Expert Cloud Consulting
Enhance Optimise & Scale
ASCP GPUonCLOUD Pvt Ltd

“Expert Cloud Consulting”

SOP | Netflix & DevOps: A Case Study in Innovation, Culture, and Engineering Excellence

3 Jun 2025

—

Contributed by: Dhanshri Patil
Approved by : Akshay Shinde (In Review)
Expert Cloud Consulting
Office #811, Gera Imperium Rise,
Hinjewadi Phase-II Rd, Pune, India – 411057

Netflix & DevOps: A Case Study in Innovation, Culture, and Engineering Excellence

1.0 Contents

Expert Cloud Consulting
Enhance Optimise & Scale
ASCP GPUonCLOUD Pvt Ltd

2.0 General Information:

2.1 Document Purpose

This document provides a comprehensive case study on how Netflix became a leading example of DevOps implementation—not by strictly following a defined DevOps framework, but by fostering a culture rooted in innovation, automation, and developer empowerment. It explores Netflix's strategic transition to the cloud, the use of chaos engineering tools like Chaos Monkey, the adoption of containers with Titus, and the introduction of the “Operate What You Build” model.

The purpose of this document is to analyze Netflix’s unique DevOps journey, extract key lessons from its practices, and provide insights that organizations can adapt to accelerate software delivery, enhance system reliability, and improve customer experience.

2.2 Document References

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

Date	Document	Filename / Url
2.06.2025	Netflex Documentation	https://www.simform.com/blog/netflix-devops-case-study/
2.06.2025	DevOps case study- Netflex	https://contactchanaka.medium.com/devops-case-study-netflix-7fe289311c6a

Document Overview**:**

This document, prepared by Expert Cloud Consulting, provides a comprehensive guide to DevOps principles through a case study of Netflix's microservices architecture and practical implementation details of Git version control. It blends theoretical principles with hands-on instructions for setting up infrastructure and managing code using GitHub. The key highlights include:

- Analysis of Netflix's DevOps strategies.
- Git branching strategies.
- Conflict resolution techniques for collaborative development.

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

1.0 Introduction

Even though Netflix is an entertainment company, it has left many top tech companies behind in terms of tech innovation. With its single video-streaming application, Netflix has significantly influenced the technology world with its world-class engineering efforts, culture, and product development over the years.

One such practice that Netflix is a fantastic example of is DevOps. Their DevOps culture has enabled them to innovate faster, leading to many business benefits. It also helped them achieve near-perfect uptime, push new features faster to the users, and increase their subscribers and streaming hours.

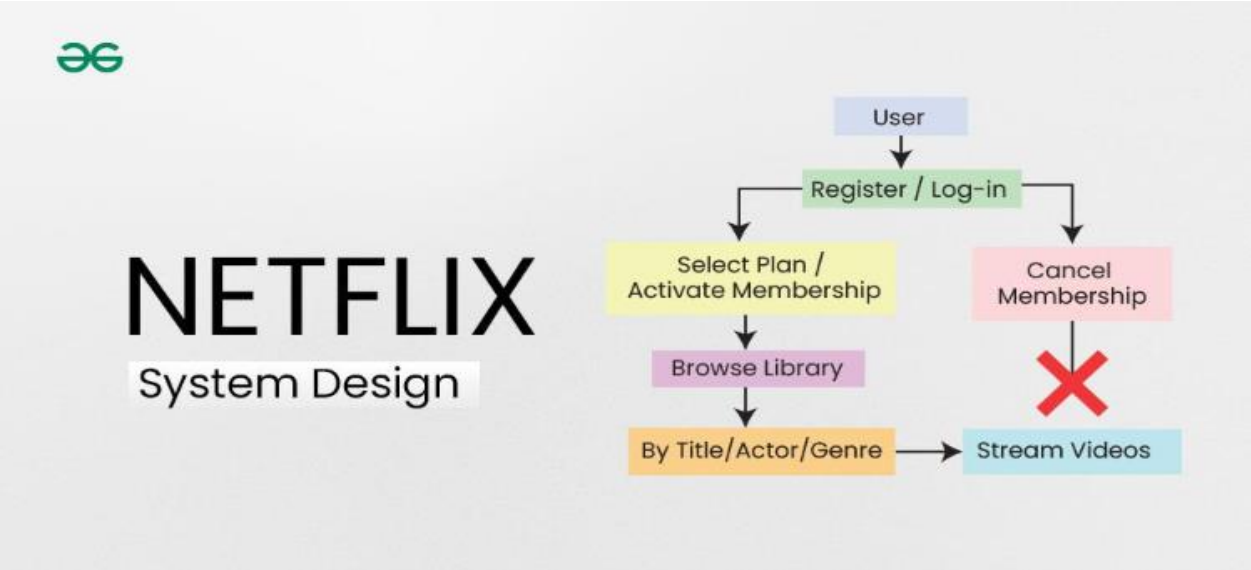
With nearly 214 million subscribers worldwide and streaming in over 190 countries, Netflix is globally the most used streaming service today. And much of this success is owed to its ability to adopt newer technologies and its DevOps culture that allows them to innovate quickly to meet consumer demands and enhance user experiences. But Netflix doesn’t think DevOps.

So how did they become the poster child of DevOps? In this case study, you’ll learn about how Netflix organically developed a DevOps culture with out-of-the-box ideas and how it benefited them.

1.System Design Netflix | A Complete Architecture

Designing Netflix is a quite common question of system design rounds in interviews. In the world of streaming services, Netflix stands as a monopoly, captivating millions of viewers worldwide with its vast library of content delivered seamlessly to screens of all sizes. Behind this seemingly effortless experience lies a nicely crafted system design. In this article, we will study Netflix's system design.

In this article we have covered the system design of Netflix but if you wish to learn system design from the scratch and want to learn how things work in these high volume app then you should enrol in our system design course.



1. Requirements of Netflix System Design

1.1. Functional Requirements

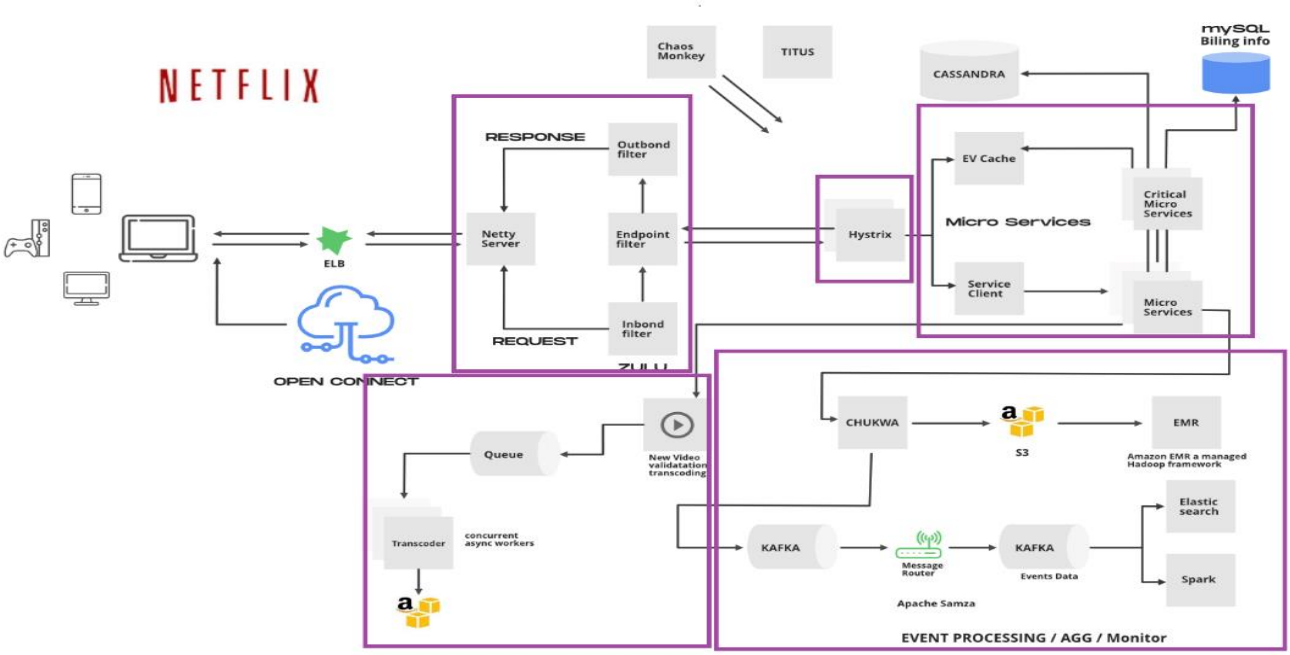
- Users should be able to create accounts, log in, and log out.
- Subscription management for users.
- Allow users to play videos and pause, play, rewind, and fast-forward functionalities.
- Ability to download content for offline viewing.
- Personalized content recommendations based on user preferences and viewing history.

1.2. Non-Functional Requirements

- Low latency and high responsiveness during content playback.
- Scalability to handle a large number of concurrent users.
- High availability with minimal downtime.
- Secure user authentication and authorization.
- Intuitive user interface for easy navigation.

2.High-Level Design of Netflix System Design

We all are familiar with Netflix services. It handles large categories of movies and television content and users pay the monthly rent to access these contents. Netflix has 180M+ subscribers in 200+ countries.



Netflix works on two clouds AWS and Open Connect. These two clouds work together as the backbone of Netflix and both are highly responsible for providing the best video to the subscribers.

The application has mainly 3 components:

Client:

Device (User Interface) which is used to browse and play Netflix videos. TV, XBOX, laptop or mobile phone, etc

OC (Open Connect) or Netflix CDN:

CDN is the network of distributed servers in different geographical locations, and Open Connect is Netflix's own custom global CDN (Content delivery network).

- It handles everything which involves video streaming.
- It is distributed in different locations and once you hit the play button the video stream from this component is displayed on your device.
- So if you're trying to play the video sitting in North America, the video will be served from the nearest open connect (or server) instead of the original server (faster response from the nearest server).

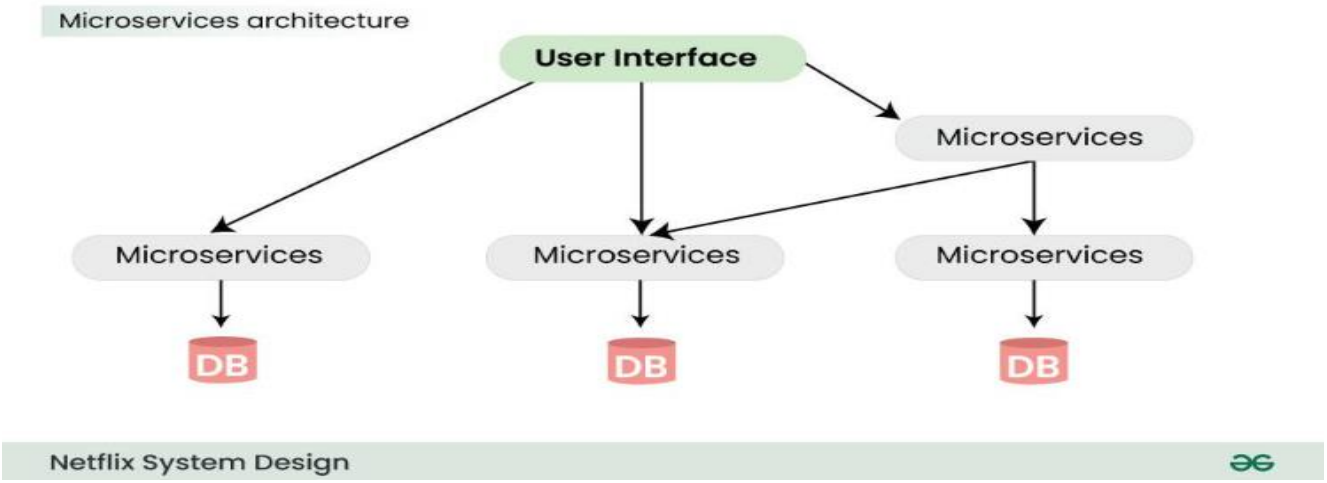
Backend (Database):

This part handles everything that doesn't involve video streaming (before you hit the play button) such as onboarding new content, processing videos, distributing them to servers located in different parts of the world, and managing the network traffic.

Most of the processes are taken care of by Amazon Web Services.

3.Microservices Architecture of Netflix

Netflix's architectural style is built as a collection of services. This is known as microservices architecture and this power all of the APIs needed for applications and Web apps. When the request arrives at the endpoint it calls the other microservices for required data and these microservices can also request the data from different microservices. After that, a complete response for the API request is sent back to the endpoint.



How to make microservice architecture reliable?

1.Use Hystrix (Already explained above)

2.Separate Critical Microservices:

- We can separate out some critical services (or endpoint or APIs) and make it less dependent or independent of other services.
- You can also make some critical services dependent only on other reliable services.
- While choosing the critical microservices you can include all the basic functionalities, like searching for a video, navigating to the videos, hitting and playing the video, etc.
- This way you can make the endpoints highly available and even in worst-case scenarios at least a user will be able to do the basic things.

3.Treat Servers as Stateless:

- To understand this concept think of your servers like a herd of cows and you care about how many gallons of milk you get every day.
- If one day you notice that you're getting less milk from a cow then you just need to replace that cow (producing less milk) with another cow.
- You don't need to be dependent on a specific cow to get the required amount of milk. We can relate the above example to our application.
- The idea is to design the service in such a way that if one of the endpoints is giving the error or if it's not serving the request in a timely fashion then you can switch to another server and get your work done.

2.0 Netflix's DevOps Transformation

2.1 Netflix's Move to the Cloud

It all began with the worst outage in Netflix's history when they faced a major database corruption in 2008 and couldn't ship DVDs to their members for three days. At the time, Netflix had roughly 8.4 million customers and one-third of them were affected by the outage. It prompted Netflix to move to the cloud and give their infrastructure a complete makeover. Netflix chose AWS as its cloud partner and took nearly seven years to complete its cloud migration.

Netflix didn't just forklift the systems and dump them into AWS. Instead, it chose to rewrite the entire application in the cloud to become truly cloud-native, which fundamentally changed the way the company operated. In the words of Yury Izrailevsky, Vice President, Cloud and Platform Engineering at Netflix:

"We realized that we had to move away from vertically scaled single points of failure, like relational databases in our datacenter, towards highly reliable, horizontally scalable, distributed systems in the cloud."

As a significant part of their transformation, Netflix converted its monolithic, data center-based Java application into a cloud-based Java microservices architecture. This brought about the following changes:

- Denormalized data model using NoSQL databases

- Enabled teams at Netflix to be loosely coupled
- Allowed teams to build and push changes at the speed they were comfortable with
- Centralized release coordination
- Replaced multi-week hardware provisioning with continuous delivery
- Empowered engineering teams to make independent decisions using self-service tools

As a result, Netflix accelerated innovation and organically developed a DevOps culture. From 2008 to 2015, Netflix gained eight times as many subscribers and saw monthly streaming hours grow 1,000 times.

2.3 Chaos Engineering at Netflix: Chaos Monkey & the Simian Army

After migrating to the cloud, Netflix became more resilient to failures like the 2008 outage. However, to prepare for unexpected future issues, Netflix embraced a radical concept: build failure into the system deliberately**.**

Chaos Monkey

Netflix developed Chaos Monkey to continuously test the system's resilience against unexpected outages. It is a tool that runs in all Netflix environments, randomly terminating production instances and services to ensure that systems can recover automatically and continue running without affecting end users.

Chaos Monkey helps developers:

- Identify weaknesses in the system architecture
- Build automatic recovery mechanisms
- Test code under unexpected failure conditions
- Develop fault-tolerant systems as part of daily operations

The Simian Army

Following the success of Chaos Monkey, Netflix engineers expanded the idea by creating the Simian Army—a suite of tools designed to simulate different types of failures, enforce best practices, and enhance overall system robustness. Each “monkey” in the army targets a specific area of system health and reliability.

Simian Army Tools

To build greater resilience into its cloud-native infrastructure, Netflix expanded beyond Chaos Monkey by introducing a broader suite of fault-injection and system-enforcing tools collectively known as the Simian Army. These tools simulate various types of failures and enforce best practices to ensure system robustness.

Latency Monkey

Simulates network delays in the client-server communication layers to mimic service degradation. It helps verify if systems can handle latency or even simulate total service failure—ideal for testing new services' dependency handling without causing real outages.

Conformity Monkey

Identifies instances that do not meet predefined best practices or compliance rules. Nonconforming instances are shut down, prompting teams to redeploy them properly.

Doctor Monkey

Monitors the health of instances by analyzing internal metrics (like CPU load) and external signals. Unhealthy instances are flagged and removed from service to prevent downstream issues.

Janitor Monkey

Scans for unused or orphaned resources within the cloud environment and removes them, helping reduce operational clutter and optimize costs.

Security Monkey

An extension of Conformity Monkey, this tool focuses on security. It detects vulnerabilities (e.g., misconfigured AWS security groups) and invalid or soon-to-expire SSL/DRM certificates, enforcing secure system configurations.

10-18 Monkey

("10-18" is a play on the word "internationalization") This monkey detects issues in global configurations, ensuring that services running across regions and languages function properly with the correct character sets and formatting.

Resilience Engineering at Netflix

Netflix continues to lead in Chaos Engineering with a dedicated Resilience Engineering Team (formerly the Chaos Team) responsible for running chaos experiments at scale and maintaining the company's high availability and performance.

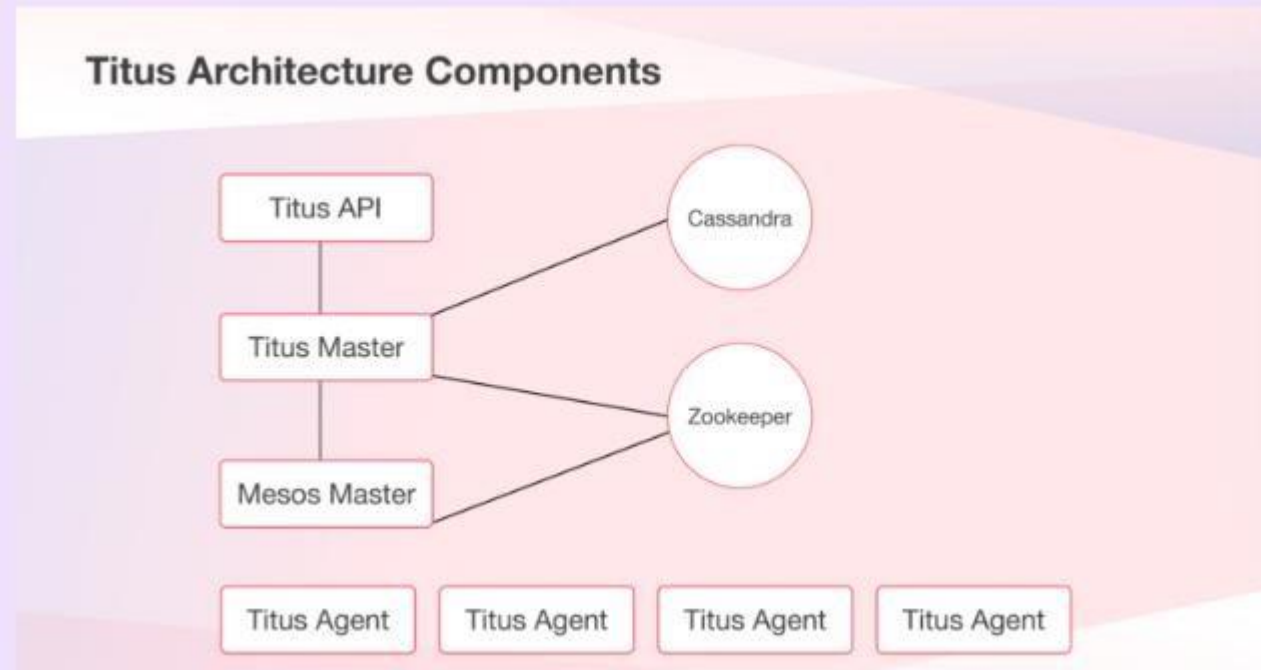
3.0 Netflix's Container Journey

Despite already operating a robust cloud-native, microservices-driven VM architecture—with features like CI/CD, elasticity, and fault tolerance—Netflix chose to invest in container technology to further streamline and scale its operations. The move wasn't about fixing weaknesses but about improving efficiency and developer velocity.

Key Reasons for Adopting Containers

- **Consistency Across Environments**:
  Containers ensure that applications run the same in development and production environments, reducing bugs and simplifying debugging.
- **Lightweight & Fast Deployment**:
  Compared to VMs, containers are more lightweight and start up faster, allowing quicker iterations and deployment cycles.

- **Simplified Packaging**:
  Containers encapsulate everything an application needs, making it easy to create consistent, appspecific images.
- **Improved Resource Efficiency**:
  Containers require fewer resources, are smaller in size, and can be packed more densely—lowering infrastructure costs.
- **Enhanced Developer Productivity**:
  By enabling faster build-test-deploy loops, containers empower developers to innovate more rapidly.



3.1:Titus Architecture Components – Explained

Netflix's container orchestration platform, **Titus**, was built to run and manage containers at scale in production. The architecture diagram includes the following components:

Titus API

- **Role**: Entry point for users to interact with Titus.
- **Function**: Developers submit jobs, query container status, and manage workloads through this API.
- **Acts as**: The front-end to the entire Titus system.

Titus Master

- **Role**: Core orchestration logic and control.
- **Function**: Responsible for:
  - Scheduling containers on Titus Agents.
  - Managing job lifecycle.
  - Ensuring coordination with Mesos and other system components.

Connects to:

- **Cassandra** (for storing state).
- **Zookeeper** (for service discovery and leader election).
- **Mesos Master** (as the underlying resource scheduler, though Titus later moved away from Mesos in newer versions).

Mesos Master

- **Role**: Resource manager (initially).
- **Function**: Allocates CPU, memory, and other resources to Titus containers.
- **Note**: Netflix has been moving toward a Titus-specific scheduler, reducing dependence on Mesos.

Cassandra

- **Role**: Distributed database.
- **Function**: Stores metadata about jobs, containers, configurations, etc.
- **Why Cassandra?**: Highly available, scalable, and fault-tolerant—ideal for storing mission-critical orchestration data.

Zookeeper

- **Role**: Distributed coordination service.
- **Function**: Manages leader election, service discovery, and configuration management within Titus.
- **Ensures**: Titus Master nodes stay in sync and failover works correctly.

Titus Agent (Multiple)

- **Role**: Worker nodes that actually run the containers.
- **Function**:
    - Pulls container images. ○ Launches and monitors containers.
    - Reports health and status to the Titus Master.
- **Runs on**: EC2 instances in AWS.

Summary Flow:

1. Developer submits job via **Titus API**.
2. **Titus Master** schedules the job using **Mesos**, consults **Cassandra** for state, and **Zookeeper** for coordination.
3. **Titus Agents** execute the job containers and report status back up the chain.

4.0 Netflix's "Operate What You Build" Culture

Netflix invests heavily in empowering its engineering teams with end-to-end responsibility. But this wasn't always the case.

Before adopting the "Operate what you build" model, Netflix had siloed teams. The software delivery life cycle (SDLC) was segmented like this:

- Developers → wrote the code
- Ops Teams → deployed, monitored, and maintained it
- Support Teams → handled issues

This handoff model caused delays, miscommunication, and reduced accountability.

4.0 Netflix's "Operate What You Build" Culture

Netflix didn't always follow a DevOps-first mindset. Initially, its engineering process was heavily siloed, with each stage of the software delivery life cycle (SDLC) owned by different roles:

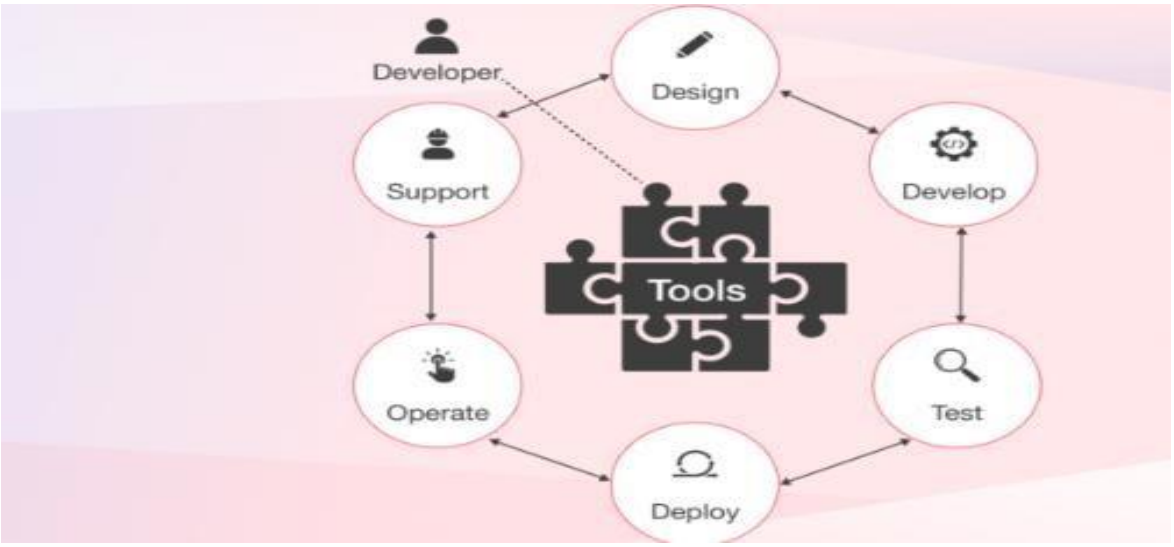
Traditional Siloed Model:

- Architect: Designs the system
- Developer**:** Writes the code
- SDET**:** Tests the code
- Release Engineer: Deploys the application
- **S**ys Admin**:** Operates and monitors the system
- Customer Support**:** Handles user issues

This fragmented ownership often caused:

- Delays in delivery
- Communication breakdowns
- Finger-pointing during outages

5.0 Full Cycle Developers



Combining the above ideas, Netflix built a more advanced model in which development teams are empowered with robust productivity tools and take full ownership of the entire software development life cycle (SDLC). This model—illustrated in the image above—enabled developers to design, build, test, deploy, operate, and support their applications end-to-end.

To support this shift, Netflix invested in:

- Ongoing training, such as developer boot camps, to help engineers acquire full-cycle skills.
- Powerful internal tools, like Spinnaker—a continuous delivery platform that allows teams to release software changes with high velocity and confidence.

While this model greatly improved agility, reliability, and ownership, it's important to note that adopting it requires a cultural and organizational mindset shift.

If you're looking to apply a similar model outside of Netflix:

- Start by evaluating your team’s capabilities and current bottlenecks.
- Consider the operational costs and complexity this model introduces.
- Prioritize simple, incremental changes.
- Most importantly, encourage a mindset of end-to-end responsibility across teams.

6.0 Lessons We Can Learn from Netflix's DevOps Strategy

While Netflix's practices are tailored to its unique environment, many core principles can inspire improvements in other organizations. Here are key lessons to consider:

- **Don't build systems that say "no" to developers** Netflix doesn't enforce push windows or deployment barriers. Every engineer has full access to production. Instead of rigid rules, they rely on trust and accountability.

- **Give engineers both freedom and responsibility**
  By hiring smart, capable individuals, Netflix allows autonomy—trusting engineers to make the best decisions while holding them responsible for outcomes.

- **Don't obsess over uptime at all costs**
  Netflix achieves near-perfect uptime but still prioritizes resilience. Chaos Engineering helps them prepare for unexpected failure, not just avoid it

- **Prioritize innovation velocity**
  Speed matters. Netflix reduces time-to-market by empowering teams to iterate quickly and build customer-focused features.

- **Eliminate unnecessary processes**
  Rather than slow teams down with bureaucracy, Netflix hires trustworthy individuals and empowers them to act independently.

- **Practice "context over control"** Managers don't micromanage. Instead, they provide business context so engineers can make well-informed decisions.

- **Enablement over enforcement** Teams choose their own languages, tools, and frameworks. Netflix doesn't mandate standards—it enables innovation.

- **Break down silos** Everyone understands their place in the larger ecosystem. Developers aren't separated from operations—they own what they build.

- **Adopt a "you build it, you run it" model**
  Netflix's "Operate What You Build" culture emphasizes ownership, supported by strong internal tooling and training.

- **Be data-driven**
  Decisions at Netflix are powered by real-time data, not gut instinct. Systems are built to detect anomalies and trigger alerts automatically.

- **Customer satisfaction is the ultimate goal**
  Every DevOps decision is made with the user experience in mind, ensuring value delivery with every release.

- **Focus on culture, not just DevOps tools**
  Netflix didn't *do* DevOps—it built a culture of trust, accountability, and experimentation. The tools and practices followed naturally.

Conclusion

Netflix's DevOps journey is not just a story of adopting tools or migrating to the cloud—it's a case study in cultural transformation. By embracing principles like full ownership, automation, resilience, and developer autonomy, Netflix created a high-velocity engineering environment that supports continuous innovation and world-class reliability.

Their approach proves that DevOps is not a process you implement, but a culture you build**.** For organizations seeking similar success, the goal shouldn't be to copy Netflix's tools, but to adopt the mindset of empowering teams, breaking down silos, and relentlessly focusing on customer experience.

Netflix didn't "do" DevOps—they became DevOps.