Who Needs Network Management in Cloud Native Environment?

Eshed Gal-Or @ Huawei
Pino de Candia @ Huawei
Outline

• Some background
• The App-centric view
• Propose actionable plan
Some background…

- Cloud started as IaaS
  - Consume datacenter through API (instead of IT requisition forms)
  - Old concepts applied (you needed to know what you were asking for)
  - Network and security were “imported”
  - But it didn’t go very well…
Why Network & Security “import” failed

• Multitenancy
  – Network devices (incl. security services, e.g. firewall) couldn’t discern between tenants

• Configuration
  – Network & security configuration was orthogonal to compute & storage, it didn’t keep up

• Performance : Cost ratio
  – Virtualization meant x10 more IP stacks on the same physical network and backplane capacity
Then came some improvements…

• Security groups
  – Decomposition of the “perimeter firewall” into per-VM/port allow rules (default drop)

• Overlay networks
  – Extra encapsulation (e.g. VXLAN) to provide disaggregation of L2 virtual networks

• Virtual Router
  – Isolation of L3 subnets
But they brought management chaos

- Synchronization of security policies between all servers, all VMs and all perimeter devices
  - Different vendors & capabilities
- Synchronization of physical network configuration with the virtual network
  - VLAN, port groups, cross-rack communication, VM-to-physical
Then came “Software-Defined”…

- **SDN**
  - Central software-defined configuration of all the network entities
  - Single topology mapping from virtual to physical
  - Northbound API for applications
  - Virtual switch & OpenFlow

- **Security Policy Orchestration**
  - One high-level policy model, disseminated to multi-backend, multi-vendor devices
And the user? What about the user?

- Users were consuming APIs
- But the APIs had a lot of “infrastructure” terms and requirements in them
- APIs were polluted with non-Application Programming related stuff
Devops to the rescue?

Devops = \( \sqrt{\text{Programmers} \times \text{Infrastructure}} \)

- Devops are Programmers that deal with infrastructure, so that Programmers that write applications won’t have to

- But isn’t that what Cloud and orchestration automation were supposed to do?
Rise of the Containers…

- Docker came up with a lean & simple app packaging process
  - Just your files, layered on top of known files (UnionFS and the Docker repo)
  - Each process runs inside a namespace
  - Don’t care about the Linux OS underneath
  - Containers launch in milliseconds
  - Great for cloud-native, share-nothing, stateless microservices
• But just packaging wasn’t enough… it didn’t deal with keeping the software running in production
• So came Kubernetes, Mesosphere, DockerSwarm, etc.

WE’LL LOAD-BALANCE, AUTO-SCALE, WATCHDOG, DISTRIBUTE AND GENERALLY BABYSIT YOUR APP FOR YOU
And the user(s)?

Now we had two of them:

**THE COE USER**
GETS A NICE, (MOSTLY) CLEAN AND COZY "CLOUD NATIVE" EXPERIENCE

**THE COE DEVOPS/ADMIN**
GETS LEFT WITH IAAS
Is this so bad?

- It is
- Cloud native took away the infrastructure pollution
- IaaS still requires all of it
- Cloud native users don’t care about it
- So why do Devops/Admins need to?
Dealing with defaults

• It’s all defaults anyways…
  – Since the app developers don’t care about IP addresses, networks, security groups, etc.

• So why isn’t it automated?
App-Centric, App-Driven
App-Centric, App-Driven

- Datacenter infrastructure needs to fulfill app needs
  - not force the app developer to engineer the infrastructure
- Network? Apps only need to connect their pieces
  - who cares about defining networks, subnets, routers, etc.?
- Security? Apps only need to connect the way the app designer decided (nothing else)
  - why do we need to care about security groups, firewalls, IDS, etc.?
So IaaS is bad?

- Not the right question
  - it is what it is

- IaaS is a laundry service and public clouds provide a laundry list API
  - Too many options, too much infrastructure-driven APIs
  - Too complicated to translate app needs to service configuration
So Kubernetes is Better?

- It is, for the app developer
- But it still depends on DevOps/Admin to handle the “app to infrastructure services” translation
  - Which VMs?
  - Set-up the cluster servers
  - Network configuration
  - Firewalls, load balancers
Better yet, Symbiotic Relationship

- App developers consume IaaS via the COE
- IaaS can focus on what apps really need, and reduce efforts on unnecessary services
What about K8s on Bare-metal?

- Becoming popular
- “Said” to have better performance
- But…
- Is it any different from legacy IT?
  - Who manages the bare metal?
  - How to do dynamic scale-out?
  - Who wires the network and sets up the underlying security?
Benefits of integrating COE and IaaS

• Defense in Depth
• Manage K8s node requirements and lifecycle
• Securely connect VMs and containers
• Securely connect to cloud services
• Cumulative resource quota across platforms
Example 1: Defense in Depth

- Layered security:
  - **Container-level**: Service mesh (e.g. Istio)
  - **Kubelet-level**: NetworkPolicy template
  - **Hypervisor-level**: OpenStack’s FWaaS and Security Groups
Example 1: Defense in Depth

- Today IaaS security is disabled
- IaaS should understand and adapt to container workload policy
Example 2: K8s node management

- Evolve resources along with app requirements
- Adapt node profiles as requirement change:
  - better ratios of CPU, memory, network
  - Specialized capabilities (GPUs, HSM)
- Automatically provision new nodes
- Trigger retirement of old nodes.
- Use IaaS properties to populate node failure domain tags.
- Test/validate/roll-out new node Operating System.
Example 3: Mixed workloads

- Today, VMs and containers can’t easily talk to each other.
- Mixed workloads get worse security than VM-only or Container-only
- IaaS and COE need security integration
How to get from here to there?

- One “system of intents” (or policy)
- COE enforces the policy on the app
- Who enforces the policy on the infrastructure?
  - Devops – Not sustainable
  - Devops automation (heat, tosca, chef/puppet, etc.) – Not adaptive, not scalable, too slow
What components do we need?
OpenStack Dragonflow

- Dragonflow is policy-driven, lightweight distributed SDN controller with pluggable DB
- Its network model is the OpenStack Neutron API
- It scales (built for 10K nodes, tested on 4K)
- Has an extensible pipeline (true SDN)
OpenStack Kuryr+Fuxi

• Kuryr is a K8s API Listener and CNI implementation
• It Enables K8s to consume OpenStack network services
• Fuxi integrates containers with OpenStack block and file storage
Orchestrates Kubernetes on OpenStack.

But it doesn’t understand the COE’s workloads or their intent…

…while **Kuryr** does
Example 1: Defense in Depth

Use Kuryr + Dragonflow to program hypervisor- or network-level security.

Kuryr infers/interprets container workload intent and programs node-level firewalls.
Example 2: K8s node management

Extend Kuryr to inform Magnum of K8s workload characteristics.

Extend Magnum to adjust the resources of new nodes it provisions.
Example 3: Mixed workloads

Dragonflow can connect and secure mixed VM/container workloads.

If each platform brings its own SDN, we could use Kuryr to sync security policy and endpoint membership (e.g. tag information) across COE and IaaS.
How to make it even better?

- Expand this approach to Storage
  - Raise your hand if you’ve recently used block storage directly
- …and Images
  - Raise your hand if you think 8GB are necessary to hold all your 200Kb of python code
  - Or if you think there’s anything unique about those 8GB
And then?

- Cloud Admin role – Will it still be necessary?
  - Hopefully not
- Remove Admin and root accounts on hypervisor hosts
  - Better security
  - More stability
  - Lower cost
  - Faster deployment