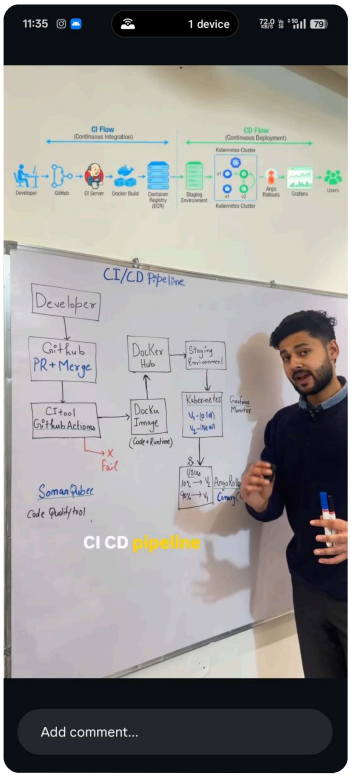

Explain step by step without missing anything.

Great — this whiteboard shows a **complete CI/CD pipeline** from code writing to users using the app. I'll explain **each block in order**, clearly and simply, without skipping anything.

🚀 What is CI/CD (quick context)

CI — Continuous Integration

👉 Automatically build & test code when developers push changes.

CD — Continuous Deployment/Delivery

👉 Automatically deploy tested code to servers so users can use it.

Goal:

✅ Faster releases

✅ Fewer bugs
✅ Automation instead of manual work

🔷 STEP-BY-STEP FLOW (Left → Right)

1️⃣ Developer

👩‍💻 A developer writes code (feature, bug fix, etc.).

Then they push code to GitHub.

2️⃣ GitHub (PR + Merge)

📦 Code goes to a GitHub repository.

Process:

1. Developer creates a **Pull Request (PR)**
2. Team reviews the code
3. If approved → **Merge to main branch**

👉 Merge triggers the CI pipeline automatically.

3️⃣ CI Tool — GitHub Actions

⚙️ This is the automation engine.

When code is merged:

GitHub Actions runs a pipeline that:

✅ Installs dependencies
✅ Runs tests
✅ Builds the project
✅ Checks code quality

🔍 SonarQube (Code Quality Tool)

Mentioned on the board.

It checks:

✔️ Code smells
✔️ Bugs
✔️ Security issues
✔️ Code coverage

If SonarQube fails ❌
➡️ Pipeline stops
➡️ Code is NOT deployed

This prevents bad code from going forward.

4️⃣ Docker Image Build

🤿 If CI passes:

The app is packaged into a **Docker Image**.

A Docker image contains:

📦 Application code
📦 Runtime (Node/Java/etc.)
📦 Dependencies
📦 Environment setup

👉 Now the app is portable and consistent everywhere.

5️⃣ Docker Hub (Container Registry)

📦 Docker image is pushed to **Docker Hub** (or AWS ECR).

This acts like storage for images.

Why?

➡️ Kubernetes will pull images from here
➡️ Version control for images

Example:

```makefile
app:v1
app:v2
```

6️⃣ Staging Environment

🧪 A test environment that mimics production.

Purpose:

✔️ Test new version safely
✔️ Catch issues before real users see it
✔️ QA team testing

If staging is good → deploy to production.

7️⃣ Kubernetes Cluster

☸️ Kubernetes manages containers in production.

It handles:

✅ Scaling
✅ Load balancing
✅ Self-healing
✅ Deployment updates

Whiteboard shows:

```ini
```

```
v1 = old version
v2 = new version
```

Both can run together during rollout.

8️⃣ Argo Rollouts (Canary Deployment)

🐤 This is advanced deployment strategy.

Instead of sending new version to all users:

Example shown:

👥 Users distribution:

```scss
10% → v2 (new)
90% → v1 (old)
```

This is **Canary Deployment**.

Why?

- ✔ Test new version on small users
- ✔ Reduce risk
- ✔ Easy rollback if issues appear

If v2 works well → increase % gradually.

9️⃣ Grafana Monitoring

📊 Grafana monitors:

- ✔ Performance
- ✔ Errors
- ✔ CPU usage

✔️ Memory
✔️ Response time

If issues appear → alerts triggered.

This helps decide:

👉 Continue rollout
👉 Rollback

🔟 Users

👥 Finally users access the app.

They may receive:

✔️ Old version
✔️ New version
(depending on rollout %)

🔁 FULL FLOW SUMMARY

mathematica Copy code

```
Developer
→ GitHub PR/Merge
→ GitHub Actions (CI)
→ SonarQube Quality Check
→ Docker Image Build
→ Docker Hub Push
→ Staging Testing
→ Kubernetes Deploy
→ Argo Canary Rollout
→ Grafana Monitoring
→ Users
```

⭐ WHY THIS PIPELINE IS POWERFUL

✅ Fully automated
✅ Safer deployments
✅ Fast releases
✅ Easy rollback
✅ High reliability
✅ Production-ready system

💡 Simple Real-World Analogy

Think of it like a car factory:

Code = Car parts
CI = Quality check machine
Docker = Packaging
Kubernetes = Delivery trucks
Argo = Deliver to few customers first
Grafana = Feedback system
Users = Customers

If you want, I can also give:

✅ Interview-ready explanation
✅ Exam answer format
✅ Real project setup guide
✅ Diagram notes for revision
✅ DevOps beginner roadmap

Just tell me 👍