

Multi-Tenant Kubernetes Platform

Studente: Giovanni Liguori

Corso: Cloud Platforms and Infrastructure as Code

[OVERVIEW](#)

[INSTALLAZIONE E SETUP](#)

[ARCHITETTURA DEL SISTEMA](#)

[SECURITY IMPLEMENTATION](#)

[HIGH AVAILABILITY E SCALING](#)

[MONITORING E OBSERVABILITY](#)

[DEMO](#)

[CONCLUSIONI](#)

OVERVIEW

Questa piattaforma implementa un ambiente Kubernetes multi-tenant con isolamento completo tra team di sviluppo.

L'architettura permette a più team di operare sullo stesso cluster mantenendo separazione delle risorse, isolamento di rete e visibilità limitata al proprio namespace. Ogni team dispone di quote di risorse definite, policy di sicurezza dedicate e monitoring isolato.

Soluzione:

1. **Isolamento Completo:** Ogni team opera nel proprio namespace dedicato con sicurezza RBAC, Network Policies e Pod Security Standards
2. **Resource Governance:** Quote di risorse definite e limiti per prevenire resource exhaustion
3. **Monitoring Isolato:** Dashboard e metriche separate per ogni namespace con observability dedicata
4. **Self-Service:** Team autonomi nei propri namespace per deployment e gestione
5. **Applicazioni Demo:** Frontend React + Backend Node.js + PostgreSQL per scenari realistici
6. **High Availability:** Auto-scaling (HPA) e Pod Disruption Budgets per zero downtime

Stack Tecnologico

Infrastructure:

- **Kind:** Kubernetes in Docker per ambiente di sviluppo locale
- **Calico:** Container Network Interface (CNI) per networking avanzato e Network Policies

Applications:

- **Frontend:** React.js 18 store demo (team-frontend namespace)
- **Backend:** Node.js 18 API con Express.js e metriche custom (team-backend namespace)
- **Database:** PostgreSQL 13 con dati demo e schema completo

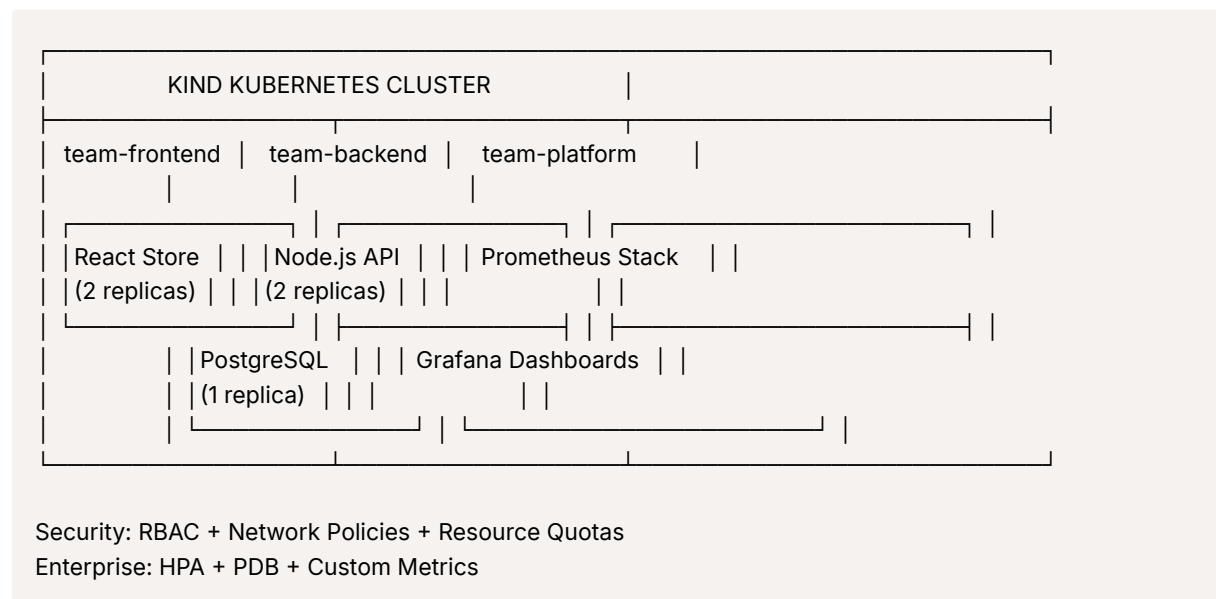
Security & Governance:

- **RBAC:** Role-Based Access Control per isolamento team
- **Network Policies:** Calico policies per isolamento rete tra namespace
- **Resource Quotas:** Limitazione uso risorse (CPU, memory, storage) per namespace
- **Pod Security Standards:** Policy di sicurezza `restricted` a livello pod

Monitoring & Observability:

- **Prometheus:** Raccolta metriche da applicazioni, database e infrastruttura
- **Grafana:** Dashboard dedicati per team e overview platform-wide
- **Custom Metrics:** Metriche applicative Node.js personalizzate
- **ServiceMonitors:** Configurazione automatica target Prometheus
- **Metrics Server:** Raccolta metriche CPU/Memory per HPA auto-scaling

Architettura High-Level



Cluster Nodes:

- **Control Plane:** 1 nodo (API Server, Scheduler, Controller Manager, etcd)
- **Worker Nodes:** 2 nodi (Kubelet, Container Runtime, Calico Node)

INSTALLAZIONE E SETUP

Prerequisiti

Software Richiesto:

```
bash
# Docker Desktop (macOS)
docker --version

brew install kind
kind --version

brew install kubectl
kubectl version --client

brew install helm
```

```
helm version
```

```
brew install jq
```

Risorse Hardware:

- **CPU:** Minimo 4 cores (consigliati 8 cores)
- **RAM:** Minimo 8GB (consigliati 16GB)
- **Storage:** Minimo 20GB spazio libero
- **OS:** macOS 12+ (Monterey o superiore)

Procedura di Installazione

Passo 1: Creazione Cluster Kind

```
bash
# 1. Creare cluster Kind con configurazione custom
kind create cluster --name multi-tenant --config kind-config.yaml

# 2. Verificare nodi del cluster
kubectl get nodes
# Atteso: 3 nodi (1 control-plane + 2 workers)
```

File di configurazione Kind:

```
yaml
# kind-config.yaml
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
networking:
  disableDefaultCNIL: true# Useremo Calico
  podSubnet: "10.244.0.0/16"
  serviceSubnet: "10.96.0.0/12"
nodes:
  - role: control-plane
  - role: worker
  - role: worker
```

Passo 2: Installazione Calico CNI

```
bash
# 1. Installare Calico per networking avanzato
kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/v3.26.0/manifests/calico.yaml

# 2. Attendere che Calico sia pronto
kubectl wait --for=condition=ready pod -l k8s-app=calico-node -n kube-system --timeout=300s
```

```
# 3. Verificare che i nodi siano Ready
kubectl get nodes
# Tutti i nodi devono essere "Ready"
```

Passo 3: Deploy Fondamenta Multi-Tenant

```
bash
# 1. Creare namespace per i team
kubectl apply -f kubernetes/01-namespaces/

# 2. Configurare identità e service accounts
kubectl apply -f kubernetes/01-namespaces/service-accounts.yaml

# 3. Implementare sicurezza (RBAC, Network Policies, Quotas)
kubectl apply -f kubernetes/02-security/

# 4. Verificare configurazione sicurezza
kubectl get namespaces | grep team-
kubectl get resourcequota -A
kubectl get networkpolicy -A
```

Passo 4: Deploy Applicazioni

```
bash
# 1. Build immagini Docker
cd applications/frontend/react-store-demo
docker build -t react-store:latest .
cd ../../../../

cd applications/backend/users-api
docker build -t users-api:latest .
cd ../../../../

# 2. Caricare immagini in Kind
kind load docker-image react-store:latest --name multi-tenant
kind load docker-image users-api:latest --name multi-tenant

# 3. Deploy workloads
kubectl apply -f kubernetes/03-workloads/backend/postgres-deployment.yaml
kubectl apply -f kubernetes/03-workloads/backend/users-api-deployment.yaml
kubectl apply -f kubernetes/03-workloads/backend/users-api-service.yaml
kubectl apply -f kubernetes/03-workloads/frontend/

# 4. Deploy funzionalità enterprise (scaling, HA)
kubectl apply -f kubernetes/04-scaling/
```

Passo 5: Setup Monitoring

```

bash
# 1. Preparare dashboard Grafana
kubectl create configmap grafana-platform-dashboards \
  --from-file=kubernetes/05-monitoring/grafana/dashboards/ \
  -n team-platform \
  --dry-run=client -o yaml | kubectl apply -f -

kubectl label configmap grafana-platform-dashboards \
  grafana_dashboard=1 -n team-platform --overwrite

# 2. Installare Prometheus Stack
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo update

helm install prometheus prometheus-community/kube-prometheus-stack \
  --namespace team-platform \
  --values kubernetes/05-monitoring/prometheus/prometheus-stack-values.yaml \
  --wait --timeout 10m

# 3. Configurare ServiceMonitors per metriche custom
kubectl apply -f kubernetes/05-monitoring/prometheus/backend-servicemonitor.yaml

```

Passo 6: Configurazione Accesso

```

bash
# Setup port-forwards per accesso locale
kubectl port-forward -n team-backend svc/users-api 3001:3000 &
kubectl port-forward -n team-frontend svc/react-store 3000:3000 &
kubectl port-forward svc/prometheus-grafana 3002:80 -n team-platform &
kubectl port-forward svc/prometheus-prometheus 9090:9090 -n team-platform &

```

Verifica Installazione

```

bash
# 1. Verificare tutti i pod sono Running
kubectl get pods -A | grep -v Running

# 2. Testare applicazioni
echo "🔧 Testing services..."
curl -s http://localhost:3001/api/health && echo "✅ Backend OK" || echo "❌ Backend KO"
curl -s http://localhost:3000 >/dev/null && echo "✅ Frontend OK" || echo "❌ Frontend KO"
curl -s http://localhost:9090/-/healthy >/dev/null && echo "✅ Prometheus OK" || echo "❌ Prometheus KO"
curl -s -u admin:admin123 http://localhost:3002/api/health >/dev/null && echo "✅ Grafana OK" || echo "❌ Grafana KO"

# 3. Verificare metriche custom
curl -s http://localhost:3001/metrics | grep http_requests_total

```

```
# 4. Testare database
curl -s http://localhost:3001/api/products | jq '.metadata.total_products'

# 5. Generare traffico iniziale per dashboard
for i in {1..20}; do
  curl -s http://localhost:3001/api/products >/dev/null
  curl -s http://localhost:3001/api/categories >/dev/null
  sleep 0.5
done

# 6. Controllare isolamento sicurezza
kubectl auth can-i get pods --namespace=team-backend --as=system:serviceaccount:team-frontend:team-frontend-sa
# Risultato atteso: no
```

URLs di Accesso

Applicazioni:

- **Frontend:** <http://localhost:3000>
- **Backend API:** <http://localhost:3001>
- **API Health:** <http://localhost:3001/api/health>
- **API Metrics:** <http://localhost:3001/metrics>

Monitoring:

- **Grafana:** <http://localhost:3002> (admin / admin123)
- **Prometheus:** <http://localhost:9090>

API Endpoints per Demo:

- `GET /api/products` - Lista prodotti dal database
- `GET /api/categories` - Categorie prodotti con conteggi
- `GET /api/server-info` - Info load balancing (hostname pod)
- `GET /api/db-test` - Test connessione database
- `GET /metrics` - Metriche Prometheus custom

CLEANUP E RIMOZIONE

Rimozione Completa

```
bash
# 1. Fermare tutti i port-forwards
pkill -f 'port-forward'

# 2. Rimuovere monitoring stack Helm
helm uninstall prometheus -n team-platform
```

```
# 3. Eliminare cluster Kind
kind delete cluster --name multi-tenant

# 4. Pulire immagini Docker (opzionale)
docker rmi react-store:latest users-api:latest
docker system prune -f

echo "✅ Cleanup completo eseguito"
```

Docker storage cleanup:

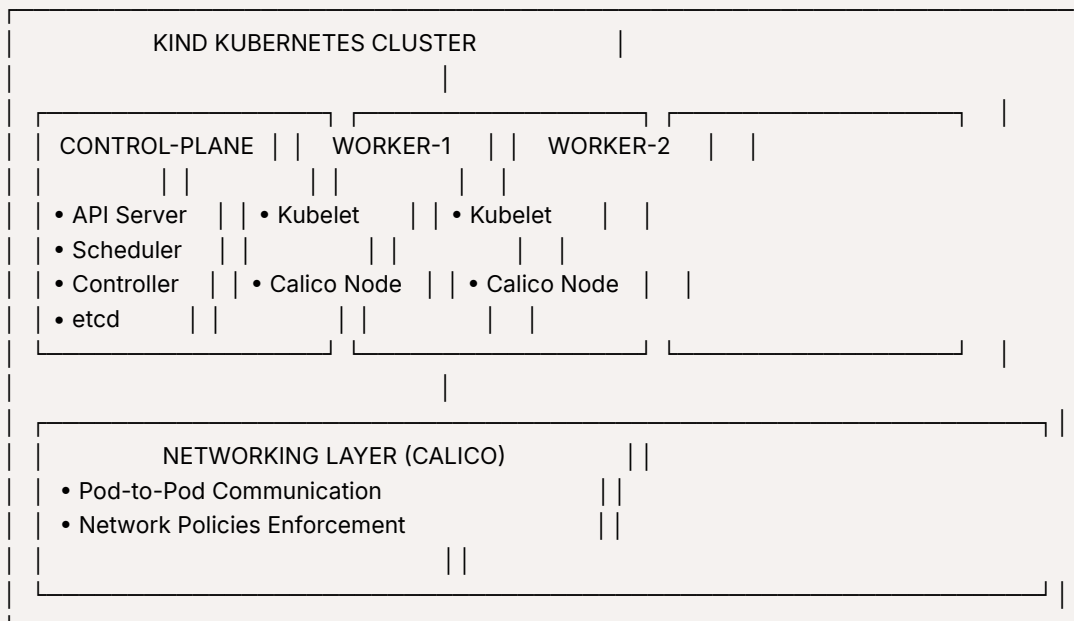
```
bash
# Cleanup completo Docker
docker system df# Mostra uso spazio# Remove tutto unused
docker system prune -a -f --volumes

# Remove Kind specific
docker volume ls | grep kind
docker volume rm $(docker volume ls -q | grep kind)
```

ARCHITETTURA DEL SISTEMA

Panoramica Architetture

L'architettura del sistema è progettata seguendo il **pattern multi-tenant** con **isolamento a più livelli**:



Struttura Namespace

Il cluster è organizzato in **tre namespace principali**:

team-frontend Namespace

```

yaml
Namespace: team-frontend
├── Scopo: Applicazioni frontend e interfacce utente
├── Workloads:
│   ├── Deployment: react-store (2 replicas)
│   └── Service: react-store (ClusterIP)
├── Security:
│   ├── ServiceAccount: team-frontend-sa
│   ├── Role: team-frontend-role
│   ├── ResourceQuota: CPU=4, Memory=8Gi, Pods=50
│   └── NetworkPolicy: isolamento da altri team
└── Monitoring:
    └── Dashboard Grafana dedicato frontend

```

Caratteristiche specifiche:

- **Pod Security Standard:** `restricted`
- **Resource Limits:** Container max 2 CPU, 4Gi RAM
- **Network Access:** Internet HTTPS/HTTP, DNS, stesso namespace
- **RBAC:** Accesso completo solo al proprio namespace

team-backend Namespace

```

yaml
Namespace: team-backend
├── Scopo: API, microservizi, database
├── Workloads:
│   ├── Deployment: users-api (2 replicas)
│   ├── Deployment: postgresql (1 replica)
│   ├── Service: users-api (ClusterIP)
│   └── Service: postgresql (ClusterIP)
├── Security:
│   ├── ServiceAccount: team-backend-sa
│   ├── Role: team-backend-role
│   ├── ResourceQuota: CPU=6, Memory=12Gi, Pods=75
│   └── NetworkPolicy: isolamento + accesso database
└── Monitoring:
    ├── ServiceMonitor: users-api metrics
    └── Dashboard Grafana backend + database

```

Caratteristiche specifiche:

- **Pod Security Standard:** `restricted`
- **Resource Limits:** Container max 4 CPU, 8Gi RAM
- **Network Access:** Solo HTTPS esterno, database interno
- **Custom Metrics:** Node.js app metrics + PostgreSQL stats

team-platform Namespace


```

yaml
Namespace: team-platform
├── Scopo: Infrastruttura, monitoring, gestione piattaforma
├── Workloads:
│   ├── StatefulSet: prometheus-prometheus
│   ├── Deployment: prometheus-grafana
│   ├── DaemonSet: node-exporter
│   └── Deployment: kube-state-metrics
├── Security:
│   ├── ServiceAccount: team-platform-sa
│   ├── ClusterRole: team-platform-cluster-role (admin)
│   ├── ResourceQuota: CPU=8, Memory=16Gi, Pods=100
│   └── NetworkPolicy: accesso cross-namespace per monitoring
└── Monitoring:
    ├── Dashboard: Platform Overview
    ├── Dashboard: Per-Team Dashboards
    └── Alerting Rules

```

Caratteristiche specifiche:

- **Pod Security Standard:** `privileged` (per monitoring tools)
- **Cluster Permissions:** Full admin access per operations
- **Cross-Namespace Access:** Monitoring di tutti i namespace
- **High Resource Allocation:** Per stack monitoring enterprise

Service Accounts

Ogni namespace dispone di **ServiceAccount dedicati** per l'identità e autenticazione delle applicazioni:

team-frontend ServiceAccount

```

yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: team-frontend-sa
  namespace: team-frontend
labels:
  team: frontend
  component: identity
annotations:
  description: "ServiceAccount per applicazioni frontend e RBAC"
  kubernetes.io/managed-by: "platform-team"
automountServiceAccountToken: true

```

Caratteristiche:

- **Scope:** Solo namespace team-frontend
- **Permissions:** Read/Write su risorse frontend
- **Token:** Auto-mount per pod authentication

- **Usage:** Assegnato ai pod React store

team-backend ServiceAccount

```

yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: team-backend-sa
  namespace: team-backend
labels:
  team: backend
  component: identity
annotations:
  description: "ServiceAccount per API backend e database"
  kubernetes.io/managed-by: "platform-team"
automountServiceAccountToken: true

```

Caratteristiche:

- **Scope:** Solo namespace team-backend
- **Permissions:** Full access su API e database resources
- **Database Access:** Credentials per PostgreSQL
- **Usage:** Assegnato ai pod Node.js API e PostgreSQL

team-platform ServiceAccount

```

yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: team-platform-sa
  namespace: team-platform
labels:
  team: platform
  component: identity
annotations:
  description: "ServiceAccount per operazioni infrastruttura"
  privilege-level: "cluster-admin"
  kubernetes.io/managed-by: "platform-team"
automountServiceAccountToken: true

```

Caratteristiche:

- **Scope:** Cluster-wide access
- **Permissions:** Cluster admin per monitoring e operations
- **Cross-namespace:** Accesso a tutti i namespace per monitoring
- **Usage:** Prometheus, Grafana, platform tools

Isolation Matrix

Risorsa	team-frontend	team-backend	team-platform
Pods (proprio NS)	✓ Full Access	✓ Full Access	✓ Full Access
Pods (altri NS)	✗ No Access	✗ No Access	✓ Read Access
Services (proprio NS)	✓ Full Access	✓ Full Access	✓ Full Access
Secrets (proprio NS)	✓ Full Access	✓ Full Access	✓ Full Access
Secrets (altri NS)	✗ No Access	✗ No Access	✓ Read Access
NetworkPolicies	✓ Manage Own	✓ Manage Own	✓ Manage All
ResourceQuotas	✗ View Only	✗ View Only	✓ Full Management
Cluster Resources	✗ No Access	✗ No Access	✓ Full Access

Frontend: React Store Demo

Tecnologie:

- **Framework:** React 18 con Hooks
- **Build Tool:** Create React App
- **Container:** Node.js 18 Alpine

Funzionalità:

- **Product Catalog:** Lista prodotti da database
- **Category Filter:** Filtro per categoria prodotti
- **Load Balancing Test:** Button per testare distribuzione carico
- **Real-time Stats:** Connessione backend per demo

Sicurezza:

- **Non-root user:** UID 1000
- **Read-only filesystem:** Dove possibile
- **Resource limits:** 200m CPU, 1Gi RAM request

Networking:

- **Service Type:** ClusterIP (internal)
- **Port:** 3000 (HTTP)

Backend: Node.js API

Tecnologie:

- **Runtime:** Node.js 18
- **Framework:** Express.js
- **Database Client:** pg (PostgreSQL)
- **Metrics:** Custom Prometheus metrics

API Endpoints:

```
GET /                # Service info
GET /api/health      # Health check
GET /api/products    # Lista prodotti
GET /api/categories  # Categorie con conteggi
GET /api/db-test     # Test connessione DB
```

```
GET /api/server-info    # Info pod (load balancing)
GET /metrics            # Prometheus metrics
```

Database: PostgreSQL

Configurazione:

- **Version:** PostgreSQL 13 Alpine
- **Security:** Non-root user, resource limits

Schema Database:

```
sql
Table: products
├── id (SERIAL PRIMARY KEY)
├── name (VARCHAR(255))
├── description (TEXT)
├── price (DECIMAL(10,2))
├── category (VARCHAR(100))
├── stock_quantity (INTEGER)
├── created_at (TIMESTAMP)
└── updated_at (TIMESTAMP)

Indexes:
├── idx_products_category (category)
├── idx_products_price (price)
└── idx_products_stock (stock_quantity)
```

Dati di Esempio:

- **Electronics:** iPhone 15 Pro, Samsung Galaxy S24
- **Computers:** MacBook Pro M3, Dell XPS 13
- **Audio:** AirPods Max, Sony WH-1000XM5
- **Gaming:** PlayStation 5, Xbox Series X, Nintendo Switch

SECURITY IMPLEMENTATION

La sicurezza è implementata attraverso **quattro layer principali**:

SECURITY LAYERS	
🔒 RBAC (Role-Based Access Control)	
• Namespace-scoped permissions	
• ServiceAccount isolation	
🚧 Network Policies (Calico)	
• Default deny all traffic	
• Whitelist-based communication	
💰 Resource Quotas & LimitRanges	
• CPU/Memory limits per namespace	

<ul style="list-style-type: none"> • Pod count restrictions • Storage allocation limits 	
<ul style="list-style-type: none"> • Pod Security Standards • Non-root containers • No privileged escalation 	

RBAC (Role-Based Access Control)

Il sistema RBAC implementa il **principio del least privilege** con tre livelli di accesso:

Permission Matrix

Team	Scope	Resources	Verbs
Frontend	team-frontend only	Pods, services, deployments, configmaps, secrets	get, list, create, update, delete
Backend	team-backend only	Pods, services, deployments, statefulsets, jobs	get, list, create, update, delete
Platform	Cluster	ALL resources	ALL verbs

Implementazione

Frontend Team:

```
# Role (namespace-scoped) → RoleBinding → ServiceAccount
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: team-frontend-role
  namespace: team-frontend
rules:
- apiGroups: ["", "apps", "networking.k8s.io", "autoscaling"]
  resources: ["pods", "services", "deployments", "networkpolicies", "hpa"]
  verbs: ["get", "list", "create", "update", "patch", "delete"]
```

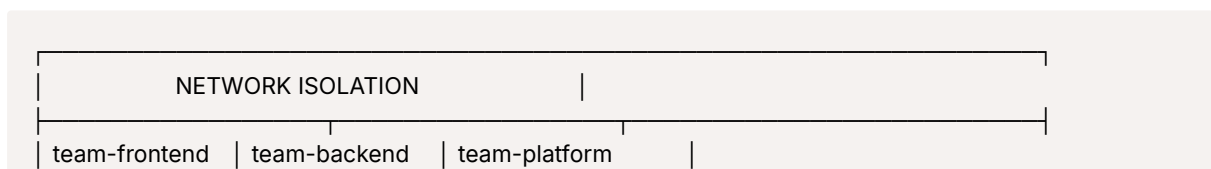
Platform Team:

```
# ClusterRole per accesso cluster-wide
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: team-platform-cluster-role
rules:
- apiGroups: ["*"]
  resources: ["*"]
  verbs: ["*"]
```

Network Policies

Implementano **microsegmentazione** con strategia **"default deny all"** + whitelist esplicite.

Policy Architecture



DEFAULT DENY	DEFAULT DENY	FULL ACCESS	
↓ WHITELIST:	↓ WHITELIST:	(monitoring needs)	
• Own namespace	• Own namespace		
• DNS (k8s)	• DNS (k8s)		
• Internet HTTPS	• Internet HTTPS		
• Monitoring	• Monitoring		

Key Policies

Default Deny All:

```
# Blocca tutto il traffico per sicurezza
spec:
  podSelector: {} # Tutti i pod
  policyTypes: [Ingress, Egress]
# Nessuna regola = blocco totale
```

Intra-Namespace Communication:

```
# Permette comunicazione stesso namespace
ingress:
- from:
  - namespaceSelector:
      matchLabels:
        team: frontend # Solo stesso team
```

DNS Access:

```
# Necessario per service discovery
egress:
- to:
  - namespaceSelector:
      matchLabels:
        name: kube-system
  ports:
  - protocol: UDP
    port: 53
```

Platform Monitoring:

```
# Eccezione per monitoring cross-namespace
ingress:
- from:
  - namespaceSelector:
      matchLabels:
        team: platform
  ports:
  - protocol: TCP
    port: 8080 # Metrics
```

Resource Quotas & LimitRanges

Garantiscono **fair sharing** e prevengono resource exhaustion.

Quota Allocation

Namespace	CPU Request	CPU Limit	Memory Request	Memory Limit	Pods	Storage
team-frontend	4 cores	8 cores	8Gi	16Gi	50	100Gi
team-backend	6 cores	12 cores	12Gi	24Gi	75	200Gi
team-platform	8 cores	16 cores	16Gi	32Gi	100	500Gi

Implementation

ResourceQuota Example:

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: team-frontend-quota
  namespace: team-frontend
spec:
  hard:
    requests.cpu: "4"
    limits.cpu: "8"
    requests.memory: 8Gi
    limits.memory: 16Gi
    pods: "50"
    persistentvolumeclaims: "10"
```

LimitRange Example:

```
apiVersion: v1
kind: LimitRange
metadata:
  name: team-frontend-limits
  namespace: team-frontend
spec:
  limits:
    - type: Container
      default:      # Se non specificato
        cpu: 200m
        memory: 256Mi
      defaultRequest: # Richiesta minima
        cpu: 100m
        memory: 128Mi
      max:          # Limite massimo per container
        cpu: "2"
        memory: 4Gi
```

Pod Security Standards

Enforcing **security baselines** a livello pod per prevenire privilege escalation.

- **Non-root user:** `runAsNonRoot: true`
- **No privilege escalation:** `allowPrivilegeEscalation: false`



- **Drop capabilities:** `capabilities.drop: [ALL]`
- **Read-only filesystem:** `readOnlyRootFilesystem: true`

Controllo di Sicurezza Implementazione Copertura Stato

Controllo di Sicurezza	Implementazione	Copertura	Stato
Isolamento Identità	ServiceAccounts + RBAC	Permessi limitati ai namespace	✓ Attivo
Segmentazione Rete	Calico Network Policies	Default deny + accesso selettivo	✓ Attivo
Governance Risorse	ResourceQuotas + LimitRanges	Limiti CPU/Memory/Storage	✓ Attivo
Sicurezza Container	Pod Security Standards	Non-root + no escalation	✓ Attivo
Gestione Privilegi	Security context restricted	Rimozione capabilities	✓ Attivo
Audit Trail	Kubernetes API audit logs	Tracciamento violazioni RBAC	✓ Disponibile
Accesso Monitoring	Eccezioni team platform	Osservabilità cross-namespace	✓ Attivo

HIGH AVAILABILITY E SCALING

La piattaforma implementa **meccanismi** per garantire alta disponibilità e scaling automatico:

ENTERPRISE FEATURES	
 HORIZONTAL POD AUTOSCALER (HPA) <ul style="list-style-type: none"> • Scaling automatico basato su metriche • CPU e Memory utilization triggers 	
 POD DISRUPTION BUDGET (PDB) <ul style="list-style-type: none"> • Disponibilità garantita durante maintenance • Zero downtime rolling updates 	

Horizontal Pod Autoscaler (HPA)

Per garantire il corretto funzionamento dell'**HPA (Horizontal Pod Autoscaler)**, è fondamentale installare il **Metrics Server**.

Il problema senza Metrics Server si manifesta immediatamente quando si tenta di verificare lo stato dell'HPA:

```
bash
kubectl get hpa -n team-backend
# Output: cpu: <unknown>/60%, memory: <unknown>/70%# L'HPA NON può scalare automaticamente
```

In questa situazione, l'HPA non può prendere alcuna decisione di scaling perché non ha visibilità sull'utilizzo effettivo delle risorse dei pod

Installazione Metrics Server (obbligatoria):

```
bash
# Installa Metrics Server
kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.
```


yaml

```
# Configurazione per Kind (bypass TLS)
kubectl patch deployment metrics-server -n kube-system --type='json' -p='[
{
  "op": "add",
  "path": "/spec/template/spec/containers/0/args/-",
  "value": "--kubelet-insecure-tls"
}
]'
```

Verifica funzionamento

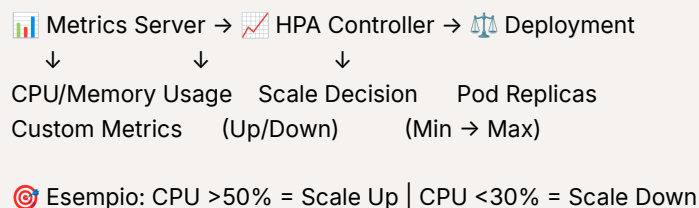
```
kubectl wait --for=condition=ready pod -l k8s-app=metrics-server -n kube-system --timeout=60s
kubectl top nodes
kubectl top pods -n team-backend
```

Con Metrics Server funzionante:

```
bash
kubectl get hpa -n team-backend
# Output: cpu: 45%/60%, memory: 32%/70%# L'HPA può ora scalare automaticamente basandosi su metriche reali
```

L'HPA garantisce **scaling automatico** basato su metriche CPU e memoria, adattando la capacità al carico.

Architettura HPA



Configurazione HPA

Namespace	Target	Min Replicas	Max Replicas	CPU Threshold	Memory Threshold
team-frontend	react-store	2	5	50%	60%
team-backend	users-api	2	4	60%	70%

Behavior Policies:

- **Scale Up:** Veloce (60s stabilization)
- **Scale Down:** Graduale (300s stabilization)

Esempio Configurazione HPA

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: react-store-hpa
```

```

namespace: team-frontend
spec:
  scaleTargetRef:
    kind: Deployment
    name: react-store
  minReplicas: 2
  maxReplicas: 5
  metrics:
  - type: Resource
    resource:
      name: cpu
    target:
      type: Utilization
      averageUtilization: 50

```

Pod Disruption Budget (PDB)

I PDB garantiscono **disponibilità continua** durante maintenance e rolling updates.

PDB Strategy

Normal State: [Pod1] [Pod2] [Pod3]



During Maintenance: [Pod1] [---] [Pod3]



PDB Guarantee: Minimum 1 pod always available

PDB Configuration

Service	Min Available	Impact
react-store	1 pod	Zero downtime UX
users-api	1 pod	API sempre disponibile
postgresql	N/A	

Esempio PDB:

```

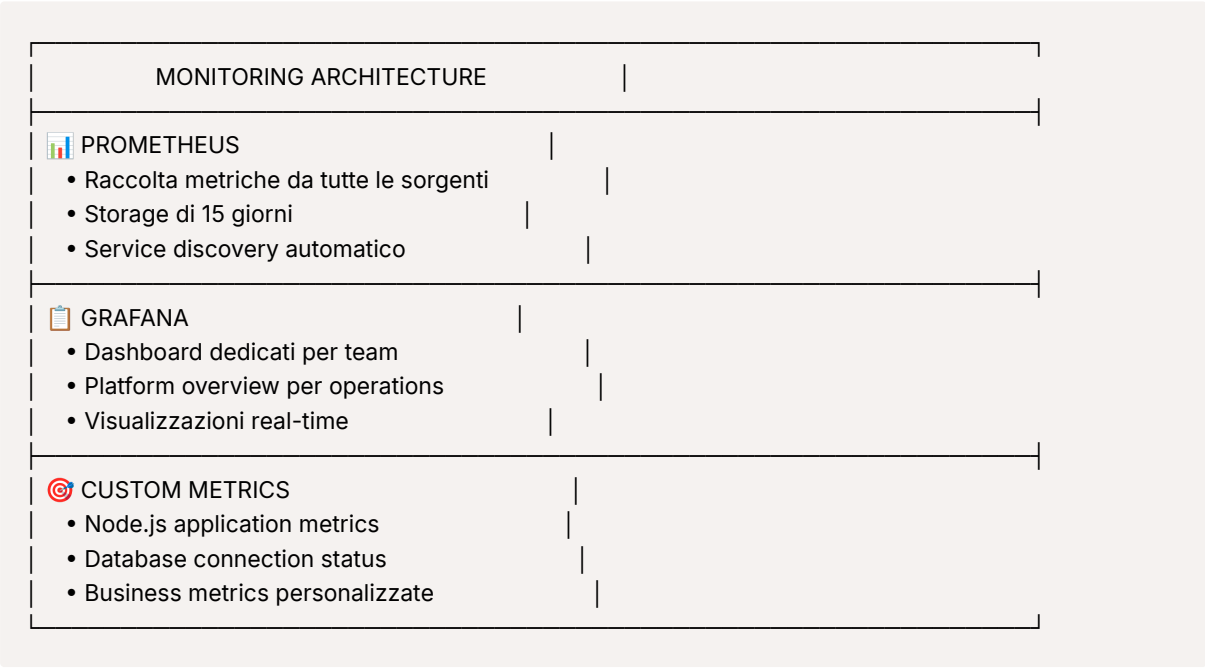
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: react-store-pdb
  namespace: team-frontend
spec:
  minAvailable: 1
  selector:
    matchLabels:
      app: react-store

```

MONITORING E OBSERVABILITY

Panoramica Stack di Monitoring

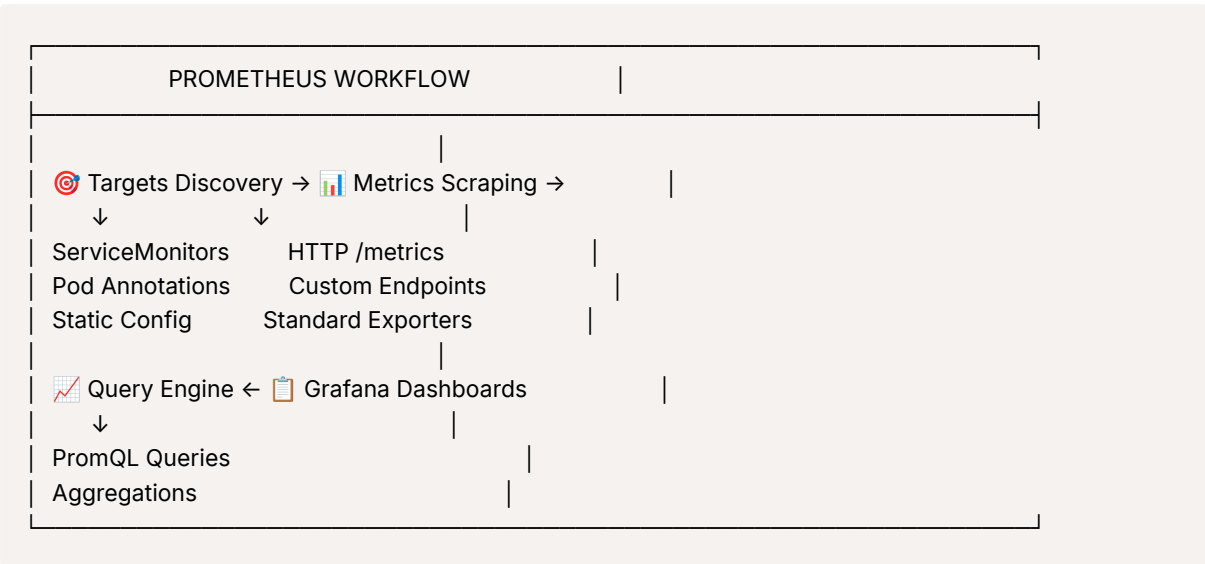
La piattaforma implementa un **sistema di observability completo** per garantire visibilità su performance, utilizzo risorse e health dei servizi:



Prometheus Stack

Prometheus fornisce i mezzi **per la raccolta metriche**.

Architettura Prometheus



Configurazione Stack

Componente	Versione	Namespace	Storage
Prometheus	v2.45+	team-platform	10GB
Grafana	v10.0+	team-platform	Emptydir
Node Exporter	v1.6+	kube-system	N/A
Kube State Metrics	v2.9+	team-platform	N/A

Helm Installation

```
# Installazione via Helm con values custom
helm install prometheus prometheus-community/kube-prometheus-stack \
  --namespace team-platform \
  --values prometheus-stack-values.yaml \
  --wait --timeout 10m
```

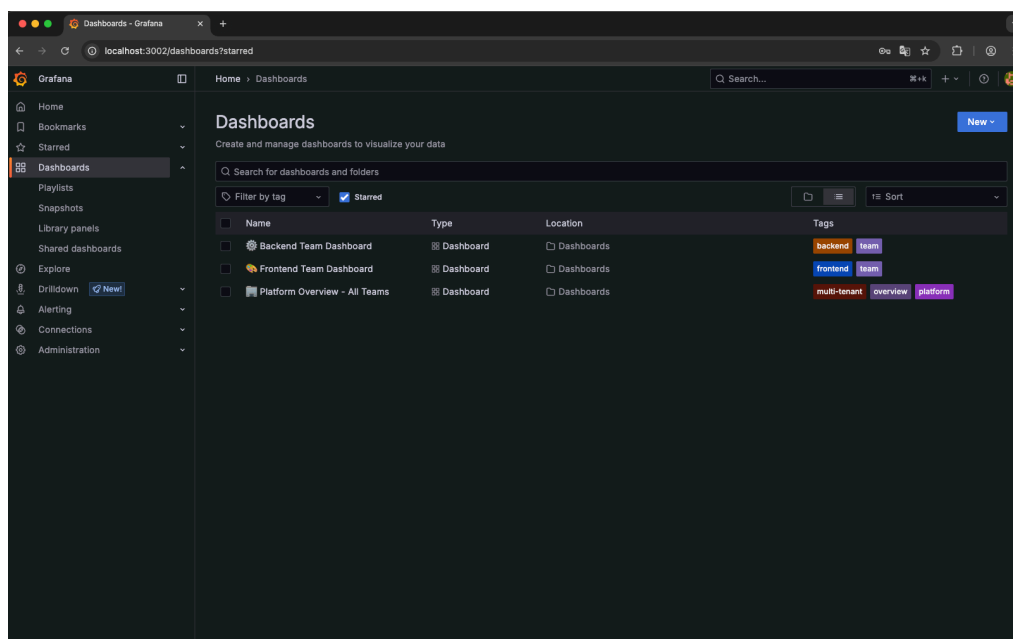
Configurazione Values Key:

```
prometheus:
prometheusSpec:
  retention: 15d
  retentionSize: 10GB
resources:
  requests:
    cpu: 200m
    memory: 512Mi
  limits:
    cpu: 1000m
    memory: 2Gi

grafana:
  adminUser: admin
  adminPassword: admin123
sidecar:
  dashboards:
    enabled: true
    label: grafana_dashboard
```

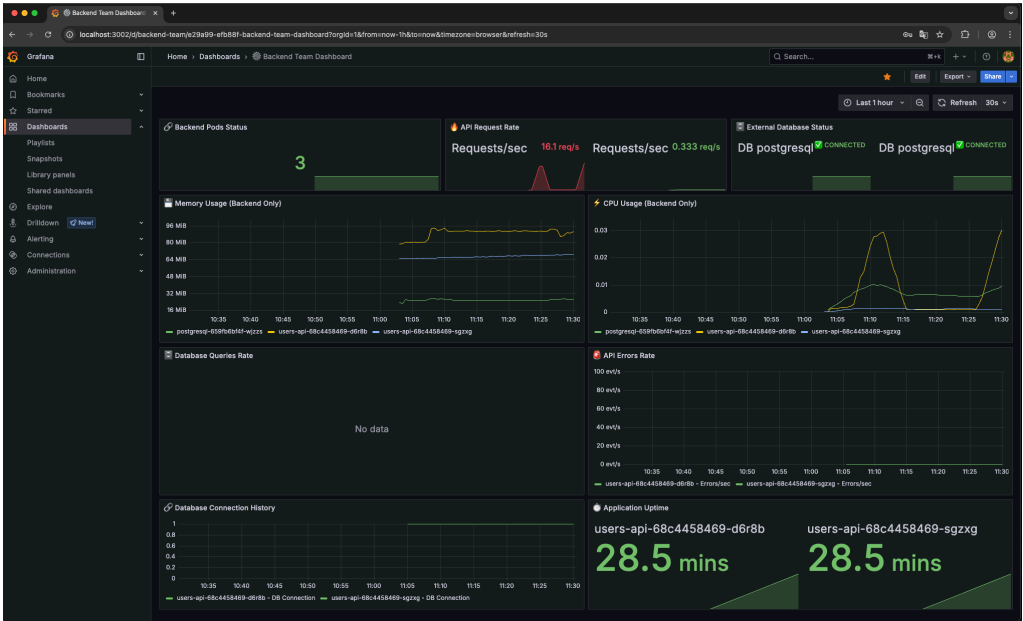
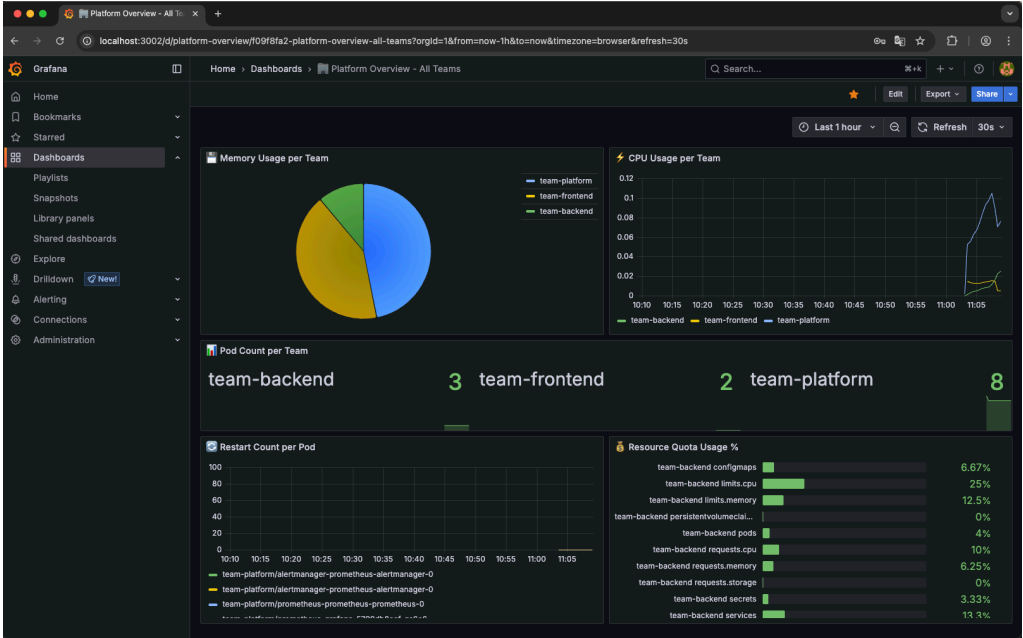
Grafana Dashboards

Grafana fornisce **visualizzazioni dedicate** per ogni team e overview platform-wide.



Dashboard Overview

Dashboard	Target Audience	Key Metrics	Update Frequency
Platform Overview	Platform Team, Management	Cross-namespace resources, costs	30s
Frontend Team	Frontend Developers	React performance, UX metrics	30s
Backend Team	Backend Developers	API metrics, DB performance	30s



Dashboard Configuration

```
# ConfigMap per dashboard auto-loading
apiVersion: v1
kind: ConfigMap
metadata:
  name: grafana-platform-dashboards
```

```

namespace: team-platform
labels:
  grafana_dashboard: "1" # Auto-discovery label
data:
  platform-overview.json: |
    {
      "dashboard": {
        "title": "📊 Platform Overview",
        "panels": [...]
      }
    }
  }
}

```

Dashboard Deployment:

```

# Auto-discovery tramite sidecar
kubectl create configmap grafana-platform-dashboards \
  --from-file=dashboards/ \
  -n team-platform

kubectl label configmap grafana-platform-dashboards \
  grafana_dashboard=1 -n team-platform

```

Custom Metrics

Le applicazioni espongono **metriche personalizzate** per monitoring approfondito.

Node.js Application Metrics

Il backend Node.js implementa **metriche custom** usando il pattern Prometheus standard:

Metric Name	Type	Description	Labels
<code>http_requests_total</code>	Counter	Total HTTP requests	<code>namespace</code> , <code>pod</code> , <code>service</code>
<code>db_queries_total</code>	Counter	Database queries executed	<code>namespace</code> , <code>pod</code> , <code>database</code>
<code>api_errors_total</code>	Counter	API errors count	<code>namespace</code> , <code>pod</code> , <code>service</code>
<code>db_connection_status</code>	Gauge	Database connection health	<code>namespace</code> , <code>pod</code> , <code>database</code> , <code>host</code>
<code>app_uptime_seconds</code>	Gauge	Application uptime	<code>namespace</code> , <code>pod</code> , <code>service</code>

Metrics Implementation

Metrics Collection nel Codice:

```

// Custom metrics tracking
let metrics = {
  http_requests_total: 0,
  db_queries_total: 0,
  api_errors_total: 0,
  db_connection_status: 1, // 1=connected, 0=disconnected
  startup_time: Date.now()
};

// Endpoint /metrics per Prometheus
app.get('/metrics', (req, res) => {
  const uptime = Math.floor((Date.now() - metrics.startup_time) / 1000);

```

```

const prometheusMetrics = `
# HELP http_requests_total Total HTTP requests
# TYPE http_requests_total counter
http_requests_total{namespace="team-backend",pod="${hostname}",service="users-api"} ${metrics.http
_requests_total}

# HELP db_connection_status Database connection status
# TYPE db_connection_status gauge
db_connection_status{namespace="team-backend",pod="${hostname}",database="postgresql"} ${metric
s.db_connection_status}
`;

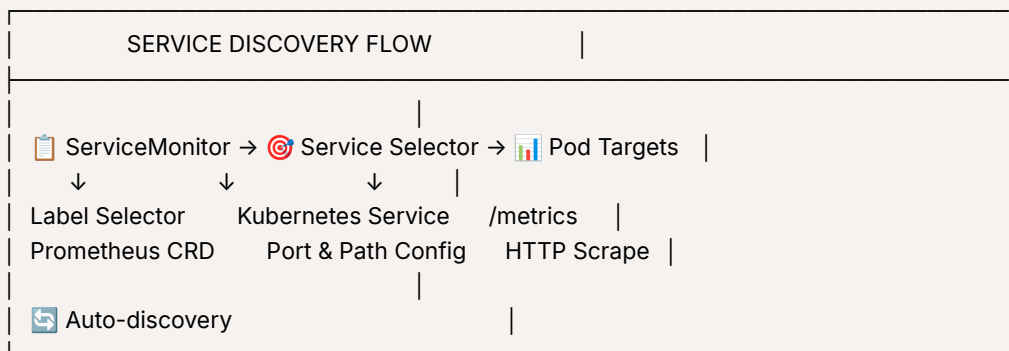
res.set('Content-Type', 'text/plain');
res.send(prometheusMetrics);
});

```

Service Discovery

Prometheus utilizza **ServiceMonitors** per auto-discovery delle metriche senza configurazione manuale.

ServiceMonitor Architecture



ServiceMonitor Configuration

Backend API ServiceMonitor:

```

apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: users-api-metrics
  namespace: team-backend
labels:
  app: users-api
  team: backend
  release: prometheus # Required per discovery
spec:
  selector:
    matchLabels:
      app: users-api # Target service labels
  endpoints:

```

```
- port: http
  path: /metrics
  interval: 30s
  scrapeTimeout: 10s
```

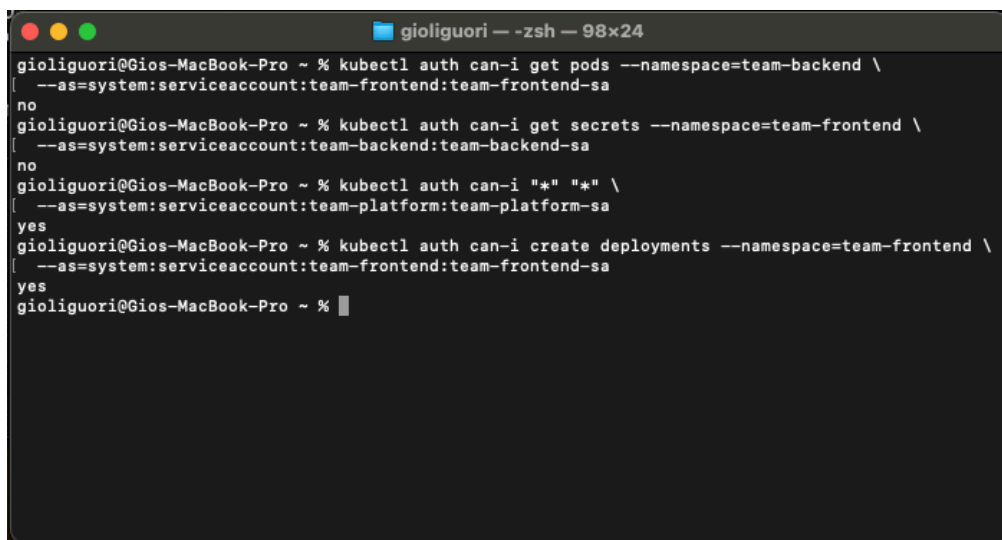
Service con Annotation:

```
apiVersion: v1
kind: Service
metadata:
  name: users-api
  namespace: team-backend
  annotations:
    prometheus.io/scrape: "true"
    prometheus.io/port: "3000"
    prometheus.io/path: "/metrics"
spec:
  ports:
    - port: 3000
      name: http # Nome importante per ServiceMonitor
  selector:
    app: users-api
```

DEMO

Demo 1: RBAC Isolation

Obiettivo: Dimostrare isolamento completo tra team tramite RBAC



```
gioliguori@Gios-MacBook-Pro ~ % kubectl auth can-i get pods --namespace=team-backend \
[ --as=system:serviceaccount:team-frontend:team-frontend-sa ]
no
gioliguori@Gios-MacBook-Pro ~ % kubectl auth can-i get secrets --namespace=team-frontend \
[ --as=system:serviceaccount:team-backend:team-backend-sa ]
no
gioliguori@Gios-MacBook-Pro ~ % kubectl auth can-i "*" "*" \
[ --as=system:serviceaccount:team-platform:team-platform-sa ]
yes
gioliguori@Gios-MacBook-Pro ~ % kubectl auth can-i create deployments --namespace=team-frontend \
[ --as=system:serviceaccount:team-frontend:team-frontend-sa ]
yes
gioliguori@Gios-MacBook-Pro ~ %
```

Codice di Riferimento

File: [kubernetes/02-security/rbac.yaml](#)

```
yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
```



```

metadata:
  name: team-frontend-role
  namespace: team-frontend
rules:
- apiGroups: ["", "apps"]
  resources: ["pods", "services", "deployments"]
  verbs: ["get", "list", "create", "update", "delete"]

---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
subjects:
- kind: ServiceAccount
  name: team-frontend-sa
  namespace: team-frontend
roleRef:
  kind: Role
  name: team-frontend-role

```

Conclusioni

- **Isolamento perfetto** tra team
- **Platform team** ha accesso cluster-wide
- **Self-service** funziona (team gestiscono proprio namespace)

Demo 2: Network Policy Isolation

Obiettivo: Dimostrare isolamento di rete tra namespace

```

kubectl apply -f test-network-isolation.yaml
kubectl wait --for=condition=ready pod network-test-frontend -n team-frontend --timeout=60s

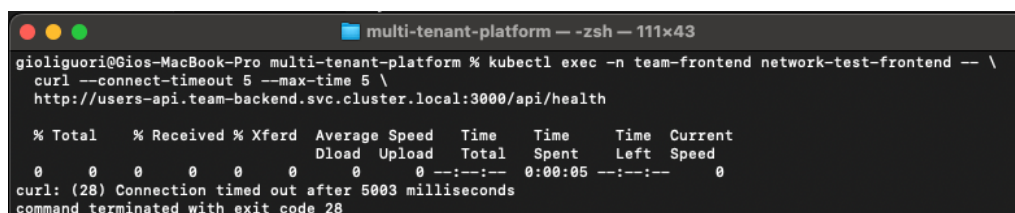
```

Network Policy blocca accesso cross-namespace

```

kubectl exec -n team-frontend network-test-frontend -- \
  curl --connect-timeout 5 --max-time 5 \
  http://users-api.team-backend.svc.cluster.local:3000/api/health

```



```

multi-tenant-platform --zsh -- 111x43
gioliguori@Gios-MacBook-Pro multi-tenant-platform % kubectl exec -n team-frontend network-test-frontend -- \
  curl --connect-timeout 5 --max-time 5 \
  http://users-api.team-backend.svc.cluster.local:3000/api/health

```

% Total	% Received	% Xferd	Average Speed	Time	Time	Time	Current
			Dload Upload	Total	Spent	Left	Speed
0	0	0	0	0	0	0	0

```

curl: (28) Connection timed out after 5003 milliseconds
command terminated with exit code 28

```

Comunicazione intra-namespace permessa (HTML response ricevuto)

```

kubectl exec -n team-frontend network-test-frontend -- \
  curl -s http://react-store.team-frontend.svc.cluster.local:3000

```

```
multi-tenant-platform --zsh -- 111x43
gioliguori@Gios-MacBook-Pro multi-tenant-platform % kubectl exec -n team-frontend network-test-frontend -- \
curl --connect-timeout 5 --max-time 5 \
http://react-store.team-frontend.svc.cluster.local:3000
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload  Total   Spent    Left     Speed
100  503    100  503    0    0  72666    0 --:--:-- --:--:-- --:--:-- 83833
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta name="description" content="Multi-Tenant Kubernetes Platform Demo" />
    <title>Multi-Tenant Store Demo</title>
    <script defer src="/static/js/bundle.js"></script></head>
    <body>
      <noscript>You need to enable JavaScript to run this app.</noscript>
      <div id="root"></div>
    </body>
  </html>
```

DNS resolution funziona correttamente per servizi autorizzati

```
kubectl exec -n team-frontend network-test-frontend -- \
nslookup react-store.team-frontend.svc.cluster.local
```

```
kubectl exec -n team-frontend network-test-frontend -- \
nslookup users-api.team-backend.svc.cluster.local
```

```
multi-tenant-platform --zsh -- 97x43
gioliguori@Gios-MacBook-Pro multi-tenant-platform % kubectl exec -n team-frontend network-test-frontend -- \
nslookup react-store.team-frontend.svc.cluster.local
;; Got recursion not available from 10.96.0.10
;; Got recursion not available from 10.96.0.10
;; Got recursion not available from 10.96.0.10
;; Got recursion not available from 10.96.0.10
Server:      10.96.0.10
Address:     10.96.0.10#53

Name:   react-store.team-frontend.svc.cluster.local
Address: 10.103.185.24
;; Got recursion not available from 10.96.0.10
```

```
06-testing --zsh -- 80x24
gioliguori@Gios-MacBook-Pro 06-testing % kubectl exec -n team-frontend network-test-frontend -- \
nslookup users-api.team-backend.svc.cluster.local
;; Got recursion not available from 10.96.0.10
;; Got recursion not available from 10.96.0.10
;; Got recursion not available from 10.96.0.10
;; Got recursion not available from 10.96.0.10
Server:      10.96.0.10
Address:     10.96.0.10#53

Name:   users-api.team-backend.svc.cluster.local
Address: 10.108.71.40
;; Got recursion not available from 10.96.0.10
gioliguori@Gios-MacBook-Pro 06-testing %
```

Test Accesso Internet

```
kubectl exec -n team-frontend network-test-frontend -- timeout 10 curl -I https://google.com
kubectl exec -n team-backend network-test-backend -- timeout 10 curl -I https://google.com
```

```
kubectl exec -n team-frontend network-test-frontend -- timeout 10 curl -I http://httpbin.org/get || echo "Block"
kubectl exec -n team-backend network-test-backend -- timeout 10 curl -I http://httpbin.org/get || echo "Block"
```

```

06-testing -- zsh -- 104x31
gioliguori@Gios-MacBook-Pro 06-testing % kubectl exec -n team-frontend network-test-frontend -- timeout 10 curl -I https://google.com
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left   Speed
0         0     0    0    0    0     0      0  --:--:-- --:--:-- --:--:--    0
location: https://www.google.com/
content-type: text/html; charset=UTF-8
content-security-policy-report-only: object-src 'none';base-uri 'self';script-src 'nonce-700ZeV9ta8ksrJ K03cM2Q' 'strict-dynamic' 'report-sample' 'unsafe-eval' 'unsafe-inline' https: http:;report-uri https:// csp.withgoogle.com/csp/gws/other-hp
date: Mon, 07 Jul 2025 14:55:26 GMT
expires: Wed, 06 Aug 2025 14:55:26 GMT
cache-control: public, max-age=2592000
server: gws
content-length: 220
x-xss-protection: 0
x-frame-options: SAMEORIGIN
alt-svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000

0 220 0 0 0 0 0 0 --:--:-- --:--:-- --:--:-- 0
gioliguori@Gios-MacBook-Pro 06-testing %

```

```

06-testing -- zsh -- 104x31
gioliguori@Gios-MacBook-Pro 06-testing % kubectl exec -n team-backend network-test-backend -- timeout 10 curl -I https://google.com
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left   Speed
0 220 0 0 0 0 0 0 --:--:-- --:--:-- --:--:-- 0
HTTP/2 301
location: https://www.google.com/
content-type: text/html; charset=UTF-8
content-security-policy-report-only: object-src 'none';base-uri 'self';script-src 'nonce-blEWESZvIG65YAM N04MjVA' 'strict-dynamic' 'report-sample' 'unsafe-eval' 'unsafe-inline' https: http:;report-uri https:// csp.withgoogle.com/csp/gws/other-hp
date: Mon, 07 Jul 2025 14:55:46 GMT
expires: Wed, 06 Aug 2025 14:55:46 GMT
cache-control: public, max-age=2592000
server: gws
content-length: 220
x-xss-protection: 0
x-frame-options: SAMEORIGIN
alt-svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000

gioliguori@Gios-MacBook-Pro 06-testing %

```

```

06-testing -- zsh -- 104x31
gioliguori@Gios-MacBook-Pro 06-testing % kubectl exec -n team-frontend network-test-frontend -- timeout 10 curl -I http://httpbin.org/get || echo "Bloccato"
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left   Speed
0 254 0 0 0 0 0 0 --:--:-- --:--:-- --:--:-- 0
HTTP/1.1 200 OK
Date: Mon, 07 Jul 2025 14:56:02 GMT
Content-Type: application/json
Content-Length: 254
Connection: keep-alive
Server: gunicorn/19.9.0
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true

gioliguori@Gios-MacBook-Pro 06-testing %

```

```

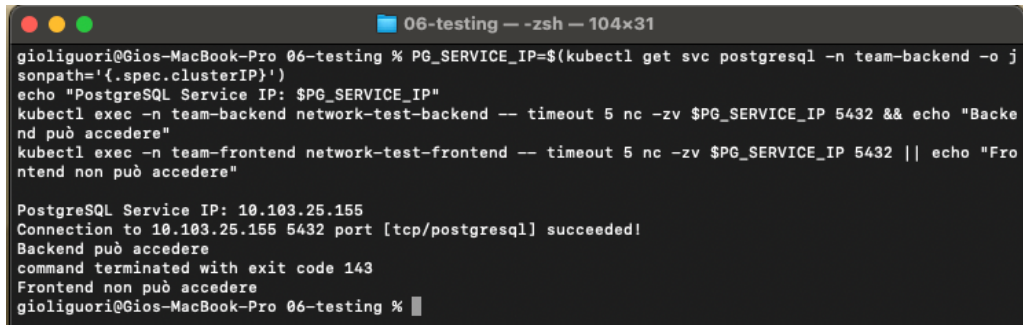
06-testing -- zsh -- 104x31
gioliguori@Gios-MacBook-Pro 06-testing % kubectl exec -n team-backend network-test-backend -- timeout 10 curl -I http://httpbin.org/get || echo "Bloccato"
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left   Speed
0 0 0 0 0 0 0 0 --:--:-- --:--:-- --:--:-- 0
it code 143
Bloccato
gioliguori@Gios-MacBook-Pro 06-testing %

```

Database Connectivity

```
PG_SERVICE_IP=$(kubectl get svc postgresql -n team-backend -o jsonpath='{.spec.clusterIP}')
echo "PostgreSQL Service IP: $PG_SERVICE_IP"
```

```
kubectl exec -n team-backend network-test-backend -- timeout 5 nc -zv $PG_SERVICE_IP 5432 && echo "Backend può accedere"
kubectl exec -n team-frontend network-test-frontend -- timeout 5 nc -zv $PG_SERVICE_IP 5432 || echo "Frontend non può accedere"
```



```
gioliguori@Gios-MacBook-Pro 06-testing % PG_SERVICE_IP=$(kubectl get svc postgresql -n team-backend -o jsonpath='{.spec.clusterIP}')
echo "PostgreSQL Service IP: $PG_SERVICE_IP"
PostgreSQL Service IP: 10.103.25.155
kubectl exec -n team-backend network-test-backend -- timeout 5 nc -zv $PG_SERVICE_IP 5432 && echo "Backend può accedere"
Connection to 10.103.25.155 5432 port [tcp/postgresql] succeeded!
Backend può accedere
kubectl exec -n team-frontend network-test-frontend -- timeout 5 nc -zv $PG_SERVICE_IP 5432 || echo "Frontend non può accedere"
command terminated with exit code 143
Frontend non può accedere
gioliguori@Gios-MacBook-Pro 06-testing %
```

Codice di Riferimento File: [kubernetes/02-security/network-policies.yaml](#)

```
yaml
# Default Deny All Policy - Security by Design
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-all
  namespace: team-frontend
spec:
  podSelector: {} # Tutti i pod del namespace
  policyTypes:
    - Ingress
    - Egress
# Nessuna regola = blocco totale del traffico

---
# Whitelist: Allow Same Namespace Communication
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-same-namespace
  namespace: team-frontend
spec:
  podSelector: {}
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              team: frontend # Solo stesso team
  egress:
    - to:
        - namespaceSelector:
```

```
matchLabels:
  team: frontend
```

Conclusioni

- **Network isolation** perfetta: team-frontend ❌ → team-backend
- **Intra-namespace communication** preservata: team-frontend ✅ → team-frontend
- **DNS service discovery** funzionante

Demo 3: Resource Quota Enforcement

Obiettivo: Dimostrare limitazione risorse per namespace

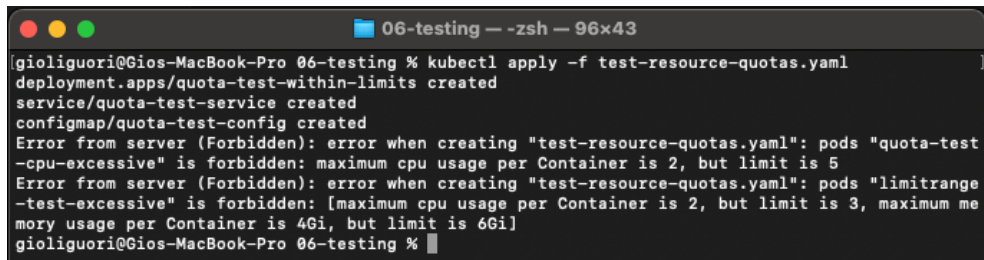
```
gioliguori@Gios-MacBook-Pro 06-testing % kubectl describe resourcequota team-frontend-quota -n team-frontend
kubectl get pods -n team-frontend
Name:                team-frontend-quota
Namespace:           team-frontend
Resource              Used  Hard
-----
configmaps            1    20
limits.cpu            2     8
limits.memory         6Gi  16Gi
persistentvolumeclaims 0    10
pods                  2    50
requests.cpu          400m  4
requests.memory       2Gi   8Gi
requests.storage      0    100Gi
secrets               0    20
services              1    10
services.loadbalancers 0     1
services.nodeports    0     2
NAME                  READY  STATUS   RESTARTS  AGE
react-store-8647886d87-78zc5  1/1    Running   0          94m
react-store-8647886d87-8np7q  1/1    Running   0          94m
gioliguori@Gios-MacBook-Pro 06-testing %
```

Codice di Riferimento

File: [kubernetes/02-security/resource-quotas.yaml](#)

```
yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: team-frontend-quota
  namespace: team-frontend
spec:
  hard:
    requests.cpu: "4"
    requests.memory: 8Gi
    limits.cpu: "8"
    limits.memory: 16Gi
    pods: "50"
    services: "10"
---
apiVersion: v1
```

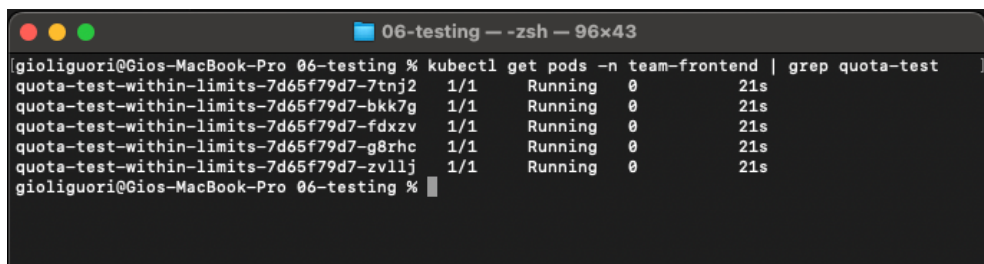
```
kind: LimitRange
metadata:
  name: team-frontend-limits
  namespace: team-frontend
spec:
  limits:
  - max:
      cpu: "2"# Max per container
      memory: 4Gi# Max per container
    type: Container
```



```
gioliguori@Gios-MacBook-Pro 06-testing % kubectl apply -f test-resource-quotas.yaml
deployment.apps/quota-test-within-limits created
service/quota-test-service created
configmap/quota-test-config created
Error from server (Forbidden): error when creating "test-resource-quotas.yaml": pods "quota-test-cpu-excessive" is forbidden: maximum cpu usage per Container is 2, but limit is 5
Error from server (Forbidden): error when creating "test-resource-quotas.yaml": pods "limitrange-test-excessive" is forbidden: [maximum cpu usage per Container is 2, but limit is 3, maximum memory usage per Container is 4Gi, but limit is 6Gi]
```

Test 1: Deployment entro limiti

```
yaml
spec:
  replicas: 5 # Entro limite 50 pods
  containers:
  - resources:
      requests:
        cpu: 50m # 5 × 50m = 250m (entro 4 CPU)
        memory: 64M # 5 × 64Mi = 320Mi (entro 8Gi)
```



```
gioliguori@Gios-MacBook-Pro 06-testing % kubectl get pods -n team-frontend | grep quota-test
quota-test-within-limits-7d65f79d7-7tnj2 1/1 Running 0 21s
quota-test-within-limits-7d65f79d7-bkk7g 1/1 Running 0 21s
quota-test-within-limits-7d65f79d7-fdxzv 1/1 Running 0 21s
quota-test-within-limits-7d65f79d7-g8rhc 1/1 Running 0 21s
quota-test-within-limits-7d65f79d7-zvllj 1/1 Running 0 21s
```

Test 2: Pod CPU eccessivo

```
yaml
spec:
  containers:
  - resources:
      limits:
        cpu: "5" # VIOLAZIONE: 5 CPU > 2 CPU max LimitRange
```

```
gioliguori@Gios-MacBook-Pro 06-testing % kubectl get pod quota-test-cpu-excessive -n team-fronte
nd
[kubectl get pod limitrange-test-excessive -n team-frontend
Error from server (NotFound): pods "quota-test-cpu-excessive" not found
Error from server (NotFound): pods "limitrange-test-excessive" not found
gioliguori@Gios-MacBook-Pro 06-testing % ]
```




Test 3: Pod memory eccessivo

```
yaml
spec:
  containers:
  - resources:
      limits:
        memory: 6Gi# VIOLAZIONE: 6Gi > 4Gi max LimitRange
```

Test 4: Scaling oltre quota: Solo 2/60 pod attivi (ResourceQuota enforcement)

```
gioliguori@Gios-MacBook-Pro 06-testing % kubectl scale deployment react-store --replicas=60 -n t
eam-frontend
kubectl get deployment react-store -n team-frontend
kubectl get pods -n team-frontend | wc -l
[kubectl get events -n team-frontend --sort-by='.lastTimestamp' | tail -5
deployment.apps/react-store scaled
NAME          READY    UP-TO-DATE    AVAILABLE    AGE
react-store   2/60     2             2            96m
11
0s    Normal    SuccessfulCreate    replicaset/react-store-8647886d87
Created pod: react-store-8647886d87-ttt4r
0s    Normal    Scheduled          pod/react-store-8647886d87-g6279
Successfully assigned team-frontend/react-store-8647886d87-g6279 to multi-tenant-worker2
0s    Normal    ScalingReplicaSet  deployment/react-store
Scaled up replica set react-store-8647886d87 from 2 to 60
0s    Normal    Scheduled          pod/react-store-8647886d87-ttt4r
Successfully assigned team-frontend/react-store-8647886d87-ttt4r to multi-tenant-worker2
0s    Normal    Scheduled          pod/react-store-8647886d87-88k96
Successfully assigned team-frontend/react-store-8647886d87-88k96 to multi-tenant-worker
gioliguori@Gios-MacBook-Pro 06-testing % ]
```

Risultati Test:

-  **Deployment entro limiti:** 5 pod creati (250m CPU, 320Mi RAM totali)
-  **Pod CPU eccessivo:** Bloccato da LimitRange (5 CPU > 2 CPU max)
-  **Pod memory eccessivo:** Bloccato da LimitRange (6Gi > 4Gi max)

Conclusioni

- **LimitRange:** blocca container con risorse eccessive
- **ResourceQuota:** limita creazione pod oltre limite namespace

Demo 4: Load Balancing tra Pod

Obiettivo: Dimostrare distribuzione carico automatica tra repliche backend all'interno del cluster


```

06-testing -- -zsh -- 96x43
gioliguori@Gios-MacBook-Pro 06-testing % kubectl run curl-test --image=curlimages/curl -n team-p
platform \
  --restart=Never --rm -it -- sh
If you don't see a command prompt, try pressing enter.
~ $ for i in 1 2 3 4 5; do
>   echo "Request $i:"
>   curl -s http://users-api.team-backend.svc.cluster.local:3000/api/server-info | grep hostname
>   sleep 1
> done
Request 1:
{"hostname":"users-api-68c4458469-dkqbc","podIP":"10.244.84.2","nodeName":"multi-tenant-worker2",
,"namespace":"team-backend","database":"postgresql.team-backend.svc.cluster.local:5432","timesta
mp":"2025-06-30T15:23:07.910Z","message":"Request handled by users-api-68c4458469-dkqbc","uptime
":6069,"architecture":"kubernetes-native","load_balancer":"kubernetes-service"}

Request 2:
{"hostname":"users-api-68c4458469-dkqbc","podIP":"10.244.84.2","nodeName":"multi-tenant-worker2",
,"namespace":"team-backend","database":"postgresql.team-backend.svc.cluster.local:5432","timesta
mp":"2025-06-30T15:23:08.920Z","message":"Request handled by users-api-68c4458469-dkqbc","uptime
":6070,"architecture":"kubernetes-native","load_balancer":"kubernetes-service"}

Request 3:
{"hostname":"users-api-68c4458469-j66sx","podIP":"10.244.121.133","nodeName":"multi-tenant-work
er","namespace":"team-backend","database":"postgresql.team-backend.svc.cluster.local:5432","timesta
mp":"2025-06-30T15:23:09.937Z","message":"Request handled by users-api-68c4458469-j66sx","upti
me":"6071","architecture":"kubernetes-native","load_balancer":"kubernetes-service"}

Request 4:
{"hostname":"users-api-68c4458469-dkqbc","podIP":"10.244.84.2","nodeName":"multi-tenant-worker2",
,"namespace":"team-backend","database":"postgresql.team-backend.svc.cluster.local:5432","timesta
mp":"2025-06-30T15:23:10.959Z","message":"Request handled by users-api-68c4458469-dkqbc","uptime
":6072,"architecture":"kubernetes-native","load_balancer":"kubernetes-service"}

Request 5:
{"hostname":"users-api-68c4458469-dkqbc","podIP":"10.244.84.2","nodeName":"multi-tenant-worker2",
,"namespace":"team-backend","database":"postgresql.team-backend.svc.cluster.local:5432","timesta
mp":"2025-06-30T15:23:11.967Z","message":"Request handled by users-api-68c4458469-dkqbc","uptime
":6073,"architecture":"kubernetes-native","load_balancer":"kubernetes-service"}
~ $ exit

```

Codice di Riferimento

File: [kubernetes/03-workloads/backend/users-api-service.yaml](#)

```

yaml
apiVersion: v1
kind: Service
metadata:
  name: users-api
  namespace: team-backend
spec:
  type: ClusterIP
  ports:
    - port: 3000
      targetPort: 3000
      protocol: TCP
      name: http
  selector:
    app: users-api
    team: backend
  sessionAffinity: None # Round-robin load balancing

```

Test interno cluster:

```

bash
# Scale a 3 repliche per demo
kubectl scale deployment users-api --replicas=3 -n team-backend

```



```
# Test load balancing da pod interno
kubectl run curl-test --image=curlimages/curl -n team-platform \
  --restart=Never --rm -it -- sh

for i in 1 2 3 4 5; do
  echo "Request $i:"
  curl -s http://users-api.team-backend.svc.cluster.local:3000/api/server-info | grep hostname
  sleep 1
done
```

Risultati distribuzione:

- **Request 1,2,4,5:** `users-api-68c4458469-dkqbc` (multi-tenant-worker2)
- **Request 3:** `users-api-68c4458469-j66sx` (multi-tenant-worker)

Conclusioni

- **Round-robin distribution** automatica tra pod backend

Load balancing funziona solo internamente - il port-forward bypassa il Service connettendosi direttamente a un singolo pod, mentre in produzione si usa Ingress Controller o LoadBalancer per il traffico esterno

Demo 5: HPA Auto-scaling

Obiettivo: Dimostrare scaling automatico basato su metriche CPU per gestione carico enterprise

```
gioliguori@Gios-MacBook-Pro ~ % kubectl get hpa -n team-backend
echo ""
kubectl get pods -n team-backend -l app=users-api
echo ""
[kubectl top pods -n team-backend
NAME                REFERENCE                TARGETS                MINPODS
MAXPODS  REPLICAS  AGE
users-api-hpa      Deployment/users-api      cpu: 0%/60%, memory: 30%/70%  2
4              2          15m

NAME                READY   STATUS    RESTARTS   AGE
users-api-68c4458469-kk44v  1/1     Running   0          15m
users-api-68c4458469-qhjbs  1/1     Running   0          15m

NAME                CPU(cores)  MEMORY(bytes)
postgresql-659fb6bf4f-161b9  7m          23Mi
users-api-68c4458469-kk44v    1m          62Mi
users-api-68c4458469-qhjbs    2m          92Mi
gioliguori@Gios-MacBook-Pro ~ %
```

stato iniziale: HPA configurato, 2 pod attivi, CPU baseline 0%

```
gioliguori@gioliguori-MacBook-Pro ~ % kubectl get hpa -n team-backend
echo ""
kubectl get pods -n team-backend -l app=users-api
echo ""
kubectl top pods -n team-backend
NAME                                REFERENCE                                TARGETS                                MINPODS
users-api-hpa                       Deployment/users-api                     cpu: 0%/60%, memory: 30%/70%          2
4                                   2                                   15m

NAME                                CPU(cores)                                MEMORY(bytes)
users-api-68c4458469-kk44v          1m                                         62Mi
users-api-68c4458469-qh3bs          1m                                         92Mi
gioliguori@gioliguori-MacBook-Pro ~ % kubectl get hpa -n team-backend
echo ""
kubectl get pods -n team-backend -l app=users-api
echo ""
kubectl top pods -n team-backend
NAME                                REFERENCE                                TARGETS                                MINPODS

gioliguori@gioliguori-MacBook-Pro ~ % for i in {1..50000}
do
  curl -s http://localhost:3001/api/products >/dev/null &
  curl -s http://localhost:3001/api/categories >/dev/null &
done
[2] 5984
[3] 5985
[4] 5986
[5] 5987
[6] 5988
[7] 5989
[8] 5990
[9] 5991
[10] 5992
[11] 5993
[12] 5994
[13] 5995
[14] 5996
[15] 5997
[16] 5998
[17] 5999
[18] 6000
[19] 6001
[20] 6002

gioliguori@gioliguori-MacBook-Pro ~ % kubectl get hpa -n team-backend -w
NAME                                REFERENCE                                TARGETS                                MINPODS
users-api-hpa                       Deployment/users-api                     cpu: 0%/60%, memory: 27%/70%          2
4                                   2                                   19m
users-api-hpa                       Deployment/users-api                     cpu: 0%/60%, memory: 27%/70%          2
4                                   2                                   19m
users-api-hpa                       Deployment/users-api                     cpu: 188%/60%, memory: 31%/70%        2
4                                   2                                   20m
users-api-hpa                       Deployment/users-api                     cpu: 188%/60%, memory: 31%/70%        2
4                                   2                                   20m
users-api-hpa                       Deployment/users-api                     cpu: 163%/60%, memory: 32%/70%        2
4                                   2                                   20m
users-api-hpa                       Deployment/users-api                     cpu: 159%/60%, memory: 32%/70%        2
4                                   2                                   20m
users-api-hpa                       Deployment/users-api                     cpu: 138%/60%, memory: 31%/70%        2
4                                   2                                   21m
users-api-hpa                       Deployment/users-api                     cpu: 106%/60%, memory: 26%/70%        2
4                                   2                                   21m
users-api-hpa                       Deployment/users-api                     cpu: 54%/60%, memory: 25%/70%         2
4                                   2                                   21m
users-api-hpa                       Deployment/users-api                     cpu: 2%/60%, memory: 21%/70%          2
4                                   2                                   21m
users-api-hpa                       Deployment/users-api                     cpu: 1%/60%, memory: 21%/70%          2
4                                   2                                   21m

gioliguori@gioliguori-MacBook-Pro ~ % kubectl get pods -n team-backend -w
NAME                                READY                                STATUS                                RESTARTS                                AGE
postgresql-659fb5bf4f-l6lb9        1/1                                  Running                                0                                         20m
users-api-68c4458469-kk44v          1/1                                  Running                                0                                         19m
users-api-68c4458469-qh3bs          1/1                                  Running                                0                                         19m
users-api-68c4458469-rw8fq          0/1                                  Pending                                0                                         0s
users-api-68c4458469-rw8fq          0/1                                  Pending                                0                                         0s
users-api-68c4458469-rw8fq          0/1                                  Pending                                0                                         0s
users-api-68c4458469-rw8fq          0/1                                  Pending                                0                                         0s
users-api-68c4458469-rw8fq          0/1                                  ContainerCreating                     0                                         0s
users-api-68c4458469-rw8fq          0/1                                  ContainerCreating                     0                                         0s
users-api-68c4458469-rw8fq          0/1                                  ContainerCreating                     0                                         1s
users-api-68c4458469-rw8fq          0/1                                  ContainerCreating                     0                                         2s
users-api-68c4458469-rw8fq          0/1                                  Running                                0                                         3s
users-api-68c4458469-rw8fq          0/1                                  Running                                0                                         12s
users-api-68c4458469-rw8fq          0/1                                  Running                                0                                         12s
users-api-68c4458469-rw8fq          1/1                                  Running                                0                                         12s
users-api-68c4458469-rw8fq          1/1                                  Running                                0                                         12s
users-api-68c4458469-rw8fq          1/1                                  Terminating                          0                                         5m45s
users-api-68c4458469-rw8fq          1/1                                  Terminating                          0                                         5m46s
```

la CPU ha raggiunto il 188% superando la soglia del 60%. Il sistema di auto-scaling è intervenuto automaticamente, portando i pod da 2 a 4 repliche.

```
users-api-hpa                       Deployment/users-api                     cpu: 106%/60%, memory: 26%/70%        2
4                                   2                                   21m
users-api-hpa                       Deployment/users-api                     cpu: 54%/60%, memory: 25%/70%         2
4                                   2                                   21m
users-api-hpa                       Deployment/users-api                     cpu: 2%/60%, memory: 21%/70%          2
4                                   2                                   21m
users-api-hpa                       Deployment/users-api                     cpu: 1%/60%, memory: 21%/70%          2
4                                   2                                   22m

users-api-68c4458469-rw8fq          0/1                                  Running                                0                                         2s
users-api-68c4458469-rw8fq          0/1                                  Running                                0                                         3s
users-api-68c4458469-rw8fq          0/1                                  Running                                0                                         12s
users-api-68c4458469-rw8fq          0/1                                  Running                                0                                         12s
users-api-68c4458469-rw8fq          1/1                                  Running                                0                                         12s
users-api-68c4458469-rw8fq          1/1                                  Running                                0                                         12s
users-api-68c4458469-rw8fq          1/1                                  Terminating                          0                                         5m45s
users-api-68c4458469-rw8fq          1/1                                  Terminating                          0                                         5m46s
```

Scale-down automatico: terminato il carico, HPA riduce gradualmente le repliche (4→2) dopo periodo di stabilizzazione

Codice di Riferimento

File: `kubernetes/04-scaling/backend-scaling.yaml`

```
yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: users-api-hpa
  namespace: team-backend
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: users-api
  minReplicas: 2
  maxReplicas: 4
  metrics:
  - type: Resource
```

```

resource:
  name: cpu
  target:
    type: Utilization
    averageUtilization: 60
- type: Resource
  resource:
    name: memory
    target:
      type: Utilization
      averageUtilization: 70
behavior:
  scaleUp:
    stabilizationWindowSeconds: 15
    policies:
      - type: Pods
        value: 2
        periodSeconds: 15
  scaleDown:
    stabilizationWindowSeconds: 30
    policies:
      - type: Pods
        value: 1
        periodSeconds: 15

```

Test auto-scaling:

```

bash
# Verifica HPA status
kubectl get hpa -n team-backend
kubectl describe hpa users-api-hpa -n team-backend

# Load generation intenso
for i in {1..6000}; do
  curl -s http://localhost:3001/api/products >/dev/null &
  curl -s http://localhost:3001/api/categories >/dev/null &
done

# Monitoring real-time
kubectl get hpa -n team-backend -w
kubectl get pods -n team-backend -w

```

Conclusioni

- **Auto-detection** spike CPU (188% > 60% threshold)
- **Automatic scaling** 2→4 pod in <30 secondi
- **Stabilization** con scale-down automatico dopo normalizzazione carico

Pod Disruption Budget

Il PDB garantisce che durante manutenzioni o update rimanga sempre almeno 1 pod attivo.

```
yaml
minAvailable: 1
```

non funziona con port-forward:

Il port-forward si collega direttamente al service. Durante un rolling update, per qualche secondo tutti i pod sono in transizione (vecchio che muore, nuovo che sta partendo) e il service perde la connessione.

CONCLUSIONI

Sviluppi Futuri

L'architettura è stata progettata per essere estendibile. Tra le funzionalità che avrei voluto implementare, ma che non sono riuscito a completare per motivi di tempo, c'è l'integrazione di **ArgoCD** per il GitOps deployment:

L'integrazione con **GitHub Actions** avrebbe permesso di implementare un workflow CI/CD completo:

```
yaml
# .github/workflows/deploy.yml
name: Deploy to Kubernetes
on:
  push:
    branches: [main]
jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Build and Push Docker Image
        run: |
          docker build -t ${ secrets.REGISTRY }}/app:${ github.sha } .
          docker push ${ secrets.REGISTRY }}/app:${ github.sha }
      - name: Update ArgoCD Application
        run: |
          # Trigger ArgoCD sync via webhook o CLI
          argocd app sync backend-apps
```

Altri miglioramenti futuri potrebbero includere:

- **Backup e Disaster Recovery:** Integrazione con Velero per backup automatici

Questo progetto dimostra come Kubernetes, con la giusta architettura e configurazione, possa fornire una piattaforma multi-tenant robusta, sicura e scalabile.