Automating R-PHY in the Transition to vCMTS

A technical paper prepared for presentation at SCTE TechExpo24

Douglas Johnson
VP, Software Architecture
Vecima Networks
douglas.johnson@vecima.com

Moe Iqbal
Sr Lead Architecture
Cox
moe.iqbal@cox.com

Table of Contents

List of Figures

1. Introduction

Automated configuration management is a powerful approach to managing systems that replaces manual configurations with an automated, code-driven processes. This transition offers numerous benefits, including cost reduction, increased agility, improved reliability, configuration compliance, and enhanced security.

Automated configuration is an organizational investment that goes beyond simple tool adoption. It involves applying Software Development Lifecycle (SDLC) approaches to manage complex, organization-wide configurations applicable to the virtual Cable Modem Termination Systems (vCMTS) and supporting infrastructure. This process requires a systematic approach to planning, developing, testing, deploying, and maintaining configuration changes, ensuring they are consistent, secure, and scalable.

For many organizations, adopting automated configuration management also necessitates additional training and skill set alignment. Employees may need to learn new tools and technologies, such as Infrastructure as Code (IaC), Continuous Integration (CI) / Continuous Delivery (CD) pipelines, and specific automation frameworks like Ansible or Puppet. Moreover, there is often a cultural shift required, as teams adopt DevOps practices that emphasize collaboration, continuous improvement, and a shared responsibility for system reliability and performance. This alignment not only improves the efficiency of configuration management but also enhances the overall agility and responsiveness of the organization, allowing it to better meet changing business needs and technological advancements.

This paper mixes a practical-approach with foundational outcome-oriented discussions on organizational change. The included Pipelines are based on implemented or aspirational Pipelines as a real-world example of using Pipeline-based automation. We hope this approach enables Operators to customize the automation approach to their organizational needs and goals.

2. Automating Configuration

Investing in automated configuration management brings many benefits, including increased agility, responsiveness, reliability, resilience, consistency, standardization, and security. To achieve these benefits, organizations must allocate design and development resources to create seamless and empowering automation workflows. This investment will drive the organization forward, making it more efficient and competitive.

Automation enables faster deployments, allowing the organization to quickly adapt to changing customer needs and market conditions. This capability is particularly vital in today's fast-paced business environment where time-to-market can be a competitive differentiator. Moreover, automated systems can scale seamlessly to handle increased workloads without requiring proportional increases in manual effort, ensuring that the organization remains agile and responsive.

Improved network reliability and resilience are achieved through proactive monitoring and maintenance facilitated by automation tools. These tools can continuously monitor the network, perform routine maintenance tasks, and detect issues before they escalate into significant problems. Automated systems also support automatic failover and recovery mechanisms, which ensure minimal disruption in the event of a failure. This proactive approach reduces downtime and enhances overall service reliability.

Automated configuration management enforces uniform procedures across the infrastructure, ensuring that all configurations adhere to predefined templates and best practices. This standardization minimizes the risk of human error, leading to more reliable and predictable outcomes. By reducing the variability

introduced by manual interventions, organizations can achieve a higher level of operational stability and efficiency.

Enhanced security and compliance are paramount in today's regulatory landscape where broadband services are considered critical infrastructure. Automated configuration management ensures that security policies are consistently applied across all systems, significantly reducing vulnerabilities and improving the organization's security posture. Regular, automated compliance checks help maintain adherence to regulatory requirements, thereby minimizing the risk of non-compliance penalties. By embedding security into the automation process, organizations can achieve a higher level of protection and regulatory compliance with less effort.

Implementing automated configuration management within an organization requires adopting a software development mindset and workflow toward configuration management tooling. Particularly suited are the DevOps and DevSecOps methodologies because these approaches foster collaboration, streamline workflows, and integrate security from the start. DevOps bridges the gap between development and operations teams, ensuring faster, more reliable deployments through continuous integration and delivery (CI/CD). This cultural shift promotes efficiency and consistency, which are crucial for managing configurations automatically. DevSecOps extends this by embedding security practices into the development lifecycle, ensuring that security is not an afterthought but an integral part of every process.

By leveraging these modern software development practices, organizations can achieve robust, secure, and scalable automation, ultimately enhancing their overall performance and resilience.

2.1. DevOps / DevSecOps

Successfully automating configuration management is deeply intertwined with successfully bringing DevOps and DevSecOps practices into an organization. These methodologies are not just technical frameworks but cultural shifts that emphasize collaboration, automation, and continuous improvement, all of which are essential for effective automated configuration management.

Adopting DevOps within your organization is a transformative step towards breaking down the silos between network architecture, development, and operations teams. This approach fosters a unified, efficient environment where continuous integration (CI), continuous delivery (CD), and infrastructure as code (IaC) become the standard. These components are the backbone of automated configuration management, ensuring that infrastructure and applications are deployed quickly, reliably, and consistently. By integrating DevOps practices, your organization can streamline workflows, reduce errors, and accelerate the deployment process, leading to enhanced agility and responsiveness in meeting subscriber needs.

However, it's not enough to focus solely on speed and efficiency; security must be an integral part of this transformation. This is where DevSecOps comes into play. DevSecOps extends the principles of DevOps by embedding security directly into the CI/CD pipeline and infrastructure management processes. This approach ensures that security considerations are built into every stage of the development lifecycle, rather than being an external concern or, worse, an afterthought. By incorporating automated security checks and compliance validations into the development process, DevSecOps helps safeguard your systems against vulnerabilities and ensures regulatory compliance.

To truly leverage the power of automated configuration management, your organization should consider adopting these modern methodologies. Embrace DevOps to foster a culture of collaboration and efficiency, and integrate DevSecOps to embed security into your development processes. This dual focus will not only enhance your operational capabilities but also build a resilient, secure foundation for future growth. Now is the time to invest in these practices, empowering your teams to develop robust, automated

configuration management tools that drive the organization forward, ensuring reliability, consistency, and security in every deployment.

2.2. Open Loop / Closed Loop

Open loop configuration management and closed loop configuration management are two approaches to managing network configurations, each with distinct characteristics and implications. Open loop configuration management relies on manual or scripted changes to maintain system configurations, lacking continuous feedback mechanisms. This approach involves periodic checks and audits to ensure the system aligns with the desired state, but it is prone to configuration drift and requires ongoing manual intervention to correct discrepancies. In most organizations current configuration management practices are open loop.

In contrast, closed loop configuration management integrates continuous monitoring and automated remediation to maintain the system's desired state as defined by declarative configuration files. This approach creates a feedback loop where the actual state of the system is constantly compared to the desired state, and any deviations are automatically corrected in real-time. This ensures high consistency, reliability, and resilience, as the system is always aligned with the predefined configurations. The continuous enforcement of the desired state prevents configuration drift and reduces the need for manual interventions, making closed loop systems more robust and efficient.

In closed loop systems, declarative state-based configuration plays a crucial role in enabling continuous, automated enforcement of configurations, ensuring the system remains in the desired state without manual effort. This continuous alignment is facilitated by real-time monitoring and automated corrective actions.

Integrating AI and machine learning (AI/ML) into closed loop configuration management further enhances its capabilities. AI/ML can analyze historical data to predict potential issues, optimize resource allocation, and enforce security policies automatically. These technologies enable proactive issue detection, intelligent remediation, and dynamic scaling, all of which contribute to a more efficient and secure system. The continuous feedback loop in closed loop systems provides the necessary data for AI/ML algorithms to learn and improve, making AI/ML integration feasible and beneficial. In contrast, the lack of continuous monitoring and feedback in open loop systems prevents effective utilization of AI/ML technologies.

To ease a transition from a more traditional open loop to a closed loop system, or for new closed loop development features, an organization can consider implementing a "man in the middle" approach where automated configuration changes by code are audited by a human before deployment. This method addresses concerns about potential bugs or runaway code causing widespread outages by adding a layer of human oversight to catch errors that automated systems might miss. It allows for a controlled, gradual implementation of automation, building trust and confidence in the system’s accuracy and reliability.

The human auditing approach helps mitigate risks by ensuring that all automated changes comply with organizational policies and maintain high standards during the transition. This strategy provides valuable feedback for improving the accuracy and effectiveness of the automated system over time, which gradually reduces the need for human intervention as the system becomes more reliable. By incorporating human review, a reluctant organization can build trust in the automation process, as stakeholders can observe the system's reliability and effectiveness firsthand. Over time, as the automated system consistently demonstrates its accuracy and compliance through human-audited changes, the organization can gain confidence in fully transitioning to a closed loop configuration management system.

2.3. Security

Effective configuration management is vital to establishing and maintaining the security of systems. Security-focused configuration management can be broken down into four key steps:

1. Planning and Governance,
2. Identifying and Implementing Configurations,
3. Controlling Configuration Changes,
4. and Monitoring and Compliance Checks.

Automation can play a key role in steps 2, 3, and 4, significantly reducing the costs of implementing secure configuration management within the organization.

Planning and Governance involves defining security requirements and policies, setting objectives, understanding compliance and regulatory requirements, and establishing a clear roadmap for secure configuration management. This step lays the foundation for all subsequent activities and cannot be automated.

Identifying and Implementing Configurations entails specifying the desired state of systems and components based on security requirements. Automation helps ensure consistent and accurate application of these configurations across the entire infrastructure, reducing the likelihood of human error.

Controlling Configuration Changes focuses on managing changes to the system configurations to prevent unauthorized or harmful modifications. Automated tools can enforce policies, track changes, and provide real-time alerts, ensuring that only approved changes are implemented.

Monitoring and Compliance Checks involve continuously overseeing the system configurations to detect deviations from the desired state and ensure ongoing compliance with security standards. Automation streamlines these tasks by providing continuous monitoring, automated compliance checks, and instant remediation of identified issues.

As NIST Guide for Security-Focused Configuration Management o Information Systems (SP-800-128) states, “The configuration of a system and its components has a direct impact on the security of the system.” By leveraging automation in these key steps, organizations can enhance security, reduce costs, and maintain a robust and compliant configuration management process.

2.4. CI/CD

Continuous Integration (CI) is a key DevOps practice that involves developers frequently merging their code changes into a shared repository, often multiple times a day. Each integration triggers automated builds and tests, enabling rapid detection and resolution of errors. This practice aligns with DORA (DevOps Research and Assessment) principles, which emphasize the importance of reducing deployment pain, improving code quality, and shortening integration times. By integrating code regularly, teams can catch and fix bugs early, maintain a high-quality codebase, and enhance collaboration. CI is foundational to automation in software development, as it ensures that new code changes do not introduce errors, thus facilitating smoother and more reliable deployments.

Continuous Delivery (CD) builds on the foundation of CI by automating the entire software release process, ensuring that code changes are automatically prepared for deployment to production. This involves deploying builds to staging environments where they undergo further automated and manual testing to ensure they are production-ready. DORA’s research highlights that high-performing teams implement CD to achieve faster lead times, higher deployment frequency, and lower change failure rates.

By automating the deployment pipeline, CD reduces human error, enhances consistency, and allows for rapid, reliable releases. This practice is crucial for maintaining the security and compliance of automated configuration management systems, as it ensures that all changes are rigorously tested and verified before being deployed to production.

2.5. vCMTS APIs and Infrastructure as Code

Having Application Programming Interface (API) functionality on vCMTS is critical for automated configuration management to allow and enable automated configuration tools direct and structured access to the configuration and operational state of the vCMTS. APIs support structured data representation, often using formats like JSON or XML, which provide standardized ways to describe, manipulate, and transfer hierarchical and complex configuration data across various systems and network devices. This consistency is crucial for the automation process, enabling infrastructure teams to develop and deploy scripts and applications that automate repetitive configuration tasks, thereby supporting the principles of Infrastructure as Code (IaC).

Infrastructure as Code is a key concept in modern IT operations where infrastructure configurations are managed and provisioned through code rather than manual processes. This approach treats infrastructure setup and management in the same way software code is handled: using version control, automated testing, and continuous integration/deployment (CI/CD) pipelines. IaC is important because it brings consistency, scalability, and repeatability to infrastructure management, reducing the chances of errors that typically occur with manual configuration. By using APIs within an IaC framework, organizations can ensure that configuration changes are scripted, version-controlled, and automatically deployed across environments, leading to more reliable and secure IT operations.

To enable IaC, vCMTS APIs need to fulfill a few properties:

- Powerful and complete access to the configuration and operational models of the vCMTS,
- Transaction-based operations, allowing configuration changes to be batched and ensuring that these changes are either fully applied or rolled back in case of errors, which helps maintain system stability and consistency,
- Support secure transport protocols, such as SSH or HTTPS, safeguarding configuration data during transmission, enhancing the security of the management process,
- Use standardized data formats, like XML or JSON, which facilitates ease of human understanding and allows for effective audits of configurations across various devices or systems.

vCMTS API Protocols such as NETCONF, RESTCONF, GRPC, and REST are all able to fulfill these operator needs for IaC.

3. Automation through Github

3.1. Gitops

GitOps is a modern approach to continuous delivery and operational management that uses Git as the single source of truth for declarative infrastructure and configuration. It combines DevOps practices with Git-based workflows to automate infrastructure provisioning, configuration, and deployment. The key idea behind GitOps is to manage infrastructure configuration as code, stored in Git repositories, and use automated processes to ensure that the actual state of the system matches the desired state as defined in the repositories.

Git is a tool, commonly used by software developers, to manage and track changes in code repositories. It is capable of significant, historical, and detailed change tracking of any text-based file including detailed “diffs” and historical notes tracked with each change. It helps multiple people work on the same project without overwriting each other's work and keeps a history of every change made, so you can go back to any previous version if needed. Git stores all this information in repositories, which are like project folders that contain not only the files but also the entire history of changes made to those files. This makes collaboration, troubleshooting, and improving code much easier and more organized.

Git was created in 2005 by Linus Torvalds, best known for inventing Linux, to be a fast, distributed, source code repository that could handle large-scale projects with numerous contributors. It’s now the most used version control system in the world. GitHub was founded in 2008 and was acquired by Microsoft in 2018. GitHub is a social platform, Git workflow, and CI/CD framework built on top of Git and is the largest Open-Source and proprietary source code hosting platform in the world. Importantly, GitHub offers a private Enterprise version which has all the high-availability, reliability, and features of the public GitHub but in a closed, secure, private instance for businesses.

GitOps, using Git and GitHub, is a powerful approach that brings the benefits of version control, automation, and continuous delivery to infrastructure and application management. By using Git as the single source of truth and leveraging automated tools to ensure the desired state is always applied, organizations can achieve greater reliability, security, and efficiency in their operations.

3.2. Github Actions

GitHub Actions is a powerful automation platform integrated directly into GitHub repositories, enabling developers to create custom workflows that automate their software development processes. It allows users to define workflows in YAML files (a simple, structured configuration markup language), which can be triggered by various events such as pushes, pull requests, or scheduled tasks. These workflows can automate tasks like building, testing, and deploying code, making continuous integration and continuous delivery (CI/CD) seamless and efficient. GitHub Actions provides a vast marketplace of pre-built actions, as well as the flexibility to write custom actions, enhancing collaboration, productivity, and the reliability of the development pipeline by integrating directly with the tools and processes developers already use.

GitHub Actions can be an effective tool for implementing an automated configuration management system by leveraging its CI/CD capabilities and seamless integration with Git repositories. GitHub Actions allow automation workflows to be designed and developed with each step creating a version controlled intermediary asset, usable for audit and compliance purposes.

In GitHub Actions defines workflows in YAML format. These files define the steps for automating configuration management tasks such as provisioning infrastructure, applying configuration changes, and performing compliance checks. Workflows are setup to trigger on specific events, like commits, GitHub interactions, or schedule-based triggers. By chaining commits and events, a full workflow can be developed with intermediary steps recorded in durable version control.

Github Actions can also run automated tests on configuration changes to verify their correctness before deployment. This can include syntax validation, policy checks, and integration tests with lab test benches. Once all the validation is complete, the Github Action workflow can push the configuration change to production systems and monitor the application through automated checks and metrics analysis. Finally, Github Actions can be designed to provide notifications to various communication systems (such as Slack, E-Mail) about changes as they move through the automation workflow.

Because git version control is the backbone of the automation workflow full audit trails with complete historical logs are available at all times to ensure engineers can inspect and understand the automation and gain trust in the overall system.

3.2.1. Self-Hosted Runners

An important aspect of using GitHub Actions to drive automation pipelines is using GitHub Actions self-hosted runners. Self-hosted runners offer organizations the ability to run GitHub Actions workflows on their own infrastructure rather than using GitHub's hosted runners. This setup provides greater control over the environment in which workflows execute, allowing for customized hardware configurations and specific software dependencies. Importantly, self-hosted runners can safely access resources on a private network, such as the vCMTS Application Programming Interfaces (APIs), internal databases, and other services which is crucial for automation pipelines that need to interact with internal systems, allowing seamless integration with existing infrastructure.

4. Pipeline Visualization

The most powerful way to align an organization and make progress in automation is to rally around pipeline-based deployment diagrams. Visualizing automated configuration pipelines makes it easier for all stakeholders to understand the processes involved. This common understanding helps in aligning goals, identifying potential bottlenecks, and discussing improvements. A visual pipeline provides clear visibility into where changes are in the process, who is responsible for each step, which steps are automated or manual, and where issues may have occurred in the process. This transparency helps teams quickly identify and address problems, reducing downtime and improving overall efficiency.

To create the first pipeline, an organization should start by identifying key processes and objectives that the automated configuration pipeline needs to address. This involves gathering input from various stakeholders, including engineers, developers, testers, operations, and business leaders, to understand the requirements and expectations around the automation tooling and investments. During realization of the Pipeline new requirements will be discovered and the organization will gain a better understanding of automation, so it is essential to success to adopt an Agile/DevOps approach, not a Waterfall, to the steps detailed below.

Each organization will have multiple and unique pipelines. This paper includes real-world pipeline examples, but each pipeline must be tailored to each unique operator. Use the following framework to build an automation pipeline within your organization.

It’s important to design and map discrete finite-state machine (FSM) states as part of your pipeline design. These states are shared internally within the organization, are well-known, and are universally used to help the many parts of the organization understand and track the automation process without having to fully understand each discrete part of all the Pipelines. The design of the FSM states must also embrace failure states, since there will always be failures that automation cannot manage itself. By discretely and visibly designing failure states into the Pipeline FSM the organization and understand and respond to failures in a measured and responsible manner.

It can be useful to keep Transaction IDs (txid) with pipeline executions. When developing internal tool APIs to support the Pipelines, consider allowing/requiring the txid be provided in all API calls to help with troubleshooting tools and Pipeline integrations.

Key iconography should be defined by the organization to reflect their needs. The icons should be used uniformly across all Pipelines to aid understanding. The icons provide at-a-glance understanding of the

primary actor or interaction at each stage of the Pipeline. Consider icons for people, teams, code, and assets/outputs.

4.1. Define Pipeline Objectives and Requirements

The goals of the pipeline should be determined before starting, such as achieving faster RPD deployment times, reducing Service Group errors, or improving overall quality assurance. It is important to identify the scope, which includes specifying which configuration, systems, services, or components will be integrated into the pipeline. Engaging all relevant stakeholders is essential to gather comprehensive requirements and understand the specific needs and expectations of different teams, ensuring that the pipeline aligns with organizational goals. This often results in a few simple paragraphs and/or bullet points that help scope the intent of the Pipeline.

4.2. Tools and Technologies

A Pipeline needs to be realized through tools and code. The first consideration is choosing a version control system, such as GitHub, to manage code and configuration changes effectively. Next, CI/CD tools need to be chosen based on factors like integration capabilities, ease of use, and scalability. Popular options include GitHub Actions, Jenkins, and GitLab CI. Additionally, understanding how to interact with key systems such as the vCMTS, Networking components, and various quality Probes is imperative to understanding if a Pipeline step is even feasible. Particularly, consider if additional investments in passive Probe technologies and tools are needed to enable automated closed loop systems.

4.3. Design the Pipeline

Designing the pipeline workflow involves defining the stages that the code will go through in a visual whiteboarding and diagramming tool (such as Lucid, Miro, or Figjam). It is important to decide what events will trigger the steps of the pipeline, including but not limited to

- vCMTS triggers: Pre-defined Events, Alarms, or telemetry thresholds
- IaC triggers: commits, pull requests, or
- Regularly scheduled tasks.

Conditions and gates should also be set for transitioning between stages, including requirements for automated tests, human reviews, or manual approvals, ensuring that quality checks are in place at each stage. The pipeline is designed abstractly, but concretely. Each Step visualized as a box, each transition as a line with the trigger noted.

Additionally, the FSM for the Pipeline should be designed at the same time. It is also common for multiple Pipelines to use a shared FSM when they are operating within the same business domain.

4.4. Implement the Pipeline

Implementation is focused on realizing each Step and trigger through a development process. Consider investing in reusable component libraries, especially for common Probes that may be used in multiple Pipelines. Reusable components may be Cable Modem health checks, passive network probes, and vCMTS health metrics. All step implementations are done and managed through source control (such as GitHub).

4.5. Monitor and Improve

Monitoring the pipeline's behavior and performance is essential for maintaining its effectiveness. This involves implementing tools to track key metrics like running times, test coverage, and execution frequency, as well as detecting failures and issues. Establishing feedback loops with stakeholders allows for continuous improvement by gathering insights and making necessary adjustments to the pipeline. Finally, maintaining detailed documentation of the pipeline setup, processes, and any changes made over time is crucial for transparency and ongoing optimization. This documentation serves as a valuable resource for troubleshooting and onboarding new team members.

5. Example DAA vCMTS + RPD Pipelines

The iconography for this pipeline is defined in the following figure. The Automated (code) stages run on GitHub Actions self-hosted runners or as long-lived microservices that the runner can reach for API access. Stored assets are kept in GitHub for other stages to consume, auditing, compliance, or human review. Message Queues allow for asynchronous communication between systems, usually across business units. Scheduled, Approvals, or manual steps are also able to be identified quickly. The Poll/wait steps execute in a polling loop waiting for external stimulus or eventually timeout.

Figure 1 - Pipeline Iconography

All the Pipelines in these examples use the same FSM states. These states are stored in a system-of-record that all parts of the organization have access to (a ticketing system) and the ticket states are well-known within the organization. Some states are still largely manual actions by skilled personnel, such as the actions to leave the Validation Failed or Rollback/Failed state.

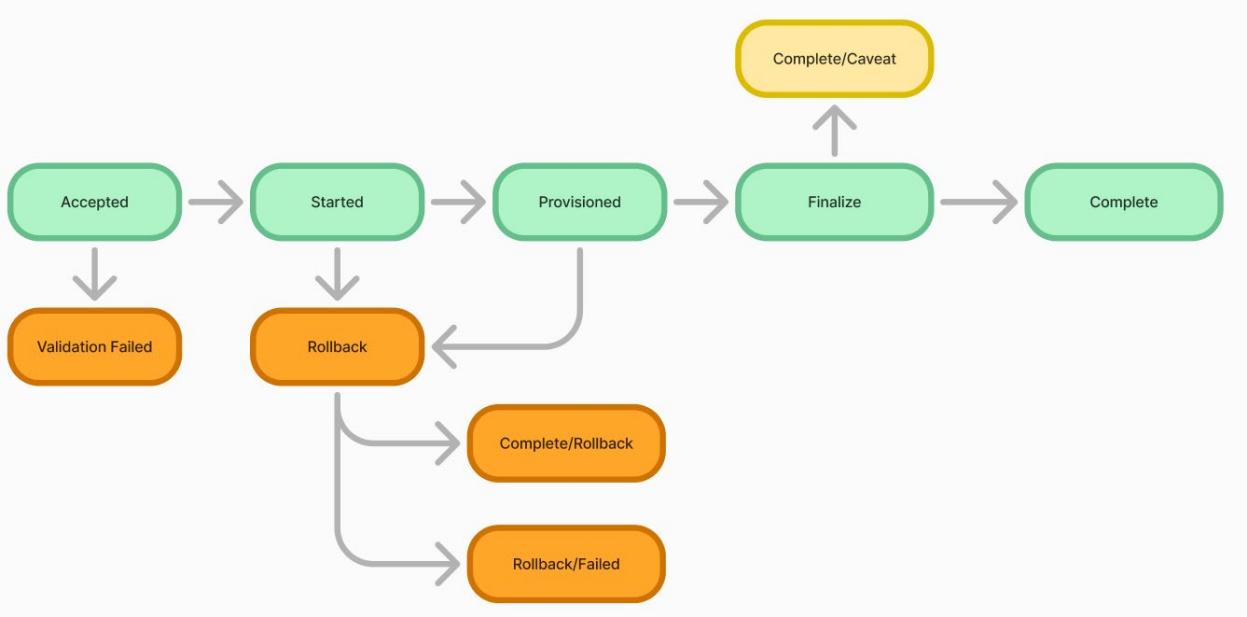

Figure 2 - Pipeline FSM

5.1. RPD Install / Birth Certificate

The purpose of this Pipeline is to define an automated mechanism for the organization to action, configure, and onboard a new RPD installation. It starts with Node Actions input, in CSV format or via a provisioning API interface, that includes a bulk definition of multiple RPD node configurations. During the Pipeline, OSP (Outside Plant) technicians input key RPD information (via a QR code scan or manually) through a cellphone application during the physical installation procedure. For each RPD in the Node Action CSV input an instance of the Pipeline is (logically) executing. Any individual RPD Pipeline may take a few days, a week, or more to fully complete.

The Pipeline is broken down into multiple stages, each with multiple steps. The full pipeline is below, however, to fit within a document format the stages are presented in discrete figures and some steps have been simplified.

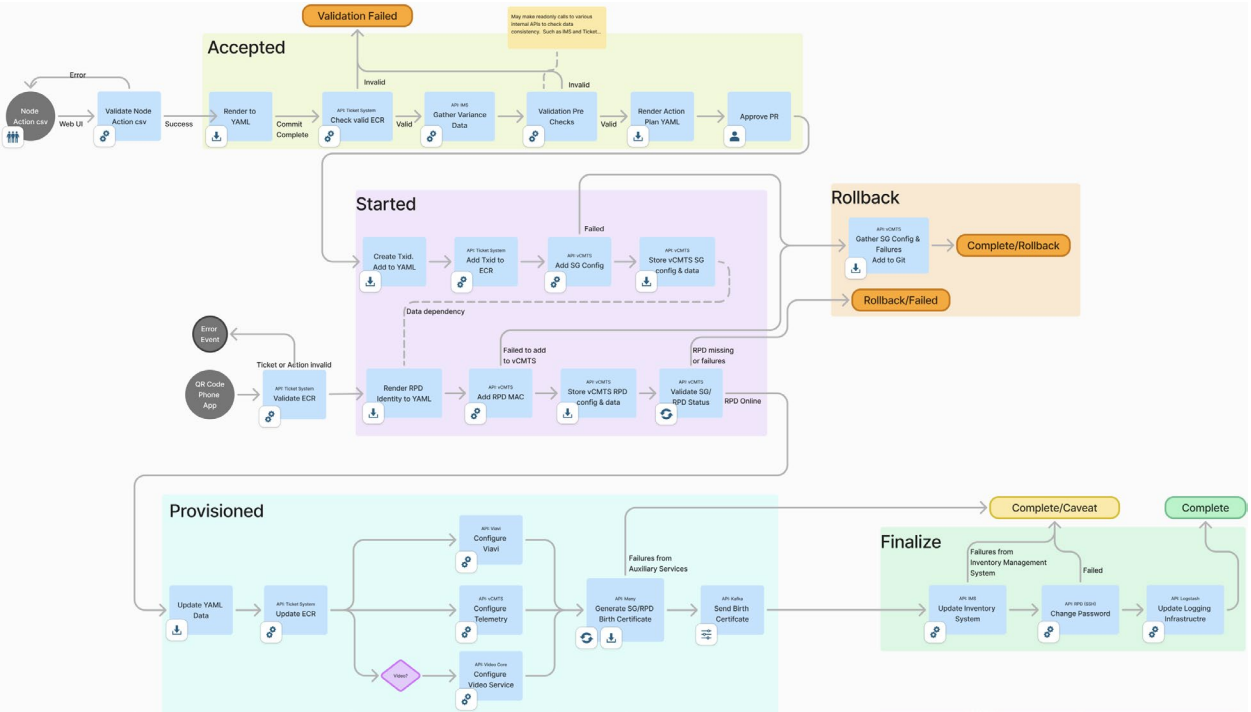

Figure 3 - Full RPD Install Pipeline

In the first stage, the Node Actions are provided in CSV format. These are converted to an expanded YAML format and committed individually (per Service Group (SG) or RPD) to GitHub for traceability and auditing purposes. The YAML is then processed through various discrete steps which either enrich the YAML or validate the configuration against internal systems. Finally, a complete, actionable, YAML file is created and submitted to GitHub. Optionally this final, enriched YAML may require a human to review and accept the PR (Pull-Request) through the GitHub UI before the next stage is executed.

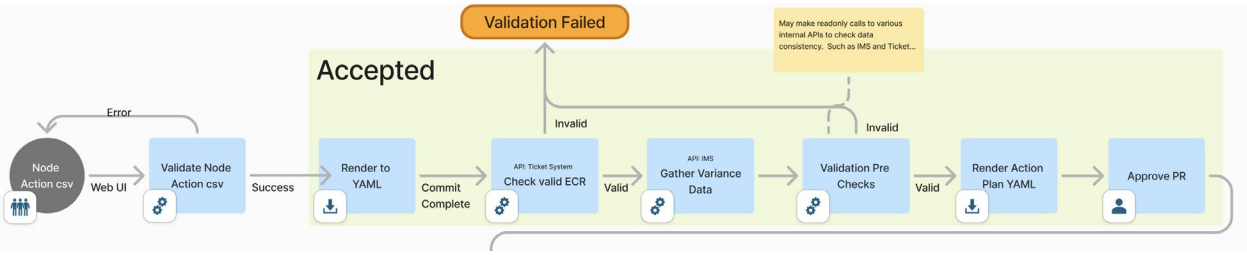

Figure 4 - RPD Install Pipeline: Accepted

The next stage is focused on vCMTS configuration, resource assignment and allocation, then finally getting an RPD online. This pipeline highlights a "disjoint" process: there is often a noticeable delay (days/weeks) between the vCMTS being pre-provisioned for the RPD and the physical act of installation of the RPD.

The network identity (MAC, Serial, etc) of the RPD is not known until the QR Code is scanned with a Phone App and matched to an install ECR (Engineering Change Request / Service Ticket), which then uses a service to update the GitHub YAML driving the rest of the pipeline. The OSP Technician uses the

Provisioned state visible in their tools to signal when the RPD is physically online and they can move to the next ticket/job.

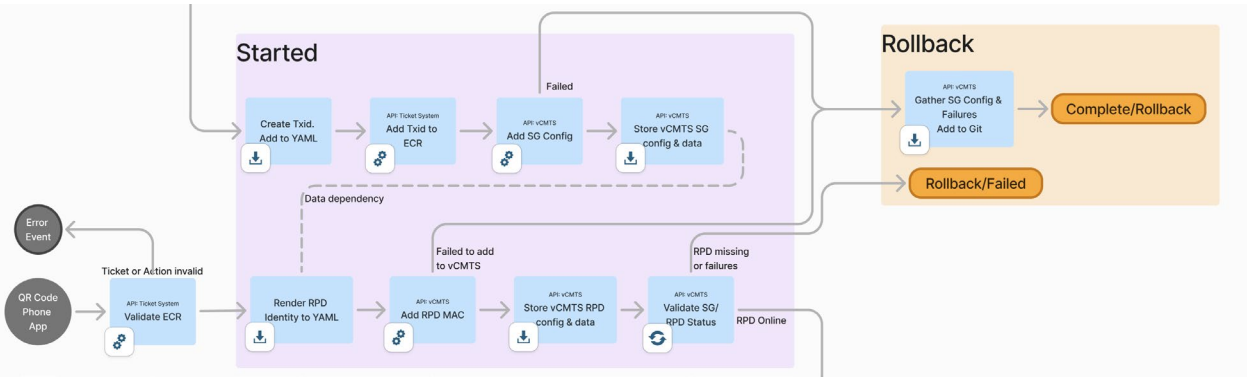

Figure 5 - RPD Install Pipeline: Started

The final stage handles post-install checks, onboarding, and the "birth certificate" for the RPD.

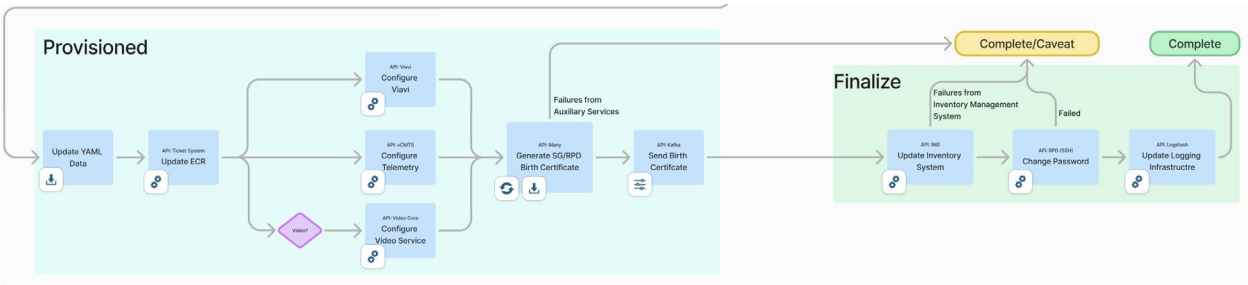

Figure 6 - RPD Install Pipeline: Complete

The Complete/Caveat state is used to signal that the RPD is online and the SG functional, but there was a failure automatically integrating the SG/RPD into various back-office systems. This state is handled by organizational personnel through a ticketing system to reconcile the failure. The Kafka birth certificate metadata (in JSON format) is used by the onboarding workflow microservices; It is also used by supporting business units such as compliance, billing, and NOC in various asynchronous processes.

5.2. RPD Deletion / Removal

The purpose of this Pipeline is to define an automated and safe process for removing an RPD from the network. This includes maintaining key RPD identity information, traceability, and compliance records. This does not include physical removal or shipping.

Making the deletion call (Phone App or an API call) will synchronously update the main systems (Ticket and vCMTS) in a single transaction. If any of those fail the error is immediately returned (and events/logs are generated). In the case of success, a "Death Certificate" is stored in GitHub and sent on a Kafka topic. Asynchronously, Death Certificates drive the de-provisioning of auxiliary systems and is used by other business units to drive other workflows.

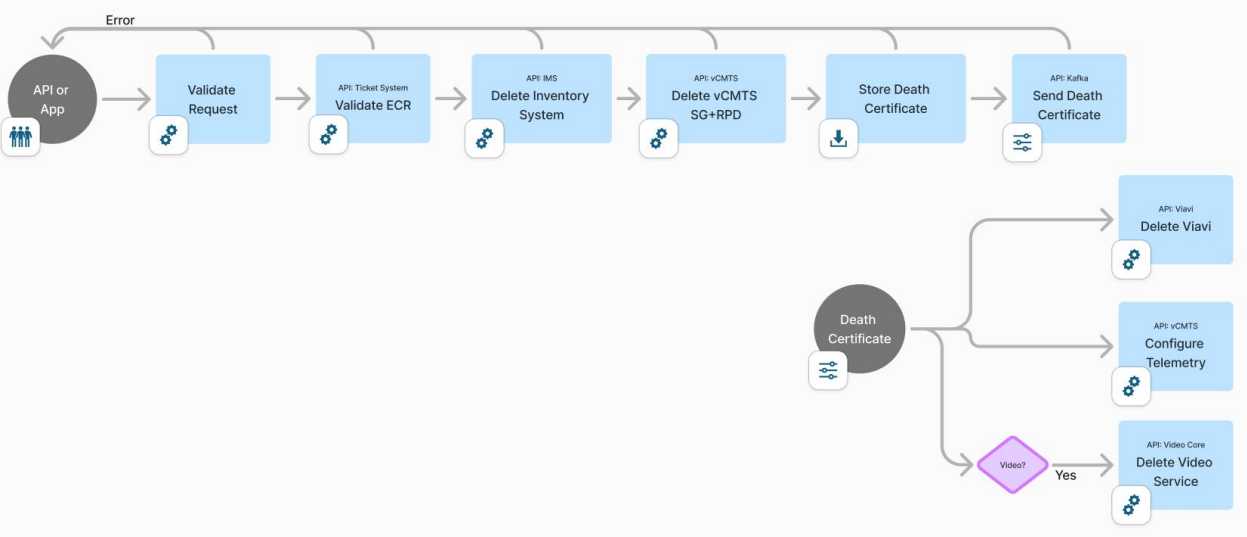

Figure 7 - RPD Removal Pipeline

5.3. Automation Based Configuration Change

Making configuration changes to running systems can be supported through the automation system (this section) or through an Out-of-Band Change (next section). Moving the organization to this automation workflow is incremental and an ongoing improvement process.

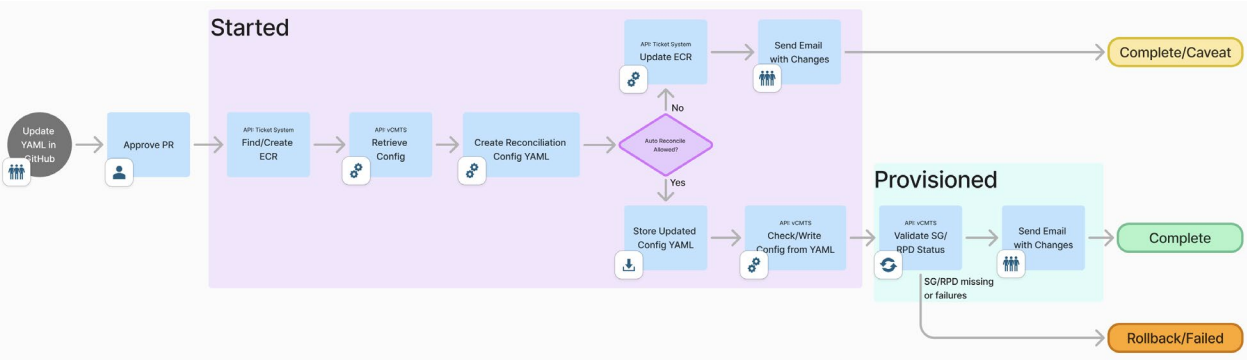

Figure 8 - Automated Change Pipeline

As discussed in the earlier sections, this workflow allows for incrementally moving toward automation. If the vCMTS isn't configured to allow for auto-reconciliation the configuration changes are, instead, emailed to the team to action through manual processes rather than automatically being applied. This allows the organization to become comfortable with the automation system and gain trust in the automatically generated CMTS configuration changes.

5.4. Out-of-Band Change / vCMTS Reconciliation

For some Operators, it's important to support co-existence of CLI-based workflows, Method of Procedures, and break-fix cycles. This workflow allows for changes to occur on the vCMTS, which are reconciled back into the automated workflow. It wasn't feasible to immediately switch to full automation

based provisioning and deny all "write access" directly on the vCMTS, so this workflow was developed to allow the organization to transition toward full automation over time.

A pre-requisite for this workflow is that the vCMTS supports a commit log event stream, such that each commit is sent to the automation system.

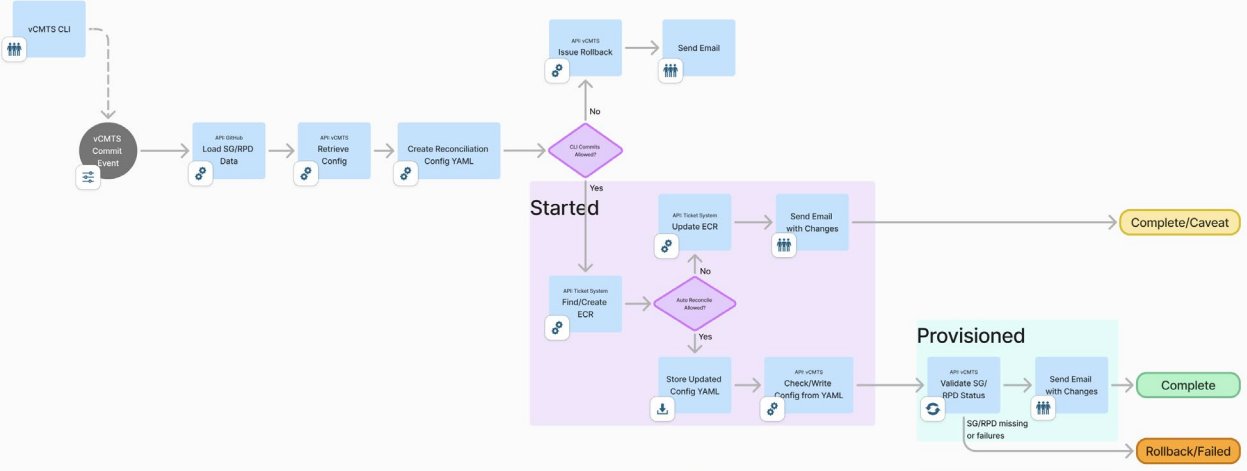

Figure 9 - OOB Change Pipeline

To keep the organization informed, no matter how the system decides to handle (or not handle) the out-of-band change an email is generated to key internal groups. This email may generate additional manual or out-of-band actions. Another interesting aspect, which needs to be communicated to internal stakeholders, is the automatic rollback. The automation system can have "locked down" vCMTS instances where no deviation is allowed, and any configuration change will cause the system to rollback. This might be the ideal end-state for an automation system, but can cause friction if engineers are trying to commit changes only to see the automation system rollback the changes.

5.5. Benefits and Challenges to Adoption

Automating the RPHY device lifecycle has had many inherent benefits, both direct and indirect. The two key benefits that are important to highlight are 1) Reducing the time required to provision and enable a RPHY device on the network and 2) preservation of data integrity by reducing the number of touch points where manual human input is required.

Enabling a new RPHY device on the network entails the interaction of many systems orchestrating the data dance between them. Some of the actions as part of this journey are resource reservations, physical installation of the device, updating inventory systems, notifying OSS systems that a network element is enabled and ready for service enablement. Having automation ensures that all these systems are updated with the correct sequence and allows the network technician to focus on the physical network enablement in the field while automation takes care of all the backend system provisioning including fallout logic.

Another key benefit of Automation is that it allows a Service Provider to declare the state of the network upfront and has key resources reserved as soon as the intent to deploy is pushed to the Automation system. This model ensures that the object data is consistent along all systems and reduces the number of touchpoints where users must input actual data versus validating the data the system is presenting at each stage of the lifecycle.

Two of the challenges in adopting an automation first strategy are 1) Organizational and Process Challenges and 2) Exception handling. One of the biggest challenges to automation in general is changing the organization culture and processes. These traditional organizations are structured in silo like artifacts with very rigid operational and finance models. To build efficient automation models, the design is focused on the business outcome and does not have contexts around the organizational rigidity or have awareness of north-south processes. This inertia naturally causes many automation models to fail or not be instantiated as they don't conform to the organizational construct everyone is comfortable with. One needs a strong sponsor and advocate to keep everyone focused on the actual benefit for the automation and be willing to partner with their peers on enabling new operational models.

Another challenge when deploying RPHY Automation is how to handle exception handling. As one designs automation constructs to be consumed by users, there is a tight rope to walk in how much ability is given to a user to override a system presented option or data. In the ideal world we would never need overrides or exception handling when things fail, but history has taught us never to put yourself in a box. In the Service Provider world, where we are still dealing with back-end data systems and complex relationship of service addressability, we always should have knobs and options to allow the operator to change default behavior choices with variable options but also build and provide tools to senior layers of technology support to override the automation system to "unglue" things when they become stuck with their own logic flaws. In addition to building these tools to override the system, the automation construct MUST have a process to capture transactions with exceptions. This is a key consideration in automation, never leave any transaction behind. Each one of these must have a relief valve routed to a human in a programmatic way to ensure network disruptions are identified clearly and not slip through the proverbial cracks.

6. Conclusion

Automated configuration management of Distributed Access Architecture (DAA) networks offers a robust solution to many of the challenges faced by Operators today. By reducing operational expenses, increasing agility, standardizing operations, improving reliability, and enhancing security, automation empowers organizations to respond more effectively to customer needs and market demands. A structured implementation plan, combined with clear communication and stakeholder engagement, is essential for a successful transition to automated configuration management.

Automation of the Remote PHY (RPHY) device lifecycle, in particular, provides direct and indirect benefits such as significantly reducing the time required to provision and enable an RPHY device on the network and preserving data integrity by reducing the number of manual touchpoints. Enabling a new RPHY device involves interactions among multiple systems, including resource reservations, physical device installation, inventory system updates, and notifications to Operational Support Systems (OSS) that a network element is ready for service. Automation ensures that these tasks are performed in the correct sequence, allowing network technicians to focus on physical enablement while backend system provisioning is handled automatically.

Using CI/CD pipelines within the framework of automated configuration management facilitates efficient and consistent deployment processes. CI/CD enables continuous integration and continuous delivery, ensuring that all configuration changes are automatically tested, validated, and deployed. By visualizing these pipelines, stakeholders can understand and align around a shared workflow, enhancing communication and transparency. This approach ensures that configurations are applied consistently across all environments, reducing errors and improving system reliability.

The GitOps methodology enhances the automation process by using Git as the single source of truth for configuration management. This approach leverages Git workflows for managing infrastructure as code, ensuring consistency and version control. GitOps facilitates continuous monitoring and automated reconciliation of the actual state with the desired state, enhancing reliability and security. When combined with tools like GitHub Actions, organizations can automate workflows directly within their repositories, using self-hosted runners to leverage local network resources and customized configurations.

Implementing NETCONF as part of the automated configuration management strategy provides additional benefits, such as transaction-based operations, secure transport, and standardized data modeling. NETCONF enables efficient and secure management of network device configurations, although it requires careful handling due to its complexity.

Organizations must also address challenges such as organizational resistance and the need for effective exception handling mechanisms. Cultural shifts and process changes are often required to fully realize the benefits of automation. Strong sponsorship and advocacy, coupled with clear communication and stakeholder engagement, are crucial for overcoming these challenges and achieving successful automation.

In conclusion, adopting automated configuration management, CI/CD, GitOps, and protocols like NETCONF can significantly enhance the efficiency, reliability, and security of DAA networks. By following a structured implementation plan and engaging all relevant stakeholders, organizations can successfully transition to automated systems that better meet customer needs and market demands. This paper has outlined a specific approach to automating DAA deployments within a framework of desired outcomes, offering a starting point for organizations to tailor these strategies to their unique requirements.

Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
CD	Continuous Deployment
CI	Continuous Integration
CLI	Command Line Interface
CSV	Comma Separated Values
DAA	Distributed Access Architecture
DevOps	Development and Operations
DevSecOps	Development, Security, and Operations
DORA	DevOps Research and Assessment
ECR	Engineering Change Request
FSM	Finite-State Machine
GitOps	Git and/with Operations
gRPC	gRPC Remote Procedure Calls
IaC	Infrastructure as Code
JSON	JavaScript Object Notation
ML	Machine Learning
NETCONF	Network Configuration Protocol
NIST	National Institute of Standards and Technology
NOC	Network Operations Center
OOB	Out-Of-Band
OSP	Outside Plant
OSS	Operational Support Systems
PR	Pull-Request
QR Code	Quick-Response Code
REST	Representational State Transfer
RESTCONF	Representational State Transfer Configuration Protocol
R-PHY	Remote PHY
RPD	Remote PHY Device
SDLC	Software Development Lifecycle
SG	Service Group
SSH	Secure Shell
TLS	Transport Layer Security
Txid	Transaction ID
vCMTS	Virtual Cable Modem Termination System
XML	Extensible Markup Language
YAML	Yet Another Markup Language