Beyond Security: Leveraging OPA for FinOps in Kubernetes

SATHISH KUMAR VENKATESAN

#whoami

SATHISH KUMAR VENKATESAN

DevOpsCloudJunction - Founder & CEO

LinkedIn

Website

Interests and Hobbies:

- Passionate about computers, gadgets, electronics, photography, cloud computing, DevOps, and open-source technologies.
- Enjoy participating in tech communities and contributing to collaborative projects.

What I'll Cover Today

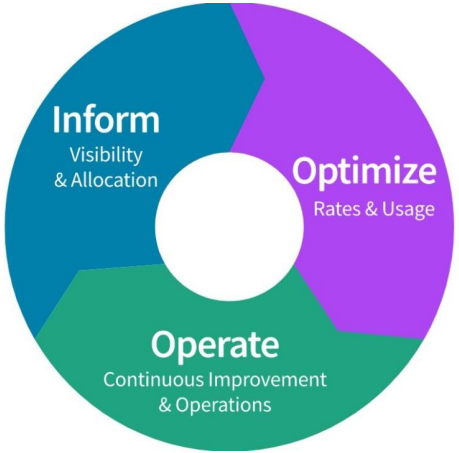
- The FinOps challenge in Kubernetes environments
- OPA fundamentals and capabilities
- Extending OPA beyond security to cost governance
- Practical FinOps policies with Rego
- Adopting strategies and best practices
- Live demo: OPA FinOps in action

FinOps

FinOps is the practice of bringing financial accountability to the cloud's dynamic environments.

Key Goals:

- Inform: Create visibility into cloud costs and usage.
- Optimize: Maximize cloud value through right-sizing and reservations
- Operate: Establish continuous processes for financial control



Source: finops.org

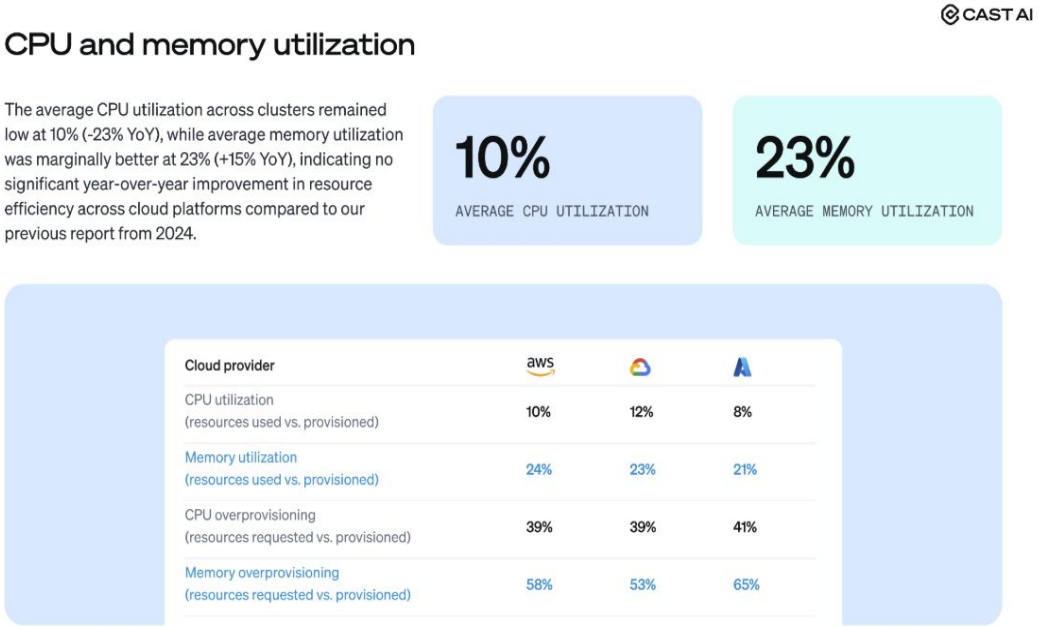
The FinOps Challenge

Average Kubernetes cloud waste: 32% of resources underutilized

Common issues:

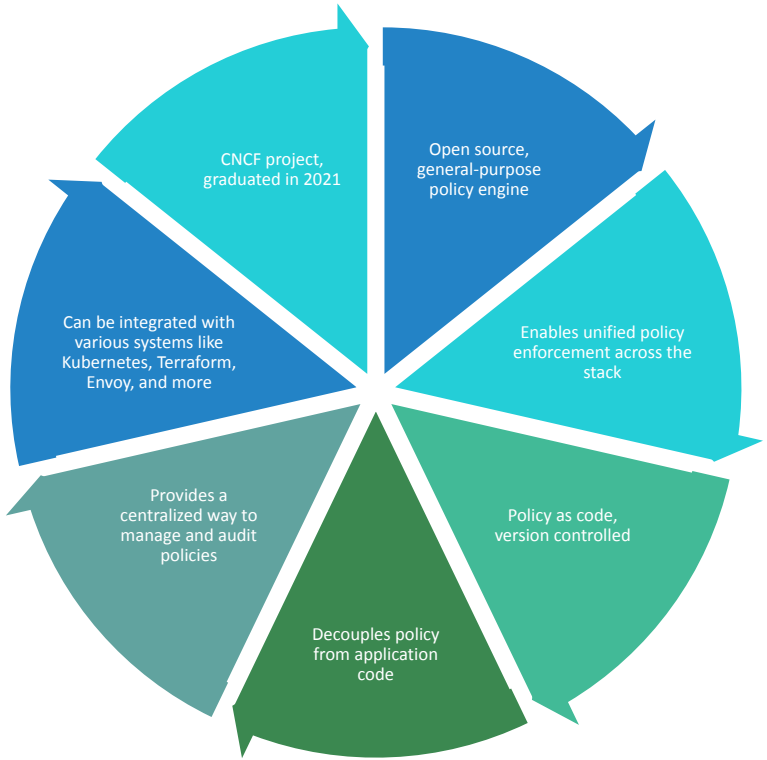
- Overprovisioned resources
- Idle workloads
- Non-optimal node selection
- Missing resource quotas
- Untagged resources

The Cost of Kubernetes Inefficiency



- Average CPU utilization remains critically low at 10%
- Memory utilization hovers at 23% across cloud providers
- Over 50% of requested memory resources remain unused

Source: Cast AI



What is Open Policy Agent (OPA)?

What are Policies in OPA?

```
package example

allow {
 input.user == "alice"
 input.method == "GET"
 input.path[0] == "users"
}
```

Rules in Rego: Policies are rules written in the Rego language.

Evaluate Data: Policies make decisions based on input data

Allow or Deny: The core outcome is typically to allow or deny an action

Policy as Code: Policies are text, enabling version control

Decoupled Logic: Policy decisions are separate from applications

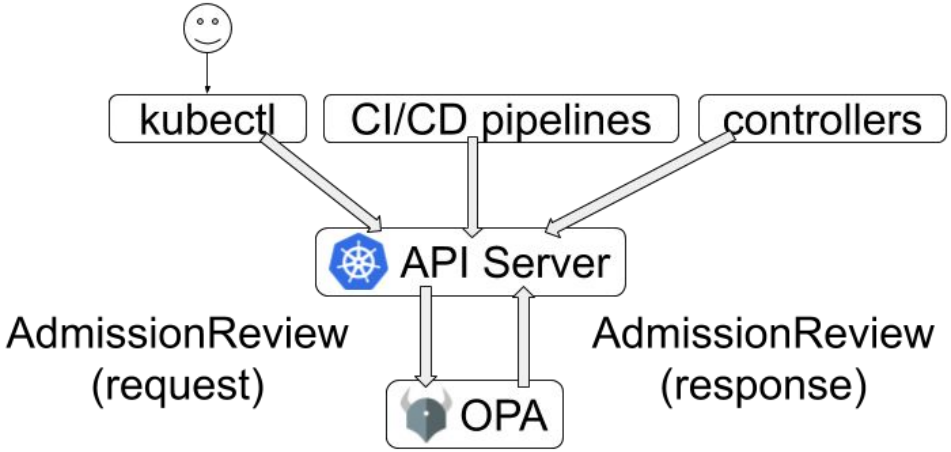
Centralized Control: OPA provides a single place to manage policies

Why OPA for FinOps?

- Reuse existing security tooling
- Single policy framework for multiple domains
- Shift-left approach to cost management
- Consistent enforcement across environments
- GitOps-friendly policies as code

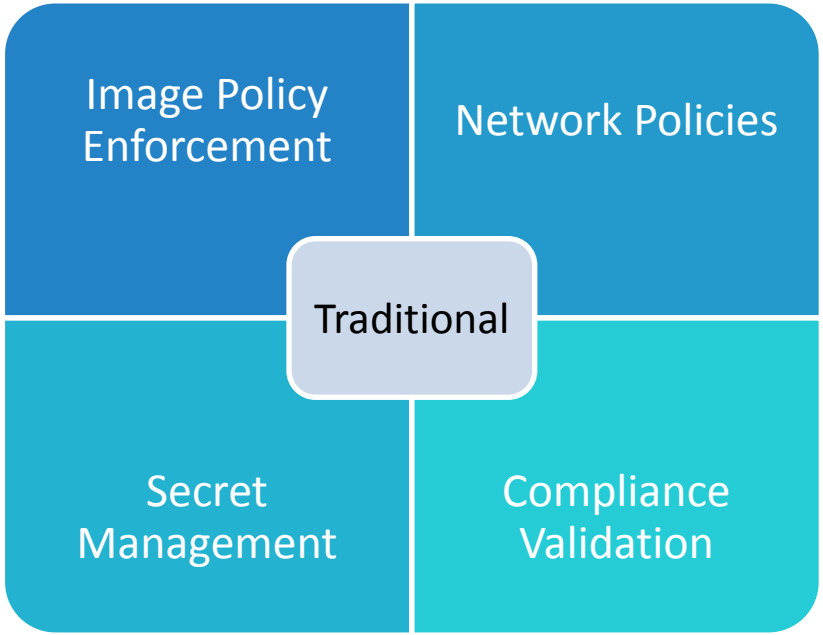
What is OPA Gatekeeper?

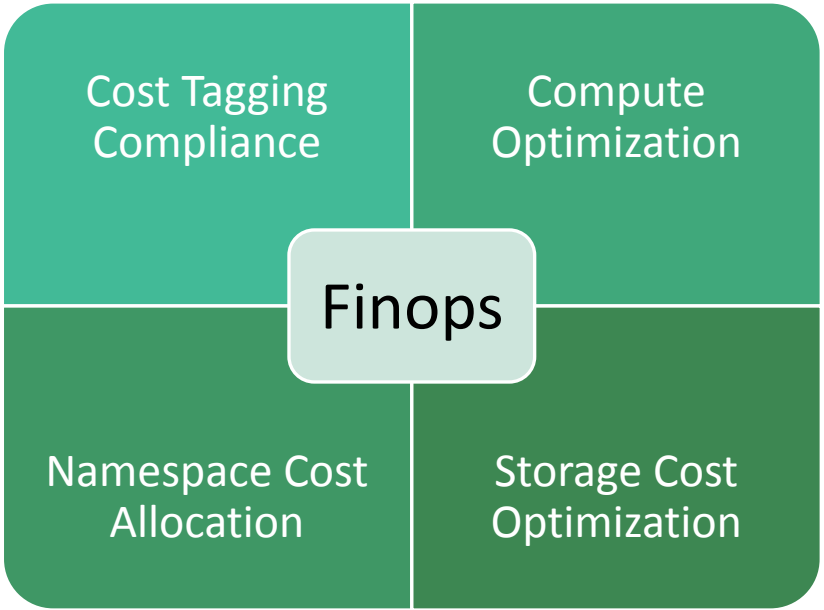
- Kubernetes-native implementation of OPA that enforces policies on Kubernetes resources
- Uses CustomResourceDefinitions (CRDs) to define and manage policies within the Kubernetes ecosystem
- Validates resources against policies during admission control before they're created or modified
- Provides audit capabilities to detect and report on existing policy violations in a cluster



source: www.openpolicyagent.org

Traditional Vs FinOps - OPA Use Cases





```
package templates.resourcerequests

violation[{"msg": msg}] {
  container := input.review.object.spec.containers[_]
  cpu_request := container.resources.requests.cpu
  memory_request := container.resources.requests.memory

  # Convert CPU to millicores for calculation
  cpu_millicore := to_number(cpu_request) * 1000

  # Check if CPU request exceeds allowed thresholds
  cpu_millicore > data.parameters.maxCPUMillicores

  msg := sprintf("Container '%v' CPU request of '%v' exceeds maximum allowed (%v millicores)",
    [container.name, cpu_request, data.parameters.maxCPUMillicores])
}
```

Example: Resource Request Optimization

GOAL: PREVENT OVER-PROVISIONING OF RESOURCES

Example: Enforce Spot Instances for Dev Workloads

GOAL: ENSURE ALL DEVELOPMENT WORKLOADS RUN ON COST-EFFICIENT SPOT INSTANCES

```
package kubernetes.finops.spot_instances

violation[{"msg": msg}] {
    input.review.kind.kind == "Pod"

    # Identify dev workloads via namespace or labels
    is_dev_workload

    # Check if spot instance selector is missing
    not has_spot_instance_selector

    msg := sprintf("Dev workload '%v' must use spot instances",
           [input.review.object.metadata.name])
}

# Match dev environments
is_dev_workload {
    contains(input.review.object.metadata.namespace, "dev")
}

is_dev_workload {
    input.review.object.metadata.labels.environment == "dev"
}

# Check for appropriate spot instance configuration
has_spot_instance_selector {
    input.review.object.spec.nodeSelector["node.kubernetes.io/lifecycle"] == "spot"
}
```

Example: Mandatory Cost Allocation Tagging Policy

GOAL: ENSURE CONSISTENT LABELING FOR ACCURATE COST TRACKING, ATTRIBUTION, AND FINOPS REPORTING

```
package finops.tagging

# Required cost labels for all resources
required_labels = {
 "cost-center",
 "team",
 "environment",
 "project-id"
}

# Check for missing labels on namespaces
violation[{"msg": msg}] {
 input.request.kind.kind == "Namespace"
 missing := required_labels - {label | label := object.get(input.request.object.metadata.labels, [_],
""")}
 count(missing) > 0
 msg := sprintf("Namespace missing required labels: %v", [concat(", ", missing)])
}

# Check for missing labels on workload resources
violation[{"msg": msg}] {
 workload_kinds := {"Deployment", "StatefulSet", "DaemonSet", "Job", "CronJob"}
 input.request.kind.kind == workload_kinds[_]
 missing := required_labels - {label | label := object.get(input.request.object.metadata.labels, [_],
""")}
 count(missing) > 0
 msg := sprintf("%v missing required labels: %v", [input.request.kind.kind, concat(", ", missing)])
}
```

```
package budgets

violation[{"msg": msg}] {
  # Get team from namespace labels
  team := input.review.object.metadata.namespace.labels.team

  # Calculate estimated monthly cost for this workload
  estimated_cost := calculate_workload_cost(input.review.object)

  # Compare against team's remaining budget from external data
  team_budget := data.external.budgets[team].remaining
  estimated_cost > team_budget

  msg := sprintf("Deployment would exceed team '%v' remaining budget of $%v", [team, team_budget])
}
```

Example: Budget Enforcement

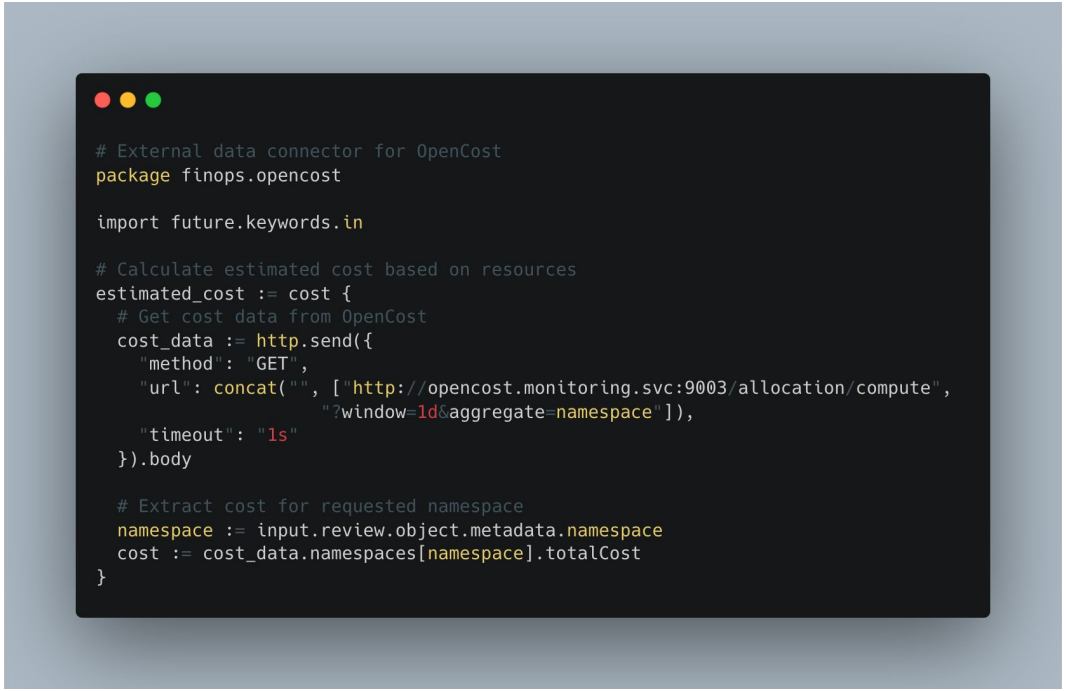
GOAL: ENFORCE TEAM-LEVEL SPENDING LIMITS

Example: Storage Optimization

GOAL: PREVENT USE OF PREMIUM STORAGE FOR NON-CRITICAL DATA

```
package storage

violation[{"msg": msg}] {
  # Check if it's a PVC request
  input.review.kind.kind == "PersistentVolumeClaim"

  # Get storage class
  storage_class := input.review.object.spec.storageClassName

  # Check if premium SSD
  premium_ssd_classes := {"premium-ssd", "io1", "premium-lrs", "ultra-disk"}
  storage_class == premium_ssd_classes[_]

  # Check for justification
  not input.review.object.metadata.annotations["finops.company.com/premium-storage-justification"]

  # Not in production critical tier
  not (input.review.object.metadata.namespace == "production" and
       input.review.object.metadata.labels.tier == "critical")

  msg := sprintf("Premium SSD storage class '%v' requires justification", [storage_class])
}
```

Integration with External Cost Tools



```
# External data connector for OpenCost
package finops.opencost

import future.keywords.in

# Calculate estimated cost based on resources
estimated_cost := cost {
  # Get cost data from OpenCost
  cost_data := http.send({
    "method": "GET",
    "url": concat("", ["http://opencost.monitoring.svc:9003/allocation/compute",
                       "?window=1d&aggregate=namespace"]),
    "timeout": "1s"
  }).body

  # Extract cost for requested namespace
  namespace := input.review.object.metadata.namespace
  cost := cost_data.namespaces[namespace].totalCost
}
```

Example: OpenCost Integration

- Connect real-time cost metrics to OPA decisions

  Example: Pulling cost data from OpenCost API

Start Small:

- Begin with visibility-only policies (dry-run mode)
- Focus on highest-cost resources first (GPU, high-memory)

Phased Implementation:

Adoption Strategy & Best Practices

Adoption Strategy & Best Practices

Team Collaboration:

Involve platform, security, finance, and application teams

Create exception processes

Document and communicate policy decisions

Measuring Success:

Track cost reduction metrics

Monitor policy violations and trends

Calculate ROI of policy enforcement

Common Challenges & Solutions

- **Solution**: Use efficient Rego patterns, test at scale
- **Solution**: Start with dry-run mode, provide clear documentation
- **Solution**: Create annotation-based override system
- **Solution**: Regular reconciliation with cloud billing
- **Solution**: GitOps workflow for policy management

Resources & Next Steps

GitHub Repository:

https://github.com/open-policy-agent/opa
https://github.com/open-policy-agent/gatekeeper-library

Documentation:

OPA/Gatekeeper:

https://www.openpolicyagent.org/
https://open-policy-agent.github.io/gatekeeper/website/

Community:

OPA Slack:
https://slack.openpolicyagent.org/

Demo

Q & A

Feedback