KubeCon | CloudNativeCon

North America 2019

KubeCon | CloudNativeCon

North America 2019

Kubernetes Multitenancy Working Group – Deep Dive

Sanjeev Rampal
Cisco Systems

Adrian Ludwin
Google

Where to find us

- Home page: https://github.com/kubernetes-sigs/multi-tenancy/
- https://github.com/kubernetes/community/tree/master/wg-multitenancy
- Slack channel: Kubernetes Slack, #wg-multitenancy
- Google Group: https://groups.google.com/forum/#!forum/kubernetes-wg-multitenancy
- Bi-weekly meeting (join google group for invite)
  - Tuesday 11am Pacific Time

WG community

- Project leads
  - @Adrian Ludwin
    - Hierarchical Namespace Controller (“HNC,”)
    - Software Engineer @ Google
  - @Fei Guo
    - Virtual Clusters, Tenant Controller
    - Software Engineer @ Alibaba
  - @Jim Bugwadia
    - Multi-tenancy Benchmarks
    - Founder & CEO at Nirmata
- Chairs
  - @tasha
  - Tasha Drew, Product Line Manager @ VMware
  - @srampal
  - Sanjeev Rampal, Principal Engineer @ Cisco

- Additional Project contributors
  - Ryan Bezdicek
    - Support and review across projects
  - Many many more contributors across the Working Group – Thank you!!

Agenda

- Overview and Architecture
  - What is Kubernetes Multitenancy ?
  - Architectural models for Multitenancy
- Community initiatives: Multitenancy control plane
  - Tenant controller & namespace grouping
  - Hierarchical namespaces
  - Virtual clusters
- Community initiatives: Data plane and benchmarking
  - Benchmarking
  - Data plane models
- Demo
- Q & A

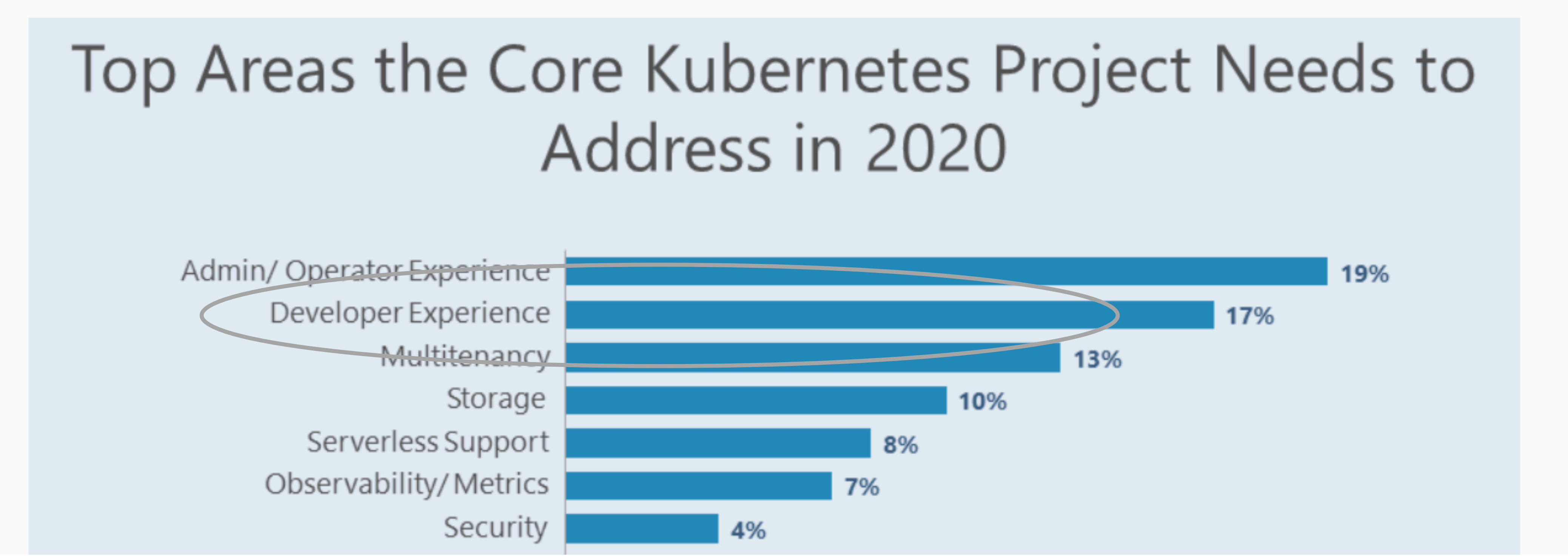
Overview & Architecture

What is Kubernetes Multitenancy ?

- What is it ?
  - Ability to share a Kubernetes cluster between multiple independent teams
- Why is it useful ?
  - Improved resource efficiencies (esp when move to containers on BM)
  - Reduced cluster sprawl
  - Lower capex and opex for the cluster operator
  - Resource usage burstability -> Higher application performance
  - Essentially a bin-packing & statistical multiplexing problem
- Potential challenges
  - Kubernetes not designed for Multitenancy at its core
    - Unlike say Openstack, there are no core K8s resources for ”Users”, “Tenants”, “Projects”
  - Wide spectrum of loosely defined scenarios and potential use case
    - Defining “Standardization” vs best practice vs implementation choice

The community feels this area needs work

KubeCon | CloudNativeCon North America 2019



- The New Stack poll (*newstack.io November 2019*)

What is Kubernetes Multitenancy ? ...

- Categories of Multitenancy (high level use cases)

- “Soft” Multitenancy
  - Ex. Multiple teams within the same enterprise sharing a K8S cluster

- “Hard” Multitenancy
  - Ex. Service provider hosting multiple independent tenants on a shared cluster
  - “Coke & Pepsi on the same K8s cluster”

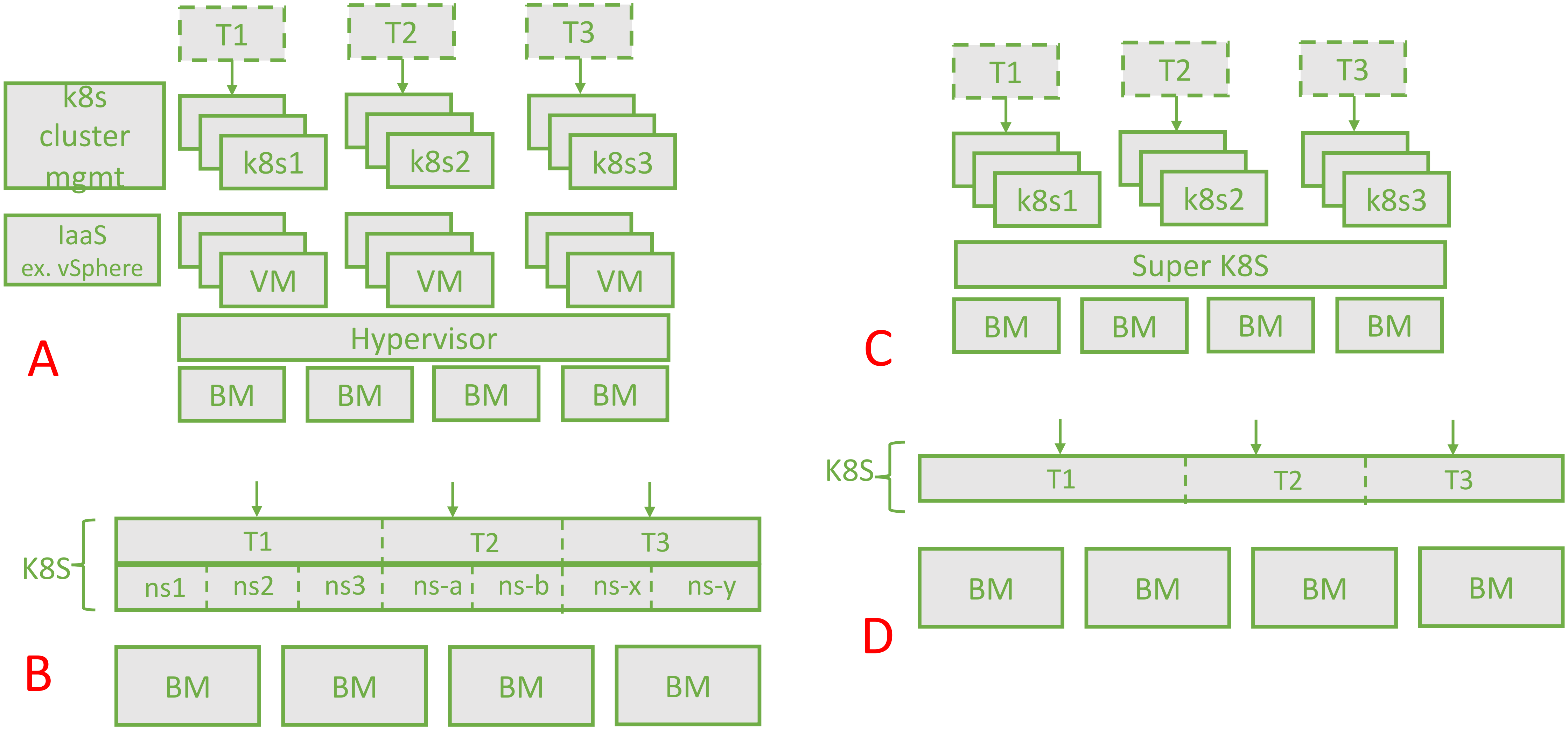
- Other
  - SaaS multitenancy

What is Kubernetes Multitenancy ? …

- Available solutions

1. Community Kubernetes + DIY solution using namespaces, network policies etc
2. Vendor/ commercial distributions with features built on these
    - E.g. Openshift “Projects”, Rancher “Projects”
3. Emerging community initiatives tracked within K8s Multitenancy Working group & others

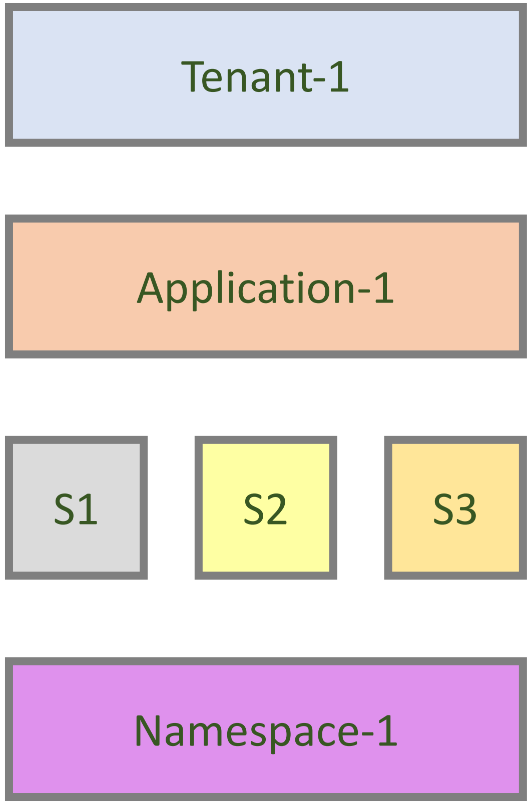
Architectural Models



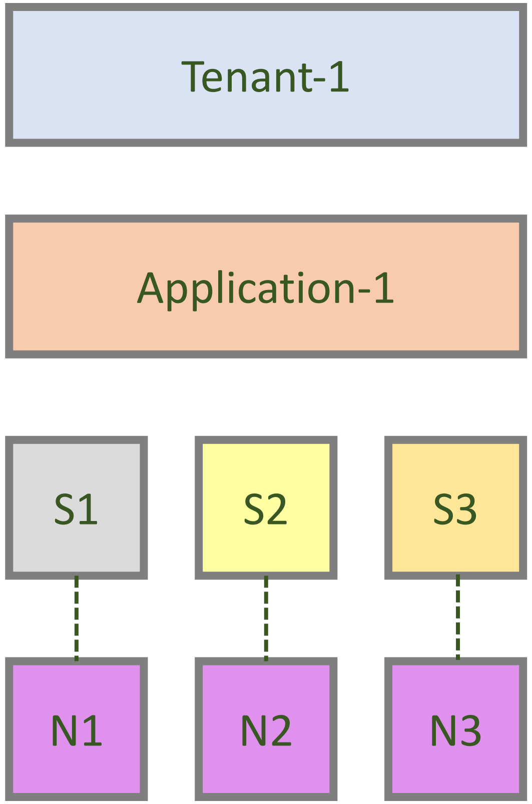
Architecture Options

Multitenancy Architecture Model	Resource efficiency	Level of Tenant isolation	Tenant/ application Config restrictions	All "Cloud Native" architecture	Architecture maturity & production readiness
A: Multiple K8S clusters on top of a Virtualization IaaS	Low-medium	High	No	No (multiple separate platforms, orch.)	Medium-High
B: Namespace grouping with Tenant resources	High	Medium-High	Some restrictions eg cluster scoped rescs.	Yes	Medium
C: Virtual Kubernetes Clusters	High	High	No (?)	Yes	Early
D: Core Kubernetes change (Tenant as 1st class resource)	High	High	No (?)	Yes (in theory)	Very low (design does not exist)

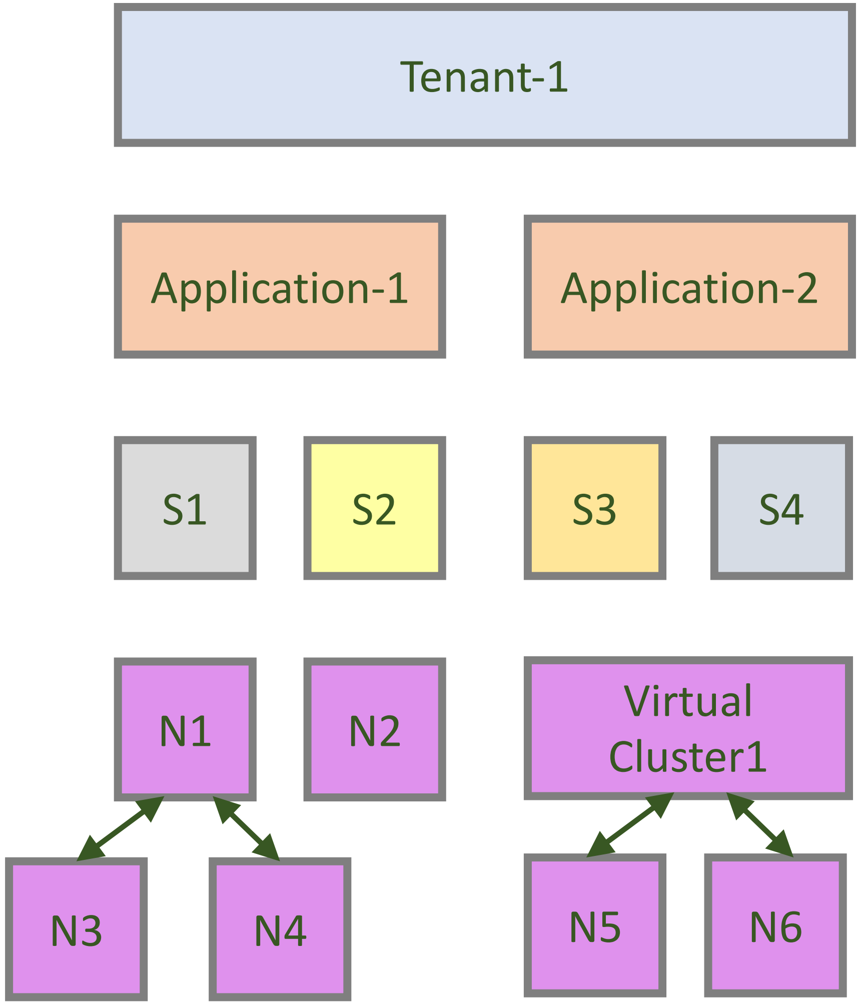
Mapping Tenants, Applications, Services



1 tenant <> 1 app <> 1 NS
(M micro-services all in 1 NS)
Need to resolve naming conflicts

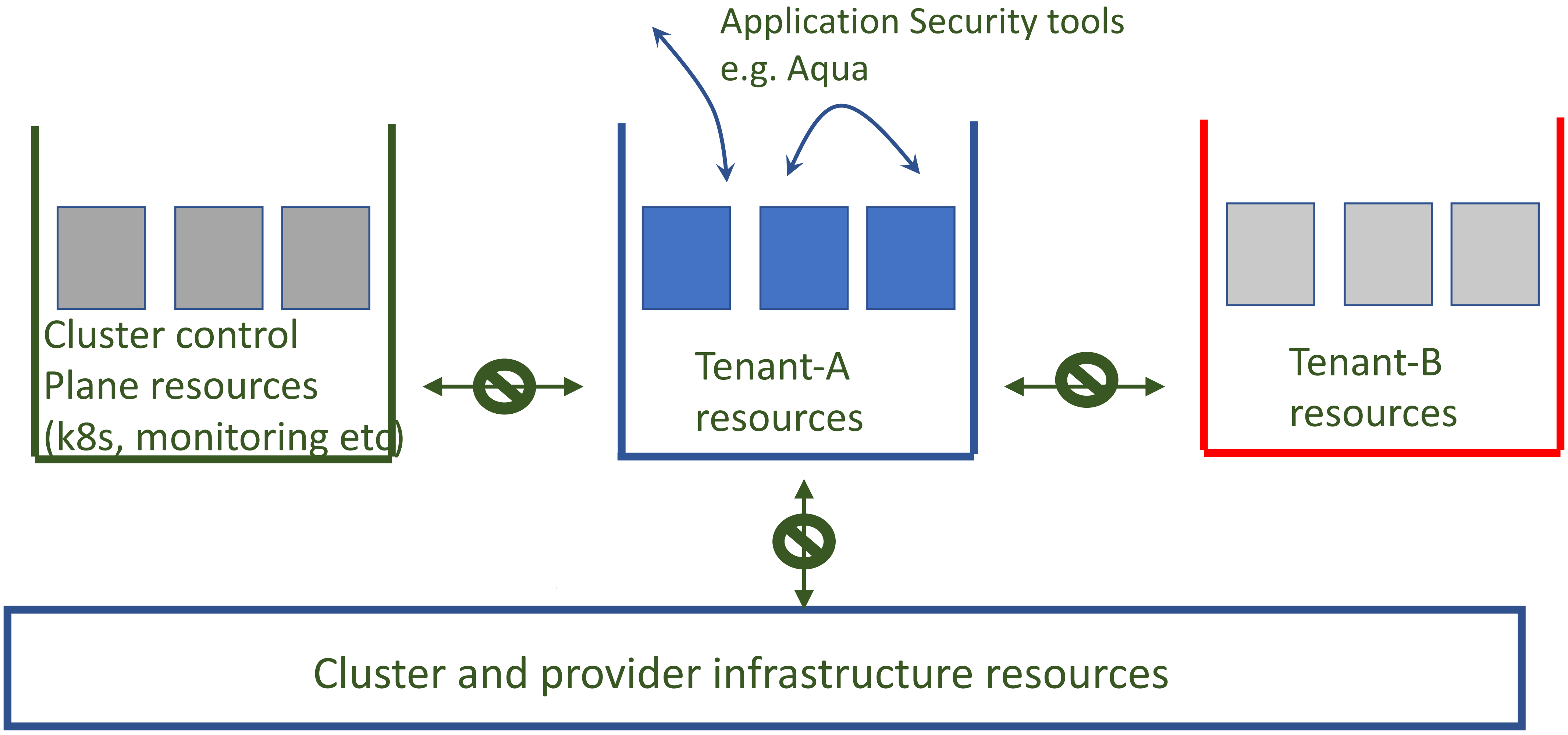


1 tenant <> 1 app <> M NS
(1 service per NS)
Better service portability



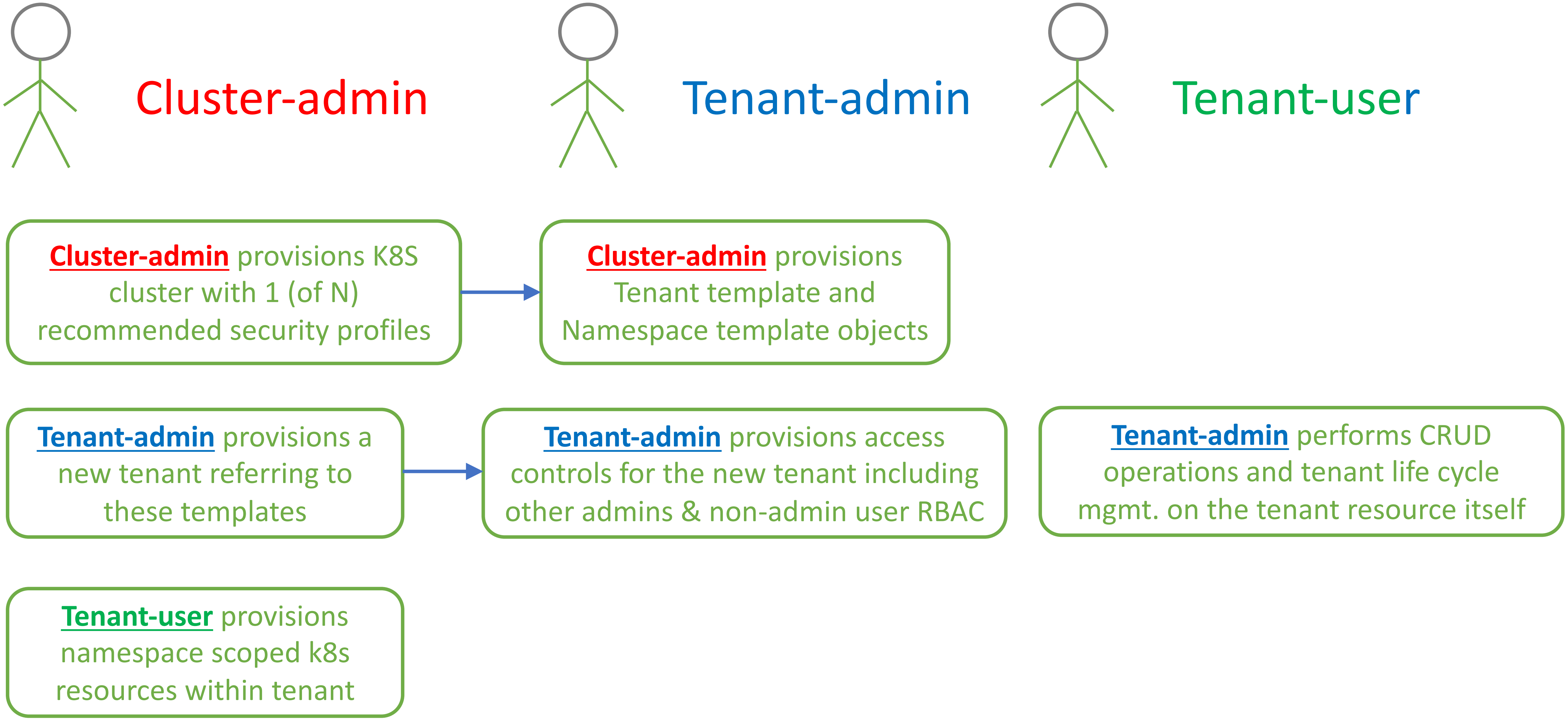
1 tenant <> M apps <> mix of H-NSs & VCs

Tenant vs Application Security Responsibility Model



Community Initiatives: Multitenancy Control Plane

Operational Model: Personas and workflows



Cluster-admin

Tenant-admin

Tenant-user

Cluster-admin provisions K8S cluster with 1 (of N) recommended security profiles

Cluster-admin provisions Tenant template and Namespace template objects

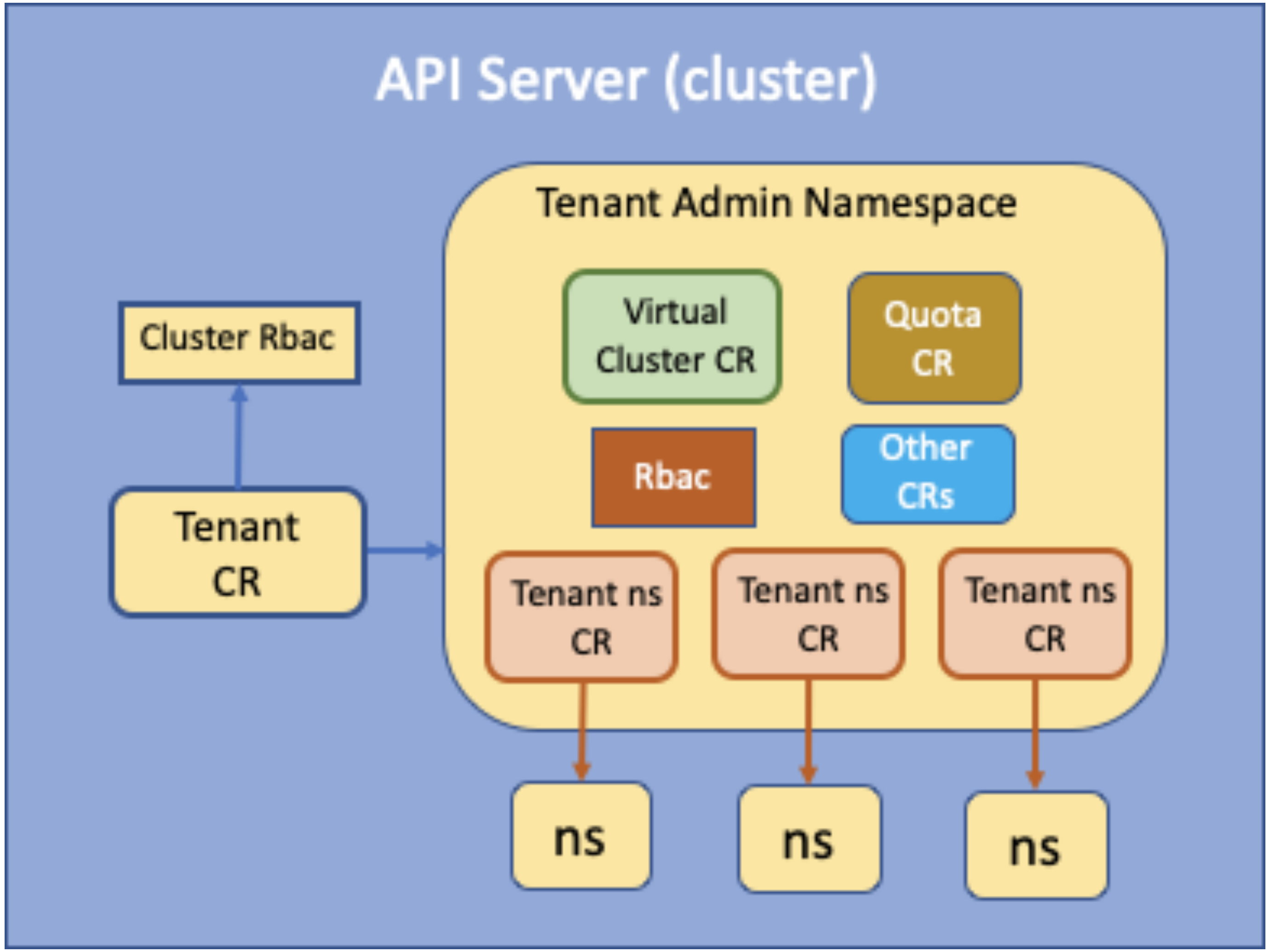
Tenant-admin provisions a new tenant referring to these templates

Tenant-admin provisions access controls for the new tenant including other admins & non-admin user RBAC

Tenant-admin performs CRUD operations and tenant life cycle mgmt. on the tenant resource itself

Tenant-user provisions namespace scoped k8s resources within tenant

Tenant Operator Model



- Self-service or Admin-provisioned Tenants
- Each Tenant-CR manages a collection of namespaces, virtual clusters and associated resources via corresponding CRs that eventually own those K8s resouces
- Named admins + named resource RBAC

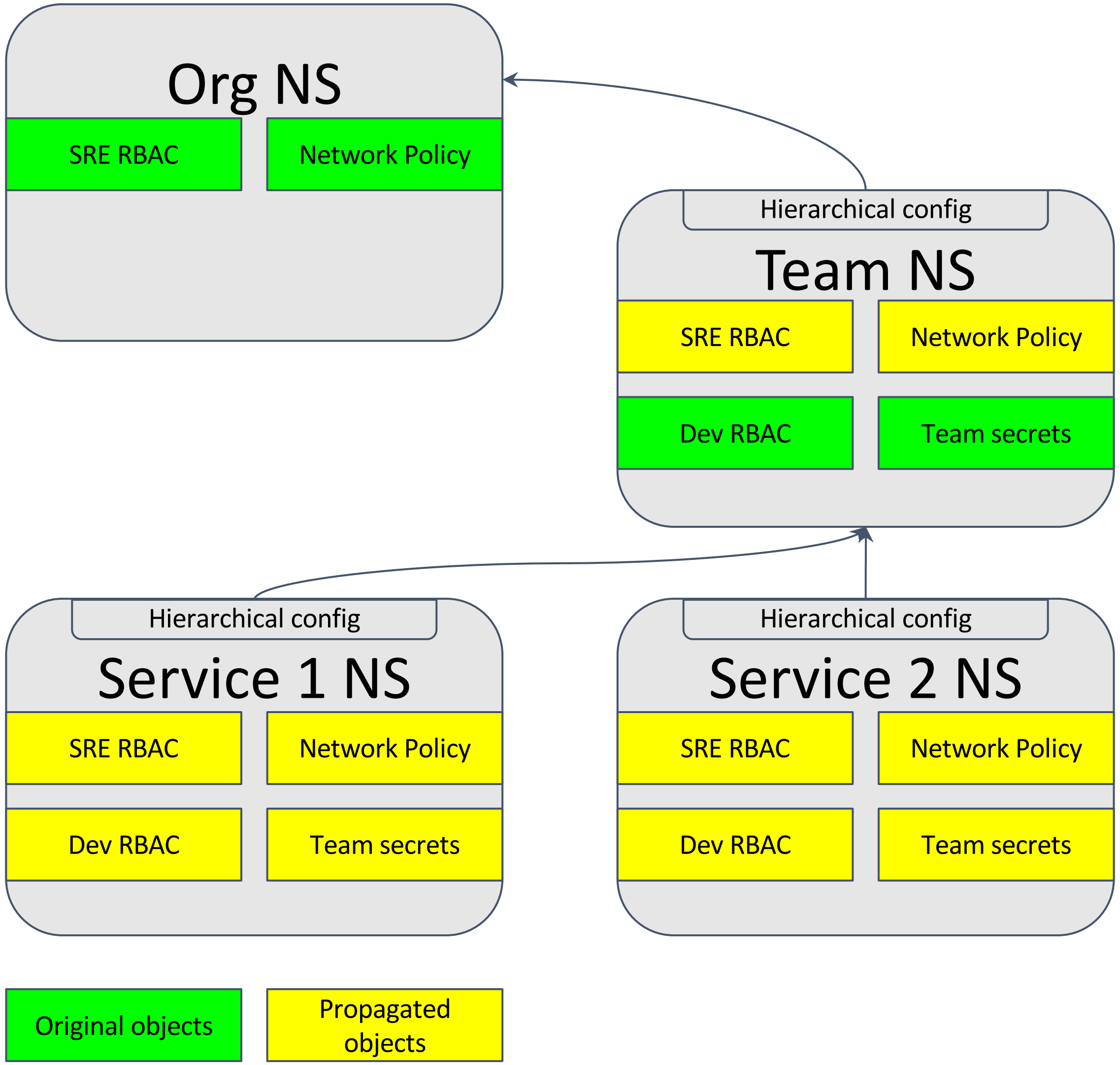
Sample config

```
apiVersion: tenancy.x-k8s.io/v1alpha1
kind: Tenant
metadata:
  labels:
    controller-tools.k8s.io: "1.0"
  name: tenant-t1
spec:
  tenantAdminNamespaceName: t1-adm
  requireNamespacePrefix: true
  tenantAdmins:
    - kind: ServiceAccount
      name: t1-user1
      namespace: default
```

```
apiVersion: tenancy.x-k8s.io/v1alpha1
kind: TenantNamespace
metadata:
  labels:
    controller-tools.k8s.io: "1.0"
  name: tns-t1-n1
  namespace: t1-adm
spec:
  # Add fields here
  name: t1-adm-ns1
```

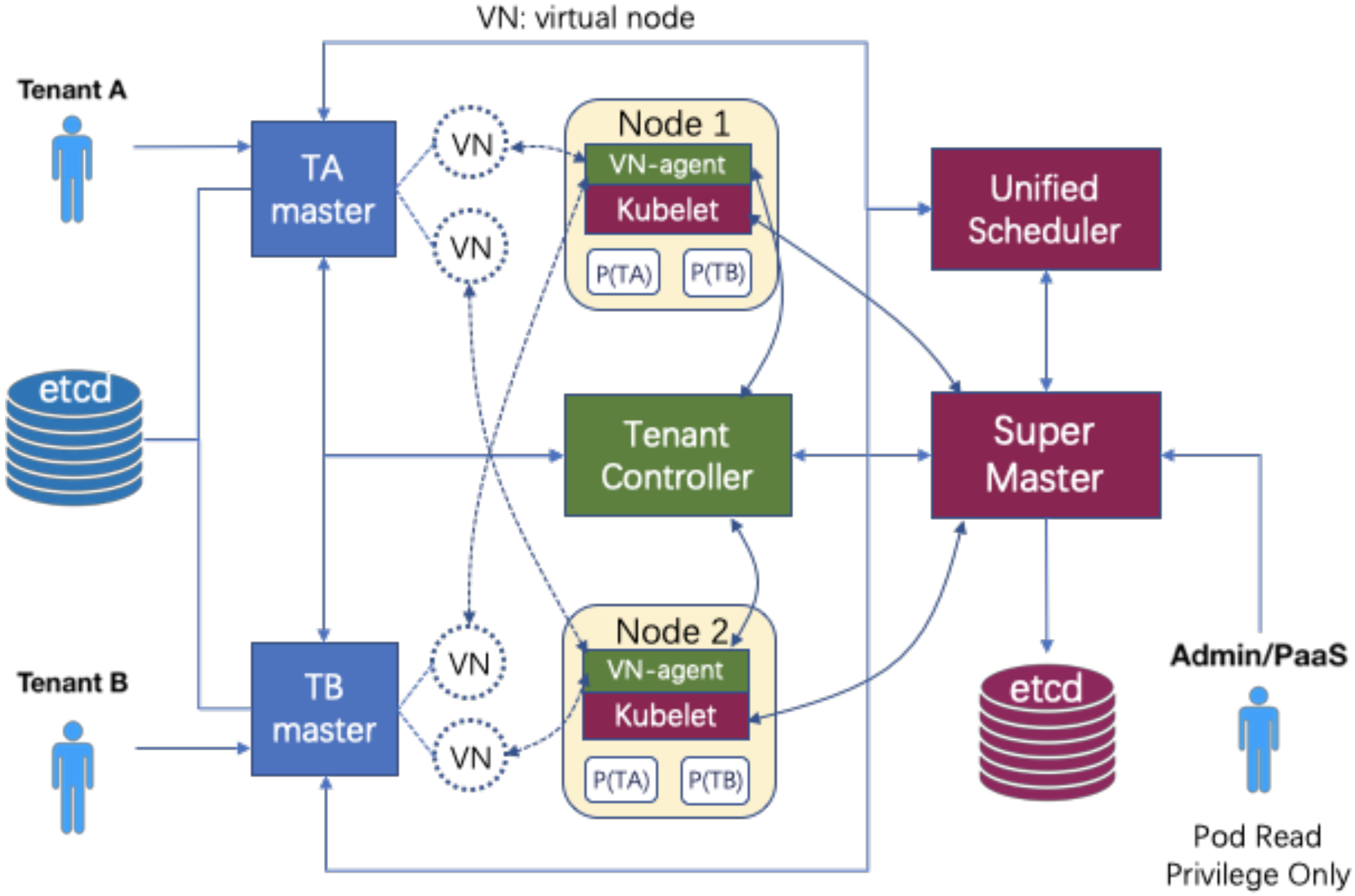
Hierarchical Namespace Controller

- Propagates policy objects from parents to children
  - Hardcoded list in v0.1 (Nov), aim to be configurable in v0.3 (early 2020)
- Self-service subnamespaces
  - No need for cluster-level privileges to create subnamespaces
- Hierarchical authz checks
  - “Subadmins” cannot deprive “superadmins” of access
- Integrations via K8s labels
  - Namespaces receive labels indicating the subtrees they’re in.



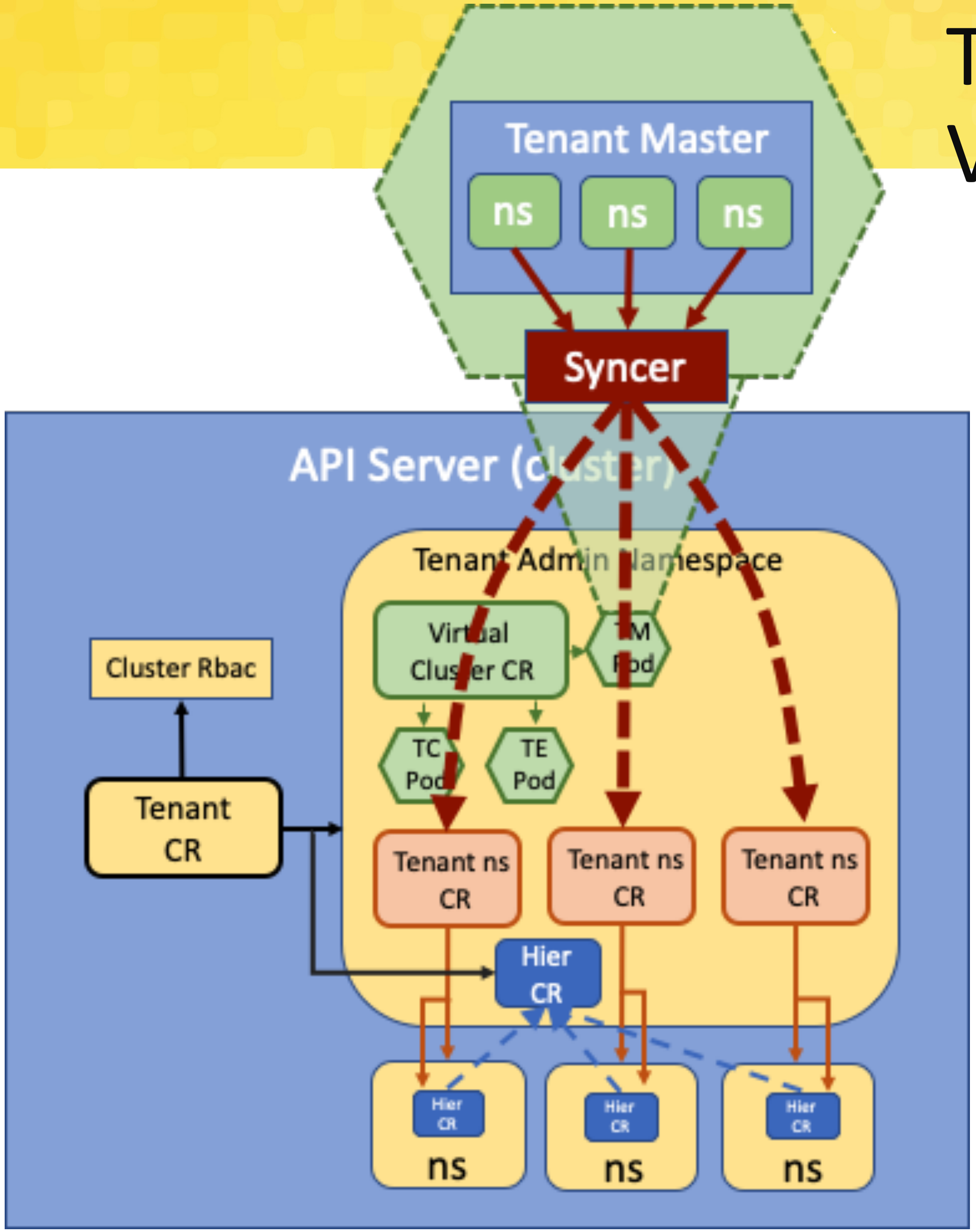
🟩 Original objects
🟨 Propagated objects

Virtual Kubernetes Clusters Model



Virtual Cluster Architecture Proposal; F Guo et al; Alibaba Cloud

Tenant Operator + Virtual Cluster + HNC (optional)



Data plane and Benchmarking

Multitenancy Benchmarks

- **Goals**: validate whether multi-tenancy has been achieved, independently of how its configured
- Decouple how multi-tenancy is provisioned and managed from the desired state.
- Define the desired states for multi-tenancy
- Provide automated tests for validating the desired states

MT Profile Level	Intent
Level 1	Uses K8s API objects; can be manually configured; limited tenancy features
Level 2	Level 1 + allow extensions for self-service DevOps i.e. namespace creation, etc.
Level 3	Level 2 + ability to create CRDs,etc. (virtual control plane)

Benchmark Categories & Formal Definition

- Categories:
  1. **Control Plane Isolation (CPI)**
  2. **Tenant Isolation (TI)**
  3. **Network Isolation (NI)**
  4. **Host Isolation (HI)**
  5. **Data Isolation (DI)**
  6. **Fairness (FNS)**
  7. **Self-Service Operations (OPS)**
- Formatted similar to CIS benchmarks
- Test suite implemented using k8s e2e tests framework
- Open development model: community submits PRs for candidate benchmark tests and implementations

Example: MTB-PL1-CC-CPI-1

- **Profile Applicability:**
  - Level 1
- **Type:**
  - Behavioral Check
- **Category:**
  - Control Plane Isolation
- **Description:**
  - Tenants should not be able to …
- **Rationale:**
  - Tenants should not be able to access control plane resources ...

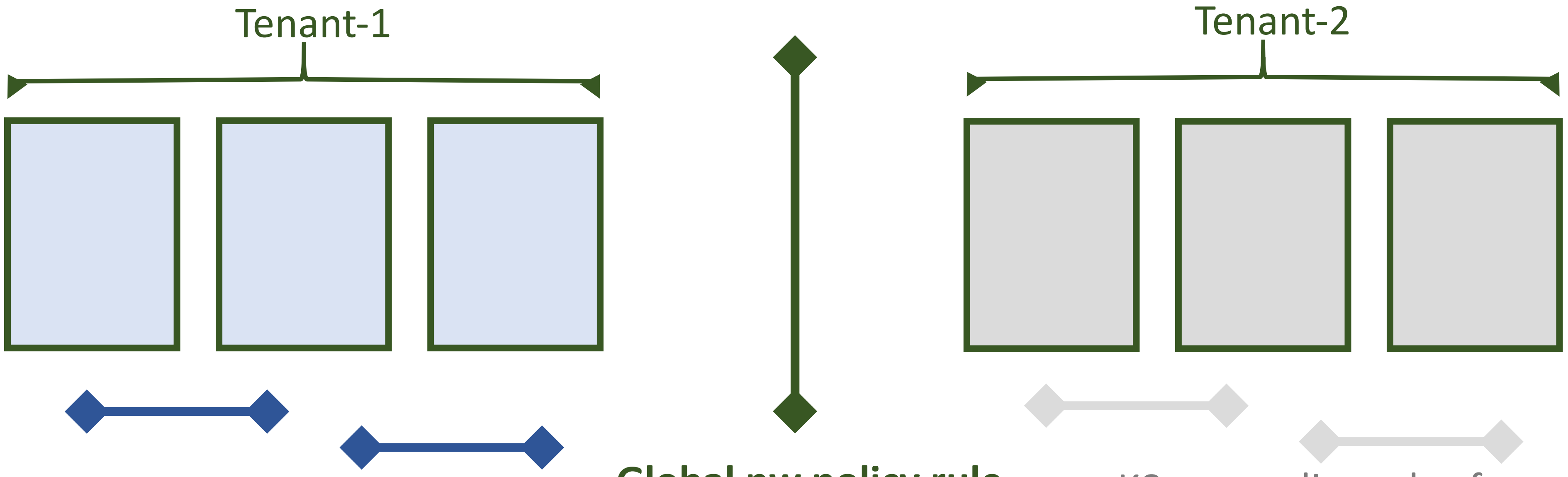
- **Audit:**
  - Run the following commands to retrieve the list of non-namespaced resources:
  - kubectl --kubeconfig cluster-admin api-resources --namespaced=false For all non-namespaced resources, and each verb (get, list, create, update, patch, watch, delete, and deletecollection) issue the following commands:
  - kubectl --kubeconfig tenant-a auth can-i <verb> <resource> Each command must return 'no'

Example Baseline Reference Implementation:

- Control Plane:
  - Namespace Grouping Model (Tenant Operator based)
- Data Plane:
  - containerD/ CRI-O runtime
  - Container sandboxing
    - Pod Security Policy (+Apparmor, Seccomp)
    - Kata containers
  - K8s Network Policy
  - (CNI vendor specific) Global Network Policy
    - Supported by Calico, Cisco ACI, Cilium, (others ?)
- Dynamic policy admission controller/ framework
  - Open Policy Agent/ Gatekeeper/ Kyverno/ K-rail ..

Network Policy: Global Policy + K8s Policy

- Current K8s Network Policy is namespace scoped only non-ideal for Multi-tenancy
- Recommendation: Use a combo of K8s Network Policy + (CNI-specific) Global Network Policy
- Global Network Policy: Tool for Cluster Admin to isolate tenants
- K8s Network Policy: Developers, Devops use for micro-segmentation



Global nw policy rule
For tenant isolation

K8s nw policy rules for
App team microsegmentation

Global Network Policy Calico v3.7 (demo only) example

(ps. use Calico 3.10 namespaceselector for better rule options)

```
---
kind: GlobalNetworkPolicy
apiVersion: crd.projectcalico.org/v1
metadata:
  name: isolate-tenant-1
spec:
  types:
  - Ingress
  - Egress
```

```
order: 10
ingress:
- action: Deny
  source:
    namespaceSelector: tenant != 't1'
  destination:
    namespaceSelector: tenant == 't1'
- action: Allow
egress:
- action: Deny
  source:
    namespaceSelector: tenant == 't1'
  destination:
    namespaceSelector: tenant != 't1'
- action: Allow
```

Sample Cluster Setup Reference Configurations

Profile 1: Basic

- Secure by default Kubernetes configuration
  - Disable anonymous authentication
  - Disable ABAC, disable local authorization,
  - K8S secrets encryption enabled
  - CIS Kubernetes benchmarks Level 2 requirements
- Enable RBAC
- Recommended default set of admission controllers (NodeRestriction, AlwaysPullImages, PodSecurityPolicy etc)
- Pod Admission controller (PodSecurityPolicy)
- CNI Container Network Policy enabled including ingress and egress policies
- Docker run-time with Seccomp, AppArmor/ SELinux default profiles
- Best effort multi-tenancy for services (monitoring, logging etc)

Profile 2:

- Profile 1 + additional required enhancements including:
- Dynamic policy engine (e.g. OPA) based enhancement for
  - Access control/ RBAC
  - Admission control (beyond Pod Security policies)
  - Advanced policy controls (e.g. ingress route policies)
- Newer container runtimes & runtime sandboxing options (CRI-O, containerD w/ Kata runtime, Firecracker/ gVisor)
- Complete solution for multi-tenancy across monitoring, logging, storage, service mesh ..
- Tenancy across Multi-cluster, multi-cloud

Demo