**Devops Tools and AWS for Java Microservice Developers**

**https://capgemini.udemy.com/course/devops-tools-and-aws-for-java-microservice-developers/learn/lecture/14181203#overview**

**What are Micro Services?**

We understand what microservices are: we first understand what Monolithic Application?

Monolithic Application and their characteristics?

**For example**: We are developing a Hospital Management Application:

* Patient Registration module which captures the patient details
* Patient clinical module which will captures all clinical information from x-ray, blood test etc.
* Bed management, this will handle allocation a bed
* Claim management module which make sure the insurance claims are in place.

Now, we are following a monolithic approach we put all this code in one single application   
**Characteristics:**

* As this application grows a huge codebase
* Even Fix a bug or defect in the application it will be very complex. Because we are finding out for all this module where the defect has happened.
* Every time we fix a bugs or defects, every time we need to deploy our application.
* Adding new features, it is also difficult make sure the one of the other modules will also affect when we are making the change.
* We added our whole application to only one programming language
* One of the component or service system fails then that will not bring down the entire application

For this reason, behind we are mainly focused on Micro Services?

* It is a small and focused services which gather those pieces the change for the same reason.
* **For example:** We can split our monolithic hospital application into a clinical service, Registration service, bed management service, and a claim management service. Four different micro services applications and the code boundaries on the each of this is defined by the business boundary or the business problem they are solving that way we can avoid them to go to large.
* **Characteristics:**
* Micro services are small and focussed
* Autonomous, Because the package each micro services to different servers or machines and deploy them on to their own machines.
* Once the deployed all the communication through across these services happens through network calls.
* Each of the services will expose out an API, so that the other services can use that APIs to consume the services provided by them.
* **Heterogenous**: It means each of the services we can written different programming languages like java, phyton, .Net and Nodejs and we can run different platform or operating system. And they can all communicate using APIs they expose with usually are REST.
* **Robustness**: It means if one of the component or service system fails, we can gracefully degrade of service.
* **Scalability**: As of our application is user based grows, A lot of users start accessing our application, our application should be scale, simply deploy those micro services to multiple servers depending on the multiple users using that module or micro service.
* **Easy to deploy**: Deploy a particular service to bug fix in the service or even a new feature could be easily added an existing micro service and only that service will be deployed to production.
* **Reusability and replaceability**: We can transfer and using the data or APIs to one service into another, we can replaceable the services if any issues on that.

The Golden rule to evaluate that the application is a micro service?

* We can change and deploy our application without changing/impacting other services.

**Cloud Computing:**

What cloud computing is and why it’s so popular?

We are providing all the micro services application needs like: Storage, Networking, Databases, Security, and Auto-scaling, Load balancing and all this on our own they are machines available locally or a corporate level.

Cloud takes it to a different level, and it provides all this services like:   
**OnDemand**: We can access this service as required charges only for what we are used,   
**Ambiguity**: meaning we can access this service from anywhere and everywhere over a network and finally these services can also be shared.   
**Shared**: By sharing the cost will go down for **Ex**: If we have a 100 GB of storage if we are using 40-50 GB the rest GB can be shared by another application or a different company all together which will bring down the cost for us.

**Some of the popular clouds are:**

1. AWS
2. Azure
3. GCP
4. Oracle
5. IBM
6. Alibaba etc.

We can even have our own cloud on our corporate data centre, but we need is networking, some space and Ram. The advantage of using existing cloud providers is there is no capital, there is no initial investment which we need when we choose an existing cloud provider like AWS, GCP etc. They will provide all the infrastructures for us will only charges based on how much we use them. And maintenance is very low because we have all the set-up that is required and then lot of engineering team on already, so it will cost is less as well.

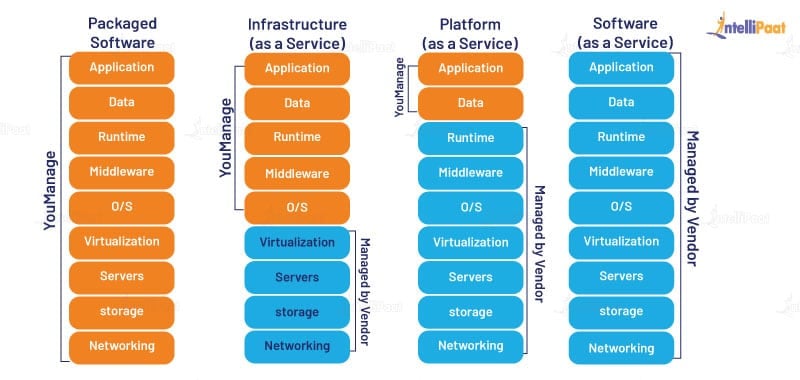
**Cloud Types:**

There are three types of cloud, or cloud providers are:

**1) Public:** Public is open for all; anyone can create an account use the services providers by the cloud providers. Ex: AWS, Azure, GCP, IBM, Oracle and many more, so these cloud providers will charge on-demand and on usage basis and given supports and maintenance for the services they provide.

**2) Private:** A private cloud is where organization will create and maintaining their own cloud Ex: Dell, 3M, Siemens etc. these are all taking the help of Devops and make the own cloud.

**3) Hybrid:** Hybrid cloud where we can mix-up the public and private clouds Ex: public + public, public + private, some of private + private also

Service Models:  
  
There are three different service modules are when we are working with cloud providers:  
Even In interview they might ask which service models’ previous organization was using or do you know about service models

1. **IaaS (Infrastructure as a service):**

If the organization using the IaaS service models then that means it is using a cloud provider or a third party for networking storage, Services, and virtualization so it gets all the machines, networking and all that setup by using a cloud provider like AWS, Azure and so on. Ex: EC2, VPC, Security Groups, Cloud formation, and Terraform

And the rest of the services installing the operating system, middleware, runtime the servers, then the databases will be taken care by the organization itself.

**Ex**: Organization handles:

* Application, Data, Runtime, Middleware
* Operating System handles:
* Virtualization, Services, Storage, Networking

1. **PaaS (Platform as a service):**

This is where we will get the other software like the operating systems, middleware’s to communicate and then even the runtime as a part of the cloud. From the cloud provider we are going to take our all the services, in case of AWS we are using EBS (Elastic bean stack) this gives us a ready to use environment to run our applications, then we have SNS do messaging in the AWS.

**Ex**: Organization handles:

* Application and data rest all will be taken care by the cloud providers.

1. **SaaS (Software as a Service):**

We need not worry anything here as developers we simply focusing the developing the application, everything else will be ready on a button click or as soon as we commit the code all the infrastructure that will be created for us on the fly all the software’s required that are installed for us, databases will be ready, all the application will be tested or even we can do a continuous deployment and continuous delivery. It is a beauty of using SaaS models.  
In case of AWS, we use RDS to manage databases for us S3 storage to maintain all the data and using Lambda we can quickly write code and executed in any programming language, it is moving to complete serverless we need not worry about which server of our application has to be deployed and so on.

**AWS From a micro service perspective:**

We assuming for working hospital management software and we have a four different service like: Clinicals, Bed management, Patient Registration, Claims.  
User can assess this software we need take care of several other things, starting with:

* **Step 1** **-** **Environment**: Where this application must be deployed.  
  **Ex**: In AWS EBS brings all the AWS services or integrate with all the other AWS services that was required for our application to work.
* **Step 2 - Security**: Users can access our application securely.  
  **Ex**: In AWS for security, we have IAM(Identity access management) /Security groups
* **Step 3 - Load Balancing**: It is very important as a load our application is having to increase configure a load balancer.  
  **Ex**: In AWS for load balancing which provides EBS (Elastic load Balancing) which is super easy to configure once we configure it, it will balance the load across the AWS instances or servers we create.
* **Step 4 - Auto Scale**: Otherwise, we manually scale them as the load grows when million users hit out our micro services don’t won’t to server go down.  
  **Ex**: In AWS For scaling, we don’t to manually do it AWS automatically scales it, once we configure scaling, the scaling will start happening when the load increases automatically the new server will be created, when the load goes down automatically the server taken out. We need not pay too much.
* **Step 5 - Health checks and monitoring**: We must make sure health of our application is okay and we need to continuously monitor them.  
  **Ex**: In AWS also has cloud watch for health checks on the applications that we deploy, and it continuously monitors them and notify as a there is any issue.

So, these all steps are our responsibility as well only when these are done. Then we can claim that our applications are ready for the cloud are they are ready for continuous deployment.

**Continuous Integration, Delivery and Deployment: (CI/CD)**

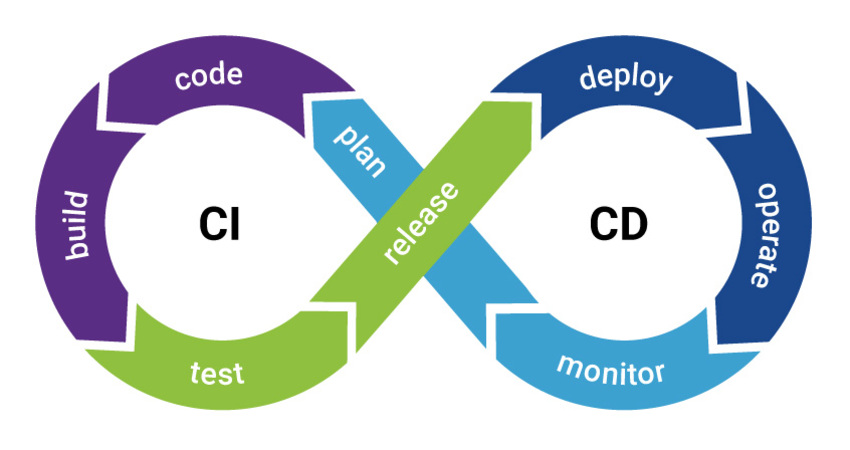
It defined as the End goal is to get out the product or features ASAP to production Env to the customers.  


For this Typical delivery pipelines for all the stages, so as soon as developer commits the code, or application will build test are run it will go to staging and then to production.

**Continuous Integration:** where we will right several test cases run them on a continuous basis as new code keeps coming in from the developers. So continuous integration will pretty much stop at the build stage.

**Continuous Delivery:** Extends continuous integrations and runs production like test usually in a stage environment. This stage environment is where we will mimic the production environment, it will have all the data of the production, all the use cases of the production and we will automate all those tests as well, so continuous delivery is one more step forward and it can get us deliver the feature of the product to production or customer. Usually in continuous delivery there will be some manual process involved, this could be an approval from a devops architect and deployment manager, or it could just be some manual test we have to run.

**Continuous Deployment:** Completely automates from commit all the way to the production or customer. There are no manual steps at all.

  
**Source**  **Build** **Test** **Deploy** **Monitor**  
Commit Compile Integration Pre-Prod Health and   
Code Review Unit Test System Prod Unusual  
Program war/jar/dll Load Activities  
 Image UI  
So continuous deployment is the end goal, but continuous integration is a starting point once we get better with continuous integration we move on to continuous delivery where we still have manual step, once we get to continuous deployment that means as soon as developer commits the code, our feature will hands up the customers.

**Devops:**

Devops simply enables us to do continuous delivery and deployment, so its provides us with various tools, devops is a process that we are follow and in that process we come across various tools and this tools keep changing depending on the organizational priorities.

Go through the stages are:  
**Source** **Build** **Test** **Deploy** **Monitor** Git Maven war/jar war/jar Cloud watch Docker or Gradle container Container Prometheus  
 Image Kubernetes  
 or AWS EBS

* **Git** - Git is a popular tool, using git we push our code to central repository, then comes the build stage before we going to build stage using **Docker** to dockerize our application.
* **Maven or Gradle:** It will take the code from git repositories and at the end when the build is complete it will compile our classes run the test it will generate a war/jar or if we docerize of our project which is easily, once we build this the result will be docker Image.
* **Docker Image:** For this image will have everything required for our application to run we can take war/jar or images and deployment to test environment 🡪 then take the war/jar deploy to pre-prod as well and we will be monitoring tools we will get for **Kubernetes** or **AWS**.  
  We simply deploy our container war/jar files to cloud like AWS we can use the EBS service and quickly deploy our application with all the environment the database etc. and test it in a few min.
* **Kubernetes:** will take our docker container orchestrate them, means it can create any number of containers for us once our application is ready we can scale that application if we want 100 containers as we get millions of request of our applications we can do that in a few seconds, once we configurate we need not worry about it Kubernetes will automatically span of the our containers and one container is goes down it will bring up automatically another container that is the beauty of Kubernetes. This is the reason Kubernetes is the part of every cloud.   
  As a developer how to use Kubernetes and will deploy our application or push our application to Kubernetes cluster, we will test it we can be pretty sure the application is work on a clustered environment. Then a cloud engineer or we take the same application a cloud which as Kubernetes hosted or clouds like AWS, Azure, GCP etc. all of them supports Kubernetes instead of using load balancing all that comes in the cloud use that within Kubernetes as well.

Then comes the monitoring using the cloud watch for AWS and Prometheus for Kubernetes and many more monitoring tools.

* **Jenkins:** It is a tool that will integrate all these, it can work with all these and help us to build continuous integration, delivery, and deployment pipelines.

**AWS EC2 and Linux basics:**

AWS Regions and zones: The AWS zones spread across the world and these geographical zones are:

USA East - Zone A, Zone B  
Europe - Zone X, Zone Y  
India - Zone 1, Zone 2

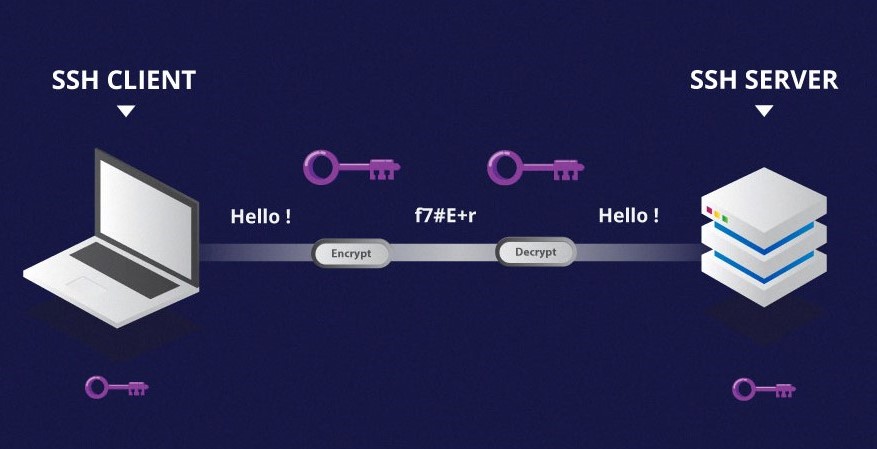
* At least two zones for every region. Because the one zone is goes down and the other zone is up and running.
* AWS maintains the Edge locations in between the zones and customers themselves. So that the request is catch in this edge locations. These edge locations act like a cash increasing the response time for us, but we developers will not be directly touching the regions and zones we should be aware of it.
* <https://aws.amazon.com/about-aws/global-infrastructure/>

**EC2(Elastic cloud computing):** The AWS machine gives us a virtual machine to run our applications on them. AWS gives us to AMI (Amazon machine image) these images will have all the required software like the operating system (OS) and the software depending on the want to deploy and work with **Ex**: java, phyton, MySQL, Docker will combination of all these. If we pick which we want all the software’s will be ready to use.

On windows we can access our instance through secured socket layer or secured shell (SSH) using putty tool. On Mac machine, simply use for Mac Terminal.

**Launch a EC2 Instance:** It is a 7 steps process which is simple just like how we launch a virtual machine on our desktop or when we buy a new laptop we will be considering.

1. Operating system (OS)
2. CPU (Processor capacity)
3. RAM
4. Hard Disk (Storage)  
   **AWS console -> services -> search ec2 – running instances -> Launch instances ->**  
   **Step 1:** Amazon machine image(AMI) -> Select Amazon Linux, Because it is very simple to use ->   
   **Step 2:** Choose instance type, pick the free tier again ->   
   **Step 3:** Configure storage, pick the hard disk capacity by default it is 8 ->   
   **Step 4:** Security groups are the way to restrict access to our new instance by default only way is through the access this using SSH protocol ->   
   **Step 5:** Create new key pair add a name as **Ex**: awskeys once we click on create the key will downloaded on our machine. It is a private key; the public key will automatically store on Aws.  
   **Step 6:** Last step is review and launch ->

**SSH**: (Secured protocol) Aws EC2 uses SSH to for communication that is once we launch an AWS instance or machine to communicate with that machine to access that machine to our local laptop or computer, we need to use the SSH protocol.  
It is a protocol that enables secured communication between any two machines on the network. The communication has happened over port 22 and it will use encrypt/decrypt when the data goes out and comes in. It uses public key cryptography, because it will generate the two key public and private in case of AWS the public key is stored on the remote instance and then the private is stored by the client.  
If we want to it manually use the ssh-keygen -t rsa rsa is the popular cryptography algorithm. Once will generate the keys it will store .ssh/id\_rsa, go to the EC2 instance -> network and security -> key pairs folder -> store our all-key pairs. Amazon EC2 uses a superpower rsa keys 2048-bit SSH-2 RSA keys it is very hard and impossible to decrypt. we can create 5000 keys per region.  
  
**How to connect remote EC2 instance from our windows box using MobaXTerm:**

Go to our browser download mobaxterm -> Download a portable edition -> open MobaXTerm -> click on Session -> SSH -> In the remote host add our EC2 Public Ipv4 DNS -> click on Advanced SSH settings -> add our pom file then click ok ->   
login as: ec2-user -> sudo -I -> pwd use these commands.

**How to connect remote EC2 instance from our windows box using putty and puttyGen:**

It is required to connect to any machine using SSH mainly AWS. -> Go to browser download putty -> [www.putty.org](http://www.putty.org) -> download based on our windows 64 bit or 32 bit -> Along with download puttyGen also it will use to converts pen file to ppk file. Once its download launch puttyGen -> click on load -> load the private key file (pem file) -> change will to All files -> open it will automatically load us, then we save with .ppk file -> save private key -> yes -> the save the ppk file in our local -> Then launch putty -> Host name past place paste our public EC2 Public Ipv4 DNS -> Need to provide the key details to do that -> Expand SSH under the connection -> Expand Auth -> This place we need to load our ppk file -> yes -> then click ok -> login as: ec2-user -> sudo -I -> pwd use this commands.

**Some EC2 tips:**

* Use top command is showing usage of processes that are running and all that, if we execute the top command keeps checking continuously and our session will not end.
* 30 GB Storage of our free tier on AWS, if we not using instances terminate that and instead of using another instance. and 750 hours per month, if we not using that instance, we can stop that select -> instance state -> stop instance.

**Linux Important Directories:**

1. ls / : All the important folders on a Linux box / forward slash represents the root directory, it is a top level of directory on the Linux file system.
2. ls /bin : Bin directory is where all the command line utilities are stored.
3. ls /boot : This includes all the Linux start-up files including the Linux curler.
4. ls /etc : The etc folder will have all the configuration files for the system to run.
5. ls /dev : All the hardware and software devices for everything starting from the terminal all the way to the cd drive etc.
6. ls /home : This is the home directory for every user.
7. ls /lib : Libraries for the carnal and various command line utilities are stored in this lib folder.
8. ls /mnt : For mounting, the mount points removable media like cd drive to mount a particular folder on a Linux machine to an external server, external cd drive or even a USB drive etc.
9. ls /opt : All the application binaries are stored when we install certain software’s like world’s perfect star office etc.
10. ls /proc : It shows this directory holds all the currently running kernel related processes.
11. ls /root : The home directory to the root user.
12. ls /sbin : This is for the system administration command.
13. ls /temp : It is temporary directory, and Linux often clears this directory periodically.
14. ls /usr : All the small programs that are all accessible to various users including many systems administration command line utility etc. are stored here.
15. ls /var : All the variable data including in log files printers pool etc. are stored here.

**Browsing Directories and files:**

pwd -> This will display the present working directory -> ex: /root -> ls -> list out all the sub-directories -> mkdir -> Make a directory name it as mydirectory -> ls -> mydirectory -> cd mydirectory -> it will take into this directory -> pwd ->  
/root/ mydirectory -> cat> myfile.txt -> It will append whatever data we enter on the command line -> Ex: All the power within you, you can do anything and everything. To come out of this hit control c -> ls -> new file called myfile.txt -> ls -l -> It gives a long listing format display various information -> ls -a -> will list out all the files including the hidden files -> ls -r -> will include all the contents of all the sub directories as well. -> mkdir subdir -> creates a sub directory -> pwd -> /root/mydir/subdir -> cd subdir -> cat >myfile.txt , you are the creator of your destiny hit control c ->ls -> myfile.txt -> cd ..  
->pwd -> /root/mydir -> ls -r -> subdir myfile.txt -> copy a file using cp myfile.txt myfile1.txt-> This command copy the contents of one file to another file -> See the content of the file -> cat myfile1.txt -> see the same content what I entered before -> If we move the file one directory to another use -> mv myfile1.txt subdir -> to remove a file use -> rm myfile.txt -> ask warning -> yes -> it will removed -> remove a sub directory use -> rm -r subdir -> yes -> prompting the text file yes-> yes-> ls-> no sub directory.

**How to use VI Editor to create and edit our files:**

Use the command -> vi myfile.txt -> if the file exists creating a new file, If not goes to editing   
The VI Editor is works in three different modes.

1. Open VI Editor it will be command mode it will be waiting for our command, in this mode we can copy data, paste data, delete data, undo, redo, move data, save and quit etc. So, all that can be done in the command mode.
2. To add or edit data we must go into the insert mode.
3. Execute mode where we can set the line numbers, delete line numbers, save, and quit and more.   
   we enter into insert mode add just press the I --INSERT – - mode inserting the values -> press escape -> INSERT mode is gone -> ready for receiving a another command i -> it will goes into starting of the line -> I -> esc -> to upend data where we are use a -> start upending the data after the point or after the curser point -> esc -> I -> use A -> it will go to the end of the current line -> create a new line use o -> O -> insert a new line before the current line -> type dd -> it will removes the entire line 2dd -> it will removes a two lines yy -> For copying that line -> p -> pasted the line   
   Execute mode -> Shift : -> take into the execute mode this is where we can save our file wq -> stands for write and quit ->  
   q -> do not save the file simply quit, q! -> quit forcefully without saving anything, :/String -> any string we want to check

**Yum command:** To install software packages on our machine. Yum will pull the software from the repository and install them in our machines.

1. yum repolist – List out all the repos
2. yum list installed – This will show all the software’s that are all already installed.
3. grouplist – The software’s can be installed as a group.
4. can install the package use yum install package name which we want to install.
5. To launch the web server, use service webserver name start.
6. yum install curl – it is a command help to hit a server on a particular port.
7. service webserver name start – This will stop the server.
8. Want to delete the service yum remove package name.
9. Want to see the logs it will store under the m log directory use – we see last 5 lines of logs   
   tail -5/var/log/yum.log.
10. yum history – This command will bring us all the commands we executed at before.
11. Search for a particular package if its available use – yum search package name

**Git walkthrough:**

It is popular version control system, the reason for its popularity is free and open source, fast and light weight. It uses SHA1 (Secured hashes algorithm) to exchange data to save all the files as well on the remote repository, branching for git creating is very easy, so for this reason git is very popular.

**Version control system:**

**What is a “version control system”?**

Version control systems are a category of software tools that helps in recording changes made to files by keeping a track of modifications done in the code.

**Why Version Control system is so Important?**

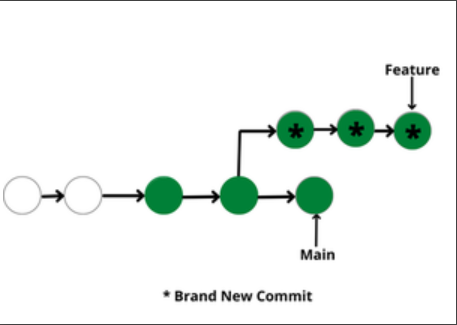
As we know that a software product is developed in collaboration by a group of developers they might be located at different locations and each one of them contributes to some specific kind of functionality/features. So, to contribute to the product, they made modifications to the source code (either by adding or removing). A version control system is a kind of software that helps the developer team to efficiently communicate and manage(track) all the changes that have been made to the source code along with the information like who made and what changes have been made. A separate branch is created for every contributor who made the changes, and the changes aren’t merged into the original source code unless all are analysed as soon as the changes are green signalled, they merged to the main source code. It not only keeps source code organized but also improves productivity by making the development process smooth.  
Basically Version control system keeps track on changes made on a particular software and take a snapshot of every modification. Let’s suppose if a team of developer add some new functionalities in an application and the updated version is not working properly so as the version control system keeps track of our work so with the help of version control system, we can commit the new changes and continue with the previous version.

**Benefits of the version control system:**

* Enhances the project development speed by providing efficient collaboration,
* Leverages the productivity, expedites product delivery, and skills of the employees through better communication and assistance,
* Reduce possibilities of errors and conflicts meanwhile project development through traceability to every small change,
* Employees or contributors of the project can contribute from anywhere irrespective of the different geographical locations through this **VCS,**
* For each different contributor to the project, a different working copy is maintained and not merged to the main file unless the working copy is validated. The most popular example is **Git, Helix core, Microsoft TFS,**
* Helps in recovery in case of any disaster or contingent situation,
* Informs us about Who, what, When Why changes have been made.

**Use of Version Control System:**

* **A repository:** It can be thought of as a database of changes. It contains all the edits and historical versions (snapshots) of the project.
* **Copy of Work (sometimes called as checkout):** It is the personal copy of all the files in a project. You can edit to this copy, without affecting the work of others and you can finally commit your changes to a repository when you are done making your changes.
* **Working in a group:**Consider yourself working in a company where you are asked to work on some live project. You can’t change the main code as it is in production, and any change may cause inconvenience to the user, also you are working in a team, so you need to collaborate with your team to and adapt their changes. Version control helps you with the, merging different requests to main repository without making any undesirable changes. You may test the functionalities without putting it live, and you don’t need to download and set up each time, just pull the changes and do the changes, test it and merge it back. It may be visualized as.



**Types of Version Control Systems:**

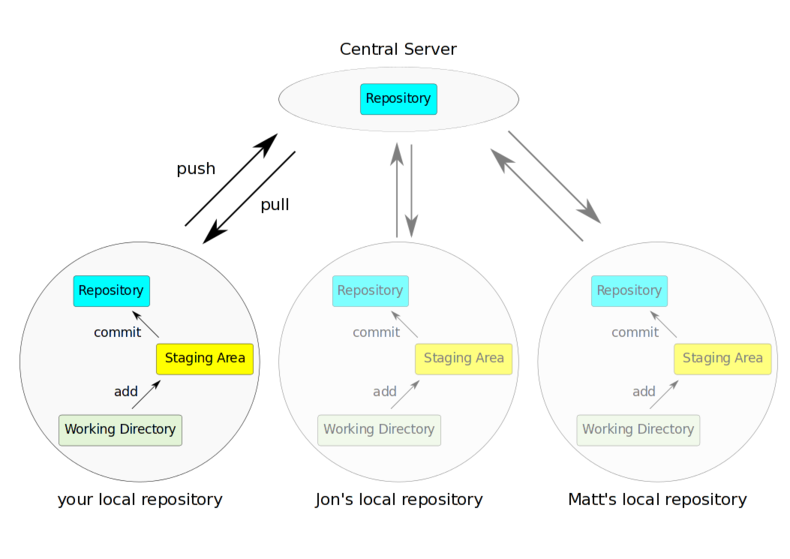
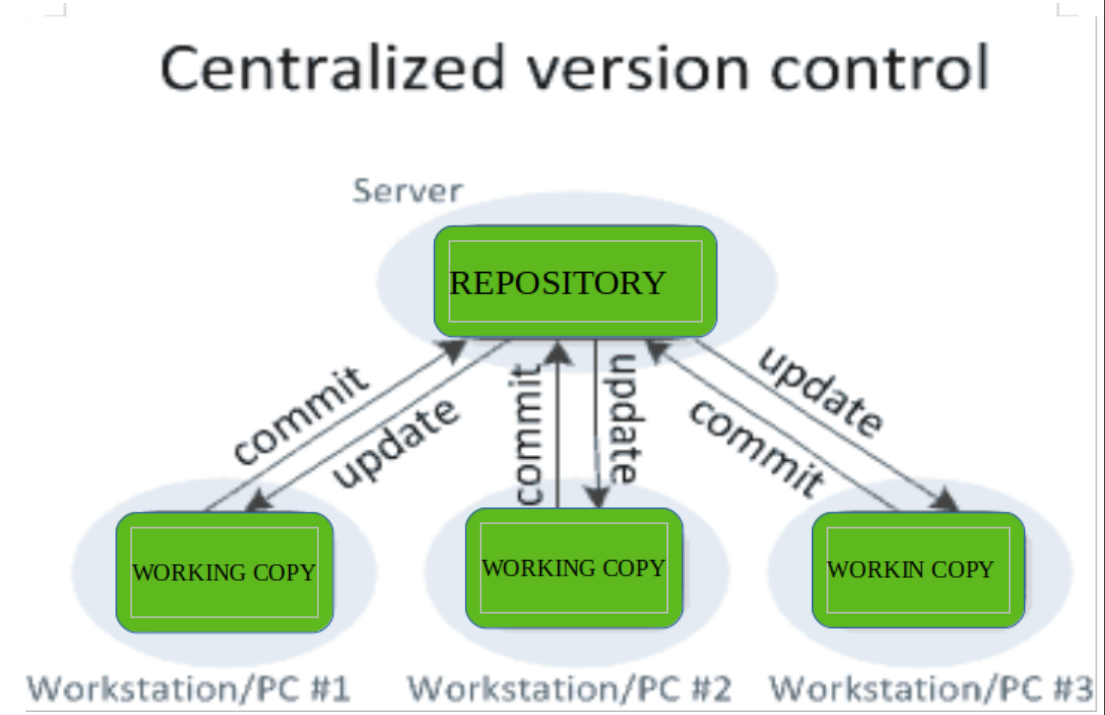
* Local Version Control Systems
* Centralized Version Control Systems
* Distributed Version Control Systems

**Local Version Control Systems:** It is one of the simplest forms and has a database that kept all the changes to files under revision control. RCS (Revision control system) is one of the most common VCS tools. It keeps patch sets (differences between files) in a special format on disk. By adding up all the patches it can then re-create what any file looked like at any point in time.

**Centralized Version Control Systems:** Centralized version control systems contain just one repository globally and every user need to commit for reflecting one’s changes in the repository. It is possible for others to see your changes by updating.

Two things are required to make your changes visible to others which are:

* You commit.
* They update.



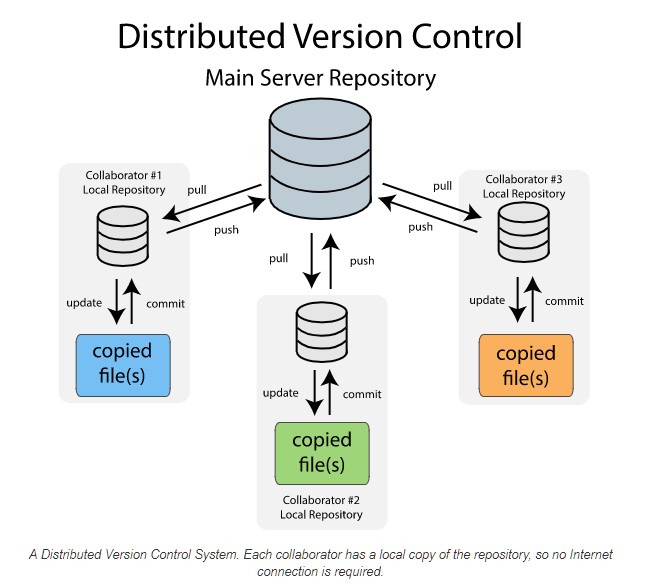
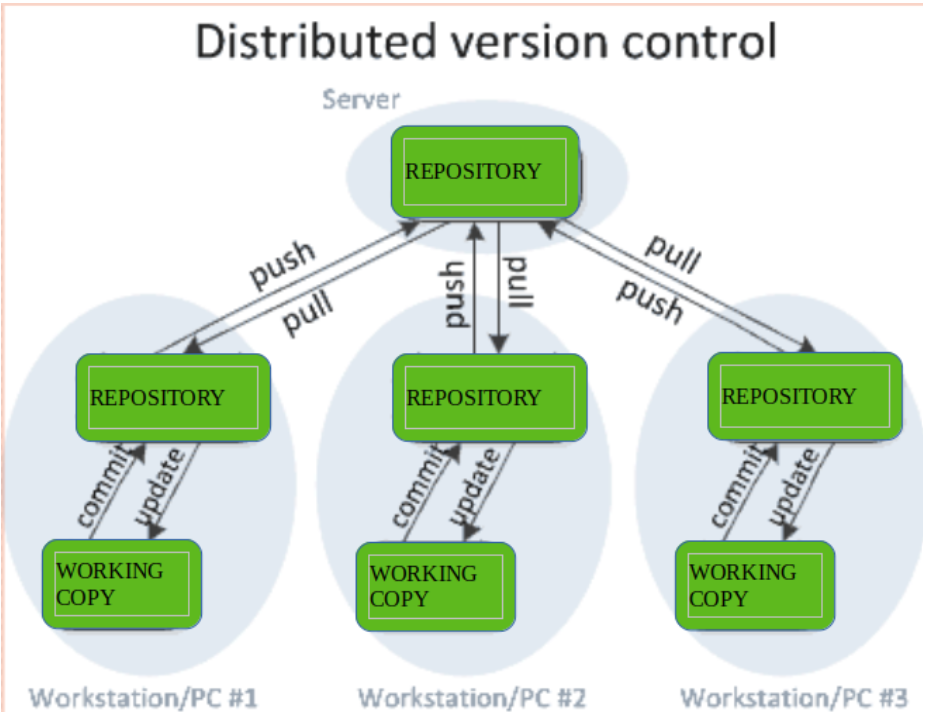
The **benefit** of CVCS (Centralized Version Control Systems) makes collaboration amongst developers along with providing an insight to a certain extent on what everyone else is doing on the project. It allows administrators to fine-grained control over who can do what.

It has some **downsides** as well which led to the development of DVS. The most obvious is the single point of failure that the centralized repository represents if it goes down during that period collaboration and saving versioned changes is not possible. What if the hard disk of the central database becomes corrupted, and proper backups haven’t been kept? You lose absolutely everything.

**Distributed Version Control Systems:** Distributed version control systems contain multiple repositories. Each user has their own repository and working copy. Just committing your changes will not give others access to your changes. This is because commit will reflect those changes in your local repository, and you need to push them in order to make them visible on the central repository. Similarly, when you update, you do not get others’ changes unless you have first pulled those changes into your repository.

To make your changes visible to others, 4 things are required:

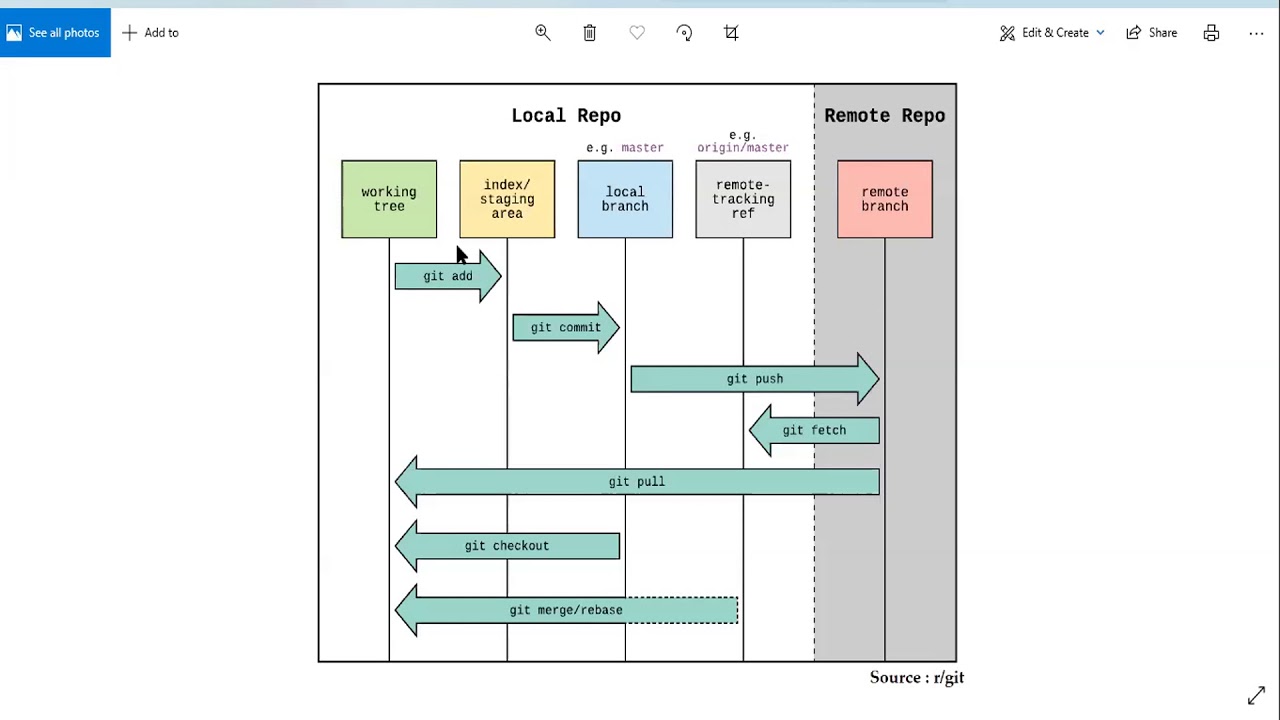
* You commit.
* You push.
* They pull.
* They update.

The most popular distributed version control systems are Git, and Mercurial. They help us overcome the problem of single point of failure.  
 

**Purpose of Version Control:**

* Multiple people can work simultaneously on a single project. Everyone works on and edits their own copy of the files, and it is up to them when they wish to share the changes made by them with the rest of the team.
* It also enables one person to use multiple computers to work on a project, so it is valuable even if you are working by yourself.
* It integrates the work that is done simultaneously by different members of the team. In some rare cases, when conflicting edits are made by two people to the same line of a file, then human assistance is requested by the version control system in deciding what should be done.
* Version control provides access to the historical versions of a project. This is insurance against computer crashes or data loss. If any mistake is made, you can easily roll back to a previous version. It is also possible to undo specific edits that too without losing the work done in the meanwhile. It can be easily known when, why, and by whom any part of a file was edited.

**Git Workflow and some important terms used in every git project:**

When we join a new company or new project all these projects will maintain a remote repository, so which ever project we are working on we clone that project to our local machine once install the git client in our machine when we clone that we get entire project in our local repository, make changes to which ever we want stage. Those changes on to the local repository on to the staging area and we are sure all the changes are correctly then we can use a commit command, commit to the local repository each commit will have own unique id and finally when we are ready, we push all committed those changes to the remote repository.

So, each commit will have unique committed that will be generated by git and it will maintain all that history for us Head is a pointer to the latest commit. Delete the latest commit head is move onto the previous commit and so on.

**Git configuration:**

How to provide configuration information to git using to config option.

* 1. git config - - local: Meaning for the current local repository we are working with.  
     we need to add - git config user.name “Arun” we need check the users - git config user.name   
     we need to add - git config user.email [varun@gmail.com](mailto:varun@gmail.com) we need to check email - git config user.email
  2. git config- - global: Global means for the user no matter how many repositories user works on same configuration will be applied to all the repositories.
  3. git config- - system: It is for the entire system, no matter how many users’ login into that system they all do commits, add etc.  
     .gitconfig: Git internally stores all this information in this file.

After git init, using ls.git - command it will shows an all folders inside the git  
vi index.html esc a written a html code, after written use esc key and :wq enter git add index.html -> git commit -m “initial version” -> git log : will shows all the commits history we make, always latest commit comes on top   
we want to compare two commits use git diff paste the two commit ids this command will show which one we modified.

We can see the entire commit history in one line use git log –oneline

How to undo or revert the changes we make, it happens

1. Before staging
2. After Staging and
3. After commit: This topic is coming with while we are working remote repository

cat index.html: It will show a modified html file.

git checkout index.html: This will get the latest from the local repository and replace my local copy.

cat index.html: This will show again have the latest one.

I can modify the file and staged the file, in the stages updated the modified file, if we want to revert the changes.

git reset Head index.html so head always point to the latest commit on our branch, this will only reset the staged changes.

Again, use the cat index.html I still have the same index file after we use git checkout index.html then use a cat index.html

Finally, we got the earlier version.

This is how we can revert the changes.