DevOps-UNIT-II

Software development models and DevOps: DevOps Lifecycle for Business Agility, DevOps, and Continuous Testing. DevOps influence on Architecture: Introducing software architecture, The monolithic scenario, Architecture rules of thumb, The separation of concerns, Handling database migrations, Microservices, and the data tier, DevOps, architecture, and resilience.

DEVOPS LIFE CYCLE

DevOps lifecycle is a series of automated development processes or workflows within an iterative development lifecycle. It follows a continuous approach; hence its lifecycle is symbolized in the form of an infinity loop. This loop depicts the collaborative and iterative approach throughout the application lifecycle, consisting of tools and technology stacks for each stage. The left part deals with software development and testing. And in contrast, the right side of the infinity loop represents the deployment and operations cycle.

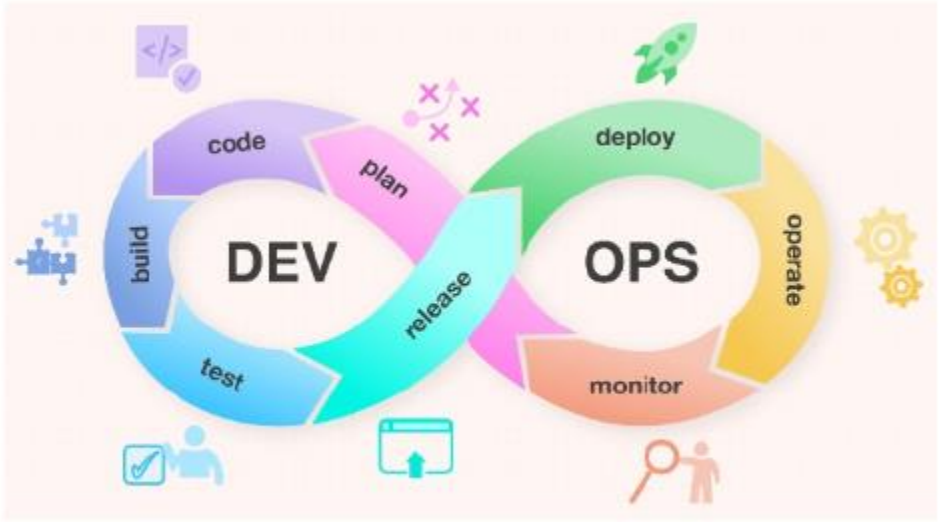

Fig: DevOps Life Cycle

Let's briefly overview how the DevOps lifecycle works at every stage.

1. **Plan:** In this stage, teams identify the business requirement and collect end-user feedback. They create a project roadmap to maximize the business value and deliver the desired product during this stage.

2. **Code:** The **code development** takes place at this stage. The development teams use some tools and plugins like *Git* to streamline the development process, which helps them avoid security flaws and lousy coding practices.
3. **Build:** In this stage, once developers finish their task, they commit the code to the shared code repository using build tools like Maven and Gradle.
4. **Test:** Once the build is ready, it is deployed to the test environment first to perform several types of testing like user acceptance test, security test, integration testing, performance testing, etc., using tools like JUnit, Selenium, etc., to ensure software quality.
5. **Release:** The build is ready to deploy on the production environment at this phase. Once the build passes all tests, the operations team schedules the releases or deploys multiple releases to production, depending on the organizational needs.
6. **Deploy:** In this stage, Infrastructure-as-Code helps build the production environment and then releases the build with the help of different tools.
7. **Operate:** The release is live now to use by customers. The operations team at this stage takes care of server configuring and provisioning using tools like Chef.
8. **Monitor:** In this stage, the DevOps pipeline is monitored based on data collected from customer behavior, application performance, etc. Monitoring the entire environment helps teams find the bottlenecks impacting the development and operations teams' productivity.

DevOps Importance and Benefits:

1. Speed
2. Time to Market
3. Reliability
4. Scale
5. Improved Collaboration
6. Contribution to Quality Assurance
7. Contribution to Services
8. Contribution to Information System Development

1. **Speed:** Accelerated delivery allows businesses to align to changes in market and to grow more efficiently while driving business results by innovating faster for end customer
2. **Time to Market:** It's all about reduction in time taken to deliver changes in the product to customers. This also demands increase in release frequency and pace of releases to incorporate innovations and improvements to product. This enables businesses to respond faster to market needs. CI and CR are two practices to automate Software development and release process.
3. **Reliability**: Quality of changes to software includes:
   - ✓ Application Update
   - ✓ Modification to Infrastructure
   - ✓ Define reliability from end user experience perspective.

   DevOps adoption makes delivery process repeatable, thereby making delivery more predictable by increasing probability of successful delivery.
4. **Scale**: Automation and Optimum development plus delivery process make it possible to scale the size of functional releases on demand. Agile and DevOps adoption ensures increase in delivery without impacting the delivery quality and timelines.
6. **Contribution to Quality Assurance**: Devops significantly affects quality assurance in information systems which integrates development , operations and support team with customer. By bringing these parties closer using tools

and improving cooperation the QA process becomes more predictable. Devops process increases the day to day data gathering process which helps in enhanced delivery quality.

7. **Contribution to Services**: DevOps principles can significantly affect output of implementation of service management frameworks. Service management framework rely on the extent of cooperation between development and operation teams. Business models can aligned to service delivery models such as SaaS. Organizations can offer services to others which opens new business opportunities.
8. **Contribution to Information System Development**: Devops brought major changes in the process of development of information systems. It removed communication barriers and gaps between teams and end user. CICD make it possible for end user to validate the software more frequently than earlier. CICD made it possible to refine product to match exact requirements.

7 C'S OF DEVOPS LIFE CYCLE FOR BUSINESS AGILITY

Everything is continuous in DevOps – from planning to monitoring. So let's break down the entire lifecycle into seven phases where continuity is at its core. Any phase in the lifecycle can iterate throughout the projects multiple times until it's finished.

The 7Cs of the DevOps lifecycle are :

Fig: The 7Cs of the DevOps lifecycle

1. Continuous development

This phase plays a pivotal role in delineating the vision for the entire software development cycle. It primarily focuses on project planning and coding. During this phase, project requirements are gathered and discussed with stakeholders. Moreover, the product backlog is also maintained based on customer feedback which is broken down into smaller releases and milestones for continuous software development.

Once the team agrees upon the business needs, the development team starts coding for the desired requirements. It's a continuous process where developers are required to code whenever any changes occur in the project requirement or in case of any performance issues.

Tools Used: There are no specific tools for planning, but the development team requires some tools for code maintenance. GitLab, GIT, TFS, SVN, Mercurial, Jira, BitBucket, Confluence, and Subversion are a few tools used for version control. Many companies prefer agile practices for collaboration and use Scrum, Lean, and Kanban. Among all those tools, GIT and Jira are the most popular ones used for

complex projects and the outstanding collaboration between teams while developing.

2. Continuous integration

Continuous integration is the most crucial phase in the entire DevOps lifecycle. In this phase, **updated code** or add-on functionalities and features are developed and integrated into existing code. Furthermore, bugs are detected and identified in the code during this phase at every step through **unit testing**, and then the source code is modified accordingly. This step makes integration a continuous approach where code is tested at **every commit**. Moreover, the tests needed are also planned in this phase.

3. Continuous testing

Quality analysts continuously test the software for bugs and issues during this stage using Docker containers. In case of a bug or an error, the code is sent back to the integration phase for modification. Automation testing also reduces the time and effort to deliver quality results. Teams use tools like *Selenium* at this stage. Moreover, continuous testing enhances the test evaluation report and minimizes the provisioning and maintenance cost of the test environments.

Tools Used: JUnit, Selenium, TestNG, and TestSigma are a few DevOps tools for continuous testing. Selenium is the most popular open-source automation testing tool that supports multiple platforms and browsers. TestSigma, on the other hand, is a unified AI-driven test automation platform that eliminates the technical complexity of test automation through **artificial intelligence**.

4. Continuous deployment

This phase is the crucial and most active one in the DevOps lifecycle, where final code is deployed on production servers. The continuous deployment includes

configuration management to make the deployment of code on servers accurate and smooth. Development teams release the code to servers and schedule the updates for servers, keeping the configurations consistent throughout the production process. Containerization tools also help in the deployment process by providing consistency across development, testing, production, and **staging environments**. This practice made the continuous delivery of new features in production possible.

Tools Used: Ansible, Puppet, and Chef are the configuration management tools that make the deployment process smooth and consistent throughout the production process. Docker and Vagrant are another DevOps tool used widely for handling the scalability of the continuous deployment process.

5. Continuous feedback

Continuous feedback came into existence to analyze and improve the application code. During this phase, customer behavior is evaluated regularly on each release to improve future releases and deployments. Businesses can either opt for a structural or unstructured approach to gather feedback. In the structural approach, feedback is collected through surveys and questionnaires. In contrast, the feedback is received through social media platforms in an unstructured approach. Overall, this phase is quintessential in making continuous delivery possible to introduce a **better version** of the application.

Tools Used: Pendo is a product analytics tool used to collect customer reviews and insights. Qentelli's TED is another tool used primarily for tracking the entire DevOps process to gather actionable insights for bugs and flaws.

6. Continuous monitoring

During this phase, the application's functionality and features are monitored continuously to detect **system errors** such as low memory, non-reachable server, etc. This process helps the IT team quickly **identify issues** related to app performance and the **root cause** behind it. If IT teams find any critical issue, the application goes through the entire DevOps cycle again to find the solution.

However, the security issues can be detected and resolved automatically during this phase.

Tools Used: Nagios, Kibana, Splunk, PagerDuty, ELK Stack, New Relic, and Sensu are a few DevOps tools used to make the continuous monitoring process fast and straightforward.

7. Continuous operations

The last phase in the DevOps lifecycle is crucial for reducing the planned downtime, such as scheduled maintenance. Generally, developers are required to take the server offline to make the updates, which increases the downtime and might even cost a significant loss to the company. Eventually, continuous operation automates the process of launching the app and its updates. It uses container management systems like Kubernetes and Docker to eliminate downtime. These container management tools help simplify the process of building, testing, and deploying the application on multiple environments. The key objective of this phase is to boost the application's uptime to ensure uninterrupted services. Through continuous operations, developers **save time** that can be used to accelerate the application's time-to-market.

Tools Used: Kubernetes and Docker Swarm are the container orchestration tools used for the high availability of the application and to make the deployment faster.

DEVOPS, AND CONTINUOUS TESTING

Continuous Testing refers to the execution of automated tests that are carried out at regular intervals every time code changes are made. These tests are conducted as a part of the software delivery pipeline to drive faster feedback on recent changes pushed to the code repository. Continuous Testing was introduced initially with the intention of reducing the time taken to provide feedback to developers.

The primary goal here is to test more often, particularly at an individual level in the early stages of development, and then testing the unified codebase as a whole. Continuous testing is an integral part of the continuous integration with Agile and DevOps pipeline. The process of Continuous Integration and delivery requires ***Continuous Testing***.

How does Continuous Testing play a vital role in DevOps?

It's evident that every software application is built uniquely, and it needs to be updated regularly to meet end-user requirements. Earlier, as the development and deployment process was rigid, changing and deploying features required a considerable amount of time. This is because projects earlier had definite timelines for development and QA phases, and the codebase was transferred between teams.

However, with the Agile approach becoming mainstream, making changes even in real-time has become more convenient, primarily due to Continuous Testing and the CI/CD pipeline. This is because the code is continually moving from **Development -> Testing -> Deployment Stages.**

With an Agile mindset being at the core, Continuous Testing in DevOps helps teams to explore critical bugs in the initial stages itself. This ensures that the risk of critical bugs is mitigated beforehand, saving the cost of bug fixing in later stages.

Continuous Testing encourages automating tests wherever possible throughout the development cycle. Doing so ensures that teams evaluate the code validity and overall quality of the software at each stage. These insights can help organizations identify whether the software is ready to go through the delivery pipeline.

The benefits of Continuous Testing in DevOps are:

- Early discovery of critical bugs
- Seamless collaboration among developers, QA and Operations team
- Helps to assess the quality of software developed at each stage
- Can be seamlessly incorporated into DevOps
- Helps drive faster test results which leads to improved code quality
- Repeated testing ensures minimal failure rate for new releases
- Faster time to market with a viable product and continuous feedback mechanism

What is Monolithic Architecture?

A monolithic architecture is a single-tiered, traditional, unified architecture for designing a software application. In context, monolithic refers to "composed all in one", "unable to change", and "too large". Monolithic applications are self-contained, complex, and tightly coupled. It is simple to develop, deploy, test, and scale horizontally.

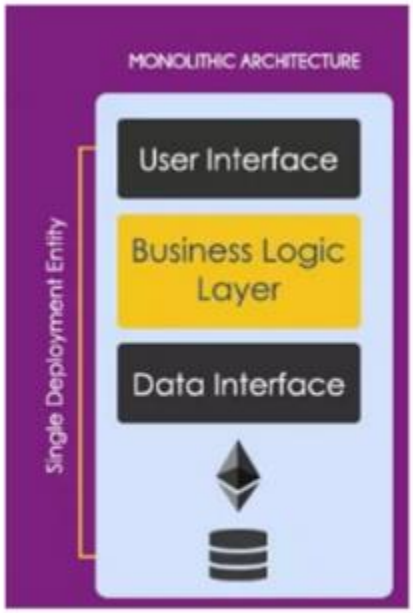

Fig: Monolithic

Monolithic architecture can be used in projects which do not require a real-time response and they can withstand downtime. These kinds of projects are limited to a certain size; if the web application's size is expected to go beyond that then Microservice Architecture should be adopted. Most of the web applications that we normally use are based on monolithic architecture.

However, maintaining such a large application is difficult and its size can slow down the time to start up. Even a small update, requires redeploying the entire application, and scaling modules with conflicting resource requirements can also become challenging. Not only that it is unreliable and can have difficulty in advancing and adopting new tools and technologies.

Web Applications usually consists of three parts

1. Front End client-side application is written in JavaScript and other languages.
2. Back End Server-side application which contains business logic written in java, PHP, ***Python*** or some other language.

3. Database of whole web application

All of these parts are closely coupled and frequently communicate with each other. Hence the whole web application works as a monolith where every part is dependent on others.

Advantages of Monolithic Architecture

The code structure of monolithic architecture is small as compared to microservices architecture. Hence the Monolithic architecture-based web applications are easy to develop, easy to test, easy to deploy and easy to scale.

Disadvantages of Monolithic Architecture

- The complexity in Monolithic Architecture increases too much with bigger size which makes this approach limited to a certain size of projects.
- The increase in the size of the web application increases startup time.
- Bigger web applications become more complex and consequences in reduced code readability, difficulty in development and debugging.

- Changes are one section of the code can cause an unanticipated impact on the rest of the code.
- Extensive testing and debugging are required for integrating new code.
- Continuous Integration and continuous deployment become difficult.
- In case a part of the web application shuts down then the rest of the web application will go down as well.

DEVOPS INFLUENCE ON ARCHITECTURE

The architecture of a system describes its major components, their relationships (structuresfi, and how they interact with each other. Architecture serves as a blueprint for a system. DevOps affects the architecture of our applications, while devops teams seek productivity through automation; also seek architecture agility in production environments. This architecture agility made us to go from monolithic architecture to micro service architecture.

Architecture rules of thumb

There are a number of architecture rules that might help us understand how to deal with the traditional undesirable situation.

- **The separation of concerns**: The fundamental principle is simply that we should consider different aspects of a system separately.

- **The principle of cohesion**: In computer science, cohesion refers to the degree to which the elements of a software module belong together. Cohesion can be used as a measure of how strongly related the functions in a module are. It is desirable to have strong cohesion in a module. We can see that strong cohesion is another aspect of the principle of the separation of concerns.
- **Coupling**: Coupling refers to the degree of dependency between two modules. We always want low coupling between modules. Again, we can see coupling as another aspect of the principle of the separation of concerns.

In this figure, we can see the separation of concerns idea in action:

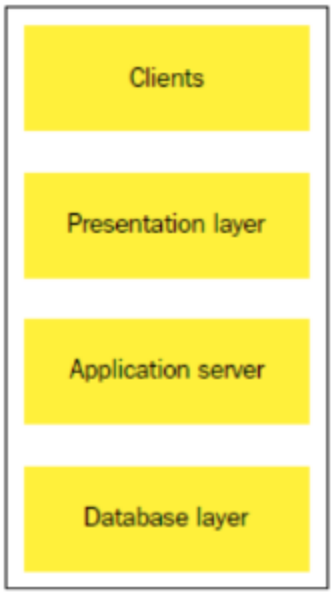

Fig: Software architecture pattern with three- tier

- **The presentation tier:** The presentation tier will be a web frontend implemented using the React web framework. It will be deployed as a set of JavaScript and static HTML files.
- **The logic tier:** The logic tier is a backend implemented using the Clojure language on the Java platform. The Java platform is very common in large organizations, while smaller organizations might prefer other platforms based on Ruby or Python.
- **The data tier:** In our case, the database is implemented with the PostgreSQL database system. PostgreSQL is a relational database management system. While arguably not as common as MySQL installations, larger enterprises might prefer Oracle databases.

From a DevOps point of view, the three-tier pattern looks compelling, at least superficially. It should be possible to deploy changes to each of the three layers

separately, which would make it simple to propagate small changes through the servers.

Handling database migrations

- It is the process of moving data from one or more database to another target database.
- There are several reasons for migrating from one database to another.
- For example, a business might want to save resources by switching to a cloud-based database.
- Similarly, another organization could move because they find a particular database suitable for their unique business needs.

Fig: Database Migration

Why Data Base Migration?

- The data ecosystem of an enterprise comprises a variety of applications. Over time, a business may migrate from an existing database to save costs,

✓ To enhance reliability,

✓ To achieve scalability, or

✓ To improve Security

✓ To improve Performance

✓ To reduce the cost.

- Even though they are essential, data migration projects can be very complex. Data migration requires downtime, which may lead to interruption in data management operations.
- This is why it is important to understand DB migration's risks and best practices and the tools that can help perform a smooth process.

Database Migration Challenges

Data Base Migration is a complex process, Some key challenges companies encounter while migrating their data include:

- **Data Loss**: The most common issue firms face data loss during the DB migration.
- **Data Security**: Data is a business's most valuable asset. Therefore, its security is of utmost importance. Before the DB migration process occurs, data encryption should be a top priority.

- **Difficulty during planning**: Large companies usually have disparate databases in different departments of the companies. During the planning stage of database migration, locating these databases and planning how to convert all schemas and normalize data is a common challenge.

- **Migration strategy**: A common question asked is how to do DB migration. Companies miss out on some crucial aspects and use a database migration strategy that is not suitable for their company. Therefore, It is necessary to conduct ample research before DB migration occurs

How To Do Database Migrations

- DB migration is a multi-step process that starts with assessing the source system and finishes at testing the migration design and replicating it to the product build.

- It is essential to have an appropriate database migration strategy and the right DB migration tools to make the process more efficient.

Let's take a look at the different steps to understand how to do database migration:

- **Understanding the Source Database**: A vital data migration step to understand is the source data that will populate your target database before starting any database migration project.
  - ✓ **What is the size of the source database?** The size and complexity of the database you are trying to migrate will determine the scope of your

migration project. This will also determine the time and computing resources required to transfer the data.

- ✓ **Does the database contain 'large' tables?'** If your source database contains tables that have millions of rows, you might want to use a tool with the capability to load data in parallel.
- ✓ **What kind of data types will be involved?** If you migrate data between different databases, such as an SQL database to an Oracle one, you will need schema conversion capabilities to successfully execute your DB migration project.

- **Assessing the Data**: This step involves a more granular assessment of the data you want to migrate. We would like to profile our source data and define data quality rules to remove inconsistencies, duplicate values, or incorrect information. Data profiling at an early stage of migration will help you mitigate the risk of delays, budget overruns, and even complete failures. We will also be able to define data quality rules to validate your data and improve its quality and accuracy, resulting in efficient DB migration.

- **Converting Database Schema**: Heterogeneous migrations involving migration between different database engines are relatively more complex than homogenous migrations. While schemas for heterogeneous database migrations can be converted manually, it is often very resource-intensive and time-consuming. Therefore, using a data migration tool with database schema migration conversion capability can help expedite the process and migrate data to the new database.
- **Testing the Migration Build**: It's a good idea to adopt an iterative approach to testing a migration build. You can start with a small subset of your data, profile it, and convert its schema instead of running a full migration exercise at once. This will help you ensure that all mappings, transformations, and data quality rules are working as intended. Once you have tested a subset on your database migration tool, you can increase the data volume gradually and build a single workflow.

- **Executing the Migration**: Most companies plan migration projects for when they can afford downtimes, e.g., on weekends or a public holiday. That said, it is now more important than ever before to plan DB migrations to minimize or outright ,eliminate interruptions to everyday data management processes.

ROLLING UPGRADES

Another thing to consider when doing database migrations is what can be referred to as rolling upgrades. These kinds of deployments are common when you don't want your end user to experience any downtime, or at least very little downtime.

Here is an example of a rolling upgrade for our organization's customer database.

When we start, we have a running system with one database and two servers. We have a load balancer in front of the two servers. We are going to roll out a change to the database schema, which also affects the servers. We are going to split the customer name field in the database into two separate fields, first name and surname. This is an incompatible change. How do we minimize downtimeffi Let's look at the solution:

1. We start out by doing a database migration that creates the two new name fields and then fills these new fields by taking the old name field and splitting the field into two halves by finding a space character in the name. This was the initial chosen encoding for names, which wasn't stellar. This is why we want to change it. This change is so far backward compatible, because we didn't remove the name field; we just created two new fields that are, so far, unused.

2. Next, we change the load balancer configuration so that the second of our two servers is no longer accessible from the outside world. The first server chugs along happily, because the old name field is still accessible to the old server code.

3. Now we are free to upgrade server two, since nobody uses it. After the upgrade, we start it, and it is also happy because it uses the two new database fields.

4. At this point, we can again switch the load balancer configuration such that server one is not available, and server two is brought online instead. We do the same kind of upgrade on server one while it is offline. We start it and now make both servers accessible again by reinstating our original load balancer configuration.

Now, the change is deployed almost completely. The only thing remaining is removing the old name field, since no server code uses it anymore.

What is the Microservice Architecture?

Microservices architecture arises as a solution to several problems developers had to face in Monolithic architecture. It gained popularity with the emergence of cloud computing, Agile development methods, virtualization, and DevOps.

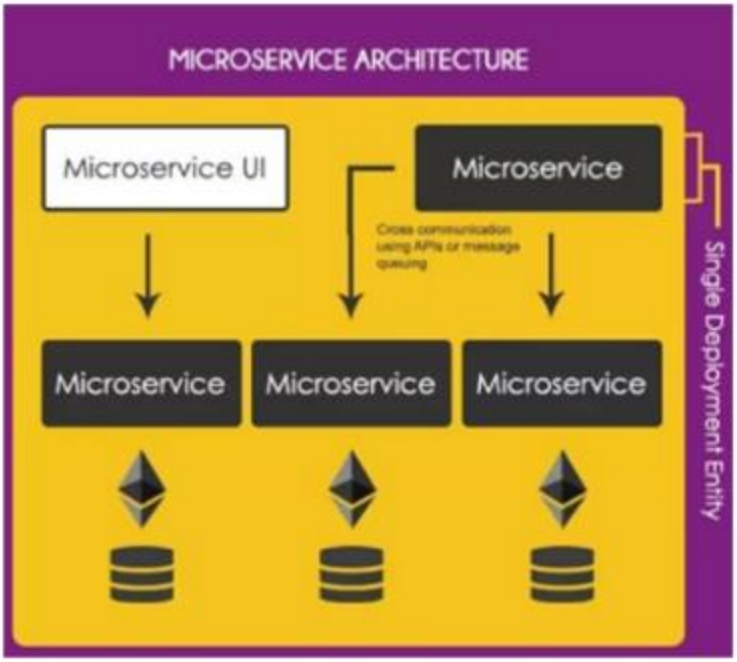

Fig: Micro service Architecture

Microservice architecture is an approach to structuring an application as a collection of services that are loosely coupled, independently deployable, highly testable and maintainable, owned by a small team, and organized around one's business needs. It allows developers to make rapid, reliable, and frequent delivery of complex and large applications. Moreover, it allows developers to adapt and evolve according to the new technology stack.

In Microservice Architecture big web applications are divided into smaller services, where each web service is responsible for executing a particular set of functionalities. If we take the example of a gigantic eCommerce platform then that Web application can be divided into smaller applications as following:

- Product searching application
- Inventory Management application
- Product selection & Shopping application
- Payment Application
- Allocating delivery to logistics partner
- The upselling application that uses data analytics and machine learning.

Microservice Architecture should be used only in big projects which require either real-time responses or the projects which cannot withstand downtime and can have abnormal spikes in users at a point of time. Projects on which

telecommunication, television networks, Ridesharing apps, food delivery apps and gigantic eCommerce platforms run are usually based on Microservice Architecture. The execution of Microservice architecture is complex and requires extensive planning.

Advantages of Microservice Architecture

- The big web applications when broken down into smaller services increase code readability.
- The smaller services are easy to develop, integrate, test and debug.
- The smaller services can be independently developed by different teams with their choice of the technology stack.
- Easier Continuous Integration and continuous deployment.
- The unanticipated impacts on code integration are drastically reduced.
- Lesser bugs and resource conflicts will be created on code integration. Hence the Cost of testing and debugging reduces.
- Since the code being used frequently can be deployed on function as a service platform, the service will offer almost zero downtime with the agility to handle abnormal spikes in requests.
- The distributed deployment eliminates the need of buying expensive server hosting and resources.

Disadvantages of Microservice Architecture

- To build Microservice Architecture based web applications highly experienced and expensive resources are required.
- Building Microservices based web applications requires web developers, cloud architects, DevOps engineer, Quality Analyst, Project managers, Business Analysts, Product Managers and lots of other team members that are defined in a scaled agile framework.
- Microservice Architecture adds the complexity of distributed systems. The web developers have to write the code to handle partial code failure, service failure and discrepancies occurred on using distributed systems.
- Overhead of Implementing an inter-process communication mechanism based on either messaging or RPC.
- Testing microservices is typical because in order to test a service that is dependent on other services the developer and the quality analyst will have to run all the base services first.
- Nowadays Kubernetes and Dockers are being used to host the services.
- Every service in Microservice architecture-based web applications uses a different database. Therefore, there is an overhead for developers to reflect the changes in common columns of different databases.
- Making changes in services can become really tough when service is widely being used among other services.

DevOps, architecture, and resilience:

a DevOps point of view. An important goal of DevOps is to place new features in the hands of our user faster. This is a consequence of the greater amount of modularization that microservices provide. Microservices do offer challenges of their own.

We want to be able to deploy new code quickly, but we also want our software to be reliable. Microservices have more integration points between systems and suffer from a higher possibility of failure than monolithic systems.

Automated testing is very important with DevOps so that the changes we deploy are of good quality and can be relied upon. This is, however, not a solution to the problem of services that suddenly stop working for other reasons. Since we have more running services with the microservice pattern, it is statistically more likely fora service to fail.

We can partially mitigate this problem by making an effort to monitor the services and take appropriate action when something fails. This should preferably be automated. In our customer database example, we can employ the following strategy:

- We use two application servers that both run our application
- The application offers a special monitoring interface via JsonRest
- A monitoring daemon periodically polls this monitoring interface
- If a server stops working, the load balancer is reconfigured such that the offending server is taken out of the server pool

This is obviously a simple example, but it serves to illustrate the challenges we face when designing resilient systems that comprise many moving parts and how they might affect architectural decisions.

Important Questions:

1. Explain how devops life cycle affects business agilityffi
2. What is Monolithic architecture explain its pros and consffi
3. What is data base migration and describe different type’s migration issuesffi
4. Explain in detail about Micro servicesffi
5. Explain about rolling updatesffi