Why Nephio?
Nephio R1 Concepts and Tutorials
Episode 2
July 2023

Prerequisites:
- Episode 1 - Series Introduction

https://nephio.org/learn

John Belamaric, Sr Staff Software Engineer, Google
Nephio SIG Automation Chair
Kubernetes SIG Architecture Co-chair
Stepping Back

- **Why cloud?**
  - On-demand, API-driven consumption of data center resources

- **Why MEC / Distributed Cloud?**
  - On-demand, API-driven consumption of edge resources

- **Managing workloads on cloud is hard**
  - Many projects for this in areas like gitOps, App Delivery, Workflows, Platform Engineering
  - Not really a solved problem

- **Managing workloads on thousands - or tens of thousands - of little clouds is much, much harder**
Imagine deploying complex, interconnected workloads across many geographically distributed sites.
Simplified / Minimal 5G Network

- CU-CP
- UPF
- SMF
- AMF
- RU
- vDU

- MEC
- Telco Remote Edge
  - 100,000+ sites

- MEC
  - Telco Pre-agg
    - 1,000 sites

- MEC
  - Telco Agg
    - 100 sites

- Cloud
  - Public Cloud Region
    - 10s sites

- Observability & Analytics
  - Logging
  - Analytics
  - Function Monitoring
  - SLO Monitoring

- Service Policy Management
  - Config Mgmt
  - Cluster Mgmt
  - Service Mesh

- Telco Pre-aggregation
  - 1,000s sites

- Telco Aggregation
  - 100s sites

- Telco Aggregation
  - 100s sites
Planning, Planning, Planning

What does it take to roll this out? **Some** of it, for **Day 1**: 

- Identify available, applicable sites - edge and cloud regions
- Determine which workloads should run where and how they interconnect
- Determine the infrastructure needed - clusters, nodes, special hardware
- Configure cross-site networking: allocate subnets, IPs, VLANs, VRFs, etc.
- Configure the underlying nodes for specialized telco requirements
- Configure the workload specifications - their Kubernetes manifests
- Configure the workloads themselves to know about each other
Day 2 adds more complexity:

- Monitor that the stated intent is still expressed
  - Workloads are up and running
  - Configuration hasn’t drifted

- Handle changes to topology
  - Spin up a new aggregation site, adding a UPF
  - UPF needs to talk to an SMF
  - Each of these needs to be configured to see each other

- Resize workloads as topology changes
  - As we add UPFs, we need to vertically scale the SMF.

- Enable upgrade of workloads and infrastructure with progressive rollout
It Gets Worse…

Each layer is managed by different systems!

- **Topology**: Manually encoded in powerpoint slides and spreadsheets
  - Maybe end-to-end orchestration workflows
- **Cloud infrastructure**: Cloud Provider APIs
  - Maybe Terraform, scripts, or e2e orchestration
- **Networking within and between sites**: manual router configuration
  - Maybe some vendor or other proprietary automation
- **Nodes**: K8s extensions, manual or scripted kernel and other configs
- **Workload specifications**: stored in Kubernetes manifests, maybe in Git or scripts
- **Workloads configs**: proprietary, vendor-specific network element managers
And Worse…

Different systems means different teams.

- Each layer and system has a different team, probably even broken up by region
- Existing methods such as Helm charts assume you have already figured out all the inputs
- Teams must negotiate all these values ahead of time on a per-site, per-workload basis
- Imagine…
  - 100 inputs per workload
  - 20 workloads per site
  - 10,000 sites
- That is 20,000,000 values!
What do we do? Where do we start?

Reduce Complexity

- Consolidate on a **single, unified platform for automation**
  - Across infrastructure, workloads, workload configs, vendors and deployment tiers.

- **Declarative configuration with active reconciliation** to support days one and two.
  - And distribute state (intent) across geography for resilience

- Configuration that can be **cooperatively managed** by machines and humans.
  - Machine-manipulable configuration is fundamental to automation.