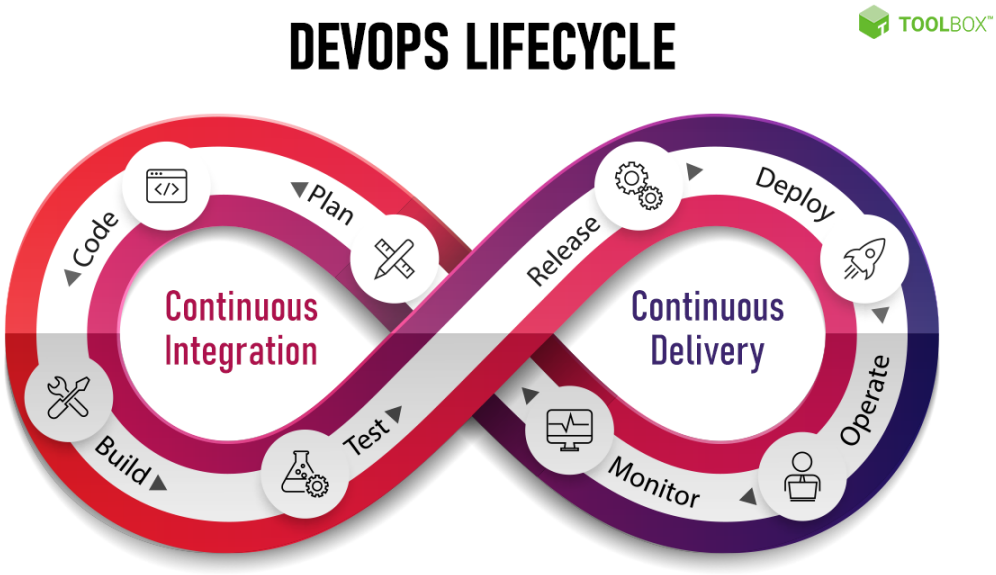
DevOps vs Agile

- DevOps and Agile are paving the way for advanced software development practices in the modern world. DevOps enables collaboration between development and operation teams, while the Agile methodology makes developers and development cycles more efficient. While there are many notable differences between Agile and DevOps, there also exist similarities

- Both Agile and DevOps have their own strengths and drawbacks. Both are highly reliable in their own ways, which is why it's best to combine them. The goal is to enhance overall productivity. Hence, teams should be able to take advantage of both approaches to work faster and more efficiently.

DevOps

DevOps is a software development method that seamlessly aligns development and IT operations in an organization to improve productivity and facilitate better collaboration between teams. DevOps is the integration of people, processes, practices, tools, and technology that enables code deployment in a robust and automated manner.



Unlike traditional software development practices, DevOps is an ongoing process of building, testing, deploying, and monitoring. Its main goal is to continuously deliver quality software properly. Incorporating DevOps has several noteworthy benefits, such as faster, reliable, and easily integrated deployments.

The principles of DevOps are based on six main ideologies: continuous integration, continuous delivery, continuous testing, continuous deployment, continuous operations, and continuous collaboration. Combining two distinct departments and processes (development and operations) and bringing them together leads to increased transparency and concentration on automated testing.

Jez Humble, the co-author of ‘The DevOps Handbook’, coined the acronym “CALMS” for the DevOps framework. It stands for culture, automation, lean, measurement, and sharing.

- **Culture** symbolizes the cultural shift that occurs in an organization when the development and operations teams work

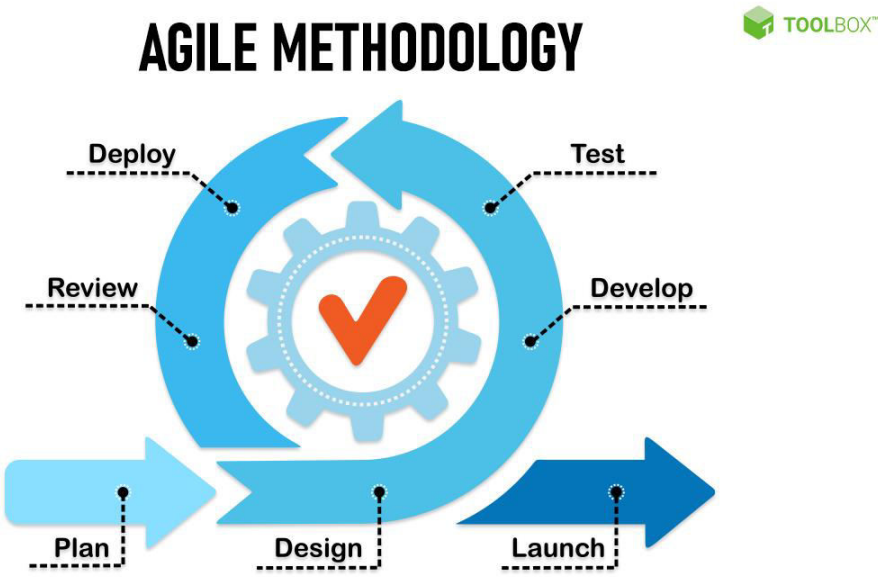
together in unison.

- **Automation** refers to replacing manual tasks with automation, which increases speed and promotes better product quality.
- DevOps takes inspiration from **lean** principles of continuous improvement and experimenting.
- **Measuring** is the process of gauging results to improve quality.
- **Sharing** refers to distributing responsibility amongst the development and operation teams to troubleshoot problems and find solutions efficiently.

Agile

The Agile methodology is a software development method that aligns itself with the principles of the Agile Manifesto. It is an iterative approach to software development and project management that focuses on collaboration between cross-functional teams to find solutions, feedback from customers to improve quality, and quick releases to meet deadlines.

Agile development breaks down processes into smaller units called '**sprints**' that typically **last from two weeks to a month**. The Agile methodology focuses on incremental deployments of each sprint and integrating them for final testing. It can be implemented using tools such as Kanban board, Scrum, XP, Active Collab, etc.



The four core values central to the Agile software development process:

1. Individuals and interactions over processes and tools

The Agile Manifesto states the importance of valuing people over processes. It has been a common practice among many software organizations to pay more attention to acquiring the best possible tools to build their software. However, even the greatest tools would fall flat in the hands of the wrong team. The key to success here is having the right team of individuals and encouraging them to communicate and collaborate to cultivate a healthy work environment and tackle problems as and when they arise.

2. Working software over comprehensive documentation

While documentation is important, it should not take center stage. In the past, it was a common practice among many software teams to spend hours documenting processes. The Agile Manifesto states that the highest priority should be given to delivering software to consumers rather than putting too much effort into documentation instead of writing code and actually building the software.

3. Customer collaboration over contract negotiation

Previously, contracts were a big deal, and a lot of consideration and effort was put toward writing them and negotiating the various terms and conditions. More often than not, this led to miscommunication, as the contract stated one thing, the product turned out to be another, and the customer's requirement was something else altogether. The Agile Manifesto states that continuous development and continuous collaboration with customers is the best way to ensure a project's success.

4. Responding to change over following a plan

We all live in a dynamic world where customers' priorities, needs, and preferences shift rapidly over a period of time. Hence, now, more than ever, it's crucial for software teams to have a flexible and adaptable approach. The Agile Manifesto is all about embracing and accepting mid-project changes for the betterment of the overall project. Agile teams have the superlative ability to change course midway instead of blindly following a rigid plan.

Key Similarities Between DevOps and Agile Methodology

The fast pace of technological advancements and ever-growing customer demands require organizations to continuously find newer approaches and solutions to problems. That being said, it becomes vital for progressive fields such as IT software development to adapt to changes and new situations.

Agile was conceived out of the shortcomings of the waterfall method that couldn't meet the demands of the modern world. Similarly, DevOps completes a missing piece of the Agile methodology and takes it a step ahead by deploying the software securely once it has been developed, something that Agile practices don't look after. This makes it safe to say that there are significant similarities between the Agile methodology and DevOps, as one is a thoughtful progression of the other.

1. Emphasis on testing & automation

DevOps and Agile are both focused on achieving stability. They do so by actively operating in a fast, secure, and quality-assessed environment. Both methodologies accomplish this through the integration of a great deal of testing on a routine basis. They are similar in their beliefs of relying on automation to bring more flexibility and security in the execution process

2. An inclination toward business productivity

Both DevOps and Agile work in tandem toward achieving one common goal for businesses — maximizing productivity. DevOps and Agile share a similar business-inclined approach. Implementing the Agile methodology in the

software development process gives teams the time to concentrate on a single target, leaving room for them to save time and be more productive. On the other hand, incorporating the DevOps culture leads to quicker releases and delivery without hampering business processes.

3. Paves the path for powerful partnerships

Even though DevOps and Agile may have a myriad of differences, both of them lay a huge amount of significance on building a collaborative workplace, where team members stay connected to share information, quickly detect issues, and troubleshoot problems easily and conveniently. There is a huge emphasis on interactions amongst team members and individuals to become more productive, efficient, and responsive to change.

4. Draws inspiration from lean philosophies

DevOps and Agile share a common ground in terms of deriving their approaches from the Lean philosophy. The Lean philosophy helps both DevOps and Agile standardize their communication process and facilitate smooth interactions amongst team members to create a healthy and productive work environment.

Key Differences Between DevOps and Agile Methodology

Parameter	DevOps	Agile
Guiding Philosophy	DevOps is a culture where the development and operations teams perform as a single unit and integrate their work to cultivate productivity and establish collaboration.	The Agile methodology is a practice that looks after continuously managing and delivering small, incremental changes of a project through iterative development and ongoing testing.
Focal Point & Purpose	DevOps emphasizes continuous testing and rapid deliveries made every few hours or even on an everyday basis. DevOps'	The Agile methodology primarily focuses on constant changes and incremental

	central purpose is to manage end-to-end engineering and business solutions and focus on fast delivery.	**deployments after each sprint. Agile is utilized in managing complicated projects and making room for mid-project changes.**
Delivery & Deployment	**Unlike Agile, DevOps uses pre-built software that is available and ready to release and looks after deploying it securely.**	**On the other hand, Agile builds and creates software and looks after launching it but plays no role in deploying the software.**
Team Size & Skills	**DevOps functions on the integration of different teams and hence comprises a larger group of people. DevOps requires people of different specializations and functional skill-sets to divide responsibility to achieve better results and success.**	**Agile focuses on working with a smaller team for faster execution and reduction of risk. Each team member working with the Agile methodology becomes highly skilled to perform any task and, in turn, becomes an all-round developer.**

Documentation	**Documentation is essential to DevOps. Documenting processes, updates, information, and communication in detail is crucial for assuring smooth team**	**One of Agile's core values highlights the importance of well-working software over comprehensive documentation. Keeping this in mind, documenting is kept light to encourage convenience and freedom**
Source of Feedback	**DevOps bridges the gaps and addresses concerns between a developer and IT operations. As such, feedback to improve speed and quality is received internally.**	**Agile bridges the gap and addresses concerns between a customer and the development & testing teams. As such, feedback is received from the customer, and adjustments are thereby made accordingly.**
Means of Communication & Interaction	**Communication in DevOps is carried out via specs and**	**The Scrum framework is widely**

	design documents.	**used in the Agile methodology. Communication is organized to discuss briefs and the status of various tasks in the form of daily scrum meetings, communication.**

DevOps Tools

DevOps is the next evolution of agile methodologies. A cultural shift that brings development and operations teams together. DevOps is a practice that involves a cultural change, new management principles, and technology tools that help to implement best practices.

When it comes to a DevOps toolchain, organizations should look for tools that improve collaboration, reduce context-switching, introduce automation, and leverage observability and monitoring to ship better software, faster.

There are two primary approaches to a DevOps toolchain: an all-in-one or open toolchain. An all-in-one DevOps solution provides a complete solution that usually doesn't integrate with other third-party tools. An open toolchain can be customized for a team's needs with different tools. Atlassian believes an open toolchain is the best approach since it can be customized with best-of-breed tools to the unique needs of an organization. Using this approach often leads to increased time efficiency and reduces time to market.

Regardless of the type of DevOps toolchain an organization uses, a DevOps process needs to use the right tools to address the key phases of the DevOps lifecycle:

- **Discover**
- **Plan**
- **Build**
- **Test**
- **Monitor**
- **Operate**
- **Continuous feedback**

With an open DevOps toolchain, the selected tools touch multiple phases of the DevOps lifecycle. The following sections showcase some of the most popular tools for DevOps, but given the nature of the market, this list changes frequently. Providers add new capabilities that enable them to span more phases of the DevOps lifecycle, new integrations are announced each quarter, and in some cases, providers consolidate their offerings to focus on a specific problem for their users.

Discover

- **Jira Product Discovery**
- **MURAL**
- **miro**

In the Discover phase, a DevOps team researches and defines the scope of a project. In particular, it involves activities such as user research, establishing goals, and defining success.

Tools like **Mural and Miro empower the entire software team to gather ideas and conduct research. Jira Product Discovery organizes this information into actionable inputs and prioritizes actions for development teams**. As you're prioritizing, you'll also need to keep your backlog of user feedback in mind.

Product discovery is the very first activity of a product design, which then becomes the baseline for decision making. During product discovery, you can collect all the crucial information about any user problems and then provide solutions for them.

We recommend looking for tools that encourage “asynchronous brainstorming”. It’s important that everyone can share and comment on anything: ideas, strategies, goals, requirements, roadmaps, and documentation.

Plan

- **jira software**
- **confluence**
- **slack**

in this phase of software development teams define the software requirements, establish project goals, and plan the overall software development process. Prime tasks are including defining user stories, prioritizing tasks, and creating a roadmap for the software development project.

Build

- **kubernetes**
- **docker**

developers use open source tools like Kubernetes and Docker to

provision individual development environments. Coding against virtual, disposable replicas of production helps you get more work done.

Infrastructure as code

- **ansible**
- **CHEF INFRA**
- **docker**
- **puppet**
- **terraform**

Developers create modular applications because they're more reliable and maintainable. So why not extend that thinking to IT infrastructure? This can be difficult to apply to systems because they are always changing. So we get around that by using code for provisioning.

Infrastructure as code means re-provisioning is faster than repairing – and more consistent and reproducible. It also means you can easily spin up variations of your development environment with similar configuration as production. Provisioning code can be applied and reapplied to put a server into a known baseline. It can be stored in version control. It can be tested, incorporated into CI (continuous integration), and peer-reviewed.

When institutional knowledge is, well, codified in code, the need for run books and internal documentation fades. What emerges are repeatable processes, and reliable systems.

Source control and collaborative coding

- **Bitbucket**
- **Github**
- **Gitlab**

It's important to have source control of your code. Source control tools help store the code in different chains so you can see every change and collaborate more easily by sharing those changes. Rather than waiting on change approval boards before deploying to production, you can improve code quality and throughput with peer reviews done via pull requests.

Pull requests tell your team about changes you've pushed to a development branch in your repository. Your team can then review the proposed changes and discuss modifications before integrating them into the main code line. Pull requests increase the quality of the software which results in less bugs / incidents, which reduces operational costs and results in faster development.

Source control tools should integrate with other tools, which allows you to connect the different parts of code development and delivery. This allows you to know if the feature's code is running in production. If an incident occurs, the code can be retrieved to shed light on the incident.

Continuous Delivery

- **jenkins**
- **aws**

- **Bitbucket**
- **circleci**
- **sonarsourse**

Continuous integration is the practice of checking in code to a shared repository several times a day, and testing it each time. That way, you automatically detect problems early, fix them when they're easiest to fix, and roll out new features to your users as early as possible.

Code review by pull-requests requires branching and is all the rage. The DevOps North Star is a workflow that results in fewer and faster branches and maintains testing rigor without sacrificing development speed.

Look for tools that automatically apply your tests to development branches, and give you the option to push to main when branch builds are successful. Along with that, you get continuous feedback through real-time chat alerts from your team with a simple integration.

Test

Automated testing

- **zephyr squad**
- **XRay**
- **mabl**

- **zephr scale**
- **snyk**
- **MEND**
- **VERACODE**
- **STACKHAWK**

Testing tools span many needs and capabilities, including exploratory testing, test management, and orchestration. However, for the DevOps toolchain, automation is an essential function. Automated testing pays off over time by speeding up your development and testing cycles in the long run. And in a DevOps environment, it's important for another reason: awareness.

Test automation can increase software quality and reduce risk by doing it early and often. Development teams can execute automated tests repeatedly, covering several areas such as UI testing, security scanning, or load testing. They also yield reports and trend graphs that help identify risky areas.

Risk is a fact of life in software development, but you can't mitigate what you can't anticipate. Do your operations team a favor and let them peek under the hood with you. Look for tools that support wallboards, and let everyone involved in the project comment on specific build or deployment results. Extra points for tools that make it easy to get Operations involved in blitz testing and exploratory testing.

Deploy

Deployment dashboards

- **Jira software**

One of the most stressful parts of shipping software is getting all the change, test, and deployment information for an upcoming release into one place. The last thing anyone needs before a release is a long meeting to report on status. This is where release dashboards come in.

Look for tools with a single dashboard integrated with your code repository and deployment tools. Find something that gives you full visibility on branches, builds, pull requests, and deployment warnings in one place.

Automated deployment

- **Bitbucket**
- **AWS CodePipeline**

There's no magic recipe for automated deployment that will work for every application and IT environment. But converting operations' runbook into a cmd-executable script using Ruby or bash is a common way to start. Good engineering practices are vital. Use variables to factor out host names – maintaining unique scripts or code for each environment .Create utility methods or scripts to avoid duplicated code. And peer review your scripts to sanity-check them.

Try automating deployments to your lowest-level environment first, where you'll be using that automation most frequently, then replicate that all the way up to production. If nothing else, this exercise highlights the differences between your environments and generates a list of tasks for standardizing them. As a bonus, standardizing deploys through automation reduces "server drift" within and between environments.

Operate

Incident, change and problem tracking

- **JiraService Management**
- **JiraSoftware**
- **Opsgenie**
- **Statuspage**

The keys to unlocking collaboration between DevOps teams is making sure they're viewing the same work. What happens when incidents are reported? Are they linked and traceable to software problems? When changes are made, are they linked to releases?

Nothing blocks Dev's collaboration with Ops more than having incidents and software development projects tracked in different systems. Look for tools that keep incidents, changes, problems, and software projects on one platform so you can identify and fix problems faster.

Observe

Application and server performance monitoring

- **APPDYNAMICS**
- **DATADOG**

- **Slack**
- **splunk**
- **New Relic**
- **Opsgenie**
- **pingdom**
- **Nagios**
- **dynatrace**
- **hostedgraphite**
- **sumo logic**

There are two types of monitoring that should be automated: server monitoring and application performance monitoring.

Manually “topping” a box or hitting your API with a test is fine for spot-checking. But to understand trends and the overall health of your application (and environments), you need software that is listening and recording data 24/7. Ongoing observability is a key capability for successful DevOps teams.

Look for tools that integrate with your group chat client so alerts go straight to your team’s room, or a dedicated room for incidents.

Continuous Feedback

- **GetFeedBack**

- **slack**
- **Jira Service Management**
- **pendo**

Customers are already telling you whether you've built the right thing – you just have to listen. Continuous feedback includes both the culture and processes to collect feedback regularly, and tools to drive insights from the feedback. Continuous feedback practices include collecting and reviewing NPS data, churn surveys, bug reports, support tickets, and even tweets. In a user comments because they help guide everything from release planning to exploratory testing sessions.

Look for applications that integrate your chat tool with your favorite survey platform for NPS-style feedback. Twitter and/or Facebook can also be integrated with chat for real-time feedback. For deeper looks at the feedback coming in from social media, it's worth investing in a social media management platform that can pull reports using historical data.

Analyzing and incorporating feedback may feel like it slows the pace of development in the short term, but it's more efficient in the long run than releasing new features that nobody wants.