



DevOps tools



1 Linux & Shell



2 Git/GitHub



3 Jenkins



4 Terraform



5 Ansible



6 Docker



7 Kubernetes



8 AWS

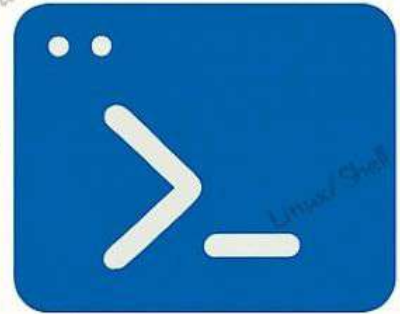


9 Prometheus + Grafana

1. Linux & Shell Scripting

Problem (Before):

People had to click around servers manually to install software, create folders, check logs, etc. Doing this repeatedly was slow, boring, and caused a lot of mistakes.



What Linux + Shell solves:

- ✓ Lets you control the whole server from the terminal
- ✓ Lets you automate things → “Do this every time, exactly the same way”
- ✓ Makes server management faster and more accurate
- ✓ Allows running the same command on multiple servers through scripts
- ✓ Base skill for everything in DevOps (Docker, Terraform, Ansible, Kubernetes—all rely on Linux concepts)

Beginner meaning:

Think of Shell like writing step-by-step instructions for the computer. Instead of clicking buttons, you tell the computer exactly what to do using commands. This makes your work fast, repeatable, and error-free.

2. Git & GitHub

Problem (Before):

Imagine three people writing in the same Word document. Everyone keeps saving over each other. You don't know who changed what, when it changed, or how to go back to an older version. If something breaks, there is no "Undo" for the whole project.



What Git/GitHub solves:

- ✓ Saves every version of your code (like a timeline)
- ✓ Tracks who made what change and when
- ✓ Allows multiple people to work safely on the same project without conflicts
- ✓ Lets you go back to any previous version if something breaks
- ✓ Stores your code online so you can access it anywhere

In simple terms:

GitHub is like Google Drive for code — but much smarter. It remembers every change, every version, and everyone who edited it. You can always go back in time if something goes wrong.

3. Jenkins (CI/CD)

Problem (Before):

To deploy new code, developers had to:

- Build the app manually
- Test it manually
- Move files manually to servers.

This took hours and caused many mistakes.



What Jenkins solves:

- ✓ Automates the entire build → test → deploy process
- ✓ Ensures the same steps run every time (no human errors)
- ✓ Triggers automatically when someone pushes new code
- ✓ Shows logs for every step so issues are easy to debug
- ✓ Works with all DevOps tools (Git, Docker, Terraform, Ansible, etc.)
- ✓ Makes deployments faster, reliable, and continuous

In simple terms:

Jenkins is like a robot that watches your code. Whenever you make a change, it automatically builds, tests, and deploys your application – without you touching anything.

4. Terraform (IaC)

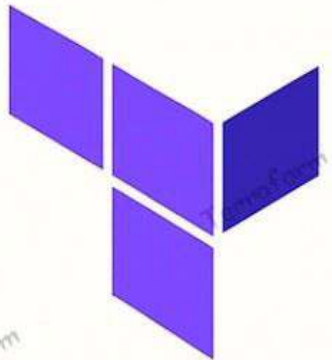
Problem (Before):

Creating servers on AWS used to be a slow, manual process.

You had to click through 10–15 screens just to create one EC2 instance.

If you needed the same setup for dev, test, and prod environments, you had to repeat the same clicks again and again.

If something went wrong, there was no easy way to recreate the exact same infrastructure.



What Terraform solves:

- ✓ Lets you write code to create servers, networks, databases, etc.
- ✓ Creates the same environment anywhere with just one command
- ✓ Makes it easy to destroy and rebuild infrastructure safely
- ✓ Tracks changes so you always know what was added or removed
- ✓ Works across all cloud providers (AWS, Azure, GCP)

In simple terms:

Instead of clicking around AWS manually, you write what you want, and Terraform builds everything for you — exactly the same every time.

5. Ansible

(Configuration Management)

Problem (Before):

After you create servers, they are empty.

- Install software
- Change settings
- Create folders
- Set up the app environment

Doing this on 10 servers, 50 servers, or 100 servers
→ takes hours and leads to mistakes.



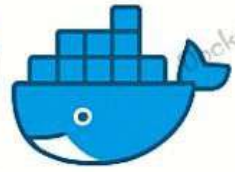
What Ansible fixes:

- ✓ Installs and configures software on many servers
- ✓ Works using simple SSH (no agents or extra setup)
- ✓ Makes sure all servers are identical with no human errors

In simple terms:

Ansible is like sending one message and all your friends' phones update at the same time — except here, the 'friends' phones' are your servers.

6. Docker (Containerization)



Problem (Before):

Apps behaved differently on every system. Why?

- Different operating systems
- Different software versions
- Missing or mismatched dependencies

So an app that worked on your laptop would fail on someone else's.

"It works on my machine, but not on yours."

What Docker Solves:

- ✓ Puts your app and everything it needs into one single package (called a container)
- ✓ Ensures your app runs *exactly* the same on any machine
- ✓ Removes environment-related issues completely

In simple terms:

Docker is like putting your app into a sealed container with all ingredients, tools, and instructions inside. Wherever someone opens that container — laptop, server, or cloud — the app works exactly the same, with no surprises.

7. Kubernetes

(Container Orchestration)



Problem (Before):

When teams started running many Docker containers, new problems appeared:

- How to restart containers when they crash?
- How to distribute user traffic across containers?
- How to scale from 5 → 50 containers during high load?
- How to update containers without downtime?

Managing all this manually caused frequent failures and long outages.

What Kubernetes solves:

- ✓ Automatically manages and monitors large groups of containers
- ✓ Restarts containers that fail
- ✓ Balances traffic across containers
- ✓ Scales up or down automatically based on load
- ✓ Enables zero-downtime updates (rolling updates)
- ✓ Keeps your application available even if some containers fail

In simple terms:

Kubernetes is a 24/7 manager for your containers — it watches them, fixes issues, balances traffic, and scales automatically so your app stays running.

8. AWS / Cloud Platforms

Problem (Before Cloud):

Companies had to buy physical servers and set them up manually.

This created many problems:



- ✓ Servers were expensive to buy and maintain
- ✓ If traffic increased, servers became slow (no easy scaling)
- ✓ If traffic decreased, servers sat unused (wasted money)
- ✓ Hardware upgrades took days or weeks
- ✓ If a server failed, the whole application could go offline
- ✓ Teams needed big IT rooms, cooling systems, and hardware engineers

Overall → slow, costly, and not flexible.

What Cloud Platforms (AWS, Azure, GCP) solve:

- ✓ Rent servers within seconds – no hardware needed
- ✓ Pay only for what you use (Huge cost savings)
- ✓ Scale up or down automatically based on load
- ✓ Global data centers → apps run fast everywhere
- ✓ Built-in security, backups, monitoring, networking
- ✓ Works seamlessly with DevOps tools (Docker, Jenkins, Terraform, Kubernetes)

In simple terms:

Cloud is like renting a computer on the internet.

You don't buy it, maintain it, or repair it —

you just use what you need, and the cloud provider handles everything else.

9. Monitoring Tools — Prometheus & Grafana

Problem (Before Monitoring):



Teams had no visibility into what was happening inside their servers or applications.

Common issues:

- ✓ Servers crashed without warning
- ✓ No idea why apps became slow
- ✓ CPU, RAM, disk usage were unknown
- ✓ No alerts when something went wrong
- ✓ Debugging took hours because issues were only found after users complained

Overall → teams were blind. They reacted after failures instead of preventing them.

What Monitoring Tools Solve:

- ✓ Track live metrics like CPU, RAM, network, database health
- ✓ Alert you immediately when something breaks or spikes
- ✓ Show performance graphs to help find issues
- ✓ Monitor container health in Docker/Kubernetes
- ✓ Detect problems early → avoid outages

In simple terms:

Prometheus = the sensor that collects all the data
Grafana = the screen that shows beautiful dashboards