SCTE CABLE-TEC EXPO FAST FORWARD 2021
VIRTUAL EXPERIENCE
OCTOBER 11-14

2021 Fall Technical Forum
SCTE • NCTA • CABLELABS

Humanoids Optional: Deploying vCMTS at Scale with Automation

A Technical Paper prepared for SCTE by

Bhanu Krishnamurthy
VP, Software Development and Test
Comcast
1800 Arch Street, Philadelphia PA
217-437-2811
Bhanushree_krishnamurthy@comcast.com

Gregory Medders
Principal Engineer
Comcast
1800 Arch Street, Philadelphia PA
Gregory_Medders@comcast.com

Table of Contents

Title **Page Number**

List of Figures

Title **Page Number**

1. Introduction

For more than two decades, cable internet has provided the means to access the internet for hundreds of millions of people. The traditional cable modem termination system (CMTS) has played a key role in facilitating this access. The advent of cloud computing and containerization has caused a shift in access network solutions, with proposals to replace the traditional "integrated" CMTS (iCMTS) with a virtual CMTS (vCMTS), where the hardware-based iCMTS is replaced with an cloud native architecture that delivers the same functionality. By decoupling the CMTS implementation from the underlying hardware, vCMTS was proposed as a way of providing a modern, more flexible alternative to traditional models. For example, when deployed in a distributed access architecture, where the digital nodes can be placed closer to the homes they serve instead of in the headend, vCMTS can provide improved service with significantly increased density while also reducing headend infrastructure costs.

Broadly, vCMTS brings cloud-native approaches to bear on a problem that was traditionally solved through iCMTS hardware/appliances (Cloud Native Computing Foundation, 2020). At Comcast, vCMTS is comprised of a collection of servers running Kubernetes to orchestrate the containerized vCMTS microservices (The Kubernetes Authors, 2021). These micro services communicate with a remote physical device (RPD) through a networking layer composed of various switches in a leaf-spine architecture (see Figure 1). While cloud providers such as Amazon Web Services (AWS) offer services that can stand up Kubernetes clusters with a click of a button, latency and timing between vCMTS containers and the RPDs is a critical consideration for DOCSIS protocol application, making it impossible to use a public cloud for compute resources. As a result, the vCMTS cluster are typically deployed at the edge in the hub sites where their iCMTS counterparts were previously deployed.

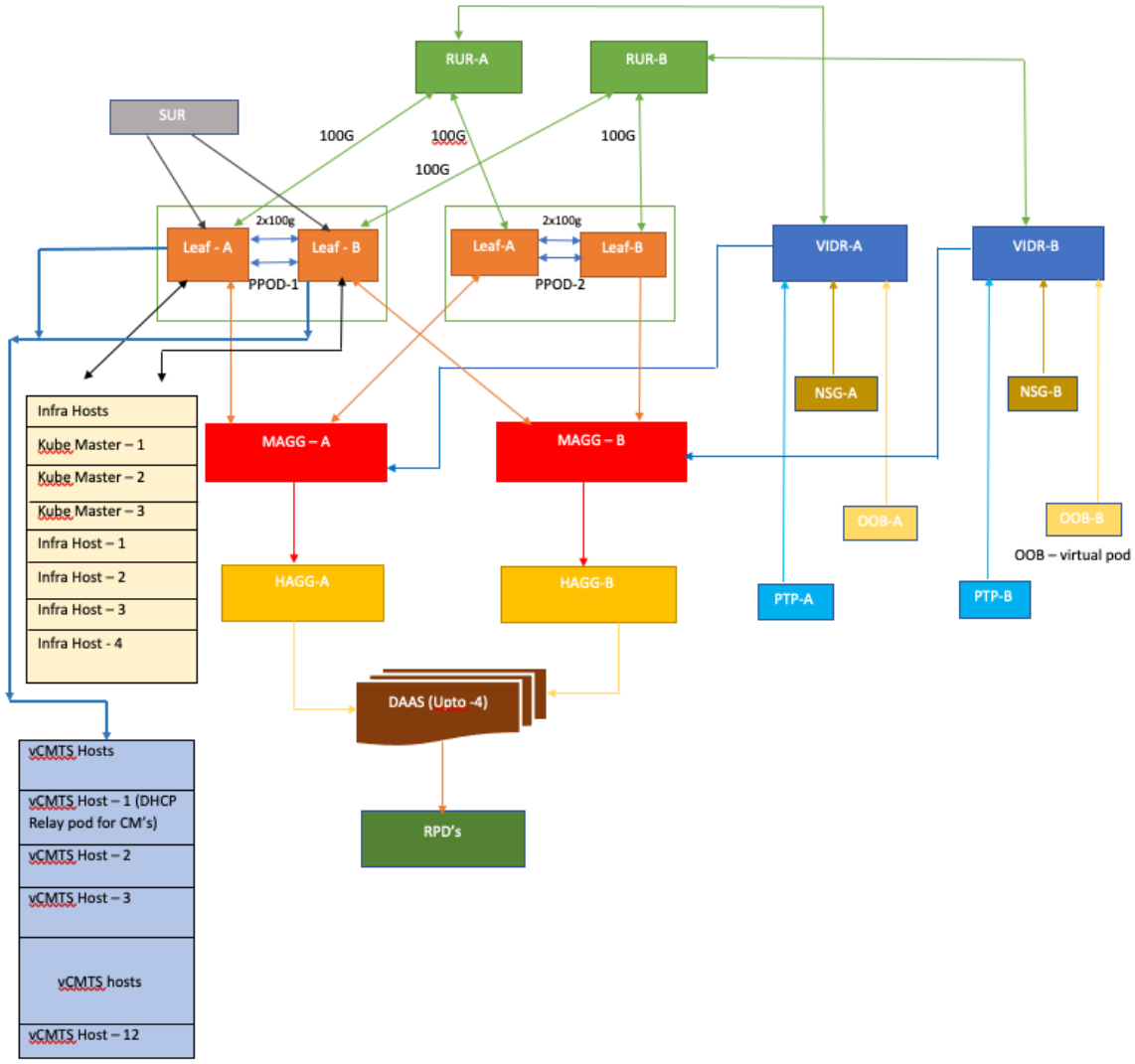

Figure 1- VCMTS Network In A Leaf/Spine Architecture, Where VCMTS Microservices Are Hosted In Servers On The Leafs And The Spine Facilitate Connectivity To The Rpds.

In 2018, Comcast deployed its first production vCMTS cluster. At that time, bringing the first customers online in vCMTS architecture was a major success, proving the feasibility and performance of the distributed access architecture (DAA) using vCMTS (Cable Television Laboratories, 2014). However, as Bob Gaydos, a Fellow at Comcast, remarked at the subsequent SCTE meeting, "Realistically, because Intel came in and helped us … make the vCMTS work on their platform, that's not the hard part. The hard part is actually the operations" (Robuck, 2018). Though the vCMTS architecture provided a huge improvement over iCMTS in terms of visibility through real-time telemetry and logging, operationalizing vCMTS proved challenging in several different ways: how the clusters were built; how changes were deployed to the clusters; how operations teams monitored the systems for issues and responded to incidents.

After our initial success, we started scaling out vCMTS across the Comcast footprint and quickly realized that our operational model would not scale. While each subsequent cluster was identical to the original in terms of hardware and software, each required unique configuration (e.g., IP addressing), making each new cluster build a significant effort. It was not uncommon for a cluster to take months to build from the time when the hardware arrived in the data center until we were ready to cut the first customers onto the new cluster. Because the clusters were painstakingly built by hand by different people, the vCMTS clusters were inevitably considered "pets" — each with their own slightly different personalities. While effort was made to keep the cluster definition in code, because the clusters were hand-configured from inception, it was a constant battle to prevent the clusters from "drifting" from their initial configuration over time.

The culture of customization proved particularly problematic when scheduled maintenances were performed. Since the clusters were built manually by humans, changes to the clusters generally also occurred by humans, often involving a long and complicated standard method of procedure (SMOPs) to both implement and validate the change. Though the SMOPs were scripted as much as possible, because the running configuration of a cluster could differ from the configuration stored in source code, the same change may go perfectly on one cluster but face an unexpected issue in another cluster. These issues could compound; when an unexpected issue was encountered, changes that should have been trivial to revert could snowball into outages that were difficult to resolve. Consequently, changes to software were approached with wariness and generally avoided, resulting in a delay or outright cancelation of feature deployment in this first generation of clusters.

Having proved the validity of the vCMTS model, it became clear that something needed to change if we were to deploy on a national scale. Staffing estimates predicted that we would require between 10-100 times the number of current operational resources to support the vCMTS clusters across the Comcast footprint; more importantly, root cause analyses of incidents indicated that a significant number of issues encountered were caused by human error. While DAA improves the fault tolerance of vCMTS, this comes at the cost of complexity. Fortunately, we were able to use much of the learnings from the DevOps movement to identify three key areas where we needed to improve: clusters must be identical, changes must become "boring" (i.e., frequent and predictable), and incident detection and mitigation must be automated (Kim, Humble, Debois, & Willis, 2016). Practically what this meant was that everything in the cluster lifecycle, from creation to maintenance to incident detection and mitigation, needed to be automated.

2. Infrastructure as Code: GitOps and a Beginning to the End of Customization

Even before the commitment to "automate everything", it was very clear that we were contributing to our own problems by requiring humans to deploy changes. After the first successful vCMTS customer trial, significant effort was invested in codifying the vCMTS deployment into an "infrastructure as code" (IaC). This idea, which has been central to the DevOps movement, is that the desired state of a system should be expressed as a versioned (e.g., Git) code that completely describes how to build that state. Therefore, any desired change to the running state should be achieved by altering the desired state.
A common variant of IaC is the "GitOps" model, where the infrastructure code is stored in Git and, upon commit to Git, the change conveyed in that commit is applied to the running system via a continuous integration/continuous deployment (CI/CD) pipeline. In our initial vCMTS systems (aka "Gen1") human operators were still required to implement changes; nonetheless, investing in IaC enabled us to rapidly rebuild systems, bringing us a step closer toward eliminating the culture of customization that made changes unpredictable.

3. Automating the vCMTS cluster stand up

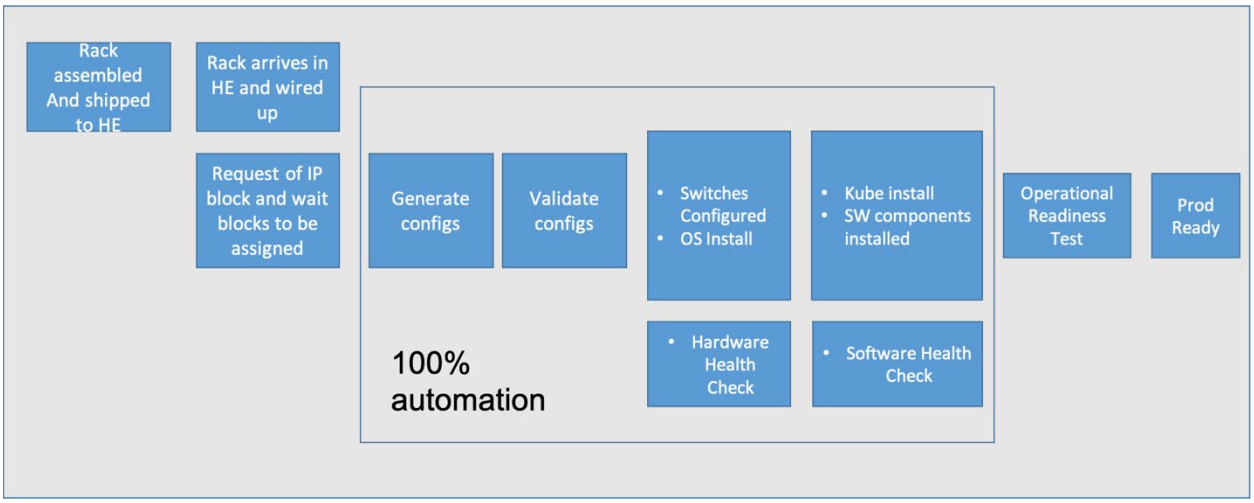

Figure 2 - Automated vCMTS cluster build process

As we were beginning the process of automating the vCMTS cluster build, an obvious but important step was identifying exactly what was required to build a cluster. A cluster build would typically occur over the course of many weeks. Eventually, we found that nearly a dozen teams or specific individuals were responsible for the various steps in a cluster build; this makes sense given that the original build was an "all hands" effort, different individuals or teams naturally became the subject matter experts on different aspects of the build. Since it evolved organically, the resulting build process resembled a web of dependencies as opposed to an assembly line. In the same way that the physical cluster assembly benefitted by consolidating within a team that "owned" the assembly process, as the dependencies were untangled and codified into a CI/CD process (Figure 2), the network and cluster build process was also recentralized around a dedicated team.

Recall that vCMTS consists of microservices (performing the DOCSIS scheduling and control functionality) and that communicate with RPDs via a networking layer. Having streamlined the issues in cluster builds, the remaining challenges can be grouped into two categories: network (switch OS and configuration) and vCMTS (cluster operating system, Kubernetes, configuration, and microservices). In both cases, the configuration (primarily related to IP assignments) is automatically generated based on the planned topology. Running a vendor-supplied network operating systems (NOS), the network primarily implements IS-IS to provide dynamic routing between the microservices and RPDs. When the cluster racks arrive in the local data center and are powered on, the first part to provision are the network devices. The switches automatically begin zero-touch provisioning (ZTP), download the latest OS (IaC, versioned in Git), and load the generated configuration (again, versioned in Git). When the switch becomes reachable, health checks are automatically performed to verify that the switch has the correct OS, configuration, and that the network is healthy.

Once the networking layer is functioning, a CI/CD pipeline brings the vCMTS online. This begins with the servers loading their base operating systems via ZTP. Subsequently, firmware is updated, Kubernetes is installed, and the vCMTS and supporting microservices are installed. A constant source of friction

during the original vCMTS deployments was integrating new clusters with external applications within Comcast (i.e., services that were not part of vCMTS itself, but that were required for a vCMTS cluster to function normally); this process was initially informal (e.g., an email), but by formalizing these dependencies as APIs, a constant stumbling point with external teams became a seamless integration. The CI/CD pipeline concludes by performing health checks for the application level, verifying that the vCMTS system is functionally working, and concludes by registering itself as a built cluster. By structing the build as a CI/CD process, we prevent an unhealthy cluster from accidentally being used to serve customers.

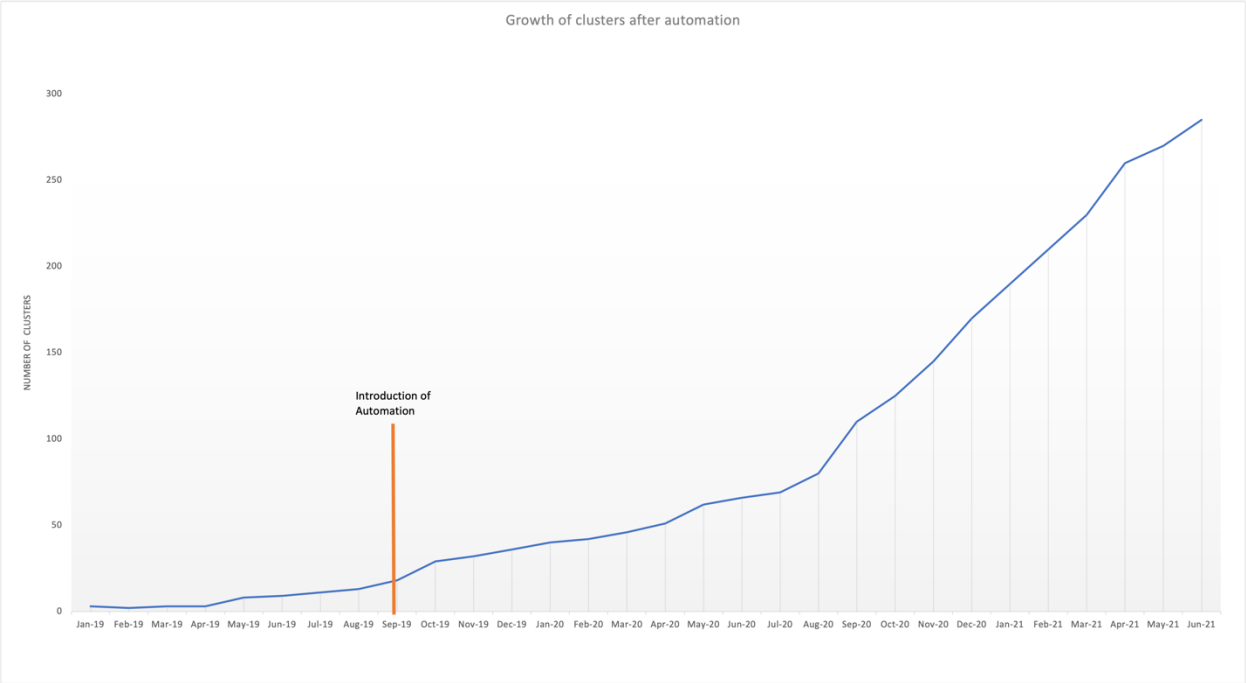

Figure 3 - Number Of VCMTS Clusters Versus Time. The Automated Build Process Began In September 2019, Significantly Accelerating The Cluster Build Process.

While the cluster could theoretically cut-over customers upon conclusion of the CI/CD pipeline, we have maintained a final step of human-driven manual testing (e.g., a cable modem is brought online, voice calls are performed, speed check, connectivity to backend tools, video channel lineup tests). This is consistent with our overall goal of automation—humans should: be removed from tedious and error-prone work that can be readily performed by machines, function as supervisors of machines by providing high-level instruction (e.g., "build a cluster with these parameters"), monitor for patterns and issues that are currently unknown and difficult to detect through automation.

Revisiting the overall build process, human involvement has been limited to 1) physically wiring the cluster/network, 2) some upstream network configuration into the core network (which currently cannot be automated), 3) final manual validation testing. This automated build process was released in September 2019 and as shown in Figure 3, has enabled us to grow the number of vCMTS clusters deployed across the Comcast footprint exponentially without a corresponding exponential growth in human resources as previously predicted. Concretely, provided that the clusters have been wired correctly, the automation has decreased the cluster build time from weeks to a few hours. The cluster

build process has become so predictable that all clusters which had been hand-built prior to automation were rebuilt in automation with essentially no customer impact. Now that the clusters were all identical and consistent with their IaC definitions, we were able to focus on the automation of changes.

4. Automating software and network changes

A critical issue we encountered in Gen1 was that, over time, the clusters drifted from their configuration defined in Git. This was primarily due to humans needing to deploy changes and address incidents, each which inevitably introduced small (unintentional) changes that compounded into significant drift. Thus, having spent considerable effort to ensure that the vCMTS clusters were built with no human involvement from a CI/CD pipeline using (automatically generated) configuration stored in Git, it was critical that any subsequent change to that git also be deployed to the cluster automatically.

While DevOps is often framed in terms of using tooling to achieve CI/CD, a mantra of the DevOps community is that a DevOps transformation is a cultural shift, not just a change in tooling. One of the first significant steps toward this was realized by Eric S, Principal Engineer at Comcast. A frequently encountered issue in Gen1 was that people tended to inadvertently work outside of version control. Whether it be by writing a SMOP and storing it in the change ticket (only to have the underlying cluster definition change in Git, rendering the SMOP "stale"), applying a script that was valid in a previous software version but invalid in the currently deployed one, or contaminating one site by accidentally using configuration from another, we constantly encountered issues with incorrect configuration being accidentally applied to clusters. By 1) breaking apart the deployment scripts for the various microservices into isolated and independently versioned bundles and 2) changing the workflow to require users to always import the current IaC into a local, containerized environment any time they wished to interact with the cluster, Eric was able push the organization toward that DevOps transformation by making it significantly more convenient to practice IaC.

A second step toward DevOps maturity was achieved through the CICD pipeline (Figure 4). Capitalizing on Eric's work to isolate cluster components into visioned collections of deployment scripts (in our case, Ansible roles), deployment pipelines were developed using the same IaC. Operating at the vCMTS cluster level, the pipelines were automatically triggered when a software component version was upgraded for that specific cluster. In Gen1, each change was required to have a series of pre-deployment and post-deployment checks; these were generally included in the change ticket so that if a deployment did not go as planned, the incident could be better triaged. As part of the Gen2 deployment pipeline, these pre-/post-check deployment snapshots were automatically captured and linked with the associated change ticket. By automating this tedious but important part of deployment process, we were able to further incentivize adoption of the CICD process.

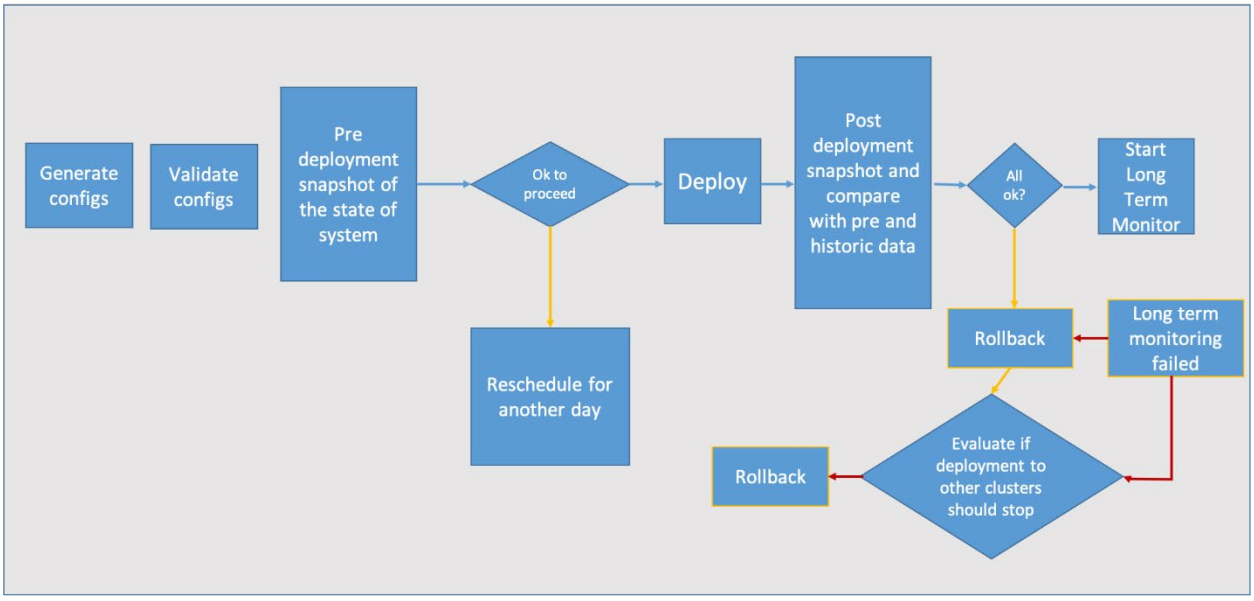

Figure 4 - Software And Network Change Deployment Pipeline

However, as both the number of clusters (due to stand up automation) and the number of changes (due to the cluster deployment automation) began to exponentially increase, we had an increasingly urgent need to orchestrate the deployment of changes across clusters. Since each cluster was identical, we were able to track changes as they were deployed and be confident that any issues encountered were due to the change, as opposed to unexpected differences in the clusters. Here, there are two competing interests. An important theme in DevOps is that developers should have the ability to easily introduce changes themselves and rapidly receive feedback about the success/failure of that change; however, the reality is that ISP operations are critical services, limiting the use of practices like A/B testing of potentially service impacting changes. Here, a compromise is achieved using canary testing, where each change is introduced gradually across the footprint. Changes are categorized by risk and deployed to minimize potential customer impact—first in QA, then staging environments (those without subscribers), and ultimately staggered into production environments.

In terms of changes to the network layer, the switches require both upgrades to their NOS as well as configuration changes. There is redundancy in the network so that if any given switch goes down, the redundant path is used with no impact to service. Our automation has taken advantage of this redundancy to turn maintenance into a zero-downtime event. Maintenance actions are first performed on one device (“A”) in the redundant pair (“A”, ”B”); traffic is then shifted to “A” side and only if everything behaves normally does the maintenance continue to patch the “B” side; if unexpected behavior occurs, the change is reverted. Through this entire process, the subscriber served by that network is unaware of any change being performed—their services continue uninterrupted.

Due to the large number of switches in the vCMTS access network (currently in the thousands), it was imperative that we fully automate the network operations and maintenance. To this end, configurations are generated purely through automation. Network operators are not generally allowed to make changes to the configuration on any given switch. Instead, network engineers edit the switch’s template file and commit it to version control where, consistent with the GitOps model, changes are detected and automatically scheduled for deployment across the entire network. Similar to the software

deployment automation, a network change will ripple across the Comcast footprint, with the engineer who performed the change having direct visibility into the impact and success/failure of their change through health checks. Consistent with the DevOps principles, providing this immediate feedback to engineers has significantly increase the overall quality of network changes and stability. Rather than the classic pattern of engineers "throwing a change over the wall" to operations to implement, the person who initiated the change is empowered to safely deploy a change across multitude of devices with a click of a button.

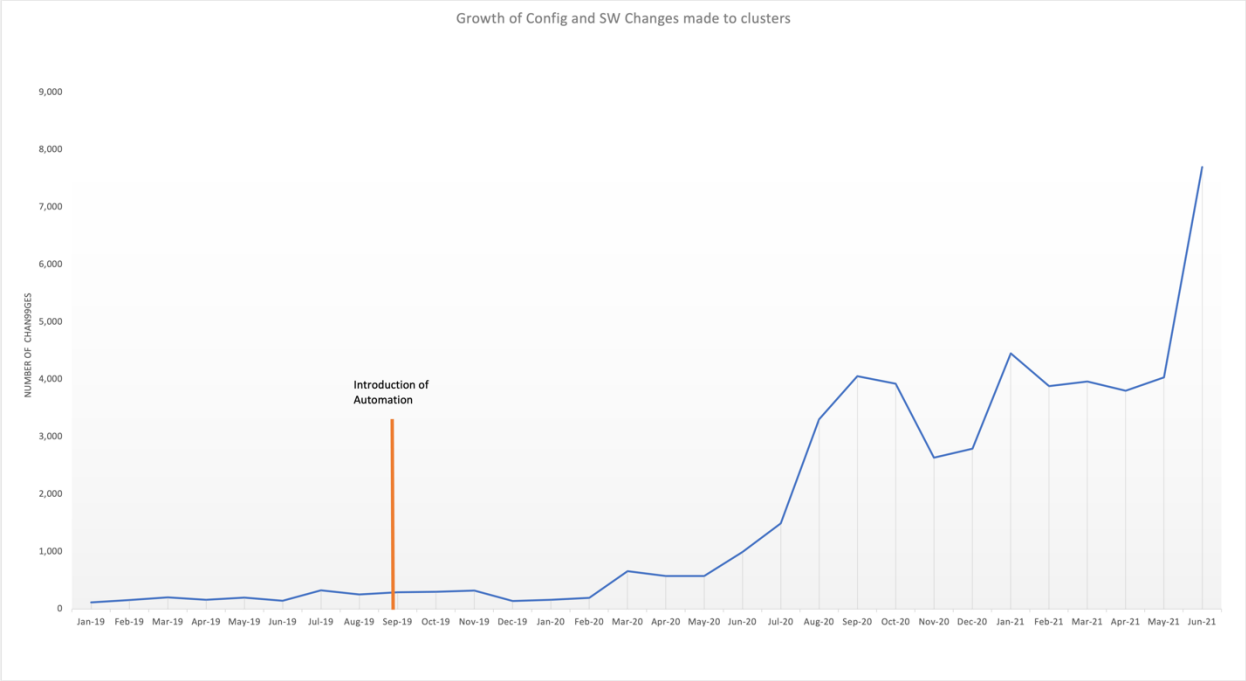

Figure 5 - Number Of Changes Versus Time. Automated Change Deployment Pipelines Were Introduced In September 2019

As is shown in Figure 5, automation was crucial in enabling our ability to deploy changes across the ever-expanding footprint of vCMTS devices. In Gen1, software and network changes were high-stress and high-risk events. Even though we had fewer than 25 Gen1 clusters, any given change would take several weeks to roll out since each change was manually performed by a human; furthermore, due to the high incidents of human error causing accidental outages, all changes (even to supporting applications that were not required for customer service) were typically performed in a scheduled maintenance window in the middle of the night out of a general fear of an unexpected issue escalating into a customer outage. As will be shown in the subsequent section, automation was crucial to our ability to scale out and rapidly deploy changes. Furthermore, the DevOps adage that "infrequent changes decrease service availability, frequent changes (e.g., continuous deployment) increases availability" was borne out; frequent deploys, including empowering the engineer who is making the code change to deploy the code, has significantly improved reliability and service availability, making changes "boring".

5. Automating incident detection and mitigation

After automating the VCMTS cluster build and change processes, our next focus was to automate the incident detection and mitigation. Because each cluster is identical and because no humans interact with the clusters directly during normal operations, our overall incidence of human induced errors dropped to zero essentially overnight. Correspondingly, the availability of the clusters across the

footprint has increased significantly (Figure 6). While issues certainly arise day-to-day (e.g., hardware failures), due to the built-in redundancy of the vCMTS clusters, these issues are typically not service impacting. Furthermore, because these issues are often routine, the response to them is known and can be codified in a script. For example, when a server in the Kubernetes cluster becomes NotReady, we automatically determine the cause of the Kube node being in NotReady state; if the problem is fixable and non-impacting (e.g., by rebooting the host or even reinstalling the host OS), it is done automatically and noted for subsequent analysis rather escalating to on-call operational support. If the issue does require human intervention (e.g., arranging an RMA with a vendor, re-seating an optical device in a maintenance window) steps like scheduling the maintenance are automated. By using machines for these routine and repetitive tasks, we significantly decrease the load on our ops team, enabling them to act as supervisors of the automation by codifying their responses into scripts and reserving them to troubleshoot non-trivial issues.

Figure 6 - System Availability Vs Time. Availability Is Plotted On The Right Side (Excluding Scheduled Maintenances) And The Number Of Incidents Per Month Caused By Human Error Is Plotted On The Left.

A hallmark of cloud native architectures is the improved visibility into the internal workings of the systems. Using a combination of open-source solutions, our vCMTS operators have visibility into everything ranging from network health to the performance of individual microservices. When an incident arises, this makes troubleshooting that cluster a much easier process because nearly everything that could go wrong is monitored. However, considering that we now operate more than hundreds of clusters, we cannot rely on humans having "eyes on glass" monitoring the clusters for potential issues. In a very real sense, we are simply inundated with data. Fortunately, in the age of data science and machine learning, being inundated with data is a great problem to have. A customer going offline could be caused by many sources: a fiber cut, power outage, network issues, backend unavailability, cluster software errors, etc. Troubleshooting the root cause can therefore be a lengthy process, resulting in a long mean time to repair (MTTR). Instead of immediately escalating to a human to investigate, our automated triage system can quickly probe various failure domains and rank likely root causes. This allows us to rapidly deploy the correct fix-agent.

During a deployment, variable usage patterns in the network can make immediately identifying issues difficult. While many changes are known to be non-service affecting in advance, some changes are intrinsically higher risk. Hence, long-term monitoring for anomalies is required. Using machine learning, external systems monitor telemetry for deviations from historical patterns. Since higher-risk changes are staggered across the footprint, this enables the automation system to detect anomalies and recommend that deployment be halted and already-deployed changes be reverted. This is integrated in our deployment orchestration system, allowing for the safe and hand-off deployment of changes.

6. Conclusion

In the past few years, vCMTS has evolved from a prototype to full-fledged production infrastructure providing cable services to a considerable portion of the Comcast footprint. While our initial Gen1 deployments proved the technology of vCMTS, the last three years have been devoted to scaling vCMTS operations. Often when we discuss our work with colleagues in other departments, a common response is that their work is automated too—via scripts. We want to make the distinction that, while Gen1 was fully scripted, the automation in Gen2 is neither a collection of scripts nor a simple CI/CD pipeline. Indeed, while one can write a CI/CD pipeline for a full stack application and deploy it in the public cloud in an hour, we have (somewhat painfully) learned that is not trivial to automate the operations of hundreds of bare-metal Kubernetes edge clusters running in hundreds of data centers across the US, in total comprising of thousands of switches and tens of thousands of servers. After our early work defining our infrastructure in code, we naively thought we had solved the difficult part of the problem. In fact, someone in our organization famously predicted it should take “a few weeks” to whip up the automation to allow us to manage vCMTS at scale. Instead, what began with a few engineers has evolved over the past three years into multiple teams comprised of 15 people. Building capabilities such as self-recovery, complex decision-making logic, risk-based scheduling to name a few took months of hard work by some of our highest performing engineers.

In a traditional ISP operations model, even our current footprint was projected to require hundreds of operational support engineers. Our goal ultimately is to reduce that to tens of engineers. While we released our initial automation almost two years ago, this work has grown over that time into a full-fledged product involving the collaboration of numerous teams at Comcast. Critically, this work has enabled us to achieve the exponential growth necessary to expand vCMTS across the Comcast footprint while also improving on the reliability and maintainability of iCMTS. As this work continues, we are further integrating our operational automation into the monitoring and core network infrastructure at Comcast, with the goal of achieving a zero-downtime and high-speed experience for our customers.

Abbreviations

vCMTS	Virtual Cable Modem Termination System
iCMTS	Integrated Cable Modem Termination System
IaC	Infrastructure as Code
DAA	Distributed Access Architecture
SCTE	Society of Cable Telecommunications Engineers

Bibliography & References

Cable Television Laboratories. (2014). *Remote PHY Specification.*

Cloud Native Computing Foundation. (2020). *CNCF Annual Report.*

Kim, G., Humble, J., Debois, P., & Willis, J. (2016). *The DevOps Handbook.* Portland: IT Revolution Press, LLC.

Robuck, M. (2018, October 29). Retrieved from Fierce Telecom: https://www.fiercetelecom.com/telecom/comcast-execs-lessons-learned-from-deploying-vcmts

The Kubernetes Authors. (2021). Retrieved from Kubernetes: https://kubernetes.io

