KubeCon | CloudNativeCon

North America 2024

KubeCon CloudNativeCon

North America 2024

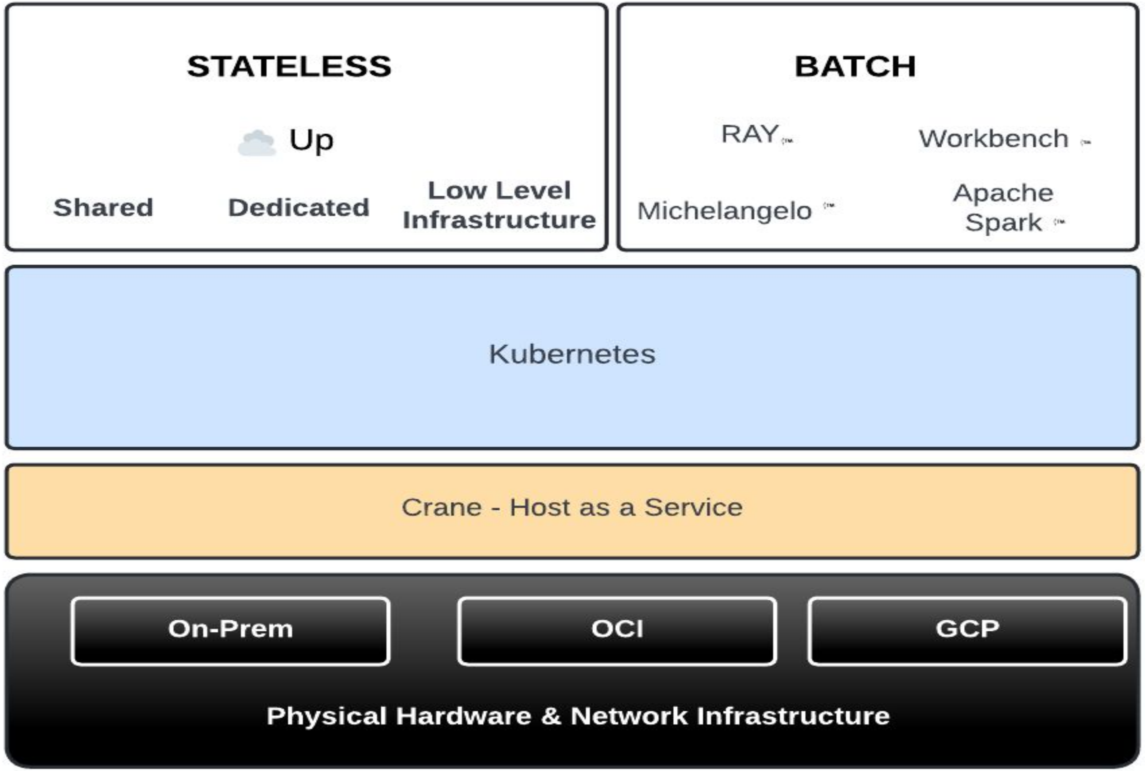
Share the ride: Robust Multi-Tenancy in Kubernetes at Uber

Apoorva Jindal
Sashank Appireddy

Agenda

- Introduction
- Multi-Tenancy Requirements
- Solution #1: Separate Cluster per Tenant
- Challenges
- Multi Tenant Single Cluster Architecture
- Migration Status
- What works
- Acknowledgements
- Q & A

Platform Overview



STATELESS

Up

Shared

Dedicated

Low Level Infrastructure

BATCH

RAY

Workbench

Michelangelo

Apache Spark

Kubernetes

Crane - Host as a Service

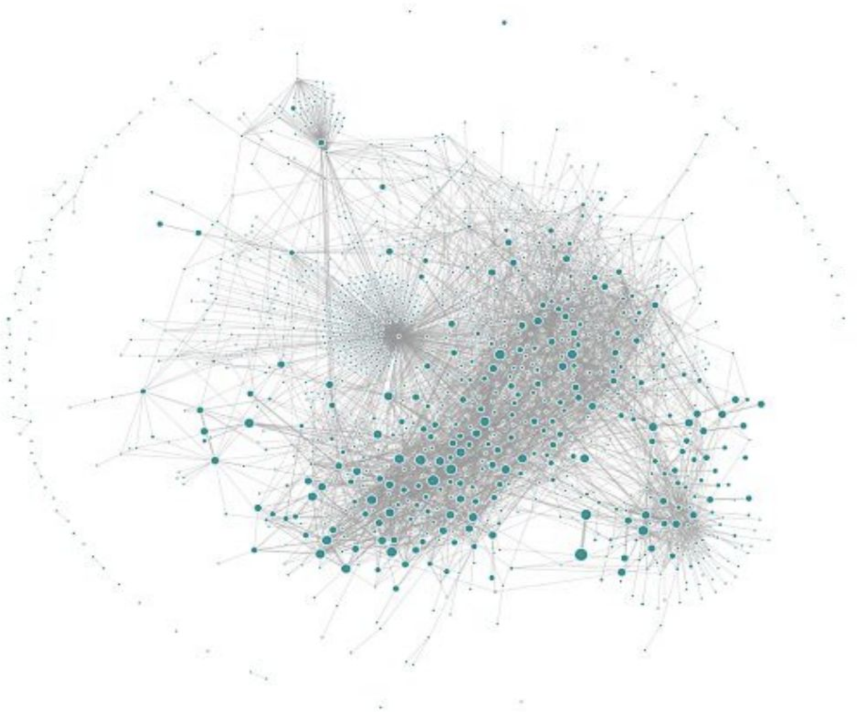
On-Prem

OCI

GCP

Physical Hardware & Network Infrastructure

Platform Overview



4000+ microservices

4.5M+ cores

100K+ service deploys per day

1.5M+ containers deployed per day

500K+ containers

KubeCon | CloudNativeCon
North America 2024

Multi-Tenancy Requirements

Multi-Tenancy Requirements

Requirement #1: Data-Plane Isolation

- Tenants do not share hosts
- Workloads belonging to the same tenant can share the host, but not across tenants

Tenant #1

Node Pool

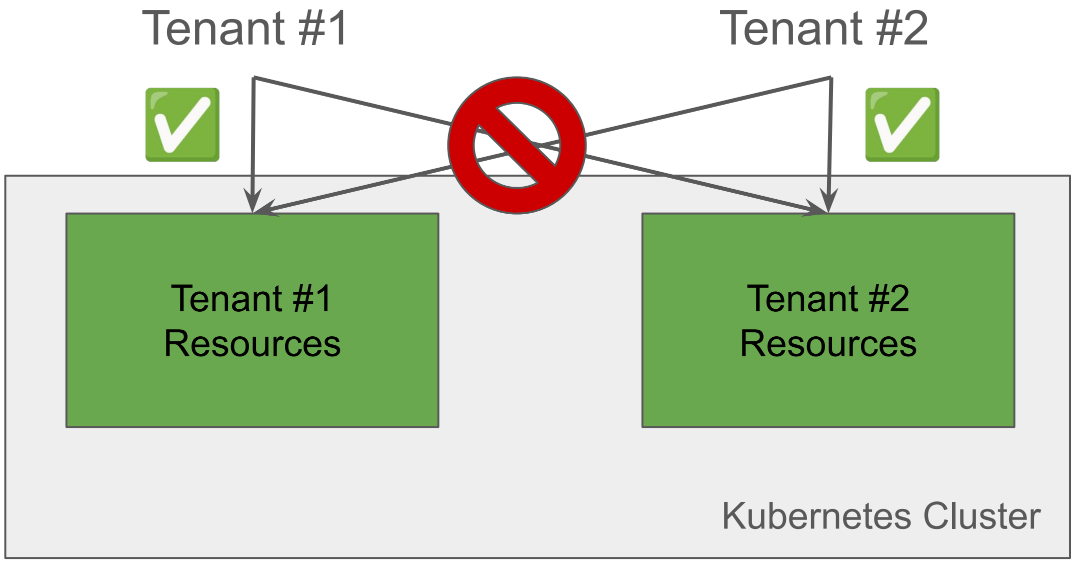
Tenant #2

Node Pool

Multi-Tenancy Requirements

Requirement #2: Access Isolation

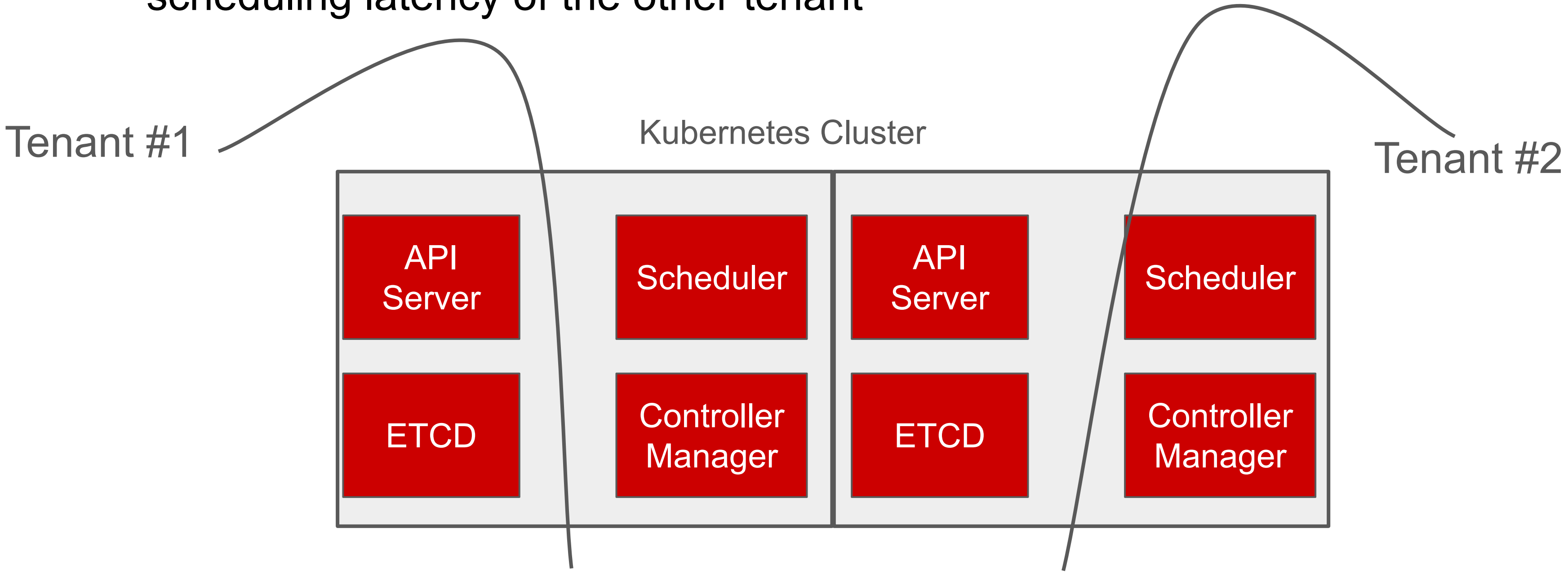
- Tenants cannot access resource information about other tenants
  - E.g. pods, nodes, deployments etc



Multi-Tenancy Requirements

Requirement #3: Control-Plane Isolation

- Tenant is not impacted by other tenants in the control plane
- E.g. a high scheduling throughput of one tenant does not impact low scheduling latency of the other tenant

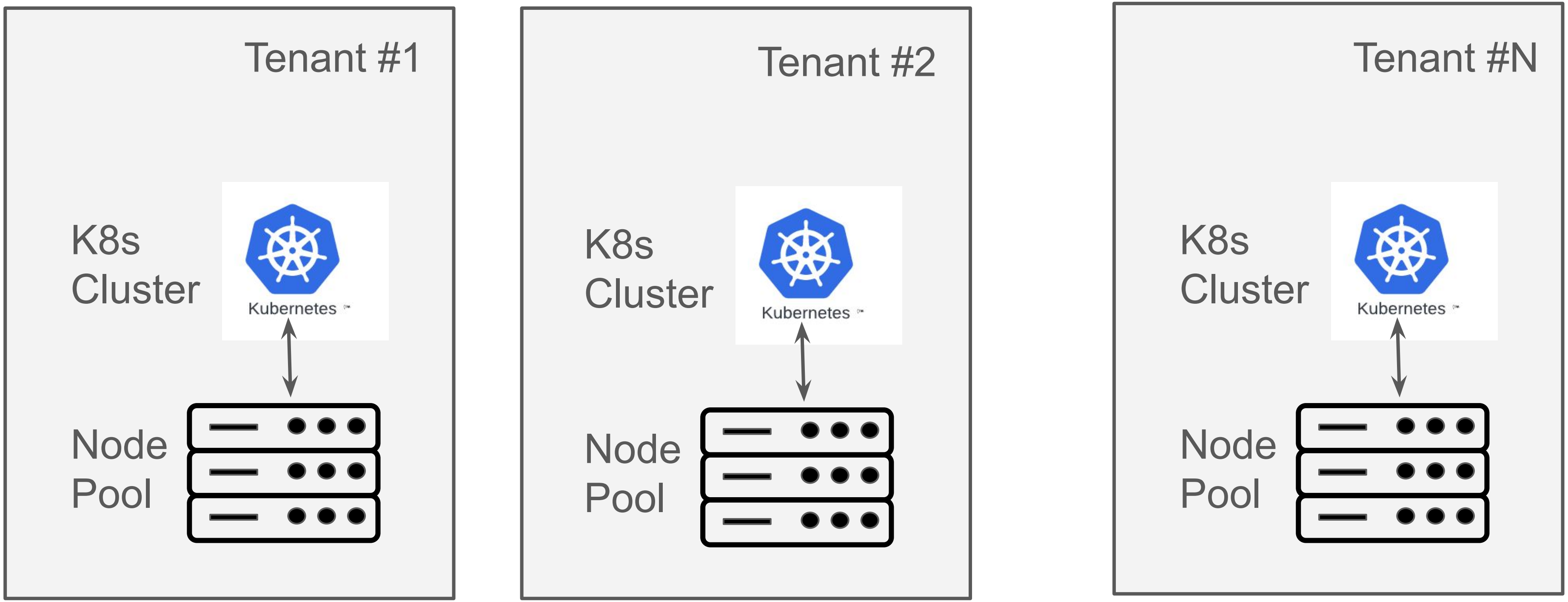


KubeCon CloudNativeCon
North America 2024

One Cluster per Tenant

Solution #1: Separate Cluster per Tenant

Each Tenant gets its own Kubernetes Cluster and a Dedicated Node Pool



Tenant #1

K8s Cluster

Node Pool

Tenant #2

K8s Cluster

Node Pool

Tenant #N

K8s Cluster

Node Pool

Use Cases

- Certain workloads isolated from others due to security concerns
  - Requires all 3 - data plane isolation, access isolation and control plane isolation

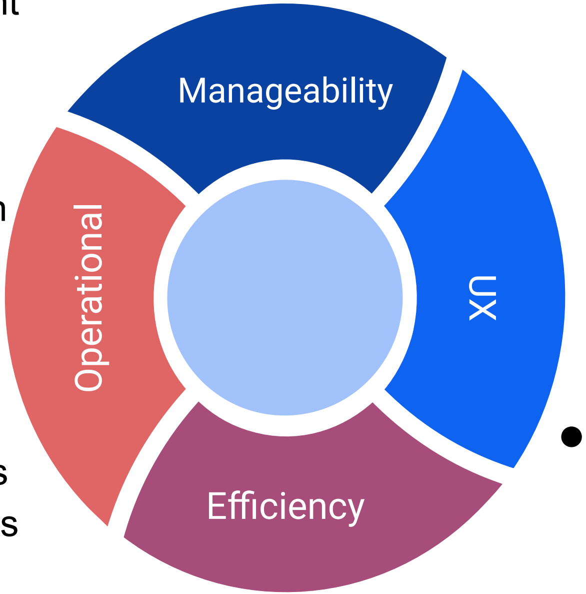
Use Cases

- Certain workloads isolated from others due to security concerns
  - Requires all 3 - data plane isolation, access isolation and control plane isolation
- Isolate noisy neighbors
  - Requires data plane isolation and control plane isolation
- Workload requiring specific hardware
  - Requires data plane isolation
- And many more

Challenges

- Manageability
  - Every cluster has a different configuration
  - Managed via cluster types
  - Error-prone
  - Feature mismatch between tenants

- Operational Concerns
  - Incident mitigation requires understanding cluster types



- User Experience:
  - Operational cost high for tenants
  - Every tenant operation requires a multi-step runbook
  - E.g. to grow in a new zone, the tenants first should request their cluster and node pool

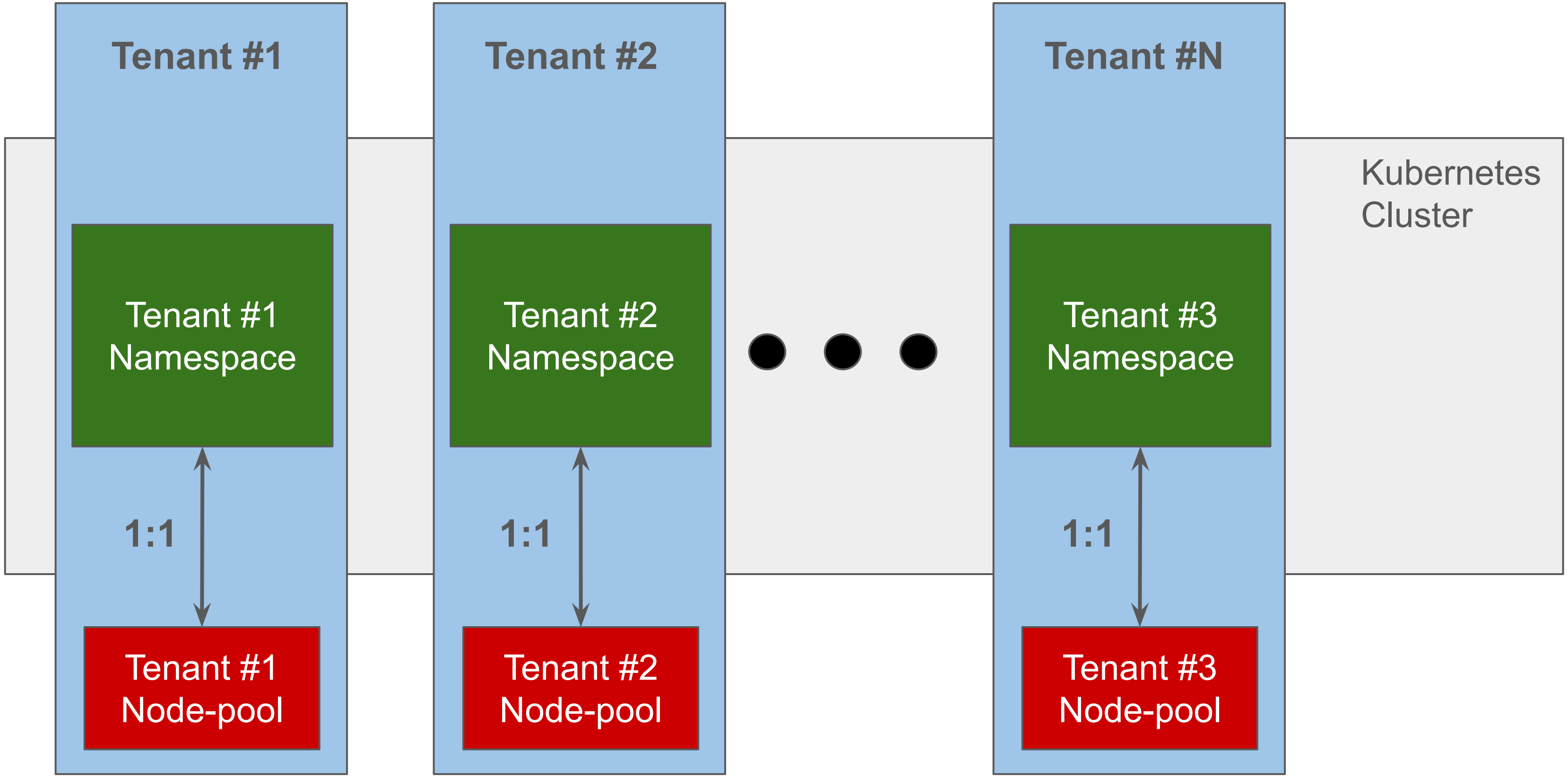
- Efficiency
  - Control plane cost per cluster
  - Every cluster maintains its own free pool buffer

KubeCon | CloudNativeCon

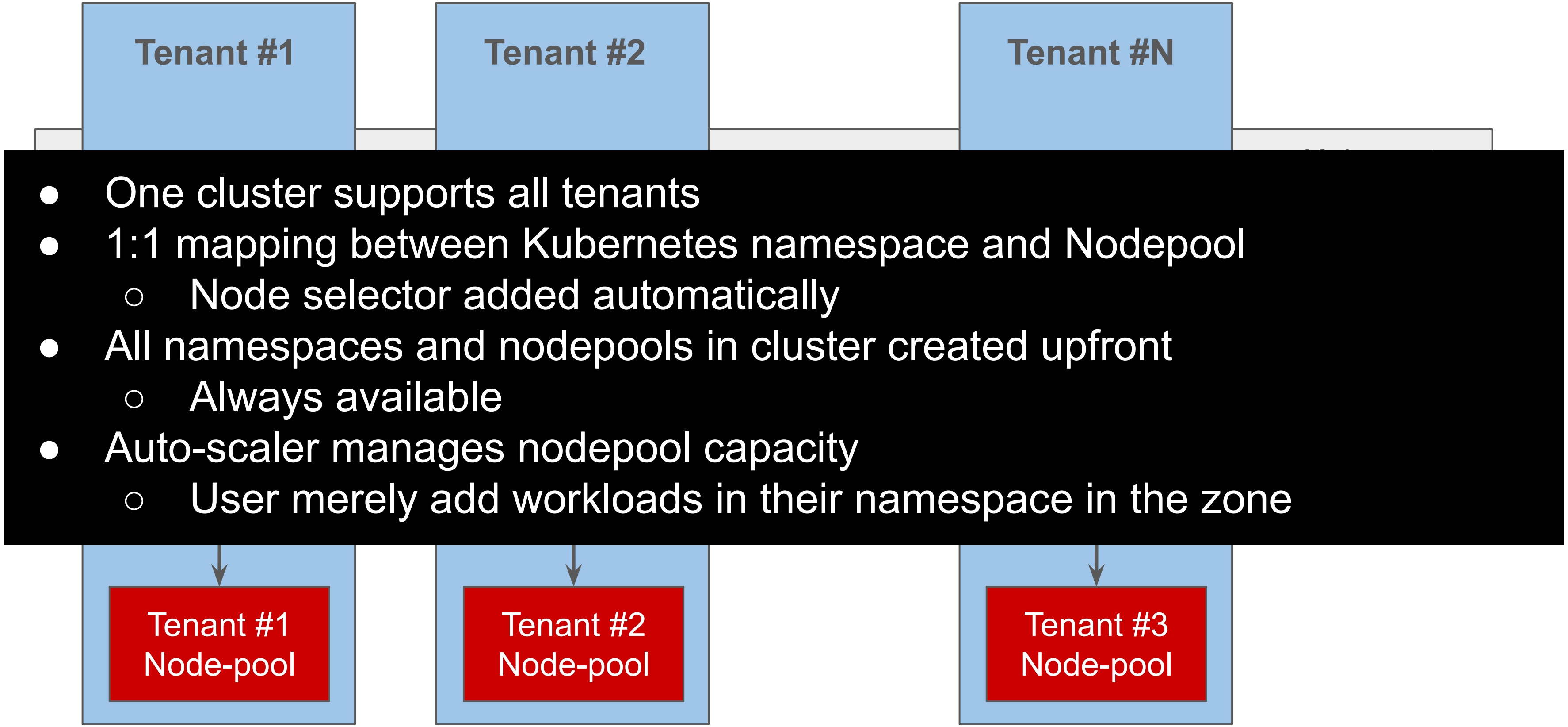
North America 2024

Multi-Tenant Single Cluster

Multi Tenant Single Cluster Architecture



Multi Tenant Single Cluster Architecture



- One cluster supports all tenants
- 1:1 mapping between Kubernetes namespace and Nodepool
  - Node selector added automatically
- All namespaces and nodepools in cluster created upfront
  - Always available
- Auto-scaler manages nodepool capacity
  - User merely add workloads in their namespace in the zone

#1 Access Isolation

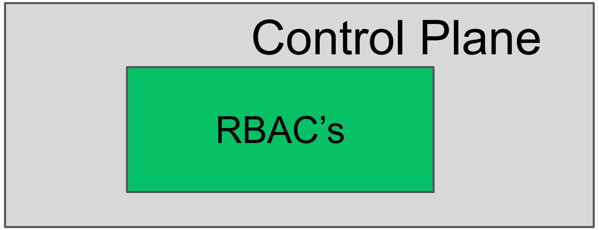
🟩 Default NS Aware

🟦 Extended Support

Per Tenant:

RBAC’s

- Roles:
  - Actions to manage resources
- RoleBindings
  - Associates tenant users to roles



Control Plane

RBAC’s

Roles

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
    namespace: <tenant-name>
    name: <tenant-role-name>
rules:
    - apiGroups: ["", "apps"]
      resources: ["pods"]
      verbs: ["get", "list", "watch", "create", "update", "delete"]
```

→

Role Bindings

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
    name: <tenant-rolebinding-name>
    namespace: <tenant-name>
subjects:
    - kind: User
      name: <tenant-user-name>
      apiGroup: rbac.authorization.k8s.io
roleRef:
    kind: Role
    name: <tenant-role-name>
    apiGroup: rbac.authorization.k8s.io
```

#2 Control Plane Isolation

APF:

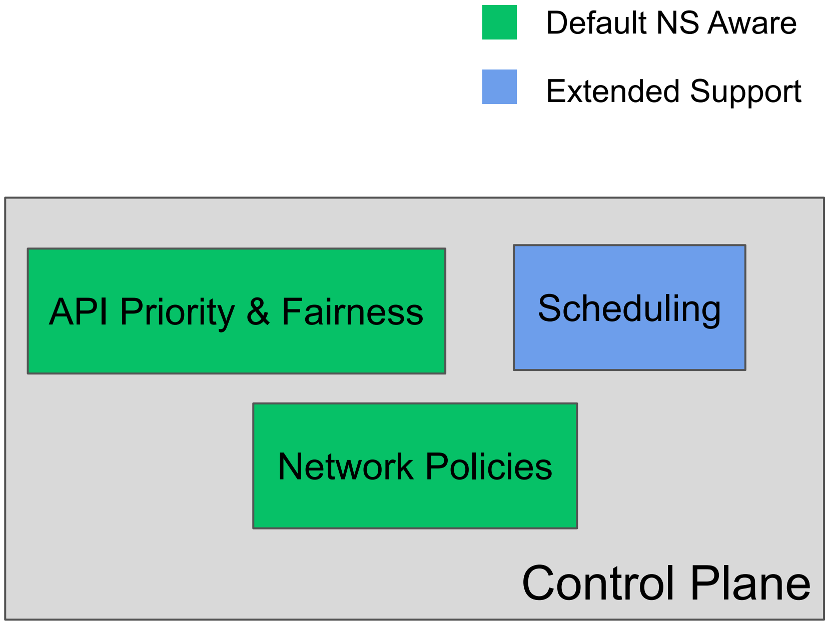
- Flow Schemas that control the fair API sharing.
- Priority Settings: Concurrency shares / queues per tenant
- Default rate limits for majority of the tenants.
- Custom overrides for specific tenants.

Network Policies:

- Prevent inter tenant communication, effectively enforcing network level isolation
- Explicit ingress/egress policies to limit access to/from external endpoints

Scheduling:

- Leveraged default scheduler with extended support for node specific labels
- Actively working on options to achieve scheduler level isolation per tenant



🟩 Default NS Aware

🟦 Extended Support

API Priority & Fairness

Scheduling

Network Policies

Control Plane

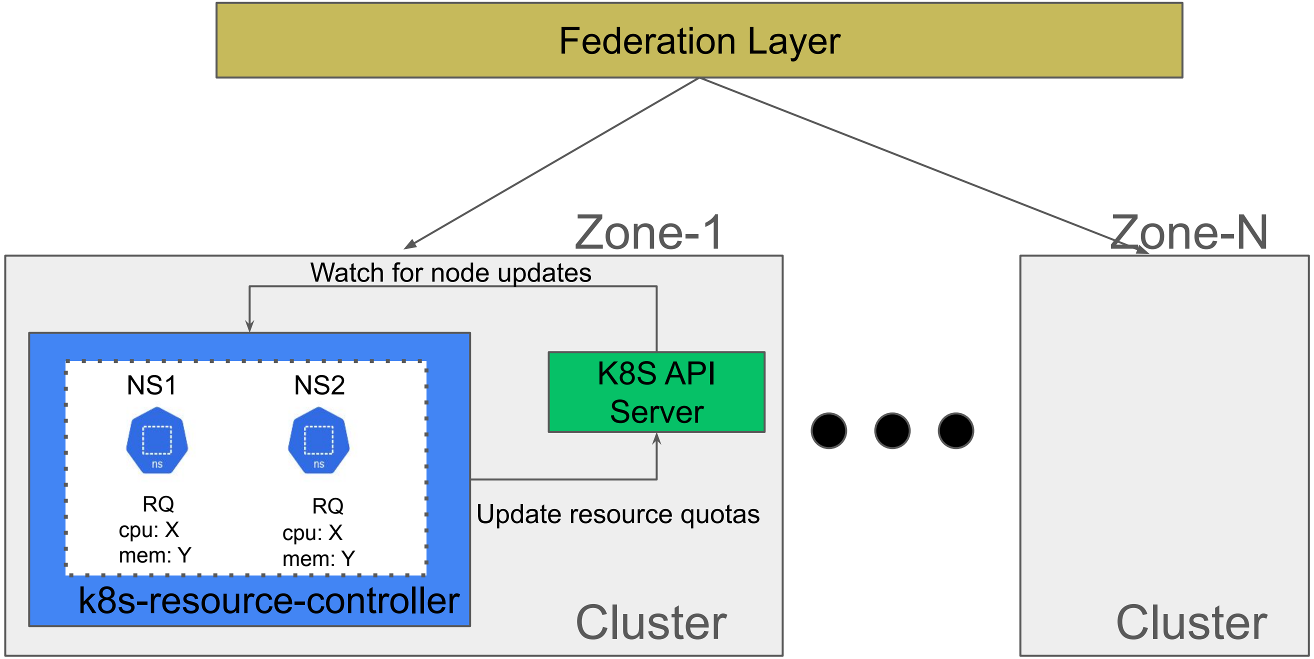
#2a Resource Quota Per Tenant

Federation layer:

- Has global view of the capacity per tenant across zones
- Picks the least loaded zones and schedules the workloads
- Contacts zonal control plane to receive up to date capacity info

Resource Controller:

- Has zonal view of the capacity per tenant
- Aggregates resources by filtering nodes matching namespace label



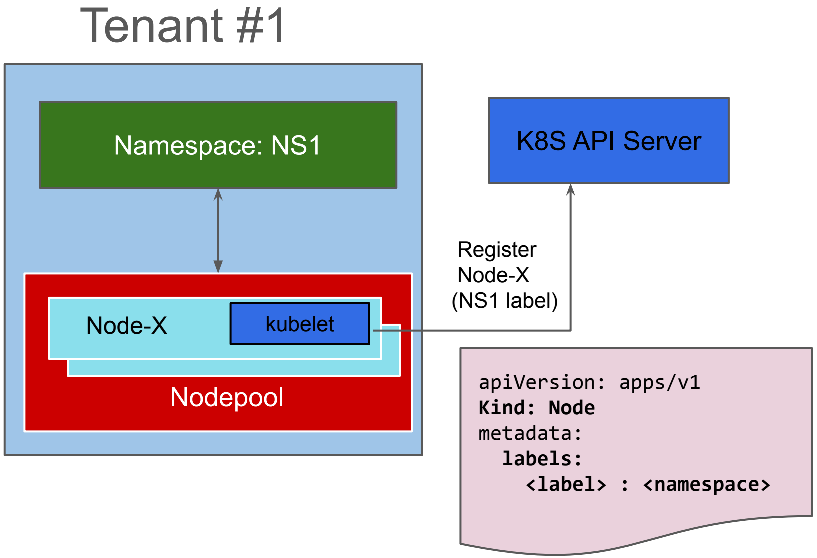
#3 Data Plane Isolation

NodePools:

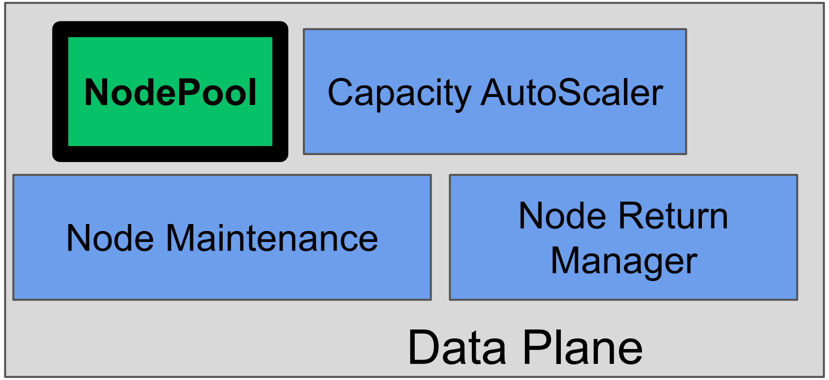
- Per tenant dedicated group of nodes
- 1:1 associated with tenant namespaces
- Rely on node labels to assign nodes to a nodepool

🟩 Default NS Aware

🟦 Extended Support



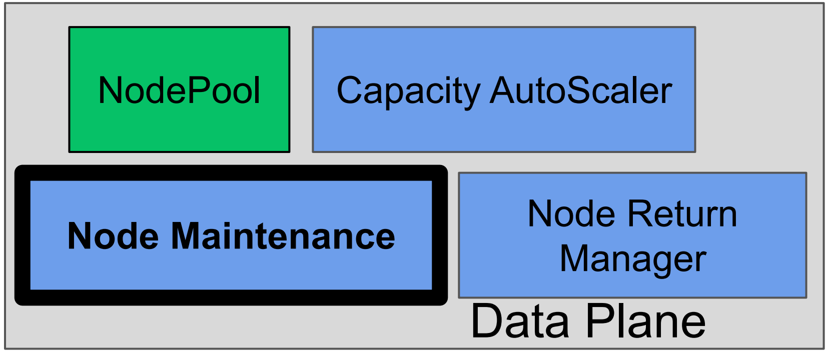
```
apiVersion: apps/v1
Kind: Node
metadata:
  labels:
    <label> : <namespace>
```



#3 Data Plane Isolation (cont’d)

Node Maintenance:

- Perform operational activities (upgrades, etc..) to keep secure, stable, and performant nodes in the cluster.
- Drain Limit: Max Concurrent nodes in maintenance
- Admission controller plugin for validating nodes in maintenance per tenant



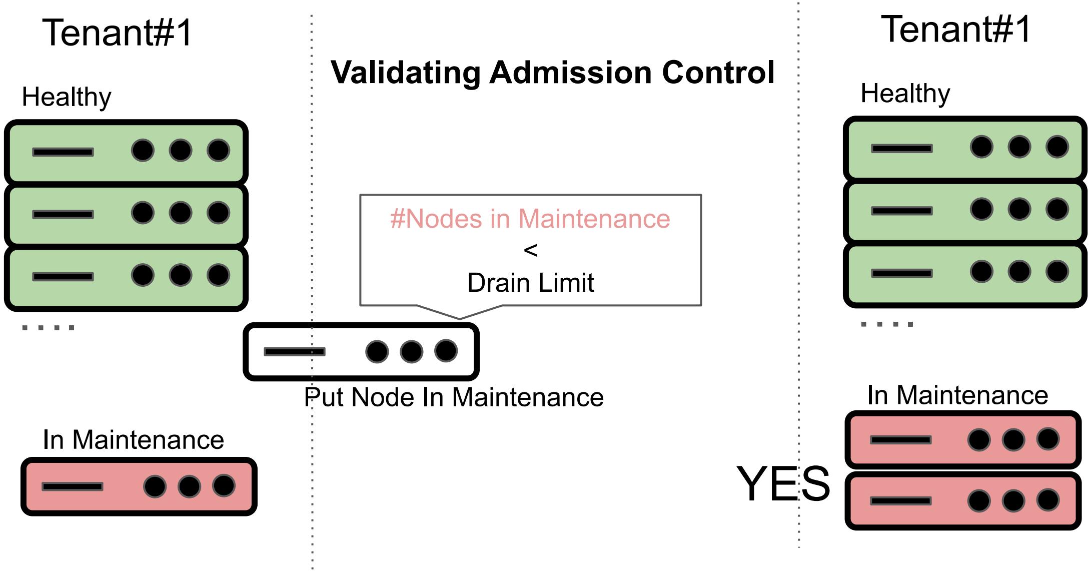
NodePool

Capacity AutoScaler

Node Maintenance

Node Return Manager

Data Plane



Tenant#1

Healthy

....

In Maintenance

Validating Admission Control

#Nodes in Maintenance < Drain Limit

Put Node In Maintenance

YES

Tenant#1

Healthy

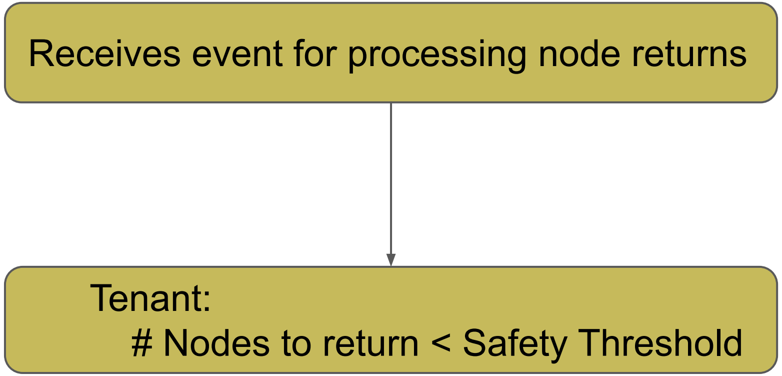
....

In Maintenance

#3 Data Plane Isolation (cont’d)

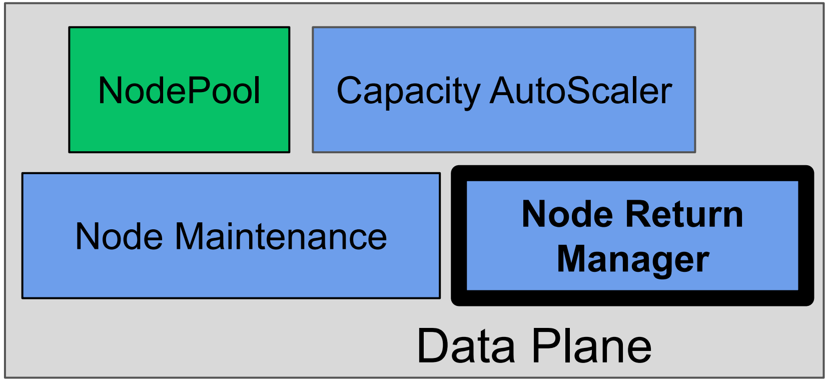
Node Return Manager:

- Evaluator library that calculates number of nodes to be returned safely
- **Safety Threshold**: Max allocation % beyond which workloads become unsafe
- Return criteria policies:
  - Allocation % of the tenant
  - Pod Topology Spread, Failure Domain, etc..


Receives event for processing node returns
Tenant:
# Nodes to return < Safety Threshold

🟩 Default NS Aware

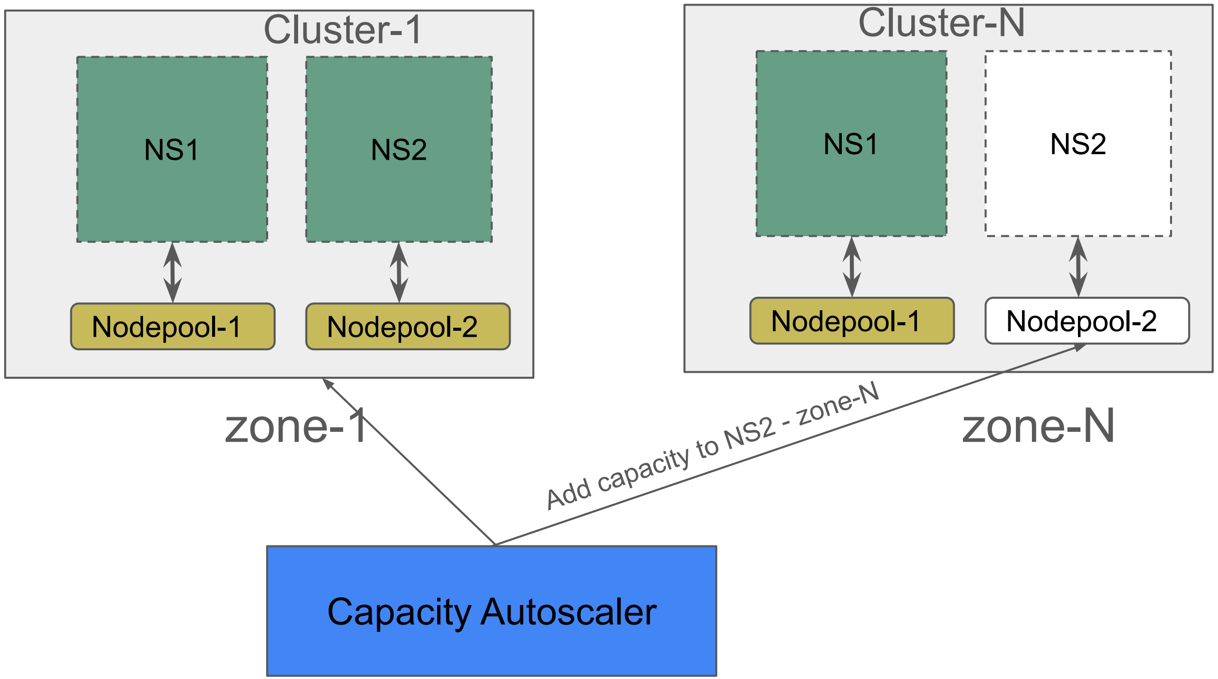
🟦 Extended Support



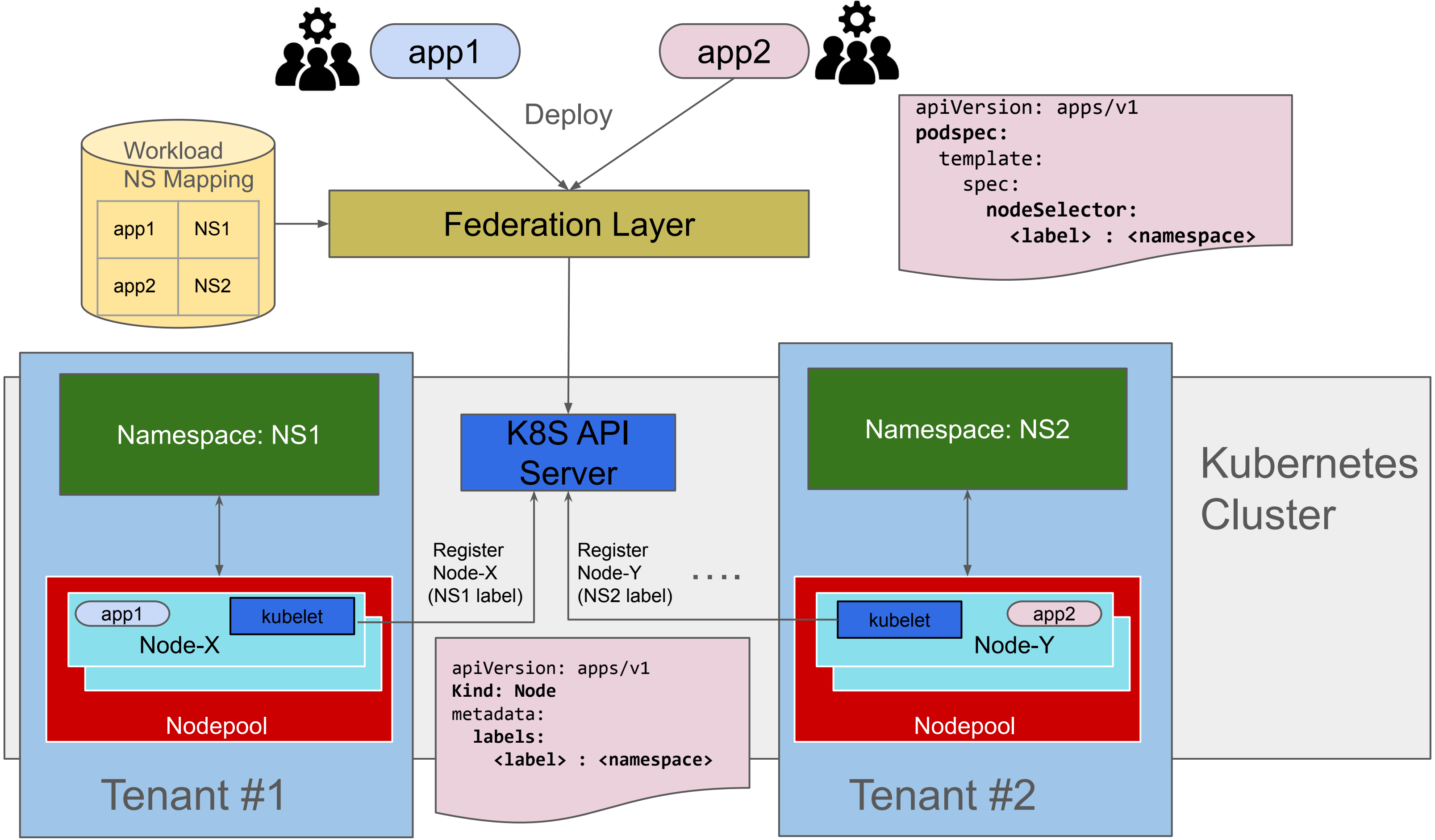
#3 Data Plane Isolation (cont'd)

Capacity AutoScaler:

- Manages the capacity across nodepools / zones.
- **+** capacity when there are unscheduled pods.
- **-** capacity when the allocation drops.
- Improved scheduling latency with hot standby buffer pool
- Inter nodepool swaps are fairly trivial with merely node label swaps



Workload Deploy



Workload NS Mapping

app1	NS1
app2	NS2

```
apiVersion: apps/v1
podspec:
  template:
    spec:
      nodeSelector:
        <label> : <namespace>
```

```
apiVersion: apps/v1
Kind: Node
metadata:
  labels:
    <label> : <namespace>
```

Migration Status

- Tens of namespaces 100% migrated to new architecture
- 30% reduction in the number of clusters needed globally
- End Target - 100% by 2025

KubeCon | CloudNativeCon

North America 2024

What Worked & Challenges

What worked vs Challenges

✅

- Operational ease in managing configs
  - Single cluster per zone
- NodeSelector scalability
  - Tens of namespaces live in production.
- Seamless capacity management
- Inter tenant capacity swaps are trivial with label change.
- Reduced control plane cost and shared free pool buffer
- Priority queues and flow control
- No loss in debuggability experience with objects being namespaced

❌

- No native support for nodes to namespaces. Custom controllers to
  - ensure the correctness of the bindings
  - aggregate resource quota
  - drain capacity
- Scheduler is not isolated
- Overhead in managing bootstrap cluster

# Problem-1: Scheduler Isolation

## Problem:

No native support for scheduler level isolation predominantly

- ○ No guarantees around scheduling latency per tenant
- ○ No per tenant pod queues
- ○ No native support for matching pods to tenant nodes

Options:

- Active - Active Scheduler :
  - Each scheduler partition to operate on a specific tenant nodepool
  - Actively exploring this option
- Per Tenant Queues:
  - Configure separate scheduling queues per tenant
- Open to new ideas

# Problem-2: Bootstrap Cluster

Bootstrap Cluster:

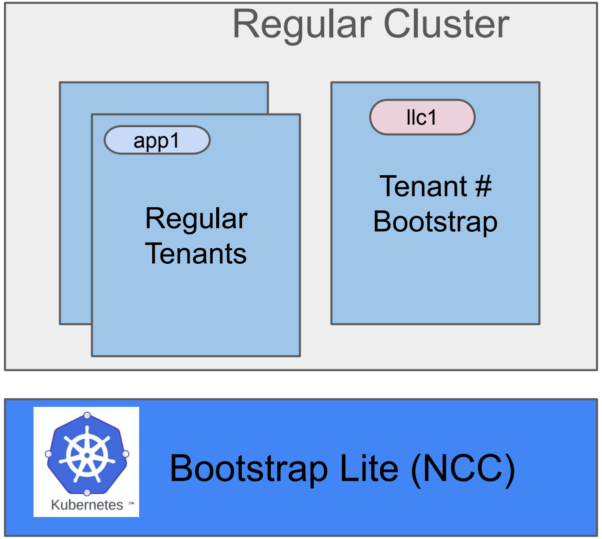
Brings up the low level infrastructure components necessary for the regular cluster to operate.

Components:

- Native Compute Components (NCC)
  - ApiServer
  - Scheduler
  - Controller Manager
- Other low level infra components (LLC):
  - Custom Controllers
  - Metrics infra
  - Logging infra
  - ….

Problem: Need to manage an additional bootstrap cluster zonally

Proposal



Acknowledgements

- Container Platform
- Service Lifecycle Team
- Host Lifecycle Team
- Security Team
- Observability Team

Q & A