughout the entire lifecycle.


Build
Code
Test
Plan
Dev
Ops
Monitor
Deploy
Operate
Release
kaashivinfotech.com

DEVOPS LIFECYCLE

- **Agile Planning**
- **Continuous Development**
- **Continuous Testing**
- **Continuous Integration and Continuous Delivery**
- **Continuous Deployment**
- **Continuous Monitoring**
- **Infrastructure as a Code**

AGILE PLANNING

- In contrast to traditional approaches of project management, Agile planning organizes work in short iterations (e.g. sprints) to increase the number of releases. This means that the team has only high-level objectives outlined, while making detailed planning for two iterations in advance. This allows for flexibility and pivots once the ideas are tested on an early product increment.

CONTINUOUS DEVELOPMENT

- The concept of continuous “everything” embraces continuous or iterative software development, meaning that all the development work is divided into small portions for better and faster production. Engineers commit code in small chunks multiple times a day for it to be easily tested.

CONTINUOUS TESTING

- A quality assurance team sets committed code testing using automation tools like Selenium, Ranorex, UFT, etc. If bugs and vulnerabilities are revealed, they are sent back to the engineering team. This stage also entails version control to detect integration problems in advance. A Version Control System (VCS) allows developers to record changes in the files and share them with other members of the team, regardless of their location.

CI/CD

- The code that passes automated tests is integrated in a single, shared repository on a server. Frequent code submissions prevent a so-called “integration hell” when the differences between individual code branches and the mainline code become so drastic over time that integration takes more than actual coding.
- Continuous delivery, is an approach that merges development, testing, and deployment operations into a streamlined process as it heavily relies on automation. This stage enables the automatic delivery of code updates into a production environment

CI/CD

- Continuous Integration is a development practice where developers integrate code into a shared repository frequently, preferably several times a day. Each integration can then be verified by an automated build and automated tests. While automated testing is not strictly part of CI it is typically implied.
- One of the key benefits of integrating regularly is that you can detect errors quickly and locate them more easily. As each change introduced is typically small, pinpointing the specific change that introduced a defect can be done quickly.
- In recent years CI has become a best practice for software development and is guided by a set of key principles. Among them are revision control, build automation and automated testing.

CI/CD

- continuous deployment have developed as best-practices for keeping your application deployable at any point or even pushing your main codebase automatically into production whenever new changes are brought into it. This allows your team to move fast while keeping high quality standards that can be checked automatically.

CONTINUOUS INTEGRATION BENEFITS

Continuous Integration brings multiple benefits to your organization:

- Say goodbye to long and tense integrations
- Increase visibility enabling greater communication
- Catch issues early
- Spend less time debugging and more time adding features
- Build a solid foundation
- Stop waiting to find out if your code’s going to work
- Reduce integration problems allowing you to deliver software more rapidly

CI PRACTICES

- Maintain a single source repository
- Automate the build
- Make your build self-testing
- Every commit should build on an integration machine
- Keep the build fast
- Test in a clone of the production environment
- Make it easy for anyone to get the latest executable version
- Everyone can see what’s happening
- Automate deployment

CI PROCESS

- Developers check out code into their private workspaces
- When done, commit the changes to the repository
- The CI server monitors the repository and checks out changes when they occur
- The CI server builds the system and runs unit and integration tests
- The CI server releases deployable artefacts for testing
- The CI server assigns a build label to the version of the code it just built
- The CI server informs the team of the successful build
- If the build or tests fail, the CI server alerts the team
- The team fixes the issue at the earliest opportunity
- Continue to continually integrate and test throughout the project

CONTINUOUS DEPLOYMENT

- At this stage, the code is deployed to run in production on a public server. Code must be deployed in a way that doesn’t affect already functioning features and can be available for a large number of users. Frequent deployment allows for a “fail fast” approach, meaning that the new features are tested and verified early. There are various automated tools that help engineers deploy a product increment. The most popular are Chef, Puppet, Azure Resource Manager, and Google Cloud Deployment Manager.

CONTINUOUS MONITORING

- The final stage of the DevOps lifecycle is oriented to the assessment of the whole cycle. The goal of monitoring is detecting the problematic areas of a process and analyzing the feedback from the team and users to report existing inaccuracies and improve the product's functioning.

INFRASTRUCTURE AS A CODE

- Infrastructure as a code (IaC) is an infrastructure management approach that makes continuous delivery and DevOps possible. It entails using scripts to automatically set the deployment environment (networks, virtual machines, etc.) to the needed configuration regardless of its initial state.
- Without IaC, engineers would have to treat each target environment individually, which becomes a tedious task as you may have many different environments for development, testing, and production use.
- Having the environment configured as code, you

1. Can test it the way you test the source code itself and
2. Use a virtual machine that behaves like a production environment to test early.

AGILE DEVELOPMENT

- Agile software development refers to a group of software development methodologies based on iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams.
- Agile methods or Agile processes generally promote a disciplined project management process that encourages frequent inspection and adaptation, a leadership philosophy that encourages teamwork, self-organization and accountability, a set of engineering best practices intended to allow for rapid delivery of high-quality software, and a business approach that aligns development with customer needs and company goals


Software Development Cycle
1 Planning
2 Analysis
3 Design
4 Implementation
5 Testing & Integration
6 Maintenance

WHAT IS SCRUM

- Scrum is a subset of Agile. It is a lightweight process framework for agile development, and the most widely-used one.
- A “process framework” is a particular set of practices that must be followed in order for a process to be consistent with the framework. (For example, the Scrum process framework requires the use of development cycles called Sprints, the XP framework requires pair programming, and so forth.)
- “Lightweight” means that the overhead of the process is kept as small as possible, to maximize the amount of productive time available for getting useful work done.

SO DEVOPS IS AGILE?

- DevOps and agile aren't strictly the *same* thing, but they're far from mutually exclusive. Both value feedback, continuous development, fast iterative releases and cross-team collaboration. The culture they each hope to inspire is very similar.
- It could be said that at its core, DevOps is essentially an extension of agile software development. It doesn't only focus on the software; instead, the entire work lifecycle becomes more efficient and teams collaborate better. When it comes to diving deeper and really defining DevOps – much like agile – it can depend on the exact requirements and nature of your organization. The biggest thing to keep in mind is that however you choose to approach DevOps, it's important all teams across the business are coming at it from the same direction.

AGILE AND DEVOPS

- While Agile and DevOps started as independent methodological movements, they share a number of traits focused on improving the efficiency and speed of teams. As organizations become more Agile and refine their project management skill sets, they increasingly depend on technical teams being able to keep pace and maintain a certain flexibility.
- This is where DevOps comes in . The DevOps approach helps development groups utilize new tools, automation, and different cultural strategies to change not just how they work themselves, but how they work with others. It becomes a symbiotic relationship where product teams work hand in hand with developers and testers and the like to ensure everyone has more contextual awareness. This promotes a greater overall quality of deliverables in a shorter period of time.

DEVOPS CULTURE

- Often, an organization's challenges with DevOps do not lie in the technology or processes themselves, but in the people part of the equation – how we work together to deliver value via cross-functional teams. To understand what building a DevOps culture means, we need to look at what culture encompasses and how our people, process and technology work together to enable a successful DevOps life cycle.

DEVOPS CULTURE

- Learn from and develop skills from mistakes
- · Focus on the important priorities
- · Capture collaboration
- · Perfect your tooling

DEVOPS PRINCIPALS

- DevOps is the collection and collaboration of people, processes and technology to deliver software value continuously. Here are some core DevOps principles you can use to guide you on your DevOps journey.

CREATE WITH THE END IN MIND

- High-performing teams work best when leaders and individuals understand their role in the team. Therefore, creating with the end in mind involves reassessing and often asking big picture questions, like “Who is this delivery for?” and “Why are we doing it this way?” Keep the vision in mind and make adjustments along the way.

CROSS-FUNCTIONAL TEAMS

- Many products and deliveries involve multiple people. For example, some of the best Marvel cinematic universe (MCU) movies were made thanks to the help of multiple production teams and partner companies coming together to build the powerhouse we know today. Not to mention, in my opinion, some of the best superhero movies of our time. We succeed with software delivery in the same way: by having fully responsible cross-functional delivery teams.

COMMITMENT TO THE JOURNEY

- Even with cross-functional delivery teams and creating with the end in mind, our teams still may not get everything right the first time around. Committing to DevOps is about accepting any failures that come during the journey. Committing to the journey doesn't mean you can't define what it involves, even if it's out of the ordinary. Experimentation is also a part of how we, as developers, adapt to change and adopt novel approaches to delivering faster, cheaper and better software.

THE DEVOPS PRINCIPLES AS GUIDES

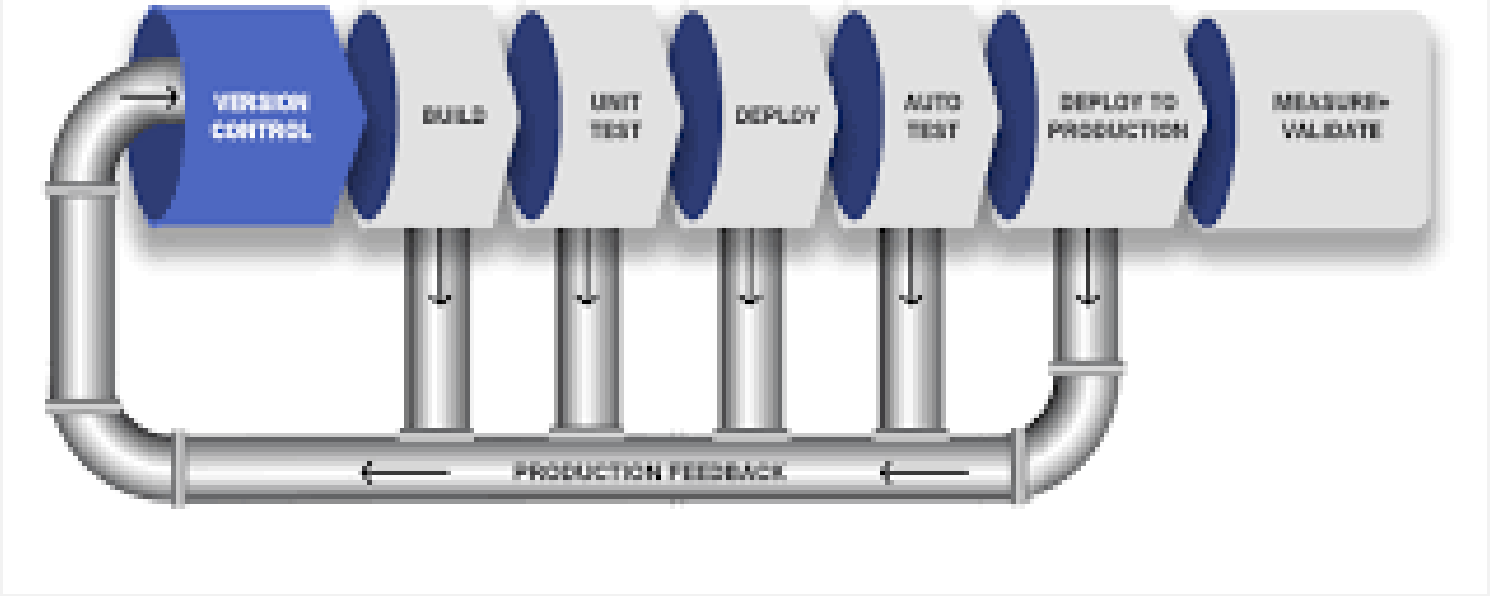
- Use these principles as the foundation for how to work together, but also use the right tools, and define and improve processes along the way. In photography, there are three components that serve as building blocks to the final images we capture: aperture, shutter speed and ISO. In DevOps, it’s our people, processes and technologies. Achieving high performance with DevOps involves all of these components working together and adjusting to our environment, situation or team.

DEVOPS TOOLS

- Docker
- Ansible
- Git
- Puppet
- Chef
- Nagios
- Splunk
- Bamboo
- Kubernetes
- Selenium
- Maven

DEVOPS PIPELINE

- To implement DevOps in any organization, you need to create a DevOps pipeline.
- A pipeline in software development consists of set of tools, flows, and automated processes, enabling teams to leverage technologies to build and deploy software.
- **Core components of DevOps pipelines:**
- **Continuous Integration Tools** –merge code in a central repository
- **Continuous Deployment Tools** — configuration and infrastructure management
- **Continuous Testing Tools**– automated tests
- CI/CD pipeline must be well-defined as per the organizational structure to create a modern DevOps environment along with strong collaboration and automated processes.


VERSION CONTROL
BUILD
UNIT TEST
DEPLOY
AUTO TEST
DEPLOY TO PRODUCTION
MEASURE+ VALIDATE
PRODUCTION FEEDBACK

SECURITY ISSUES WITH DEVOPS

- Cloud Security: Compared to a traditional, on-premise deployment, the cloud presents a much broader attack surface, with loosely defined and fuzzy network boundaries. Nearly any provisioned resource can be configured to be accessible to public internet traffic with just a few clicks or lines of code
- Poor Access Controls and Secrets Management
- With highly automated builds and deployments, secrets management and tight access controls are essential.
- Secrets may include API tokens, SSH Keys, privileged account credentials, etc. These might be used by containers, services, employees and many more entities.
  All too frequently, these critical passwords and keys are poorly managed (exposed) and are frequent targets of attackers.
- Additionally, to ensure a smooth and quick workflow, DevOps teams often allow almost unrestricted access to privileged accounts such as admin, root, etc.
- When multiple individuals use and share credentials of confidential accounts, and when processes run with elevated privileges, the possibility of these excessive permissions being abused increases significantly.

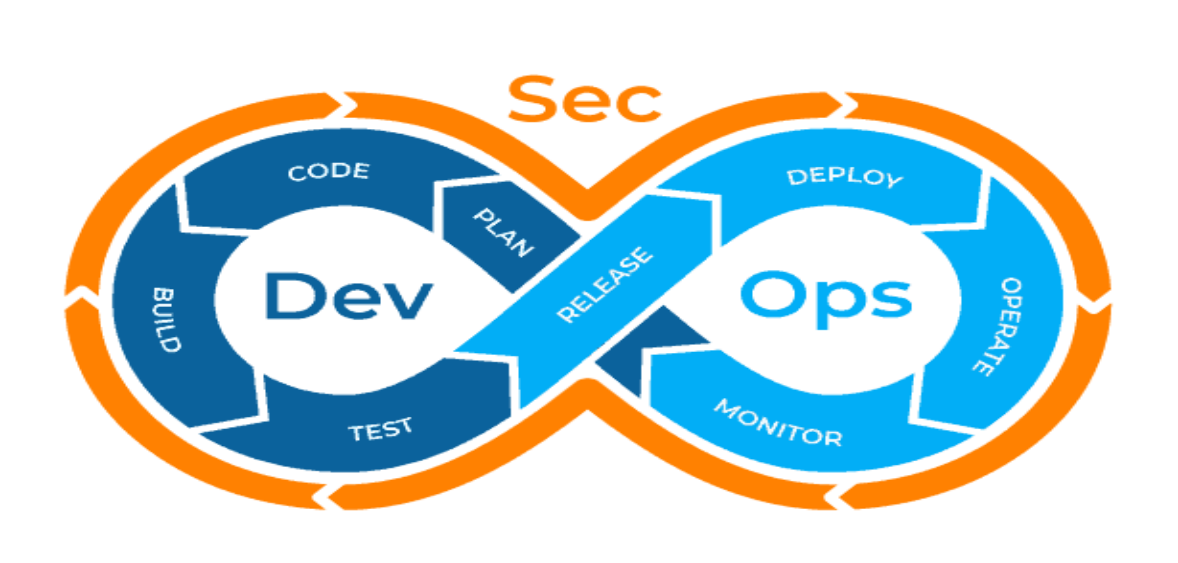
SECURITY ISSUES WITH DEVOPS

- Containerization:
- Containerization and container tools have a special role in boosting productivity in a DevOps environment. But they can often be the reason for security concerns in your DevOps process.
- Container apps enjoy the flexibility of running on any computing platform or cloud without dependencies, which is the portability advantage. But this can also be a security challenge in the absence of proper controls, as containers share OS among themselves and less-often scanned for vulnerabilities.

- Risk Involved with devops tools
- DevOps typically relies on cloud infrastructure, as well as open-source or immature tools. Some tools can dramatically increase productivity, but they can also carry potential risks for DevOps environments.

DEVSECOPS

- DEVSECOPS involves injecting security practices into an organization's DevOps pipeline. The goal is to incorporate security into all stages of the software development workflow



KEYS TO SUCCESSFUL DEVOPS SECURITY PROGRAMS

- Automation
- *With DevOps, you have to move super fast. There can be no 'manual' in that process. If you don't have automation, you'll never be successful." —Chris Romeo*
- CEO, principal consultant, and co-founder of Security Journey.

- Security controls and tests need to be embedded early and everywhere in the development lifecycle, and they need to happen in an automated fashion because organizations are pushing new versions of code into production 50 times per day for a single app
- Automation has become a key DevSecOps characteristic in organizations with highly mature DevOps practices.

KEY TO SUCCESSFUL DEVOPS SECURITY PROGRAM

- Static application security testing VS Dynamic application security testing
- Trying to run automated scans on your entire application source code each day can consume a lot of time and break developer's ability to keep up with daily changes.
- Unlike static analysis, which focuses on finding potential security issues in the code itself, DAST looks for vulnerabilities in real time, while the application runs.
- DAST can be automated and run against recent or new code changes to catch security vulnerabilities listed on OWASP list of most common flaws like SQL injection errors that might have been missed during SAST

KEY TO SUCCESSFUL DEVOPS SECURITY PROGRAM

- Code Dependencies
- Understanding open-source use is key to wider adoption of DevSecOps practices,
- The cloud has fueled innovation in unprecedented ways, enabling companies to meet customer requirements faster. This pace of innovation has also brought about the increased use of open-source software to assemble applications, rather than developing them from scratch
- Developers often don't have the time to review code in their open-source libraries or read the documentation.So automated processes for managing open-source and third-party components are a fundamental requirement for DevSecOps.
- Developer has to know if open-source usage is causing contextual and other vulnerabilities in the code, and what impact those vulnerabilities might have on dependent code.
- Code dependency checks are fundamental to DevSecOps, and utilities such as the OWASP Dependency-Check can help ensure that you do not use code with known vulnerabilities in your software

THREAT MODELING

- A threat modeling exercise can help your security organization get a better idea of threats to your assets, the types and sensitivities of your assets, the existing controls for protecting those assets, and any gaps in your controls that need to be addressed.
- Such assessments can help identify flaws in the architecture and design of your applications that other security approaches might have missed.
- Conducting threat modeling in a DevOps environment can be challenging because of the notion that it can slow down the velocity of a CI/CD environment
- You can't automate the threat-modeling processes in the same way you can for almost every other facet of DevOps. But threat modeling is still crucial for the overall success of your DevOps efforts because it gets your developers to think of their software from the perspective of an attacker

IDENTIFYING CI/CD PIPELINE AND COMPONENTS

- It's also important to have visibility into the CI/CD pipeline -- its assets and boundaries, the steps, tools and code repositories -- to establish an asset registry and a full understanding of the application architecture and software development lifecycle. The aim is to collect any data that will show changes to assets and actions. List all the sources of possible telemetry the pipeline produces -- system logs are obvious sources, but also look into repository, build and deployment logs. Ensure cloud-based resources output to relevant logs, and that logs capture software build and code repository activity.

CHECKS AND SAFEGUARDS TO COMMIT CODE

- In the development phase where code has not yet been checked into version control, the most important security tool is an IDE security plugin. This analyzes code as it is written in real time, warns developers if they will potentially introduce a vulnerability, and provides remediation advice. Examples include Microsoft DevSkim, SonarLint and Puma Scan.
- Code written by less experienced developers should be peer-reviewed using a security checklist such as OWASP's Cheat Sheet Series. Delivering code in small units makes it easier for these manual code reviews to detect any errors.

ANALYSIS OF COMMITTED CODE

- Developers should receive prompt feedback once they've committed their code. Static code analysis tools are the best for the job because they do not require the application to be running, and many provide remediation advice as well.
- Send code scan reports to security or development teams so they can prioritize any follow-up actions. Add any warning or alert generated during these tests (and any other tests) to a bug tracker, such as Jira, to ensure the vulnerability is assigned to someone and fixed, not forgotten.

TESTING FOR SECURITY

- Run security unit tests based on the application's threat model just as you would run regular unit tests against components and functions. Analyze misuse and abuse cases, as well as edge cases. Tests on authentication, session management and other security requirements must validate that both allowed and disallowed actions are handled correctly. Run these scans against the repository, not the build pipeline, otherwise failed security checks may stop a build and frustrate everyone. Brakeman and phan are commonly used analysers, while tools such as SonarSource and Veracode are available as SaaS or for local installation.

OPEN SOURCE VULNERABILITIES

- Another important check at this stage is to analyze imported open source libraries and components for known vulnerabilities. These third-party packages play a vital role in modern software development but new vulnerabilities can crop up at any time. These vulnerabilities could impact the security of an application even if its code hasn't changed.
- Software composition analysis (SCA) tools analyze open source code, third-party components, and binaries to provide real-time security alerts, and also flag compliance and licensing issues. Two popular SCA tools are WhiteSource Bolt and snyk.

CONTINUOUS MONITORING AFTER DEPLOYMENT

- Once you've introduced an application and its supporting infrastructure you must continuously scan and monitor everything to detect, prevent and remediate new vulnerabilities or weaknesses because the threat landscape constantly evolves. To help visualize application state and performance, tools such as Kibana and Grafana can bring diverse data sets into dashboard-style reports and generate alerts if thresholds are broken. Netflix and NASA use the open source StackStorm which uses an "If This Then That" engine to fix known problems and when necessary, escalate them to humans. Another tool, Detectify, continuously scans development, staging and production environments and provides notifications of vulnerabilities and remediation advice.

TOOLS REQUIRED AT EACH STAGE OF DEVOPS

- PLAN
- The plan phase is the least automated phase of DevSecOps, involving collaboration, discussion, review, and strategy of security analysis. Teams should perform a security analysis and create a plan that outlines where, how, and when security testing will be done.
- Some of the popular tools for this stage are IriusRisk and JIRA

TOOLS REQUIRED AT EACH STAGE OF DEVOPS

- Code
- DevSecOps tools for the code phase help developers write more secure code. Important code-phase security practices include static code analysis, code reviews, and pre-commit hooks.
- Some of popular security code tools are gerrit , fabricator ,find security bugs
-

TOOLS REQUIRED AT EACH STAGE OF DEVOPS

- The build phase begins once developers commit code to the source repository. DevSecOps build tools focus on automated security analysis against the build output artifact. Important security practices include software component analysis, static application software testing (SAST), and unit tests. Tools can be plugged into an existing CI/CD pipeline to automate these tests.
- Some well-known tools to execute build phase analysis include: OWASP Dependency-Check, SonarQube, SourceClear, Retire.js, Checkmarx, and Snyk.

TOOLS REQUIRED AT EACH STAGE OF DEVOPS

- Test
- The test phase is triggered after a build artifact is created and successfully deployed to staging or testing environments. A comprehensive test suite takes a considerable amount of time to execute. This phase should fail fast so that the more expensive test tasks are left for the end.
- The test phase uses dynamic application security testing (DAST) tools to detect live application flows like user authentication, authorization, SQL injection, and API-related endpoints. The security-focused DAST analyzes an application against a list of known high-severity issues, such as those listed in the OWASP Top 10.
- There are numerous open source and paid testing tools available, which offer a variety of functionality and support for language ecosystems, including BDD Automated Security Tests, JBroFuzz, Boofuzz, OWASP ZAP, Arachi, IBM AppScan, GAUNTLT, and SecApp suite.
-

TOOLS REQUIRED AT EACH STAGE OF DEVOPS

- Release
- By the release phase of the DevSecOps cycle, the application code and executable should already be thoroughly tested. The phase focuses on securing the runtime environment infrastructure by examining environment configuration values such as user access control, network firewall access, and secret data management.
- Configuration management tools are a key ingredient for security in the release phase, since they provide visibility into the static configuration of a dynamic infrastructure. The system configuration can then be audited and reviewed. The configuration becomes immutable, and can only be updated through commits to a configuration management repository. Some popular configuration management tools include Ansible, Puppet, HashiCorp Terraform, Chef, and Docker.

TOOLS REQUIRED AT EACH STAGE OF DEVOPS

- If the previous phases pass successfully, it's time to deploy the build artifact to production. The security areas of concern to address during the deploy phase are those that only happen against the live production system. For example, any differences in configuration between the production environment and the previous staging and development environments should be thoroughly reviewed. Production TLS and DRM certificates should be validated and reviewed for upcoming renewal.
- The deploy phase is a good time for runtime verification tools like Osquery, Falco, and Tripwire, which extract information from a running system in order to determine whether it performs as expected.

INTRODUCTION TO JENKINS

- Jenkins is a self-contained, open source automation server which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software.
- Jenkins can be installed through native system packages, Docker, or even run standalone by any machine with a Java Runtime Environment (JRE) installed.