SCTE·ISBE CABLE-TEC EXPO 2018 ATLANTA, GA OCTOBER 22-25

2018 Fall Technical Forum SCTE·ISBE • NCTA • CABLELABS

Orchestration: What Is Really Behind This Overloaded And Overused Term?

An Overview of What Makes An Automation And Orchestration System

A Technical Paper prepared for SCTE•ISBE by

Alon Bernstein
Distinguished Engineer
Cisco Systems

Table of Contents

List of Figures

Introduction

Automation ! Orchestration ! These two magical words solve everything… with a click of a button OPEX shrinks to nothing. But not so fast…the software that drives automation and orchestration behind the scene is complex and highly customized and it cannot eliminate the inherent complexity of a service provider network, the data center and the outside plant. It can only help manage this complexity. It can't eliminate existing processes, but it can turn them from manually operated ones to a software operated processes.

Furthermore, the transition to the cloud native world, which is even more distributed and large scale (hence complex) then existing systems, makes automation and orchestration become more then OPEX reduction tools. It's impossible to operationalize these highly distributed systems without automation and orchestration.

This paper will attempt to separate fact from fiction when it comes to automation and orchestration. It will outline the steps required to go from a swivel chair process to an automated one and along the way explain the key concepts and layering required for automation.

Automation vs. Orchestration

The terms "automation" and "orchestration" tend to be bundled together, to the point that they seem interchangeable. The distinction this paper offers is as follows:

- Automation is the overall framework for replacing manual processes with software.
- Orchestration is the specific task of coordinating activities across several domains (we discuss the term "domain" later in this paper).

Where Do I Start?

In the following sections we'll account for the processes that need to be automated. That would help focus the discussion on the scope of automation.
Let's pick a very simple example and follow it through in order to help map high-level abstractions onto concrete actions. Say we want to ping all the cable modems in a service group to validate that they respond within a well-defined time range. Most likely there are already vertical applications that can do it, this paper would explore how you would build such an application with an orchestration/automation mindset.
A good place to start from is ONAP (Open Network Automation Platform, see ref 1). Even if you don't plan to use ONAP it's still a good reference model for our discussion.
ONAP itself is a large framework, and this paper will not discuss it. The only part we explore in this paper is the ONAP automated life-cycle described in Figure 1.

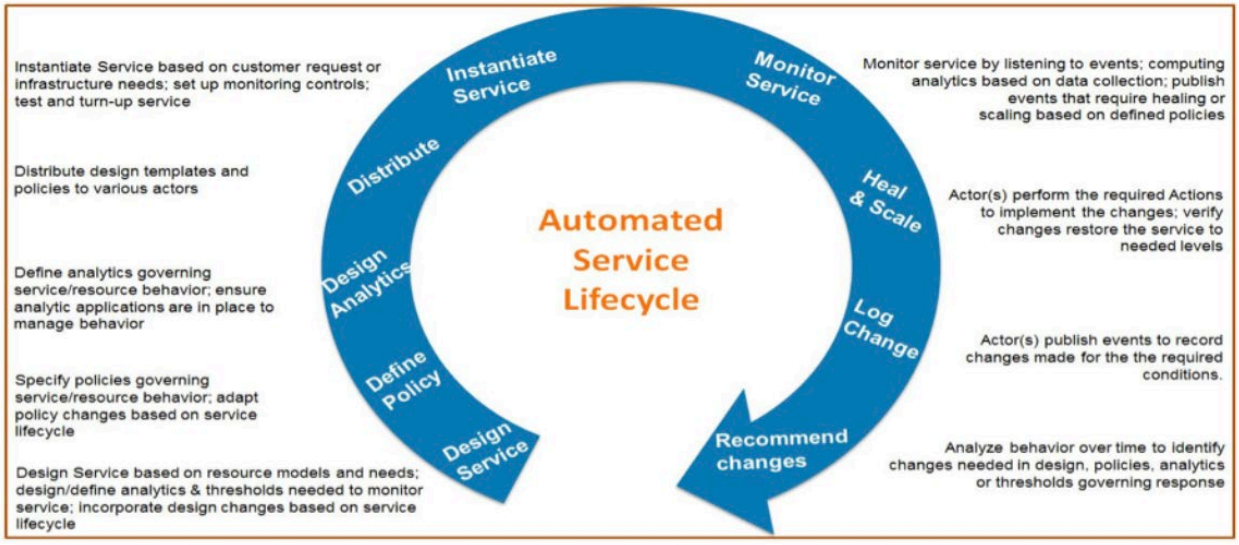

Figure 1 - ONAP closed loop automation

Let's go over the various phases here and in one line describe the key actions and requirements that have to be answered. One confusing thing about Figure 1 is that the service itself (ping) has an automated lifecycle of its own. For example, the ping service requires memory and there needs to be a policy what to do if memory allocation for the service fails. However, the focus of the study here is the user-facing service to be implemented:

1. Design service: define the scale, goal, key modules.
2. Define policy: what to do if a ping fail ? think of the actual pass/fail criteria for the ping, e.g. an average ping time greater then 20ms is a fail.
3. Define analytics: a collection of min/max/average of the ping times to modems could qualify as "analytics data" for our simple example
4. Distribute design template and policies to various actors: we can choose the network control center as the actor for are use case. More complex use cases may have multiple actors, each one with its own set of credentials, authorizations and capabilities.
5. Monitoring: that's when we actually activate the service and start monitoring the ping times
6. Based on the results of the ping take corrective action. This can trigger a whole set of other services, e.g. changing the modulation profile for a cable modem could help ping issue but it's a whole separate service.
7. Log the changes
8. Based on the analytics and observations make a long-term proposition on how to improve service.

Note that there is no orchestration or automation mentioned in each of the steps. The whole cycle is the automation of the service. The intelligence part of the automation is mostly under the analytics and recommended change parts but can be spread out to any component.

It's useful to compare this service life-cycle to the classic devops (development and operations) cycle diagram (see Figure 2):

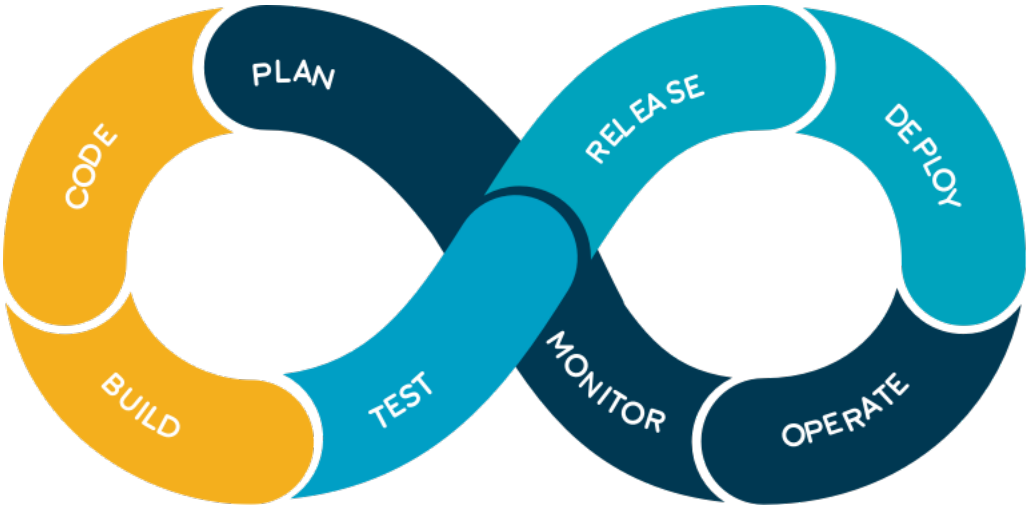

Figure 2 - Devops

As you can see the service life cycle can overlap the service life cycle, where the left-hand side of the figure (the "dev") correspond to the initial stages of the service life cycle and the right hand side of the service (the "ops") maps into the later stages of the service lifecycle .

Customer facing and Provider facing

Some of the orchestration and automation services are internal to the service provider and some are for consumers. For example, the process of installing an RPD (Remote Phy Device) is an operator facing service while providing an L2VPN service to a business customer is naturally a customer facing service. In principal the same tools can be used to automate either the provider facing or customer facing services, however there are several key differences:

1. Provider facing services are focused on what is known as "day zero" and "day one" operations where day zero is the initial install of the equipment (in the virtualization case the instantiation of the infrastructure and based containers). Day one is typically the base configuration needed to get the system up. The provider facing services are all CAPX and OPEX because they are concerned with bringing up the service provider "platform".
2. Customer facing are those that generate revenue for the service provider. These are the Day two type of operation. Note that there are operations that are not configured directly on network devices but rather are signaled e.g. voice calls.

Management Layering

A model that has been trusted and deployed for over 20 years is the TMN (ref [4]):

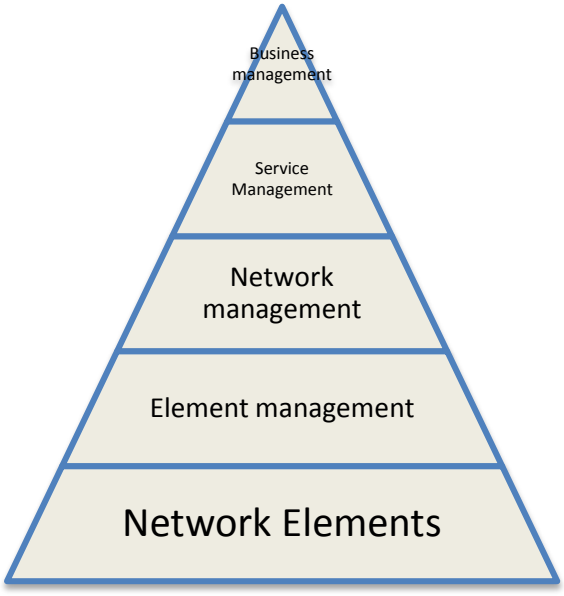

Figure 3 - TMN layering

Although this is an old model (the words "ATM" and "ISDN" sprinkled as examples throughput the recommendation hint at its age) the key concepts survived the test of time:

1. Network elements – the actual network devices. The big change since the 90's is that we have a new generation of virtual/cloud-native network elements, but functionally they are still "network elements".
2. Element management – the scope of managing a single element, for example, software update or configuration change for a specific device. In the age of SDN the concept of "controller" has been introduced and a good part of the device specific management is done by a controller. Note that even if a network element is cloud-native, and even if we use JSON for configuration, it still needs to be managed and this fits into the TMN model nicely.
3. Network management – this layer manages connectivity across a whole network comprising of various devices. This is the layer where a lot of orchestration and automation are required
4. Service management – the elements, element management and network management are all "provider facing" the actual services that the operator sells, for example, high-speed data for cable customers, are at this level. Note that this level extends beyond the network because it includes items such as credit card processing to make sure the customer paid for the service.
5. Business management: the layers on top of the services can include information that is required to run the business, for example, tracking customer satisfaction.

Domains

A complex network can be built out of several domains. For example, a cable network is comprised of the HFC plant, The DOCSIS protocols, access network, core network, back-office etc. To create services, one has to work across all these domains.

What makes the domains complex is that each one of them has its own set of tools and set of experts. The skills and tools used to manage the HFC are very different than those used to manage a data center. As a result, there is little to no sharing of expertise, and a swivel chair process where one organization (aka "domain") passes tickets to another to perform an action. The ticket typically sits in a queue and the result is that service activation can take days if not weeks.

It's the orchestration part of automation that takes care of operating end-to-end solutions across domain and does the magic of building consistency across a wire array of tools and expertise that each domain has. This is where workflow automation comes into place.

Workflows

A workflow is similar to a flow chart. It lists a set of actions and their dependencies. A standard called BPNM2.0 (Business Process Model and Notation, see ref [3]) was defined to capture workflows in a uniform way. A simple workflow for the example on cable modem scanning can look as the following:

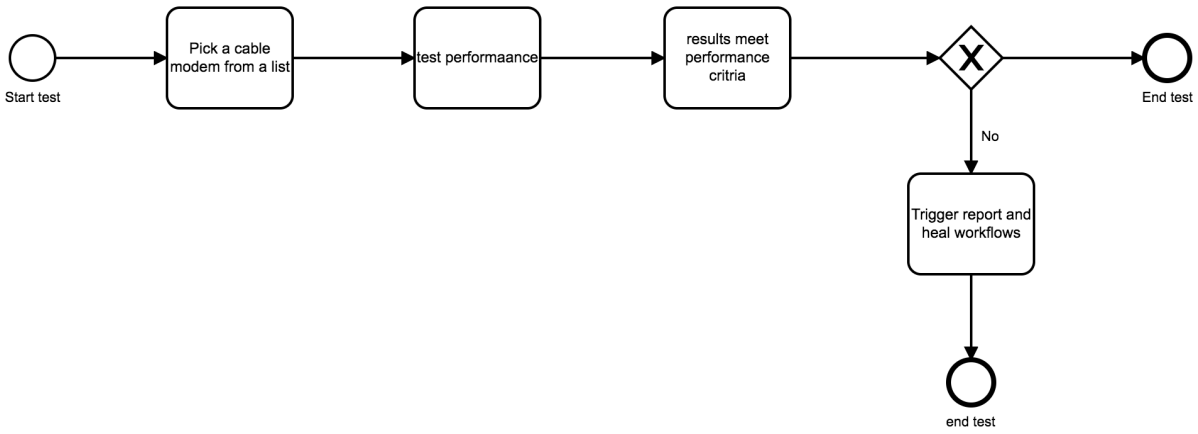

Figure 4 - Cable Modem scan workflow

Even with this simple workflow a key point can be highlighted: The workflow itself is not intelligent. It is essentially a state machine that calls APIs (Application Programming Interface) and takes actions based on the results of these API calls. The actual performance test could be a simple ping or some more complex operation. The workflow just calls the module that performs it. In other words, the workflow defines what needs to be done, not how to do it.

Another way in which workflows are useful is clarifying the use of APIs. In many cases publishing an API list is not sufficient, and a workflow example clarifies how to use the APIs correctly, e.g in what order they should be called.

There are several tools that can take a pictorial representation of a workflow and trigger actual API calls and scripts with it, so that the BPNM2.0 is not only a modelling tool, it can be an actual run-time service.

One question that comes up is the difference between writing a script vs. using a workflow engine. At the end of the day both can get the job done and it's about using the right tool for the right job. Having said that, if the answer to the questions below is positive then a workflow may be a preferred approach:

1. Do you need to communicate the process to non-software-engineers ? If yes then the workflow is your friend because it does not require coding skills. In a devops kind of environment it's a way to bring developers and operation experts closer together. Note that in a pure devops world the operators should be coders as well, but having said that in cable we have physical devices and outside plant components that are not made out of software….
2. Do you need to work across several domains ? If so the workflow provides a neutral environment that is not tied to a specific domain and focuses on API calling only.
3. Do you need an operation at the "what needs to be done" as opposed to "how to do it". The workflow excels at the former.

How does workflows relate to orchestration ? It is the view of this paper that the workflows are a way to implement orchestration, so they are one and the same. Note that there is no magic here. Orchestrion is accomplished only looking at existing operational practices, describing them in workflows and starting the process of replacing each one of the boxes that has a manual operation with a software based one.

Declarative vs. Imperative

There are two ways to achieve a network management goal:

1. Declarative: Detail the end-result, or end-state, and have the system automatically do what is needed to achieve the end goal/state. This method is referred to as "declarative" as the user declares the desired goal/state. Some very successful software solutions are declarative in nature. For example, in Kubernetes a user defines the scaling requirements (e.g how many replicas of a service are needed) and Kubernetes takes care of the actual scaling and placement of the service onto the CPUs in the data center in order to achieve the declared scale figure.
2. Imperative: Detail step by step how to reach a certain goal/state. This is more aligned with the work flow approach and is called "imperative".

The declarative model seems more attractive – what could be better than simply declaring what you want and let the system magically figure it all out ? There are two key observations about this in practice. The first one is that any service definition is in essence "declarative" because its states what needs to be done, not how to do it. For example, "I want a 20mbps upstream and 20

mbps downstream high-speed data service" is as good as declaration as any, nothing really new here. The second observation is that somehow, somewhere, someone needs to write the software to turn the declaration into reality. In other words, a script or program or workflow has to be written anyway, it's only a question of having a layer of "declarative" abstraction above it (which is indeed a useful thing to have), but nothing happens by magic.

Never repeat the same error

One of the key benefits of automation is repeatability and consistency. With manual processes there is always the risk that errors are the result of an operator mistake. Even in cases where failures occur intermittently a consistent a reparable process make the debug process and root cause analysis easier.

Automation is not static. Once an error occurs in production, and even once the error is fixed by means of software or configuration change, it's possible to write a script that validates the error does not happen again. In that sense the automation resembles a set of unit-tests in a software development process and fits into the devops model.

As the list of verification scripts gets longer it becomes a risk-reward question of how many of them needs to be run, and a choice one can made when submitting a change to a "bakeoff" setup vs. in production deployment as to how many validation scripts need to be executed.

Bill of Materials

With automation any change can be treated in a similar way, whether it's a software change or a configuration change, because any type of change carries a risk and can cause unexpected behavior. A "Bill Of Materials" (BOM) for a change can include the software modules that are changed and/or the configuration change, along with various validation scripts.

Introduce failures

Service provider networks are built for high-availability, but when are these high- availability systems tested in production ? The answer is that they should not be tested only when an unplanned failure occurs. Errors can be injected on propose. This concept has been popularized by Netflix in what's called "chaos monkeys" (see ref[5]), and can be considered part of the service automation loop.

How many automation use cases?

At this point we get to the hype around orchestration and automation. We can apply the automation lifecycle to both customer facing and provider facing services, and in each one of the layers of the 5 layers of the TMN model and across several domains (let's assume 5 domains for a cable network). That's 2x5x5 = 50 different areas where orchestration automation can apply and each can have the 9 stages of life cycle management for a total of 50x9=450 cases , and of course each one of them can have 10s if not more of use cases. It becomes clear that there is no

"finger snapping" that will make it happen, and if implementation resources are limited it's a question of figuring out which processes are the ones that benefit automation the most.
What automation represents more than reduction in human resources is a shift to devops. The brains to manage the network are still needed, but they invest their brainpower in creating scripts and workflows rather than repeating the same action again and again.
The number of automation workflows and scripts that have to be written also enforce the devops vision that everyone, dev or ops, need to be able to code, there is just too much coding work to be done !

So how did the industry survive without automation and orchestration?

Very simple. There was automation and orchestration all along, but with 3 caveats:

1. It was not called "automation/orchestration" it was called network management.
2. It was applied only when absolutely necessary. E.g. from the get go it was clear that deploying cable modems will not scale without automation and as early as the DOCSIS 1.0 spec the groundwork for automating the cable modem registration was put in place.
3. It was done in verticals: let's say an operator needed to keep track of some condition in the cable plant. In most cases a vertical application with its own set of collectors, data analysis tools, GUI, etc. was built with little sharing of other applications. The cable modem registration mentioned above is another example of a one-off mission specific vertical. Today we might have treated the cable modem registration as part of the IOT framework and use a whole different automation framework which would have been more consistent with other "things" that have to be managed.

What the new automation and orchestration advocates relative to the old way of doing network management is:

1. Build the orchestration and automation as applications on top of a common platform (e.g. ONAP).
2. Use open models such as YANG.
3. Use open source tools and code as a "standardization" framework as opposed to standard bodies paperwork.
4. Automate and orchestrate everything. Anything that has to be done more than once needs to be automated. Anything that requires coordination across multiple entities requires orchestration.

An Opportunity

The next wave of change in the SP networking space is cloud-native based networking devices. In the cloud native world functions are broken into micro-services and there can be tens, hundreds and thousands of those micro-services. At this scale automation and orchestration is no longer a nice-to-have-OPEX-reduction play. It's a must because it would be impossible to manage this level of complexity and distribution by hand. This presents the opportunity to start

with automation at the cloud native area and as tools and expertise are built, start branching out to other domains.

Conclusion

Orchestration and automation are not new. What is new is the use of open source tools, open model and workflows to support automation and the absolute necessity of having automation in the cloud world. Hopefully this paper helped in giving an overview of the key components and architectural concepts needed to create an automated network.

And last but not least, in the spirit of devops, every developer is an ops person and every ops person can write scripts and workflows. This is the key to automation.

Happy coding !!!

Abbreviations

API	Application Programming Interface
BOM	Bill Of Materials
BPNM	Business Process Model and Notation
Capex	Capital expense
DevOps	Development and Operations
ONAP	Open Network Architecture Platform
OPEX	Operating Expense
RPD	Remote Phy Device
TMN	Telecommunications Management Network
YANG	“Yet Aounter Next Generation” data Modelling langange

Bibliography & References

1. https://www.onap.org
2. ONAP closed loop Automation: https://onap.readthedocs.io/en/amsterdam/guides/onap-developer/architecture/onap-architecture.html
3. http://www.bpmn.org
4. TMN reference model : https://www.itu.int/rec/T-REC-M.3010-200002-I/en
5. Choas Monkeys : https://github.com/Netflix/SimianArmy/wiki/Chaos-Monkey

