Infrastructure LCM Automation

Introduction

In a typical telecom operator environment, infrastructure Life Cycle Management is highly complex and error-prone. The environment, with its multiple vendors and products, is maintenance expensive (both in terms of time and costs) because of the need for complex planning, testing, and the out-of-business-hours execution required to perform disruptive maintenance (e.g., upgrades) and to mitigate outages to mission-critical applications. Processes and tooling for infrastructure management across hybrid environments create additional complexity due to the different levels of access to infrastructure: hands-on access to the on-premise infrastructure but only restricted access to consumable services offered by public clouds.

Life cycle operations, such as software or hardware upgrades (including complex and risky firmware updates), typically involve time-consuming manual research and substantive testing to ensure that an upgrade is available, required, or needed, and does not conflict with the current versions of other components. In a complex and at-scale Hybrid Multi-Cloud environment, consisting of multiple on-premise and public clouds, such a manual process is ineffective and, in many cases, impossible to execute in a controlled manner. Hence, the need for automation.

The goals of LCM are to provide a reliable administration of a system from its provisioning, through its operational stage, to its final retirement.
Key functions of Infrastructure LCM:

- Hybrid, Multi-Cloud support, that is, LCM works across physical, virtual, and cloud environments, supporting on-premise, cloud, and distributed environments (like Edge)
- Complete system life cycle control (Plan/Design, Build, Provision, Operate/Manage, Retire, Recycle/Scrap)
- Enablement for automation of most maintenance tasks

Key benefits of the Infrastructure LCM Automation are:

- **Agility:** stadardisation of the LCM process by writing and running IaaC allows to quickly and easily develop, stage, and produce environments
- **Operational Consistency:** automation of lifecycle results in consistently maintaining desired state, reduces the possibility of errors and decreases the chances of incompatibility issues within the infrastructure
- **Human related Risks Mitigation:** automation reduces risks related to human errors, rogue activities, and safeguards the institutional knowledge from leakage in case any employee leaves the organization
- **Higher Efficiency:** achieved by minimizing human inaccuracies and eliminating the lack of knowledge about infrastructure installed base and its configuration, using the CI/CD techniques adapted to infrastructure
- **Cost/time Saving:** engineers save up on time and cost which can be wisely invested in performing higher-value jobs; additional cost savings on cloud more optimal use of cloud resources using LCM Automation

Infrastructure LCM Automation Framework

The following diagrams provide mapping between different stages of the lifecycle automation across all layers of the stack, to owners of infrastructure and cloud and the tenant as the consumer of the cloud services, in three very different scenarios: applications running as containers within virtual machines (CaaS on IaaS scenario), application running as containers on bare metal (CaaS on BM scenario) and a more traditional view of applications running as VNFs within virtual machines (IaaS scenario). The diagrams define the scope of the Infrastructure LCM Automation for each of these scenarios. The dotted lines symbolise the interactions between the layers of each of the model.

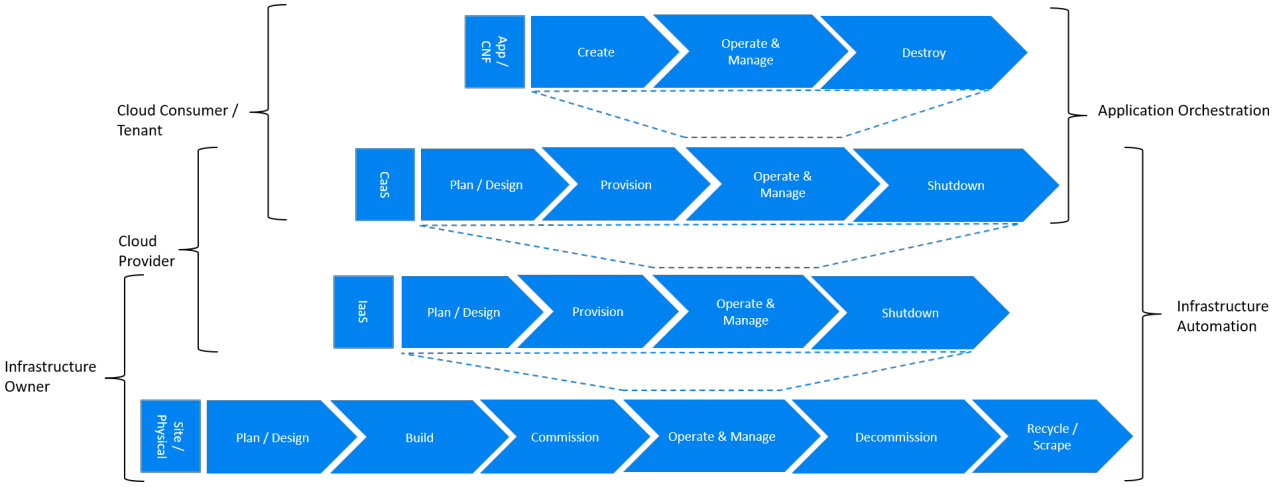

Fig 1. Infrastructure Automation in CaaS on IaaS scenario

In the CaaS on IaaS scenario, the Infrastructure Automation scope covers the Site/Physical layer, IaaS layer and CaaS layer. From the lifecycle perspective (the left hand side of the diagram), Site/Physical layer is entirely owned by the Infrastructure Owner, the virtualised infrastructure layer (IaaS) is shared between the Infrastructure Owner and the Cloud Provider. Similarly, the container orchestration layer (CaaS) is shared between the Cloud Provider and the Cloud Consumer / Tenant. These relationships can be illustrated by a situation, where a telecom operator owns the physical infrastructure on which an external cloud provider runs the virtualisation software (hypervisor). Sharing CaaS layer between the Cloud Provider and the Cloud Consumer reflects the fact that the container management/orchestration software like Kubernetes is lifecycled by the Cloud Provider (for instance when scaling out containers) but also by the Cloud Consumer because of the very close lifecycle relationship between an application and a container in this model. For instance, destroying an application means also destroying related containers, Hence CaaS can be also considered as a part of the Application Orchestration layer.

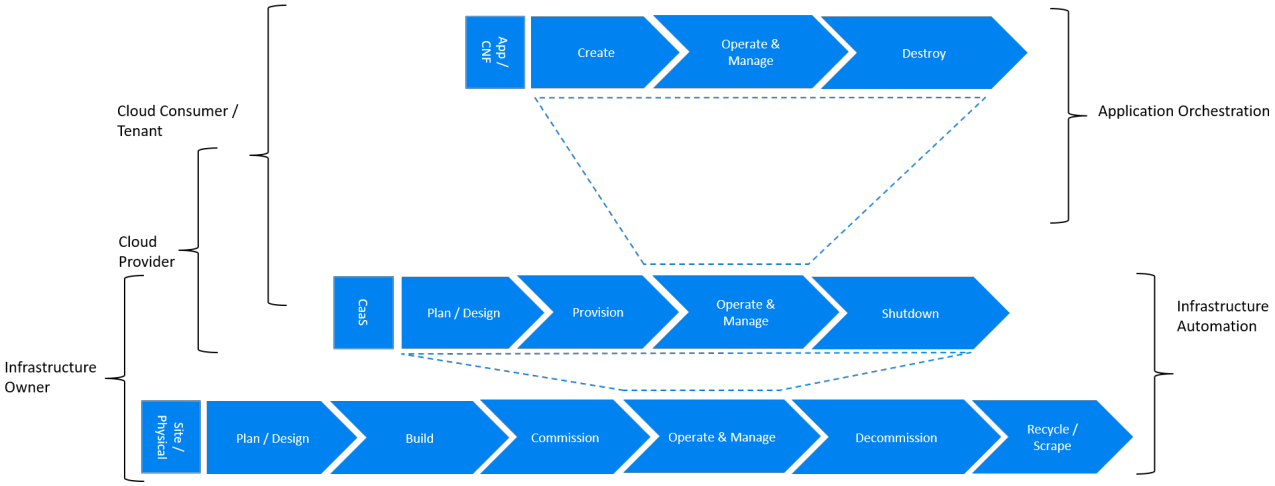

Fig 2. Infrastructure Automation in CaaS on BM scenario

The main and obvious difference in the Caas on BM scenario is lack of the IaaS layer, and hence the scope of the Infrastructure Automation is limited to only two layers: Site/Physical and CaaS. From the lifecycle ownership perspective, the CaaS layer is now shared not only between the Cloud Provider and the Cloud Consumer (for the same reasons as in the CaaS on IaaS scenario) but also with the Infrastructure Owner. The latter observation is related to the fact that in the bare metal deployments lacking the hypervisor separation, the CaaS layer is much more dependent on the underlying physical infrastructure.

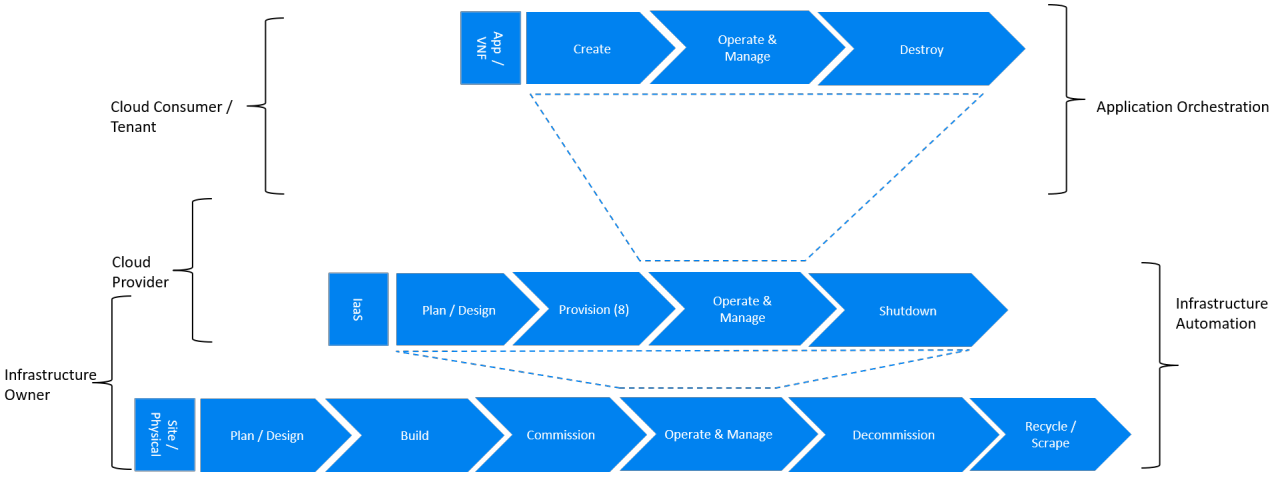

Fig 3. Infrastructure Automation in IaaS scenario

In this "classical" scenario the scope of the Infrastructure Automation is defined by the Site/Physical and IaaS layers. From the lifecycle perspective the ownership of IaaS is shared between the Infrastructure Owner and the Cloud Provider. This scenario is characterised by a clear separation between the lifecycle (and hence its automation) of infrastructure and the application lifecycle owned by the Cloud Consumer / Tenant in the role of the Application Owner.

Essential foundation functional blocks for Infrastructure LCM Automation:

- Representation Model

- Repository functions
- Available Software Versions and Dependencies
- Orchestration Engine

Automated LCM uses Representation Model to:

- abstract various automation technologies
- promote evolution from automation understood as automation of human tasks to autonomous systems using intent-based, declarative automation, supported by evolving AI/ML technologies

Automated LCM uses Repository functions to:

- store and manage configuration data
- store and manage metrics related data such as event data, alert data, and performance data
- maintain currency of data by the use of discovery of current versions of software modules
- track and account for all systems, assets, subscriptions (monitoring)provide an inventory of all virtual and physical assets
- provide a topological view of interconnected resources
- support network design function

Automated LCM uses available IAC Software Versions and Dependencies component to:

- store information about available software versions, software patches and dependency expectations
- determine the recommended version of a software item (such as firmware) and dependencies on other items in the node to ensure compliance and maintain the system integrity
- determine the recommended versions of foundation software running on the cluster

Automated LCM uses Orchestration Engine to:

- take the inputs from Repositories, Available Software Versions, and Dependencies
- run the software version changes
- dynamically remediate dependencies during the change process to optimise outcome
- ensure that the system is consistent across its life cycle by maintaining it in accordance with the intent templates

LCM Automation Principles / Best Practice

The following principles should guide best practice in the area of the Infrastructure LCM Automation:

- **Everything Codified:** use explicit coding to configure files not only for initial provisioning but also as a single source of truth for the whole infrastructure lifecycle, to ensure consistency with the intent configuration templates and to eliminate configuration drift
- **Version Controlled:** use stringent version control for the infrastructure code to allow proper lifecycle automation
- **Self-Documentation:** code itself represents the updated documentation of the infrastructure, to minimise the documentation maintenance burden and to ensure the documentation currency
- **Code Modularisation:** apply to IaaC principles of the microservices architecture where the modular units of code can be independently deployed and lifecycled in an automated fashion
- **Immutability:** IT infrastructure components are required to be replaced for each deployment during the system lifecycle to be consistent with immutable infrastructure to avoid configuration drift and to restrict the impact of undocumented changes in the stack
- **Automated Testing:** is the key for the error-free post-deployment lifecycle processes and to eliminate lengthy manual testing processes
- **Unified Automation:** use the same Infrastructure LCM Automation templates, toolsets and procedures across different environments such as Dev, Test, QA and Prod, to ensure consistency of the lifecycle results and to reduce operational costs
- **Security Automation:** security of infrastructure is critical for the overall security, dictating to use consistent automated security procedures for the threat detection, investigation and remediation through all infrastructure lifecyle stages and all environments