1. What are the fundamental differences between DevOps & Agile?

The fundamental differences between DevOps & Agile are: -

* DevOps is a practice of bringing development and operations teams together whereas Agile is an iterative approach that focuses on collaboration, customer feedback and small rapid releases.
* DevOps focuses on constant testing and delivery while the Agile process focuses on constant changes.
* DevOps requires relatively a large team while Agile requires a small team.
* DevOps leverages both shifts left and right principles, on the other hand, Agile leverage shift-left principle.
* The target area of Agile is Software development whereas the Target area of DevOps is to give end-to-end business solutions and fast delivery.
* DevOps focuses more on operational and business readiness whereas Agile focuses on functional and non-function readiness.

2. What is the need for DevOps?

DevOps is essential because for five reasons listed below

* Shorter Development Cycles, Faster Innovation: - When we have a biased response from the development and operations teams, it is often difficult to tell if the application is operational. When development teams simply submit a request, the cycle times are unnecessarily extended. With joint development and operations efforts, the team's applications are ready to use more quickly. This is important because companies succeed on the basis of their ability to innovate faster than their competitors.
* Reduce Implementation Failure, Reflections and Recovery Time: - The main reason for the failure of the teams in the implementation failure is due to programming defects. With shorter development cycles, DevOps promotes frequent code versions. This, in turn, makes it easy to detect code defects. Therefore, teams can use their time to reduce the number of implementation failures using agile programming principles that require collaboration and standard programming. Recovery time is an important issue because you should expect some failure. But recovery is much faster when development teams and operations work together to share ideas and take into account the challenges of both teams during development.
* Better Communication and Cooperation: - Improved DevOps software development culture. The common teams are happier and more productive. Culture focuses on performance rather than individual goals. When teams trust each other, they can experiment and innovate more effectively. Teams can focus on bringing the product to market or production, and their key performance indicators must be organized accordingly. It no longer involves "passing" the application to the processes and waiting to see what is happening. Processes do not need to wait for a different team to solve a problem. The process becomes increasingly transparent as all individuals work towards a common goal.
* Greater Competencies:- High efficiency helps accelerate development and makes it less prone to errors. There are ways to automate DevOps tasks. Continuous integration servers automate the code testing process, reducing the amount of manual work required. This means that software engineers can focus on completing tasks that cannot be automated.
* Reduce Costs and IT Staff: - All the benefits of DevOps translate into reduced general costs and requirements of IT staff. DevOps development teams require IT staff to be 35 percent less and IT costs 30 percent lower.

3. What are the advantages of DevOps?

The advantages of DevOps are:-

* Reduce the implementation time of new services from months to minutes
* Increase productivity of business and IT teams
* Save costs on maintenance and upgrades, and eliminate unnecessary capital expenditure
* Standardize processes for easy replication and faster delivery
* Improve quality, reliability and reusability of all system components
* Increase the rate of success for digitalization strategies and transformation projects
* Ensure that money invested in cloud infrastructure, analytics and data management are not wasted

4. Explain with a use case where DevOps can be used in industry/ real-life.

DevOps is more than software development practice. It’s an umbrella term that denotes an application development culture. As the name implies, DevOps is a combination of development and operations. DevOps encompasses people, tools, and processes to enable better collaboration and automation in enterprise software development. It represents a change in IT culture, focusing on rapid IT service delivery through the adoption of agile, lean practices in the context of a system-oriented approach. DevOps emphasizes people (and culture) and seeks to improve collaboration between operations and development teams. DevOps implementations utilize technology—especially automation tools that can leverage an increasingly programmable and dynamic infrastructure from a lifecycle perspective.

Below is the use case example: -

EasySales is a peer-to-peer e-commerce website focused on handmade or vintage items and supplies, as well as unique factory-manufactured items. EasySales struggled with slow, painful site updates that frequently caused the site to go down. It affected sales for millions of EasySales users who sold goods through an online marketplace and risked driving them to the competitor.

With the help of a new technical management team, EasySales transitioned from its waterfall model, which produced four-hour full-site deployments twice weekly, to a more agile approach. Today, it has a fully automated deployment pipeline, and its continuous delivery practices have reportedly resulted in more than 50 deployments a day with fewer disruptions.

5. What are the success factors for Continuous Integration?

The success factors for Continuous Integration are:-

* Smaller Code Changes: - One technical advantage of continuous integration and continuous delivery is that it allows you to integrate small pieces of code at one time. These code changes are simpler and easier to handle than huge chunks of code and as such, have fewer issues that may need to be repaired at a later date. Using continuous testing, these small pieces can be tested as soon as they are integrated into the code repository, allowing developers to recognize a problem before too much work is completed afterward. This works really well for large development teams who work remotely as well as those in-house as communication between team members can be challenging.
* Fault Isolations: - Fault isolation refers to the practice of designing systems such that when an error occurs, the negative outcomes are limited in scope. Limiting the scope of problems reduces the potential for damage and makes systems easier to maintain. Designing your system with CI/CD ensures that fault isolations are faster to detect and easier to implement. Fault isolations combine monitoring the system, identifying when the fault occurred, and triggering its location. Thus, the consequences of bugs appearing in the application are limited in scope. Sudden breakdowns and other critical issues can be prevented from occurring with the ability to isolate the problem before it can cause damage to the entire system.
* Faster Mean Time To Resolution (MTTR): - MTTR measures the maintainability of repairable features and sets the average time to repair a broken feature. Basically, it helps you track the amount of time spent to recover from a failure. CI/CD reduces the MTTR because the code changes are smaller and fault isolations are easier to detect. One of the most important business risk assurances is to keep failures to a minimum and quickly recover from any failures that do happen. Application monitoring tools are a great way to find and fix failures while also logging the problems to notice trends faster.
* More Test Reliability: - Using CI/CD, test reliability improves due to the bite-size and specific changes introduced to the system, allowing for more accurate positive and negative tests to be conducted. Test reliability within CI/CD can also be considered Continuous Reliability. With the continuous merging and releasing of new products and features, knowing that quality was top of mind throughout the entire process assures stakeholders their investment is worthwhile.
* Faster Release Rate: -Failures are detected faster and as such, can be repaired faster, leading to increasing release rates. However, frequent releases are possible only if the code is developed in a continuously moving system. CI/CD continuously merges codes and continuously deploys them to production after thorough testing, keeping the code in a release-ready state. It’s important to have as part of deployment a production environment set up that closely mimics that which end-users will ultimately be using. Containerization is a great method to test the code in a production environment to test only the area that will be affected by the release.
* Smaller Backlog: - Incorporating CI/CD into your organization’s development process reduces the number of non-critical defects in your backlog. These small defects are detected prior to production and fixed before being released to end-users. The benefits of solving non-critical issues ahead-of-time are many. For example, your developers have more time to focus on larger problems or improving the system and your testers can focus less on small problems so they can find larger problems before being released. Another benefit (and perhaps the best one) is keeping your customers happy by preventing them from finding many errors in your product.
* Customer Satisfaction: - The advantages of CI/CD do not only fall into the technical aspect but also in an organization scope. The first few moments of a new customer trying out your product is a make-or-break-it moment.
* Increase Team Transparency and Accountability: - CI/CD is a great way to get continuous feedback not only from your customers but also from your own team. This increases the transparency of any problems in the team and encourages responsible accountability. CI is mostly focused on the development team, so the feedback from this part of the pipeline affects build failures, merging problems, architectural setbacks, etc. CD focuses more on getting the product quickly to the end-users to get the much-needed customer feedback. Both CI and CD provide rapid feedback, allowing you to steadily and continuously make your product even better.
* Reduce Costs: - Automation in the CI/CD pipeline reduces the number of errors that can take place in the many repetitive steps of CI and CD. Doing so also frees up developer time that could be spent on product development as there aren’t as many code changes to fix down the road if the error is caught quickly. Another thing to keep in mind: increasing code quality with automation also increases your ROI.
* Easy Maintenance and Updates: - Maintenance and updates are a crucial part of making a great product. However, it’s important to note within a CI/CD process to perform maintenance during downtime periods, also known as the non-critical hour. Don’t take the system down during peak traffic times to update code changes. Upsetting customers is one part of the problem but trying to update changes during this time could also increase deployment issues. Make sure the pipeline runs smoothly by incorporating when to make changes and releases. A great way to ensure maintenance doesn’t affect the entire system is to create microservices in your code architecture so that only one area of the system is taken down at one time.

6. What are the differences between continuous integration, continuous delivery, and continuous deployment?

The differences between continuous integration, continuous delivery, and continuous deployment are:-

* Continuous integration refers to the software development practice of regularly integrating new code into the existing code base. This can be accomplished with automated tools that include the integration and testing of the code base. Often the new code is integrated into the base daily with automated tests ensuring that the new code works as expected and does not adversely impact the rest of the system.
* Continuous delivery is sometimes used to represent the end-to-end chain of regularly delivering code to production. Others state that continuous delivery means that you have the ability to deliver to production, but the actual deployment is a manual step.
* In the context in which the deployment of code to production is manual, continuous deployment represents automation of that last step. Continuous deployment creates the opportunity to deploy new code into production several times a day.

7. What role does the Quality Assurance (QA) team play in DevOps?

The roles played by QA team in DevOps are :-

* QA should aim to detect a bug at the earliest point in the cycle and also prevent potential bugs from reappearing in the production environment.
* QA should not limit their role to only finding and preventing defects, but should also highlight problems in the processes and recommend changes wherever necessary. This practice is only possible if QA takes ownership of the improvement process in DevOps.
* QA should make sure that all environments required for testing should be standardized and deployments automated. There is no room for manual testing.
* Being a quality advocate, QA has influence on both development and operations. They should not just only find defects but also notice any opportunity to improve the quality of the product. In this way, quality will be checked at every stage.

8. Describe an efficient workflow for continuous integration.

The steps for efficient workflow for continuous integration are stated below: -

* Developers check out code into their private workspaces. They then make their changes and test them locally.
* When done, developers check in their changes into the source control repository.
* The CI server monitors the source control repository and when it detects a change it triggers a build of the relevant sources.
* After a successful build, the CI server performs some or all of the following activities:
* makes deployable artefacts available for testing
* assigns a build label to the version of the code that was just built
* notifies the relevant team members that a successful build occurred
* triggers unit and integration testing
* At this point, the changes that were checked in at step 2 have been successfully built and a build label has been applied to the source code that was used for the build, meaning that the build could be recreated if necessary. In the event of a build failure, the CI server sends notifications to the relevant developers who restart the process from step 1 to make the changes necessary to resolve the build errors.
* After the unit and integration testing has taken place, the relevant team members are notified of the test results.
* At this point, the changes that were checked in at step 2 have been successfully built and tested, all with little or no manual intervention.

9. What are the best practices for DevOps implementation?

The best practices for DevOps implementation are:-

* Test Automation
* Integrated Configuration Management
* Integrated Change Management
* Continuous Integration
* Continuous Deployment
* Application Monitoring
* Automated Dashboards

10. How will you approach when a project needs to implement DevOps?

DevOps implementation can be done by following steps: -

* Evaluate the need to implement DevOps practice
* Break the organizational silos & encourage collaboration
* Put Customer / end-user satisfaction at the centre
* Don’t jump start, instead, start small and then scale up
* Automate wherever possible
* Select tools that are compatible with each other
* Define performance reviews for team and an individual
* Ensure real-time visibility into the project
* Integrate and deliver continuously
* Achieve better results with monitoring & feedback