Netflix_DevOps Case Study

Our journey to the cloud at Netflix began in August of 2008, when we experienced a major database corruption and for three days could not ship DVDs to our members. That is when we realized that we had to move away from vertically-scaled single points of failure, like relational databases in our datacentre, towards highly reliable, horizontally-scalable, distributed systems in the cloud.

How Netflix thinks about DevOps.

- Netflix doesn't prioritize DevOps.
- They prioritize Innovation.

Netflix Environment

- They don't prevent engineers from accessing the production environment. Every Netflix engineer has full access to the production environment from day 1.

- They don't prioritize uptime at all costs, to achieve 100% uptime, they need to sacrifice innovation.

- They don't focus on processes and procedures.

- They don't enforce using specific programming languages and frameworks. Instead, they give engineers the freedom to choose the best standard for the job if that means the code is optimized and the users get a better experience.

- They don't believe in gut instincts and traditional thinking but focus on data instead. The majority of the decisions Netflix makes depend upon data.

Netflix DevOps Data & Numbers

Netflix has 3 main components:

- Compute and storage, managed through Amazon Web Services.
- UI & small assets built using Akamai
- Netflix Open Connect – their purpose-built video CDM.

Netflix has:

- 100s of microservices.
- 1,000s of daily production changes.
- 10,000s of virtual instances inside Amazon.
- 100,000s of customer interactions per minute.
- 1,000,000s of customers.
- 1,000,000,000s of time series metrics.

Netflix Inhost Tools:

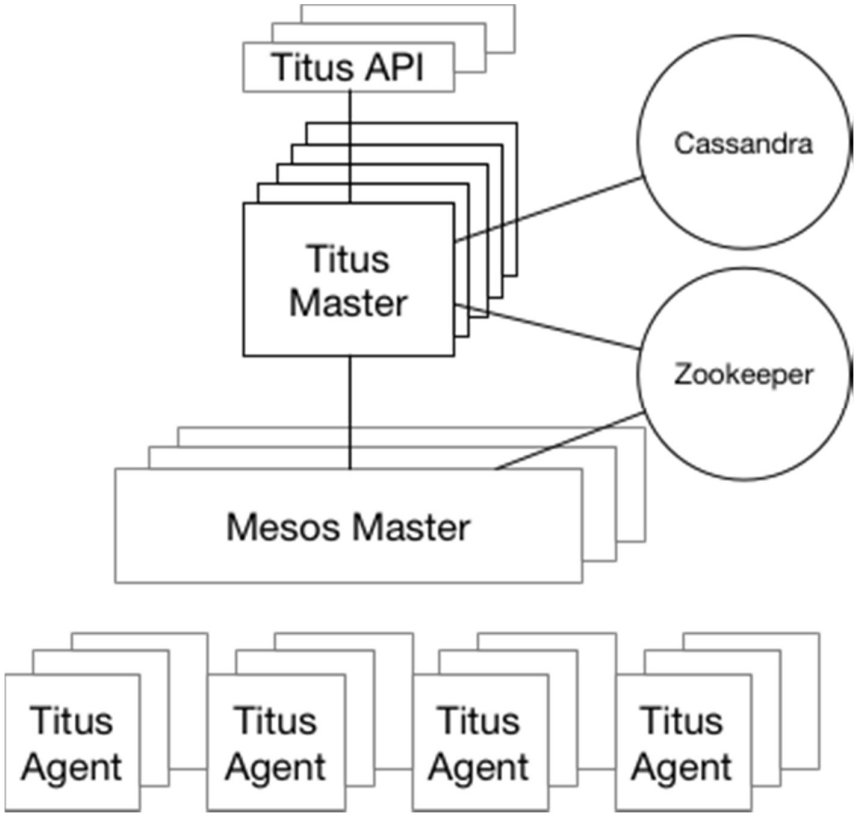
1. Titus: Titus is a container management platform that provides scalable and reliable container execution and cloud-native integration with Amazon AWS. Titus was built internally at Netflix and is used in production to power Netflix streaming, recommendation, and content systems. Titus aims to enable easy and reliable deployment of containerized batch and service applications.

   Achieving this goal requires:

- Allowing containerized applications to seamlessly interact with AWS, Netflix, and other cloud services. Interactions with other systems should not be an application burden simply because it is running in a container.
- Operability to ensure the system is capable of running mission critical workloads and meeting SLAs.
- Scalability to run tens of thousands of containers on top of thousands of hosts across a variety of use cases.

Architecture:

Titus is a framework on top of Apache Mesos, a cluster-management system that brokers available resources across a fleet of machines. Titus consists of a replicated, leader-elected scheduler called Titus Master, which handles the placement of containers onto a large pool of EC2 virtual machines called Titus Agents, which manage each container's life cycle. Zookeeper manages leader election, and Cassandra persists the master's data. The Titus architecture is shown below.



2. Chaos Monkey: Chaos Monkey is a tool developed by Netflix as part of its Simian Army to test the resilience and reliability of distributed systems. It introduces random failures in production or test environments, such as shutting down instances or services, to ensure the system can handle unexpected disruptions without impacting overall functionality.

   Key Features:
   - Random Instance Termination.
   - Focus on Resilience.
   - Integration with Cloud Environments.

Requirements:

- This version of Chaos Monkey is fully integrated with Spinnaker, the continuous delivery platform that we use at Netflix. You must be managing your apps with Spinnaker to use Chaos Monkey to terminate instances.
- Chaos Monkey should work with any backend that Spinnaker supports (AWS, Google Compute Engine, Azure, Kubernetes, Cloud Foundry). It has been tested with AWS, GCE, and Kubernetes.

Chaos Monkey is a fundamental part of Netflix's "fail fast, fail often" philosophy, promoting robust and resilient system design. Many organizations now use similar chaos engineering practices to improve system reliability.