Realize your vision

Things to consider to operate a Multi-Tenant Kubernetes Cluster

Multi-Tenant Kubernetes Cluster를 운영하기 위해 고려할 사항

2020.06.10

SAMSUNG SDS

Agenda

I. State of Multi-Tenancy in Kubernetes

II. Multi-Tenancy approach

1. Namespace per Tenant
2. Node per Tenant
3. Cluster per Tenant

III. Community's Approach

1. Hierarchical Namespace Controller
2. Virtual Cluster

IV. What's Next

1. Multi-Cluster Architecture

Multi-tenancy?

Multi-tenancy is an architecture paradigm where multiple customers are supported with single instance of application.

Benefit:

- Better use of resources
- Lower costs

Drawback:

- Possible security risks and compliance issues
- The "noisy neighbor" effect

Multi-tenancy in Kubernetes?

Share Kubernetes environment between multiple tenants

What degree of isolation do you need?

Kubernetes has a various layers of resources (cluster, node, namespace, pod and container), so isolation can be achieved at multiple levels.

Models of Multitenancy

"Soft" Multitenancy

→ Ex. Multiple teams within the same company sharing k8s environment

"Hard" Multitenancy

→ Ex. Multiple independent company in same k8s environment

Kubernetes Multi-Tenancy Draft Proposal

Multi-tenancy in Kubernetes?

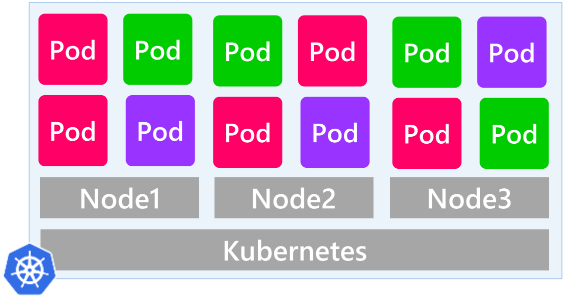
Kubernetes cannot guarantee perfectly secure isolation between tenants, it does offer features that may be sufficient for specific use cases.

Access Control: RBAC, Network policy, Admission Control, PSP

Scheduling: Resource Quota, Limit Range, Pod Affinity

Multi-Tenancy approaches in Kubernetes

Namespace per Tenant

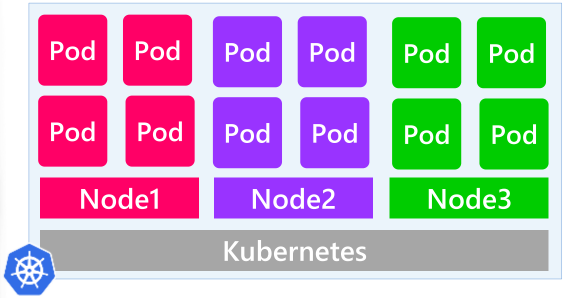


🟥 Tenant 1 Namespace

🟪 Tenant 2 Namespace

🟩 Tenant 3 Namespace

Nodes per Tenant

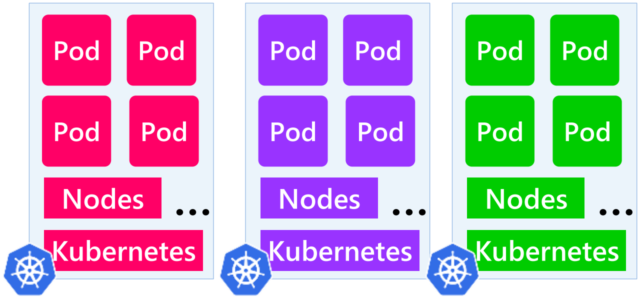


🟥 Tenant 1 Node

🟪 Tenant 2 Node

🟩 Tenant 3 Node

Cluster per Tenant



🟥 Tenant 1 Cluster

🟪 Tenant 2 Cluster

🟩 Tenant 3 Cluster

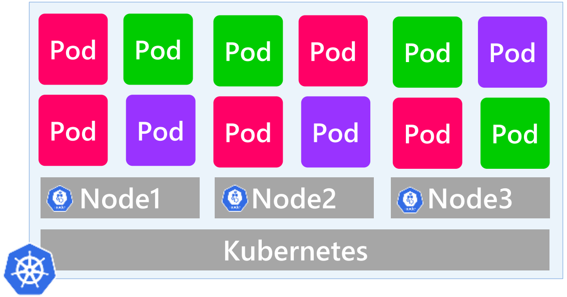
Less Isolation ⟷ More Isolation

High Utilization ⟷ Low Utilization

Namespace Per Tenant

Namespace per Tenant

Namespace Per Tenant



Pod Pod Pod Pod Pod Pod

Pod Pod Pod Pod Pod Pod

Node1 Node2 Node3

Kubernetes

🟥 Tenant 1 Namespace

🟪 Tenant 2 Namespace

🟩 Tenant 3 Namespace

Shared Resources

Cluster

etcd sched api c-m

Node

node

Individual Resources

Namespace

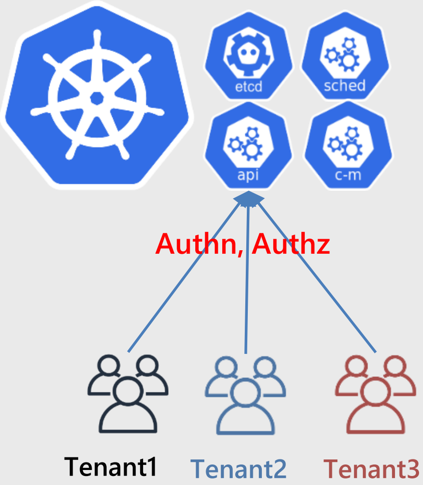
ns

Namespace per Tenant

For shared resources we need proper Isolation

Shared Resource

Cluster / Control Plane



Authn/Authz of Control Plane

1) Authn

- Kind

user
group
sa

- Method

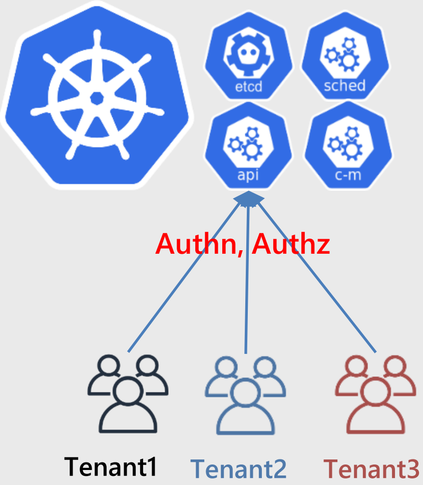
Certificate
(/CN, /O)
OIDC
Integration
Service Account
(Pods)

Namespace per Tenant

For shared resources we need proper Isolation

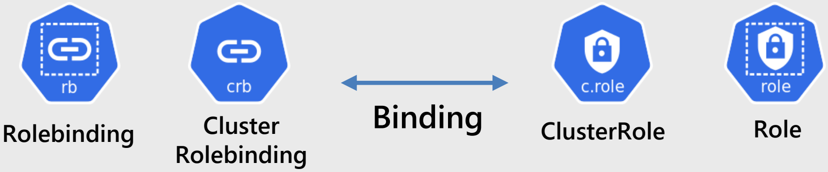
Shared Resource

Cluster / Control Plane



Authn/Authz of Control Plane

2) Authz – RBAC



```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-pods
  namespace: default
subjects:
- kind: User
  name: jane
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role #this must be Role or ClusterRole
  name: pod-reader
  apiGroup: rbac.authorization.k8s.iov
```

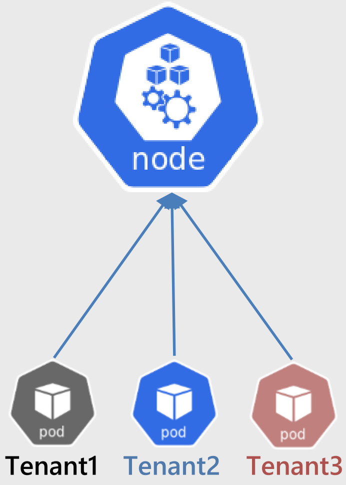
```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

Namespace per Tenant

Proper Isolation is needed for Shared Resources

Shared Resource

Node



Resource Isolation (Noisy Neighbor)

1) In-Cluster Isolation

quota limits

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources
spec:
  hard:
    requests.cpu: "1"
    requests.memory: 1Gi
    limits.cpu: "2"
    limits.memory: 2Gi
    requests.nvidia.com/gpu: 4
```

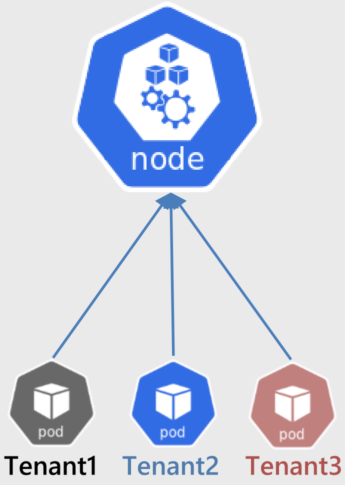
```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: cpu-min-max-demo-lr
spec:
  limits:
  - max:
      cpu: "800m"
    min:
      cpu: "200m"
    type: Container
```

Namespace per Tenant

Proper Isolation is needed for Shared Resources

Shared Resource

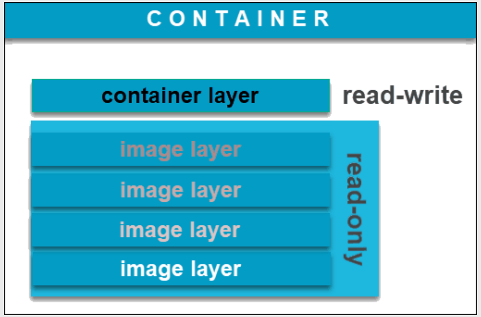
Node



Resource Isolation (Noisy Neighbor)

2) External Isolation

docker cri-o containerd



https://phoenixnap.com/kb/docker-image-vs-container

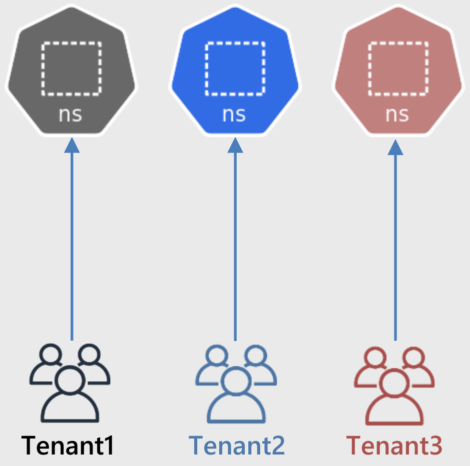
- Read-Write Layer & Log & Docker Image
  => Docker Storage (/var/lib/docker)
- Tenants Share Docker Storage Consumption
- Docker Storage & Log Options
  (Different per Runtime & Storage)

Namespace per Tenant

Isolation between Individual Resources is Needed

Individual Resource

Namespace



Isolation between Individual Resources

1) Resource Isolation

- Network
- CPU, Memory

netpol
quota
limits

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  policyTypes:
  - Ingress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          project: myproject
  podSelector: {}
```

2) User Isolation

rb
crb
c.role
role

Namespace per Tenant

Our Approach

1) Namespace Controller

- Bootstrapping Namespaces
  - Create NetworkPolicy (Default Deny, Same Namespace, kube-system namespace)
  - Create Rolebinding (admin, edit, view group)
  - Create ResourceQuota

2) User Management Through OIDC Integration

- Dynamic token per Tenant
- Kubectl Login Plugin

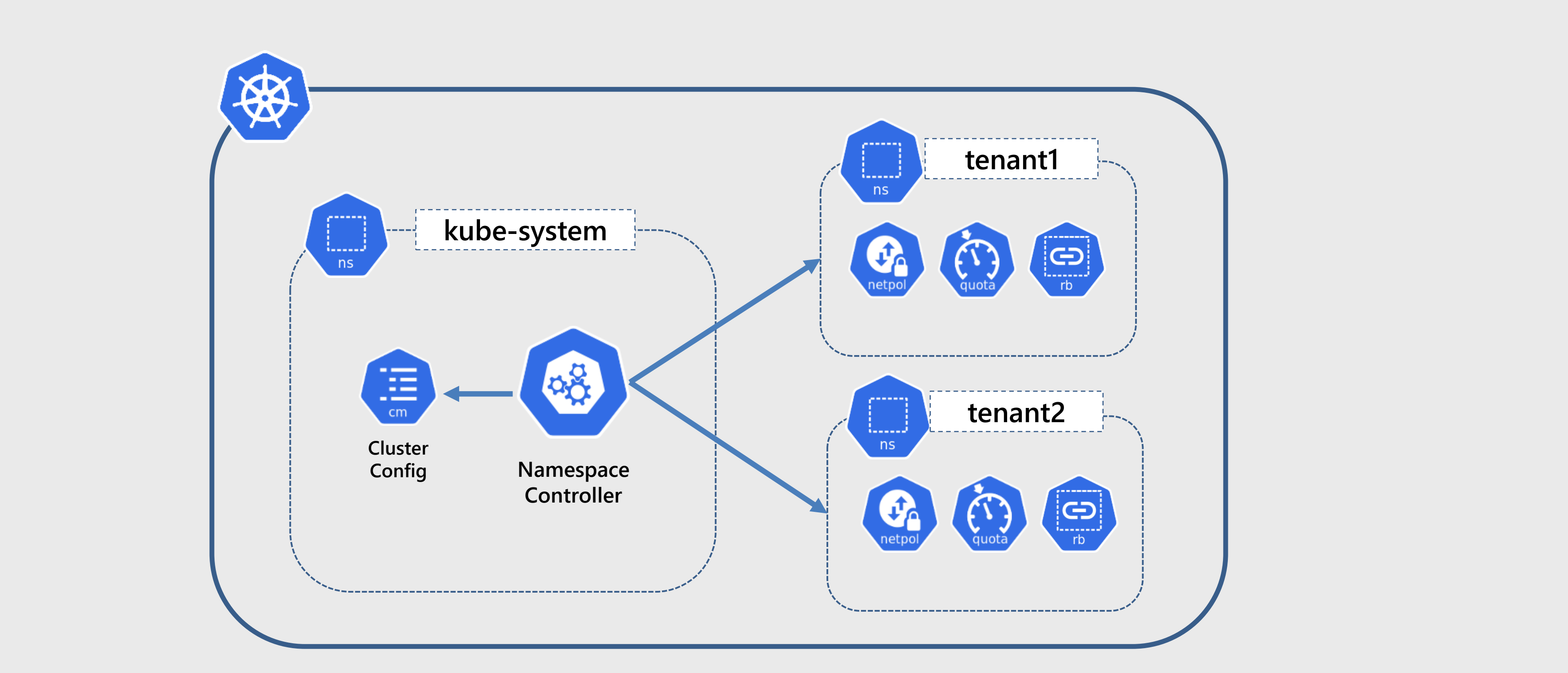
3) User Activity Insight

- Audit Log
- Log visible per Tenant

4) Docker Storage Isolation

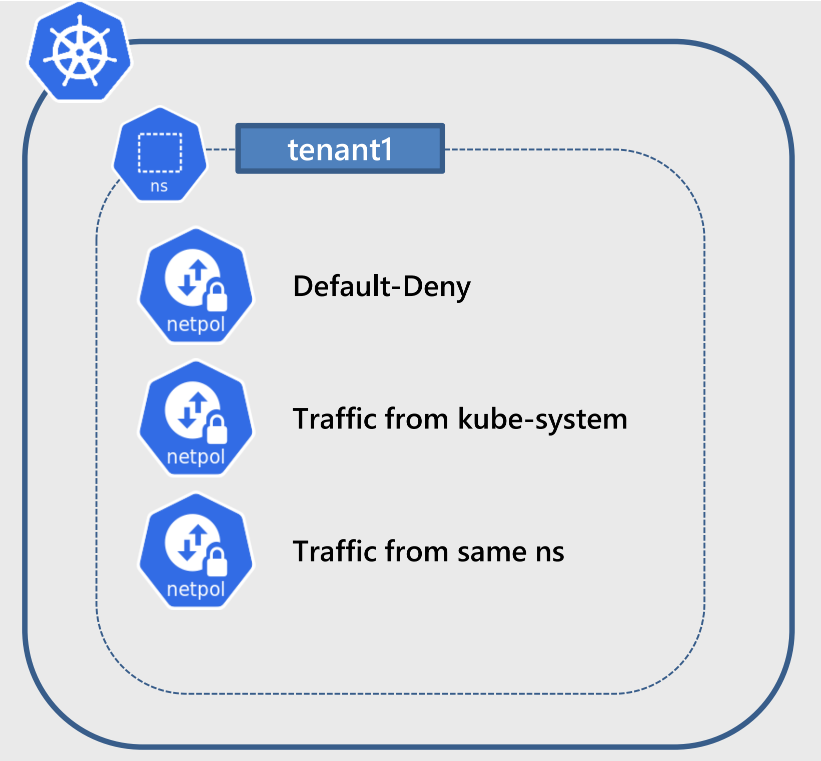
Namespace per Tenant

1) Namespace Controller – Object Bootstrapping



Namespace per Tenant

1) Namespace Controller – Object Bootstrapping



tenant1

Default-Deny

Traffic from kube-system

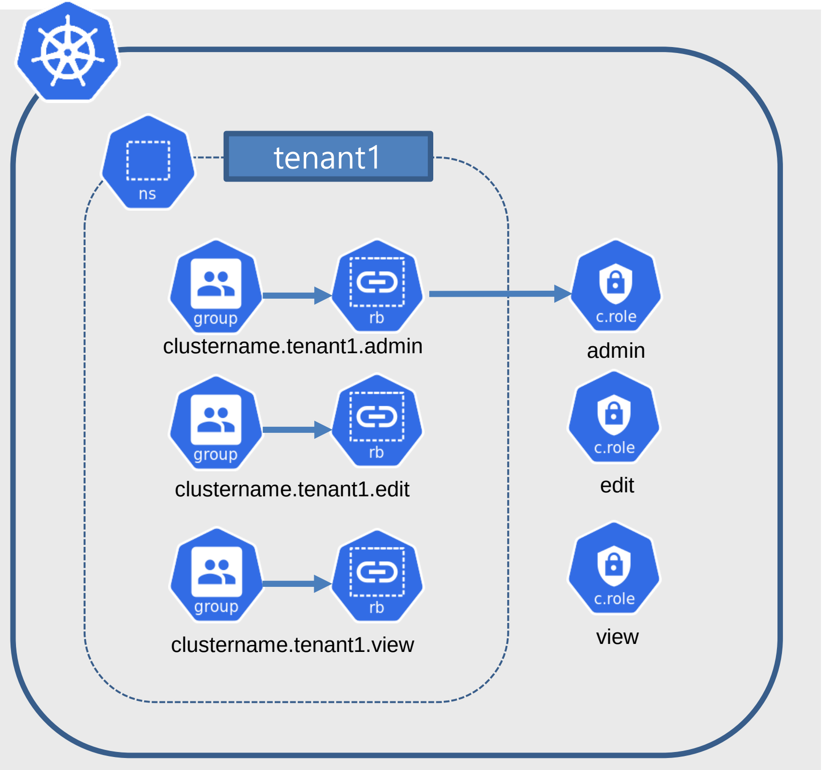
Traffic from same ns

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny
  namespace: tenant1
spec:
  podSelector: {}
  policyTypes:
  - Ingress
---

apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-kube-system-namespace
  namespace: tenant1
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/namespace: kube-system
  podSelector: {}
  policyTypes:
  - Ingress
```

Namespace per Tenant

1) Namespace Controller – Object Bootstrapping



```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: tenant1.admin.rolebinding
  namespace: tenant1
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: admin
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: clustername.tenant1.admin
  namespace: tenant1
```

Cluster-Wide Admin Group
- clustername.admin
- clustername.edit
- clustername.view

Namespace per Tenant

2) User Management Through OIDC Integration

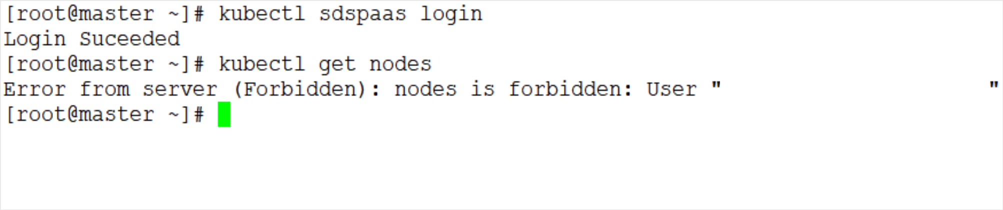
api

```
- --oidc-client-id={CLIENT_ID}
- --oidc-groups-claim=groups # Claim used for identifying Group
- --oidc-issuer-url={OIDC_URL}
- --oidc-username-claim=user_email # Claim used for identifying User
- --oidc-username-prefix=-
```

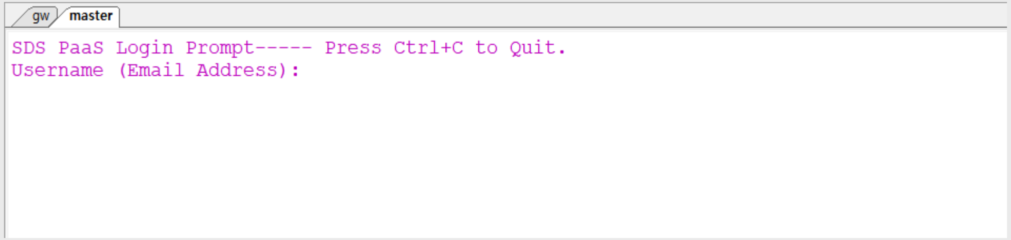
Namespace per Tenant

2) User Management Through OIDC Integration

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://[REDACTED]:6443
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: kubernetes-admin
  name: kubernetes-admin@kubernetes
current-context: kubernetes-admin@kubernetes
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
    exec:
      command: "kubectl-sdspaas"
      apiVersion: "client.authentication.k8s.io/v1beta1"
      env:
      - name: "KUBECTL_EXEC"
        value: "true"
      args:
      - "login"
```



```
[root@master ~]# kubectl sdspaas login
Login Suceeded
[root@master ~]# kubectl get nodes
Error from server (Forbidden): nodes is forbidden: User "                          "
[root@master ~]#
```



```
SDS PaaS Login Prompt----- Press Ctrl+C to Quit.
Username (Email Address):
```

Namespace per Tenant

2) User Management Through OIDC Integration – Example Token

```
{
  "at_hash": "Ai_QMB62Ypk-3cb8__Mu-w",
  "sub": “tenant1@samsung.com",
  "user_name": "parkhsol",
  "iss": “REDACTED",
  "language": {
    "country": "South Korea (KR)",
    "language_tag": "ko-KR",
    "language": "한국어"
  },
  "preferred_username": " tenant1@samsung.com ",
  "company": "삼성SDS",
  "state": "",
  "exp": 1591409174,
  "user_email": “tenant1@samsung.com",
  "groups": [
    “clustername.tenant1.admin"
  ],
  "nonce": "",
  "user_uid": 1343
}
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: tenant1.admin.rolebinding
  namespace: tenant1
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: admin
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: clustername.tenant1.admin
  namespace: tenant1
```

Namespace per Tenant

3) User Activity Insight

```
apiVersion: audit.k8s.io/v1beta1
kind: Policy
rules:
  - level: Request
    users:
      - system:serviceaccount:kube-system:helm
## Don't log system components events
  - level: None
    userGroups:
      - system:nodes
      - system:serviceaccounts:kube-system
  - level: None
    users:
      - system:apiserver
      - system:kube-controller-manager
      - system:kube-scheduler
      - system:kube-proxy
  - level: None
    users: ["system:unsecured"]
    namespaces: ["kube-system"]
    verbs: ["get"]
    resources:
      - group: "" # core
        resources: ["configmaps"]
  - level: None
    users: ["kubelet"] # legacy kubelet identity
    verbs: ["get"]
    resources:
      - group: "" # core
        resources: ["nodes", "nodes/status"]
  # Don't log these read-only URLs.
  - level: None
    nonResourceURLs:
      - /healthz*
      - /version
      - /swagger*
   ---------- REDACTED ----------
```

```
{"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Request","auditID":"03ffe866-8286-46a9-8d68-b8ce6773d5ef","stage":"ResponseComplete","requestURI":"/api/v1/namespaces","verb":"list","user":{"username":“tenant1@Samsung.com","groups":["system:authenticated“,”clustername.admin”]},"sourceIPs":[“REDACTED"],"userAgent":"dashboard/v2.0.0-beta3","objectRef":{"resource":"namespaces","apiVersion":"v1"},"responseStatus":{"metadata":{},"code":200},"requestReceivedTimestamp":"2020-06-08T02:59:24.110091Z","stageTimestamp":"2020-06-08T02:59:24.111509Z","annotations":{"authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":"RBAC: allowed by ClusterRoleBinding \"admin\" of ClusterRole \"admin\" to User \“tenant1@samsung.com\""}}
```

Namespace per Tenant

4) Docker Storage Isolation

daemon.json

```
{
  "selinux-enabled": true,
  "storage-driver": "overlay2",
  "storage-opts": [
     "overlay2.override_kernel_check=true",
     "overlay2.size=10G”                 ## Limit Write Layer to 10G
   ],
  "log-driver": "json-file",
  "log-opts": {
     "max-size": "10m”                   ## Limit Log Size to 10MB
   }
}
```

Namespace per Tenant

Limitations

1) Resources that are not isolated by Kubernetes
-> DiskIO, Network Bandwidth

2) Per Node Configuration
-> Non-namepaced OS configuration
. Namespaced sysctls: kernel.shm*, kernel.msg*, kernel.sem, fs.mqueue.*, net.*(with exception)
-> Elasticsearch needs "vm.max_map_count = 262144"

3) Egress IP
-> When traffic is sent from a Node to external Service "Node IP" is used for all services
-> Hard to integrate with legacy Firewall

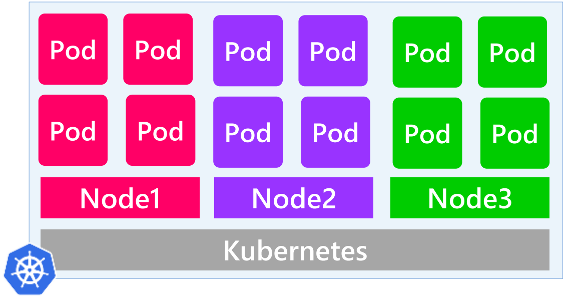
4) How to provide Kubernetes Control Plane Option(s) per Tenant

5) Managing Authz on Cluster Wide Objects
-> Namespace, Node, PV(Solvable through StorageClass) and etc

Node(s) Per Tenant

Nodes per Tenant

Nodes Per Tenant



Pod Pod Pod Pod Pod Pod

Pod Pod Pod Pod Pod Pod

Node1 Node2 Node3

Kubernetes

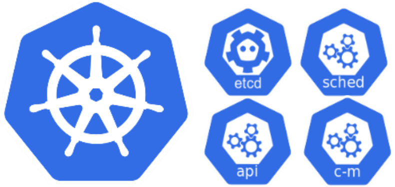
🟥 Tenant 1 Node

🟪 Tenant 2 Node

🟩 Tenant 3 Node

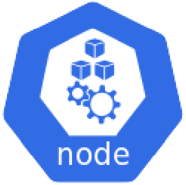
Shared Resources

Cluster



Individual Resources

Node

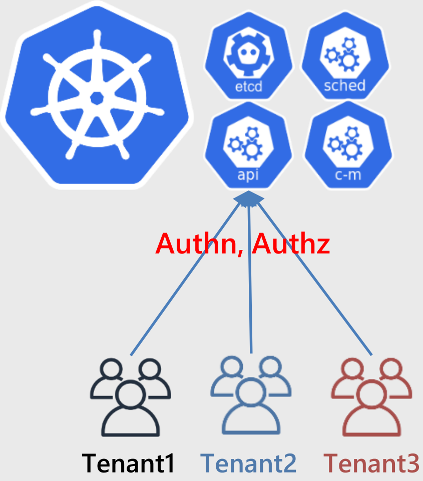


Nodes per Tenant

Proper Isolation is needed for Shared Resources

Shared Resource

Cluster / Control Plane



Authn/Authz of Control Plane

1) Authn

Certificate (/CN, /O)

OIDC Integration

Service Account (Pods)

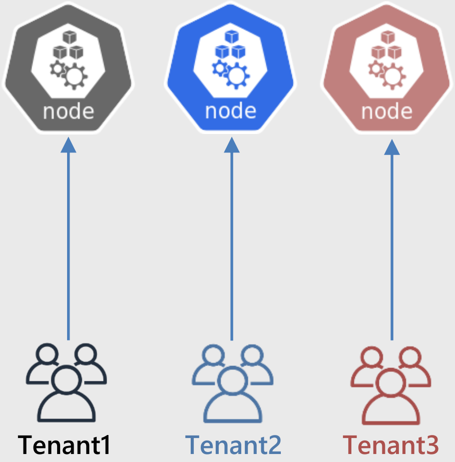
2) Authz – RBAC

Nodes per Tenant

Scheduling to the Appropriate Tenant Resource

Individual Resource

Node



Workload Scheduling

1) Node Label – NodeSelector, Node(Anti)Affinity

- Selective Approach

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  nodeSelector:
    disktype: ssd
```

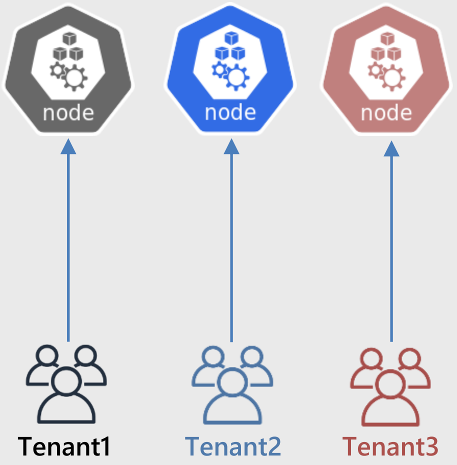
```
nodeAffinity:
     requiredDuringSchedulingIgnoredDuringExecution:
       nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/e2e-az-name
          operator: In
          values:
          - e2e-az1
          - e2e-az2
     preferredDuringSchedulingIgnoredDuringExecution:
    - weight: 1
      preference:
        matchExpressions:
        - key: another-node-label-key
          operator: In
          values:
          - another-node-label-value
```

Nodes per Tenant

Scheduling to the Appropriate Tenant Resource

Individual Resource

Node



Workload Scheduling

2) Node Taint – Toleration

- Preventive Approach

```
apiVersion: v1
kind: Node
spec:
  taints:
  - effect: NoSchedule
    key: example-key
```

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  tolerations:
  - key: "example-key"
    operator: "Exists"
    effect: "NoSchedule"
```

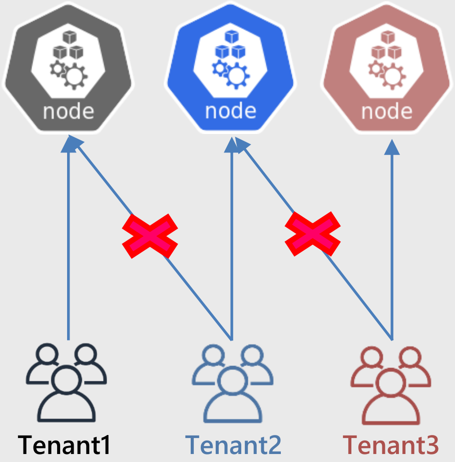
Nodes per Tenant

Enforce Workload to the Appropriate Tenant Resource

Individual Resource

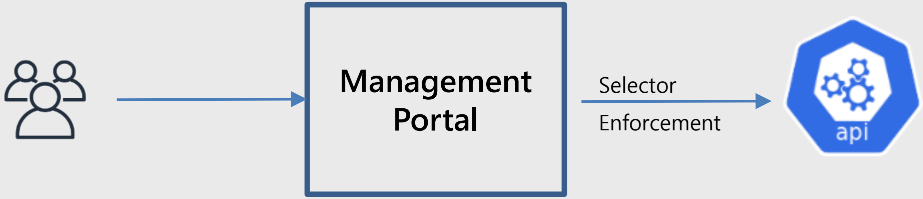
Workload Scheduling Enforcement

Node



1) Client Side Enforcement

Management Portal

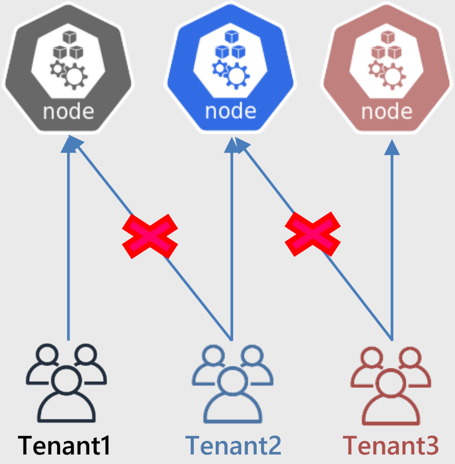


Nodes per Tenant

Enforce Workload to the Appropriate Tenant Resource

Individual Resource

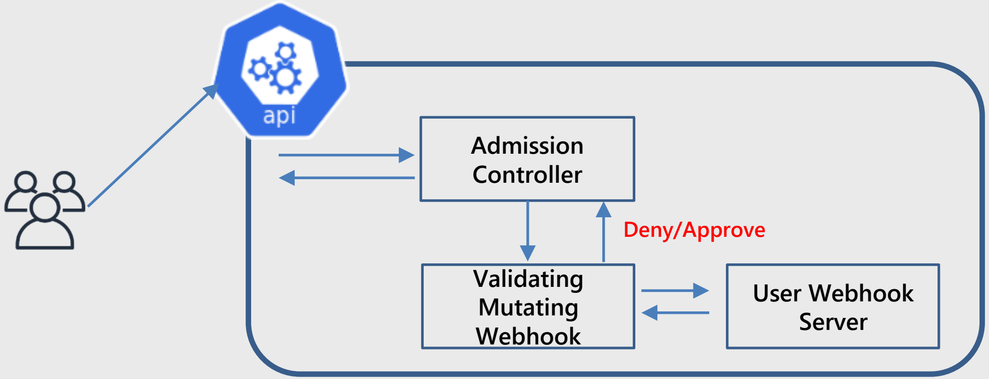
Node



Workload Scheduling Enforcement

2) Server Side Enforcement

Dynamic Admission Controller (Validating Webhook, Mutating Webhook)

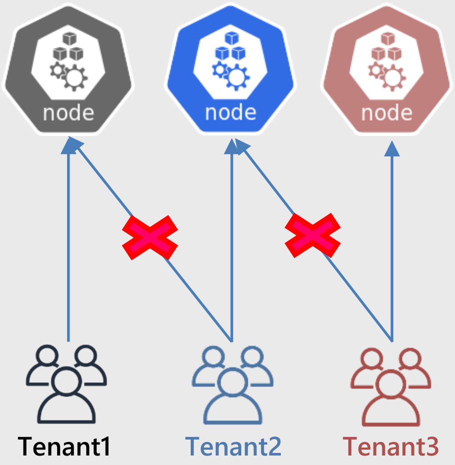


Nodes per Tenant

Enforce Workload to the Appropriate Tenant Resource

Individual Resource

Node



Workload Scheduling Enforcement

2) Server Side Enforcement

Custom Tenant Aware Scheduler

```
apiVersion: v1
kind: Pod
metadata:
  name: annotation-default-scheduler
  labels:
    name: multischeduler-example
spec:
  schedulerName: tenant-scheduler
  containers:
  - name: pod-with-default-annotation-container
    image: k8s.gcr.io/pause:2.0
```

Nodes per Tenant

Our Approach

1) Utilize Both Node Label/Taint

- Taint Node that Cluster common services resides (Ingress, Logging, Monitoring and etc.)
- Use Node Label to isolate Tenants

2) **NodeSelector** Validation through OPA

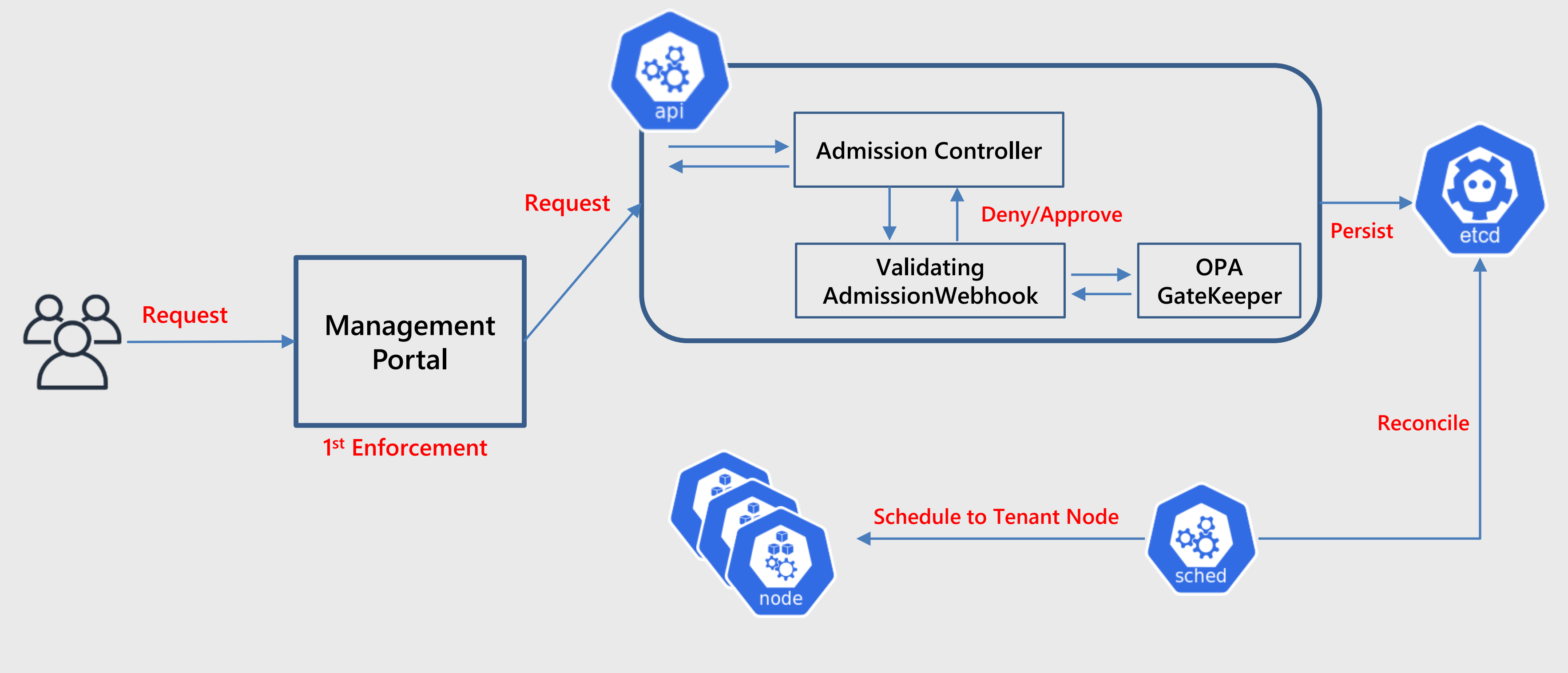
- Validate API Requests has the appropriate NodeSelector through OPA
- What is OPA? (General Purpose Policy Engine)

"The Open Policy Agent (OPA, pronounced "oh-pa") is an open source, general-purpose policy engine that unifies policy enforcement across the stack. OPA provides a high-level declarative language that let's you specify policy as code and simple APIs to offload policy decision-making from your software. You can use OPA to enforce policies in microservices, Kubernetes, CI/CD pipelines, API gateways, and more."

https://kubernetes.io/blog/2019/08/06/opa-gatekeeper-policy-and-governance-for-kubernetes/
https://github.com/open-policy-agent/opa

Nodes per Tenant

Workload Provision Flow



Nodes per Tenant

Limitations

1) ~~Resources that are not isolated by Kubernetes~~

2) ~~Per Node Configuration~~

3) ~~Egress IP~~

4) How to provide Kubernetes Control Plane Option(s) per Tenant

5) Managing Authz on Cluster Wide Objects
-> Namespace, Node, PV(Solvable through StorageClass) and etc

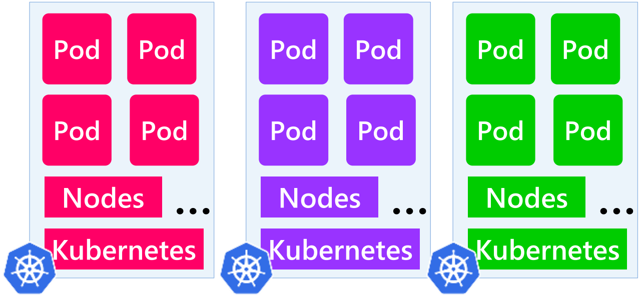
6) Lower Resource Utilization

7) Long Lead Time for Tenants
New Tenant -> Create Node -> Node Join -> Add Label/Policy -> Service Provisioning

Cluster Per Tenant

Cluster per Tenant

Cluster Per Tenant



🟥 Tenant 1 Cluster

🟪 Tenant 2 Cluster

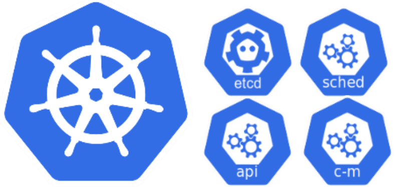
🟩 Tenant 3 Cluster

Shared Resources

None

Individual Resources

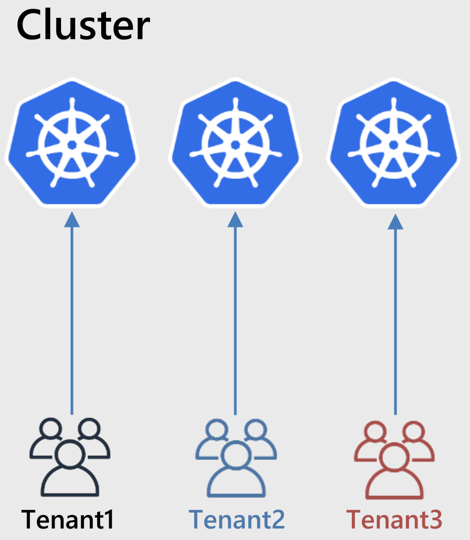
Cluster



Cluster per Tenant

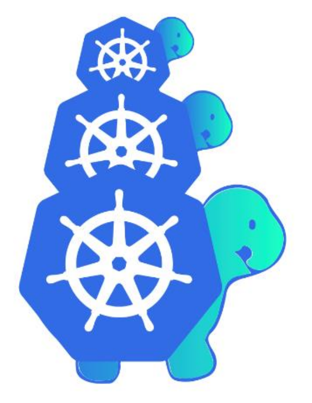
Provisioning/Managing Tenant Resource

Individual Resource



Cluster Provisioning/Management

1) Cluster API



"The Cluster API is a Kubernetes project to bring declarative, Kubernetes-style APIs to cluster creation, configuration, and management. It provides optional, additive functionality on top of core Kubernetes."
(https://github.com/kubernetes-sigs/cluster-api)

Cluster per Tenant

Our Approach

1) Cluster API Based Our Own Implementation

- Integration with ClusterAPI Vsphere, Azure, AWS Machine

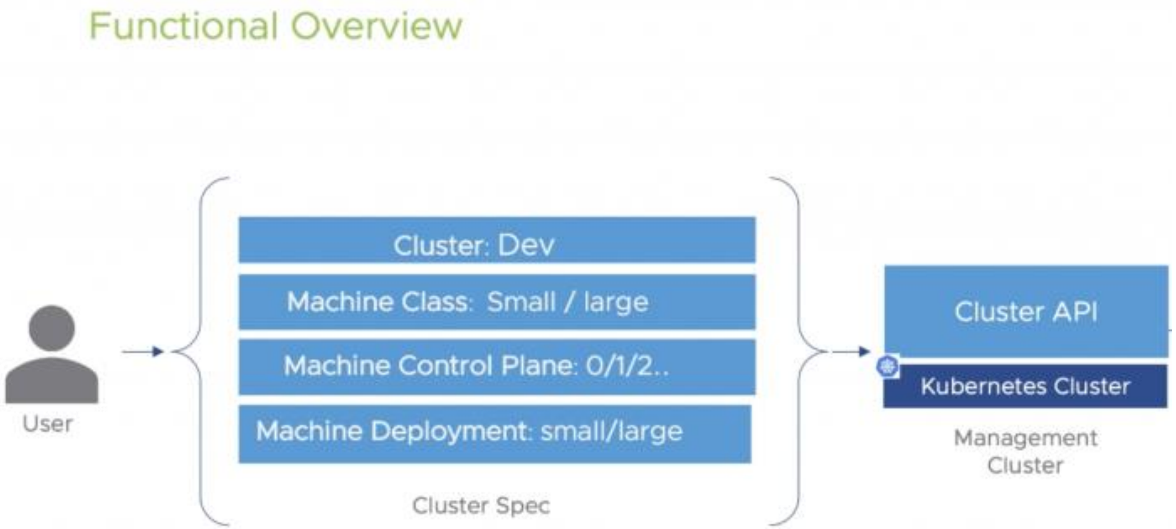


image source : https://blogs.vmware.com/cloudnative/2019/05/14/cluster-api-kubernetes-lifecycle-management/

Cluster per Tenant

Limitations

1) ~~Resources that are not isolated by Kubernetes~~

2) ~~Per Node Configuration~~

3) ~~Egress IP~~

4) ~~How to provide Kubernetes Control Plane Option(s) per Tenant~~

5) ~~Managing Authz on Cluster Wide Objects~~

6) Even Lower Resource Utilization

7) Even Longer Lead Time for Tenants
New Tenant -> Create Cluster -> Create Node -> Node Join -> Service Provisioning

8) Managing Multiple Clusters is a big struggle
100 Clusters -> 100 Endpoints -> 100 kubeconfigs

Communities Approach

Community's Approach

Multi-Tenancy Working Group

A working place for multi-tenancy related proposals and prototypes.
(https://github.com/kubernetes-sigs/multi-tenancy)

1) Hierarchical Namespace Controller (HNC)

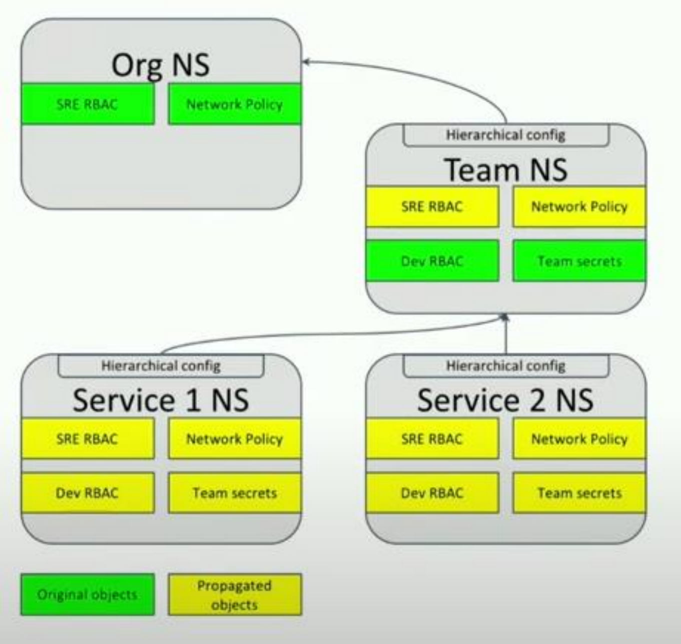
- Similar to Namespace Per Tenant Approach
- Brings Hierarchy to Namespaces
- Focuses on Namespace Bootstrapping

2) Virtual Cluster

- Control Plane Per Tenant
- ETCD, Apiserver, Controller per Tenant

Community’s Approach

1) Hierarchical Namespace Controller

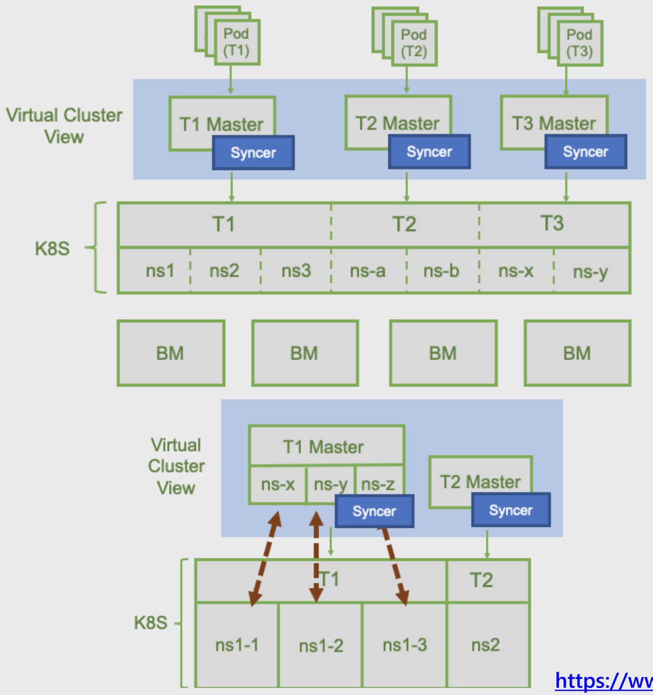


https://www.youtube.com/watch?v=PA101KUDusY

- Similar to Namespace Per Tenant Approach
- Brings Hierarchy to Namespaces
- Focuses on Namespace Bootstrapping

Community’s Approach

2) Virtual Cluster



Approach:

- Virtual Control Plane per Tenant (ETCD, API, Controller)
- Provide Control Plane Isolation
- Pods are synced through syncer

Limitation:

- Components need to be Tenant Aware
  : Kubelet, CNI, Kube-Proxy, Kube-Dns
- DaemonSets are not supported

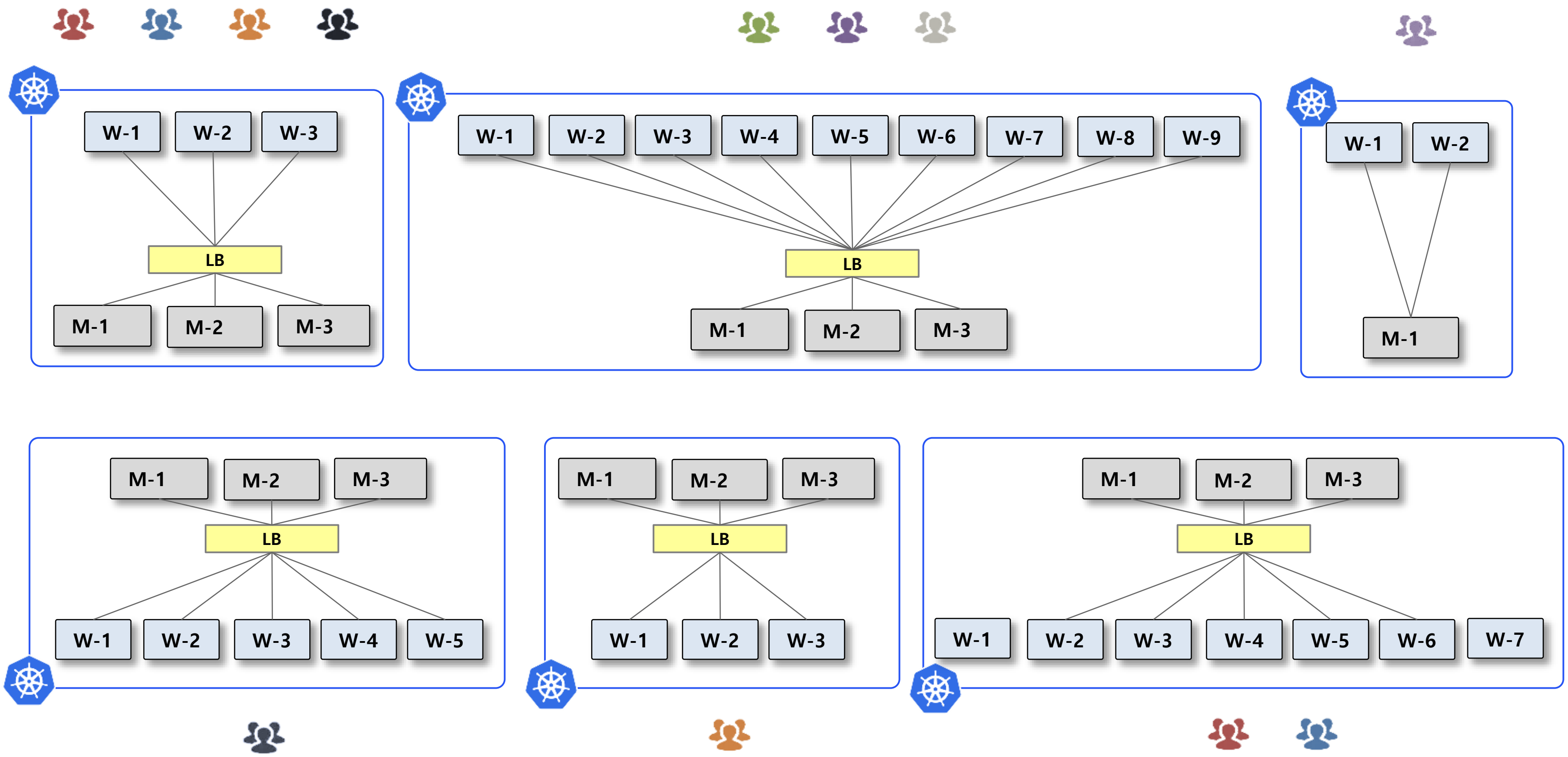
https://www.cncf.io/blog/2019/06/20/virtual-cluster-extending-namespace-based-multi-tenancy-with-a-cluster-view/

Wrap Up

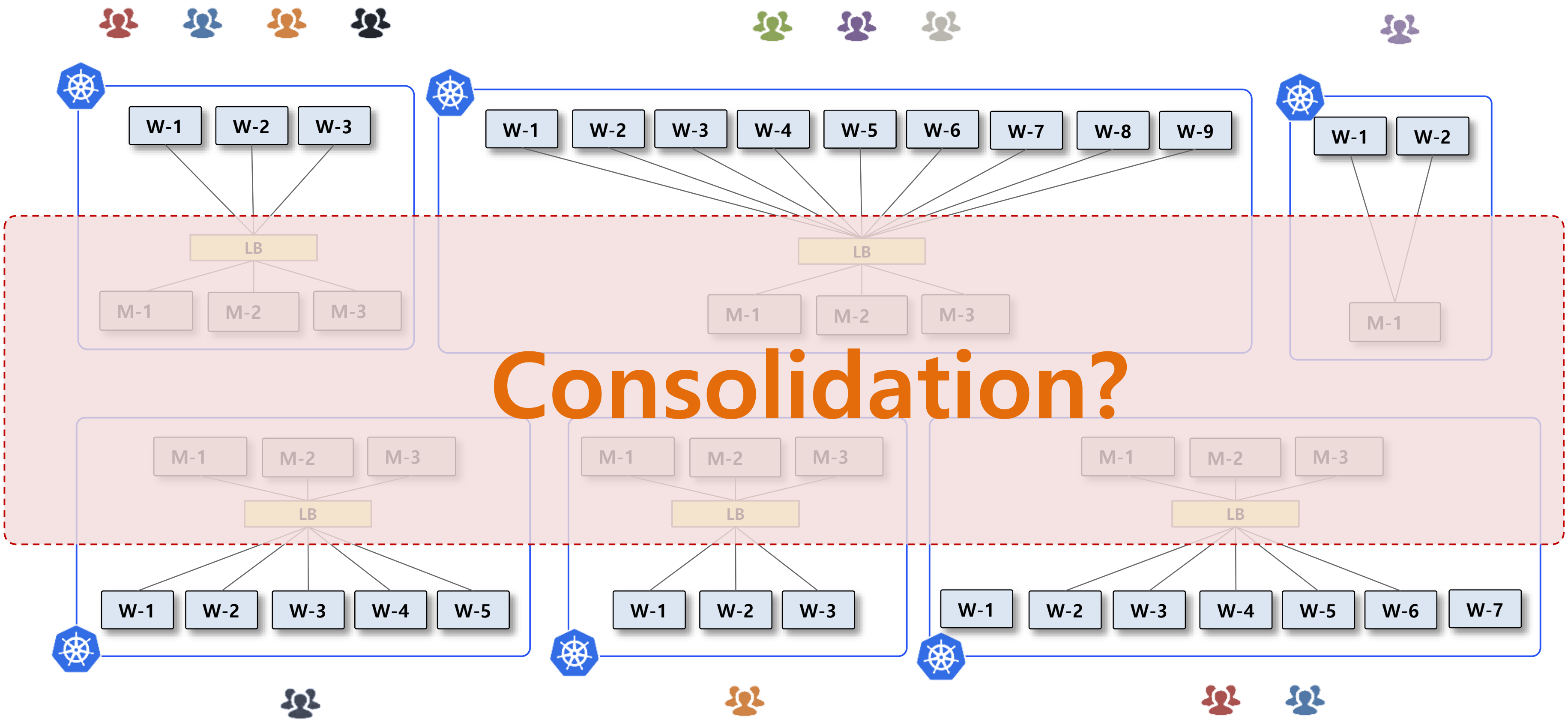
	Namespace Per Tenant	Node Per Tenant	Cluster Per Tenant
Resource Utilization	• High	• Medium	• Low - Medium
Tenant Isolation	• Low – Medium	• Medium - High	• High
Shared/Individual Resources	• Shared – Cluster, Node • Individual – Namespace	• Shared – Cluster • Individual – Node	• Individual - Cluster
Usecase	• Security Requirements are less tight • Resource Utilization is the top priority • i.e Dev Environment	• Tenants with specific node requirements - GPU Nodes • Bring your own Node	• Strict Isolation is needed - Legal Requirements - Different Data Centers

What’s Next??

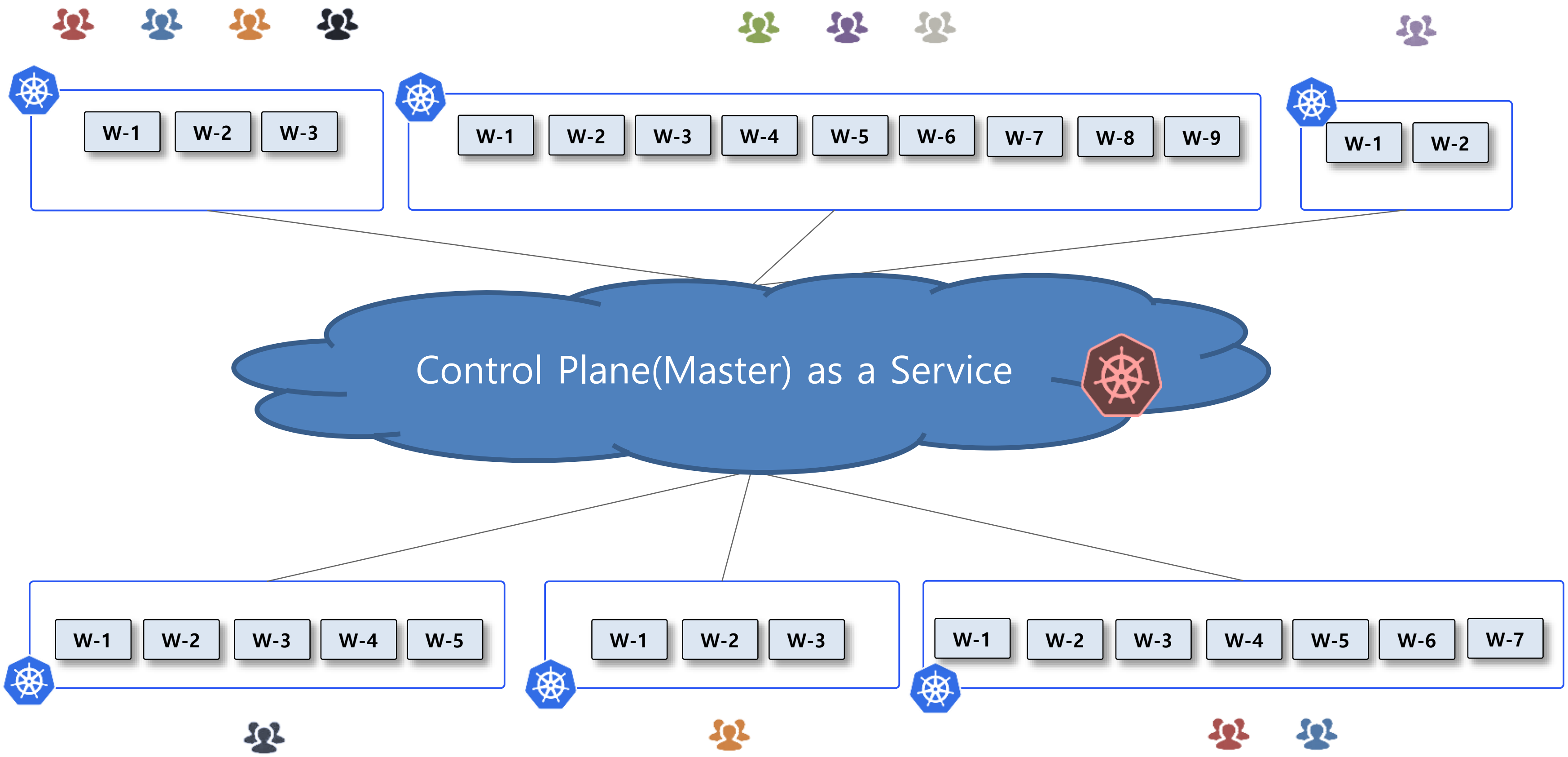
We need Kubernetes Cluster !



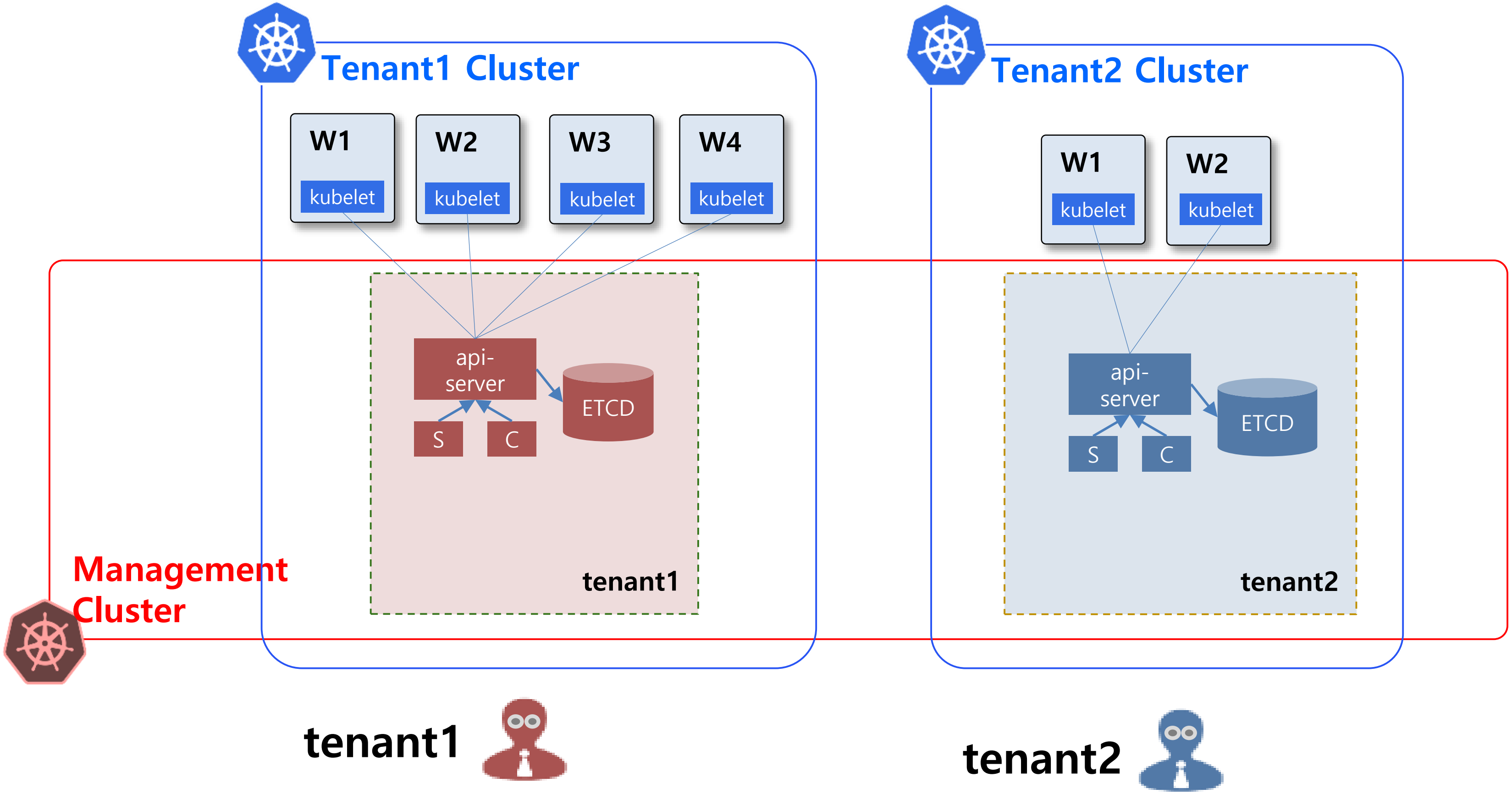
We need Kubernetes Cluster !



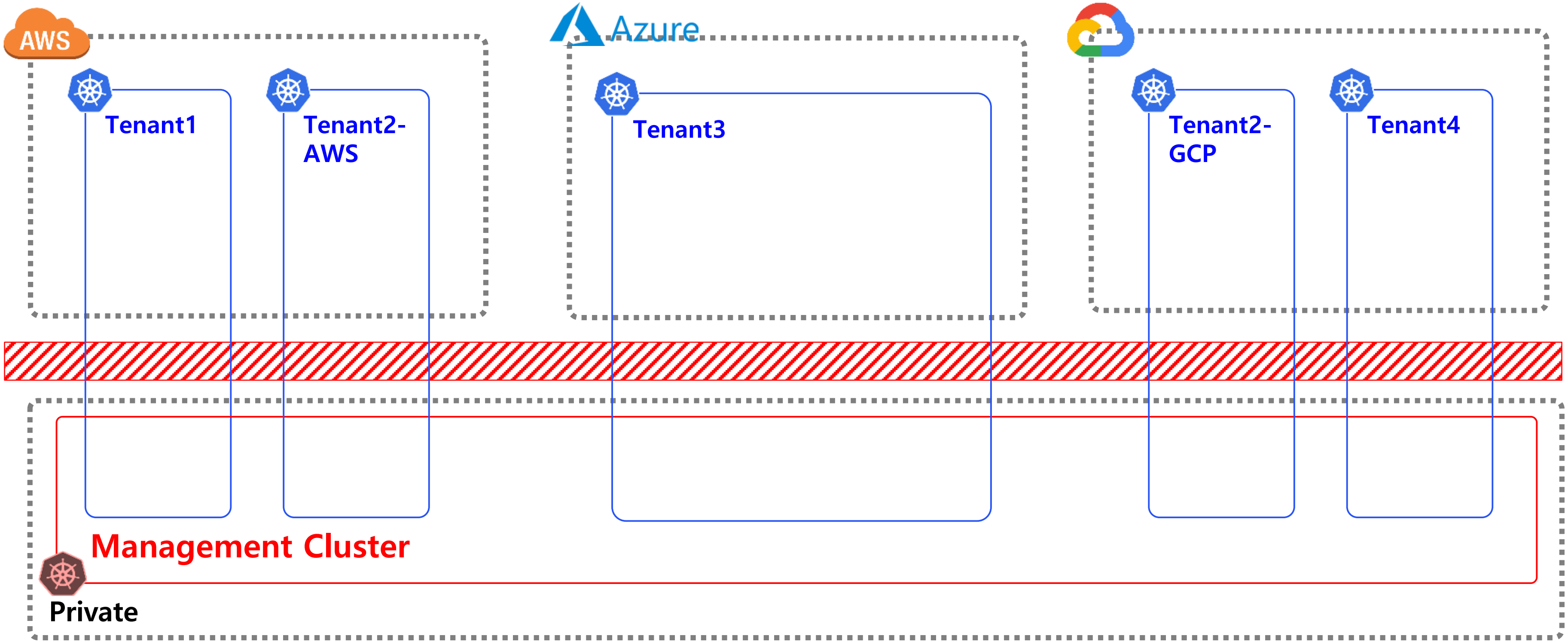
Control Plane as a Service



Control Plane as a Service



Use Case – Hybrid Cloud



Q&A

SAMSUNG SDS

Realize your vision

www.samsungsds.com