://localhost

Multitenancy in Kubernetes

Multi-tenancy in Kubernetes is sharing a single cluster among multiple users, teams, or applications while maintaining isolation and security.

Why Multi-tenancy?

I believe we all had that moment we realised it would be better to share a single cluster with isolation & limits for each tenant

- ❏ Managing single cluster can consolidate infrastructure and administrative overhead, leading to significant cost savings.
- ❏ Managing multiple development teams on a single cluster
- ❏ Supporting various projects on a single cluster
- ❏ Balancing resource allocation in a growing organization

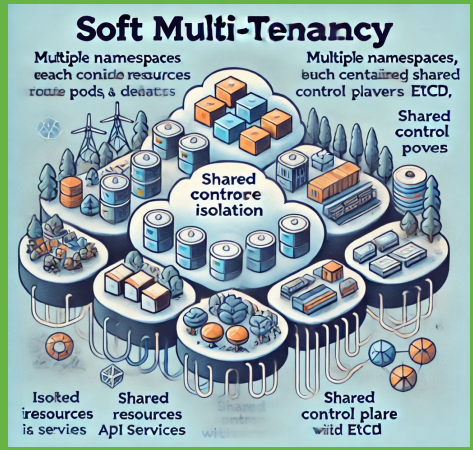
Types Of Multi Tenancy In Kubernetes

Soft Multi-Tenancy

Soft' multi-tenancy, provides logical separation with reduced isolation

Benefits of Hard Multi Tenancy

- Tenants share the same Kubernetes cluster
- Uses namespaces, network policies, and other Kubernetes features to logically separate workloads
- Resources can be shared more efficiently between tenants
- Generally more cost-effective but may have potential security concerns
- Requires careful configuration to ensure proper isolation and resource allocation



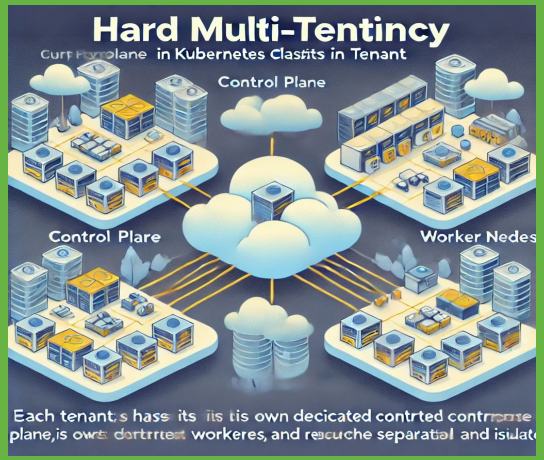
Types Of Multi Tenancy In Kubernetes

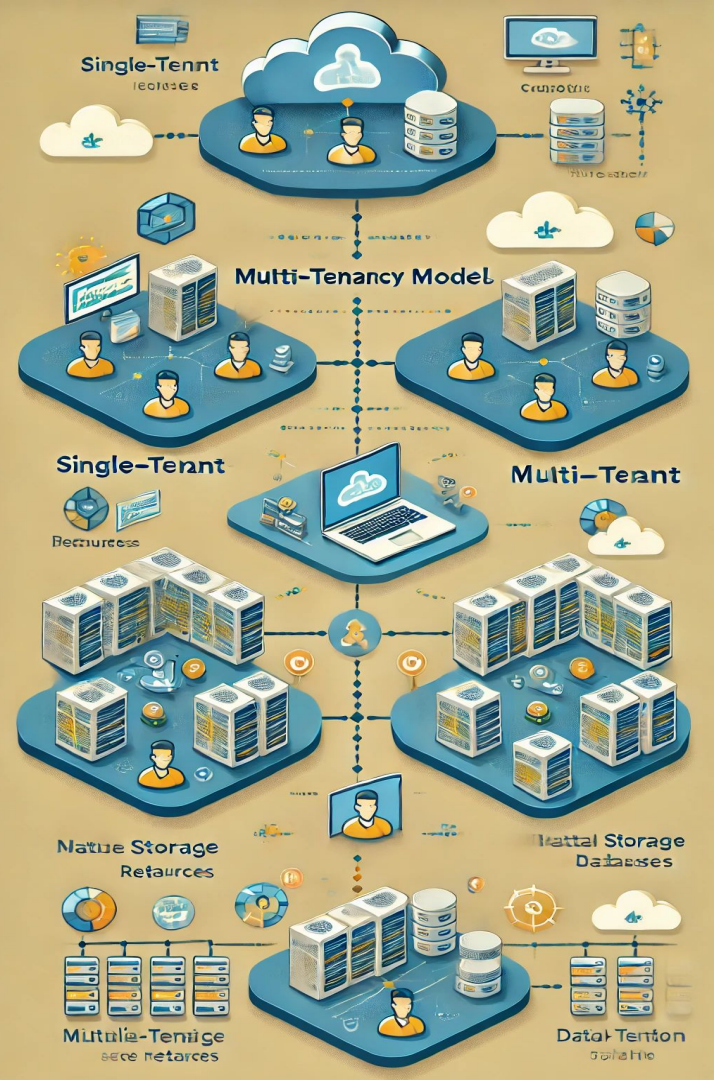
Hard Multi-Tenancy

Hard multi-tenancy, Implies stronger isolation between tenants

Benefits of Hard Multi Tenancy

- Provides stronger isolation between tenants
- Each tenant typically has their own separate Kubernetes cluster
- Offers better security and compliance as tenants are completely isolated
- More resource-intensive and potentially more expensive to maintain
- Resources are not shared between tenants
- Tenants cannot access or affect each other's resources.

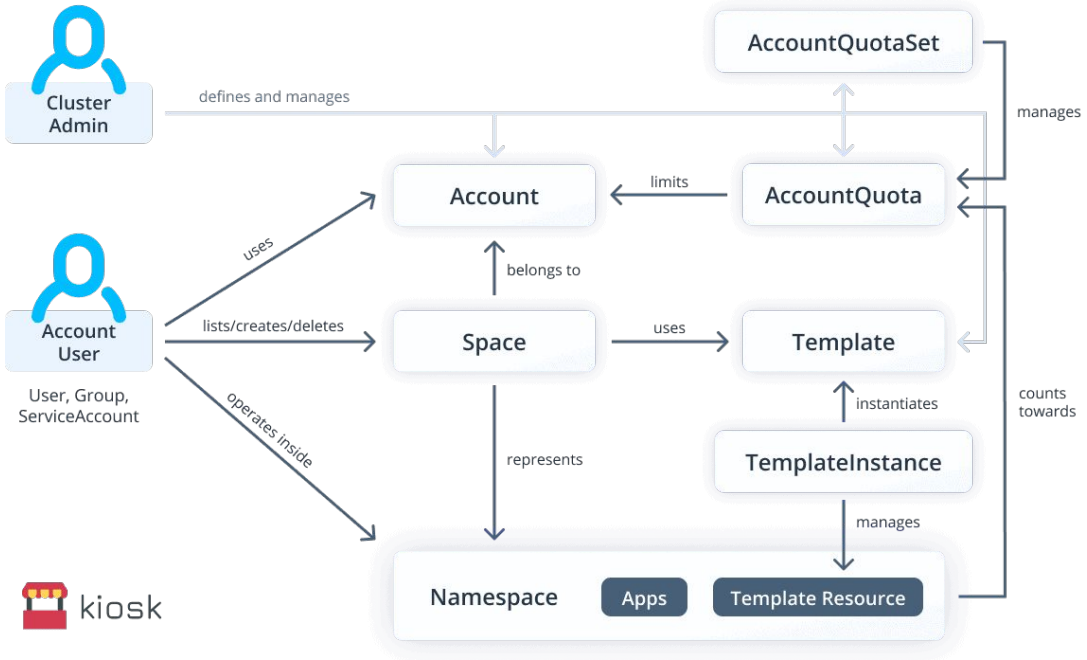




Multi-Tenancy Models

- Namespace-based: Logical separation within a cluster
- Cluster-based: Separate clusters for each tenant
- Virtual clusters: Virtualized Kubernetes control planes

Namespace Per Tenant



Key concepts and tools:

- Network policies: Control traffic between pods

- Resource quotas and limits: Manage resource consumption

- RBAC: Fine-grained access control

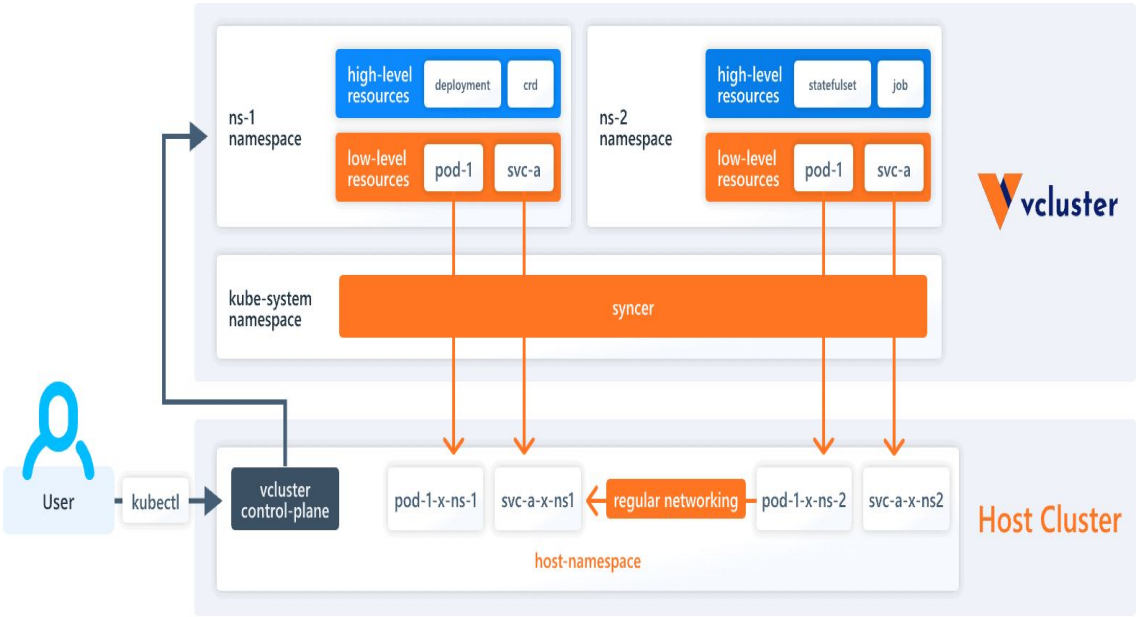
- Pod Security Policies: Define security contexts for pods

- Storage: Storage classes, persistent volume claims

Tools

1. Kiosk (Now Archived for vCluster)
2. KubeZoo
3. Capsule

Virtualized Kubernetes Control Planes



Key concepts and tools:

Virtual clusters are fully working Kubernetes clusters that run on top of other Kubernetes clusters

They have their own control plane and schedule all workloads into a single namespace of the host cluster.

Tools

1. Cluster API
2. vCluster (Semi Hard)
3. Kamaji
4. Gardener
5. HyperShift
6. Capsule Proxy ?

Challenges and Limitations

Kubernetes has several shared components. A good example is the Ingress controller, which is usually deployed once per cluster. If you submit an Ingress manifest with the same path, the last overwrites the definition and only one works. **(Soft Tenancy)**

The same challenge applies to the Kubernetes API server. Kubernetes isn't aware of the tenant, and if the API receives too many requests, it will throttle them for everyone. **(Soft Tenancy)**

Another interesting challenge is CoreDNS, *tenants abuses the DNS* The rest of the cluster will suffer too. we could limit requests with an extra plugin https://github.com/coredns/policy. **(Soft Tenancy)**

A tenant could take over nodes in the cluster just (ab)using liveness probes to solve this you could have a linter as part of your CI/CD process or use admission controllers to verify that resources submitted to the cluster are safe Using one of this library / rules Open Policy Agent. (**Soft Tenancy)**

Containers offer a weaker isolation mechanism than virtual machines to solve this we can use a container sandbox like gVisor, light virtual machines as containers (Kata containers, firecracker + containerd) or full virtual machines (virtlet as a CRI). **(Soft Tenancy)**

Hard multi-tenancy in Kubernetes, typically implemented through virtual or multi-cluster approaches, offers strong isolation but comes with significant trade-offs. **(Hard Tenancy)**

Management overhead: Maintaining multiple clusters increases operational complexity. **(Hard Tenancy)**

Longer Time To Recovery (TTR): Recovering from failures can be slower compared to soft tenancy solutions. **(Hard Tenancy)**

Inherited scheduling issues: Each tenant cluster is subject to the same scheduling challenges as a standard Kubernetes cluster. **(Hard Tenancy)**

Cascading unavailability: Node failures in a multi-tenant setup can render entire tenant clusters unreachable until the affected components (API server, scheduler, etc.) are rescheduled and operational. **(Hard Tenancy)**

API limitations: Some Kubernetes API operations may not be fully supported across all tenant clusters. **(Hard Tenancy)**

I’ve prepared some examples i would love to demo to us shortly

://localhost

Any Questions ?

Localhost-Multi-tenancy-In-Kubernetes (Github)

Localhost-Multi-tenancy-In-Kubernetes (Namespace Based)
Localhost-Multi-tenancy-In-Kubernetes (Virtual Clusters)

This repository contains examples demonstrating various methods for implementing multi-tenancy in Kubernetes. These examples are based on approaches discussed in the associated talk, providing practical implementations of the concepts presented.

By exploring these examples, developers, devops engineers and system administrators can gain hands-on experience with different multi-tenancy strategies to use in Kubernetes environments.

://localhost