GitHub

» CUSTOMER CASE STUDY

Upgrading Consul At Scale

Tasked with upgrading dozens of Consul environments of varying versions on premise and in the cloud, Github engaged Nebulaworks as the strategic partner to help with this initiative.

NEBULAWORKS

Executive Summary

Deploying distributed systems like Consul for service discovery, and key value store at scale is a complex initiative. When it comes to upgrading and maintaining these systems teams must do their due diligence to assure that operating on it will not have any drastic effects on the software consuming its services. Tasked with upgrading dozens of Consul environments of varying versions on premise and in the cloud. Nebulaworks worked with GitHub to develop and execute a plan that reduced risk and increased transparency when upgrading these environments.

Why Nebulaworks?

There was no existing solution to this problem. A consul test framework simply does not exist, nor has this distributed system (consul) lived long enough to provide concrete evidence for its claims of backwards compatibility and robustness across versions. It needed to be built from scratch with the constraints and consideration in mind by like minded engineers that understood the depths to which consul had infiltrated the foundations of the GitHub back end ecosystem. Coupling this with the fact that a rollback or recovery scenario for the distributed system as large as GitHub's multi-cluster, multi-region wan federated WAN consul platform was something that was just as risky as the upgrade, GitHub had to find a partner that understood the severity of this problem.

The Challenge

Tools that provide service discovery such as Hashicorp's Consul, provide the glue that enables different software services the ability to communicate with each other. A variety of GitHub internal services were using a wide range of Consul functionality that were critical to production services including service discovery, key value stores, semaphores and locking mechanisms. This massive Consul footprint included dozens of revenue generating multi-clustered, and multi-region deployments connected over WAN. If these services were to break, crash, or become unavailable, so would many customer-facing services. This core component to the infrastructure was 2 years and several versions behind the latest leaving an exploit waiting to be compromised thus an upgrade was necessary. The upgrade process would not be to simply replace a binary and reboot the machine. Due to the sheer size and architecture of the Consul deployments, dozens of different teams dependent on it, and the multitude of runtime languages consuming Consul services, it was critical for the team to determine how upgrading binaries would affect consumers in a transparent and automated fashion. At all costs the team wanted to prevent any downtime while operating on these clusters. Maintaining uptime of these systems was essential due to the tightly coupled dependencies of Consul and these applications GitHub needed a way to upgrade these clusters transparently, without disrupting these business critical systems.

The Solution

While considering the sensitive nature of these systems, Nebulaworks engineered a concise, strategic, and actionable testing plan for upgrading Consul server and agent binaries while taking into consideration the details of how applications were leveraging different Consul services. The key to the upgrade was twofold: First the team wanted to understand the behavior of executing a rolling update for a single Consul Cluster and two, determine the behavior of the larger geographic Consul WAN clusters when one was updated.

The vision that the team had for tackling this initiative was to build an automated test framework that would provide detailed information on what would happen when a Consul cluster is updated. The first step in this initiative was to determine the compatibility of versions in the existing Consul clusters. By using Linux Containers, the team defined a compatibility matrix, built containers running different versions of consul, and simulated the version swap to discover version incompatibilities quickly and with low risk.

Operating on the revenue generating production environment was not an option and simulating the version upgrade on containers was not a sufficient test environment, so the team developed isolated sandbox deployments of multiple-interconnected Consul Clusters in AWS using Infrastructure as Code (IaC). With these sandbox environments that mirrored production systems the team could simulate the upgrades more accurately.

> Nebulaworks engineered a concise, strategic, and actionable testing plan for upgrading Consul server and agent binaries while taking into consideration the details of how applications were leveraging different Consul services.

Serverspec was a tool used that provided server validation for the Consul Clusters deployed to the cloud. Serverspec not only provided configuration validation, it also provided the ability to execute functional tests. These functional tests were built in Go, Ruby and Bash in order to simulate different application runtimes that were currently leveraging the different consul services. A set of tests would write data pre-upgrade, then the upgrade would be performed, and the same tests would re-read that same data to determine if the upgrade did not affect existing data.

The update process itself was done in version increments to prove as a Minimum Viable Product (MVP) of the upgrade working properly. Since backwards compatibility was a requirement for some features in use, the testing procedures for the rolling update needed to incrementally test different versions to ensure that no versions in between current and target versions, would create regression failures. Any bugs that were found during these incremental updates were addressed in the test environment.

NEBULAWORKS

Outcome

Equipped with an actionable testing plan, the GitHub team was able to confidently wield the testing framework and process to validate the result of upgrades on multi-cluster, multi-region staging and production environments. What made the knowledge transfer successful was the constant collaboration and participation of GitHub and Nebulaworks Engineers. With intimate knowledge of the testing code base, the GitHub team was able to reduce the risk and increase transparency into the upgrades as they took place. Most importantly, the operations team responsible for the Consul clusters were able to fulfill a security audit request, taking their systems out of harms way, and upgrading to the latest version of Consul as of writing this. Today GitHub continues to use Open Source Consul in order to provide critical services that support customer-facing systems.

NEBULAWORKS