



****

🔧 **Git & GitHub Notes**

**📌 What is Git & GitHub?**

* **Git:** Ek distributed version control system hai jo source code ka management karta hai. Ye local machine pe changes track karta hai.
* **GitHub:** Ek remote server/service hai jo hume Git repositories store karne ke liye online jagah deta hai. Team collaboration ke liye best.

**Why Use Git?**

* **Tracks Changes** – Keeps a history of all modifications.
* **Collaboration** – Multiple developers can work on the same project.
* **Branching & Merging** – Create separate workspaces without affecting the main project.
* **Backup & Recovery** – Easy to revert to previous versions.
* **Works Offline** – No need for an internet connection to work locally.

**🛠️ Git Installation:**

* **Ubuntu/Linux:** sudo apt-get install git
* **Fedora:** yum install git
* Verify installation: git –version
* **After installing Git, configure it with your name and email (required for commits).**

git config --global user.name "Your Name"

git config --global user.email "your.email@example.com"

**🔹 Example:**

git config --global user.name "John Doe"

git config --global user.email "john@example.com"

**To check your current Git settings:**

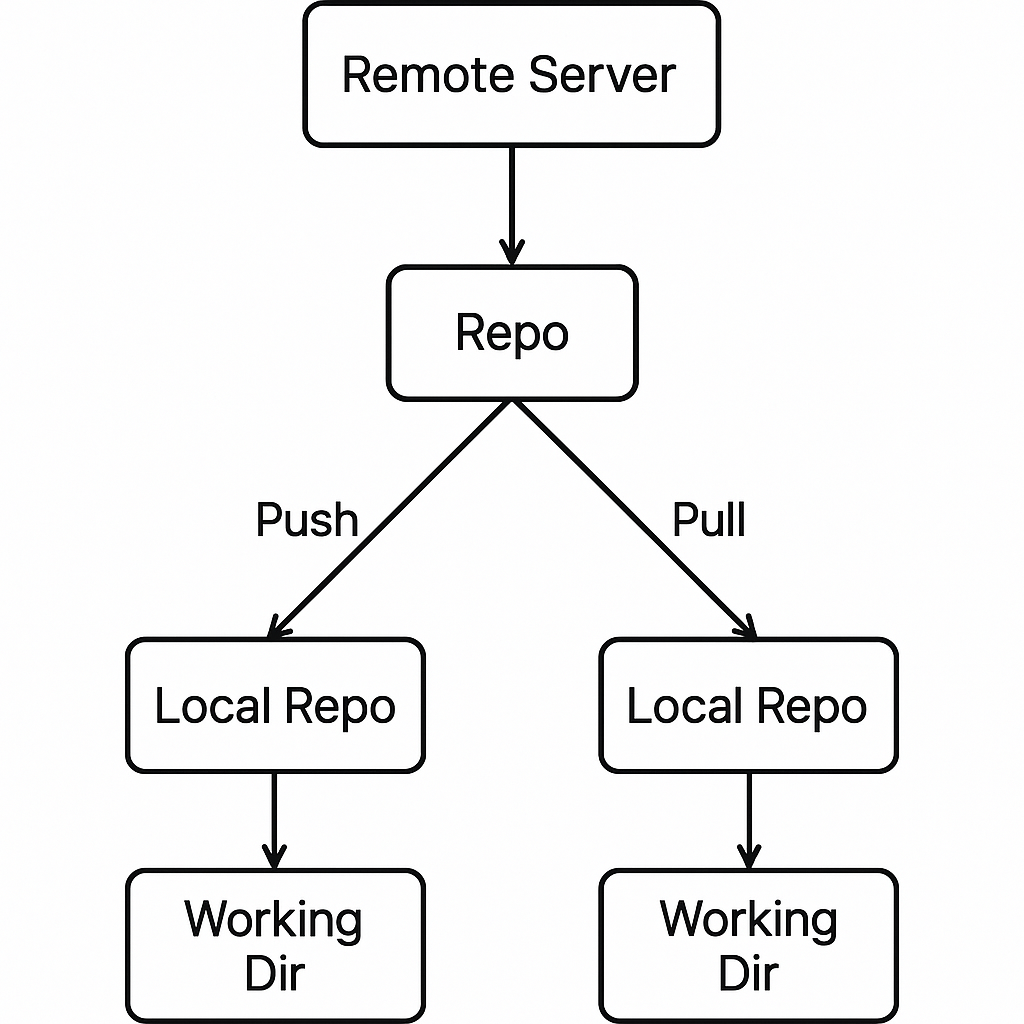
git config --list

**🔹 Expected Output:**

user.name=John Doe

user.email=john@example.com

📦 **Git Workflow:**



1**. Initialize a repository** – Start tracking a project with git init or clone an existing one with git clone.

2. **Make changes** – Modify files in your project.

3. **Stage changes** – Add updated files to the staging area using git add.

4. **Commit changes** – Save changes to Git’s history with git commit -m "message".

5. **Push changes** – Upload changes to a remote repository (git push).

6**. Pull updates** – Get the latest updates from a remote repository (git pull).

**Key Git Commands for This Section:**

**git init** # Initialize a new Git repository, Ek .git folder create hota hai us directory mein.

**git clone <url>** # Clone an existing repository

**git add <file>** # Add a file to the staging area

**git commit -m "message"** # Commit changes with a message

**git push origin main** # Push changes to remote repository

**git pull origin main**  # Pull the latest changes from remote

**Important Notes:**

* **Git Status:** git status

➤ Check karta hai file ka status — staged hai, committed hai ya nahi.

* **Agar tumhare paas bohot saari files hain aur sirf ek specific file ka status dekhna chahte ho:**

git status filename.txt

**🔍 Basic Commit History**

**git log**

Isse tumhe commit ka:

* Hash, author, date, commit message, sab kuch dikhega.

**📂 Show History for a Specific File**

git log filename.txt

Sirf uss file ke commits dikhata hai.

|  |  |  |
| --- | --- | --- |
| **Git vs. GitHub (What’s the Difference?) Feature** | **Git** | **GitHub** |
| **What it is?** | A tool that manages version control on your local system. | A platform that stores Git repositories online for collaboration. |
| **Where it runs?** | On your local computer. | On the web (GitHub, GitLab, Bitbucket, etc.). |
| **Main Use?** | Tracks and manages code versions. | Enables team collaboration, issue tracking, and pull requests. |
| **Example?** | git init starts tracking your project. | git push sends your project to GitHub. |

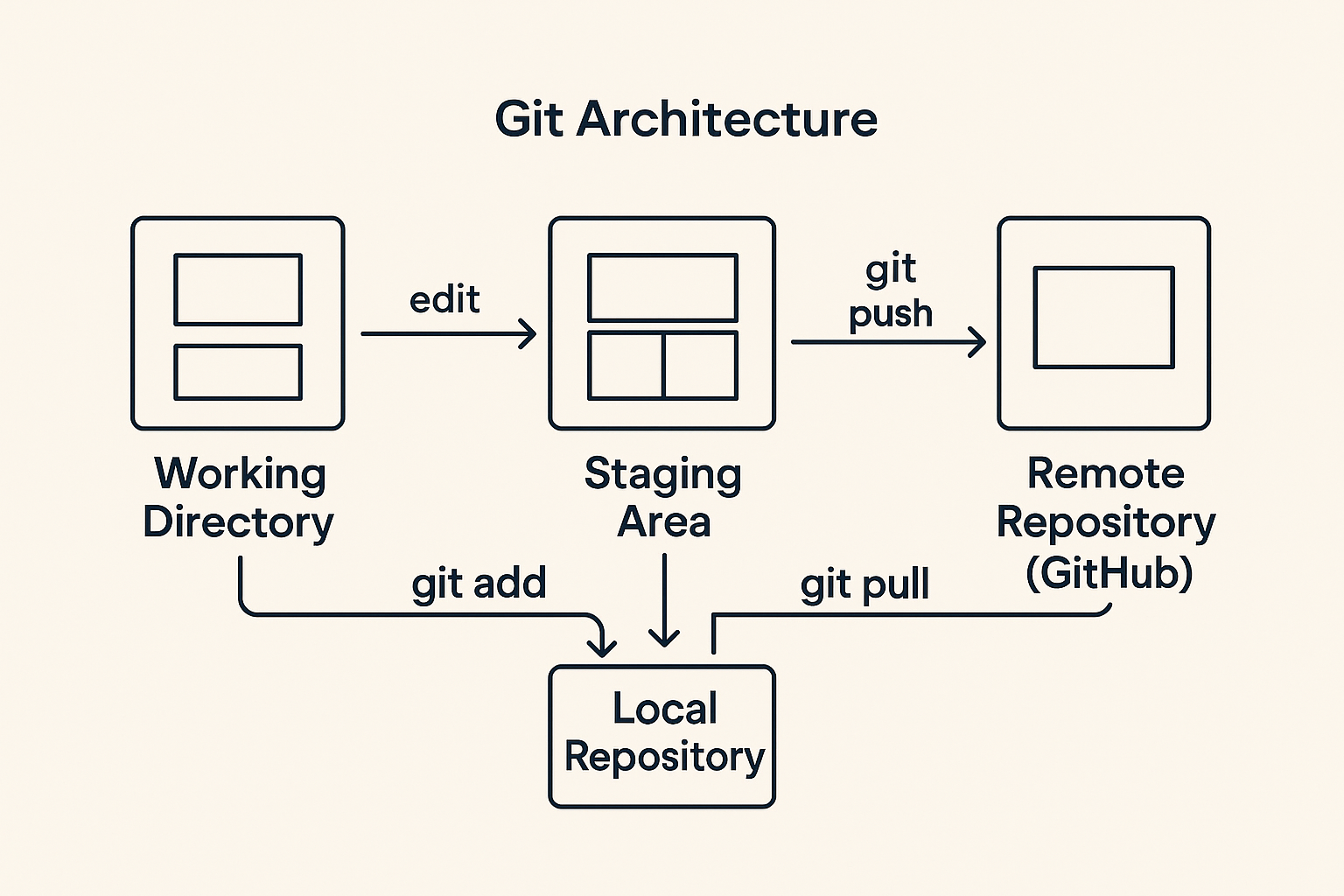
**🧠 Important Concepts: ✅** Snapshot is Incremental:

* Git ka commit **incremental** hota hai.
* Sirf wahi changes save karta hai jo **last snapshot** ke baad aaye ho.
* Matlab: Agar ek file mein thoda sa change hai, toh pura file dubara store nahi hoti.

**💬 Bonus Gyaan for Interview:**

* Git **staging area** = buffer zone before commit.
* Use git log to see commit history.
* git clone <url> se remote repo local mein copy hoti hai.
* git merge <branch> se do branches ka code ek saath aata hai

**🔄 Git Workflow (Pull/Push/Commit ka Flow)**

**🧱 Git Architecture (Based on Diagram):**

**📌 Terms & Flow samjho simple words mein:**

* **Working Directory**: Jahan hum file edit karte hain.
* **Staging Area**: Temporary space hai jahan hum file ko git add se ready karte hain commit ke liye.
* **Local Repo**: Jahan humara code local machine pe store hota hai (committed version).
* **Remote Repo (GitHub)**: Online server jahan humara code safe rahta hai team ke saath share karne ke liye.

**🔃 Command Flow:**

1. **Edit File** in working directory.
2. **git add file.txt** → staging area mein file chali jaati hai.
3. **git commit -m "message"** → changes local repo mein store ho jaate hain.
4. **git push origin branch\_name** → remote repo (GitHub) pe push ho jaata hai.
5. **git pull →** remote repo se latest changes leke aata hai local repo mein.

**🔁 Flow Diagram:**

**[Working Directory] → git add → [Staging Area] → git commit → [Repository]**

📥 **Files Ko Staging Mein Kaise Dalein**

Jab aap file create ya modify karte ho, Git automatically usko staging mein nahi daalta. Uske liye aapko manually git add command run karni hoti hai.

**🪜 Steps:**

**✅ Step 1: Current Status Check karo**

git status

**🔹 Output agar ek nayi file index.html add ki hai:**

Untracked files:

(use "git add <file>..." to include in what will be committed)

index.html

**✅ Step 2: Ek Single File ko Staging Mein Daalo**

git add index.html

**✅ Step 3: Multiple Files Ko Add Karna**

git add file1.txt file2.txt

**✅ Step 4: Saari Files Ko Ek Saath Staging Mein Add Karna**

git add .

**✅ Step 5: Check karo kya Staging Mein Gaya Ya Nahi**

git status

**🔹 Output:**

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

new file: index.html

**💾 Committing Changes**

Commit karne ka matlab hai ki aapne jo staged files hain unko **permanently Git repository** mein save kar diya. Samajh lo jaise project ka ek “checkpoint” bana diya.

**✅ Step 1: Commit karo aur message do**

git commit -m "Added homepage"

**🔹 Output:**

[main abc1234] Added homepage

1 file changed, 10 insertions(+)

create mode 100644 index.html

**✅ Step 2: Saare Modified Files Ko Commit Karna (Without git add)**

git commit -a -m "Updated multiple files"

⚠️ *Important*: Ye sirf **pehle se committed files** ke liye kaam karta hai. Nayi files ko pehle git add karna zaroori hai.

**📜 Commit History Dekhna**

**✅ Step 1: Short Version of History**

git log --oneline

**🔹 Output:**

abc1234 Added homepage

def5678 Fixed navbar issue

**✅ Step 2: Full Details ke saath History**

git log

🔹 Output mein aapko author, date, commit message sab milega.

**↩️ Staged Changes Undo Karna**

Kabhi-kabhi galti se files staging mein chali jaati hain aur unhe hataana padta hai.

**✅ Step 1: Ek File Ko Staging Se Hatana**

git reset HEAD index.html

🔹 Ab index.html wapas working directory mein aa jaayegi, staging mein nahi rahegi.

**✅ Step 2: Saari Files Ko Staging Se Hatana**

git reset

**📌 Quick Recap – Most Important Commands**

| **Command** | **Description** |
| --- | --- |
| git add <file> | Specific file ko staging mein daalta hai |
| git add . | Saare modified aur naye files ko staging mein daalta hai |
| git status | Staging aur untracked file ka status dikhata hai |
| git commit -m "message" | Staged files ko commit karta hai ek message ke saath |
| git log --oneline | Short commit history dikhata hai |
| git reset HEAD <file> | File ko staging se hataata hai |

**Commit**

* Store changes in repo and you will get a commit ID, It is of 40 alpha-numeric characters
* It uses **SHA-1 checksum** concept:  
  ➤ Codes ko binary mein convert karta hai   
  ➤ To avoid any changes  
  ➤ Agar code mein koi bhi change hua toh checksum diff ho jaayega

**Push file to GitHub:** git push -u origin master

* First time pe username/password maangega

**GitHub token generate karna hoga**: Developer setting mein → kyunki password wrong batata hai

**git show** → shows new add hua ya nahi

**Pull karna easy hai:** Connect local to central:

git remote add origin <Your GitHub URL>

git pull origin master

**Find commit by message**: git log --grep "keyword"

**Git Pull vs Git Fetch**

| **Command** | **Description** |
| --- | --- |
| git fetch | Retrieves remote changes but **doesn’t affect** current branch |
| git pull | Retrieves remote changes **and merges** them |

**git pull = git fetch + git merge,** Use git fetch when you want to **review** changes before merging

* Safe workflow:

**git fetch origin**

**git diff origin/master**

**git merge origin/master**

**🌿 Git Branching, Merging & Stashing**

**🔎 Git Branch Kya Hota Hai?**

Git mein branch ek alag line of development hoti hai. Isse aap naye feature ya bug fix pe kaam kar sakte ho bina main project ko affect kiye.

**🔹 Example Flow:**

**(**main) ────────┐

├── (feature-branch) ─── New Feature

└── (bugfix-branch) ─── Bug Fix

**origin remote repository ka default naam hota hai.**

Jab tu kisi GitHub, GitLab ya Bitbucket wale repo ko clone karta hai, to Git us remote ka naam by default origin rakhta hai.

**master Git ki pehli ya default branch ka purana naam hota tha.**

Ab naye Git versions mein is branch ka naam main hota hai by default, lekin purane projects mein master naam kaafi common hai.

🔁 Short:

* **master = default branch ka naam (old default)**
* **main = default branch ka naya naam (recommended now)**

**🛠️ Branch Create aur Switch Karna**

**✅ Step 1: Existing Branches Dekho:** git branch

🔹 Output: \* main

**\* batata hai ki aap kis branch par ho.**

**✅ Step 2: Nayi Branch Banao:** git branch feature-1

**Yeh feature-1 naam ki branch banayega, lekin aap uspar switch nahi karoge abhi.**

**✅ Step 3: Nayi Branch Par Switch Karna**

**📌 Old Method:bgit** checkout feature-1

**📌 Recommended (New Method):** git switch feature-1

**🔹 Output:** Switched to branch 'feature-1'

**✅ Step 4: Ek Hi Command Mein Branch Banao Aur Switch Karo**

git checkout -b feature-1

**OR**

git switch -c feature-1

**🔀 Branches Ko Merge Karna**

**Jab feature complete ho jaaye, to us branch ko main mein merge karna hota hai.**

**✅ Step 1: main branch pe jao:** git switch main

**✅ Step 2: Doosri Branch Ko Main Mein Merge Karo:** git merge feature-1

**Output agar koi conflict nahi hua:** Updating abc1234..def5678

Fast-forward

new file: feature.txt

**⚠️ Merge Conflicts Kaise Handle Karein**

Agar dono branches ne ek hi file ke same part ko modify kiya hai, to Git automatically merge nahi kar sakta.

**🛠️ Merge Conflict Solve Karne Ke Steps:**

1. File open karo jisme conflict hai (e.g., index.html)
2. Aapko kuch aisa dikhega:

**<<<<<<< HEAD**

**Main branch ka code**

**=======**

**Feature-1 branch ka code**

**>>>>>>> feature-1**

1. Correct version choose karke manually file ko edit karo
2. Resolve hone ke baad:

**git add index.html**

1. Fir commit karo:

**git commit -m "Resolved merge conflict"**

**🗑️ Branches Ko Delete Karna**

**✅ Step 1: Local Branch Delete Karna:** git branch -d feature-1

**⚠️ Agar wo branch abhi tak merge nahi hui hai to Git warning dega. Usko force se delete karne ke liye:** git branch -D feature-1

**✅ Step 2: Remote Branch Delete Karna:** git push origin --delete feature-1

**📌 Quick Recap – Important Commands**

| **Command** | **Description** |
| --- | --- |
| **git branch** | Saari local branches dikhata hai |
| **git branch <branch-name>** | Nayi branch banata hai |
| **git switch <branch-name>** | Branch switch karta hai (recommended) |
| **git checkout <branch-name>** | Branch switch karta hai (old method) |
| **git checkout -b <branch-name>** | Nayi branch banata aur switch karta hai |
| **git switch -c <branch-name>** | Nayi branch banata aur switch karta hai |
| **git merge <branch-name>** | Current branch mein doosri branch merge karta hai |
| **git branch -d <branch-name>** | Merged branch ko delete karta hai |
| **git push origin --delete <branch>** | Remote branch delete karta hai |

***♻️ Undoing Changes aur Reverting Commits – Git mein***

**🔄 Undo Karna – Local Changes (Commit se Pehle)**

Agar tumne file mein kuch changes kiye hain lekin abhi stage ya commit nahi kiya, to usse discard kar sakte ho.

**✅ Step 1: Ek Specific File ke Changes Undo Karo**

git checkout -- filename.txt

**⚠️ *Ye command file ko last committed state mein le jaata hai. Ye action undo nahi ho sakta.***

**✅ Step 2: Saare Files ke Changes Undo Karo**

git checkout -- .

**🚫 Unstaging Files (Commit se Pehle)**

**Agar aapne git add se file ko stage kar diya, lekin commit nahi kiya – aur aap staging se hataana chahte ho:**git reset HEAD filename.txt

🔹 Isse file staging area se hat jaayegi, **lekin file ke changes working directory mein rahenge**.

**✅ Saare Files Ko Unstage Karna:**

git reset

**🧨 Undo Karna – Commit (Remote Push se Pehle)**

Agar commit ho gaya hai lekin remote pe push nahi hua, to us commit ko undo kiya ja sakta hai:

**✅ Step 1: Soft Reset (Commit Undo, Changes Staged Rahenge)**

git reset --soft HEAD~1

🧠 *Last commit ko undo karega, lekin changes staged rahenge.*

**✅ Step 2: Hard Reset (Sab Kuch Delete – Beware)**

git reset --hard HEAD~n (n means no. of lines jitni hatani hai)

⚠️ *Yeh commit bhi delete karega aur file ke changes bhi — poori tarah clean ho jaayega. Ye action undo nahi hota.*

**🔁 Revert Karna – Jab Commit Already Push Ho Chuka Ho**

Agar commit remote repo pe push ho chuka hai, to git revert use karo. Ye safe method hai kyunki history delete nahi hoti.

**✅ Last Commit Revert Karna**

git revert HEAD

🔹 Isse ek naya commit banega jo last commit ke effect ko reverse karega.

**✅ Specific Commit Revert Karna**

git revert <commit-hash>

**🔍 Use this:**

git log --oneline

**🧳 Git Stash – Jab Feature Adhura Hai:**

**Use case**: Kaam chal raha hai kisi feature pe, par achanak doosra urgent task aa gaya — toh apne kaam ko temporary side mein rakhne ke liye stash use karo.

| **Task** | **Command** |
| --- | --- |
| Save work | git stash |
| See stash list | git stash list |
| Apply stash | git stash apply stash@{0} |
| Clear stash | git stash clear |

⚡ stash is like “save as draft” → kaam bacha ke side mein rakh do, baad mein continue karo!

**🔀 How to merge Branches**

* We use **Pulling mechanism** to merge branches  
  (New update data of one branch get copy into master branch)
* Merge karne se pehle **master branch** mein aao  
  (Jis branch mein merge karni hai, usme checkout kro)

🔁 **Git Reset vs Git Revert**

**📂 Git Reset: Powerful command** jo local changes ko undo karta hai.

**✅ To reset staging area:** git reset <file\_name> **OR** git reset .

**🔥 To reset both staging + working directory:** git reset --hard

**🧠 Extra Gyaan on Reset:**

| **Type** | **Impact** |
| --- | --- |
| git reset | Sirf staging area se hatata hai |
| git reset --soft | Commit history remove karta hai, file working dir mein rehti hai |
| git reset --mixed (default) | Staging reset karta hai, file rehti hai |
| git reset --hard | Staging + working dir dono reset – **dangerous**, **irreversible** |

📌 Mostly reset internal ya local fixes ke liye use hota hai — **private level pe kaam** ke liye.

**⏪ Git Revert:** Revert command help karta hai kisi purane commit ko undo karne mein.

| **Reset** | **Revert** |
| --- | --- |
| Kaam karta hai before commit | Kaam karta hai after commit |
| History ko badal deta hai | History ko preserve karta hai |
| Private use ke liye safe | Public/shared repos ke liye safe |
| Files ko hata deta hai | New commit banata hai jo purana wala revert karta hai |

**📜 Diagram Description:**

* Master branch ke commits → x + y + z + abc
* Revert ke baad → x + y + z wapas aa gaya with new commit
* File move hoti hai, history forward jaata hai

**❗ Notes:**

* Revert data delete nahi karta, sirf naya commit banata hai jo previous state laata hai.
* Ye safer option hai, specially jab team ke saath kaam kar rahe ho.

**🧾 Commands:**

git revert <commit-id>

**Commit ID kaise milega?:** 🡺 git log

* git reset = Dangerous in shared repos.
* git revert = Clean, traceable, and audit-friendly.
* Use git revert --no-edit to skip commit message prompt.
* Revert karne ke baad agar conflict aaye, toh woh manual resolve karna padega (just like merge conflicts).

🧹 **Git Clean, Remove & Tags**

**🧽 How to remove untracked file**

**Untracked file** = jo git ke control mein nahi hai (not added, not committed)

**git clean -n** # (dry-run) → confirmation + delete

**git clean -f** # (forcefully) → force delete

**How to remove file in git:** git rm <file name>

**➤ Agar local repo se bhi remove karna hai:** git rm --cached <file name>

**📁 How to remove directory in Git:** rm -rf <directory name>

**🏷️ TAGS in Git:** Git Tags ka use hota hai kisi specific commit ko mark karne ke liye – jaise ki version releases: v1.0, v2.1.3, etc.

🧠 Tags are like permanent labels jo batate hain: “Yeh commit ek important version hai!”

**✅ Tag a Specific Commit:** git tag -a v1.0 <commit-hash> -m "Tagging old commit"

**Versioning or release ke liye tag banate hain:** git tag -a <tagname> -m <msg> <commit-id>

➤ **Tag banane ke baad sab tag list dekhne ke liye**: git tag

**🧠 Extra Gyaan on Clean, Remove & Tags:**

**🔹 git clean:**

* -n safe option hai → pehle dikhata hai kya delete hoga.
* -f bina pooche delete kar deta hai. Be careful.

**🔹 git rm:**

* Tracked file ko hata deta hai aur next commit mein remove ho jaata hai.
* --cached sirf Git se hataata hai, file local mein rehti hai (good for .env, secrets).

**🔹 Directory delete:**

* rm -rf is a Linux command, not Git's. Careful while using — poori directory uda deta hai 💣

**Git Tags:**

* Tags mostly release versioning ke liye hote hain — jaise v1.0.0, v2.2.5
* 2 types:

**1.** **✅ Lightweight Tag:** Sirf ek pointer hota hai kisi commit pe.

git tag v1.0

Yeh v1.0 tag **current commit** pe lag jaayega

**2. ✅ Annotated Tag:** Full object create karta hai Git mein **(Recommended)**

git tag -a v1.0 -m "Release version 1.0"

🔹 -a = annotated  
🔹 -m = message

**🚀 Pushing Tags to Remote**

By default, git push **tags push nahi karta**. Alag se push karna padta hai.

**✅ Push Single Tag:** git push origin v1.0

**✅ Push All Tags:** git push origin --tags

**❌ Deleting Tags**

**✅ Delete Local Tag:** git tag -d v1.0

**✅ Delete Remote Tag:** git push origin --delete tag v1.0

🔄 **GITHUB CLONE**

To create a copy of repo on your local machine: git clone <URL of GitHub>

🟢 This will create a **new repo directory** on your local machine called my-repo (ya jo repo ka naam hoga).

**🧾 Step-by-step Flow:**

1. **git clone <repo URL>**  
   ➤ GitHub se project ka pura copy le aata hai.
2. **cd <cloned-folder>**  
   ➤ Cloned repo folder mein jaane ke liye.
3. **git init**  
   ➤ (Agar nahi tha already initialized)
4. **git remote set-url origin <new-url>**  
   ➤ Agar remote change karna hai toh ye use karo.
5. **git push -u origin main/master**  
   ➤ Push karega code GitHub pe.

**git rebase: Alternative to merge but keeps history clean.**

git rebase branch\_name

* Use when you want **linear history**
* Clean log: avoids unnecessary merge commits
* Conflicts manually resolve karne padte hain

🧠 Use when: Feature branch ko master ke latest ke upar lana ho.

**📁 What is .gitignore in Git?**

.gitignore ek **text file** hoti hai jo batati hai Git ko: "Bhai ye files/folders ko **track mat karna** – na stage kar, na commit kar."

**🧠 Use Case – Kyun Zaroori Hai?**

* Temporary files (e.g., .log, .tmp)
* Build outputs (e.g., node\_modules/, target/, dist/)
* Sensitive info (e.g., .env, secrets.json)
* System files (e.g., .DS\_Store, Thumbs.db)

**Yeh sari cheezein repo mein nahi honi chahiye.**

**🧪 Test Karne ke Liye Steps**

1. .gitignore file banao
2. Usmein likho jo files/folders ignore karne hain
3. git status se check karo
   * Jo ignore mein likha hai wo git status mein nahi aayega

**⚠️ Important Note:** Agar koi file pehle se Git mein **tracked** hai, aur baad mein usko .gitignore mein likha, to Git phir bhi usko ignore nahi karega.

**git rm --cached filename**

**🔥 Configuration Management using Ansible**

**📌 Ansible – Overview**

Ansible ek **configuration management tool** hai jo IT automation ke liye use hota hai. Yeh infrastructure ko **configure**, **deploy**, aur **orchestrate** karne mein help karta hai.

**⚙️ Use Cases of Ansible**

1. **Provisioning** – Infra create karne ke liye (like Terraform ke jaise)
2. **Deployment** – CI/CD pipeline automate karne ke liye
3. **Network Automation** – Routers, switches manage karne ke liye
4. **Orchestration** – Load balancer, database, etc. coordinate karne ke liye

**🧠 Key Features of Ansible**

1. **Agentless**
   * Koi client agent install karne ki zarurat nahi hoti
   * Bas SSH use karke servers se communicate karta hai
2. **Declarative Language**
   * Tasks ko YAML file mein likhte hain
   * In YAML files ko **playbooks** kehte hain

📘 **YAML File Example (Playbook Sample)**

---

- name: Install and Configure NGINX

hosts: webservers

become: yes

tasks:

- name: Install NGINX

apt:

name: nginx

state: present

- name: Ensure NGINX is running

service:

name: nginx

state: started

enabled: yes

**📝 Explanation:**

* --- → YAML file ki shuruaat denote karta hai
* apt: → Ansible ka module hai jo **Debian-based systems** (jaise Ubuntu) pe package manage karta hai
* become: yes → Root privileges enable karta hai (sudo use karta hai)
* state: present → Package install hona chahiye
* state: started → Service chalu honi chahiye
* enabled: yes → System reboot ke baad bhi service chalu rahe

✅ **YAML = Yet Another Markup Language**, but used here for configuration  
✅ Ansible ke modules kaafi saare hote hain: apt, yum, service, copy, template, etc.  
✅ Ansible Tower ek GUI-based version hai jo enterprise use ke liye hota hai  
✅ Inventory file mein host machines list hoti hain jinko manage karna hai

**🧱 Role in Ansible**

* Role ek **structured format** hai jisme aap tasks, variables, templates, files, aur other resources ko group karte ho.
* Ye reusability aur modularization ke liye kaam aata hai.
* Roles ka use playbooks ko organize aur simplify karne ke liye hota hai.

🎯 **Real World Benefit**:  
Large scale projects mein roles help karte hain structure maintain karne mein — ek baar likh ke baar-baar use karo (DRY principle).

**✅ Use Cases of Ansible:**

1. Ek file ko ek server se dusre pe bhejna (file copy)
2. Docker container ya server start karna
3. NGINX ya koi bhi software install karna
4. Server upgrade/update karna (OS-level ya app-level)

**📄 Ansible vs Puppet Comparison**

| **Feature** | **Ansible** | **Puppet** |
| --- | --- | --- |
| Setup | Simple, Agentless | Complex but scalable |
| Communication | SSH based | Agent-based |
| Use-case | Quick automation, config mgmt | Large infra config mgmt |
| Language | YAML (simple, readable) | Declarative DSL (domain-specific) |

🧠 **Extra Gyaan**:

* Puppet ka use kaafi large enterprises karte hain jaha infra bahut bada hota hai.
* Ansible beginners ke liye aur lightweight automation ke liye best hai.

**🛠️ First Ansible Playbook Example**

---

- name: first basic playbook

hosts: localhost

tasks:

- name: Test Connectivity

ping:

🔍 **Explanation**:

* localhost: apne local machine pe hi test kar rahe ho.
* ping: Ansible ka built-in module hai jo check karta hai ki connection ho raha ya nahi (like health-check).

📘 **Ansible Playbook Execution:**

**▶️ How to Run an Ansible Playbook**

**Normal Run:** ansible-playbook playbook.yml

**With Inventory File:**

ansible-playbook -i inventory.ini playbook.yml

🧠 inventory.ini file mein aapke remote servers listed hote hain.

🔧 **Example: Install NGINX on Ubuntu using Ansible:**

---

- name: Install NGINX on Ubuntu

hosts: webservers

become: yes # sudo access

tasks:

- name: Installing nginx

yum:

name: nginx

state: present

- name: Start and Enable NGINX

service:

name: nginx

state: started

enabled: true

**🧾 "state: present" ka matlab kya hota hai?**

Yeh ek **package management state** hai jo yeh ensure karta hai ki package system mein install hona chahiye.

**🔁 Other Possible States:**

| **State** | **Description** |
| --- | --- |
| present | Ensure the package is installed |
| latest | Install or upgrade to the latest version |
| absent | Remove the package |

**📌 Ansible Playbook ke Important Parameters**

1. **name:** Task ka naam (Readable explanation)
2. **hosts:** Target machine ya group
3. **become:** sudo ya root access (true/false)
4. **gather\_facts:** System info collect karta hai (true by default)
5. **vars:** Variables define karte ho playbook mein
6. **roles:** Predefined roles assign karte ho (modular structure)
7. **handlers:** Aise tasks jo tabhi chalte hain jab notify kiya jaaye (like restart nginx after config change)

**💡 Real Life Scenario Use:**

Imagine kar bhai:

* Tu 50 servers pe nginx install kar raha hai
* Agar nginx config file change hoti hai, toh handler usko detect karke nginx ko restart kar deta hai automatically
* Yeh hai automation with intelligence 😎

**📘 Ansible Task-Level Parameters + Copy/File Module**

**⚙️ Task-Level Parameters in Ansible**

1. **name** – Task ka naam hota hai (readability ke liye)
2. **apt/yum** – Package installation ke liye (OS dependent)
3. **copy** – File copy karne ke liye (from controller to remote)
4. **service** – Service manage karne ke liye (start/stop/restart/enable)
5. **command/shell** – Commands run karne ke liye (shell commands bhi)

🧭 **Flow Diagram Concept Samjho:**

Main Server

|

| push updates / install something

↓

Host folder (mein IP list hoti hai)

→ IP1

→ IP2

Yeh batata hai ki Ansible controller (main server) push karta hai configuration/instructions remote hosts pe (jinhe aap IP se identify karte ho).

**❓ Why Ansible is Used?**

* To automate:
  + File transfer
  + Software installation
  + Backup and restore
  + Remote configuration

**📂 Copy Module in Ansible**

**✅ Usage:**

1. **Copy a file from controller to host(s)**
2. **Copy a full directory**
3. **Backup important files from controller to remote**

📘 **Example:**

- name: Copy config file

copy:

src: /home/user/nginx.conf

dest: /etc/nginx/nginx.conf

backup: yes

**📁 File Module in Ansible**

**➕ Use for:**

* Create file/directory
* Delete file/directory
* Change permission
* Change ownership
* Create symbolic links

🧠 **Common States in File Module:**

| **State** | **Description** |
| --- | --- |
| touch | Creates a new empty file |
| absent | Ensures file or directory is deleted |

📘 **Example:**

- name: Create empty file

file:

path: /tmp/test.txt

state: touch

**📌 Quick Summary (yaad rakhne ke liye):**

| **Module** | **Use-case** |
| --- | --- |
| apt/yum | Package install/update |
| copy | File transfer |
| file | File/directory mgmt |
| service | Start/enable/stop service |
| shell | Custom shell commands |

**📘 Ansible User Module, get\_url & Ad-Hoc Commands:**

Note: Bash Script bhi run kar sakte ho Ansible se

* Aap main server se bash script ko remote servers pe run kara sakte ho
* Isme aap cron job bhi schedule kar sakte ho using Ansible

**👤 User Module in Ansible**

* Purpose: Remote server pe naya user create karne ke liye
* Aap name, password, shell, state jaise options define kar sakte ho

📘 Example:

- name: Add user

user:

name: devuser

shell: /bin/bash

state: present

**🌐 get\_url Module**

* Purpose: Internet se koi bhi file download karne ke liye remote machine pe

📘 Example:

- name: Download file

get\_url:

url: https://example.com/file.zip

dest: /tmp/file.zip

mode: '0644'

✅ mode se file permissions set kar sakte ho (jaise chmod mein karte ho)

**⚡ Ansible Ad-Hoc Commands**

🔹 Kya hota hai?

* Jab aapko quickly koi task run karna ho bina playbook banaye
* Useful for testing, connectivity check, emergency tasks

**📘 Basic Syntax**

ansible <host/group> -m <module> -a "<arguments>"

**✅ Common Ad-Hoc Examples**

1. **Ping check karne ke liye** : ansible all -m ping
2. **Copy file from local to remote**: ansible all -m copy -a "src=/home/user/file.txt dest=/tmp/file.txt mode=0644"
3. **Create empty file using file module** : ansible all -m file -a "path=/tmp/test.txt state=touch"
4. **Delete file**: ansible all -m file -a "path=/tmp/test.txt state=absent"

**📌 Summary – Final Touch**

| **Module** | **Use-case** |
| --- | --- |
| **user** | Add/delete user on remote machine |
| **get\_url** | Download file from internet |
| **ad-hoc** | Quick command execution (no playbook) |
| **file** | Create/delete/change file permissions |
| **cron** | Schedule tasks on remote via Ansible |

**📌 Ansible Ad-Hoc Commands – Idempotent vs Non-Idempotent**

❌ Ad-hoc commands are NOT idempotent by default

* One-time execution ke liye hote hain
* System state ya memory maintain nahi karte
* Same command dobara run karne pe yeh nahi check karte ki pehle se kaam ho chuka tha ya nahi

**🧠 Idempotent ka matlab kya?**

Jab koi task multiple baar run ho, fir bhi system state mein koi farak na pade after first run.

✅ Playbooks idempotent hote hain isiliye production mein ad-hoc ki jagah playbooks hi use hote hain.

**🏷️ Tags in Ansible**

**🔹 Purpose:** Tags allow karte hain specific tasks run karne ke liye playbook ke andar  
  
⚡ Efficient execution by skipping unnecessary tasks

**🔧 Tag Commands:**

| **Command** | **Description** |
| --- | --- |
| **--tags** | Sirf specific tag wale tasks execute karo |
| **--skip-tags** | Kuch tasks ko skip karo |
| **--list-tags** | Sab tags dekhne ke liye |

**🧪 Example:**

**ansible-playbook my.yml -t tag-name**

**📘 Use-case:** Maan le my.yml mein 10 tasks hain, par tujhe sirf restart-nginx tag wala task run karna hai → toh bas -t restart-nginx use kar!

**⚡ Tags ka use kab must hai?**

* Jab playbook badi ho
* Jab tu production mein sirf patch apply ya config reload karna chahta ho bina full playbook run kiye
* CI/CD pipelines mein jab deploy sirf backend/frontend karna ho separately

**🔁 Variables in Ansible**

**🧠 Purpose:**

Variables allow karte hain **dynamic values** dena playbook mein, taaki automation aur **flexible aur reusable** ban jaye.

- name: Install Web Server

hosts: all

become: yes

vars:

new: nginx

tasks:

- name: Install web server

apt:

name: "{{ new }}"

state: present

📌 **{{ new }}** → yeh hai variable jisme value nginx store hai

🧠 **Extra Tip:**  
Variables aap vars\_files, group\_vars, host\_vars, aur inventory mein bhi define kar sakte ho.

**✅ Conditions in Ansible (Using when)**

**📘 Purpose:** Tasks ko **conditionally execute** karne ke liye when ka use hota hai.

- name: Install Nginx only on Ubuntu

hosts: all

become: yes

tasks:

- name: Install Nginx

apt:

name: nginx

state: present

when: ansible\_distribution == "Ubuntu"

🔍 **Explanation:**

* ansible\_distribution ek **fact** hai jo system ka OS batata hai
* Agar OS Ubuntu hai, toh NGINX install hoga, warna **skip** ho jaayega

**🧠 Extra Gyaan:**

| **Use** | **Syntax** |
| --- | --- |
| Multiple conditions | when: os == "Ubuntu" and version == "20.04" |
| Check variable | when: my\_var is defined |
| With loops | when: item != "skip" inside loop |

**🔁 Loops in Ansible**

**📌 Kab use hota hai?**

Jab tu **multiple items pe same task repeat** karna chahta hai, like ek saath multiple users banana — tab loop ka use hota hai.

**📘 Example: Create multiple users**

- name: User Management

hosts: all

tasks:

- name: User Creation

user:

name: "{{ item }}"

comment: New user adding for QA team

shell: /bin/bash

loop:

- raju

- shyam

- anuj

🧠 {{ item }} — ye special variable hai jo loop ke andar ek-ek value leke task execute karta hai.

✅ Yehi logic tu packages, services, files pe bhi laga sakta hai — very powerful 🔥

**📦 Ansible Roles – Reusability ka King**

**📌 Kya hai Role?**

Role ek structured tarika hai **code ko organize** karne ka — taaki reusability aur maintainability bani rahe.

**📘 Why use Role?**

1. ✅ Reusability – Baar baar likhne ki zarurat nahi
2. ✅ Scalability – Badi team ya infra mein easily samajh aata hai
3. ✅ Maintainability – Folder-wise breakdown hota hai

**🧱 Typical Role Structure:**

my\_role/

├── tasks/

│ └── main.yml

├── handlers/

│ └── main.yml

├── templates/

├── files/

├── vars/

│ └── main.yml

├── defaults/

│ └── main.yml

├── meta/

│ └── main.yml

├── tests/

│ └── test.yml

└── README.md

🔍 **Note:**

* tasks/main.yml — yahin se execution start hota hai
* vars/, defaults/ — variables define karne ke liye
* files/, templates/ — remote machine pe bhejne ke liye

**💡 Bonus Tip:**

✅ Role ko use karne ke liye tu bas playbook mein likhe:

roles:

- my\_role

**📦 Creating a Role in Ansible**

**🧰 Command to create role:**

ansible-galaxy init my\_role

🔸 Isse ek **proper folder structure** ban jata hai jisme:

my\_role/

├── tasks/

├── handlers/

├── templates/

├── files/

├── vars/

├── defaults/

├── meta/

├── tests/

└── README.md

**🌐 Hosting a Website with Ansible**

**💡 Yes, you can use Ansible to:**

1. Install Apache
2. Deploy a website file (HTML/zip/anything)
3. Configure virtual host
4. Open port 80 in firewall

🎯 Ye sab kaam tu ek role ke andar daal ke **idempotent, reusable automation** bana sakta hai.

**🛒 Ansible Galaxy**

* Ye ek **marketplace** hai jahan se tu ready-made roles **download** kar sakta hai ya khud ke roles **upload** kar sakta hai.

📘 Example: ansible-galaxy install geerlingguy.apache

Yeh role automatically download ho jayega and tu use kar paayega.

**🧑‍💻 Ansible Tower (Now part of RedHat Ansible Automation Platform)**

**📌 What is it?**

* Web-based GUI + REST API to manage Ansible Playbooks
* Useful in **enterprise use-case** jahan team mein multiple log kaam kar rahe ho

**🛠️ Integrations:**

1. **GitHub** – SCM integration
2. **Jenkins** – CI/CD pipeline
3. **AWS** – Cloud resource automation

**🆕 Unique & Useful Points:**

**🔄 Pull vs Push Process**

* **Chef/Puppet:** Use **Pull Process** → system khud jaake server se config fetch karta hai
* **Ansible:** Use **Push Process** → controller se config push hota hai servers pe

**🧠 Infrastructure as Code (IaC)**

* Ansible supports **IaC** — yaani infra ko **code ki tarah manage** karna (version control, repeatability etc.)

**🧪 Ansible is written in Python**

* Ansible ki backend scripting Python mein hoti hai (Agar tu customize karna chahe toh Python help karega)

**❌ Agentless Nature (vs Chef/Puppet)**

* Chef uses **chef-client model** (requires agent)
* Puppet works in **slave-master model**
* But **Ansible is agentless** — sirf **SSH** ke through kaam karta hai (simple aur lightweight)

**📂 Important Ansible Config Files**

**🧾 Inventory File (Host List) :** vi /etc/ansible/hosts

* Yahin pe aap apne nodes ke **IP addresses/group names** define karte ho
* [demo] → group name example
* Group ke andar jitne bhi IPs hongi, unko ek jagah pe manage kar sakte ho

**⚙️ Ansible Config File:** vi /etc/ansible/ansible.cfg

* Yahan se **default settings** modify ki ja sakti hain

🧠 Tip: # se commented lines hoti hain, uncomment karke modify karni padti hai

**🔐 Give Sudo Access to Ansible User (Without Password)**

**To allow ansible user to run commands as root (without asking password):**

Step 1: Edit sudoers file

Step 2: Add this line: ansible ALL=(ALL) NOPASSWD: ALL

**🚫⚠ No space before** =, one space after it

sudo yum install httpd -y

* Agar ye command ansible user se chal gayi bina password, toh root access mil gaya

**🛡️ Ansible Vault – Secrets Protect Karne Ka Tareeka**

**📌 Purpose:**

Sensitive data (like passwords, tokens, API keys) ko encrypted format mein rakhna using Ansible Vault.

**🔐 1. Create a New Encrypted File:** ansible-vault create vault.yml

🧠 Yeh command naya file banata hai aur input ke time password maangta hai

**✏️ 2. Edit an Encrypted File:** ansible-vault edit vault.yml

→ Sirf wahi open hoga jiska correct password doge

**🔄 3. Change Vault Password:** ansible-vault rekey vault.yml

**🔐 4. Encrypt an Existing File:** ansible-vault encrypt target.yml

→ Kisi bhi normal YAML file ko vault mein convert kar deta hai

**🔓 5. Decrypt a File:** ansible-vault decrypt target.yml

→ Wapas plain text bana deta hai (CAUTION: only when safe!)

📌 Bonus Tips:

* Vault protected files aise dikhte hain: **$ANSIBLE\_VAULT;1.1;AES256**
* Playbook run karte time password maangta hai: **ansible-playbook play.yml --ask-vault-pass**
* Best Practice: Use vault password file: **ansible-playbook play.yml --vault-password-file vault.txt**

****

**Q1: How is load balancing achieved within pods in Kubernetes?**

In Kubernetes, there are mainly two levels of load balancing:

**1️. Intra-cluster load balancing (within cluster, between pods)**

* Kubernetes uses **Services** for load balancing across pods.
* When you expose a deployment via a ClusterIP or NodePort service:
  + The service gets a virtual IP (ClusterIP).
  + Behind the scenes, kube-proxy handles traffic routing.
* **kube-proxy** uses:
  + **iptables** (older versions)
  + **IPVS** (modern versions)  
    These maintain the mapping of service IP to healthy pod IPs (Endpoints).
* So when request comes to Service IP:
  + kube-proxy picks one pod based on round-robin or other algorithms.
  + In this way, load is distributed across multiple pods automatically.

**2️. Ingress level load balancing (for external traffic)**

* If traffic is coming from outside (internet):
  + You usually use an **Ingress Controller** (like Nginx, Traefik, or ALB ingress controller).
  + Ingress controller distributes traffic across multiple pods behind the service.
  + In cloud, cloud load balancers (AWS ELB/ALB) can also work together with Kubernetes ingress.

**✅ Important point for interviews:**

* Kubernetes does not load balance *within* a single pod (since pod usually runs one containerized app).
* Load balancing happens across pods.
* Health of pods is checked via **readiness probes** to ensure only healthy pods receive traffic.

**Q2: How can I fetch all EC2 instances across all AWS accounts and regions?**

**Approach (Multi-account + Multi-region)**

👉 This is actually multi-part problem:

* Multiple Accounts → need cross-account access
* Multiple Regions → need to loop through all AWS regions

**Solution Components:**

**Step 1️: Use AWS Organizations (or AWS SSO)**

* Centralize multiple AWS accounts under AWS Organizations.
* Use a master account (or delegated admin) to assume role into other accounts.

**Step 2️: Assume Role Across Accounts**

* Use AWS STS (Security Token Service) to assume role into each account programmatically.
* You'll need IAM role with proper permissions (ec2:DescribeInstances).

**Step 3️: Loop through all regions**

* Use boto3 (Python SDK) or AWS CLI.
* Get the list of all available regions: aws ec2 describe-regions --all-regions --query "Regions[].RegionName"

**Step 4️: Fetch EC2 instances**

import boto3

# List of all AWS accounts and regions

accounts = ['account-id-1', 'account-id-2', 'account-id-3']

regions = ['us-east-1', 'us-west-1', 'eu-west-1']

for account in accounts:

# Assume role into account

sts\_client = boto3.client('sts')

assumed\_role = sts\_client.assume\_role(

RoleArn=f'arn:aws:iam::{account}:role/RoleToAssume',

RoleSessionName='FetchEC2Session'

)

credentials = assumed\_role['Credentials']

for region in regions:

ec2\_client = boto3.client(

'ec2',

region\_name=region,

aws\_access\_key\_id=credentials['AccessKeyId'],

aws\_secret\_access\_key=credentials['SecretAccessKey'],

aws\_session\_token=credentials['SessionToken']

)

instances = ec2\_client.describe\_instances()

for reservation in instances['Reservations']:

for instance in reservation['Instances']:

print(instance['InstanceId'])

✅ **Pro tip for interview:**

* This problem tests both AWS & scripting knowledge.
* Always mention: "I will automate this via boto3 or AWS SDK using AssumeRole for cross-account access and iterate regions."

**✅ Normally in EC2 you see:**

**2/2 Status Checks**

* This is the *default* AWS health check system for EC2 instances.

**These two are:**

1️. **System Status Check**

* Checks hardware, networking, hypervisor level, etc.
* Basically AWS infrastructure level check.

2️. **Instance Status Check**

* Checks the OS level inside your EC2 instance.
* Example: OS boot issues, full disk, kernel panic, etc.

👉 Agar **2/2 Pass** hai => EC2 instance fully healthy both from AWS side and OS side.

**✅ Now you are seeing 3/3 Status Check**

* **AWS EC2 officially doesn't have native 3/3.**
* 3/3 usually means:
* Either:
* Your monitoring or custom health checks are adding **an additional 3rd check**.
* This might happen if:
  + You have configured **ELB health checks** (Load Balancer health check)
  + You have some **Auto Scaling Group (ASG) health check**
  + Or you have some **custom CloudWatch alarms / agent-based checks**

**✅ Real-world example when 3rd check appears:**

| **Check** | **Layer** | **Owner** |
| --- | --- | --- |
| System Status | AWS infra level | AWS |
| Instance Status | EC2 instance level | AWS |
| Application/Service Health Check | App level (Port 80/443 etc.) | You |

* Agar tum load balancer ke through target group health check kar rahe ho, toh wo bhi pass hona chahiye.
* Isliye monitoring tools kabhi kabhi 3/3 dikhate hain — **but AWS Console officially always shows 2/2 only.**

**Question 1: What is the difference between stateless and stateful applications?**

| Stateless | Stateful |
| --- | --- |
| Session/data ko client ya external store pe save karte hain. | Session/data ko khud maintain karta hai. |
| Easy to scale horizontally. | Scaling thoda complex. |
| Example: Web server, API server | Example: Databases, Kafka, Redis |

* Stateless apps easily Kubernetes pe scale ho jati hain.
* Stateful apps ke liye StatefulSet, PVC, Persistent Volume use hota hai.

**Question 2: Difference between Deployment and StatefulSet with example**

| Deployment | StatefulSet |
| --- | --- |
| Stateless apps ke liye. | Stateful apps ke liye. |
| Pods interchangeable hote hain (no fixed identity). | Pods ko unique identity milti hai (pod-0, pod-1...) |
| Random pod replacement | Ordered pod startup & termination |
| Ex: nginx deployment | Ex: MySQL cluster |

**What is PVC (Persistent Volume Claim)?**

* PVC ek abstraction hai storage ke liye.
* Pod ke liye storage dynamically allocate karta hai.
* PV (Persistent Volume) ko claim karta hai PVC.
* Organization me mostly databases ke liye use hota

**What is SonarLint?**

* IDE plugin hai.
* Code ko live linting, bug detection karta hai.
* SonarQube ka lightweight version samajh lo.

**Difference Between DaemonSet and StatefulSet**

| **Feature** | **DaemonSet** | **StatefulSet** |
| --- | --- | --- |
| **Purpose** | Run one pod instance on *each node* | Manage stateful, ordered, and stable pod deployments |
| **Pod Identity** | No unique identity for pods | Each pod has a stable identity (pod-0, pod-1 etc.) |
| **Use Case** | Node-level tasks like monitoring, logging, network plugins | Stateful apps like databases, Kafka, Zookeeper |
| **Scaling Behavior** | Automatically runs on all nodes (scales with nodes) | Scaling is manual; pods are created/deleted in order |
| **Pod Start Order** | No order, all pods start together | Strict start/stop order maintained |
| **Storage Handling** | No persistent storage usually needed | Needs Persistent Volume Claims (PVC) for stable storage |
| **Example Apps** | - Fluentd - Filebeat - Prometheus Node Exporter | - MySQL - MongoDB - Redis Cluster |

**Question) You have an instance in the private subnet and how will you access it?**

A private subnet instance doesn’t have public IP, so you cannot directly SSH.

Access methods:

* Bastion Host (Jump Server) Approach:
  + Create a Bastion Host in the public subnet with a public IP.
  + SSH into the Bastion Host.
  + From Bastion, SSH into the private instance using its private IP.
* SSM Session Manager (Best Practice in AWS):
  + Attach required IAM role with SSM permissions to the instance.
  + Enable SSM agent on the private instance.
  + Use AWS Systems Manager Session Manager to connect directly without bastion or public IP.
  + ✅ Highly secure, no open ports required.
* VPN / Direct Connect:
  + If corporate VPN or AWS Direct Connect is configured, you can directly access private subnet via internal network.

**2️. Steps in Continuous Delivery / Deployment**1️. Code Commit:

* Developer commits code to Version Control System (e.g. GitHub, GitLab, Bitbucket).

2️. Build Stage:

* Build code (Maven, Gradle, npm etc.)
* Compile, package, and prepare artifacts.

3️. Unit Tests & Static Code Analysis:

* Run automated tests.
* Use tools like SonarQube for code quality checks.

4️. Artifact Storage:

* Upload artifacts to repository (Nexus, Artifactory, S3, etc.).

5️. Deployment Stage (CD starts here):

* Create deployment package.
* Use tools like Jenkins, GitHub Actions, GitLab CI, ArgoCD, Spinnaker etc.

6️. Environment Provisioning:

* Infra as Code via Terraform, CloudFormation, Ansible.

7️. Deploy to Staging:

* Canary or Blue/Green deployment strategies used.
* Run Integration tests, Smoke tests.

8️. Approval Step (for Continuous Delivery only):

* Manual approval before prod.

9️. Deploy to Production:

* Rollout new version gradually.
* Monitor logs, metrics, health checks.

**3️. How can Route 53 contribute to disaster recovery scenarios?**

Route 53 is AWS DNS service and helps a lot in DR:

* Failover Routing Policy:
  + If primary site fails health checks, traffic automatically routed to DR site.
* Health Checks:
  + Route 53 monitors application health and redirects traffic when failures occur.
* Weighted Routing:
  + Helps distribute traffic between active and standby sites (active-active DR).
* Latency-based Routing:
  + Routes users to nearest healthy region, minimizing downtime and latency.
* Geo DNS:
  + Route traffic based on geographical location of the user.

**1️. Difference between apt update vs apt upgrade (Ubuntu OS)**

| **Command** |  | **Meaning** |
| --- | --- | --- |
| apt update |  | Updates the package lists from repositories (metadata only). It checks what new versions are available but doesn't install them. |
| apt upgrade |  | Actually installs the updated versions of the packages that were found during apt update. |

* apt update — Refresh the package database (new versions info laata hai).
* apt upgrade — Apply the updates (actual package ko update karta hai).

**What is SIGTERM and SIGKILL in Linux?**

Both are *signals* used to control processes.

| **Signal** | **Code** | **Description** |
| --- | --- | --- |
| **SIGTERM** | 15 | Termination request. Allows process to do cleanup before exiting. |

**SIGTERM (kill -15)** → "Bhai politely band ho jaa"

**SIGKILL (kill -9)** → "Bas abhi ke abhi mar jaa" (force kill)

**✅ How You Managed Terraform State File**

I used remote backend (S3) to store the Terraform state file and DynamoDB table for state locking. This ensures team collaboration and prevents concurrent changes.

terraform {

backend "s3" {

bucket = "my-terraform-state-bucket"

key = "env/dev/terraform.tfstate"

region = "us-east-1"

dynamodb\_table = "terraform-locks"

}

**✅ How are You Managing State File Conflicts?**

I use a DynamoDB table to enable state locking. If a developer tries to apply changes while another one is working, Terraform throws a lock error. This ensures no two people can update the state file at the same time.

**✅ How Are You Integrating SonarQube with Jenkins?**

I installed the SonarQube Scanner plugin in Jenkins, configured SonarQube server credentials in Jenkins global config, and used withSonarQubeEnv block inside the Jenkins pipeline to run static code analysis.

withSonarQubeEnv('SonarQubeServer') {

sh 'sonar-scanner -Dsonar.projectKey=myapp -Dsonar.sources=.'

}

**✅ How Were You Authenticating Jenkins to Push Docker Image to Registry?**

I used Jenkins credentials (username/password or token) and Docker login before pushing:

withCredentials([usernamePassword(credentialsId: 'dockerhub-creds', passwordVariable: 'PASS', usernameVariable: 'USER')]) {

sh 'echo $PASS | docker login -u $USER --password-stdin'

sh 'docker push $IMAGE'

}

**✅ Have You Worked on Kubernetes? Deployment Strategy?**

Yes, I worked with Kubernetes and followed strategies like **Rolling Update** (default) and **Blue-Green Deployment** for safer releases.

**✅ How Are You Implementing Blue-Green Deployment?**

I create two deployments — app-blue and app-green. One is live and other is idle. I use services or ingress to shift traffic between them using label selectors or DNS update. After verifying green, I shift traffic from blue → green.

**✅ What is HPA (Horizontal Pod Autoscaler)?**

HPA automatically scales pods based on CPU/memory usage or custom metrics. I define it using a YAML file or kubectl autoscale.

kubectl autoscale deployment my-app --cpu-percent=50 --min=2 --max=10

**✅ How to Roll Back in Kubernetes to a Particular Version?**

Kubernetes keeps revision history of deployments. To roll back:

kubectl rollout undo deployment my-app --to-revision=2

Or simply:

kubectl rollout undo deployment my-app

**✅ What is StatefulSet in Kubernetes?**

StatefulSet manages stateful applications like DBs. Each pod gets a persistent identity (DNS, storage). Example: MySQL, Cassandra. Pods are ordered and have stable storage.

**✅ Types of IAM Policies in AWS**

1. **Managed Policies**
   * AWS Managed
   * Customer Managed
2. **Inline Policies**
   * Directly attached to a single user, group, or role

**✅ S3 Bucket Policies vs ACLs**

| **Feature** | **Bucket Policy** | **ACL (Access Control List)** |
| --- | --- | --- |
| Scope | Fine-grained JSON-based access | Simple user/group permissions |
| Recommended? | ✅ Yes | ❌ Deprecated/legacy |
| Control | IAM-based centralized control | Object-level and bucket-level |

**How do you implement best security policies on AWS?**

Implemented **Security Best Practices** like:

* **IAM Policies**: Followed *least privilege* model
  + Used IAM roles over long-lived IAM keys
  + Enabled MFA for root users
* **VPC Security**: Used **private/public subnets** properly
  + Security Groups for EC2
  + NACLs for subnet-level filtering
* **S3 Security**: Enabled **S3 bucket encryption**
  + Enabled **block public access**
  + Used bucket policies with condition-based access
* **Logging & Monitoring**: Enabled **CloudTrail, Config, GuardDuty**
  + Setup **AWS WAF & Shield** for DDoS protection
* **KMS for Encryption**: Used **KMS** for encrypting EBS, RDS, S3

**✅ 1) What happens if your Kubernetes resource definition is accidentally deleted?**

👉 **Explanation:**

* Agar YAML file se resource hata diya, aur dobara kubectl apply kia, toh woh resource delete ho jaega cluster se kyunki declarative model follow karta hai.
* **Next deployment:** Agar accidental deletion ke baad deploy karte ho, toh conflict ya duplication aa sakta hai agar woh resource manually create hua ho.
* **Recovery:** Backup (Velero, etcd backup) ya GitOps se restore kar sakte ho.
* **Best Practice:** GitOps use karo (ArgoCD / Flux) — saare manifests version controlled hote hain.

**✅ 2) How do you handle large-scale refactoring without downtime?**

👉 **Explanation:**

* **Rolling updates** (maxUnavailable, maxSurge tune karo).
* **Canary deployments** (gradual rollout and monitoring).
* Choti-choti PR bana ke deploy karo.
* Infra changes ke liye Feature flags use karo.
* Database changes me backward compatibility dhyan rakho.

**✅ 3) What happens if a pod fails halfway through an update?**

👉 **Explanation:**

* K8s **desired state** maintain karta hai.
* Readiness & Liveness probes ki madad se unhealthy pods ko traffic nahi milta.
* ReplicaSet naye pod banata rahega jab tak healthy pod desired replicas tak nahi pahuchte.

**✅ 4) How do you manage secrets in Kubernetes?**

👉 **Explanation:**

* Native **Kubernetes Secrets** (Base64 encoded only — not secure fully).
* External Secret Managers:
  + HashiCorp Vault
  + AWS Secrets Manager
  + Azure Key Vault
  + External Secrets Operator
* Encryption at rest (EncryptionConfiguration) + RBAC restrictions lagana zaroori hai.

**✅ 5) What happens if kubectl apply shows no changes but the cluster was modified outside Kubernetes?**

👉 **Explanation:** Drift detect nahi hota kubectl apply se.

* Drift detection tools:
  + ArgoCD drift detection
  + kubectl diff (limited)
  + External tools: Terraform state drift detection.
* Always use declarative + reconciliation controller.

**✅ 6) What happens if you delete a resource definition from your configuration?**

👉 **Explanation:**

* YAML file se remove karoge aur apply karoge toh woh resource cluster se bhi delete ho jaega.
* **kubectl delete** se bhi manually hata sakte ho.
* Protection: finalizers ya kubectl annotate --overwrite ... protection annotations.

**✅ 7) What happens if a Kubernetes API version changes between releases?**

👉 **Explanation:** Deprecated APIs hat jaate hain (jaise: extensions/v1beta1 → apps/v1).

* Upgrade ke pehle:
  + kubectl deprecations
  + pluto tool se check karo.
  + Non-prod env mein test karo.
* Release notes aur changelogs padho.

**✅ 8) How do you implement zero-downtime updates in Kubernetes?**

👉 **Explanation:**

* **Rolling updates** (RollingUpdate strategy)
* **Blue-Green deployments**
* **Canary deployments + Progressive delivery tools**: Argo Rollouts, Flagger
* **Readiness probes** enable rakhna.
* StatefulSets ke sath DB failover handle karo.

**✅ 9) What happens if you have circular dependencies in your Kubernetes manifests?**

👉 **Explanation:**

* Deployment order fail hoga (example: pod dependent on configmap, but configmap dependent on pod label selector).
* Refactor karo:
  + Helm charts me dependency graph banana.
  + Init containers ka use.
  + Retry logic lagana.

**✅ 10) What happens if you rename a resource in your Kubernetes configuration?**

👉 **Explanation:**

* Rename = Old resource delete + New resource create.
* Downtime ho sakta hai (specially Stateful resources).
* Solutions:
  + Use kubectl rollout restart
  + Preserve PVCs and StatefulSets carefully.
  + Use Helm helm upgrade --reuse-values.

**How to protect data in AWS EC2:**

* **Data at Rest**: Encrypt EBS using KMS.
* **Data in Transit**: Use HTTPS, SSH (port 22 with key), SSL/TLS.
* **Backups**: Use snapshots or AMI creation for backup.
* **IAM roles**: Prevent accidental access to sensitive data.
* **Security Groups + NACL**: Only allow trusted IPs.
* **SSM Session Manager**: Avoid SSH, use SSM to connect securely without public IP.

**Connect AWS to On-prem Servers:**

**Site-to-Site VPN**: For secure connection using AWS VGW and Customer Gateway.

**Direct Connect**: Dedicated fiber link for low-latency/high-bandwidth.

**Transit Gateway**: Can act as central router between VPCs and on-prem.

**. What is Transit Gateway & Why Use It?**

* **Transit Gateway** = Central hub for connecting **multiple VPCs + on-prem networks**.
* Benefits: Scalable, simple alternative to VPC peering, Better routing control, Can connect 1000s of VPCs.
* Use Case: You have 10+ VPCs and 2 on-prem networks → TGW will help manage traffic routing centrally.

**Lost Terraform state file — what to do?**

* Without state file, Terraform **doesn't know** what it already created.
* Options:
  + **Import the resources** back into Terraform:

terraform import aws\_instance.example i-12345678

* + Rebuild your .tf code matching the infra, import all resources.
  + Then do terraform plan to check drift.

**Terraform taint**: command kisi specific resource ko forcefully destroy and recreate karne ke liye use hota hai during next apply.  
🔹 Use-case: Jab resource corrupt ho gaya ho ya config update ke baad bhi changes apply nahi ho rahe.

**terragrunt terraform**:mTerragrunt ek thin wrapper hai Terraform ke upar.  
🔹 Benefits: DRY (Don't Repeat Yourself) config, remote state management, multi-env support.

**Terraform workspace**  
Workspaces multiple state files handle karne ke liye use hote hain, jaise dev, staging, prod.  
🔹 terraform workspace new dev → ek new workspace banata hai.

**Current Vs Desired state in terraform**  
Terraform desired state ko .tf files mein define karta hai aur current state ko .tfstate file mein rakhta hai.  
🔹 Jab terraform plan run karte ho, yeh diff calculate karta hai.

**If someone deleted terraform-managed resources in Azure**  
🔹 Run terraform plan → missing resources detect honge.  
🔹 Then terraform apply se woh recreate ho jayenge (agar config mein abhi bhi defined hain).

**Probes in K8s**

* **Liveness Probe**: Check if container is alive (restart if fail)
* **Readiness Probe**: Check if container is ready to serve traffic
* **Startup Probe**: Delay other probes during app startup

**what are the types of branching stratergy u r using**

Currently we are following Git Flow model where we have separate main, develop, feature, and hotfix branches. All new features are developed in feature/\* branches and merged into develop after code review and testing. Once the sprint completes, develop is merged into a release/\* branch and finally into main after successful testing. For critical production issues, we use hotfix/\* directly from main. This strategy helps us maintain stable releases and allows easy parallel development.

✅ **Terraform file for Azure Storage Account**

# Required Provider

provider "azurerm" {

features {}

}

**# Resource Group**

resource "azurerm\_resource\_group" "rg" {

name = "myResourceGroup"

location = "East US"

}

**# Storage Account**

resource "azurerm\_storage\_account" "storage" {

name = "mystorageaccount123" # must be globally unique

resource\_group\_name = azurerm\_resource\_group.rg.name

location = azurerm\_resource\_group.rg.location

account\_tier = "Standard"

account\_replication\_type = "LRS" # locally redundant storage

tags = {

environment = "dev"

}

**# Storage Container (optional)**

resource "azurerm\_storage\_container" "container" {

name = "mycontainer"

storage\_account\_name = azurerm\_storage\_account.storage.name

container\_access\_type = "private"

}

**✅ Q1: What is the name of the physical address of a device on a network called?**

👉 **MAC Address** (Media Access Control Address)  
Ye ek unique hardware identifier hota hai jo har network interface card (NIC) ko assign hota hai. Isse data link layer pe device identify hoti hai.

**✅ Q2: When you type in a website address in your browser, how does DNS resolve that in the background?**

👉 Step-by-step:

1. **Browser Cache Check** – Sabse pehle browser dekhta hai cache mein IP hai ya nahi.
2. **OS Cache Check** – Agar nahi mila, toh OS ke DNS cache ko check karta hai.
3. **DNS Resolver Query** – OS DNS resolver (usually from ISP) se IP address maangta hai.
4. **Root Server** – Resolver root server se TLD (.com, .org) ka address poochta hai.
5. **TLD Server** – TLD server domain ke authoritative name server ka address deta hai.
6. **Authoritative Server** – Ye final IP deta hai.
7. **IP return hota hai browser ko** – Ab browser request bhejta hai us IP pe

**Q3: Can you name some standard directories in a Linux system?**

| **Directory** | **Use** |
| --- | --- |
| /bin | Essential command binaries (like ls, cp) |
| /sbin | System binaries (like ifconfig, reboot) |
| /etc | Configuration files |
| /home | Users’ home directories |
| /var | Variable data (logs, mail) |
| /tmp | Temporary files |
| /usr | User-installed software and libraries |
| /boot | Boot loader files (like kernel) |
| /proc | Virtual filesystem showing system processes and info |

**Q4) How would you troubleshoot a full file system on a Linux server?**

1. **Check disk usage:**

df -h

(Shows which partition is full)

1. **Find large files:**

du -sh /\* 2>/dev/null

Then go deeper into big folders:

du -ah /var | sort -rh | head -20

1. **Find old logs or tmp files:**

find /var/log -type f -name "\*.log" -mtime +10

1. **Clean log files or tmp:**

sudo rm -rf /var/log/\*.gz

sudo rm -rf /tmp/\*

1. **Check orphaned docker volumes/images (if Docker used):**

docker system prune

1. **Empty Trash of users (check /home/username/.local/share/Trash)**

**How would you monitor system performance and manage processes on a Linux server?**

**top / htop – Live CPU, memory, and process monitoring**

top

htop # more colorful and readable

**ps – Show running processes:**

ps aux --sort=-%mem | head

**free -h – RAM usage:**

free -h

**uptime – Check load average:**

uptime

**✅ Q1: How would you propose a solution to implement a microservices architecture?**

Answer:

To implement a microservices architecture:

1. Service Design:
   * Each service is independent, small, and owns its own database.
   * Use REST or gRPC for inter-service communication.
   * Follow Domain-Driven Design (DDD) for boundaries.
2. Containerization:
   * Containerize each microservice using Docker.
3. Orchestration:
   * Use Kubernetes (EKS) for managing containers.
   * Deploy services as Pods, use Deployments for rollout strategy.
4. API Gateway:
   * Use AWS API Gateway or NGINX ingress controller to manage traffic.
5. Service Discovery:
   * Use CoreDNS (in K8s) or AWS Cloud Map.
6. Security:
   * Use IAM roles, Secrets Manager, TLS certs.
7. Monitoring:
   * Use Prometheus + Grafana for metrics.
   * EFK or Loki for logs.
8. CI/CD: Use Jenkins/GitHub Actions with Terraform, Helm, and ArgoCD.

**Q2: Why do you prefer to start with EKS instead of ECS?**

I prefer EKS because:

* Standardized Kubernetes: It's CNCF-compliant, portable across clouds.
* Advanced Ecosystem: We can use Helm, ArgoCD, Istio, and K8s-native tools.
* Multi-Tenancy & Namespaces: Better isolation across environments.
* Declarative Configs: Manage infra and apps as code using kubectl, kustomize, helm, etc.
* Scalability: Easier to scale horizontally with HPA, Cluster Autoscaler.

**✅ Q3: What are the capabilities of EKS that make it a better choice over ECS for this project?**

* **Open Source Flexibility:** Full access to the Kubernetes ecosystem.
* **Custom Networking:** More control over pod networking using CNI plugins.
* **Multi-Cloud Migration:** No vendor lock-in compared to ECS.
* **Advanced Scheduling & Affinity Rules:** Better pod placement control.
* **Autoscaling Support:** HPA, VPA, and Cluster Autoscaler supported.
* **Observability:** Native integration with Prometheus, FluentBit, etc.

**✅ Q4: How would you configure the Elastic Load Balancer to work against the EKS cluster?**

1. **Install AWS Load Balancer Controller** in EKS.
2. Create **Kubernetes Ingress resources** with appropriate annotations:

alb.ingress.kubernetes.io/scheme: internet-facing

1. The controller will automatically provision an **Application Load Balancer (ALB)**.
2. **Security Groups:** Attach the ALB SG to allow incoming traffic on required ports (e.g., 80/443).
3. Use path-based or host-based routing using the Ingress YAML.

**✅ Q5: In which subnets are you going to locate the ECS cluster and RDS instances?**

* **ECS (or EKS) Cluster:** In **private subnets** for security.
* **RDS Instances:** Also in **private subnets**, with no public access.
* Public Subnet is only for internet-facing components like Load Balancers or NAT Gateways.

**✅ Q6: Share your experience in handling a large number of servers and admin responsibilities**

I've managed a fleet of **200+ servers** (containers/VMs/EC2):

* **Monitoring:** Used Prometheus, Grafana, ELK Stack.
* **Automation:** Wrote Ansible playbooks and shell/Python scripts for patching, provisioning.
* **Scaling:** Used auto-scaling groups and spot instances for cost optimization.
* **Security:** Configured IAM, SSH key rotation, automated backups, and access control.
* **Deployment:** Handled rolling and blue-green deployments.

**How would you handle multiple environments (dev, staging, prod) using Terraform?**

Answer:

I follow a standard practice of isolating environments using:  
👉 separate state files,  
👉 separate workspaces (sometimes),  
👉 and full configuration isolation via directory structure & modules.

**Approach 1: Directory structure based separation**

terraform/

├── modules/ # Reusable terraform modules

│ ├── network/

│ ├── eks/

│ └── rds/

└── environments/

├── dev/

│ ├── main.tf

│ ├── variables.tf

│ └── terraform.tfvars

├── staging/

└── prod/

**Approach 2: Backend isolation**

* Use separate **state files** for each environment, usually stored in an S3 bucket.
* Example backend config:

terraform {

backend "s3" {

bucket = "my-terraform-backend"

key = "dev/terraform.tfstate" # staging/terraform.tfstate for staging etc.

region = "ap-south-1"

}

}

"By separating environments at directory, backend, and variable level — we ensure complete isolation, prevent state corruption, and allow independent deployments of dev, staging, and production."

**✅ 1. How do you monitor applications, services, logs, and key metrics in an organization?**

📌 **Tools commonly used:**

* **Prometheus + Grafana** – For metrics collection and dashboards
* **ELK Stack (Elasticsearch, Logstash, Kibana)** – For log aggregation and analysis
* **CloudWatch (AWS)** – AWS-native monitoring for EC2, Lambda, RDS, logs, alarms etc.
* **Datadog, New Relic, Zabbix** – SaaS-based monitoring tools

📌 **What we monitor:**

* CPU, Memory, Disk, Network metrics
* Application response time, error rate
* Log patterns (e.g., 5xx errors, exceptions)
* Custom business metrics (e.g., no. of orders)

**✅ 2. What are the different types of AWS services used for various use cases?**

| **Use Case** | **AWS Services** |
| --- | --- |
| Compute | EC2, Lambda, ECS, EKS |
| Storage | S3, EBS, EFS, Glacier |
| Networking | VPC, Route 53, ELB, CloudFront |
| Database | RDS, DynamoDB, Aurora, ElastiCache |
| Monitoring & Logging | CloudWatch, X-Ray, CloudTrail, GuardDuty |
| CI/CD | CodePipeline, CodeBuild, CodeDeploy, CodeCommit |
| Security | IAM, KMS, WAF, Shield |
| Infrastructure as Code | CloudFormation, Terraform |

**✅ 3. If there are multiple EC2 instances (5-10), and I need to share files across all, which AWS service should I use?**

📌 Use **Amazon EFS (Elastic File System)**

* It's a shared file system that multiple EC2 instances (within same AZ/VPC) can mount.
* Ideal for centralized configuration, shared uploads, etc.

Alternative: If sharing infrequently and with large data — consider **S3 + scripts** for copy/sync.

**✅ 4. If I’m using EFS, but it consumes more than data ex: 10 TB, and making costs high, how can I reduce the cost without impacting the data?**

📌 Solutions:

1. **Switch to EFS Infrequent Access (IA):**
   * Enable Lifecycle Management → Automatically moves files not accessed for 30 days to IA tier (cheaper).
2. **Analyze & Delete Unused Data:**
   * Use tools like du, find, or AWS EFS analytics to check stale/unneeded files.
3. **Migrate archive data to S3 Glacier or S3 IA** if real-time access is not required.

**✅ 6. If an ASG scales up to 10 instances and scales down to 5, but I want to keep a critical instance running without termination, how can I configure that?**

📌 Options:

1. **Use Instance Protection:**

aws autoscaling set-instance-protection \

--instance-ids i-1234567890abcdef0 \

--auto-scaling-group-name my-asg \

--protected-from-scale-in

Prevents that instance from being terminated by ASG scaling policy.

1. **Tag that instance + apply custom logic in lifecycle hooks (advanced).**

**Q7 ) How do I find and replace all word (e.g.,"abc" with "xyz") inside a file?**

# Using sed (inline replace)

sed -i 's/abc/xyz/g' filename.txt

-i = edit file in place  
s = substitute  
g = replace all matches in each line

**✅ 8. Write a Linux automation script to extract and count HTTP status codes from a log file every 5 minutes.**

#!/bin/bash

LOGFILE="/var/log/nginx/access.log"

OUTFILE="/var/log/status\_count.log"

while true; do

TIMESTAMP=$(date '+%Y-%m-%d %H:%M:%S')

echo "[$TIMESTAMP] Status Code Count:" >> $OUTFILE

awk '{print $9}' $LOGFILE | grep -E '^[0-9]{3}$' | sort | uniq -c >> $OUTFILE

echo "-------------------------------" >> $OUTFILE

sleep 300 # 5 minutes

done

**✅ 1. What is a Parameter Group in a database?**

A **Parameter Group** in AWS RDS or other managed DB services is like a **configuration file** that contains **runtime parameters** for the database engine (e.g., MySQL, PostgreSQL).  
It controls things like:

* Memory allocation
* Timeout settings
* Log configurations

**✅ 1. Which command to dump and import a database?**

For **MySQL**:

* **Dump (Export):**

mysqldump -u username -p database\_name > backup.sql

* **Import:**

mysql -u username -p database\_name < backup.sql

For **PostgreSQL**:

* **Dump:**

pg\_dump -U username -d database\_name > backup.sql

* **Import:**

psql -U username -d database\_name < backup.sql

**✅ 2. How to hide EFS storage?**

To "hide" EFS:

* **Unmount** the EFS from the EC2 instance:

sudo umount /mnt/efs

* **Restrict access** using **Security Groups** and **Mount Targets**
* Use **IAM policies** to deny access to certain users/roles
* Encrypt data and disable unused mount targets if needed

**✅ 1. Introduction - Roles and Responsibility**

**Answer:**  
I am working as a DevOps Engineer where my responsibilities include:

* Managing CI/CD pipelines using Jenkins/GitHub Actions.
* Automating infrastructure using Terraform and Ansible.
* Containerization and orchestration using Docker and Kubernetes.
* Monitoring applications using Prometheus, Grafana, and Azure Monitor.
* Managing cloud resources in AWS/Azure and optimizing cost/security.
* Writing scripts in Bash/Python to automate repetitive tasks.

**✅ 2. Docker container lifecycle**

**Answer:**

* Lifecycle stages:
  + Create: docker create
  + Start: docker start
  + Run: docker run
  + Pause: docker pause
  + Unpause: docker unpause
  + Stop: docker stop
  + Kill: docker kill
  + Restart: docker restart
  + Remove: docker rm
* Use docker ps -a to view containers at all stages.

**✅ 3. States of Docker container**

**Answer:**

* **Created**: Container is created but not running.
* **Running**: Actively executing process.
* **Paused**: Execution temporarily stopped.
* **Restarting**: Restart loop due to failure.
* **Exited**: Container stopped (normally or with error).
* **Dead**: Container exists but not usable.

**✅ Blocks of Terraform**

**Answer:**

* **Provider block**: cloud provider info.
* **Resource block**: defines infrastructure (e.g., aws\_instance).
* **Variable block**: declares input values.
* **Output block**: defines output after apply.
* **Module block**: reusable components.
* **Data block**: fetch read-only info.
* **Locals**: declare local variables.
* **Backend block**: stores state remotely.

**1) What happens if your state file is accidentally deleted?**

**Answer:**  
Terraform loses the entire record of your managed infrastructure. On the next terraform apply, it treats all resources as **new** and may attempt to recreate them from scratch. This can lead to **duplicate resources**, **data loss**, or **conflicts with existing infrastructure** if not handled carefully.

💡 *Best practice:* Always keep the state file in a **remote backend** (like S3 with versioning) for backup and team collaboration.

**2) What happens if multiple team members run terraform apply simultaneously?**

**Answer:**  
If you're using a remote backend (like S3 + DynamoDB for locking), **Terraform applies a lock** on the state file. If another person tries to apply at the same time, they’ll get a **lock error**. Without locking, **state corruption** or **inconsistent deployments** can occur.

💡 *Use state locking and role-based access to avoid conflicts.*

**3) What happens if a resource fails halfway through a terraform apply?**

**Answer:**  
Terraform will stop execution at the failed resource. Successfully created resources will remain as-is, while the failed resource may be **marked as tainted**. On the next terraform apply, Terraform will attempt to **recreate** only the failed parts.

💡 Use terraform taint manually if needed to force re-creation.

**4) What happens when AWS API rate limits are hit during a large apply?**

**Answer:**  
Terraform retries API calls with **exponential backoff**. If limits continue to be exceeded, it fails the operation, leaving a **partial deployment**. Created resources are retained, but incomplete provisioning can cause **infrastructure drift**.

💡 *Consider breaking down deployments or requesting AWS quota increases.*

**5) What happens if terraform plan shows no changes, but infra was modified outside Terraform?**

**Answer:**  
Terraform relies on its state file. If infra is changed manually, that change is not reflected until you run:

* terraform refresh (deprecated in latest versions)
* or terraform plan -refresh-only

Without this, **Terraform has no visibility of the drift**, leading to **unexpected changes** or errors later.

**6) What happens if you delete a resource definition from your .tf file?**

**Answer:**  
Terraform sees that the resource is no longer defined in code, so on terraform apply, it will plan to **destroy** it from the cloud environment.

💡 To avoid destruction:

* Use terraform state rm to remove it from the state manually
* Or use lifecycle { prevent\_destroy = true } to protect critical resources

**7) What happens if a provider API changes between Terraform versions?**

**Answer:**  
Changes in the provider’s API can cause:

* **Resource incompatibility**
* **Failed applies**
* Or need for updated syntax/config in .tf files

💡 Always read the **Terraform Provider Changelog** before upgrading.

**8) What happens if you have circular dependencies in Terraform modules?**

**Answer:**  
Terraform fails with a **dependency cycle error** during terraform plan or apply. It can’t determine the order of resource creation.

💡 Refactor your code to **break the circular logic**, often by using **depends\_on** explicitly or by separating concerns into different modules.

**9) What happens if you exceed AWS service quotas during deployment?**

**Answer:**  
Terraform will throw an **"LimitExceededException"** or similar error from the AWS API. Affected resources won’t be created, and Terraform will stop further operations.

💡 You’ll need to **increase service limits** via AWS Support and re-run terraform apply.

**10) What happens if you lose access to the remote backend storing your state?**

**Answer:**  
Terraform can't read or write the state, so **all operations (plan, apply, refresh)** will fail. Collaboration breaks down, and infrastructure changes are blocked until access is restored.

💡 Use **S3 versioning** + **backup policies** to safeguard critical state files.

**How do you use include\_role and import\_role? What is the difference?**

**💡 Use of include\_role:**

Ye runtime par role ko include karta hai. Matlab jab playbook chal rahi hoti hai, tab ye decide karta hai ki role load karna hai ya nahi. Iska fayda ye hai ki tum isko conditions ke sath use kar sakte ho (when laga ke).

Example:

- name: Example of include\_role

hosts: localhost

tasks:

- name: Include a role conditionally

include\_role:

name: myrole

when: some\_var == 'yes'

**💡 Use of import\_role:**

Ye **parse time** pe hi role ko include kar deta hai. Matlab jab Ansible playbook ko parse kar raha hota hai, tabhi role ka pura content include ho jata hai. Isliye isme when condition **task ke andar lagani padti hai**, role ke level pe nahi.

**Example:**

- name: Example of import\_role

hosts: localhost

tasks:

- name: Import a role and apply condition inside task

import\_role:

name: myrole

when: some\_var == 'yes'

**✅ Kab Kya Use Kare?**

* Agar tumhe **condition ya loop** ke according role chalani ho → include\_role
* Agar role ko hamesha chalana hi hai bina kisi condition ke → import\_role

**ek real-life analogy:**

* import\_role = fixed menu (pehle se tayar): sabko wahi milega.
* include\_role = live kitchen: jo bola jaega, wahi milega condition ke hisaab se

**What are some use cases for block and rescue in playbooks?**

Bhai block aur rescue ka use Ansible playbooks mein tab hota hai jab tumhe error handling ya group of tasks ko logically group karna ho — jaise try/catch jaisa concept programming mein hota hai, waise hi yeh Ansible mein hota hai.

**🔧 Use Case #1: Error Handling (Try-Catch Style)**

Agar koi task fail ho jaye aur tum chaho ki uske baad kuch clean-up ya alternate action ho — toh block, rescue, always use hota hai.

tasks:

- name: Try block

block:

- name: Install package

apt:

name: nginx

state: present

rescue:

- name: Log failure

debug:

msg: "Failed to install nginx"

always:

- name: Always run cleanup

debug:

msg: "Cleanup executed"

**🧠 Explanation:**

* **block: Main task(s) jo try kar rahe ho.**
* **rescue: Agar block ke andar kuch fail ho gaya toh yeh chalega.**
* **always: Ye toh har halat mein chalega — chahe error aaye ya na aaye.**

**🔧 Use Case #2: Grouping Related Tasks**

Agar tumhe logically related tasks ko ek group mein dikhana ho — jaise sab package installation ek jagah, config changes ek jagah — toh block use hota hai clarity ke liye:

**How would you handle rolling updates using Ansible?**

**📝 Answer:**  
Use the serial keyword in your playbook to update hosts in batches instead of all at once. Isse minimal downtime hota hai.

**💡 Example:**

- name: Rolling update web servers

hosts: webservers

serial: 2

tasks:

- name: Restart service

service:

name: apache2

state: restarted

**🔍 Explanation:**  
Yeh playbook har 2 servers ka batch leke unpe update karega. Baaki wait karenge jab tak pehle wale complete nahi hote. Load balancer ke sath use karoge toh downtime zero ho sakta hai!

**What is a dynamic inventory, and how is it different from a static inventory?**

**📝 Answer:**

* **Dynamic Inventory:** Inventory that is fetched *dynamically* at runtime from sources like AWS, Azure, GCP, etc. Jaise ki EC2 instances list karni ho based on tags.
* **Static Inventory:** Hardcoded list of servers in a file (inventory.ini, inventory.yaml, etc.)

**💡 Example of Dynamic Inventory (AWS):**

ansible-inventory -i aws\_ec2.yaml --list

**🧠 Extra Gyaan:**

* Dynamic inventory uses plugins like aws\_ec2, azure\_rm, etc.
* Static is good for small infra ya testing.
* Dynamic is best for cloud/large infra jahan servers change hote rehte hain.

**What is the purpose of the meta directory in roles?**

**📝 Answer:**  
The meta/ directory contains metadata for the role — jaise ki:

* Dependencies on other roles
* Author info
* Supported platforms

**💡 File:**

# roles/myrole/meta/main.yml

dependencies:

- role: common

vars:

some\_var: value

**🧠 Extra Tip:** Agar koi role use hone se pehle kisi aur role ki zarurat ho, toh usko meta mein define karna is best practice.

**How does Ansible handle dependencies between roles?**

**📝 Answer:**  
Dependencies are defined in the meta/main.yml file of a role. Ansible automatically runs those dependent roles *before* running the main role.

**💡 Example:**

# roles/app/meta/main.yml

dependencies:

- role: install\_python

- role: setup\_user

Toh jab tum app role chalaoge, pehle install\_python aur setup\_user roles run ho jayenge.

**🧠 What are Namespaces in Docker?**

Docker uses **Linux namespaces** to isolate containers from each other and from the host machine.

Soch le har container ek **mini virtual machine** jaisa hota hai. Ye possible hota hai namespaces ke wajah se.

**🧱 Types of Namespaces Docker Uses:**

| **Namespace** | **Kaam Kya Karta Hai? 😎** |
| --- | --- |
| pid | Container ka **process tree** alag karta hai. Ek container doosre ke processes nahi dekh sakta. |
| net | Har container ka **apna network interface** hota hai (jaise virtual eth0). |
| mnt (mount) | Alag **filesystem view** deta hai (apna /, /etc, etc.) |
| ipc | **Inter-process communication** ko alag karta hai. |
| uts | Container ka apna **hostname** and domain name hota hai. |
| user | Container ke andar ka **user ID mapping** host se alag hota hai. |
| cgroup | Though not a namespace, yeh **resource limitation** ke liye use hota hai (CPU, memory, etc.) |

**🎯 Why are namespaces important?**

* **Security**: Containers ek dusre ke process ya file access nahi kar sakte.
* **Isolation**: Har container apna environment samajh ke kaam karta hai.
* **Scalability**: 100 containers ek host pe run ho sakte hain bina clash ke.

**💡 Real-Life Analogy:**

Soch le tu ek **PG (host machine)** mein 10 rooms (containers) bana raha hai. Har room ke paas apna **bathroom (filesystem)**, **address board (hostname)**, **phone line (network)** hota hai. Par koi bhi room doosre ka phone ya bathroom nahi chala sakta. Yeh hi namespaces ka kaam hai. 😄

**What is the significance of #!/bin/bash at the beginning of a script?**

**Answer:** The #!/bin/bash line at the beginning of a shell script is called a shebang or hashbang. It specifies the path to the interpreter that should be used to execute the script. In this case, it indicates that the script should be run using the Bash shell. It ensures that the script runs with the correct interpreter, regardless of the user's default shell.

**How do you debug a shell script?**

**🔍 1. Use -x for line-by-line execution**

Sabse famous flag: -x  
Yeh har command ko run karne se pehle print karega — line by line trace milta hai 💥

**👉 3 Ways to Use:**

**a. While running:**

bash -x myscript.sh

**b. Inside the script (top line ke baad):**

#!/bin/bash

set -x

**c. Specific block only:**

set -x

echo "Debug this"

some\_command

set +x # debugging off

**Q: A containerised app crashes after 2 minutes of starting. How would you debug it?**

If a containerized app crashes after 2 minutes, I would first check its logs using docker logs <container\_id> to identify any runtime errors. I’d inspect health checks via docker inspect to see if it’s being marked unhealthy. If it’s still running briefly, I’d use docker exec to get inside and manually explore logs/configs. I would also review the Dockerfile, especially the ENTRYPOINT and CMD sections, to ensure the container is set up to run as a long-lived process. If needed, I’d check system-level logs for resource issues and add debug flags or restart policies to gather more information.

**A junior developer removed an unused volume, which ended up deleting persistent user data. How could this have been prevented?**

**Answer** - Label important volumes clearly, use volume naming conventions, and avoid docker volume prune in shared environments. Implement access control and backups for critical data.

**Q: You have two containers that need to communicate, but they can't reach each other. What would you check?**

If two containers can't communicate, I'd first verify that they are both connected to the same **user-defined Docker network**, which enables DNS-based container name resolution. I'd then ensure they are trying to communicate via the **correct container name and internal port**. I would also check whether the service inside the container is listening on 0.0.0.0 rather than localhost, and confirm that the target port is correctly defined and open. For more complex setups, I'd inspect Docker network configs and container logs to identify any other connectivity issues.

**“Two employees work in different shifts (10 AM–5 PM and 6 PM–2 AM). How do you provide AWS access based on timing?”**

To restrict AWS access based on employees’ working hours, I would implement time-based IAM policies using policy conditions in AWS Identity and Access Management (IAM). Specifically, I would use the aws:CurrentTime condition key with DateGreaterThan and DateLessThan operators to define access windows for each shift.

👨‍💼 Employee A: (10 AM – 5 PM IST)

* I would attach a policy allowing access only between 10:00 AM and 5:00 PM IST (i.e., 04:30 to 11:30 UTC).
* Example condition block:

"Condition": {

"DateGreaterThan": {

"aws:CurrentTime": "2025-06-13T04:30:00Z"

},

"DateLessThan": {

"aws:CurrentTime": "2025-06-13T11:30:00Z"

}

}

**Default port for DynamoDB?**

🟢 **8000 (for local DynamoDB)**  
🟢 AWS hosted service uses HTTP(S), so mostly **443**.

**NodePort range in K8s?**

🟡 Default range: **30000–32767**  
Used when you want to expose services on a node's IP address and a static port.

**How can we expose on-premise applications?**

* Use **VPN or Direct Connect** to securely connect on-prem to cloud.
* Use **Ingress controllers, NodePorts, LoadBalancers (MetalLB)** in on-prem Kubernetes.
* Setup **Reverse proxies** or **API Gateways** to expose services externally.

**Two users, same app but different latency — troubleshoot?**

* Check **network latency, bandwidth** from both user locations.
* Check **load balancer & server logs** for slow requests.
* Check **DNS resolution** issues.
* Use monitoring tools (e.g. NewRelic, Datadog) for app performance.
* Validate client device performance.