



Expert Cloud Consulting

Enhance Optimise & Scale

ASCP GPUonCLOUD Pvt Ltd

“Expert Cloud Consulting” -

SOP | Netflix & DevOps : Case study , Architecture and Github Repository

08.August.2025

Contributed by M Bindu Sri Santhi

Approved by Akshay Sir(In Review)

Expert Cloud Consulting

Office #811, Gera Imperium Rise,

Hinjewadi Phase-II Rd, Pune, India – 411057

“Expert Cloud Consulting” Netflix & DevOps

1.0 Contents

Contents

1.0 Contents	1
2.0 General Information:	2
3.0 Document Overview:.....	3
4.0 Steps / Procedure	5
5.0 Annexure	Error! Bookmark not defined.



2.0 General Information:

2.1 Document Purpose

This document serves as a comprehensive guide for understanding and replicating DevOps practices through the case study of **Netflix**. It outlines the architectural decisions, tools, and cultural philosophies that have enabled Netflix's success in cloud-native development.

Additionally, this document details a hands-on simulation of a complete DevOps workflow using Git and GitHub. It covers:

- Repository creation and branching strategy
- Feature development with isolated branches
- Pull request creation and approval simulation
- Conflict generation and resolution during merges

The objective is to provide a real-world view of how DevOps practices such as CI/CD, automation, and version control can be integrated into a project pipeline, using Netflix as a model and GitHub as the implementation platform.

2.3 Document Revisions

Date	Version	Contributor(s)	Approver(s)	Section(s)	Change(s)
08/Aug/2025	1.0	M Bindu	Akshay Sir	All Sections	New Document Created

2.4 Document References

The following artifacts are referenced within this document. Please refer to the original documents for additional information.

Date	Document	Filename / Url
2023	Netflix case study	Case Study on Netflix A DevOps Culture - DEV Community

2023	How Netflix utilized DevOps to level up	Netflix's DevOps Journey How They Leveled Up Link
2023	Devops practices & Titus Architecture	How Netflix Became A Master of DevOps? An Exclusive Case Study
2022	GitHub Docs-Branches, PRs & Conflict Resolution	GitHub flow - GitHub Docs

3.0 Document Overview:

This document presents a comprehensive case study on how **Netflix has adopted DevOps culture and practices** to achieve scalability, high availability, and rapid software delivery. It explores the evolution of Netflix's cloud-native architecture, including the adoption of AWS infrastructure, the use of **Chaos Engineering** through tools like **Chaos Monkey**, and the deployment system **Titus** for container management.

Additionally, the document outlines the practical steps followed during a simulated DevOps project. This hands-on workflow demonstrates the application of DevOps principles such as **automation**, **collaboration**, **continuous integration**, and **continuous delivery**.



Week 2 - DevOps Principles And Version Control

Topics :

- DevOps philosophy, goals, and best practices.
- Key concepts: CI/CD, automation, collaboration.
- Version control with Git (branches, commits, pull requests).
- GitHub/GitLab workflows (forking, merging, pull requests).

Assignments:

- Analyze a real-world case study (e.g., Netflix, Etsy, or Spotify) and map their DevOps practices to key principles.
- Set up a GitHub repository, create multiple branches, and simulate a full development workflow
- Feature branches.
- Pull requests with approvals.
- Conflict resolution during merges.
- Create a detailed documentation file describing your branching strategy.

Resources:

- What is DevOps?: [AWS DevOps Guide](#)
- [Git Handbook by GitHub](#)
- Analyze a real-world case study: [Netflix DevOps Practices](#).
- GitHub workflows tutorial: [GitHub Guides](#)

4.0 Steps / Procedure

4.1 : Setup Case study on Netflix :

Netflix is a leading streaming service that has revolutionized the entertainment industry. The company has successfully implemented a DevOps culture to ensure the reliability, scalability, and fault-tolerance of its infrastructure. Here's a complete end-to-end case study of Netflix, along with the challenges it faced and how it overcame them

Background: Netflix was founded in 1997 as a DVD rental service and later pivoted to online streaming in 2007. Today, it has over 200 million subscribers in more than 190 countries. The company's streaming service runs on a cloud-based infrastructure that spans multiple regions and availability zones around the world.

Challenges: Netflix faced several challenges in building and maintaining its infrastructure, including:

Scalability: As the company grew, it needed a scalable infrastructure that could handle increasing traffic and demand for its services.

Availability: With millions of users relying on its service for entertainment, Netflix needed a highly available infrastructure that could ensure uninterrupted service.

Fault-tolerance: Netflix needed a fault-tolerant infrastructure that could withstand failures in its underlying infrastructure components, such as servers and networks.

Speed: As a streaming service, Netflix needed a fast and responsive infrastructure that could deliver content quickly to users.

Solution: To address these challenges, Netflix implemented a DevOps culture that emphasized collaboration, automation, and continuous improvement. The company's DevOps engineers are responsible for building and maintaining the infrastructure that powers its streaming service.

Here's how Netflix's DevOps team overcame the challenges:

Scalability: Netflix uses a cloud-based infrastructure that is designed to scale horizontally as demand increases. The company uses Amazon Web Services (AWS) to host its infrastructure, which allows it to quickly and easily add or remove resources as needed.

Availability: Netflix uses a distributed architecture that is designed to be highly available. The company's infrastructure is divided into several smaller services that can be scaled independently and are designed to withstand failures.

Fault-tolerance: Netflix uses a fault-tolerant architecture that is designed to handle failures in its underlying infrastructure components. The company uses tools like Chaos Monkey, which randomly shuts down servers and other components in the infrastructure to test its resiliency.

Speed: Netflix uses a content delivery network (CDN) that is designed to deliver content quickly to users. The company also uses several other techniques to optimize the delivery of content, such as adaptive bitrate streaming and caching.

Netflix also uses a variety of tools and technologies to automate and streamline its software delivery process. The company uses continuous integration and deployment (CI/CD) tools like Spinnaker to automate the deployment of its infrastructure and applications.

Results: Netflix's DevOps culture has helped the company achieve several notable results, including:

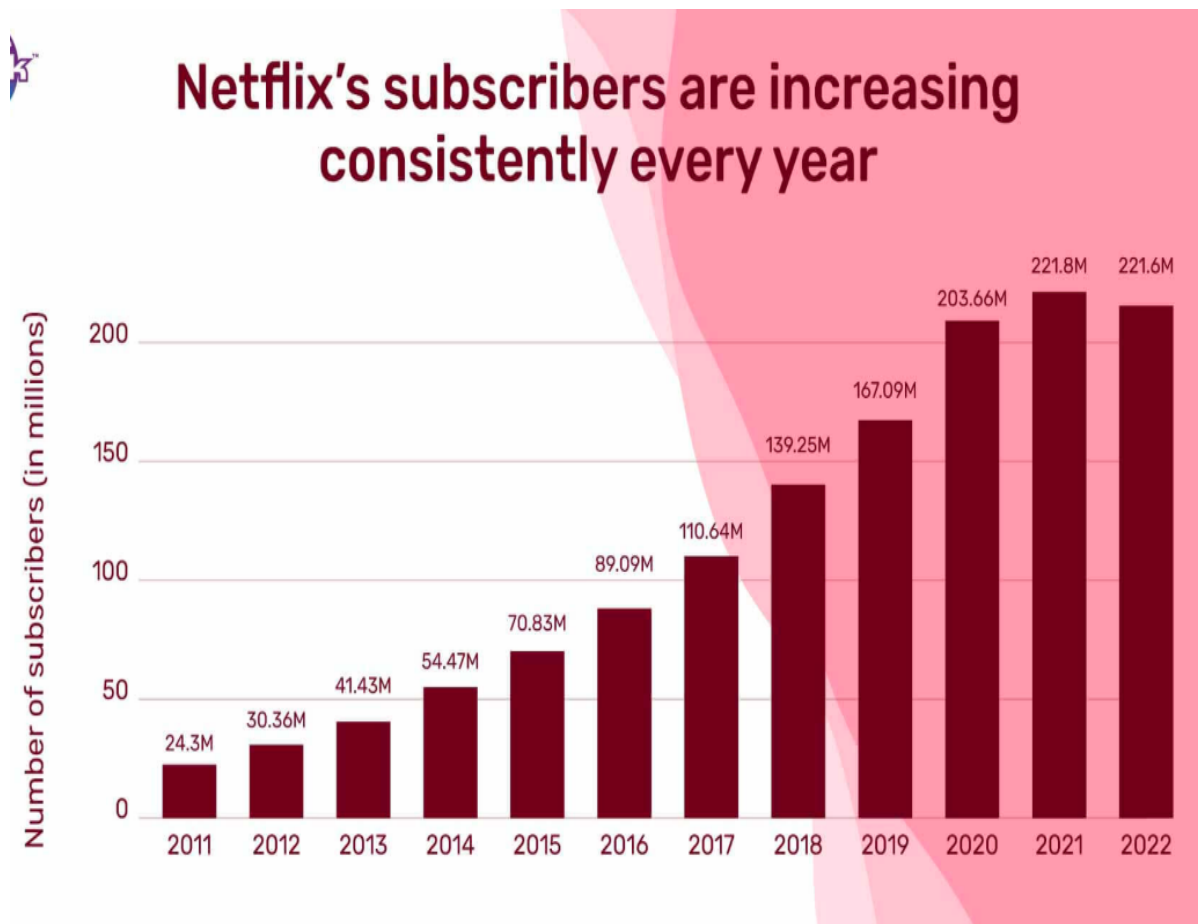


High availability: Netflix has achieved an uptime of over 99.99% for its streaming service, which is a testament to the reliability and fault-tolerance of its infrastructure.

Speed: Netflix can deliver content quickly to users, thanks to its CDN and other optimizations.

Scalability: Netflix can quickly and easily add or remove resources as needed to handle changing demand for its services.

Innovation: Netflix's DevOps culture has enabled the company to rapidly innovate and launch new features, such as the ability to download content for offline viewing.



Chaos Monkey & Simian Army: Netflix's Two Aces of Success

1. Chaos Monkey

To improve the security, reliability, and availability of its cloud infrastructure, Netflix adopted a DevOps strategy after realizing the value of frequent failure in preventing bigger catastrophes.

They were able to do this thanks to the clever development of Chaos Monkey, a program created to continuously evaluate the system's capacity to withstand unforeseen failures without impacting users. Chaos Monkey executes as a continuous script across all Netflix environments, terminating production instances and services at random inside the architecture. Chaos Monkey's implementation has proven invaluable for Netflix developers, serving multiple purposes:

- Recognizing vulnerable areas where optimization is needed.
- Progressing with automatic recovery algorithms to complement the development while addressing vulnerabilities.
- Streaming the test cases when failures occur or under similar conditions.
- Complementing the issue-resolving procedures consistently.

2. The Simian Army

Netflix engineers were motivated to increase their resilience against a wider spectrum of faults and irregularities after their success with Chaos Monkey. As a result, they created the Simian Army, a clever virtual toolkit with unique powers.

- **Latency Monkey**

This dynamic army's first soldier, Latency Monkey, simulates service degradation by adding delays to RESTful client-server communication. This enables Netflix to evaluate how upstream providers will react to certain circumstances and their capacity to do so.

They can assess the system's viability without physically pulling services offline by simulating total service outage through the use of significant delays. This was especially useful for testing new services since it allowed dependencies to fail without affecting the entire system.

- **Conformity Monkey**

Conformity Monkey, a useful instrument in the Simian Army, relentlessly searches for instances that vary from the most effective procedures and swiftly shuts them down. This move forces the service providers to appropriately relaunch these instances, guaranteeing compliance with best practices.

- **Doctor Monkey**

By using health checks and keeping an eye on external health indicators like CPU load, Doctor Monkey oversees spotting sick models. When the service owners have addressed the main problem, the discovered unhealthy instances are immediately removed from service and terminated.

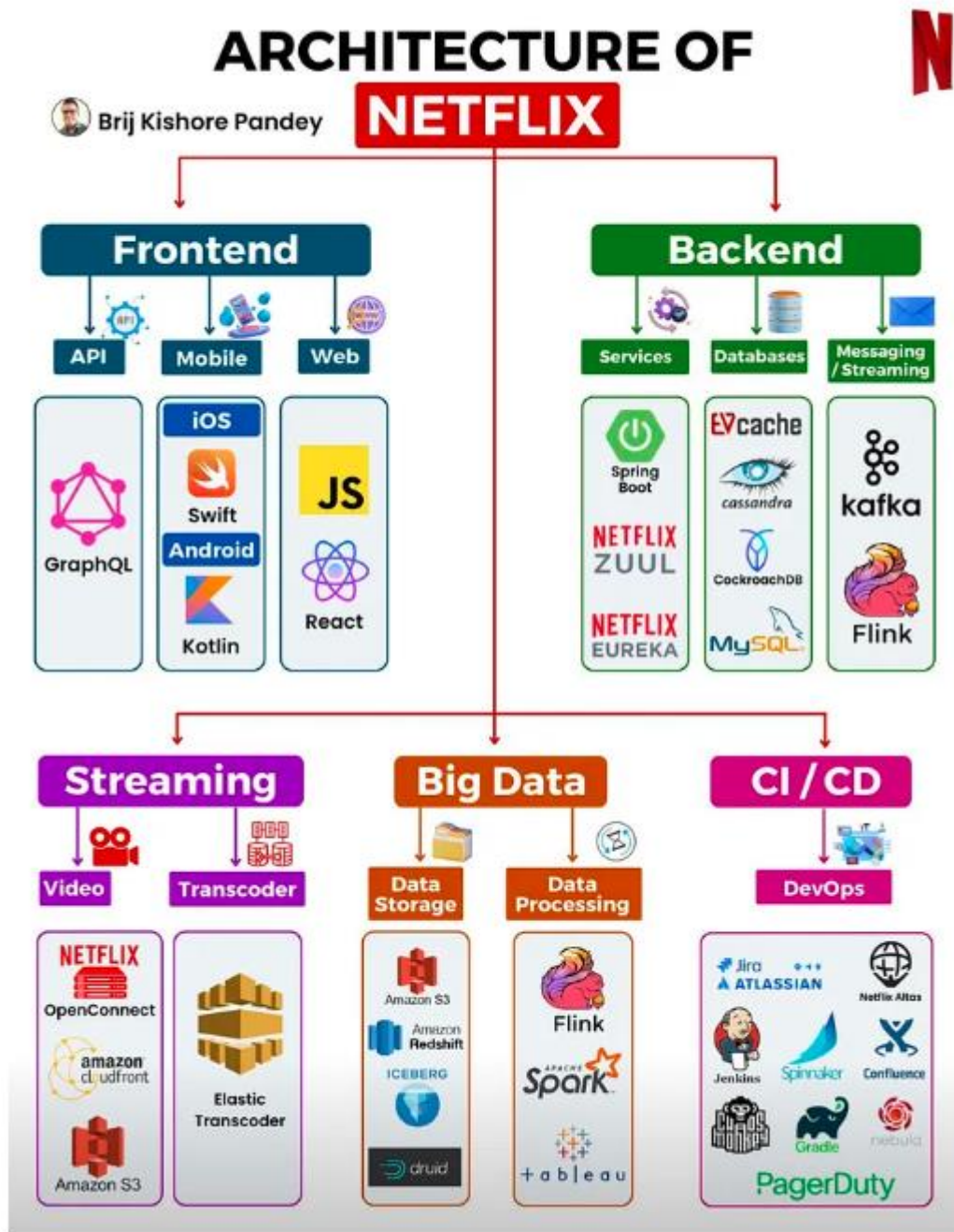
- **Janitor Monkey**

Janitor Monkey is entrusted with keeping the cloud environment clutter-free, systematically looking for and getting rid of superfluous resources, and making sure that resources are utilized to their full potential.

- **Security Monkey**

A crucial responsibility of Conformity Monkey's Security Monkey division is to spot security flaws or vulnerabilities, such incorrectly configured AWS security groups. Offending instances are swiftly removed to keep the environment secure. Security Monkey further checks the validity of SSL and DRM certificates to guarantee timely renewals as needed.

4.2: Netflix Architecture:



Frontend:

At the forefront of Netflix's user interface are various front-end technologies. The API layer is powered by GraphQL, a powerful query language that allows efficient data retrieval and flexible interactions with the backend. On the mobile front, Netflix offers dedicated applications for both iOS, built using Swift, and Android, built using Kotlin. For the web platform, Netflix relies on JavaScript, the ubiquitous language for interactive web applications, along with the popular React framework, providing a smooth and responsive user experience.

Backend:

The backend services at Netflix form the backbone of its operations. Utilizing the SpringBoot framework, Netflix ensures efficient and scalable microservices architecture. To manage service discovery and load balancing, the company relies on Netflix ZUUL and Netflix EUREKA, ensuring optimal performance and high availability.

Backend:

The backend services at Netflix form the backbone of its operations. Utilizing the SpringBoot framework, Netflix ensures efficient and scalable microservices architecture. To manage service discovery and load balancing, the company relies on Netflix ZUUL and Netflix EUREKA, ensuring optimal performance and high availability.

Database:

Netflix's massive data infrastructure demands a robust database management system. For caching, Netflix uses EVCache, an in-memory distributed cache, to optimize data access and response times. Additionally, Netflix leverages the scalability and resilience of cockroachDB and MySQL for its core database requirements, providing seamless data management.

Message/Streaming:

At Netflix, Kafka and Flink are the dynamic duo behind real-time data processing and content delivery. Kafka efficiently manages data streams, while Flink's powerful engine processes data on-the-fly, ensuring a seamless streaming experience for millions of users worldwide. Together, they enable Netflix to stay ahead in the ever-evolving world of streaming services.

Streaming:

The video streaming capabilities of Netflix are second to none. Leveraging Netflix OpenConnect, the company offers a global content delivery network (CDN) to optimize video delivery and reduce buffering. Additionally, Amazon CloudFront and Amazon S3 play essential roles in content storage and distribution. Elastic Transcoder aids in dynamic video transcoding, ensuring videos are tailored to various devices and internet speeds.

Big Data:

Netflix handles massive volumes of data to personalize content recommendations and enhance user experiences. Amazon S3 and Amazon Redshift form the foundation of Netflix's data storage and warehousing. ICEBERG and Druid, powerful distributed data systems, further enhance data processing capabilities. Spark and Flink, as data processing engines, process vast amounts of data, while Tableau empowers data visualization for valuable insights.

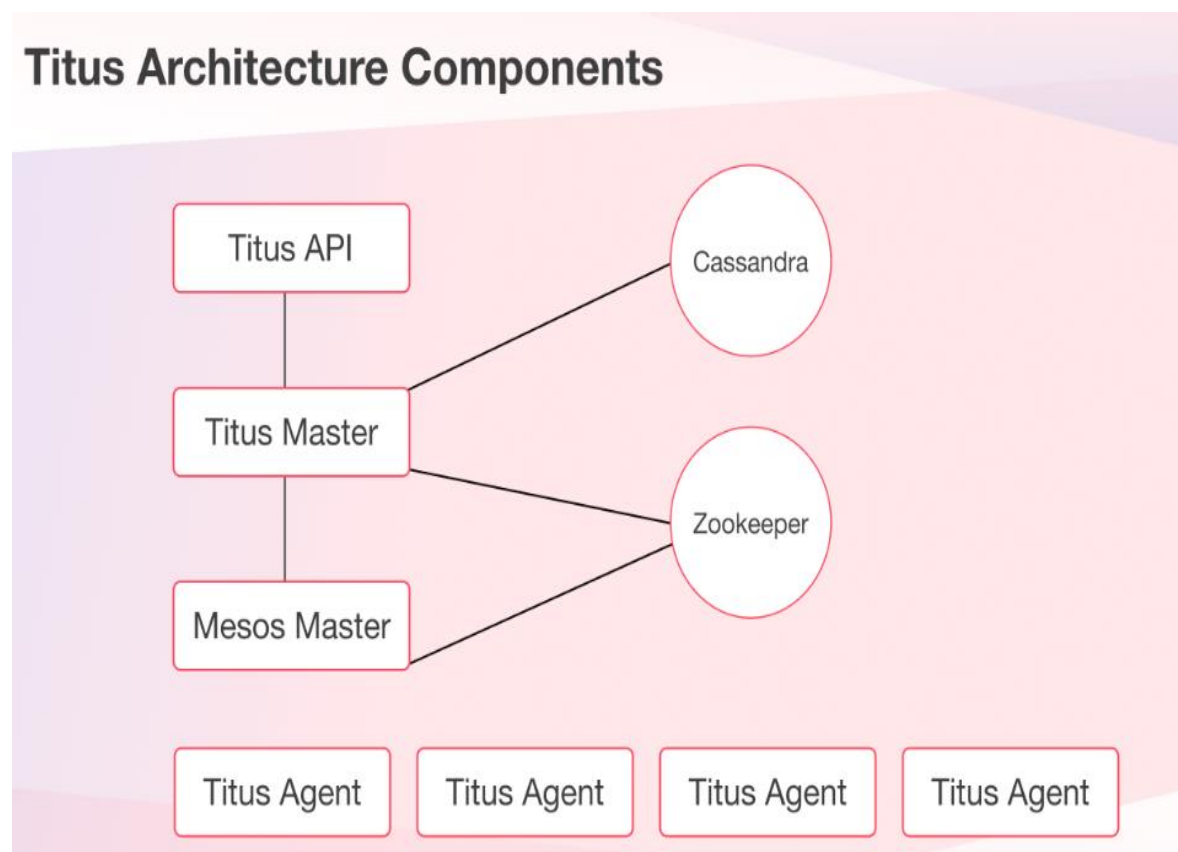
CI/CD:

To maintain a seamless development and deployment pipeline, Netflix relies on robust DevOps tools. Jira and ATLASSIAN provide project management and collaboration. Jenkins ensures continuous integration and delivery, while Spinnaker handles application deployment with ease. Gradle, Netflix Atlas, Confluence, Nebula, and PagerDuty are also key components in Netflix's CI/CD ecosystem.

Zoom image will be displayed



4.3: Titus Architecture Components:



Titus served as a standard deployment unit and a generic batch job scheduling system. It helped Netflix expand support to growing batch use cases.

- Batch users could also put together sophisticated infrastructure quickly and pack larger instances across many workloads efficiently. Batch users could immediately schedule locally developed code for scaled execution on Titus.
- Beyond batch, service users benefited from Titus with simpler resource management and local test environments consistent with production deployment.
- Developers could also push new versions of applications faster than before.

Overall, Titus deployments were done in one or two minutes which took tens of minutes earlier. As a result, both batch and service users could experiment locally, test quickly and deploy with greater confidence than before.

4.4: Implementation of full cycle developers model:

The shift to "Full Cycle Developers" has proven to be a fantastic approach, giving development teams access to effective productivity tools, and giving them full ownership of the SDLC. To encourage skill growth among new developers, Netflix supported this paradigm shift with ongoing training and assistance provided through dev boot camps.

Netflix implemented user-friendly solutions like Spinnaker, a Continuous Delivery platform, to streamline the deployment process and make it possible to release software updates quickly and confidently. The benefits of using such models are great, but dedicated development team and engineers must undergo a big mental change.

Workflow of Netflix's full cycle developers



Organizations may start by assessing their unique demands, taking the associated expenses into account, and implementing just the required complications in order to effectively deploy this approach outside of Netflix. Adopting a transformational attitude becomes essential to use this strategy effectively in any situation.

GIT & GITHUB

Introduction to GitHub

GitHub is a web-based platform used for version control and collaborative software development. It is built around Git, a distributed version control system that allows multiple developers to work on the same project efficiently. GitHub provides a central repository where code can be stored, reviewed, and managed through features like pull requests, issues, and branching.

Developers use GitHub to track changes in their codebase, collaborate with others, and ensure seamless integration of new features through version control workflows. It plays a crucial role in modern DevOps practices, supporting Continuous Integration and Continuous Deployment (CI/CD) pipelines, code reviews, and project management.

In this project, GitHub is used to manage the entire development workflow—from setting up the repository to branching, collaboration through feature branches, creating pull requests, resolving conflicts, and final merging into the main codebase.

4.4.1: Created GitHub Repository :

Created a New Repository on GitHub

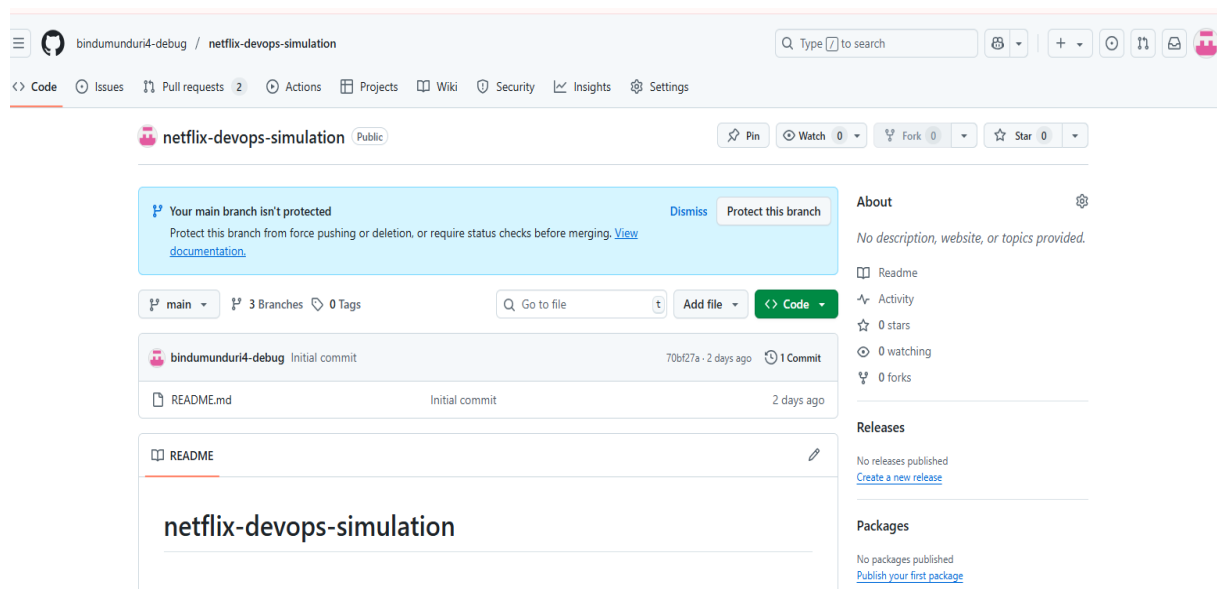
Go to <https://github.com>

Log in to your account.

Click the **+** icon in the top right corner and select **“New repository”**.

I Created New Repository

Repository name: netflix-devops-simulation



Opened the Terminal:

To Initialize Repository:

git init

Connect Local Project to GitHub Repo:

git remote add origin <https://github.com/bindumunduri4-debug/netflix-devops-simulation.git>

Add Files and Make Initial Commit:

git add .

git commit -m "Initial commit"

Push Code to GitHub:

git branch -M main

git push -u origin main

Created Multiple Branches:

```
PS C:\Users\CorpAid-Admin\Desktop\my project\netflix-devops-simulation> git branch
dev
feature/homepage
feature/seo-update
* main
```

bindumunduri4-debug / netflix-devops-simulation

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

Branches

Overview Yours Active Stale All

Q Search branches...

Default

Branch	Updated	Check status	Behind / Ahead	Pull request
main	1 hour ago		Default	

Your branches

Branch	Updated	Check status	Behind / Ahead	Pull request
feature/homepage	3 hours ago		1 0	#1
dev	3 hours ago		7 0	
feature/seo-update	2 days ago		9 0	#2



Active branches

Branch	Updated	Check status	Behind / Ahead	Pull request
--------	---------	--------------	----------------	--------------

Pull Requests with Approvals:



Git push origin feature/homepage





Git push origin feature/seo-update



 bindumunduri4-debug / netflix-devops-simulation


[Code](#) [Issues](#) [Pull requests](#) 2 [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)


Add SEO content to homepage #2


 Open bindumunduri4-d... wants to merge 4 commits into `main` from `feature/seo-update` 


 Conversation 0  Commits 4  Checks 0  Files changed 1

 Open Add SEO content to homepage #2
bindumunduri4-debug wants to merge 3 commits into `main` from `feature/seo-update` 


 bindumunduri4-debug closed this 2 days ago

 bindumunduri4-debug reopened this 2 days ago














 No conflicts with base branch
Merging can be performed automatically.

[Merge pull request](#) You can also merge this with the command line. [View command line instructions](#).

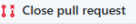
 Add a comment


Write Preview

H B I           

Add your comment here...

☐ Markdown is supported ☒ Paste, drop, or click to add files

 Close pull request [Comment](#)

Development 


Successfully merging this pull request may close these issues.


None yet


Notifications [Customize](#)

[Unsubscribe](#)

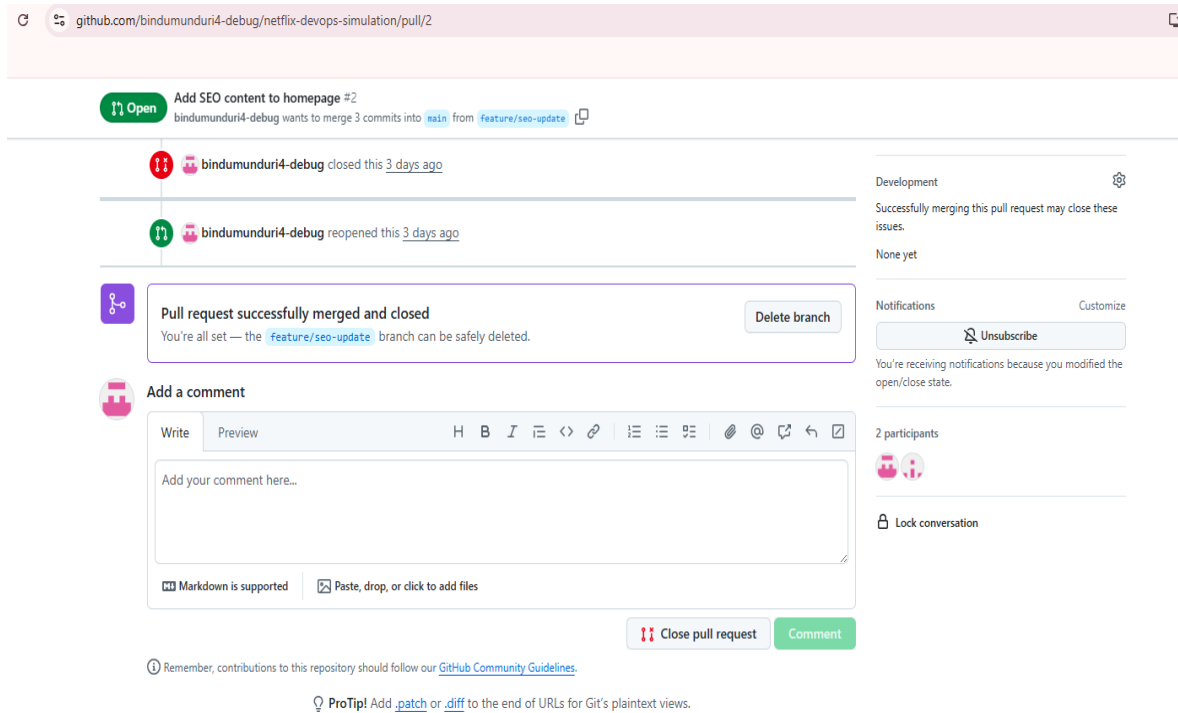
You're receiving notifications because you modified the open/close state.

2 participants 

 Lock conversation

 Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).





4.4.2: Conflict resolution during merges:

git checkout -b main c2

git add .

git commit -m "main c2"

git merge dev

git push origin main

I have created main and dev branches and merged them it merged properly

```
PS C:\Users\CorpAid-Admin\Desktop\my project\netflix-devops-simulation> git merge dev
Auto-merging test1.txt
CONFLICT (add/add): Merge conflict in test1.txt
PS C:\Users\CorpAid-Admin\Desktop\my project\netflix-devops-simulation> git merge dev
```

It successfully merged


```

70b1278 Initial Commit
PS C:\Users\CorpAid-Admin\Desktop\my project\netflix-devops-simulation> git show aedb78e
>>
commit aedb78ee5704303347b9c9dbc4f27695db74042b
Merge: 17c2ec7 dc5015a
Author: Sri <Sri@gmail.com>
Date: Thu Aug 7 16:37:02 2025 +0530

    main branch c3

diff --cc test1.txt
index 4277019,9802d2b..fbeb6dc
--- a/test1.txt
+++ b/test1.txt
@@@ -1,1 -1,1 +1,5 @@@
- master branch line number 1
- dev branch line 1
++<<<<<< HEAD
++master branch line number 1
++=====
++dev branch line 1
++>>>>>> dev

```

Now, I merged seo-update and main branch also and it also merged successfully.
I Did Commands to merge.

git checkout main

git pull origin main

git branch -d feature/homepage

git branch -d feature/seo-update

PR Request and Merge successfully:



The screenshot shows a GitHub pull request interface. At the top, the browser address bar displays 'github.com/bindumunduri4-debug/netflix-devops-simulation/pull/2'. The pull request title is 'Add SEO content to homepage #2', and it indicates that 'bindumunduri4-debug' wants to merge 3 commits into 'main' from 'feature/seo-update'. A green 'Open' button is visible. Below the title, a timeline shows the pull request was closed and then reopened by 'bindumunduri4-debug' 3 days ago. A purple notification box states 'Pull request successfully merged and closed', advising that the 'feature/seo-update' branch can be safely deleted, with a 'Delete branch' button. An 'Add a comment' section is present with a text area and a 'Write' tab. On the right sidebar, there are sections for 'Development' (with a note about merging), 'Notifications' (with an 'Unsubscribe' button), and '2 participants'. At the bottom, there are buttons for 'Close pull request' and 'Comment'. A footer note mentions GitHub Community Guidelines and a ProTip about adding '.patch' or '.diff' to URLs.

"I created a pull request from feature/seo-update to main to add SEO content. However, GitHub reported a merge conflict because both branches made changes to the same homepage.html file. I'll pull the latest from main, resolve the conflict manually, and push the fix. Once done, the PR can be merged successfully."

To see Logs:

Git log --oneline --graph --all

To see files merged:

Git log --merges

```

PS C:\Users\CorpAid-Admin\Desktop\my project\netflix-devops-simulation> git log --oneline --graph --all
>>
* fe94a6b (HEAD -> feature/seo-update, origin/main, origin/HEAD, main) main c4
* aedb78e main branch c3
| \
| * dc5015a (origin/dev, dev) Add test1.txt file in dev branch
| * c80d37a Add branching strategy documentation
| * b948ea7 (origin/feature/seo-update) Resolve conflict with Netflix Global heading
| \
| * 35b112a Update heading in homepage.html from seo-update branch
| * 1feebb8 Add SEO content in homepage from seo-update branch
| * 17c2ec7 main c2
| /
| /
| * 4f399f5 Update heading in homepage.html from MAIN branch
| * bd32962 (origin/feature/homepage, feature/homepage) Update heading in homepage.html from homepage branch
| * d3d78f7 Update heading in homepage.html from homepage branch
| /
| * f805a76 Add homepage.html with heading
| /
* 70bf27a Initial commit
PS C:\Users\CorpAid-Admin\Desktop\my project\netflix-devops-simulation> git log --merges
>>
commit aedb78ee5704303347b9c9dbc4f27695db74042b
Merge: 17c2ec7 dc5015a
Author: Sri <Sri@gmail.com>
Date: Thu Aug 7 16:37:02 2025 +0530

    main branch c3

commit b948ea7d91faf4dc68878538da511e3fd773e9c3 (origin/feature/seo-update)
Merge: 35b112a 4f399f5
Author: Sri <Sri@gmail.com>
Date: Wed Aug 6 11:41:07 2025 +0530

    Resolve conflict with Netflix Global heading
PS C:\Users\CorpAid-Admin\Desktop\my project\netflix-devops-simulation>

```

Conclusion:

In the modern DevOps landscape, GitHub plays a vital role in enabling collaboration, accelerating development, and ensuring high-quality software delivery. By offering powerful version control, streamlined branching workflows, and native CI/CD integrations like GitHub Actions, it empowers teams to build, test, and deploy with confidence. This case study demonstrated how GitHub can be effectively used throughout the DevOps lifecycle—from code commits and pull requests to conflict resolution and automation. By adopting these practices, teams not only enhance productivity but also foster a culture of transparency and continuous improvement. Whether you're a solo developer or part of a large enterprise team, mastering GitHub is essential for thriving in today's fast-paced software development environment.



