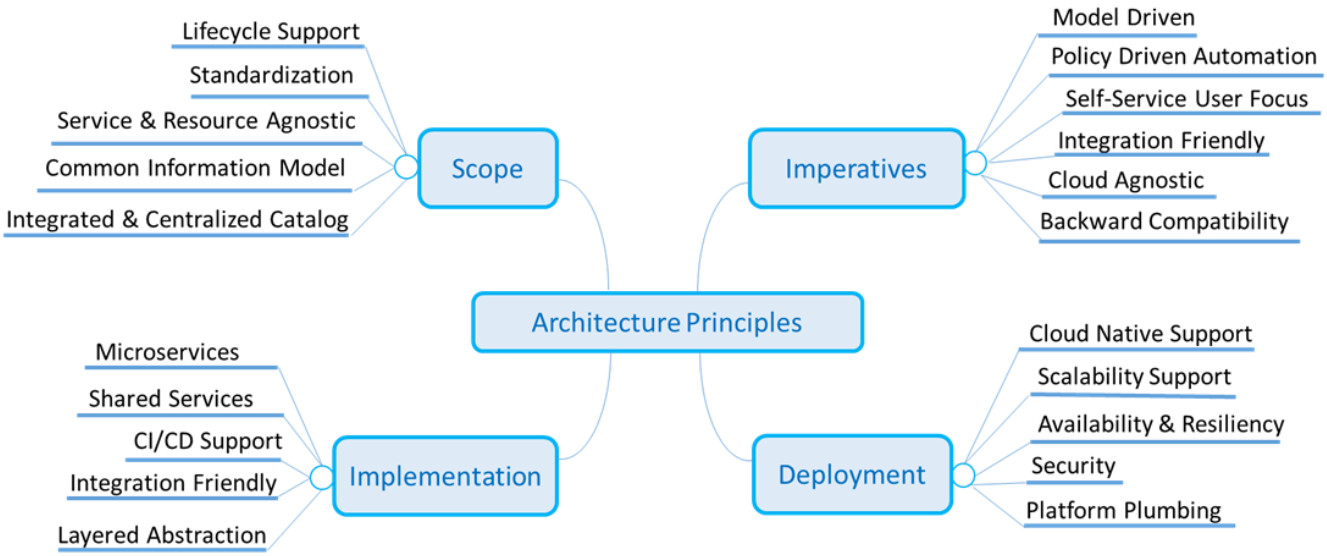
Architecture Principles

(from Architectural principles_v3.docx)

Architectural principles provide guidelines when making architecture decisions. The principles are *not* requirements (functional or non-functional), nor should they specify the design of the system.



1. **ONAP Scope:**

- [x] **Lifecycle Support**: ONAP platform must support a complete life cycle management of software-defined network functions / services: from VNF /PNF On-Boarding, Resources / Service Definition, VNF / PNF and Service Instantiation, Monitoring & Management, Change Management, Software Upgrade, to retirement
- [ ] **Strive for Standardization:** ONAP must support standardized common approach to manage various network functions from different vendors
    1. Standard templates for instantiations
    2. Standard language for configuration
    3. Standard telemetry for monitoring and management
- [ ] **Resources, Vendors, & Service Agnostic**: ONAP Platform must be PNF / VNF, Resources, and Service agnostic. Each service provider or integrator that uses ONAP can manage their specific environment (Resources, PNFs / VNFs, and services) by creating necessary meta-data / artifacts using Design Studio to support their needs / environment.
- [ ] **Common Information Model approach:** ONAP should define a standardized common information model for all vendors to follow. This will allow ONAP users to quickly onboard and support new PNF / VNFs and services.
- [ ] **Pluggable Modules**: The ONAP architecture should develop and promote PNF / VNF standards to allow delivery of Lego block-like pluggable modules, with standard interfaces for all aspects of lifecycle management (e.g. instantiation, configuration, telemetry collection, etc.) from various vendors.
- [ ] **Integrated & Centralized Catalog**: All meta-data / artifacts required by various ONAP components should be designed from a central ONAP design studio, cataloged and shared from central repository.

2. **Business Imperatives to Address:**

- **Model Driven**: All ONAP modules should be model-driven, avoiding, as much as possible, new programming code. This allows for a catalog-based reusable repository for resources, network & services lifecycle management.
- **Meta-data & Policy Driven Automation**: ONAP should support high levels of automation at every phase of lifecycle management – e.g. onboard, design, deployment, instantiation, upgrade, monitoring, management, to end of life cycle. These automations should be policy driven, allowing users to dynamically control automation behavior via policy changes.
- **Self-Service & User Focused**: ONAP Platform should support a self-service model with a fully integrated user-friendly design studio to design all facets of lifecycle management (resources / service design, operational automation, etc.). All interfaces and interactions with ONAP should be user friendly and easy to use.
- **Integration Friendly**: When an ONAP component relies on software outside of the ONAP project, the dependency on that external software should be designed to pluggable, API-oriented, supporting multiple possible implementations of that dependency.
    - The ONAP APIs are clearly documented and independent of the ONAP component implementation technologies (messages, procedures and contract).
    - The ONAP APIs are forward and backwards compatible.
- **Cloud Agnostic:** ONAP platform should be able to run and support multiple cloud environments (simultaneously), including container environments like Kubernetes and other Cloud Native technologies.
- **Backward Compatibility**: Every new ONAP platform release should support backward compatibility with at least one previous release.

3. ONAP Implementation Approach:

- [x] **Microservices**: ONAP modules should be designed as microservices: service-based with clear, concise function addressed by each service with loose coupling.
- [ ] **Shared Services**: Where applicable, reusable components can be provided as shared services across ONAP components.
- [x] **CI / CD Support**: ONAP is predicated on an accelerated lifecycle for network services. As such, agility is key in all aspects of ONAP: development of ONAP, designing of network services, and operation of both ONAP and network services. Principles of continuous integration and deployment should be followed by all modules of the ONAP platform.
- [x] **Standard APIs**: ONAP must support a rich set of external APIs (aligned with standard bodies such as MEF, TMF, etc., when possible) to easily integrate ONAP with external OSS / BSS as well as other management systems.
- [ ] **Layered Abstraction:** Define ONAP as a layered architecture similar to the OSI model for the internet. Define abstract interfaces between the different layers to support information and request flowing between the layers in an implementation-independent manner. Such a layered architecture provides flexibility to swap technology or replace an individual **ONAP** module, if desired.

4. ONAP Deployment / resiliency / scalability Support:

- [x] **Cloud Environment Support**: All components in ONAP should be virtualized, preferably with support for both virtual machines and containers. All components should be software-based with no requirement on a specific hardware platform.
- [x] **Scalability**: ONAP must be able to manage a small set of PNF / VNFs to highly distributed, very large network and service environment deployed across the globe. It should be possible to deploy multiple instances of ONAP and create a network of inter-working ONAP instances.
- [ ] **Availability & Resiliency**: ONAP must support various deployment and configuration options to meet varying availability and resiliency needs of various service providers.
- [x] **Security:** All ONAP components should keep security considerations at the fore-front of all architectural decisions. Security should be a pervasive underlying theme in all aspects of ONAP**.** The ONAP architecture should have a flexible security framework, allowing ONAP platform users to meet their security requirements.
- [ ] **Platform Plumbing:** Identifies areas of commonality and implements reusable solutions that can be used to support generic needs such as (a) resiliency and traffic control, (b) observability (e.g. logging), (c) security, and (d) data persistence, alleviating the burden of this on the module developers, and speeding up the process accordingly. This also helps optimize platform and component footprint.

Mind map diagram above maintained in ONAP Architecture Principles and Notes.xmind

See also Draft Architecture Principles