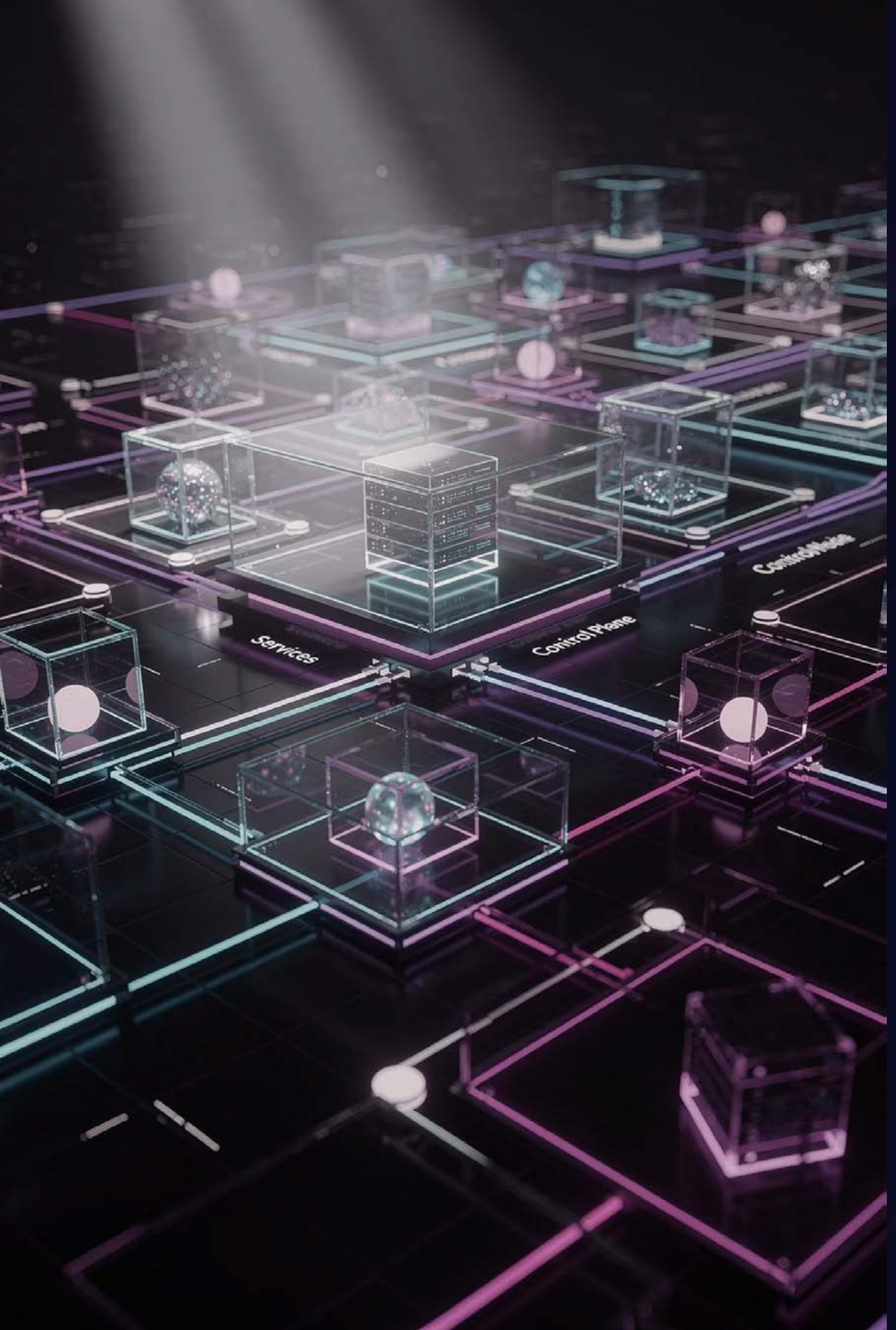


Scaling Microservices Development A Shared-Cluster Deployment Platform for Faster, Cost-Efficient SaaS Delivery

By : Ravi Babu Dasari, NetApp Inc.,

Conf42 DevOps 2026

The Microservices Promise and Challenge

The Promise

Microservices architectures deliver unparalleled scalability and modularity, enabling rapid innovation and independent service deployment across modern SaaS platforms.

The Challenge

However, they introduce complexity in environment management, testing workflows, and cross-team coordination that traditional infrastructure approaches struggle to address efficiently.

Traditional Approach: The Bottleneck

Dedicated Clusters

Provisioning separate Kubernetes clusters for every feature branch creates isolation but at tremendous cost.

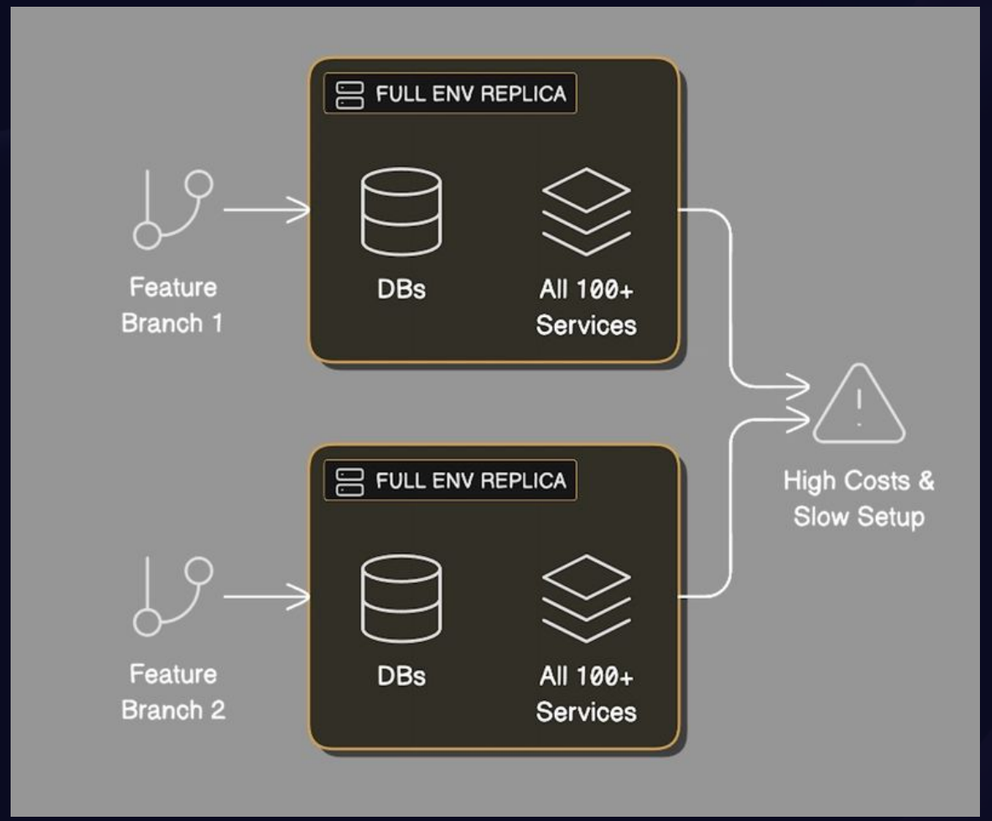
Resource Waste

Excessive infrastructure usage with underutilised compute resources scattered across numerous environments.

Slow Provisioning

Long wait times for cluster creation become critical bottlenecks that slow delivery pipelines significantly.

Traditional Architecture



Why Canary Deployments Fall Short for Feature Branch Testing

- Designed for Production Rollouts, Not Development
- Mostly limited to just one canary deployment at a time
- Complications with Stateful Services
- Limited Isolation in Shared Clusters
- Better Alternative: Namespace-Based Smart Isolation for any number of feature branches

Introducing SDDP

The Scalable Development Deployment Platform

1

Namespace Isolation

Leverage Kubernetes namespace boundaries to provide secure, isolated environments within shared clusters.

2

Dynamic Routing

Implement intelligent ingress routing that directs traffic to feature-specific microservices seamlessly.

3

Shared Infrastructure

Enable feature services to run alongside production-grade components without full-stack duplication.

Deploying a feature branch to a namespace with dynamic routing

YAML

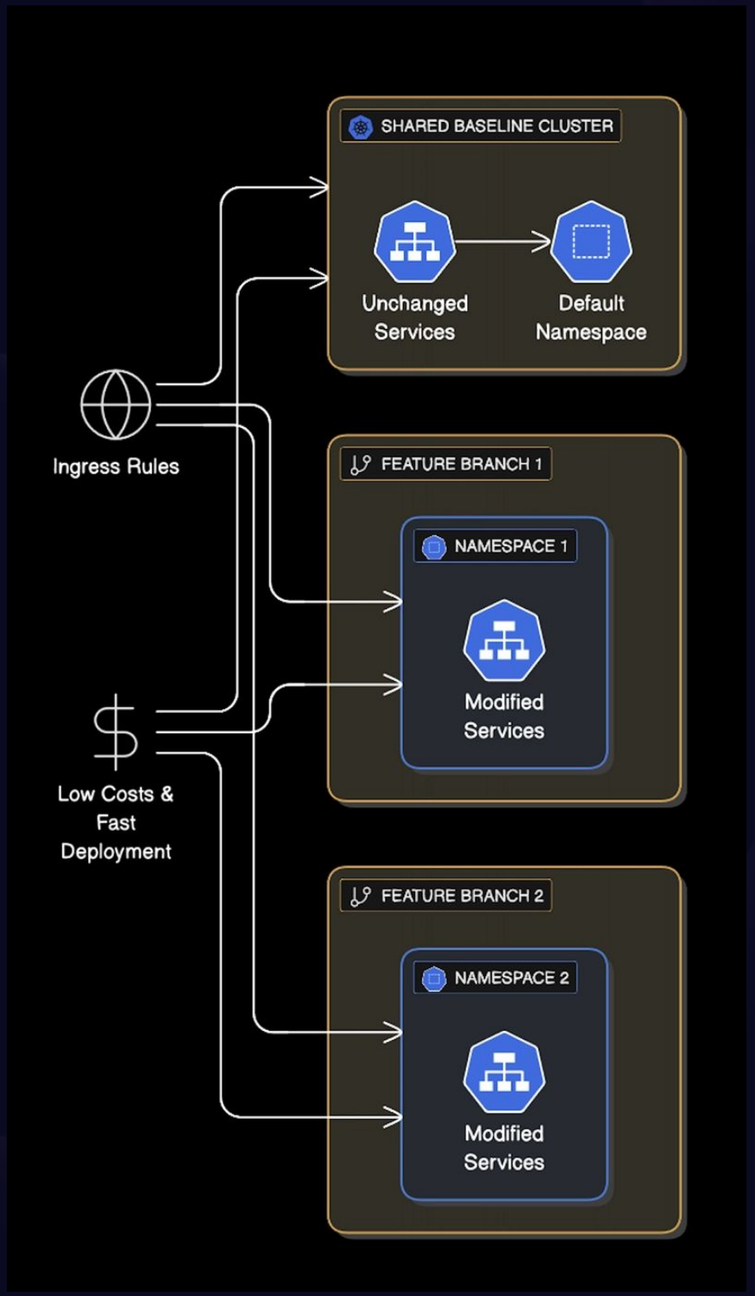
Copy

```yaml
# values-feature-mybranch.yaml (Helm override for feature branch)
replicaCount: 1
image:
  tag: "mybranch-abc123"

# Auto-generated by SDDP portal
namespace: feature-mybranch

ingress:
  enabled: true
  host: mybranch.myapp.dev.example.com
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/rewrite-target: /
```

Proposed Architecture



SDDP Architecture Overview

1

Management Portal

Web-based interface for orchestrating deployments, managing environments, and providing visibility across teams.

2

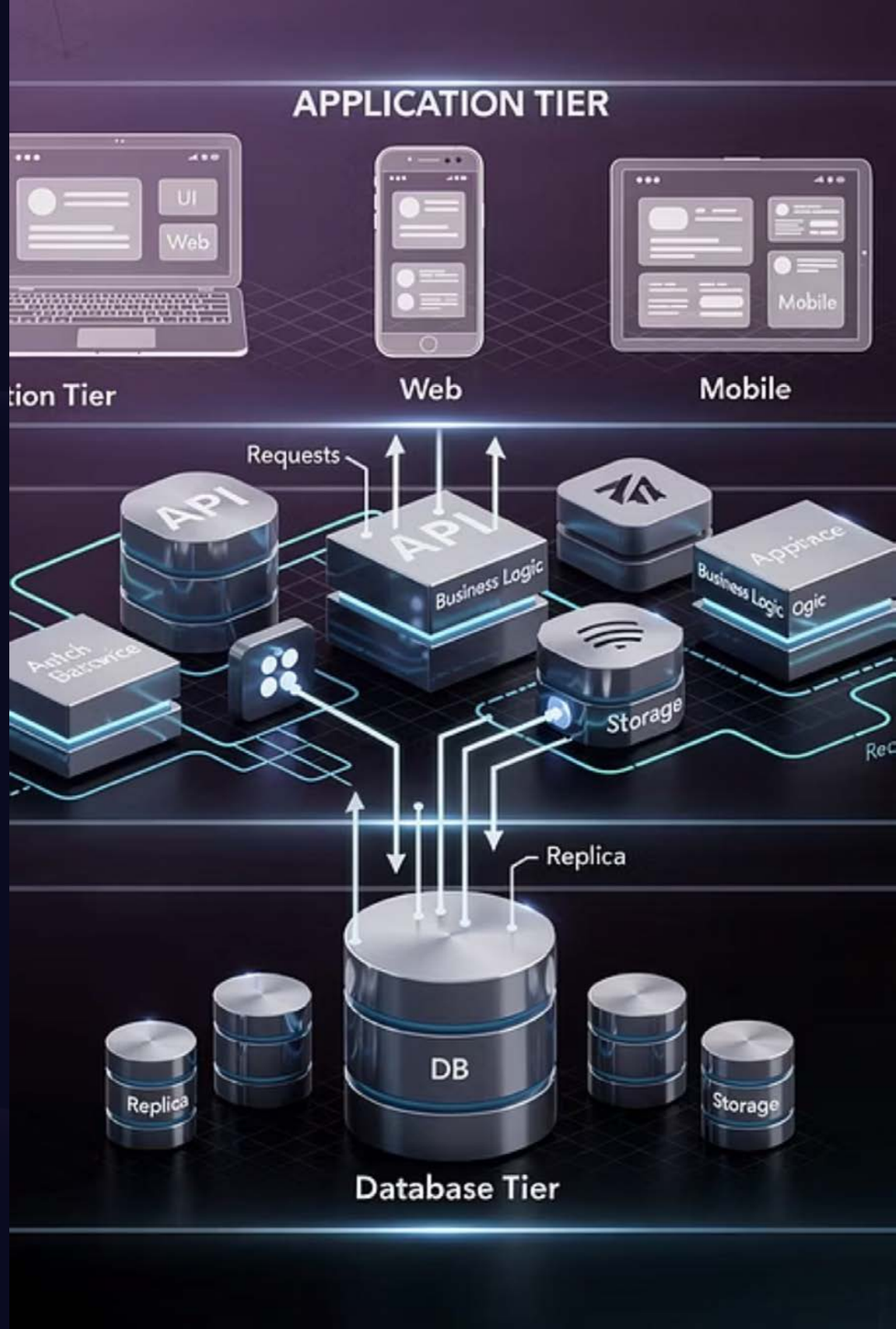
Service Layer

Integrates seamlessly with CI/CD pipelines through Helm-based automation and declarative configuration management.

3

Infrastructure Layer

Kubernetes foundation optimised for multi-tenancy, efficient resource allocation, and horizontal scaling capabilities.



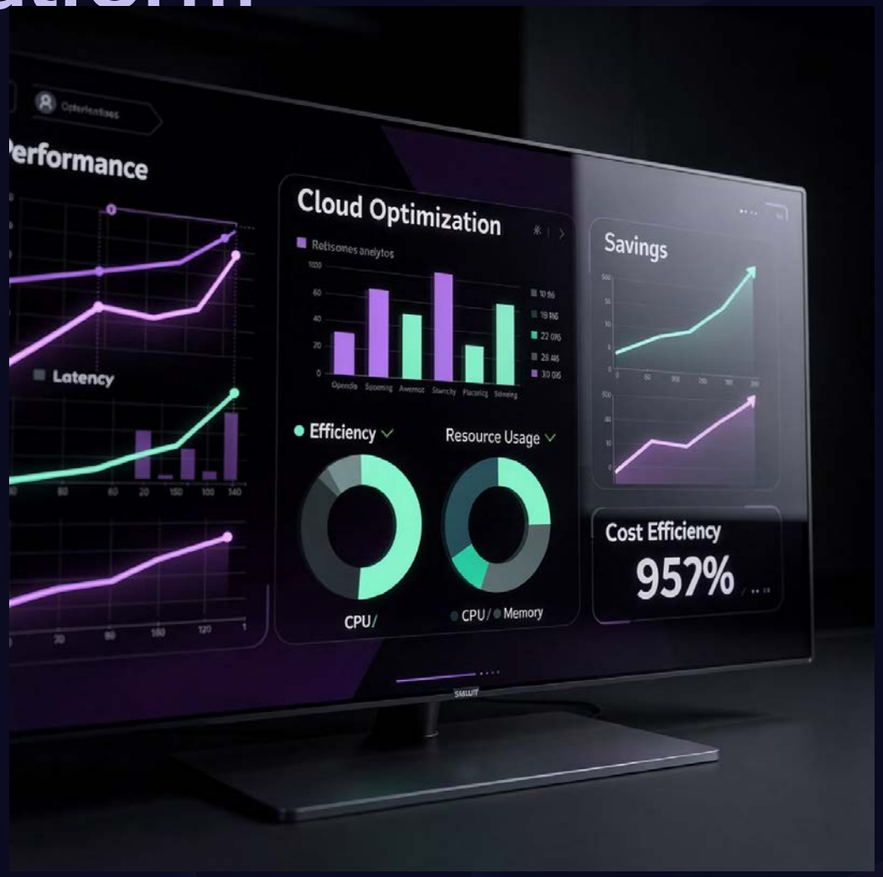
Demonstrate namespace isolation with NetworkPolicy

YAML

Copy

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-cross-namespace
  namespace: feature-mybranch
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: feature-mybranch  # Only same namespace
    - podSelector: {}  # Allow from baseline services if needed
  egress:
  - to:
    - namespaceSelector:
        matchLabels:
          app: baseline-services
```

Core Benefits of the Platform



Streamlined Development

Accelerated workflows that maintain strict isolation whilst reducing operational overhead.

Cost Efficiency

Dramatic reduction in infrastructure spending through intelligent resource sharing and utilisation.

Environment Consistency

Unified configuration management ensures parity between development, staging, and production environments.

Security Without Compromise

Role-Based Access Control

Granular permissions ensure teams access only their designated namespaces and resources, maintaining strict security boundaries.

Network Policies

Kubernetes network policies enforce traffic segregation between namespaces, preventing unauthorised service-to-service communication.

Centralised Secret Management

Integrated secret management solutions provide encrypted storage and controlled access to sensitive configuration data.

Handling Outliers: Challenges with Stateful Services Core

Challenge

Stateful services (e.g., databases, caches, blob storage) introduce side effects when shared across namespaces in Kubernetes.

Risks

Data corruption, inconsistent states, or interference between feature branches/testing environments.

Outliers in Microservices

Most services are stateless and easy to isolate, but stateful ones and complex dependencies require special handling.

Goal: Balance isolation for safe testing with resource efficiency in shared clusters.

Strategies for Isolation and Replication

Tools & Techniques

Kubernetes namespaces for resource segregation.

Ingress/service meshes for traffic routing (e.g., connection strings).

Cache isolation: Namespace-scoped keys or dedicated instances.

Automation with AI

Analyze code changes, recommend services to replicate, and auto-generate Kubernetes manifests (YAML for namespaces/ingress).

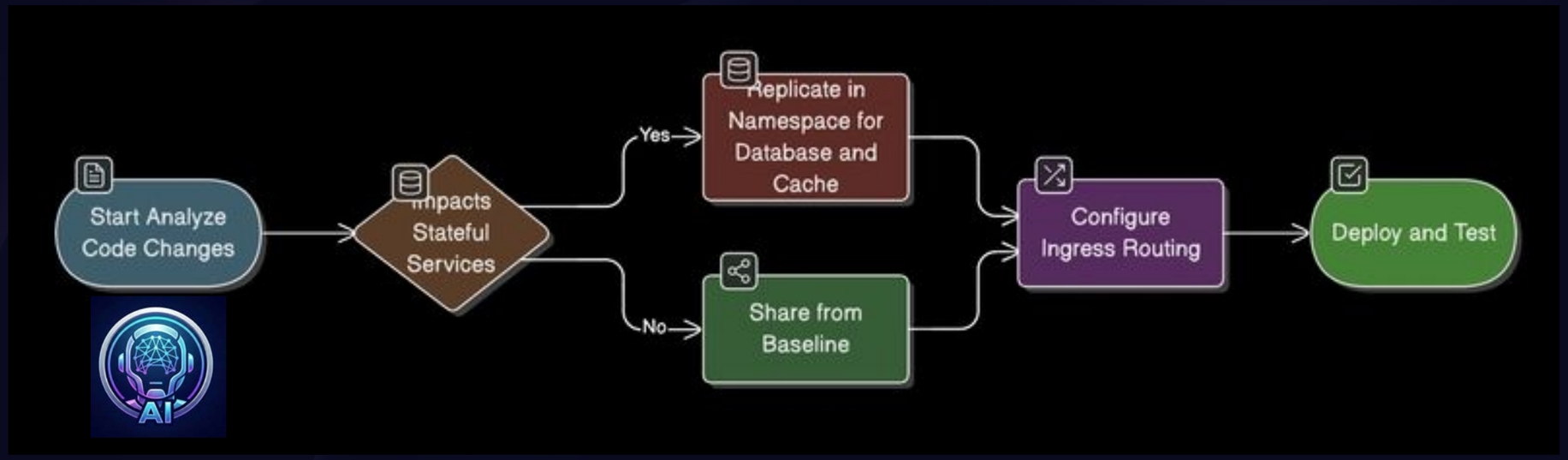
Change Analysis

Scan feature branch diffs to detect impacts on stateful components.

Examples:

- New endpoint (no schema change) → Share existing database.
- Schema migration → Replicate affected database in namespace.

Analyze with the help of AI



AI-driven change analysis logic (pseudo-code in Python)

Python

Copy

```python
def analyze_changes(diff):
    stateful_impacts = []

    if any('migration' in file or '.sql' in file for file in diff):
        stateful_impacts.append({
            "service": "postgresql",
            "action": "replicate",
            "reason": "schema migration detected"
        })

    if any('redis' in file and 'KEYS' in content for file in diff):
        stateful_impacts.append({
            "service": "redis",
            "action": "namespace-scoped-keys",
            "reason": "direct key access - risk of collision"
        })

    return stateful_impacts

# SDDP uses this to decide what to replicate vs. share
```

Dynamic connection string routing via environment variable injection

YAML

Copy

```yaml
# Deployment snippet auto-injected by SDDP
env:
  - name: DATABASE_URL
    value: "postgres://baseline-db.prod.svc.cluster.local:5432/app"
  - name: REDIS_URL
    value: "redis://redis-feature-mybranch:6379"  # Replicated instance
  - name: CACHE_PREFIX
    value: "mybranch:"  # Namespace-scoped keys when sharing Redis
```

ResourceQuota to prevent noisy neighbor issues

YAML

Copy

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: feature-quota
  namespace: feature-mybranch
spec:
  hard:
    requests.cpu: "4"
    requests.memory: 8Gi
    limits.cpu: "8"
    limits.memory: 16Gi
    pods: "20"
```

Integration with CI/CD Workflows

1

Commit Trigger

Developer pushes code to feature branch, initiating automated pipeline.

2

Build & Package

CI system builds container images and packages Helm charts automatically.

3

Deploy to SDDP

Platform provisions namespace and deploys services with dynamic routing.

4

Test & Validate

Automated tests run against isolated environment before promotion.

Measured Impact and Outcomes

- **Faster Environment Creation**
  Reduction in time from request to usable development environment
- **Shorter Testing Cycles**
  Decrease in end-to-end testing duration through parallel execution
- **Resource Utilisation**
  Improvement in compute resource efficiency across infrastructure

Engineering teams adopting SDDP observed substantial improvements across all key delivery metrics whilst significantly reducing operational costs.

Enhanced Team Collaboration

Development Teams

- Self-service environment provisioning
- Rapid iteration without infrastructure delays
- Clear visibility into deployment status
- Consistent tooling across all environments

DevOps Teams

- Centralised management and monitoring
- Reduced operational burden and toil
- Standardised deployment patterns
- Improved resource governance

Key Technical Considerations

Resource Quotas

Implement namespace-level resource limits to prevent resource exhaustion and ensure fair allocation across teams.

Ingress Strategy

Design dynamic routing patterns that support multiple ingress controllers and custom domain configurations.

Data Management

Establish clear patterns for database provisioning, test data management, and state isolation between environments.

Observability

Deploy comprehensive logging, metrics, and tracing infrastructure to maintain visibility across shared resources.

Your Path to Adoption

Phase 1: Foundation

Establish shared cluster infrastructure and core namespace isolation patterns.

Phase 2: Automation

Build CI/CD integration and self-service deployment capabilities.

Phase 3: Security

Implement RBAC, network policies, and secret management solutions.

Phase 4: Scale

Optimise resource allocation and expand to additional teams and services.

Transform Your Development Workflow

- **Accelerate Delivery**

  Reduce environment provisioning from hours to minutes

- **Optimise Costs**

  Dramatically decrease infrastructure spending through sharing

- **Scale Confidently**

  Support growing teams without proportional infrastructure growth

Real-World Success: Six Years of Proven Results

- **The Bottleneck**

  Manual cluster setup for each feature branch wasted hours daily, severely hindering development progress.

- **The Solution**

  A shared cluster with a dedicated portal streamlined environment management, delivering six years of continuous success.

- **Key Outcomes**

  Deployment times dropped from hours to minutes, accelerating development, boosting productivity, and improving team morale and system reliability.

The Deployment Portal: Your Command Center

- Automated Sandbox Creation

  Instantly create sandboxes for any feature branch, automating deployment and baseline selection for rapid testing.

- Effortless Collaboration

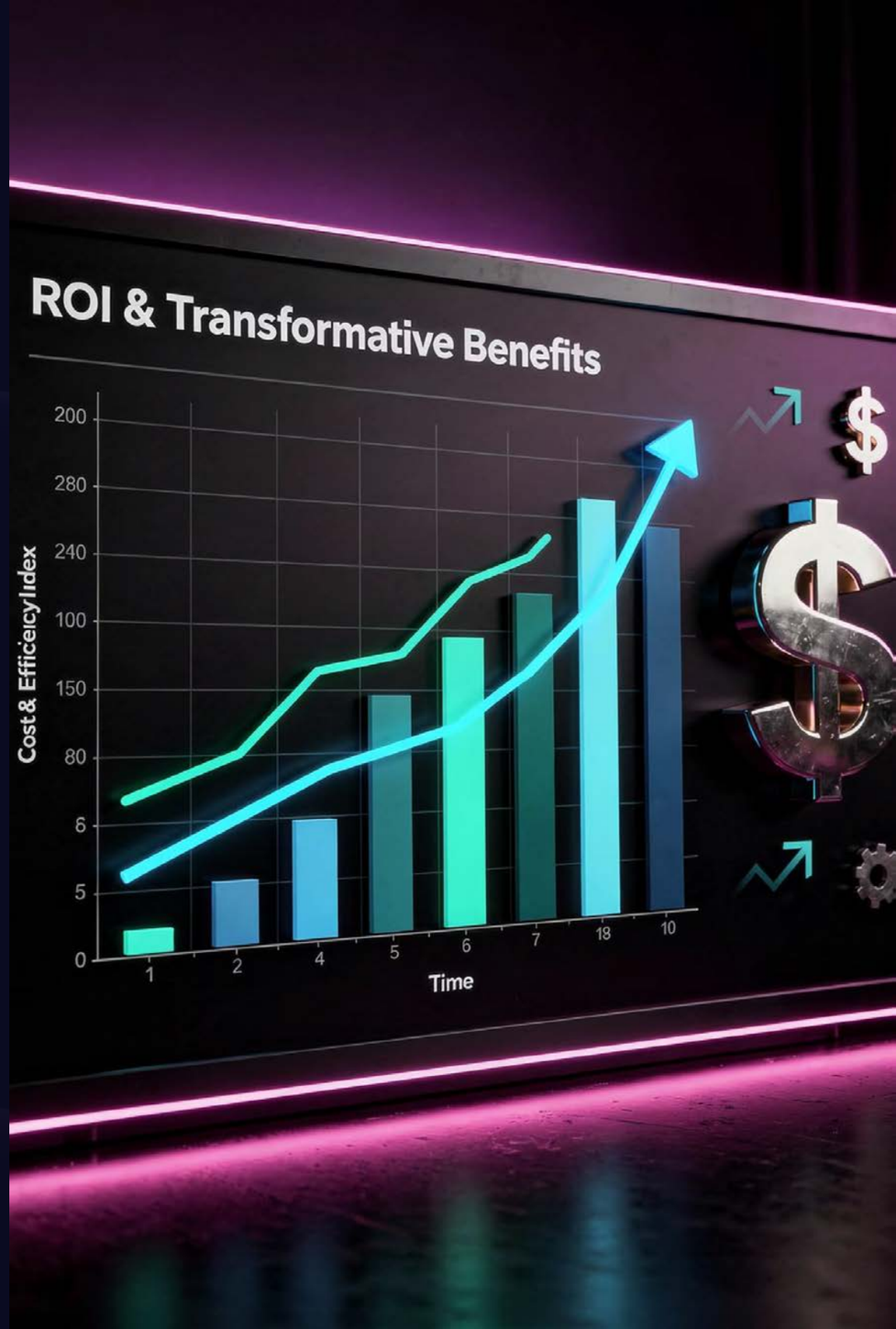
  Share sandbox links for quick team reviews and demos, accelerating feedback and alignment.

- AI-Powered Intelligence

  AI analyzes code changes, predicts side effects, and suggests optimal configurations for efficient, intelligent deployments.

Transformative Benefits and ROI

- Cost Reduction Infrastructure cost savings.
- Setup Time & Deployment time reduction.
- Engineers Supported Engineers enabled at scale.
- Operational Load Consistent operational overhead.



Start Your Transformation Today

- Revolutionize Development

  Redefine microservices development using robust infrastructure and AI-driven tools for enhanced efficiency and innovation.

- Proven Impact

  This methodology delivers substantial cost reductions, faster delivery cycles, and enhanced collaboration.

- Start Your Journey

  Initiate transformation with a strategic pilot program, establish a baseline, and scale success across your organization.

Join our community and share your transformation journey to shape the future of development.

Demo

Source code: https://github.com/RaviDasari/sddp

Thank You!
Questions?

Ravi Babu Dasari
NetApp Inc.,
IncConf42 DevOps 2026.

linkedin.com/in/ravi-babu-dasari-6528621b @ravibabudasari github.com/RaviDasari