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

2021 Fall Technical Forum
SCTE • NCTA • CABLELABS

# Strategies for Continuous Integration and Continuous Deployment at Scale at the Network Edge

## …a.k.a. The Pursuit of the Zero-Downtime Headend…

A Technical Paper prepared for SCTE by

**Quincy Iheme**
Engineering Manager
Comcast TPX
Philadelphia
267.260.2923
Quincy_Iheme@Comcast.com

 

# Table of Contents



## List of Figures



 

# 1. Introduction

The next several years will go down as the time when an astounding amount of work happened at the edges of the network, in part to get to the coveted "zero downtime headend." A perpetually-up headend matters because it sets the stage for edge compute services. Much of that work at the edge of the network is benefitting from the role of software, and in particular, the branch of software engineering that is Continuous Integration / Continuous Deployment (CI/CD).

This paper discusses how operators can take advantage of the new flexibility that comes with the deployment of software at the network edge, more quickly and securely. It will provide an overview of CI/CD, with a specific focus on how it is being applied at the edges of the network, in virtual Cable Modem Termination System (vCMTS) deployments and related efforts.

Software engineering is unquestionably infiltrating many aspects of the physical infrastructure that historically were hardware-only. As a result, the establishment of and adherence to a common CI/CD platform can set the stage for the anticipated increase in software deployment at the network's edges, especially for services or activities that require very low latency and/or wider throughput.

This paper will describe how Comcast arrived at a common CI/CD platform, then put it to work at the network edge to maintain network uptime and increase upstream throughput – such as blue/green and automated deployments, as well as Distributed Access Architecture (DAA) components like vCMTS and edge switching, to enable continuous configuration and deployment at scale. It will also discuss tools and automation, and how to configure and push software to the edge to enable customers to experience faster broadband speeds.

# 2. A Brief History of the Edge of the Network

The so-called "edge of the network" is both a moving target and a destination that carries different actual locations, depending on who's describing it. It's a moving target largely because of 60+ decades of technological advancement – what was "the edge" changed with each new chapter in capacity expansion, from microwave to coaxial tree-and-branch to modern HFC (Hybrid Fiber-Coax) topologies. Generally speaking, as capacity increases, the "edge of the network" moves closer and closer to consumers.

Network engineers tend to define "the edge" as being somewhere near where optical signals hand off to RF, or, near the output of optical nodes. Other disciplines define the "edge" as being at the output of the set-top box or broadband gateway. From a competitive perspective, there are still only three physical wires – three edges – that reach directly into U.S. households: The power line, the phone line, and the "cable" line. The power utility industry, despite commendable effort, has yet to successfully provide broadband connectivity. The telephone industry is showing, through its heavy investments in wireless 5G, what it knows to be true about the throughput potential of twisted-pair copper wires. Our network edge really is the only one that can consistently deliver high-bandwidth services, over a wire (or not), to IP/connected devices.

For the purposes of this paper, the "edge of the network" is located at the headend, where headend is defined as where we are actively transforming traditional CMTS devices, with integrated QAM modulator, into virtual CMTSs, where the modulation function is relocated into Remote PHY Devices, or RPDs.

The network edge is a high-stakes place, coveted by all in the broadband ecosystem, from cloud purveyors to networked gamers to any of a dozen adjacent industries that wish to be as close as possible to end consumers. As engineers, it is our responsibility to make sure our network edges are as healthy, fortified and ready as they can be, for whatever is coming. A big part of being healthy, fortified and ready is applying the principles of Continuous Integration and Continuous Deployment.

## 3. Why Continuous Integration/Continuous Deployment Matters

Continuous Deployment (CD) is a software engineering methodology that takes any code change that is deemed stable and (normally through parallel methods of Continuous Integration, or CI) and makes it available in production. This usually includes automated building, testing, and deployment, without human intervention.

Our CI/CD progression began as it often does within hardware-centric industries that began long before software engineering principles emerged: Organically, within certain parts of the company responsible for software delivery. In the beginning, many of the commercially available and/or open-source CI/CD systems were used throughout the organization (see Figure 1 as an example of the vast CI/CD landscape).

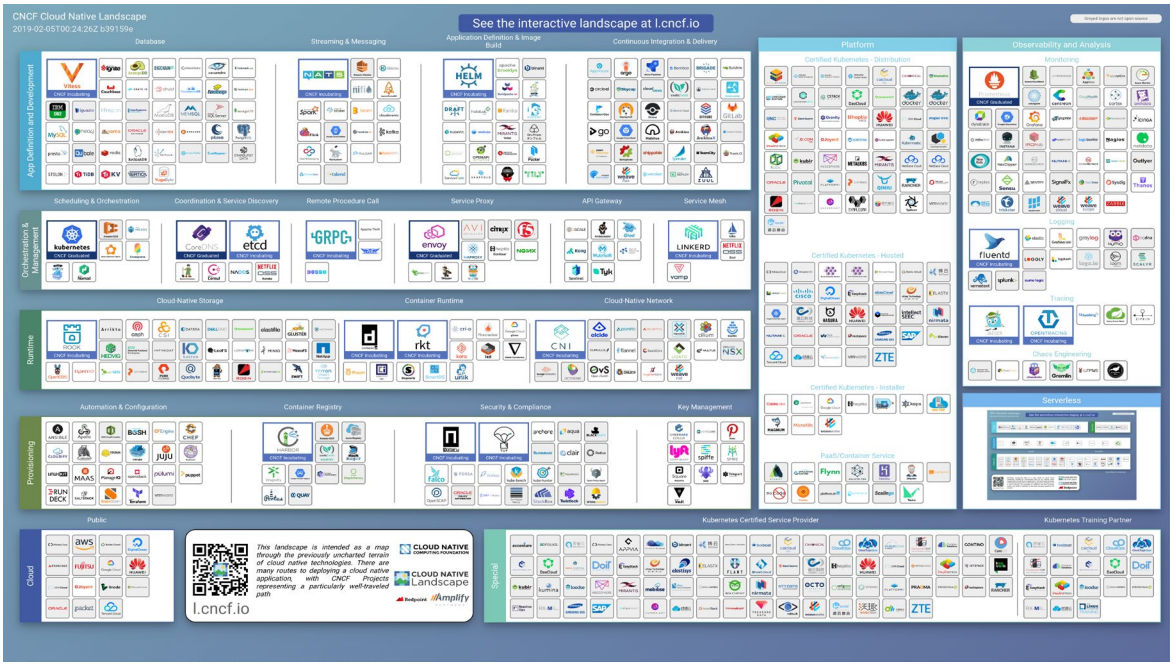


**Figure 1 - CNCF Cloud Native Interactive Landscape**

The practice of automating software delivery was one that was developed and documented outside the company, by the software industry at large. As more internal software teams began individually doing CI/CD themselves – including maintaining the cost of their own infrastructure to do so. Jenkins [1] (another common choice for implementing CI/CD within the engineering lifecycle) was one of the most popular options, as was Concourse, and a few others.

As more teams adopted CI/CD and related tooling, a need arose for a reasonably standardized pipeline tool. The main driver: Eliminating the technical debt associated with maintaining multiple infrastructures necessary to host different CI/CD tools, while creating a unified language that could be leveraged by all (or most.) By adopting a configuration-based CI/CD platform, component re-use can be applied to share many of the patterns and implementations that are very similar from one team to the next team. By establishing a shared platform and community around one tool, teams spend less time "reinventing wheels," among them code onboarding, because they can leverage already-public configurations.

As the reach of CI/CD platforms widened, a need arose to take stock of everything that was being done, in software, within Comcast's Technology, Product and Experience (TPX) organization. (Answer: A lot!) The CI/CD survey and its results happened within the construct of an internal TPX Architecture Guild, which consists of a group of engineers and technologists who were brought together to help guide technological decisions for the broader TPX organization.

The Architecture Guild uncovered a large amount of duplicate effort: Most teams were running their own instances of CI/CD software. The guild also identified varying degrees of implementation inconsistency: Teams were either as far along as deploying code into production every day (continuously deploying/integrating), or as far back as manually dropping compiled artifacts onto a web server.

Therefore, the guild recommended to Comcast TPX leadership that there be a dedicated team for maintaining CI/CD infrastructure, and that a single platform be chosen as the solution of choice. The recommendation was made with the idea that if you can get to a common platform and community, it would be easier to learn and onboard new teams, because they would only need to learn about a new CI/CD tool once. Questions like "has anyone done _____ with this platform?" could be answered by the more advanced users, further reducing the maintenance burden. The Architecture Guild was the ideal spot for a discussion as it was somewhat removed from the already formed opinions of specific teams. Asking team members to relinquish favored software tools, in favor of a different tool, is never easy; for more on this, see "The Tooling Abyss" by Joann Schuman, in this year's Fall Technical Forum papers.

Our then-Chief Software Architect and Senior Fellow, Jon Moore, was the main facilitator of the CI/CD unification discussions. As he describes it, in an interview conducted specifically to inform this paper, finding consensus on a unified CI/CD tool involved reconciling a couple of key points, including concerns around items like security and team separation. "The idea was that if we could get to a common CI/CD platform, with a community, it would be easier to learn and to on-board new teams, hand over ownership, because it wouldn't involve learning a new CI/CD tool," Moore explained [2].

## 4. The Architecture Decision Record (ADR)

With a few options then being proposed, an Architecture Decision Record (ADR) was formed to begin to track updates, offer reviews and other general comments. An ADR is a document that

captures decisions, including context (why was the decision made?) and consequences (what will adopting this decision impact?).

Since the ADR was in source control, interested/involved parties offered comments via pull requests and subsequent reviews. After a short period of proposals and reviews, a couple of options remained. Technically, any of the final options would have worked; as such, the goal was to pick the CI/CD tool that would cause the least amount of unhealthy friction, especially for those who would have to move over from another platform.

Instead of an outright vote, the guild proposed the use of a confidence poll, where each CI/CD software platform was ranked on a scale of 1-5, with a 5 representing, essentially, "the best," and 1 signaling "this would be a mistake." After rating the options on that scale, the scores closest to a 5 were interpreted as "acceptable to a majority of people." Ultimately, and instead of a straight-up vote for the "right" platform, Moore made the decision, based on the results of the CI/CD review process and confidence poll. As a result, the company now uses an open-source CI/CD tool called "Concourse" [3] to build CI/CD into various engineering lifecycles.

## 5. CI/CD at the Network Edge with vCMTS

With a tool of choice in place, teams could begin to implement an automation-driven software lifecycle to further increase the speed at which we could roll out changes. Such an effort was leveraged by several teams; in particular, and as it relates to this paper, a team that manage the Virtualized Cable Modem Termination System (vCMTS). Of course, there is a good deal of manual work to be completed beforehand. The racks that support the vCMTS need to be built. All the servers and switches that are part of the rack are shipped to headends. In the headend, wires are connected, to the RUs, power, and the GPS antenna. At that point, all the manual work needed for the connection is complete.

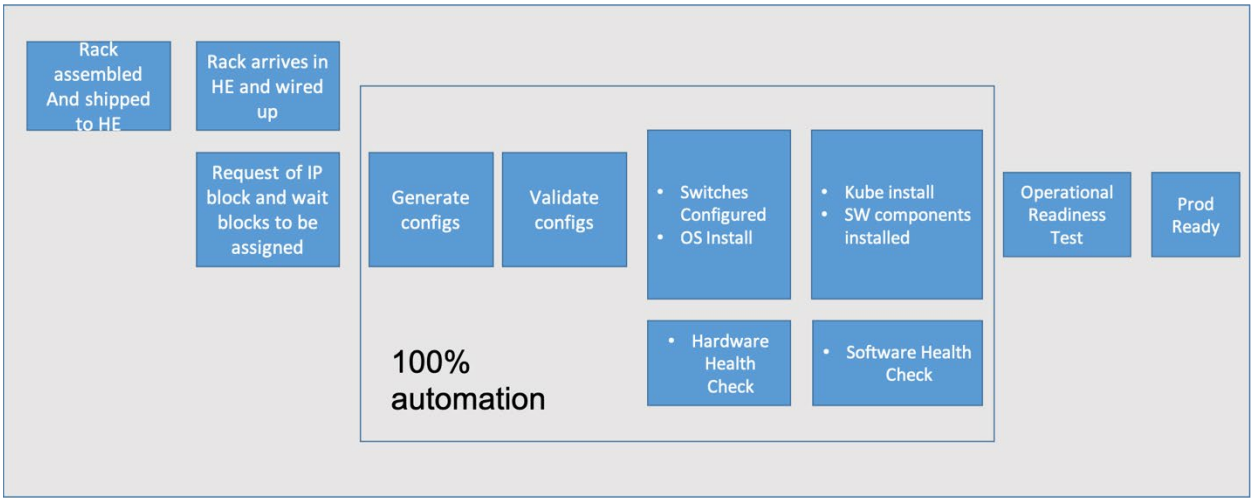


**Figure 2 - Automated vCMTS Cluster Build Process**

At this point, automation of the switch configuration is possible. Previously, if provisioning was needed, a ticket would need to be filed and a technician dispatched, to make the change onsite. An IP block would have to be requested and used to perform the requested provision. With automation, the IP block and configuration is known beforehand so the requested provision can be generated automatically. All of that now happens in Concourse, using a template to input the IP addresses, and to generate the configuration. Every cluster includes at least 2 switches and around 42 servers; up to 24 clusters can serve a headend. Looking at each pipeline (a set of tasks that run pre-defined operations), there is one pipeline per cluster. Each can have up to 192 vCMTSs. In that sense, each service group is a tiny vCMTS.

Every time there is a need to trigger a pipeline, a set of questions is asked: What am I deploying, and is it service-affecting or not? What kind of tests do I need to perform? The config, generated by Concourse, is then pushed to all the devices (servers and switches, numbering in the tens of thousands.) Afterwards, the OS (Operating System) is installed, and then software is installed. After automated hardware and software health checks are performed, the clusters are deemed production ready.

One thing to remember is all these different processes are resources. Concourse is a resource, running on servers. It needs a good amount of hardware to perform the operations being requested of it. One of the consequences of a shared CI/CD ecosystem is the sharing of resources. More resources could be allocated to speed up deployments to the vCMTS, but that would risk less availability of resources for other teams to use. This project in particular limits itself to 30 processes at a time.

For every change that needs to be made, a plan needs to be built to map out its journey to production. That mapping is fronted by a couple of questions, starting with reach: Changes are not rolled out to every single cluster at once. The focus is usually on new clusters that do not have new software yet. After clusters with older software are prioritized, the rest of the software updates are planned out. Based on the deployment type, and the service-affecting level, the deployment plan is calculated.

Once the deployment plan is approved, it is automatically rolled out during a maintenance window. First, a “pre” snapshot is taken of the current state of the systems in production. Health checks are performed, to make sure current systems are working as expected. Every change is made inside of a maintenance window. During that period, alarms are silenced because of the need to take systems down. After alarms are paused, the change is deployed to production. After the deployment, multiple post-deployment checks are run, with the aid of machine learning (ML). If there is a failure, there are 3 retries to access the systems and to ensure that the any change-related activity has settled down. Any failure that happens is logged, to let us know exactly what happened.

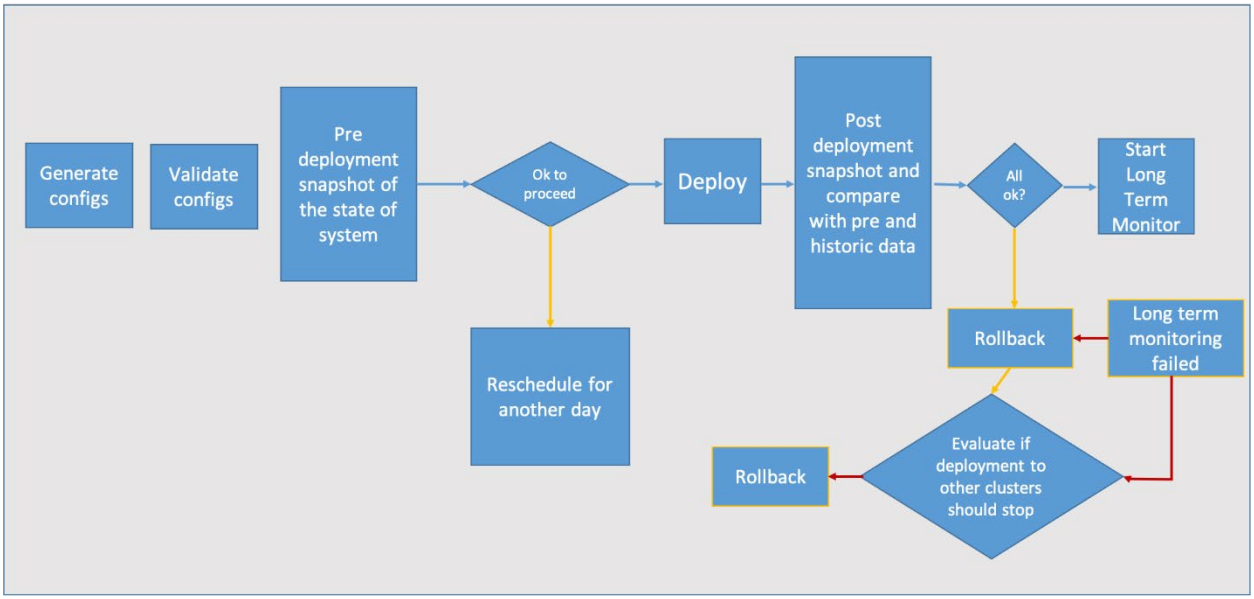


**Figure 3 - Software and Network Change Deployment Pipeline**

The process of connecting the RPDs ("Remote PHY Devices" / QAM modulators) can then begin. On a daily basis, and during the transition from integrated (which is to say, "traditional") CMTSs to their virtual counterparts, around 20 to 30 new clusters are being built in various places. Given common errors that can occur during config generation, or when scripts are run in a different order, or when switches malfunction, this process would traditionally take about three to four days. With the process of automating vCMTS deployments through Concourse, changes like these take anywhere from 2-4 hours. The variability of time depends on the platform, how many other jobs are running, platform health, etc. The beauty of automation is that there is a single source of truth from which configurations are read.

In the case that a rollback is needed for an upgrade that was deployed, for instance, checks exist to ensure that the rollback instances are healthy and ready to receive traffic. If there needs to be a change to the vCMTS core software, there's a check to see if it supports all CPEs (Customer Premise Equipment / set-tops and gateways). With a CI/CD methodology in place, you would expect any rollback to be done as quickly as (if not faster than) the roll out to production. In this case, rolling back is a progressive process. There is sampling of all kinds of data (with the aid of machine learning) that gives visibility around what specific failures there were and whether a rollback is needed in a certain area.

A zero-downtime headend necessarily means removing changes that can cause stoppages, and removing elements that can become points of failure. The iCMTS, with integrated RF modulators, was an example: From the analog node, you connected all the RFs. With vCMTS, the RF portions are removed from the CMTS, and put into RPDs. The physical component – the PHY, is in the RPDs; the nodes, in fiber-deep configurations, could be on a pole, in a pedestal, or underground. Upgrading them should be non "hitful" – which is to say, not service impacting.

At the headend, where the CMTS servers run, the intent is to complete upgrades with minimal scheduled downtime, on the order of 3 minutes. With a traditional iCMTS, the downtime tied to a scheduled upgrade is typically around 2 hours; so far, with the vCMTS, scheduled downtime related to upgrades is 15 minutes.

Our ultimate goal is to reduce headend downtime to close to or exactly zero. If there a catastrophic failure occurs with a small "blast radius," can take 15 minutes to redeploy software to a vCMTS, rather than take the risk of a lengthy time to diagnose and fix.

We are currently (summer 2021) also working on an in-service upgrade functionality. The intent is to get to 15 minutes or less of downtime for 2 components: The RPD, and the vCMTS, to simultaneously address the service-affecting route.

## 6. Conclusion

The journey into adopting a culture of Continuous Innovation and Continuous Deployment has been one that mirrors the actual process of continuously iterating and deploying: Problems are defined by processes or issues in progress. Updates and improvements are added frequently, without sacrificing stability. Changes are rolled out while ensuring reliability and adequate steps to rollback. Rolling out CI/CD (along with a unified platform and growing community) was done without jeopardizing the customer experience. What would've once been considered a phenomenon is now common practice in software engineering. Those lessons are now able to be transferred into the hardware side of engineering, to help drive improvements at a wider scale and without the need of constant manual intervention.

I would like to take the time to thank those who lent their time and expertise into the completion of this paper and presentation. Frankyln Athias, Sherita Ceasar, Leslie Ellis, Jon Moore, Max Knee and Bhanu Krishnamurthy.

# Abbreviations

| | |
|---|---|
| ADR | Architecture Decision Record |
| CI/CD | Continuous Integration/Continuous Deployment |
| CPE | Customer Premise Equipment |
| DAA | Distributed Access Architecture |
| HFC | Hybrid Fiber Coax |
| iCMTS | Integrated Cable Modem Termination System |
| PHY | Physical Circuit |
| QAM | Quadrature Amplitude Modulation |
| RF | Radio Frequency |
| RPD | Remote PHY Device |
| vCMTS | Virtualized Cable Modem Termination System |

# Bibliography & References

[1] https://landscape.cncf.io/

[2] Informational interview, Jon Moore, Senior Fellow, Comcast

[3] https://concourse-ci.org/

Additional references:

[4] *Solving The Mysteries of the Distributed Access Architecture*; Matthew Stehman, Ramya Narayanaswamy, Jude Ferreira, Robert Gaydos, 2021

[5] *The Tooling Abyss*; Joann Shumard, 2021

[6] *Humanoids Optional: Deploying vCMTS at Scale with Automation*; Bhanu Krishnamurthy, Gregory Medders, 2021

[7] *Architecture with 800 of My Closest Friends;* The Evolution of Comcast's Architecture Guild; Jon Moore, 2019: https://www.infoq.com/articles/architecture-guild-800-friends/

 