Being a good citizen of the multi-Operator world

Devdatta Kulkarni

Founder @ CloudARK

devdatta@cloudark.io

CloudARK

cloudark.io

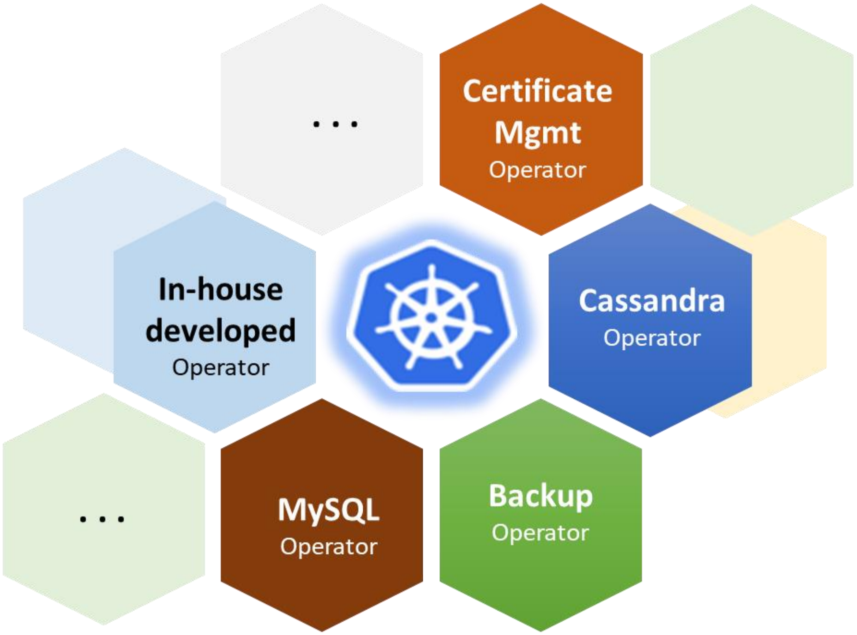
About me

Devdatta Kulkarni
Founder, CloudARK

PhD, Distributed systems,
University of Minnesota Minneapolis

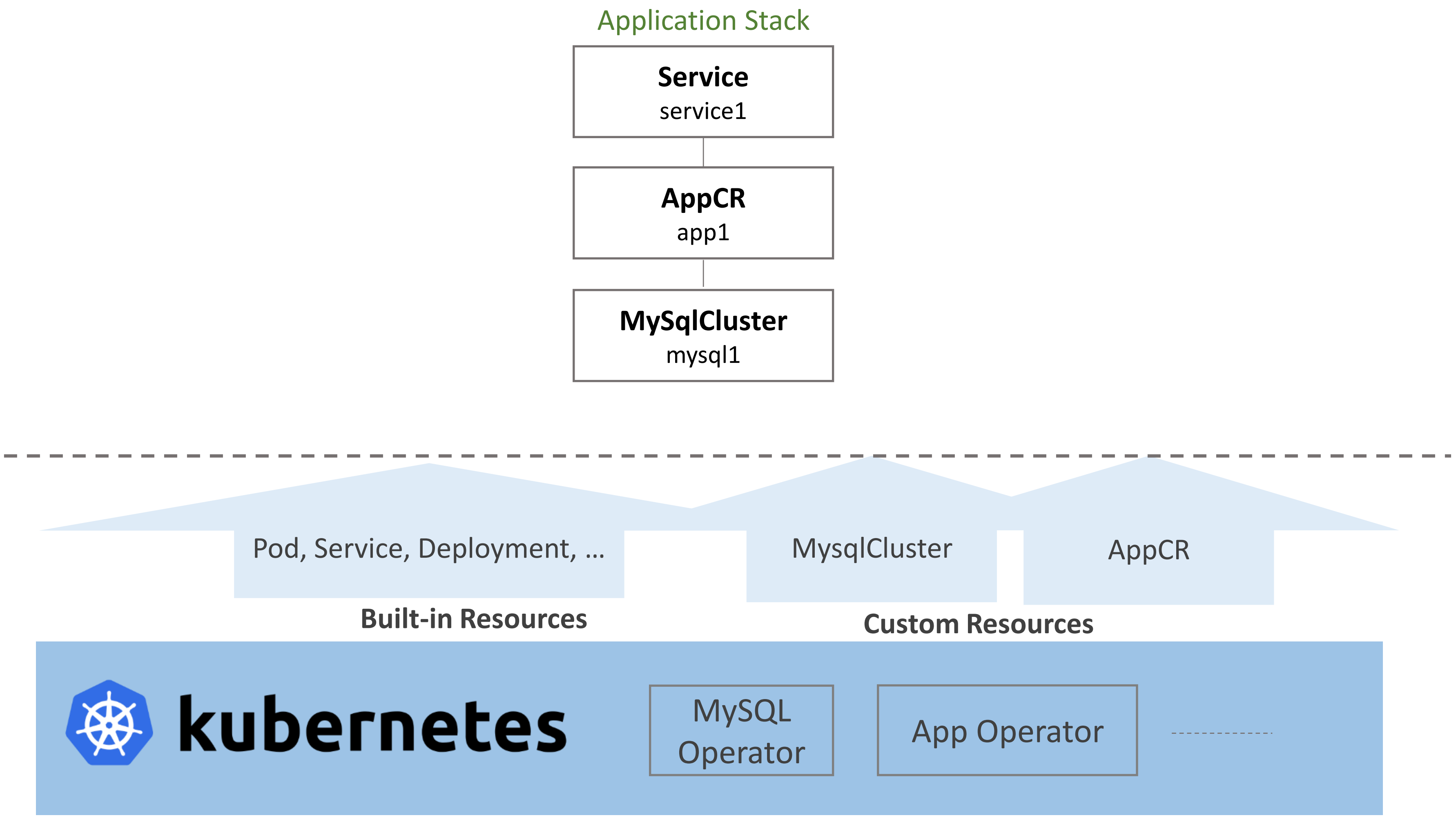
10+ years industry experience in the areas of cloud native technologies, OpenStack, PaaS and distributed systems. Adjunct Professor at UT Austin (CS Dept) teaching courses on cloud computing and modern web applications.
LinkedIn

Kubernetes Operator pattern has become widely popular for running various kinds of applications on Kubernetes.



Operators are getting built for stateful applications, complex services, internal IT workflows, etc.

Operators add Custom Resources to a Cluster



Application Stack

Service
service1

AppCR
app1

MySqlCluster
mysql1

Pod, Service, Deployment, …

MysqlCluster

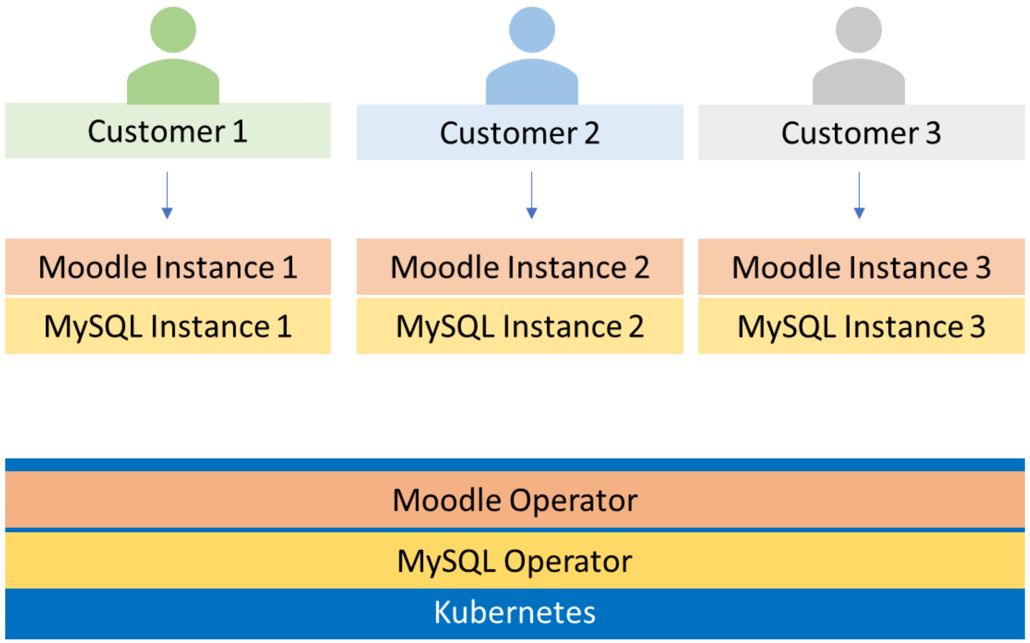
AppCR

Built-in Resources

Custom Resources

PaaSes with Kubernetes Operators

DevOps teams are now using Kubernetes Operators to build their custom PaaSes.



Customer 1
Customer 2
Customer 3

Moodle Instance 1
Moodle Instance 2
Moodle Instance 3

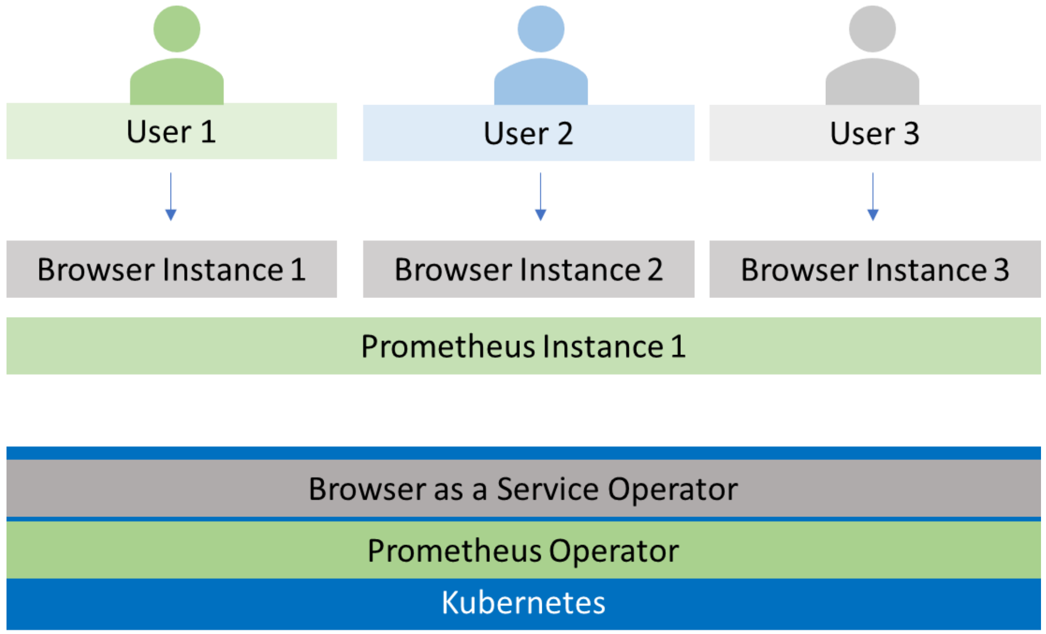
MySQL Instance 1
MySQL Instance 2
MySQL Instance 3

Moodle Operator

MySQL Operator

Kubernetes

Moodle PaaS



User 1
User 2
User 3

Browser Instance 1
Browser Instance 2
Browser Instance 3

Prometheus Instance 1

Browser as a Service Operator

Prometheus Operator

Kubernetes

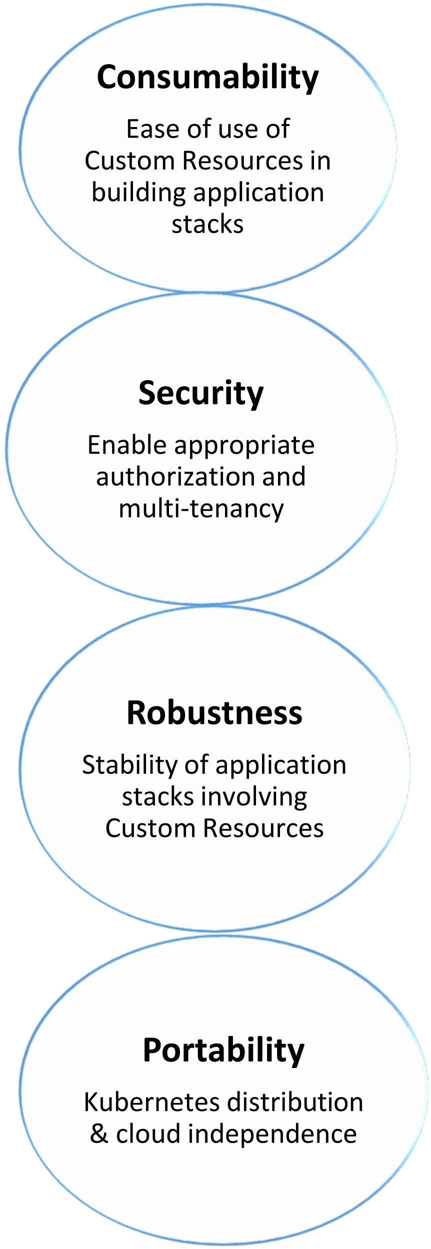
Browser-as-a-Service PaaS

Multi-tenant and multi-Operator environments

Challenge in building PaaSes using Operators

Is my Operator ready for the multi-tenant and multi-Operator world?

Kubernetes Operator Maturity Model



Consumability

Ease of use of Custom Resources in building application stacks

Security

Enable appropriate authorization and multi-tenancy

Robustness

Stability of application stacks involving Custom Resources

Portability

Kubernetes distribution & cloud independence

Consumability

Consumability guidelines focus on design of Custom Resources.

1. Design Custom Resource as a declarative API and avoid inputs as imperative actions
2. Make Custom Resource Type definitions compliant with Kube OpenAPI
3. Consider using kubectl as the primary interaction mechanism
4. Expose Operator developer's assumptions and requirements about Custom Resources
5. Use ConfigMap or Custom Resource Annotation or Custom Resource Spec definition as an input mechanism for configuring the underlying software

Security

Security guidelines help in applying authorization controls and building multi-tenant application stacks using Kubernetes resources (built-in and custom).

6. Define Service Account for Operator Pod
7. Define Service Account for Custom Resources
8. Define SecurityContext and PodSecurityPolicies for Custom Resources
9. Make Custom Controllers Namespace aware
10. Define Custom Resource Node Affinity rules
11. Define Custom Resource Pod Affinity rules
12. Define NetworkPolicy for Custom Resources

Robustness

Robustness guidelines offer guidance in designing Custom Resources so that application stacks built using them are stable and robust.

13. Set OwnerReferences for underlying resources owned by your Custom Resource
14. Define Resource limits and Resource requests for Custom Resources
15. Define Custom Resource Spec Validation rules as part of Custom Resource Definition YAML
16. Design for robustness against side-car injection into Custom Resource Pods
17. Define Custom Resource Anti-Affinity rules
18. Define Custom Resource Taint Toleration rules
19. Define PodDisruptionBudget for Custom Resources
20. Enable Audit logs for Custom Resources
21. Decide Custom Resource Metrics Collection strategy

Portability

Portability guidelines focus on Operator and Custom Resource properties that enable deploying the Operators and the application stacks on any Kubernetes distribution, on-prem or on cloud.

22. Package Operator as Helm Chart
23. Register CRDs as YAML Spec in Helm chart rather than in Operator code
24. Include CRD installation hints in Helm chart

https://github.com/cloud-ark/kubeplus/blob/master/Guidelines.md

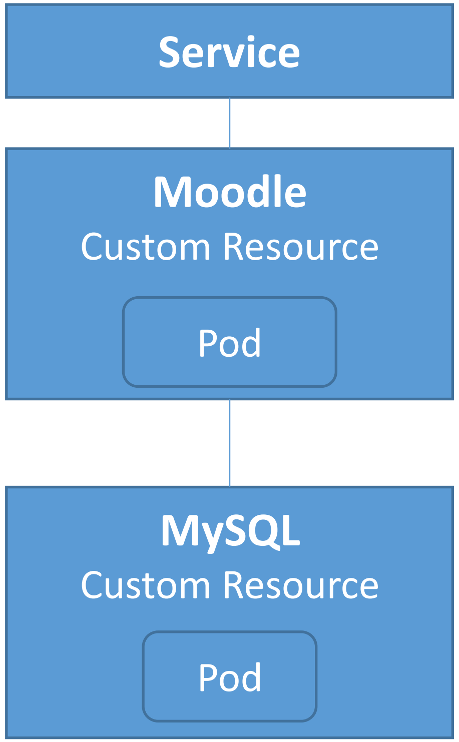
Representative questions

CloudARK

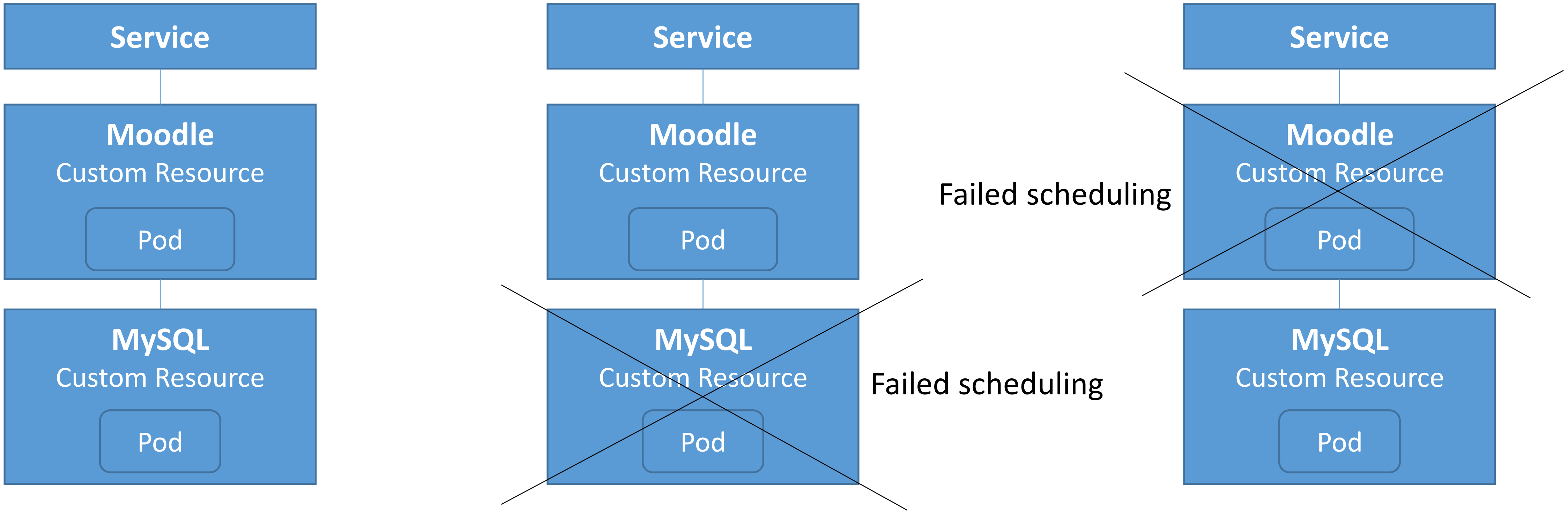
- How to enable *atomic deployments* of application stacks?
- How to enable *co-location* of stack components?
- How to make application stacks *robust against Pod restarts*?
- How to perform *accurate chargebacks* per application stack*?*

https://github.com/cloud-ark/kubeplus/blob/master/Guidelines.md

Moodle PaaS


Service
Moodle
Custom Resource
Pod
MySQL
Custom Resource
Pod

Atomic deployments

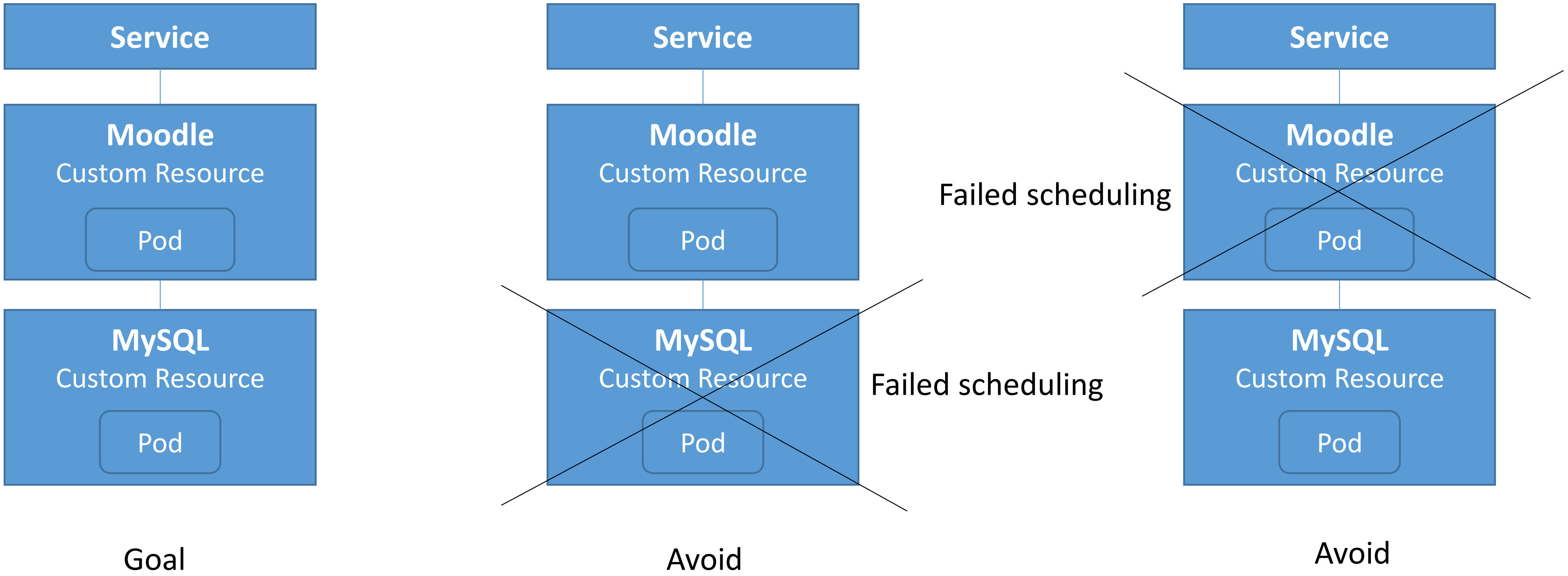


Goal

Avoid

Avoid

Atomic deployments

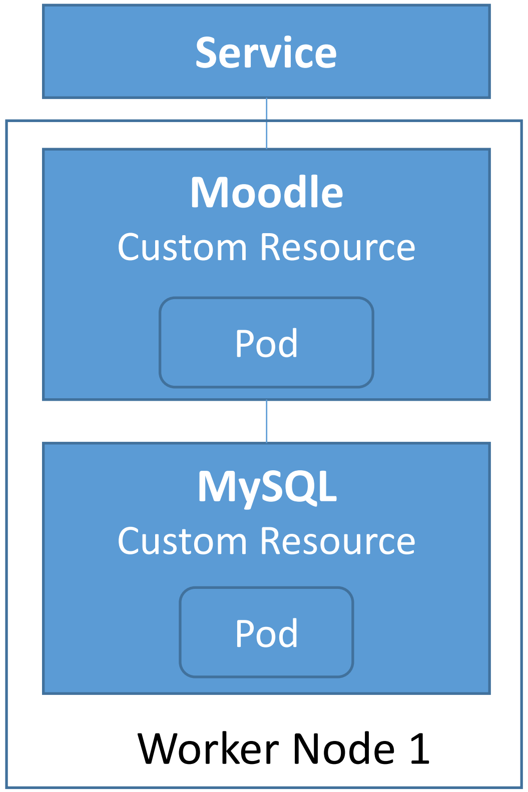


Achieve atomic deployments (guaranteed scheduling) by:

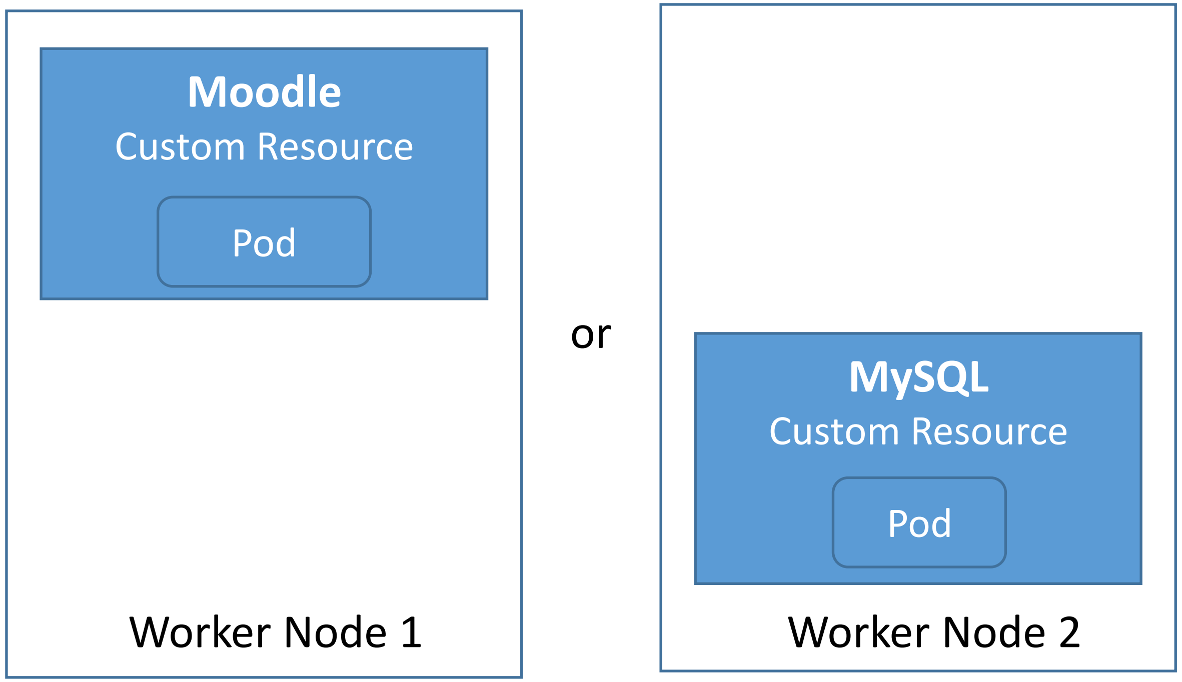
- Defining properties for ‘resource requests’ and ’resource limits’ in your Custom Resource Spec
- Implementing Custom Controller to pass these scheduling hints to the underlying Pod

Co-location

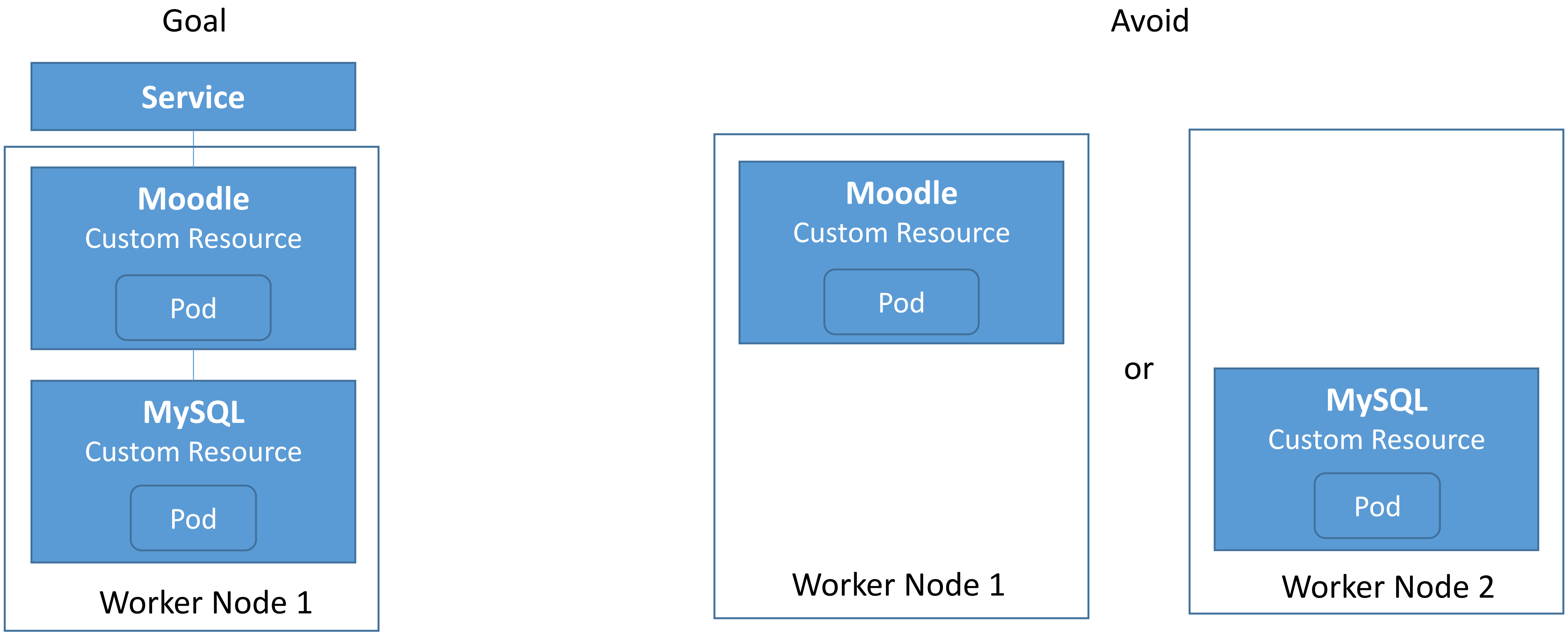
Goal



Avoid



Co-location

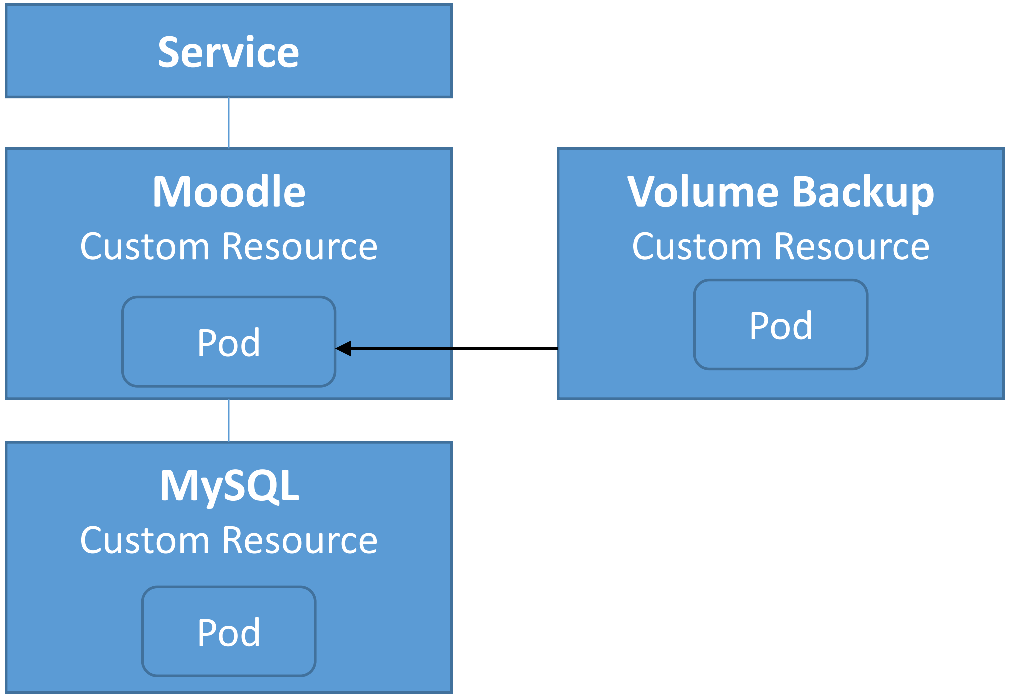


Achieve co-location by:

- Defining properties for Node selection (nodeSelector, Pod Affinity labels and selectors) in your Custom Resource Spec
- Implementing Custom Controller to pass the Node selection criteria to the underlying Pod

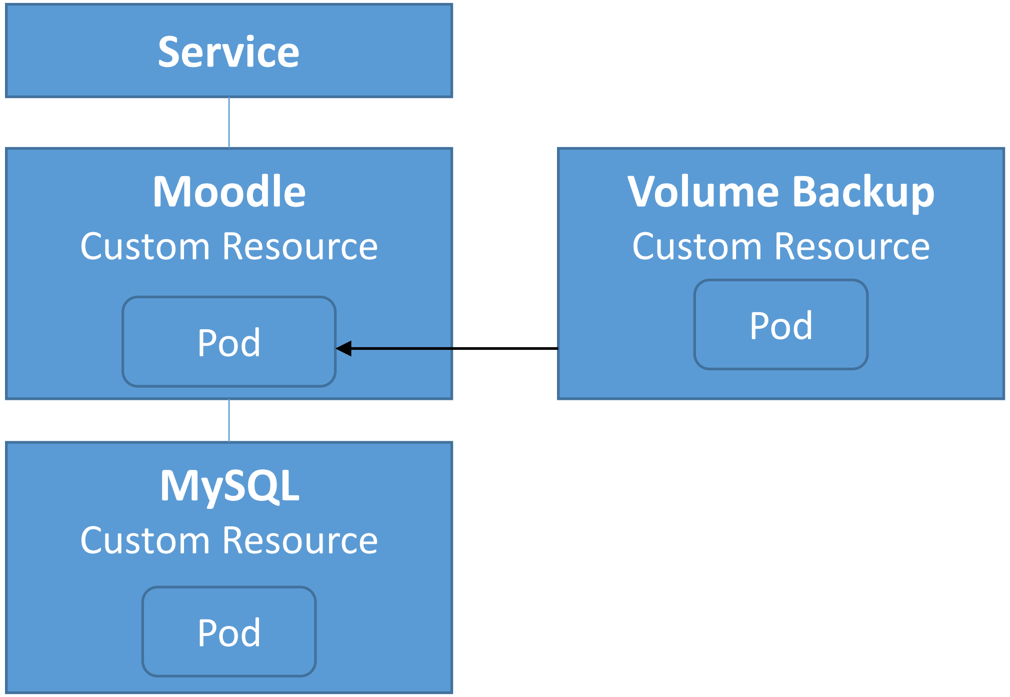
Pod restarts

Example – Side car injection



Pod restarts

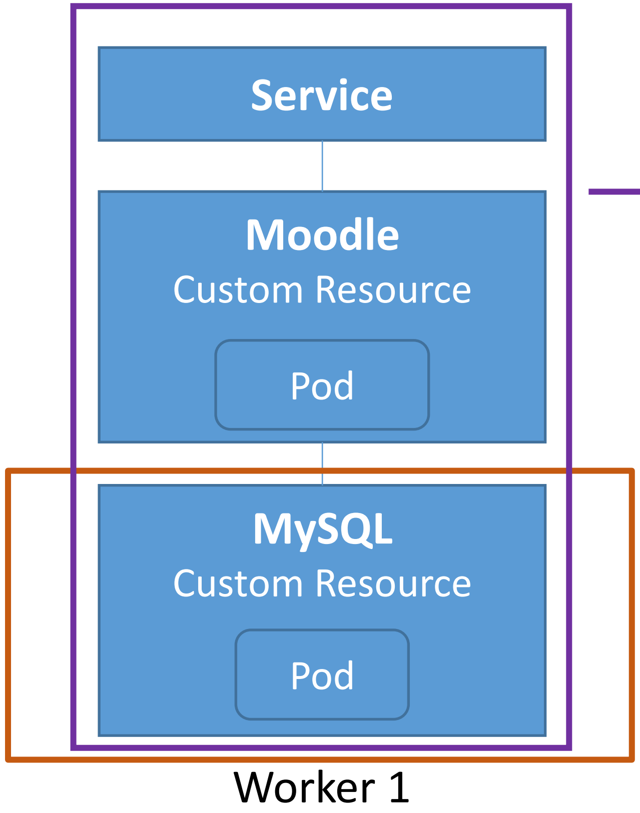
Example – Side car injection



Handle side-car injections by:

- Subscribing to Pod restart events
- Implement Operator Custom Controller to reconcile based on sub-resource events

Stack level charge-backs



What is CPU/Memory consumption of an application stack?

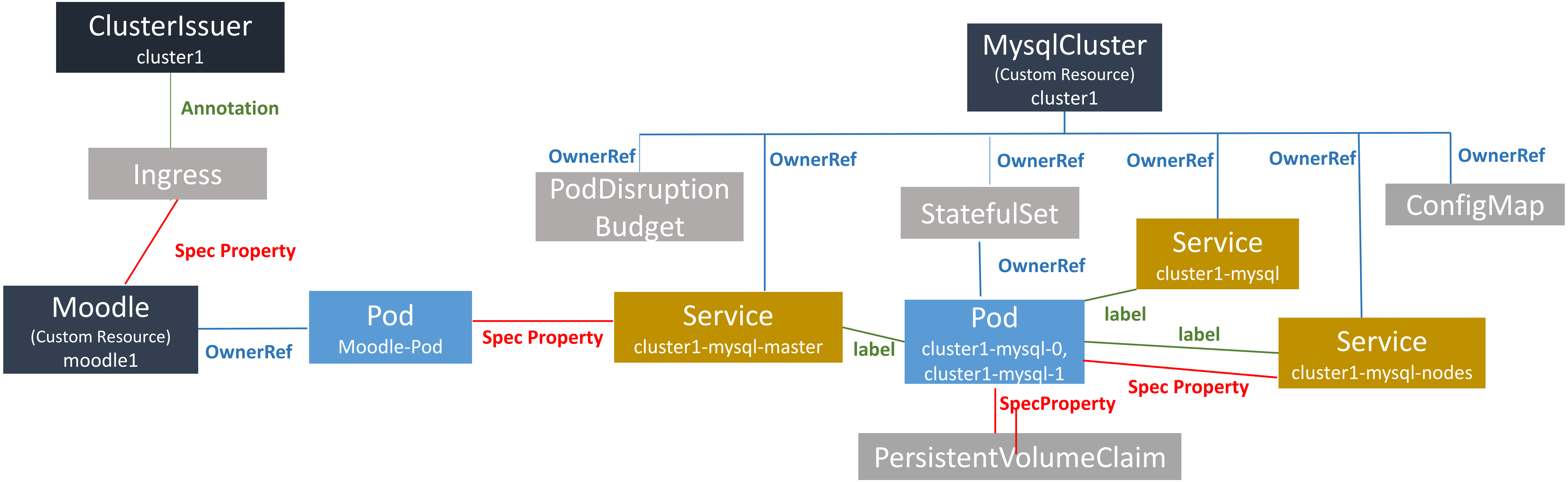
What is CPU/Memory consumption of a Custom Resources?

Stack level charge-backs

CloudARK

Kubernetes resource relationship graphs

https://github.com/cloud-ark/kubeplus



Types of Kubernetes resource relationships:

(1) OwnerRef

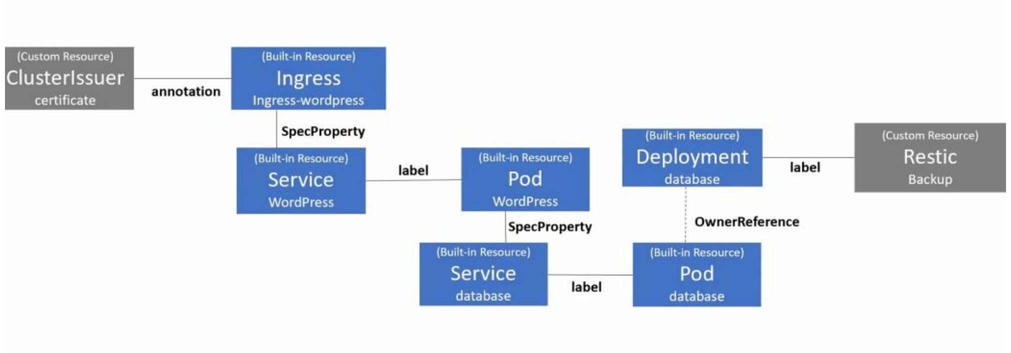
(2) Labels and (3) Annotations

(4) Spec Property

KubePlus

CloudARK

Open source tooling to simplify use of Custom Resources introduced by Kubernetes Operators



KubePlus constructs resource relationship graphs at run-time that include Custom Resources and their dependencies.

Kubectl Plugins

Inventory
# kubectl connections

Custom Resource aware
Resource topologies

Troubleshoot
# kubectl grouplogs

Aggregate logs at stack or
Custom Resource level

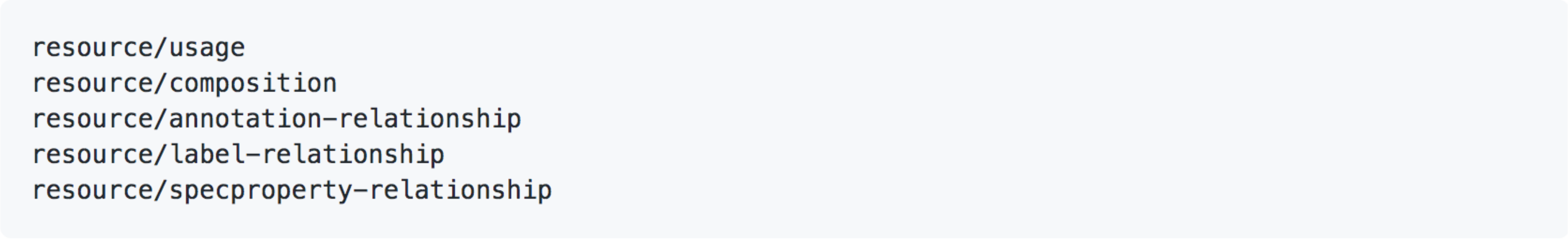
Monitor
# kubectl metrics

Ability to aggregate CPU/ Memory/
Storage for Custom Resources

https://github.com/cloud-ark/kubeplus

Capturing Operator developer’s assumptions

CRD annotations



```
resource/usage
resource/composition
resource/annotation-relationship
resource/label-relationship
resource/specproperty-relationship
```

1.) On MysqlCluster CRD:
- resource/composition: StatefulSet, Service, ConfigMap, Secret, PodDisruptionBudget

2.) On Moodle CRD:
- resource/composition: Deployment, Service, ConfigMap, Secret, PersistentVolumeClaim

3.) On ClusterIssuer CRD:
- resource/annotation-relationship: on:Ingress, key:cert-manager.io/cluster-issuer, value:INSTANCE.metadata.name

Community Operator CRD Annotations:

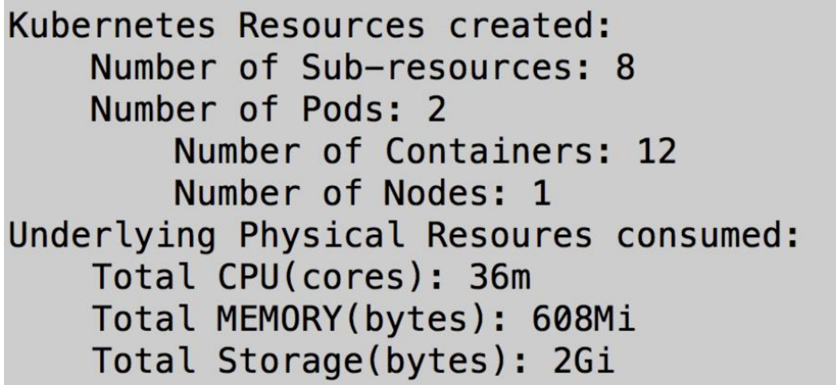
- https://github.com/cloud-ark/kubeplus/blob/master/Operator-annotations.md

Foundation for application stack level charge backs

# kubectl metrics

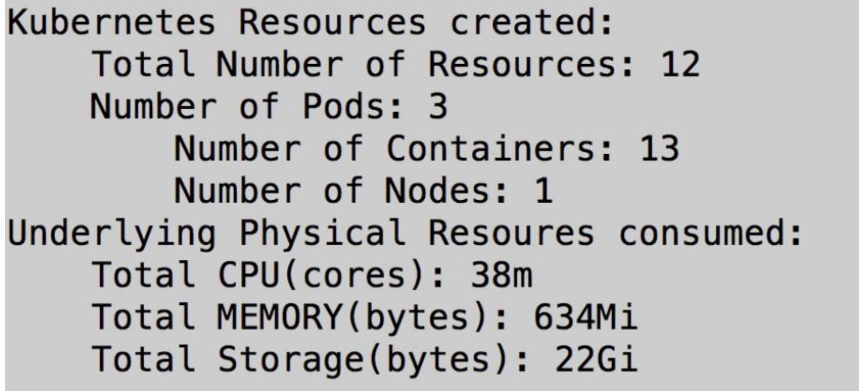
Aggregate CPU/ Memory/ Storage at application stack level

1. # kubectl metrics cr MysqlCluster cluster1



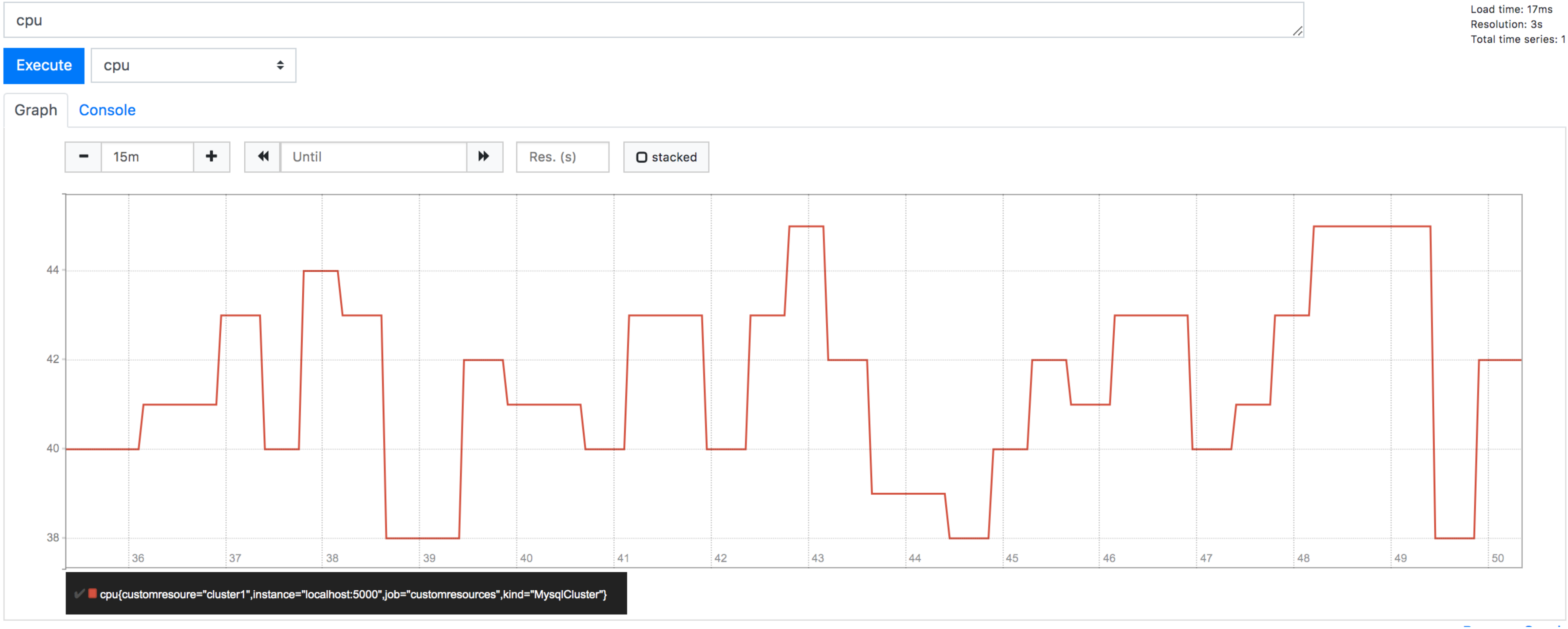
```
Kubernetes Resources created:
    Number of Sub-resources: 8
    Number of Pods: 2
        Number of Containers: 12
        Number of Nodes: 1
Underlying Physical Resoures consumed:
    Total CPU(cores): 36m
    Total MEMORY(bytes): 608Mi
    Total Storage(bytes): 2Gi
```

2. # kubectl metrics service moodle



```
Kubernetes Resources created:
    Total Number of Resources: 12
    Number of Pods: 3
        Number of Containers: 13
        Number of Nodes: 1
Underlying Physical Resoures consumed:
    Total CPU(cores): 38m
    Total MEMORY(bytes): 634Mi
    Total Storage(bytes): 22Gi
```

Prometheus Integration



cpu

Execute cpu

Graph Console

15m Until Res. (s) □ stacked

Load time: 17ms
Resolution: 3s
Total time series: 1

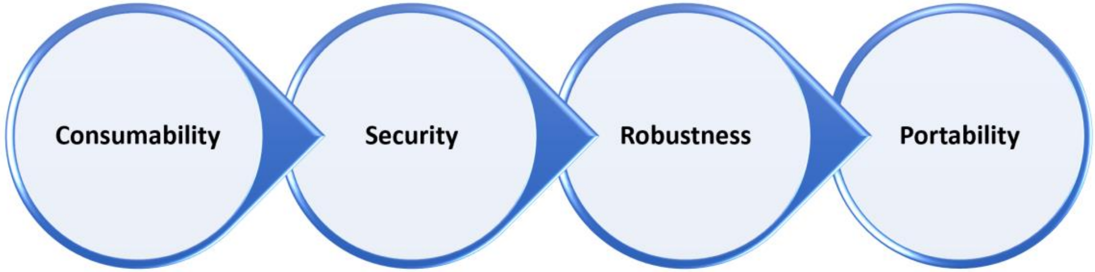
✓ 🟥 cpu{customresoure="cluster1",instance="localhost:5000",job="customresources",kind="MysqlCluster"}

Platform-as-Code

Create PaaSes using Kubernetes Operators

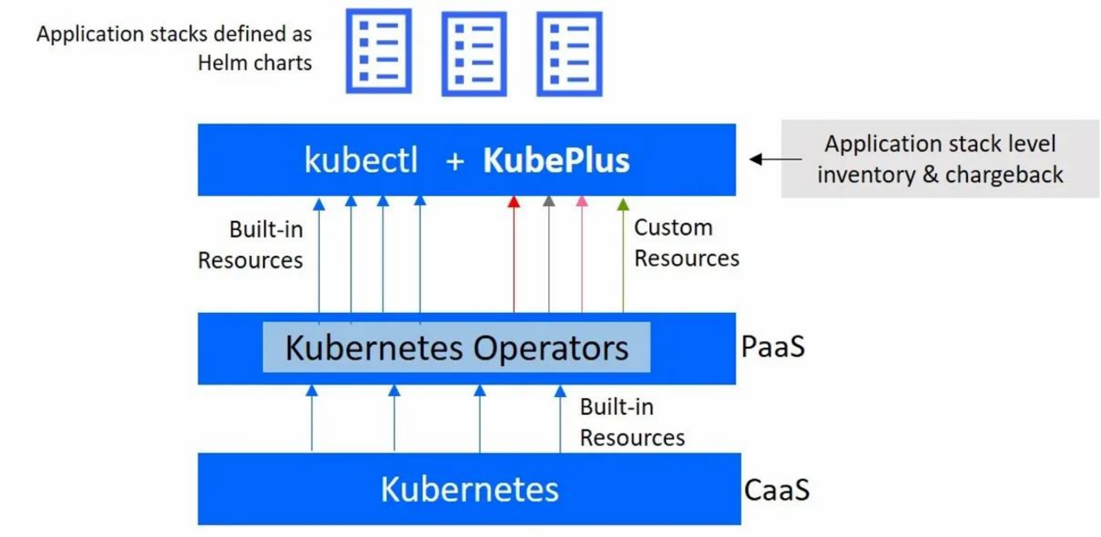
Operator maturity model

Operator readiness guidelines for multi-tenant and multi-Operator environment



KubePlus

Inventory and charge-back for application stacks built using Operators



https://github.com/cloud-ark/kubeplus

Thank you!

Devdatta Kulkarni

Founder @ CloudARK

devdatta@cloudark.io

CloudARK

cloudark.io