Rafay Platform

GPU PaaS Multi-Tenancy Controls

RAFAY

Version History

Version	**Date**	**Updates**
0.1	July 3, 2024	Initial version based on Rafay docs
1.0	Dec 8, 2024	Standalone version

TABLE OF CONTENTS

Introduction

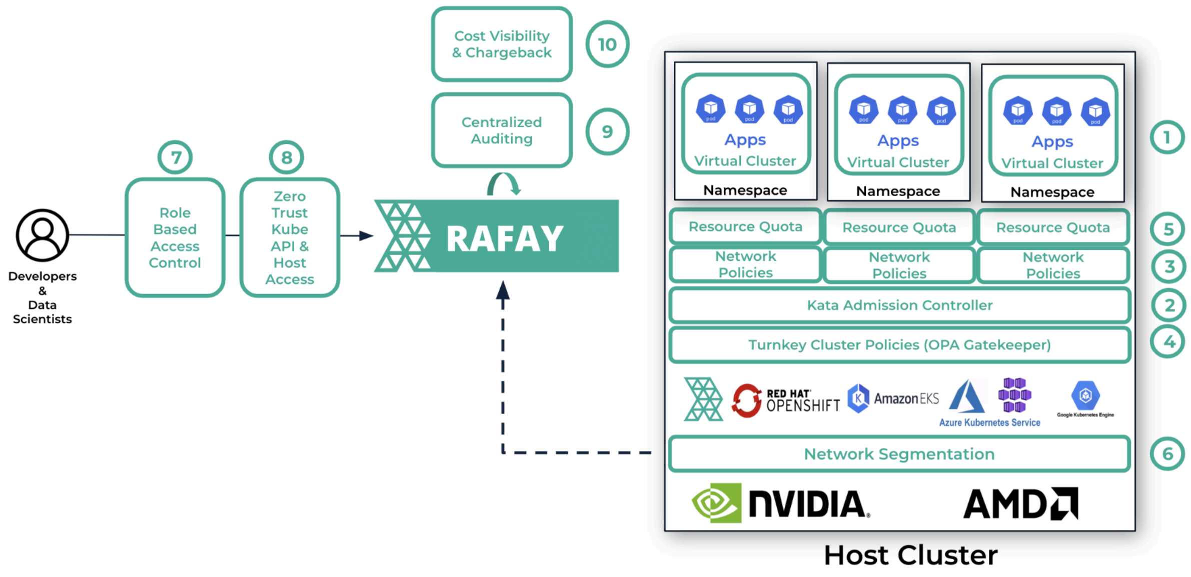
This document discusses how the Rafay Platform enables software-based multi-tenancy controls that enables operators to deploy and operate multiple isolated virtual clusters on a single cluster. These controls allow multiple tenants to co-exist on the same host cluster.These controls are automatically implemented and enforced to ensure that risks associated with threats such as lateral escalation, etc., can be blocked.

Multi-Tenancy Controls

The Rafay Platform is a multi-tenant platform that enables CSPs to deliver GPU, compute resources, AI-ML/Gen tools and services to their customers on their GPU infrastructure in a Platform-as-a-Service (PaaS) fashion. The Rafay platform allows CSPs to partition their GPU infrastructure and allocate it to multiple organizations.

With Rafay, CSPs can offer various flavors and sizes of GPU and compute resources (e.g. bare metal nodes, VMs, Kubernetes Clusters, Virtual Kubernetes clusters with one or more GPUs, and fractional GPUs). They can also provide their users with turnkey services (such as Jupyter notebooks), AI/ML tools (such as Kubeflow for MLOps and Ray) and GenAI playgrounds for LLM evaluation, fine-tuning, and inference.

To ensure that all variety of use cases can be delivered to enterprise users as well as one-off developers, being able to share infrastructure amongst multiple users is key. With the right set of controls in place, customers are able to deliver shared infrastructure to users, resulting in lower costs and higher utilization.



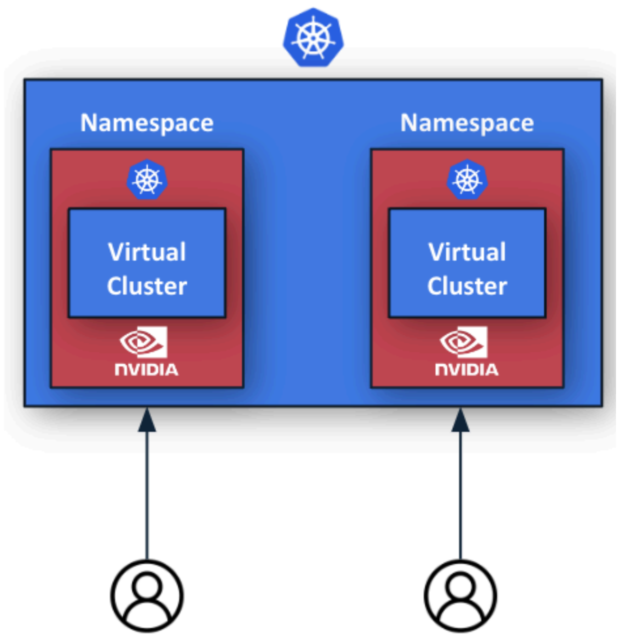
The Rafay Platform supports the following controls to ensure comprehensive, secure and simple-to-deliver multi-tenant environments:

1. Virtual Clusters

vClusters (i.e. virtual clusters) are lightweight, namespace-scoped virtual Kubernetes clusters that allow you to create **isolated Kubernetes clusters** within a larger “host” Kubernetes cluster.

vClusters behave like a full-fledged Kubernetes cluster for users and applications but are backed by the resources and API of the host cluster. vClusters run their own Kubernetes API server within a namespace of the host cluster.

VIRTUAL CLUSTER



Key Characteristics

Here’s how vClusters help achieve isolation

Control Plane Isolation

Each vCluster is isolated within a namespace on the host cluster. The virtual cluster has its own API server and control plane components, separate from the host cluster. Users interact with the vCluster as if it were an independent Kubernetes cluster.

Workload Placement

Workloads scheduled in the vCluster are translated into pods in the host cluster’s namespace.

Networking & RBAC Isolation

The vCluster has its own Kubernetes networking, RBAC, and policies, distinct from the host cluster.

Lightweight

Because vClusters are virtual, they are faster to spin up and require fewer resources than creating dedicated Kubernetes clusters.

Why is this needed?

Let's review the use cases and challenges addressed by vClusters.

Multi-Tenancy

In a multi-tenant environment, tenants often need isolated clusters. Creating dedicated clusters for each tenant is resource-intensive and operationally complex. vClusters provide tenant isolation within a single host cluster, reducing overhead while maintaining independence.

Dev/Test Environments

Developers need isolated environments for testing, debugging, and prototyping. Provisioning dedicated clusters is time-consuming and expensive. Developers can spin up lightweight vClusters quickly, test changes, and discard them when done.

Multiple Kubernetes Versions

Users may need specific Kubernetes versions for their applications which may require setting up clusters with specific versions. Since each vCluster has its own API server, you can run different Kubernetes versions in parallel on the same host cluster.

Cost Optimization

Running multiple dedicated Kubernetes clusters for isolated workloads increases costs for infrastructure and management. vClusters leverage a single host cluster’s resources, reducing infrastructure and operational expenses.

Avoid Namespace Collisions

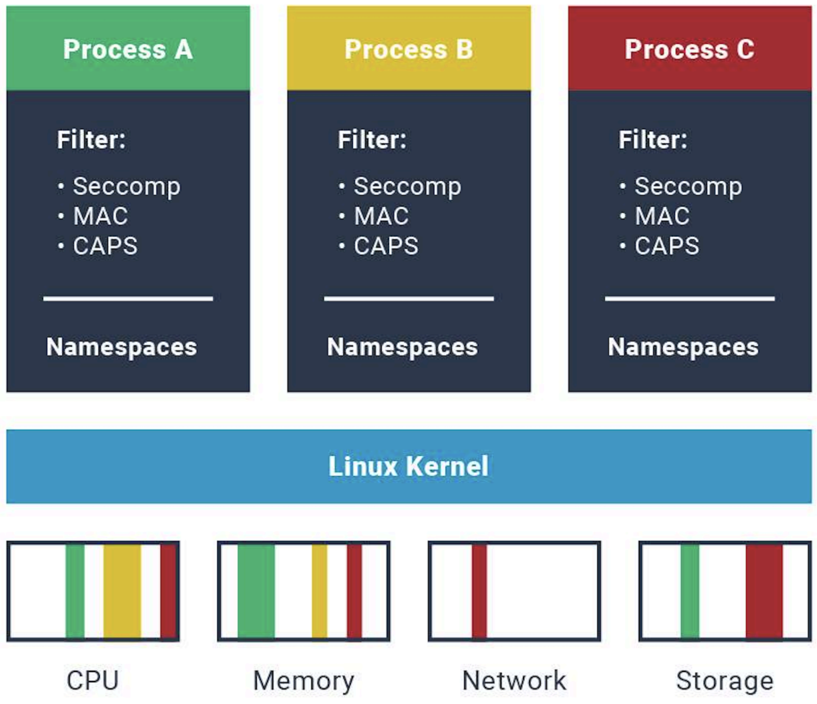
Using namespaces for multi-tenancy in Kubernetes can lead to conflicts in resource names, RBAC policies, and network configurations. Each vCluster operates independently and avoids namespace-level conflicts.

Isolation for Third-Party Applications

Third-party applications may require specific configurations or elevated permissions, which can conflict with other workloads in the host cluster. vClusters isolate the application and provide a separate control plane for configuration.

2. Runtime Isolation

Containers operating on Kubernetes clusters share the Linux kernel (i.e. default behavior). The image below shows three containers using the same Linux kernel. For shared, multi-tenant Kubernetes clusters especially with users that the service provider cannot control, there is always the risk of container escapes.

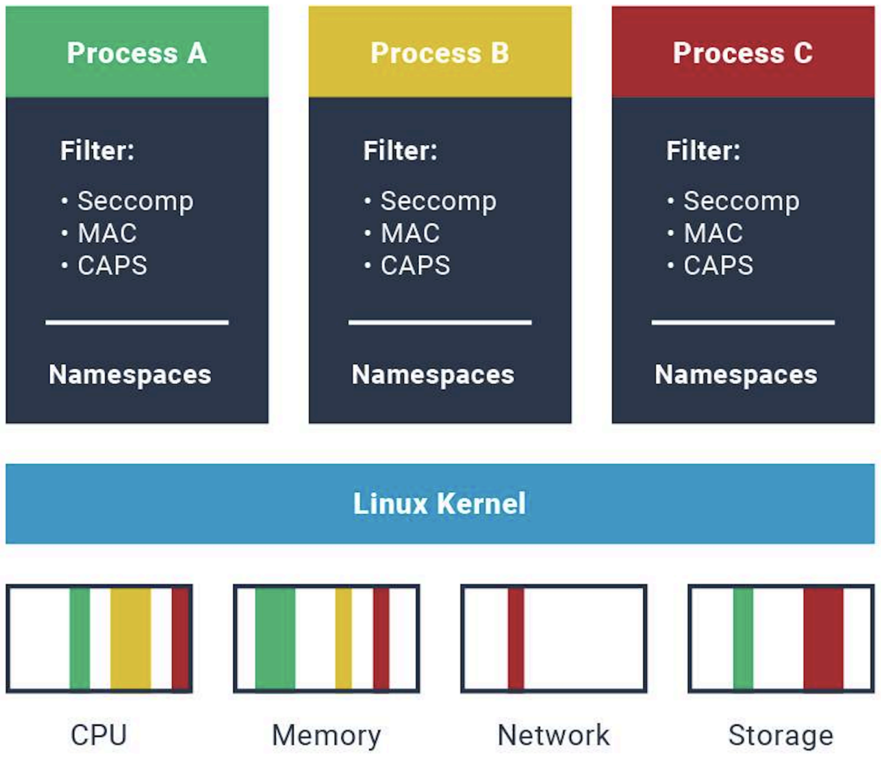


Kata containers are a secure alternative to address the issue above. Rafay's blueprint for soft multi-tenancy provides a turnkey option for Kata Containers. Rafay has developed an admission controller that will automatically deploy end user resources as kata containers without the end user having to make any changes to their application manifests.

Kata is a container runtime technology designed to provide the security advantages of virtual machines (VMs) while maintaining the lightweight performance and agility of containers. The image below visually shows how Kata containers are different from regular container runtime.

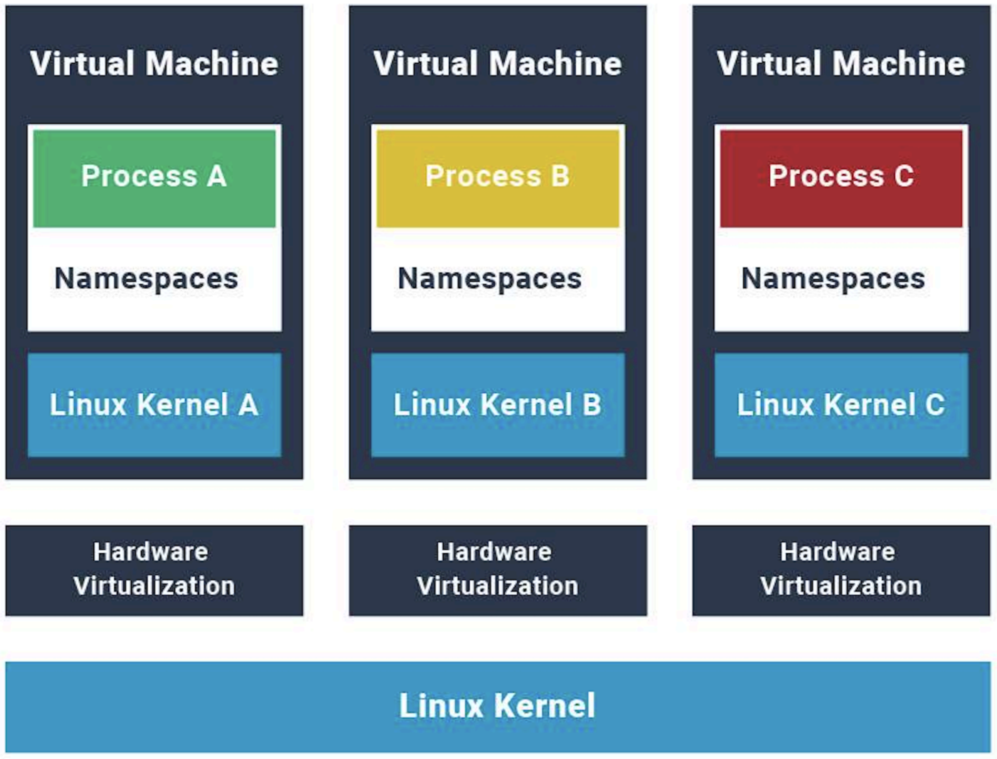
Without Kata based Isolation

Notice that with this mode, there is a single Linux kernel shared by multiple processes.



With Kata based Isolation

Notice that with Kata, each container is essentially operating inside a “microvm” on the underlying Linux kernel.



Why is this needed?

Enhanced Security

- **VM-Level Isolation**: Kata Containers run each container inside a lightweight virtual machine, providing strong isolation between containers. This VM-level isolation mitigates the "noisy neighbor" problem and reduces the risk of security breaches spreading from one container to another.
- **Protection from Kernel Vulnerabilities**: Since each container has its own kernel instance inside the VM, Kata Containers protect against vulnerabilities in the host kernel. Even if a container is compromised, it cannot directly affect the host or other containers running on the same host.

Compliance and Multi-Tenancy

- **Security Compliance**: For organizations operating in regulated industries (e.g., finance, healthcare), Kata Containers help meet strict security and compliance requirements by offering enhanced isolation. This makes them suitable for workloads that require higher levels of security.

- **Multi-Tenant Environments**: Kata Containers are ideal for multi-tenant environments, such as public cloud platforms or shared Kubernetes clusters, where multiple users or applications share the same infrastructure. The strong isolation provided by VMs helps ensure that each tenant is securely isolated from others.

Running Untrusted or High-Risk Workloads

- **Sandboxing Applications**: Kata Containers can be used to sandbox untrusted or high-risk workloads. By running these workloads in a separate virtual machine, they are isolated from the rest of the system, reducing the risk of potential security issues.
- **Securing Sensitive Data**: Workloads that handle sensitive data, such as financial transactions or personal information, can benefit from the enhanced security provided by Kata Containers, as they ensure that data is isolated and protected from other processes.

Compatibility with Existing Container Ecosystems

- **Kubernetes Integration**: Kata Containers are compatible with Kubernetes, meaning they can be used as a drop-in replacement for standard container runtimes. This allows organizations to leverage Kata Containers for workloads that require additional security without disrupting existing Kubernetes workflows.
- **Container Runtime Interface (CRI) Support**: Kata Containers are compliant with the Kubernetes Container Runtime Interface (CRI), ensuring they can seamlessly integrate with Kubernetes and other container orchestration tools.

Lightweight Virtualization

- **Performance Efficiency**: While Kata Containers provide the security benefits of VMs, they are designed to be lightweight and efficient (i.e. a microVM vs. a regular VM). The overhead is minimized compared to traditional VMs, allowing Kata Containers to offer near-native container performance.
- **Fast Boot Times**: Kata Containers leverage optimized virtual machine technologies that offer fast boot times, making them suitable for dynamic and high-performance environments where rapid scaling and responsiveness are critical.

Flexibility

- **Hybrid Workloads**: Kata Containers enable organizations to run a mix of traditional containers and more isolated Kata Containers on the same infrastructure. This flexibility allows different levels of security and isolation based on the specific requirements of each workload.
- **Edge Computing**: In edge computing scenarios, where security and performance are both critical, Kata Containers can provide secure and efficient container runtimes for edge devices that may have limited resources.

Resilience and Fault Tolerance

- **Fault Isolation**: Because each Kata Container runs in its own VM, faults or crashes within one container are isolated and do not affect other containers or the host system. This enhances the overall resilience and fault tolerance of the infrastructure.

In summary, Kata Containers are designed to bridge the gap between the high performance of traditional containers and the strong isolation of virtual machines. They are particularly useful in scenarios where security, compliance, and isolation are paramount, such as in multi-tenant environments, for running untrusted workloads, or in regulated industries. By using Kata Containers, organizations can achieve a higher level of security without sacrificing the agility and efficiency that containers provide.

3. Network Policies

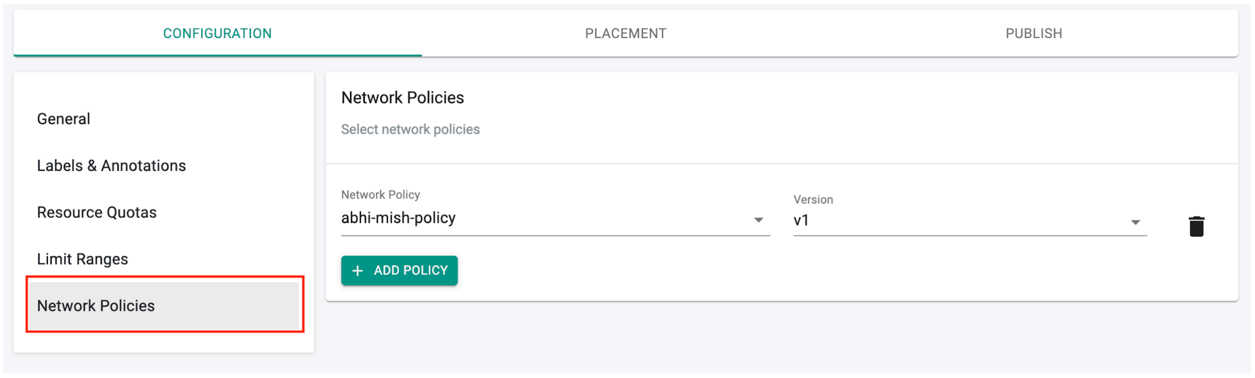
Network Policies are a mechanism to control network traffic flow within and from/to Kubernetes clusters.

With the Rafay blueprint, all namespaces are locked down with a default network policy that **blocks** all resources in the namespace to exchange network traffic with "other namespaces" and "outside the cluster"

#	Primary Threat Vectors	Default Policy

1	Rogue container tries to connect to resources in other namespaces	BLOCKED
2	Rogue container tries to connect to resources outside cluster to exfiltrate data or download malicious software	BLOCKED

Shown below is a screenshot of what network policy enforcement looks like in Rafay



Network Policy: abhi-mish-policy
Version: v1

Why is this needed?

Network policies are crucial for multi-tenant environments because of the following reasons:

Isolation Between Tenants

- **Security Boundaries:** In a multi-tenant environment, different teams, departments, or organizations share the same Kubernetes cluster. Network policies help enforce isolation between tenants by controlling which pods can communicate with each other across different namespaces. Without network policies, there could be unintended or unauthorized cross-tenant communication, potentially leading to data leaks or security breaches.

- **Namespace Segmentation:** Network policies can restrict traffic between namespaces, ensuring that each tenant's workloads are isolated from others. This is particularly important for maintaining the confidentiality and integrity of data and applications within each tenant's namespace.

Enhanced Security Posture

- **Minimizing Attack Surface:** By default, Kubernetes allows unrestricted communication between pods within the same cluster. Network policies allow administrators to restrict this communication, minimizing the attack surface by only allowing necessary traffic. This reduces the risk of lateral movement by attackers within the cluster.
- **Preventing Data Exfiltration:** Network policies can be configured to block traffic from sensitive workloads to the internet or other untrusted networks, preventing unauthorized data exfiltration and ensuring that sensitive data remains secure within the cluster.

Compliance and Regulatory Requirements

- **Data Protection:** Many industries have stringent regulations around data protection, particularly when dealing with multi-tenant environments. Network policies help ensure that only authorized communication occurs within the cluster, which is crucial for meeting compliance requirements such as GDPR, HIPAA, or PCI-DSS.
- **Auditable Controls:** Implementing network policies provides auditable controls over network traffic, which is often required for regulatory compliance. This helps demonstrate to auditors that appropriate measures are in place to protect data and control access.

Traffic Management

- **Controlled Communication:** Network policies enable fine-grained control over pod-to-pod and pod-to-service communication, ensuring that only essential traffic flows within the cluster. This can help optimize network performance and reduce unnecessary traffic, leading to more efficient use of resources.
- **Service Isolation:** Critical services can be isolated from non-essential or less secure services using network policies, ensuring that high-value services have the necessary protection and performance guarantees.

Mitigating Misconfigurations

- **Preventing Miscommunication:** In complex multi-tenant environments, it’s easy to misconfigure applications or services, leading to unintentional exposure of sensitive data. Network policies act as an additional layer of defense, ensuring that even if there’s a configuration error, unauthorized traffic is still blocked.
- **Default-Deny Posture:** By implementing a "default-deny" network policy, administrators can ensure that no communication is allowed unless explicitly permitted. This proactive approach helps mitigate the risk of accidental exposure due to misconfigurations.

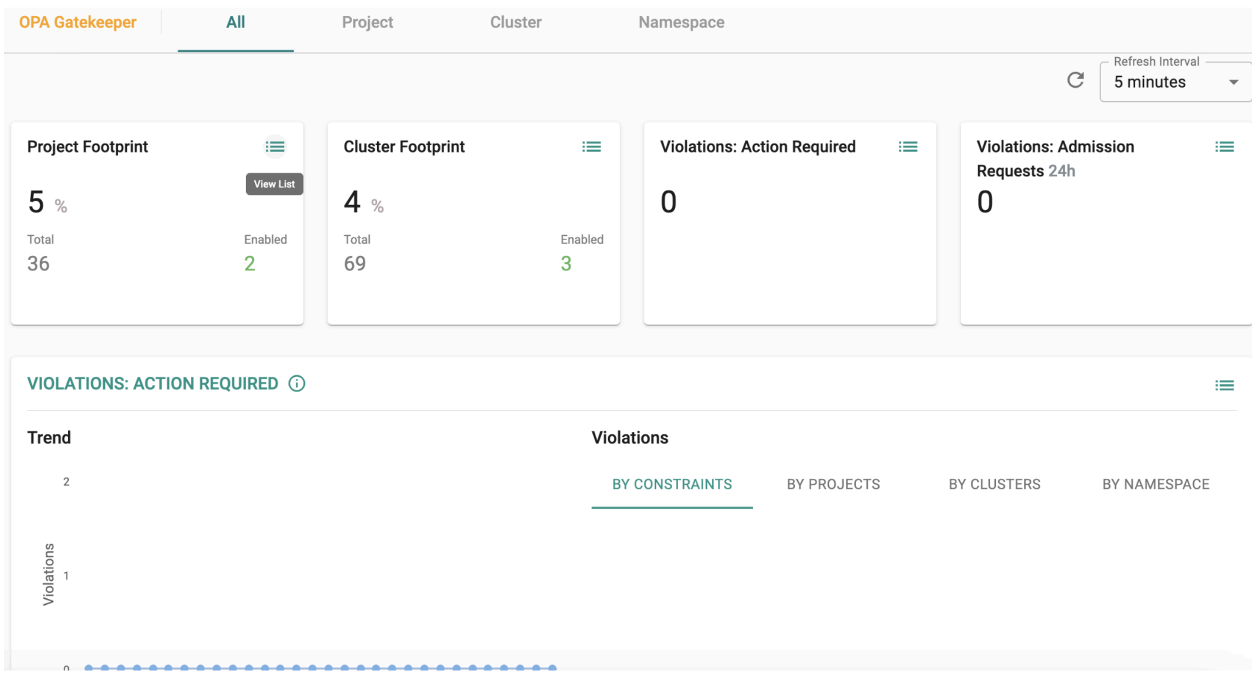
Supporting DevOps and Microservices

- **Limited Blast Radius:** If a tenant's application is compromised, network policies can limit the impact (blast radius) of the breach by preventing the attacker from spreading to other parts of the cluster.

In summary, network policies are essential in a multi-tenant environment to enforce isolation, enhance security, comply with regulations, and manage traffic effectively. They provide a necessary layer of control that ensures tenants can coexist securely within the same host cluster.

4. Cluster Policies

With all clusters under management, a default cluster level policy (based on **OPA Gatekeeper with out-of-box policies**) is automatically enforced and implemented to strengthen governance. This provides the means to control what users can/cannot do on the cluster. This also ensures that the clusters are always in compliance with centralized policies. The cluster policies are closely coordinated with network policies to ensure there is defense in depth and completeness.



Refresh Interval: 5 minutes

Project Footprint: 5 %, Total 36, Enabled 2

Cluster Footprint: 4 %, Total 69, Enabled 3

Violations: Action Required: 0

Violations: Admission Requests 24h: 0

VIOLATIONS: ACTION REQUIRED

Why is this needed?

In a Kubernetes based multi-tenant environment, cluster policies are crucial for enforcing security, resource management, and compliance across different tenants. Here are some important policies that are particularly relevant:

Security Policies

- **Security Policies for Pods**
  - Policy Purpose: Enforce security best practices for pods, such as running as non-root, restricting the use of privileged containers, and controlling the

    use of hostPath volumes. This ensures that tenants cannot deploy potentially insecure workloads that might compromise the cluster.
  - Example: Enforce that all pods must run as non-root and disallow privileged mode.

- **Image Whitelisting/Blacklisting**
  - Policy Purpose: Restrict the use of container images to approved repositories or specific image versions. This helps prevent the deployment of untrusted or vulnerable images by tenants.
  - Example: Enforce that all images must be pulled from an internal registry or a specific list of trusted registries.

Label and Annotation Policies

- **Mandatory Labels and Annotations**
  - Policy Purpose: Enforce the use of specific labels and annotations on resources for tracking, auditing, and management purposes. This is important for organizing resources, applying policies selectively, and ensuring consistent tagging across tenants.
  - Example: Enforce that every resource has labels identifying the tenant, environment (e.g., dev, prod), and application.

- **Namespace Ownership**
  - Policy Purpose: Ensure that namespaces have specific labels or annotations that identify the owning team or organization, which helps in managing tenant-specific policies and access controls.
  - Example: Enforce that each namespace has an owner label that maps to the responsible team or tenant.

Admission Control Policies

- **Restrict Certain Resource Types**
  - Policy Purpose: Prevent tenants from creating or modifying certain resource types that could impact the entire cluster (e.g., preventing tenants from creating ClusterRoleBindings or mutating webhooks).
  - Example: Enforce that tenants cannot create or modify ClusterRoles, ClusterRoleBindings, or custom resource definitions (CRDs).

- **Disallow Specific Configurations**

    - Policy Purpose: Prevent tenants from applying certain configurations that might be insecure or violate cluster policies (e.g., disallowing the use of certain annotations that might bypass security controls).
    - Example: Enforce that hostNetwork is not enabled in pod specifications, or block the use of the kubernetes.io/ingress.class annotation unless it adheres to predefined classes.

Compliance Policies

- **Compliance with Regulatory Standards**
    - Policy Purpose: Ensure that tenant workloads comply with regulatory standards such as GDPR, HIPAA, or PCI-DSS by enforcing specific configurations and controls.
    - Example: Enforce that all sensitive workloads have encryption enabled for data at rest and in transit.

In summary, centralized policy enforcement plays a crucial role in enforcing security, resource management, compliance, and operational consistency. By implementing these policies, you can ensure that each tenant operates within their allocated boundaries, adheres to security best practices, and complies with organizational or regulatory requirements.

5. Role-Based Access Control

Kubernetes RBAC is a critical security control to ensure that users and workloads only have access to resources required to execute their tasks. Users are allocated a namespace on the host cluster where the virtual cluster is deployed for them. The users only have access to the virtual cluster and do not even know about the existence of the underlying Kubernetes namespace.

#	**Primary Threat Vector**	**Default Policy**

1	Ensure that tenants can only see and operate on resources inside their assigned virtual cluster operating inside a namespace in the host cluster	ENFORCED

The tenant/user is automatically mapped to a "ClusterRole" for the virtual cluster. Note that although the virtual cluster is operating inside a namespace (with a role) in the host cluster, the user is not provided access to the namespace itself.

Why is this needed?

Role-Based Access Control (RBAC) is required because of the following reasons:

Security and Access Management

- **Fine-Grained Permissions**: RBAC allows administrators to define granular access controls, ensuring that users, applications, and services have only the permissions they need to perform their tasks—no more, no less. This principle of least privilege minimizes the risk of unauthorized actions.
- **User Authentication and Authorization**: RBAC helps in authenticating users and authorizing their actions based on their roles, preventing unauthorized access to sensitive resources within the Kubernetes cluster.

Multi-Tenancy Support

- **Isolated Environments**: In a multi-tenant Kubernetes environment where multiple teams or organizations share the same cluster, RBAC helps isolate resources and operations between tenants. This prevents one team from inadvertently or maliciously accessing or modifying another team's resources.
- **Namespace-Specific Controls**: RBAC allows for the application of different access controls to different namespaces, supporting environments where different teams or projects operate within their own isolated namespaces.

Compliance and Auditing

- **Regulatory Compliance**: Many industries have strict regulations around access control and data handling. RBAC allows organizations to enforce these controls within their Kubernetes clusters, helping to meet compliance requirements.
- **Auditability**: RBAC provides a clear and auditable record of who has access to what resources and what actions they are allowed to perform. This is critical for tracking changes, troubleshooting issues, and ensuring accountability.

Operational Efficiency

- **Centralized Management**: RBAC centralizes access management, making it easier for administrators to define, modify, and manage permissions across a large and complex Kubernetes environment.
- **Scalability**: As a Kubernetes cluster scales, the number of users and resources grows, making manual permission management impractical. RBAC provides a scalable way to manage access across many users and resources.

Protection Against Insider Threats

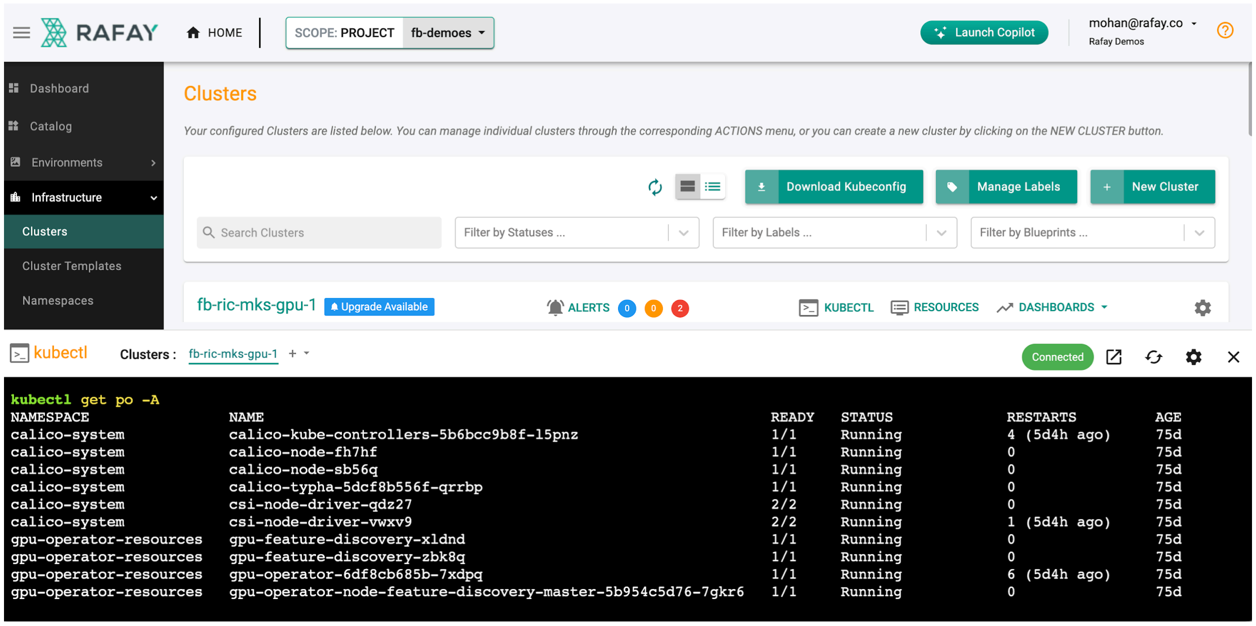
- **Minimizing Internal Risks**: RBAC reduces the risk of insider threats by ensuring that users only have access to the resources they need to do their jobs. This limits the potential damage from accidental or malicious actions by insiders.
- **Role Segregation**: By assigning roles based on specific job functions, RBAC helps enforce role segregation, ensuring that no single user has excessive control or access across the system.

In summary, RBAC is a critical component of Kubernetes that enhances security, supports multi-tenancy, ensures compliance, and improves operational efficiency by providing a flexible and scalable way to manage access to resources.

6. Secure, Remote Access

To ensure highest levels of security, all users are required to centrally authenticate using the configured Identity Provider (IdP). Once successfully authenticated, an ephemeral service account for the user is federated on the remote cluster in a **Just in Time (JIT)** manner.

Users are provided with the means to remotely access their namespace and perform Kubectl operations using the Kubectl CLI or an integrated browser based shell.



```
kubectl get po -A
NAMESPACE                NAME                                                     READY   STATUS             RESTARTS       AGE
calico-system            calico-kube-controllers-5b6bcc9b8f-15pnz                 1/1     Running            4 (5d4h ago)   75d
calico-system            calico-node-fh7hf                                        1/1     Running            0              75d
calico-system            calico-node-sb56q                                        1/1     Running            0              75d
calico-system            calico-typha-5dcf8b556f-qrrbp                            1/1     Running            0              75d
calico-system            csi-node-driver-qdz27                                    2/2     Running            0              75d
calico-system            csi-node-driver-vwxv9                                    2/2     Running            1 (5d4h ago)   75d
gpu-operator-resources   gpu-feature-discovery-xldnd                              1/1     Running            0              75d
gpu-operator-resources   gpu-feature-discovery-zbk8q                              1/1     Running            0              75d
gpu-operator-resources   gpu-operator-6df8cb685b-7xdpq                            1/1     Running            6 (5d4h ago)   75d
gpu-operator-resources   gpu-operator-node-feature-discovery-master-5b954c5d76-7gkr6   1/1     Running            0              75d
```

Why is this needed?

Secure remote access is critical for enhancing security, governance, and operational efficiency. Here are some reasons why it is required:

Zero Trust Principles

- **Identity Verification**: Enforce strict identity verification before granting access. Every request is authenticated and authorized, ensuring that only legitimate users with the appropriate permissions can interact with the cluster.
- **No Implicit Trust**: In a Zero Trust model, no user or device is inherently trusted, even if they are within the network perimeter. Reduce the risk of insider threats or compromised credentials leading to unauthorized access.

Granular Access Control

- **Least Privilege Access**: Allows administrators to enforce the principle of least privilege by defining granular roles and permissions. Users are granted only the minimal level of access necessary for their tasks, which limits the potential impact of a compromised account.
- **Contextual Access**: Access can be restricted based on various factors such as time of day, location, or the security posture of the device being used. This adds

another layer of control, ensuring that even authorized users can only access resources under specific conditions.

Compliance and Auditing

- **Detailed Audit Logs**: Comprehensive and centralized logging of all actions performed within the cluster. This is crucial for meeting compliance requirements, as it ensures that all access and modifications are fully traceable and auditable.
- **Regulatory Compliance**: For organizations subject to stringent regulatory frameworks (e.g., GDPR, HIPAA, PCI-DSS), ensure that access controls are enforced and that all actions within the Kubernetes environment can be audited.

Operational Efficiency and Governance

- **Centralized Management**: Centralized management of access controls, policies, and auditing across multiple clusters. This simplifies governance and ensures consistent policy enforcement across all environments, whether on-premises or in the cloud.
- **User-Friendly Interface**: The platform offers an intuitive interface for managing kubectl access, making it easier for administrators to configure and enforce policies without needing to manually manage complex configurations or scripts.

Multi-Tenancy Support

- **Tenant Isolation**: In multi-tenant environments, ensure that tenants are properly isolated from each other. Restrict users' access to only the resources within their own namespace or tenant, preventing accidental or malicious cross-tenant access.
- **Per-Tenant Policies**: Administrators can define different policies for different tenants, ensuring that each tenant's security and access requirements are met without compromising the overall security of the cluster.

Seamless Integration with Existing Workflows

- **Compatibility with Existing Tools**: Integrates seamlessly with existing CI/CD pipelines, development workflows, and Kubernetes management tools. This ensures that security enhancements do not disrupt ongoing operations or require significant changes to established processes.

- **Automation and Scripting Support**: The solution supports automation and scripting, enabling teams to maintain their productivity while adhering to stringent security controls.

Zero Trust Network Access (ZTNA)

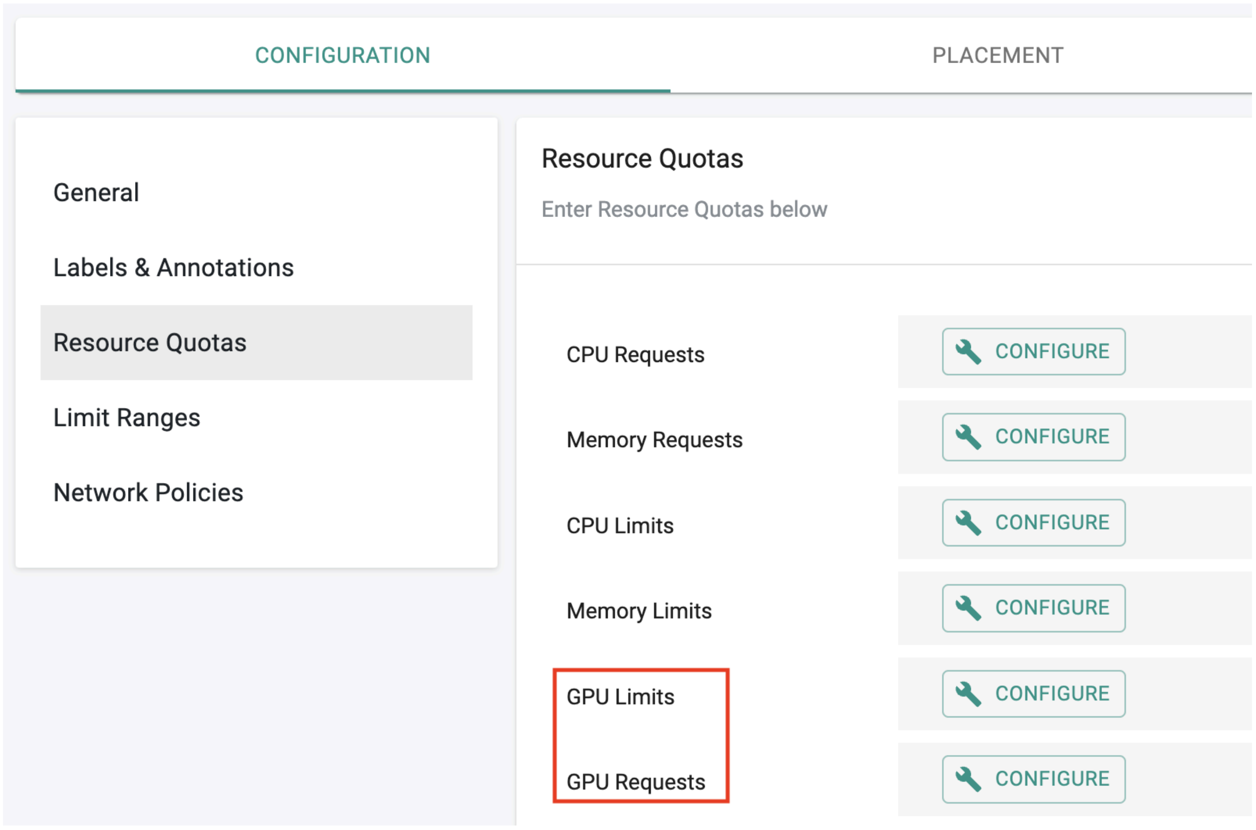
- **Secure Access**: For remote teams or distributed environments, provide a Zero Trust Network Access (ZTNA) model to provide secure, authenticated access to private/remote Kubernetes clusters without requiring a VPN. This enhances security for remote operations while maintaining ease of access for users.

Significantly enhance the security, compliance, and operational efficiency by implementing Zero Trust principles. Provide granular access controls, robust auditing capabilities, and centralized management, making it an essential tool for organizations that need to secure their environments, particularly in multi-tenant or highly regulated settings.

7. Resource Quotas

Resource quotas and limits are automatically implemented and enforced. This prevents one tenant's workloads from affecting the performance or availability of another tenant's workloads.

Shown below is an example of resource quotas for GPU in this namespace.


CONFIGURATION | PLACEMENT

General
Labels & Annotations
Resource Quotas
Limit Ranges
Network Policies

Resource Quotas

Enter Resource Quotas below

CPU Requests	CONFIGURE
Memory Requests	CONFIGURE
CPU Limits	CONFIGURE
Memory Limits	CONFIGURE
GPU Limits	CONFIGURE
GPU Requests	CONFIGURE

Why is this needed?

Resource quotas are essential for managing and controlling the allocation of resources such as CPU, memory, and storage. Here's why they are necessary:

Prevent Resource Exhaustion

- **Cluster Stability**: In a multi-tenant environment where multiple users or teams share the same Kubernetes cluster, resource quotas ensure that no single namespace or application can consume all available resources, which could lead to resource exhaustion and destabilize the entire cluster.
- **Fair Distribution**: Quotas help ensure a fair distribution of resources among different teams or applications, preventing situations where a single team could monopolize resources, leaving others with insufficient capacity.

Limit Overprovisioning

- **Cost Management**: By setting resource quotas, administrators can prevent users from overprovisioning resources (e.g., requesting excessive CPU or memory) that might not be fully utilized, leading to unnecessary costs.
- **Efficiency**: Quotas encourage efficient use of resources, as users must request only the resources they actually need, optimizing the overall utilization of the cluster.

Control Resource Allocation

- **Budget Enforcement**: In environments where different teams have allocated budgets or resource limits, quotas enforce these limits, ensuring that teams do not exceed their allowed resource consumption.
- **Prioritization**: Quotas can be used to prioritize critical workloads by ensuring they have sufficient resources, while less critical workloads are restricted to a smaller quota.

Improve Performance Isolation

- **Quality of Service (QoS)**: By setting resource quotas, you can better manage the Quality of Service for different applications. This ensures that one application's resource-intensive tasks don't degrade the performance of other applications running in the same cluster.
- **Avoid Resource Contention**: Quotas reduce the likelihood of resource contention, where multiple applications compete for the same resources, potentially leading to degraded performance or failures.

Compliance and Governance

- **Regulatory Compliance**: In regulated environments, there may be requirements to limit and monitor resource usage to ensure compliance with industry standards or regulations.
- **Auditing and Reporting**: Quotas facilitate auditing and reporting by providing a clear record of how resources are allocated and consumed across different namespaces and teams.

Simplify Capacity Planning

- **Predictable Usage**: Quotas provide predictable resource usage patterns, making it easier for administrators to plan for future capacity needs and scale the cluster appropriately.
- **Cluster Sizing**: By knowing the resource limits for each namespace, administrators can better estimate the size and capacity requirements of the entire cluster.

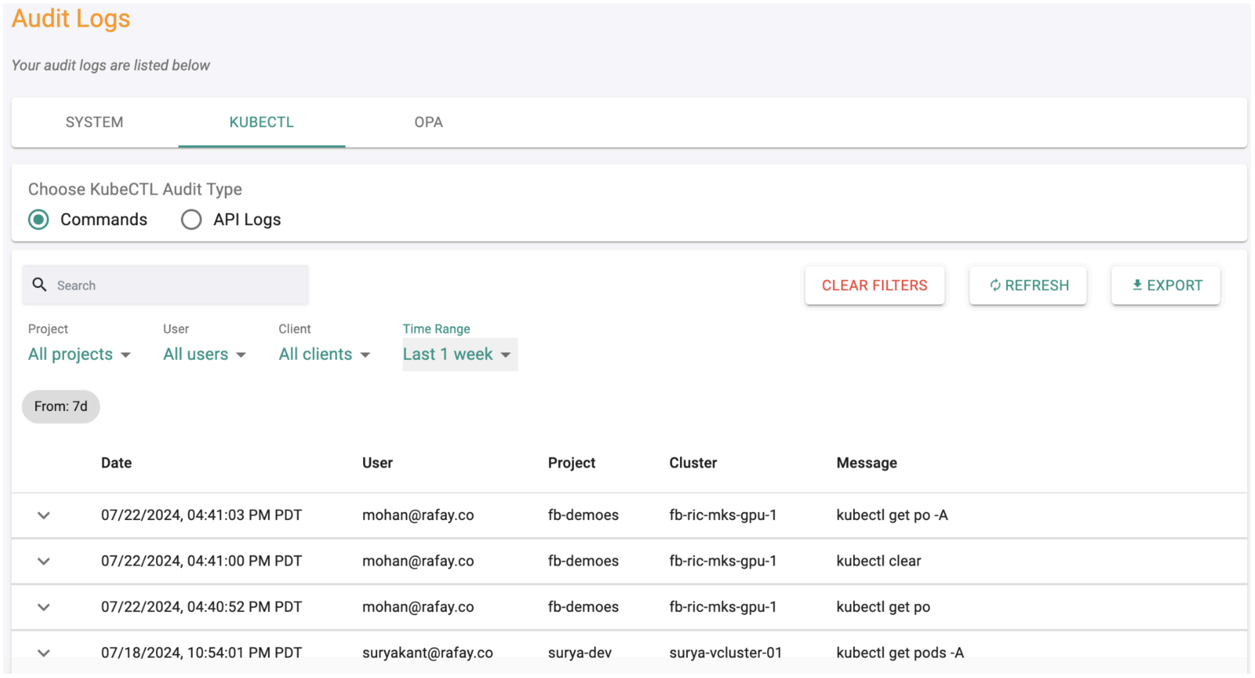
Support Multi-Tenancy

- **Tenant Isolation**: In multi-tenant environments, where multiple organizations or teams share the same Kubernetes cluster, quotas help ensure that tenants are isolated in terms of resource usage. This prevents one tenant's workloads from affecting the performance or availability of another tenant's workloads.

In summary, resource quotas are critical for ensuring efficient, fair, and predictable resource usage in a multi-tenant environment. They help maintain cluster stability and support overall resource management strategies.

8. Audit Logging

A centralized and immutable audit trail is automatically maintained for all activity performed by the users via all supported interfaces (UI and programmatic). Administrators are provided with centralized access to the audit logs. The audit logs can also be configured to be streamed in real time to a configured SIEM.



Project: All projects · User: All users · Client: All clients · Time Range: Last 1 week

From: 7d

	Date	User	Project	Cluster	Message
	07/22/2024, 04:41:03 PM PDT	mohan@rafay.co	fb-demoes	fb-ric-mks-gpu-1	kubectl get po -A
	07/22/2024, 04:41:00 PM PDT	mohan@rafay.co	fb-demoes	fb-ric-mks-gpu-1	kubectl clear
	07/22/2024, 04:40:52 PM PDT	mohan@rafay.co	fb-demoes	fb-ric-mks-gpu-1	kubectl get po
	07/18/2024, 10:54:01 PM PDT	suryakant@rafay.co	surya-dev	surya-vcluster-01	kubectl get pods -A

Why is this needed?

Centralized audit logging is crucial for multi-tenant environments for several important reasons:

Security and Incident Response

- **Comprehensive Monitoring**: Centralized audit logs provide a complete and consistent record of all activities, including user actions, API requests, and changes to resources. This is essential for monitoring suspicious activities, detecting security incidents, and responding to breaches.
- **Forensic Analysis**: In the event of a security incident, centralized audit logs enable detailed forensic analysis to understand the scope and impact of the breach. This helps identify the root cause, the sequence of events, and any affected resources.

Compliance and Regulatory Requirements

- **Audit Trail**: Many regulatory frameworks (such as GDPR, HIPAA, PCI-DSS) require organizations to maintain an audit trail of all access and changes to sensitive systems. Centralized audit logging ensures that these requirements are met by capturing all relevant actions across the entire cluster.

- **Reporting and Verification**: Centralized logs make it easier to generate compliance reports and verify that policies and procedures are being followed, as all logs are aggregated in one place and can be queried or analyzed systematically.

Operational Efficiency

- **Automated Alerts and Monitoring**: With centralized logging, it's easier to set up automated alerts and monitoring for specific events or patterns of behavior that may indicate an issue. This can help in proactive detection and resolution of problems before they escalate.

Visibility and Transparency

- **Unified View of Activity**: Centralized audit logs provide a unified view of all activities within the tenant, making it easier for administrators, security teams, and auditors to understand what is happening across the environment. This visibility is crucial for maintaining control over complex and dynamic environments.
- **Transparency for Multi-Tenant Environments**: In multi-tenant environments, centralized logging ensures that activities from all tenants are logged and can be reviewed. This promotes transparency and accountability, ensuring that tenants' actions are visible and can be audited if necessary.

Troubleshooting and Debugging

- **Root Cause Analysis**: When issues arise, centralized audit logs provide a complete history of actions and events that can help in diagnosing and resolving the problem. By having all logs in one place, it becomes easier to trace issues across different components and identify the root cause.
- **Cross-Component Correlation**: Centralized logging allows for correlating events across different components, making it easier to understand complex interactions and dependencies.

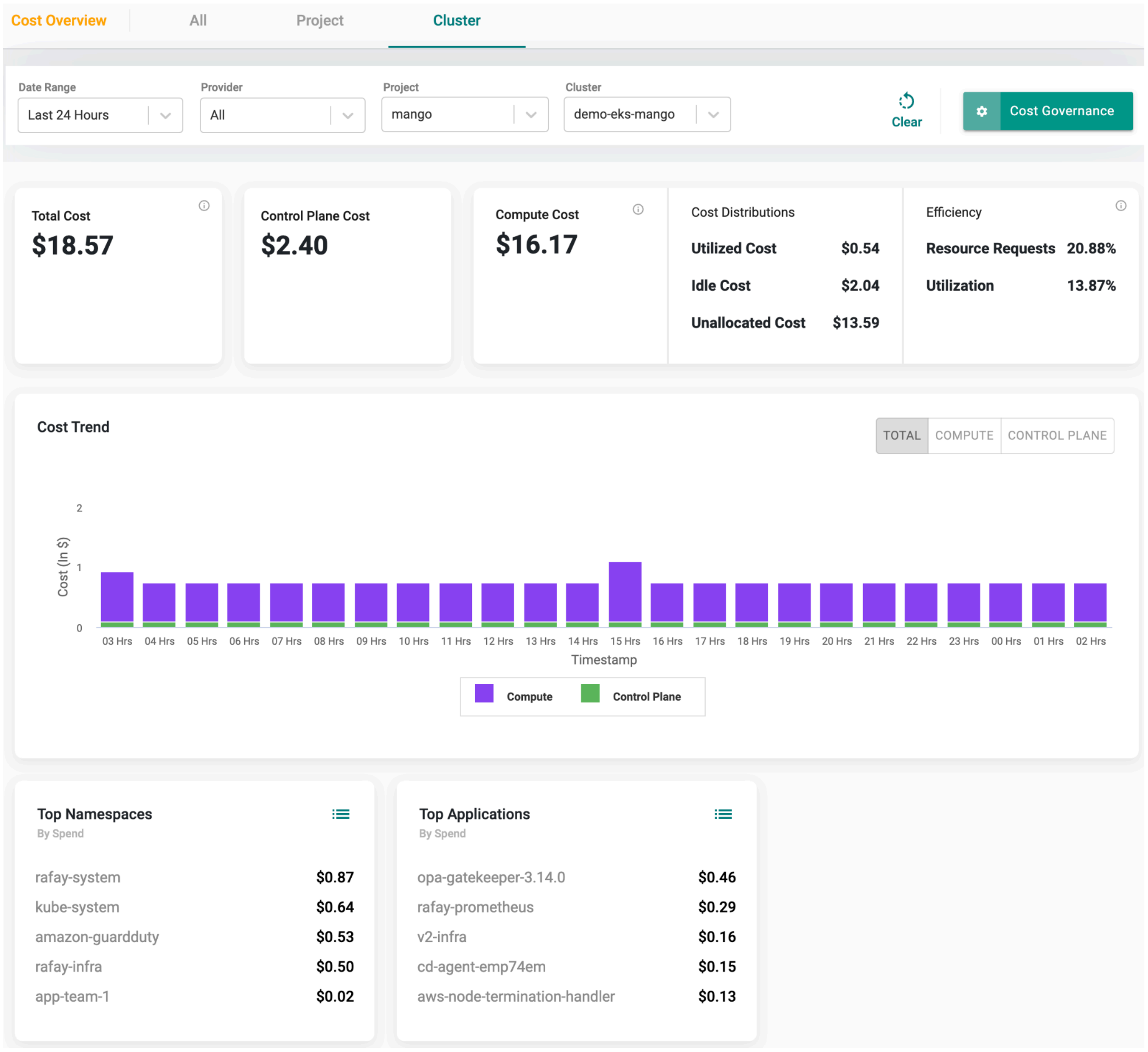
Consistent Policy Enforcement

- **Audit Policy Compliance**: By aggregating logs in a centralized system, organizations can ensure that their audit policies are consistently applied and enforced across the entire environment.

In summary, centralized audit logging is essential for maintaining security, compliance, operational efficiency, and visibility. It provides a single, reliable source of truth for all activities enabling effective monitoring, troubleshooting, and governance.

9. Cost Allocation

Administrators that configure and enable cost profiles for their Kubernetes clusters will benefit from the integrated cost visibility and allocation/governance capabilities in the platform. Enabling this is considered an industry best practice because it will provide the organization with a view into total spend, spend by workspace, spend by user etc. This data can then be used for internal billing or charge back workflows.


Cost Overview | All | Project | Cluster

Date Range	Provider	Project	Cluster
Last 24 Hours	All	mango	demo-eks-mango

Clear | Cost Governance

Total Cost	Control Plane Cost	Compute Cost
$18.57	$2.40	$16.17

Cost Distributions

Utilized Cost	$0.54
Idle Cost	$2.04
Unallocated Cost	$13.59

Efficiency

Resource Requests	20.88%
Utilization	13.87%

Cost Trend

TOTAL | COMPUTE | CONTROL PLANE

Top Namespaces
By Spend

rafay-system	$0.87
kube-system	$0.64
amazon-guardduty	$0.53
rafay-infra	$0.50
app-team-1	$0.02

Top Applications
By Spend

opa-gatekeeper-3.14.0	$0.46
rafay-prometheus	$0.29
v2-infra	$0.16
cd-agent-emp74em	$0.15
aws-node-termination-handler	$0.13

Why is this needed?

Cost allocation is crucial for several reasons, particularly in environments where resources are shared across multiple teams, projects, or applications.

Visibility into Resource Consumption

- **Detailed Cost Insights**: Kubernetes environments are often complex, with many services, applications, and teams sharing the same infrastructure. Cost allocation provides visibility into how resources (like CPU, memory, and storage) are consumed by different entities, helping organizations understand where their cloud spend is going.

- **Identifying Cost Drivers**: By breaking down costs at the level of namespaces, labels, or specific workloads, cost allocation helps identify which applications or teams are the major cost drivers. This insight is crucial for making informed decisions about resource usage and cost management.

Accountability and Chargeback

- **Team Accountability**: Cost allocation allows organizations to assign costs to specific teams or departments based on their resource usage. This promotes accountability, as teams are more likely to manage their resources efficiently when they are responsible for the associated costs.
- **Chargeback and Showback**: In multi-tenant environments, cost allocation enables chargeback (billing teams or departments for their usage) or showback (informing teams of their usage costs without actual billing). This is essential for fair cost distribution and for promoting cost-conscious behavior across the organization.

Budget Management and Planning

- **Budget Tracking**: With accurate cost allocation, organizations can set budgets for different teams or projects and track how actual spending aligns with these budgets. This helps prevent cost overruns and ensures that resources are used within predefined financial constraints.
- **Capacity Planning**: Understanding how costs are distributed across different parts of the organization aids in capacity planning. It allows organizations to anticipate future resource needs and budget accordingly, ensuring that they can scale efficiently without unexpected costs.

Optimizing Resource Usage

- **Cost Efficiency**: Cost allocation highlights inefficient use of resources, such as over-provisioned instances or idle resources. By identifying these inefficiencies, organizations can take steps to optimize resource usage, such as rightsizing workloads or implementing auto-scaling, to reduce unnecessary costs.
- **Prioritizing Investments**: With detailed cost data, organizations can prioritize investments in areas that deliver the most value relative to their cost, helping to align spending with business objectives.

Multi-Tenancy Support

- **Fair Resource Distribution**: In multi-tenant clusters, cost allocation ensures that each tenant (whether a team, department, or external customer) is charged fairly for their usage. This is critical for maintaining fairness and preventing disputes over resource costs in shared environments.
- **Tenant Isolation and Cost Tracking**: Proper cost allocation also supports tenant isolation by clearly tracking and reporting costs per tenant, which is important for both internal cost management and for billing external customers.

Supporting Financial Accountability

- **FinOps Integration**: Cost allocation is a key component of FinOps (Financial Operations), which focuses on optimizing cloud spending while maintaining agility. By integrating cost data into DevOps workflows, organizations can ensure that development and operations teams are making cost-effective decisions throughout the software development lifecycle.
- **Continuous Cost Monitoring**: In a DevOps culture, where changes are frequent, continuous monitoring of costs through allocation helps teams react quickly to unexpected spending spikes, ensuring that cost efficiency is maintained even in dynamic environments.

Compliance and Reporting

- **Regulatory Compliance**: In some industries, there may be regulatory requirements to track and report how resources are used, especially in shared environments. Cost allocation facilitates compliance by providing detailed, auditable records of resource consumption and associated costs.
- **Transparent Reporting**: Transparent reporting of costs helps stakeholders understand the financial impact of Kubernetes operations, enabling better decision-making at the executive level and fostering trust in the organization’s financial management practices.

Encouraging Efficient Cloud Usage

- **Behavioral Impact**: When teams are aware of their resource usage and its associated costs, they are more likely to adopt practices that reduce waste, such as optimizing application performance, using spot instances, or implementing efficient data storage strategies.
- **Incentivizing Savings**: Cost allocation can be used to incentivize teams to find ways to save costs, perhaps through bonuses or recognition programs for those

who keep their spending under control while meeting their performance objectives.

Cost allocation is essential for providing visibility, accountability, and control over resource usage and associated costs. It supports financial planning, promotes efficient use of resources, and ensures that costs are fairly distributed among users or tenants.

10. Network Segmentation

Network segmentation is a common practice to ensure isolation between tenants. In Kubernetes, Kube-OVN is a network plugin that provides advanced networking capabilities such as isolation through Virtual Private Clouds (VPCs), security policies, and subnet segregation.

Kube-OVN, as its name implies, is an integration of the OVN/OVS based networking with Kubernetes. Multi-tenancy with VPCs is generally considered a best practice. Kube-OVN enables multi-tenant networking in Kubernetes by introducing a new set of networking CRDs:

- VPC
- Subnet
- NAT-Gateway

Why is this needed?

In the section below, we describe why network segmentation is a critical control to achieve multi-tenancy in a shared Kubernetes environment.

Fine-Grained Network Isolation

Native Kubernetes networking assumes all Pods can communicate with each other. For multi-tenancy, strict isolation is essential, especially in shared environments. Kube-OVN supports the creation of VPCs, subnets, and logical routers, enabling network segmentation at a tenant level. With this, you can:

- Isolate network traffic between different tenants (e.g., teams, departments, or applications).
- Prevent unauthorized cross-communication between namespaces.

Support for Virtual Private Clouds (VPCs)

Multi-tenancy often requires each tenant to feel as though they have their own dedicated network environment, akin to traditional cloud VPCs. Kube-OVN allows you to create VPC-like constructs in Kubernetes, where each tenant can have its own private network with separate subnets, routes, and policies.

Advanced Security Features

Native Kubernetes NetworkPolicies are limited in scope and lack some of the more advanced features needed for multi-tenant environments. Kube-OVN supports advanced security capabilities such as:

- Security groups and ACLs: Define ingress/egress rules at the network level to restrict traffic between Pods, subnets, or namespaces.
- Policy-based traffic control: Apply fine-grained policies to allow or deny traffic between tenants or external resources.

Tenant-Level Resource Allocation

In a shared cluster, ensuring fair and isolated resource allocation is critical to prevent tenant interference. With Kube-OVN, you can assign unique subnets and IP pools to specific tenants or namespaces. This ensures:

- No IP conflicts between tenants.
- Network traffic accounting and bandwidth control per tenant.

Logical Routing and NAT

Multi-tenancy often requires tenants to access shared external services while maintaining internal network isolation. Kube-OVN includes a built-in logical router to handle routing between subnets, VPCs, and external networks. It also supports NAT for accessing external services securely.

Cross-Tenant and External Connectivity

Tenants often need controlled access to shared databases, APIs, or legacy systems without compromising isolation. Kube-OVN provides static routes and BGP support, allowing tenants to connect securely to external networks, shared services, or other clusters.