What is Devops ?

Did you know that **DevOps** can accomplish task in minutes, which used to take hours or even days of work? So the companies can focus more on their business or the product? And maybe that's why IT industry is adopting DevOps at such a rapid pace.

Let me help you understand what is DevOps, and how it can solve some business-critical problems.

Everyone likes a good story and here it is.

Meet Emma. She's an artist, and has a passion of collecting artworks.

Emma has her own **art gallery** from where she sells artwork to the public. Emma wants to expand her business online through a mobile app.Users around the world can access her gallery, and make a purchase through a **mobile app**. Emma does not have a team who can do this.

She'll need a team of developers, testers and admin like people who can accomplish this task.

So Emma approaches a software consulting firm, and explains her idea.

Reggie, the Director of Dev and Ops Team, explains her process of development, delivery and service of the software.

Avi is the Project Manager of Software Development Team. Avi explains the software development process to Emma.

Freddy, Head of the Operations Explains Emma, how the mobile app will be hosted on the cloud server.So Emma decides to sign the deal after meeting all these experts.

Emma is curious by nature, and she wants to understand the software development process. So here we go.

Software development is a very well-defined, and organized process.

- **First phase—> requirement gathering and analysis.**

Information will be collected like product features for the users. How it'll be used.Some user requirements.Current state of the market, and few other things.

- **Second phase—> planning**

What do we want? It determines the cost and resource required for implementation of the product, and also the risks associated with it.

- **Third phase—>designing**

Architects will design the software based on the inputs from previous phase.Architects will produce design documents.These will be basically **roadmap for the developers.**

- **Development phase**

This is where developers will write the software code based on the design, and they already look very excited about it.

- **Software testing phase**

the software will be tested by software testers for any defects.

Software will be promoted to production only after fixing all the issues.

- **Deployment Phase**

At this stage, software is deployed to the production environment
so users can start using the product.It's the responsibility of system admin,and the entire operations team to make sure software is up and running all the time.

- **maintenance phase**

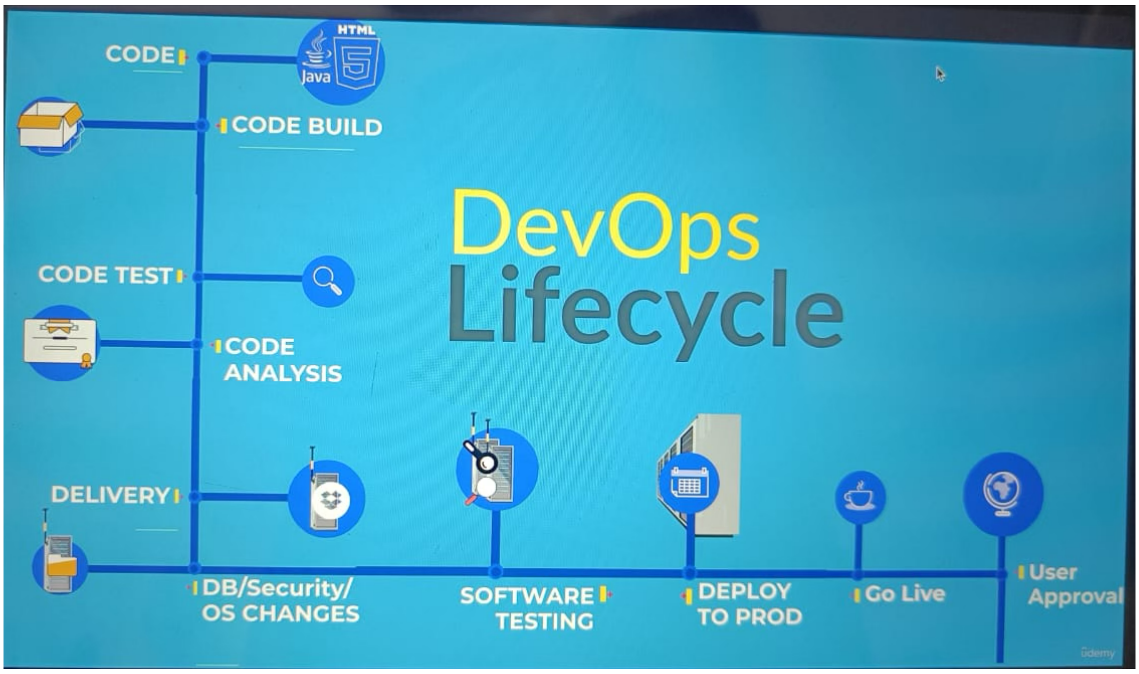
It is a balance between regular changes and uptime.

Requirement gathering—> planning—> Architects designs —> developement —> testing—> deployment—> maintenance
This entire process is called software development lifecycle(SDLC)
There are different models in SDLC.

- waterfall model
- Agile.
- Spiral.
- Big Bang and there are few others.

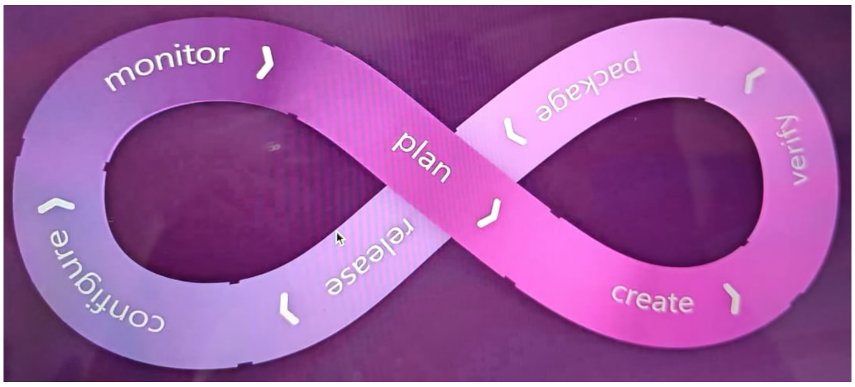
WATERFALL MODEL:
In a waterfall model, each phase must be completed before the next phase can begin.
Requirement completes then only planning begins, and then development starts. When everything is completed, all the features will be tested, and then maintenance phase starts.
It is very difficult here to go back, and change something that was not well-thought out in the planning.
Working software is produced very late in the lifecycle. It may take months in our case.
Emma is not sure about all the requirement at once. She would like to observe the product development, and alter or add new ideas along the line.
So Waterfall model is a no-go.
AGILE MODEL:
So instead of developing all the requirements for months, it can be divided into smaller lists. Work on the list for two to four weeks,
and then move on to the next list and so on and so forth.
This lifecycle is called as Agile.
Each iteration could be two to four weeks.
Demonstration of the features can be given to Emma after every iteration. and based on the feedback from Emma, new ideas can be injected in the next iteration.
So the development gets started with Agile SDLC.
Developers has started coding after all the planning. Avi has instructed Freddy from the operations team to deploy the code on the servers so it can be further tested by the software testers.

The operations team started deploying the code to the server as per the instructions, but at times, testers are not able
to access the servers or even their test cases are failing.
Avi has informed Freddy about the failed deployment. Freddy is confused as everything was done as per the instructions, and servers looks healthy. Agile SDLC puts extra stress on the ops team. There will be regular code changes. These changes needs to be deployed on the servers so testers can conduct their testings.
And this happens several times in a single iteration like that.
You have several iterations to go.
Ops team is tired with regular deployment requests. There are clear instructions which causes the deployment failures. Ops team is also occupied with production support. They need to maintain system uptime all the time. In-between these failures and fixes, Avi has passed the deadline of demonstration. Emma is waiting for a long time to see the first glimpse of her mobile app. Somehow the team convinces Emma, and explains her about the delay. Avi is not happy with the ops team. He reaches out to Freddy to tell him that this cannot be repeating.
Dev and ops are all support.Dev is Agile.
All about regular and quick changes.
Ops is ITIL-driven. Provides stable environment for the product.
There's a big wall of confusion in-between these two parties.
A developer task their code over the wall and ops team responsibility is to deploy the core on the servers.
Developers complain about delay in deployments. Ops team complains about unclear instructions, and tossing their work over the wall.
Emma has existing customers who wants to use the app, but there are frequent delays and errors. Emma is not so happy.
Reggie, the director understands this very well, that unhappy customer could mean direct business loss.
Reggie once attended an Agile conference where the host talked about DevOps as how it fixes the code delivery issues.
Reggie understood that Dev is Agile, but OPS is still waterfall.
So Reggie decides to bring in a DevOps consultant who can fix the code delivery issue, and skyrocket the business.
He explains to everyone that he does not have a magic wand.
**Everyone has to work collaboratively, communicate effectively,
and integrate the entire code delivery process.**
He explains and train dev team on ops concepts, so they can communicate with ops team more effectively. He also trained ops team on Agile concepts so ops can collaborate with dev team,
and the most important factor, automation training and instruction



keep monitoring

to every team across the board.
Automation of each and every task in code delivery process, like code build, code testing, software testing, infra changes, deployments and everything that comes along the way.
So the dev and ops start working together.Each and every task is automated in the delivery process by every team so the whole process can be integrated together. So finally, they automated,
and integrated the delivery process.
Let's see the entire automated



DevOps lifecycle now.
Aa a completely automated DevOps lifecycle,no human intervention, so no human errors.It's a reputable and time-saver of course.Emma, any request now can be delivered very quickly
because everything is automated. Emma is happy to see her happy customers.

code-developers commit code
Code build-deployable software architecture

Code test-unit and integration test
Code Analysis vulnerability best practices
Delivery-deploy changes to staging
Db/security/os changes
Software testing-QA/functional load,performance tests
Go live-users traffic diverted to new changes
Devops lifecycle

CONTINUOUS INTEGRATION

- Continuous integration is an automated process in DevOps.Which generates software and its features quickly and efficiently.

Developers write several lines of code while creating a software working in a team.
It's an ideal practice to store all this code at a centralized place.
This centralized repository is called a version control system like GitHub.
Everyday developers will pull and push code such repositories several times in a day. So code changes or code commit happens continuously.
This code will be moved to build server on build server.
This code will be build, tested and evaluated, which generates the software, or we call it artifact at this stage.
This artifact or software will be stored in a software repository.
Artifact or software is really an archive of files generated from the build process based on the programming language.
This artifact will be packaged in a specific format.
Artifact packaging format could be war or jar in Java. DLL/EXE/MSI in Windows or even ZIP/Tar ball From repository
It will be shipped to servers for further testing. After deploying this artifact on the servers, software testers can conduct further testing, and once they approve, it can be shipped to production servers. So that's how it works or does it?
These developers are creating a software model and have worked for three weeks straight. That's a lot of code, really.
And as per the process, all this code will be fetched by the Build server, and this code is build tested. and oh ,lots of errors, bugs, conflicts, build failures.
Now Developers have to fix all these defects, have to rewrite the code at several places. Lot of rework, really.
This could have been much easier if the problem was detected very early in the process, but then code was collected with defects for three weeks.
And now, yeah, you have to fix all that. So the code is getting merged into the repository, but not really getting integrated.
The solution to this is a very simple and a continuous process after every single commit from the developers, the code should be built and tested, so no waiting

and collecting all these codes with bugs, but then developer commits several times in a day, so it's not humanly possible to do a build & release several times in a day.
I mean, the manual process so that's simple, just automated.
So when the developer commits any code and automated process, will fetch the code, build it tested and send a notification if there is any failure.
As soon as the developers receives a failed notification, he or she will fix the code and commit it again. So again, build and test new changes, if it's good, then it can be versioned and stored in a software repository.
And it's all automated.
Any defects can be caught as soon as it's merged with the code.
This automated process is called **continuous integration,** or CI.
The goal of CI is to detect defects at a very early stage, so it does not multiply. IDE is used by developers for coding.
These IDE will be integrated with version control system to store and version the good. Build tools based on the programming language.
Software repositories to store artifacts.
Continuous integration tools that integrate everything.

CONTINUOUS DELIVERY

Continuous delivery is an automated process of delivering code changes to servers quickly and efficiently.
Continuous delivery is the extension of continuous integration,
We have seen continuous integration is automation of our code build and test, developers any code, it will be automatically built and tested.
If everything is good, the artifact generated from this process will be stored in software repositories.
The goal of CI is to detect defects at very early stage so it does not multiply.
Ops team will get regular requests to deploy the artifacts generated from the CI process on servers for further testing.
Ops team with all the info they got deploy the artifacts to the servers, at times the deployment also fails, which leads to higher lead time.
Dev & Ops team needs to work together to fix such deployment failures.
And this happens on and off. We have to understand that in agile development, there will be regular code changes which needs to be deployed on servers for further testing. Deployment is not just about shipping the software to the servers. It's more than that.
A deployment could mean also server provisioning, installing dependencies on servers, configuration changes, network or firewall rules changes, and then deploy the artifact to the server and there could be many more things.

Ops team will be flooded with such requests as CI process will generate faster and regularly.
After the manual deployment, information will be sent to the QA team for testing after conducting testing, QA team will send information back.
There is too much of human intervention and manual approval in this process.
So as this terminator says automate it and save your time and also failures.
Any and every step in deployment should be automated.
There are a lot of automation tools available in the market, like ansible, puppet, chef for system automation, terraform confirmation for cloud infrastructure, automation, Jenkins Octopus deploy for CICD automation.
And there are many others to choose from based on your need.
Software testing also has to be automated. Any test process like functional, load, performance, databases, network and security and any other test cases.
So Ops team, will write automation code for deployment ,testers will write automation code for software testing and sync it with developers source code.
So now we have a process integrated with deployment automation, which triggers software testing, all three teams and processes integrated together.
Continuous delivery process.Automate every step and then stitch everything together.That gives you continuous delivery automation.