Q4. How do all these tools work together?

Developers develop the code and this source code is managed by Version Control System tools like Git etc.

Developers send this code to the Git repository and any changes made in the code is committed to this Repository.

Jenkins pulls this code from the repository using the Git plugin and build it using tools like Ant or Maven.

Configuration management tools like puppet deploys & provisions testing environment and then Jenkins releases this code on the test environment on which testing is done using tools like selenium.

Once the code is tested, Jenkins send it for deployment on the production server (even production server is provisioned & maintained by tools like puppet).

After deployment It is continuously monitored by tools like Nagios.

Docker containers provides testing environment to test the build features.

Q5. What are the advantages of DevOps?

Technical benefits:

Continuous software delivery

Less complexity to manage

Faster resolution of problems

Manpower is reduced.

Business benefits:

Faster delivery of features

More stable operating environments

Improved communication and collaboration

More time to innovate (rather than fix/maintain)

Q6. What is the most important thing DevOps helps us achieve?

According to me, the most important thing that DevOps helps us achieve is to get the changes into production as quickly as possible while minimizing risks in software quality assurance and compliance. This is the primary objective of DevOps.

And also it provides clearer communication and better working relationships between the Ops team and Dev team to collaborate together to deliver good quality software which in turn leads to higher customer satisfaction.

Q7. Explain with a use case where DevOps can be used in industry/ real-life.

Etsy is a peer-to-peer e-commerce website focused on handmade and supplies, as well as unique factory-manufactured items. Etsy struggled with slow, painful site updates that frequently caused the site to go down. It affected on sales.

Later With the help of a new technical management team, Etsy transitioned from its waterfall model, which produced four-hour full-site deployments twice weekly, to 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.

Q8. Explain your understanding and expertise on both the software development side and the technical operations side of an organization you have worked with in the past.

DevOps engineers almost always work in a 24/7 business-critical online environment. I was adaptable to on-call duties and was available to take up real-time, live-system responsibility. I successfully automated processes to support continuous software deployments.

\\

9)
GIT:

Q9. What are the anti-patterns of DevOps?

- We neoed a separate DevOps groupe
- Rebrand your ops/dev/any team as the devops

- Everyone is forced to keep ongoing.
- A lot of overtime demanded
- Provide appropriate resources
- ///////////////////////////////

Version Control System (VCS) Interview Questions

Now let’s look at interview questions on VCS:

Q1. What is Version control?

It is a system that records changes to a file over time so that you can recall specific versions later. Version control systems consist of a central shared repository where teammates can commit changes to a file .

uses of version control:

Version control allows you to:

Revert files back to a previous state.

Revert the entire project back to a previous state.

Compare changes over time.

See who last modified something that might be causing a problem.

Who introduced an issue and when.

Q2. What are the benefits of using version control?

1. With Version Control System (VCS), all the team members are allowed to work freely on any file at any time. VCS will later allow you to merge all the changes into a common version.
2. All the past versions and variants are neatly packed up inside the VCS. When you need it, you can request any version at any time and you’ll have a snapshot of the complete project right at hand.
3. Every time you save a new version of your project, your VCS requires you to provide a short description of what was changed. Additionally, you can see what exactly was changed in the file’s content. This allows you to know who has made what change in the project.
4. A distributed VCS like Git allows all the team members to have complete history of the project so if there is a breakdown in the central server you can use any of your teammate’s local Git repository.
5. Branching and merging: Creating a "branch" in VCS tools keeps multiple streams of work independent from each other while also providing the facility to merge that work back together,

Q3. Describe branching strategies you have used.

Feature branching
A feature branch model keeps all of the changes for a particular feature inside of a branch. When the feature is fully tested and validated by automated tests, the branch is then

merged into master.

- Task branching
  In this model each task is implemented on its own branch with the task key included in the branch name. It is easy to see which code implements which task, just look for the task key in the branch name.
- Release branching
  Once the develop branch has acquired enough features for a release, you can clone that branch to form a Release branch. Creating this branch starts the next release cycle, so no new features can be added after this point, only bug fixes, documentation generation, and other release-oriented tasks should go in this branch. Once it is ready to ship, the release gets merged into master and tagged with a version number. In addition, it should be merged back into develop branch, which may have progressed since the release was initiated.

Q) What is the difference between CVCCS (Centralized VCS) an DVCS?

Traditional version control (CVCS) helps you backup, track and synchronize files. Distributed version control (DVCS) makes it easy to share changes.

Centralized version control focuses on **synchronizing, tracking, and backing up files.**

- Distributed version control focuses on **sharing changes**; every change has a guid or unique id.
- **Recording/Downloading** and **applying** a change are separate steps (in a centralized system, they happen together).
- **Distributed systems have no forced structure**. You can create “centrally administered” locations or keep everyone as peers.

New Terminology

- **push**: send a change to another repository (may require permission)
- **pull**: grab a change from a repository

Key Advantages

- **Everyone has a local sandbox.** You can make changes and roll back, all on your local machine. No more giant checkins; your incremental history is in your repo.
- **It works offline.** You only need to be online to share changes. Otherwise, you can happily stay on your local machine, checking in and undoing, no matter if the “server” is down or you’re on an airplane.
- **It’s fast.** Diffs, commits and reverts are all done locally. There’s no shaky network or server to ask for old revisions from a year ago.
- **It handles changes well.** Distributed version control systems were *built* around sharing changes. Every change has a guid which makes it easy to track.
- **Branching and merging is easy.** Because every developer “has their own branch”, every shared change is like reverse integration. But the guids make it easy to automatically combine changes and avoid duplicates.
- **Less management.** Distributed VCSes are easy to get running; there’s no “always-running”

server software to install. Also, DVCSes may not require you to "add" new users; you just pick what URLs to pull from. This can avoid political headaches in large projects.

Key Disadvantages

- **You still need a backup.** Some claim your "backup" is the other machines that have your changes. I don't buy it — what if they didn't accept them all? What if they're offline and you have new changes? With a DVCS, you still want a machine to push changes to "just in case". (In Subversion, you usually dedicate a machine to store the main repo; do the same for a DVCS).
- **There's not really a "latest version"**. If there's no central location, you don't immediately know whether to see Sue, Joe or Eve for the latest version. Again, a central location helps clarify what the latest "stable" release is.
- **There aren't really revision numbers.** Every repo has its own revision numbers depending on the changes. Instead, people refer to change numbers: *Pardon me, do you have change fa33e7b?* (Remember, the id is an ugly guid). Thankfully, you can tag releases with meaningful names.

Q) What is GIT?

Git is a Distributed Version Control system (DVCS). It can track changes to a file and allows you to revert back to any particular change.

it does not rely on a central server to store all the versions of a project's files. Instead, every developer "clones" a copy of a repository to his local machine so that when there is a server outage, all you need for recovery is one of your teammate's local Git repository.

Q6. Explain some basic Git commands?

Command	Function
git config --global user.name "name" git config --global user.mail "mail id" git config --list	Configure the author name configure the author email list the configure detaails

Command	Function
git config --global user.name “name” git config --global user.email “E-mail”	Configure the author name and email address to be used with your commits
Git init	Create a new local local respository
Git clone /path/to/repository	Create a working copy of a local repository
git clone username@host:/path/to/repository	For a remote server, use
git add <Filename.> git add *	Add one or more file to staging
git commit -m “Commit message”	Commit changes to head
git commit -a	Commit any files you've added with git add, and also commit any files you've changed since then
git push origin master	Send changes to the master branch of your remote repository
git status	List the files you've changed and those you still need to add or commit
git remote add origin <server>	If you haven't connected your local repository to a remote server, add the server to be able to push to it

Q7. In Git how do you revert a commit that has already been pushed and made public?

There can be two answers to this question :

Remove or fix the bad file in a new commit and push it to the remote repository. This is the most natural way to fix an error. Once you have made necessary changes to the file, commit it to the remote repository for that I will use

git commit -m “commit message”
Create a new commit that undoes all changes that were made in the bad commit.to do this I will use a command

git revert < name of bad commit>

Q8. How do you squash last N commits into a single commit?

There are two options to squash last N commits into a single commit. Include both of the below mentioned options in your answer:

If you want to write the new commit message from scratch use the following command

```
git reset –soft HEAD~N &&

git commit
```

If you want to start editing the new commit message with a concatenation of the existing commit messages then you need to extract those messages and pass them to Git commit for that I will use

```
git reset –soft HEAD~N &&

git commit –edit -m”$(git log –format=%B –reverse .HEAD@{N})”
```

Q9. What is Git bisect? How can you use it to determine the source of a (regression) bug?

Git bisect is used to find the commit that introduced a bug by using binary search. Command for Git bisect is

```
git bisect < subcommand> < options>
```

Now since you have mentioned the command above, explain what this command will do, This command uses a binary search algorithm to find which commit in your project’s history introduced a bug. You use it by first telling it a “bad” commit that is known to contain the bug, and a “good” commit that is known to be before the bug was introduced. Then Git bisect picks a commit between those two endpoints and asks you whether the selected commit is “good” or “bad”. It continues narrowing down the range until it finds the exact commit that introduced the change.

Q10. What is Git rebase and how can it be used to resolve conflicts in a feature branch before merge?

According to me, you should start by saying git rebase is a command which will merge another branch into the branch where you are currently working, and move all of the local commits that are ahead of the rebased branch to the top of the history on that branch.

Now once you have defined Git rebase time for an example to show how it can be used to resolve conflicts in a feature branch before merge, if a feature branch was created from master, and since then the master branch has received new commits, Git rebase can be used to move the feature branch to the tip of master.

The command effectively will replay the changes made in the feature branch at the tip of master, allowing conflicts to be resolved in the process. When done with care, this will allow the feature branch to be merged into master with relative ease and sometimes as a simple fast-forward operation.

Q11. How do you configure a Git repository to run code sanity checking tools right before making commits, and preventing them if the test fails?

I will suggest you to first give a small introduction to sanity checking, A sanity or smoke test determines whether it is possible and reasonable to continue testing.

Now explain how to achieve this, this can be done with a simple script related to the pre-commit hook of the repository. The pre-commit hook is triggered right before a commit is made, even before you are required to enter a commit message. In this script one can run other tools, such as linters and perform sanity checks on the changes being committed into the repository.

Finally give an example, you can refer the below script:

```
#!/bin/sh
files=$(git diff –cached –name-only –diff-filter=ACM | grep
‘.go$’)
if [ -z files ]; then
exit 0
fi
unfmtd=$(gofmt -l $files)
if [ -z unfmtd ]; then
exit 0
fi
echo “Some .go files are not fmt’d”
exit 1
```

This script checks to see if any .go file that is about to be committed needs to be passed through the standard Go source code formatting tool gofmt. By exiting with a non-zero status, the script effectively prevents the commit from being applied to the repository.

Q12. How do you find a list of files that has changed in a particular commit?

For this answer instead of just telling the command, explain what exactly this command will do so you can say that, To get a list files that has changed in a particular commit use command

```
git diff-tree -r {hash}
```

Given the commit hash, this will list all the files that were changed or added in that commit. The -r flag makes the command list individual files, rather than collapsing them into root directory names only.

You can also include the below mention point although it is totally optional but will help in impressing the interviewer.

The output will also include some extra information, which can be easily suppressed by including two flags: git diff-tree –no-commit-id –name-only -r {hash}

Here –no-commit-id will suppress the commit hashes from appearing in the output, and –name-only will only print the file names, instead of their paths.

Q13. How do you setup a script to run every time a repository receives new commits through push?

There are three ways to configure a script to run every time a repository receives new commits through push, one needs to define either a pre-receive, update, or a post-receive hook depending on when exactly the script needs to be triggered.

Pre-receive hook in the destination repository is invoked when commits are pushed to it. Any script bound to this hook will be executed before any references are updated. This is a useful hook to run scripts that help enforce development policies.

Update hook works in a similar manner to pre-receive hook, and is also triggered before any updates are actually made. However, the update hook is called once for every commit that has been pushed to the destination repository.

Finally, post-receive hook in the repository is invoked after the updates have been accepted into the destination repository. This is an ideal place to configure simple deployment scripts, invoke some continuous integration systems, dispatch notification emails to repository maintainers, etc.

Hooks are local to every Git repository and are not versioned. Scripts can either be created within the hooks directory inside the ".git" directory, or they can be created elsewhere and links to those scripts can be placed within the directory.

Q14. How will you know in Git if a branch has already been merged into master?

I will suggest you to include both the below mentioned commands:

git branch –merged lists the branches that have been merged into the current branch.

git branch –no-merged lists the branches that have not been merged.