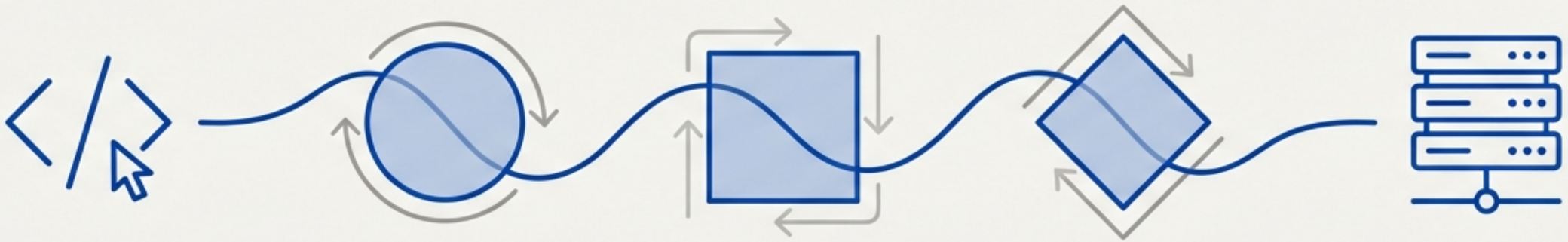
From Push to Production: Anatomy of an Automated Deployment

Deconstructing a robust CI/CD pipeline with GitHub Actions

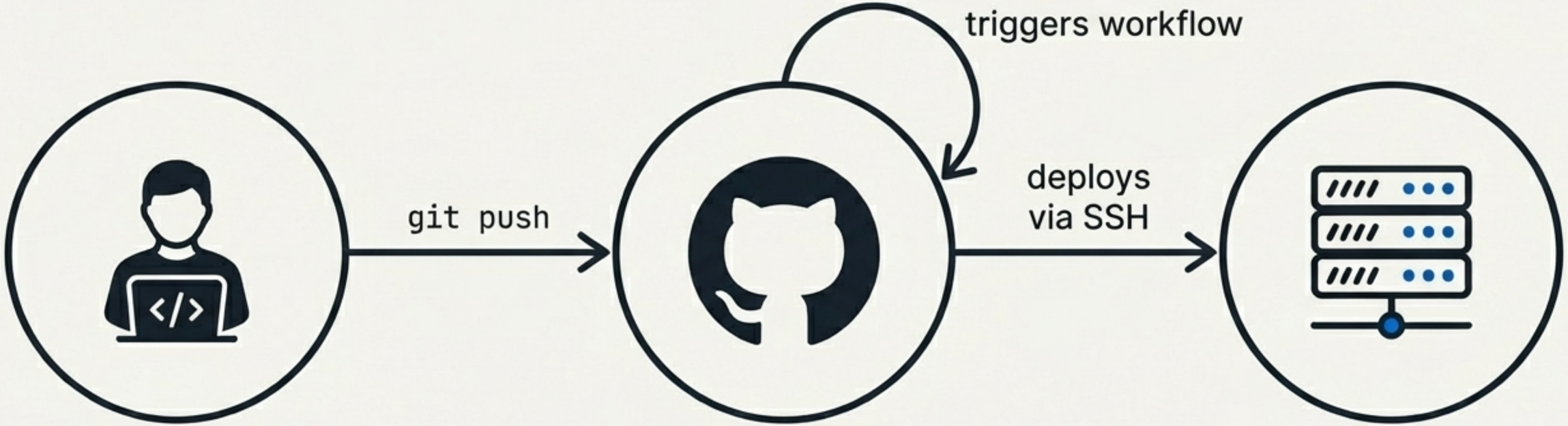


Manual deployments are fragile. Automation provides strength and speed.

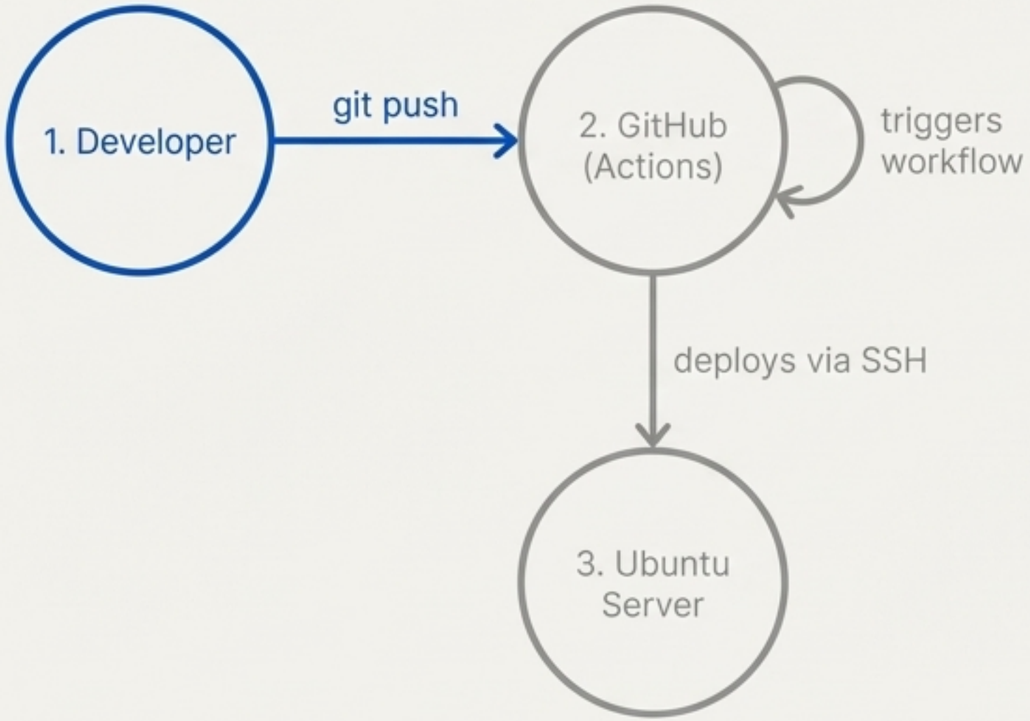
Manual

Automated

Our workflow connects three key actors in a single, automated process.

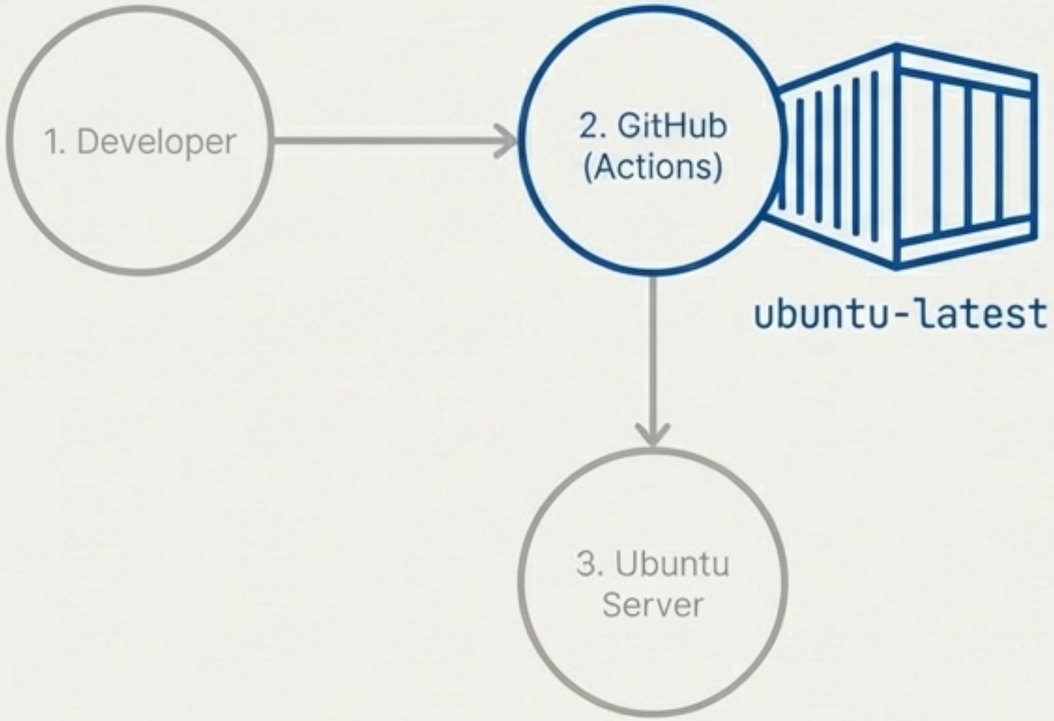


The entire workflow is triggered by a push to the `main`. branch.



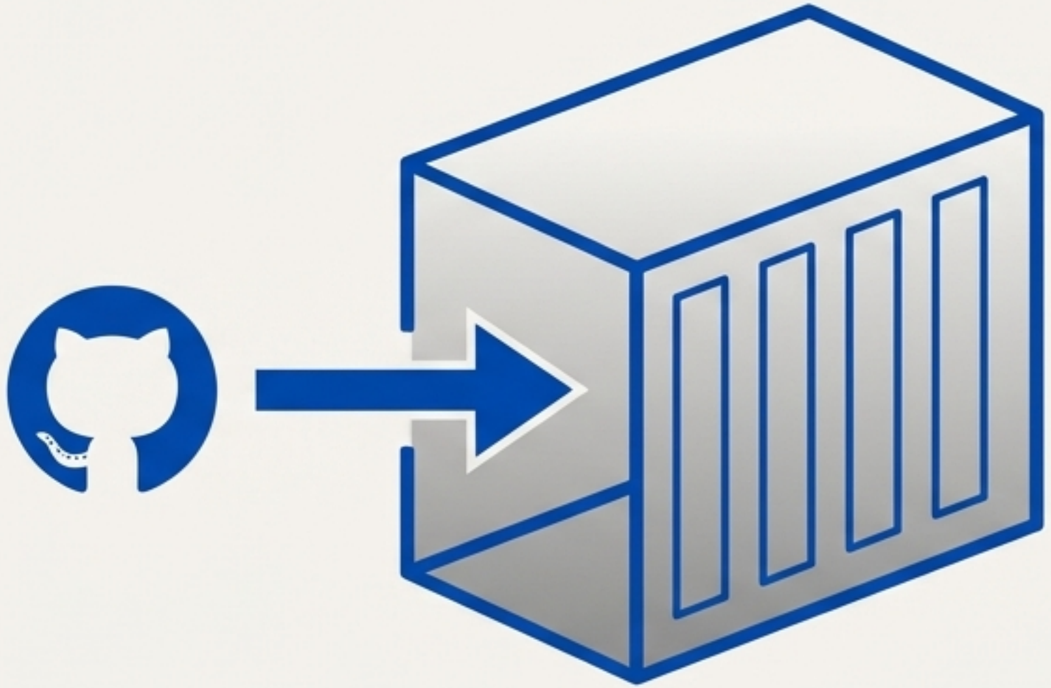

```yaml
# Trigger: Run this workflow when code is pushed
on:
  push:
    branches:
      - main
```

GitHub provides a fresh, containerized environment for every run.



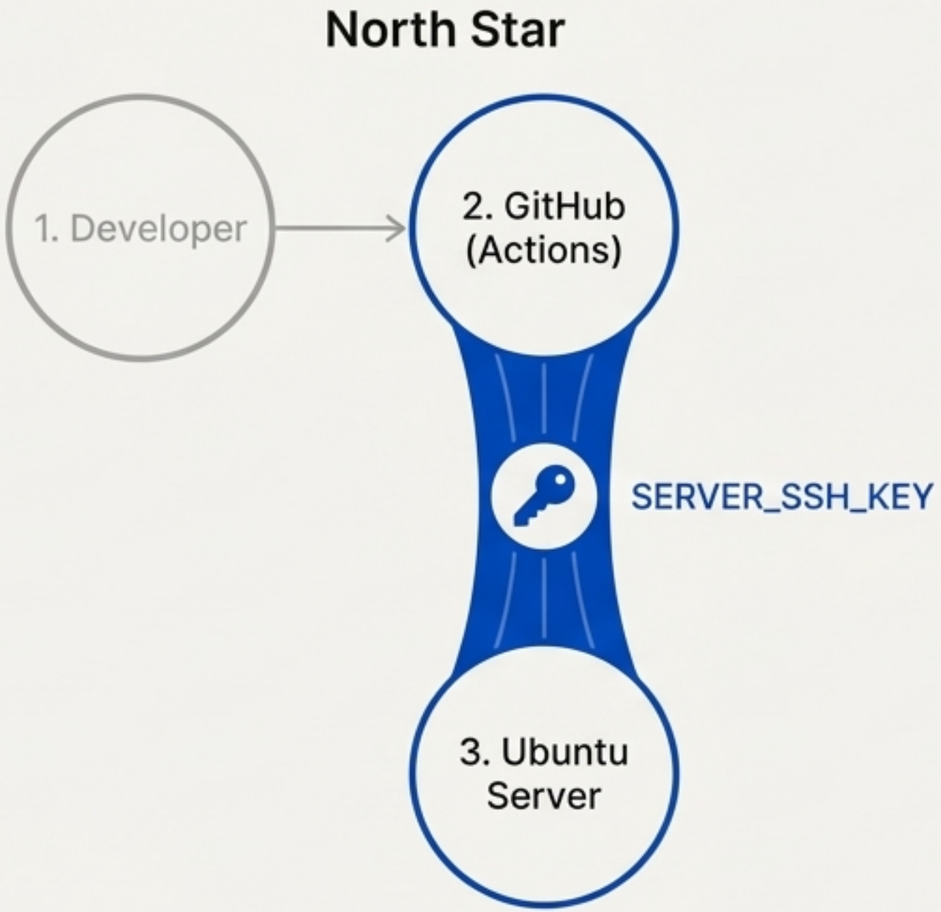

```
jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      ...
```

The first step is to check out the repository's latest code.



```
- name: Checkout code
  uses: actions/checkout@v4
```

We then establish a secure SSH tunnel to the production server.




```yaml
- name: Setup SSH Agent
  uses: webfactory/ssh-agent@v0.9.0
  with:
    ssh-private-key: ${{ secrets.SERVER_SSH_KEY }}
```

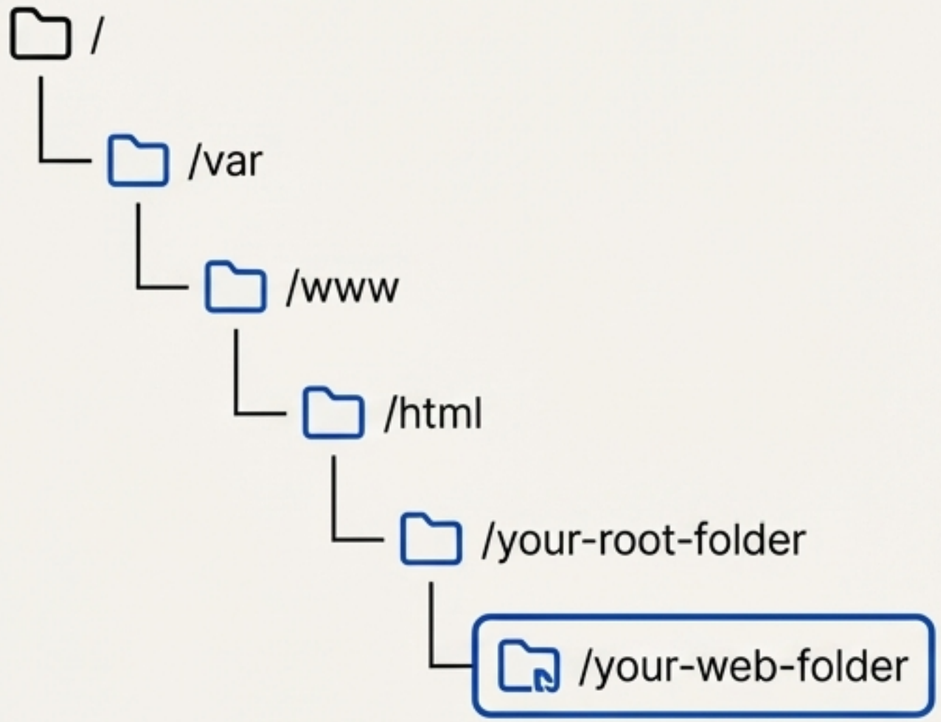
The core deployment logic runs within a single, atomic SSH session.

- All commands are sent in one block using a heredoc (<< 'EOF').
- The script is designed to be **fail-safe**, exiting immediately on any error.

```
run: |
  set -e # Exit immediately if a command fails
  set -o pipefail # Exit on pipe failures

  ssh ... << 'EOF' || exit 1
    set -e # Also set within the remote script
    ...
  EOF
```

Phase 1: The script first prepares the server environment.



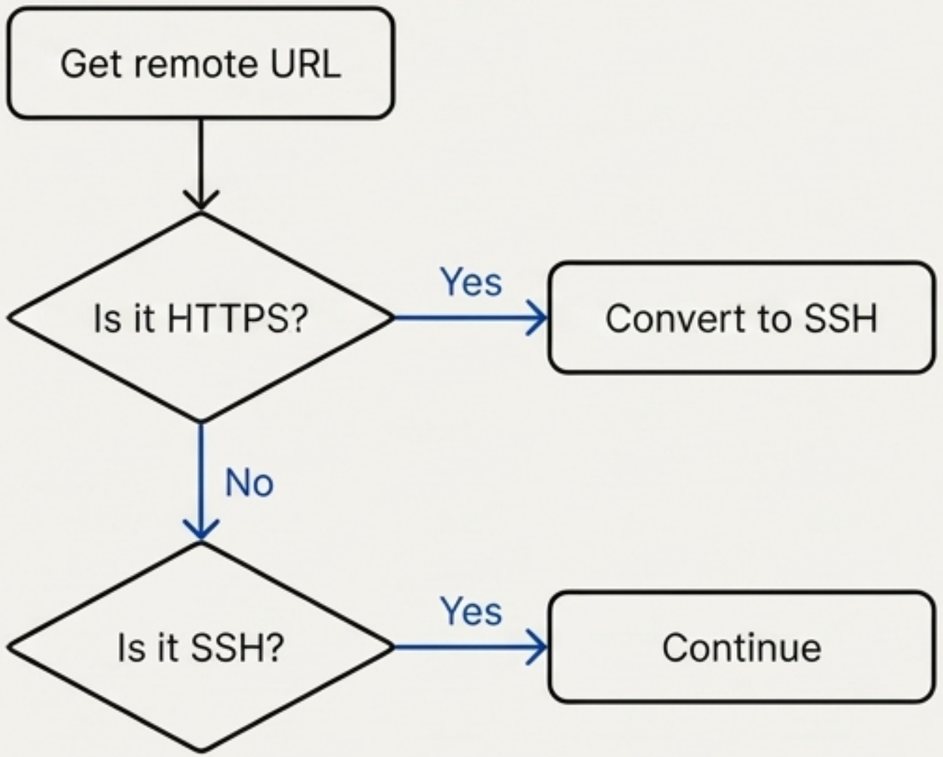

```
# Navigate to the project directory
cd /var/www/html/your-root-folder/your-web-folder
|| { exit 1 }
```

Navigates to the target deployment directory. Fails fast if the path doesn't exist.

```
# Fix ownership to prevent permission issues
echo "🔧 Fixing directory permissions..."
sudo chown -R $USER:$USER . || { exit 1 }
```

Changes ownership of all files in the directory to the current user, ensuring write permissions and preventing errors.

Phase 2: It intelligently ensures the git remote uses SSH for secure pulls.




```
echo "🔗 Configuring git remote to use SSH..."

# Logic to check and convert HTTPS remote to SSH format
# ...

echo "📥 Pulling latest code from GitHub..."
git pull origin main || { exit 1 }
```

Phase 3: With the latest code synced, we build the application.

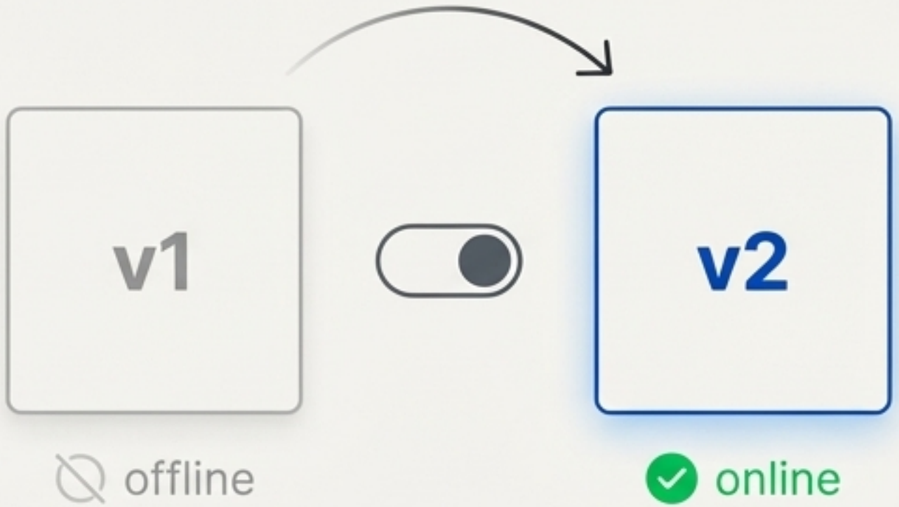
npm ci

npm run build

```
echo "📦 Installing dependencies..."
npm ci || { exit 1 }

echo "🔨 Building Next.js app..."
npm run build || { exit 1 }
```

Phase 4: Finally, we restart the application to go live with the new build.




```
echo "🔄 Restarting PM2 application..."
pm2 restart ecosystem.config.js || { exit 1 }

pm2 save || echo "⚠️ PM2 save failed, but continuing..."
```

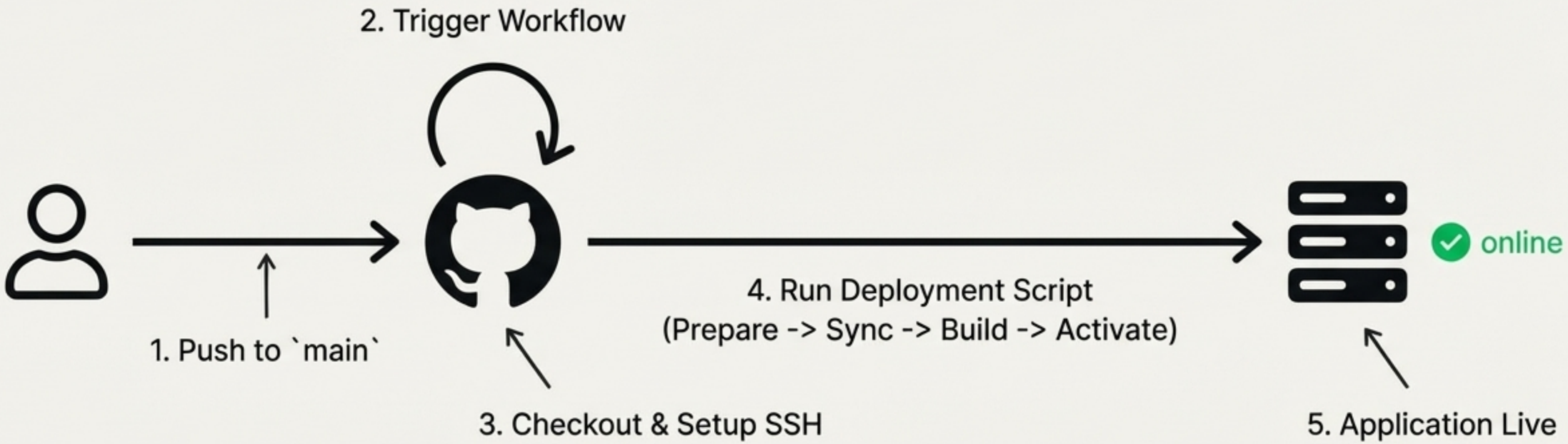
The workflow confirms success and reports the final application status.

✅ Deployment successful! 🚀

pm2 status

id	name	mode	status
0	your-app-name	cluster	online

From a single push to a live update in minutes.



This pipeline is built on three core principles of modern DevOps.

Robustness

Every critical command is checked. The `set -e` and `|| exit 1` patterns ensure that the pipeline fails fast and never leaves the server in a broken state.

Security

Sensitive credentials like the `SERVER_SSH_KEY` are never exposed in the code. exposed in the code. They are managed securely through GitHub Actions secrets.

Simplicity

The entire deployment is a single job with a clear, linear flow. The complexity is encapsulated within the script, making the workflow easy to understand and maintain.