

# KubePlus

Multi-tenant application stacks on Kubernetes

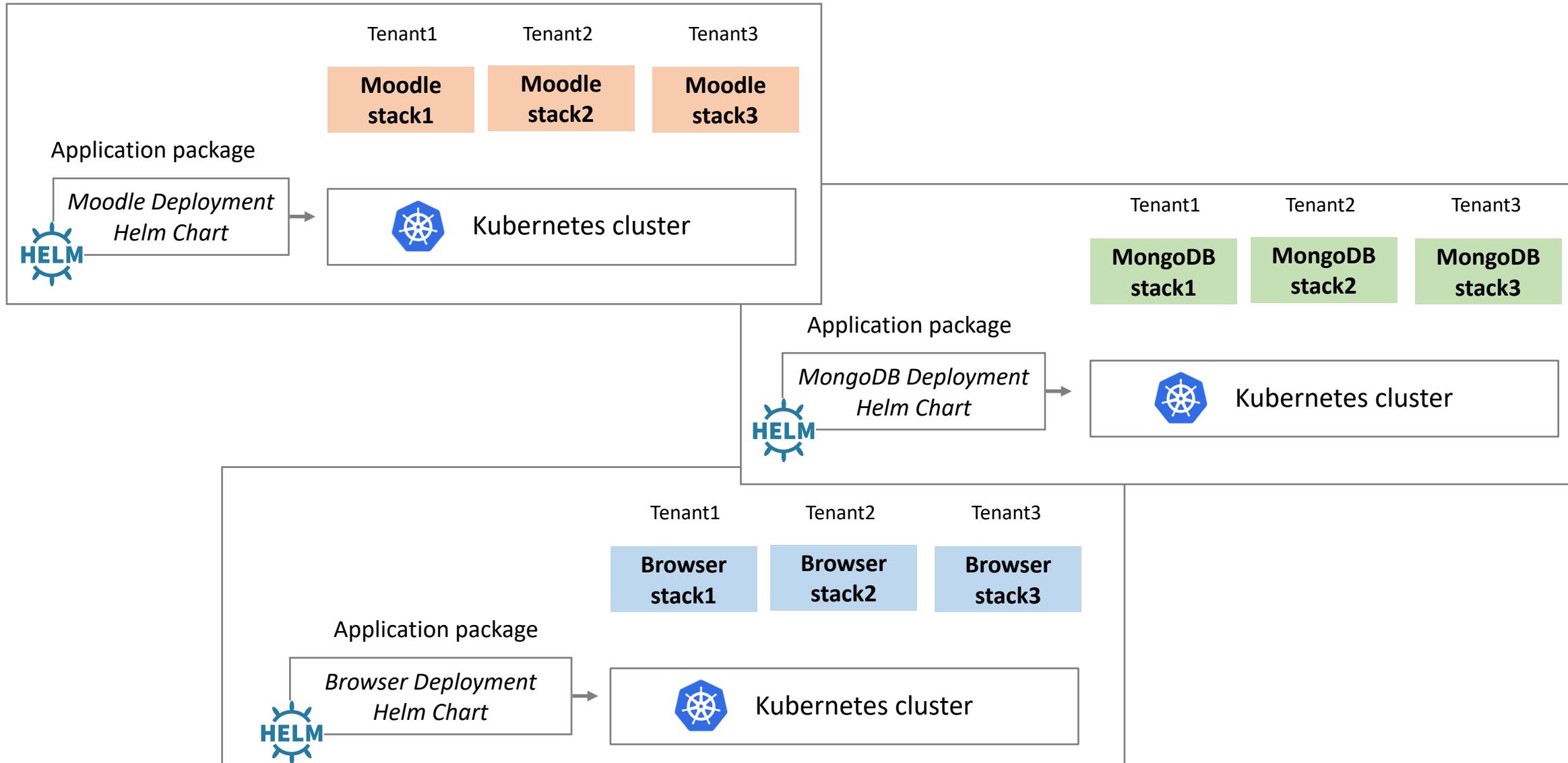
<https://github.com/cloud-ark/kubeplus>

Devdatta Kulkarni  
Founder, CloudARK

<https://www.linkedin.com/in/devdatta-kulkarni-192ab46/>

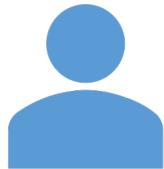
# Application as-a-Service on K8s

Separate application stack per tenant



# Challenge: Multi-tenancy Management

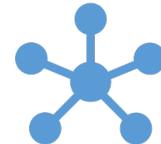
How to isolate and manage per-tenant Kubernetes resources?



Tenant level policies on Helm charts  
E.g., Separate node per tenant



Tenant level consumption metrics  
E.g., CPU, Memory, Storage, Network consumption per tenant



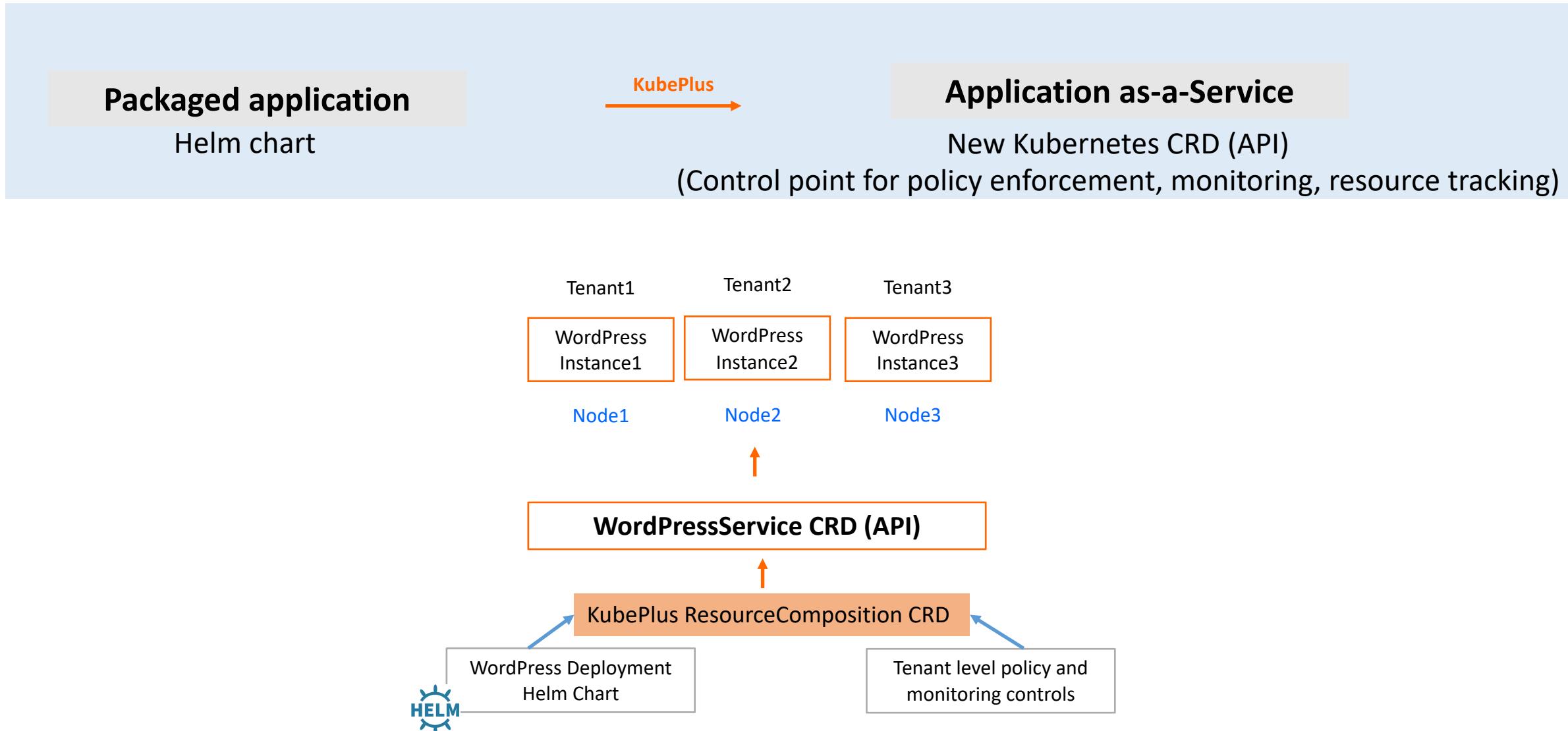
Tenant level resource topologies  
Kubernetes resource relationship graph with relations between resources like Pod, Service, Custom Resource etc.

DIY approach: Based on Labels is limiting

There is no easy way to apply labels on all Kubernetes resources in an application stack, especially if the stack contains Custom Resources for which finding all the sub-resources is not easy.

# Our solution – KubePlus

Basic Idea: Wrap a Kubernetes-native API around Helm chart



# CRD for CRD - Create WordpressService

## Create a sample WordPress Service

```
apiVersion: workflows.kubeplus/v1alpha1
kind: ResourceComposition
metadata:
  name: wordpress-service-composition
spec:
  # newResource defines the new CRD to be installed define a workflow.
  newResource:
    resource:
      kind: WordpressService
      group: platformapi.kubeplus
      version: v1alpha1
      plural: wordpressservices
  # URL of the Helm chart that contains Kubernetes resources that represent a workflow.
  chartURL: https://github.com/cloud-ark/kubeplus/blob/master/examples/multitenancy/wordpress-mysqlcluster-stack/wordpress-mysqlcluster-chart-0.0.1.tgz?raw=true
  chartName: wordpress-mysqlcluster-chart

respolicy:
  apiVersion: workflows.kubeplus/v1alpha1
  kind: ResourcePolicy
  metadata:
    name: wordpress-service-policy
  spec:
    resource:
      kind: WordpressService
      group: platformapi.kubeplus
      version: v1alpha1
    policy:
      # Add following requests and limits for the first container of all the Pods that are related via
      # owner reference relationship to instances of resources specified above.
      podconfig:
        limits:
          cpu: 200m
          memory: 2Gi
        requests:
          cpu: 100m
          memory: 1Gi
        nodeSelector: values.nodeName
```

# Creating instance of WordPress Service

Create a WordPress Service instance – wp-service-tenant1

```
apiVersion: platformapi.kubeplus/v1alpha1
kind: WordpressService
metadata:
  name: wp-service-tenant1
  namespace: wp-stack-ns1
spec:
  namespace: wp-stack-ns1
  tenantName: tenant1
  nodeName: gke-cluster-4-default-pool-dacc3ab3-1x4v
```

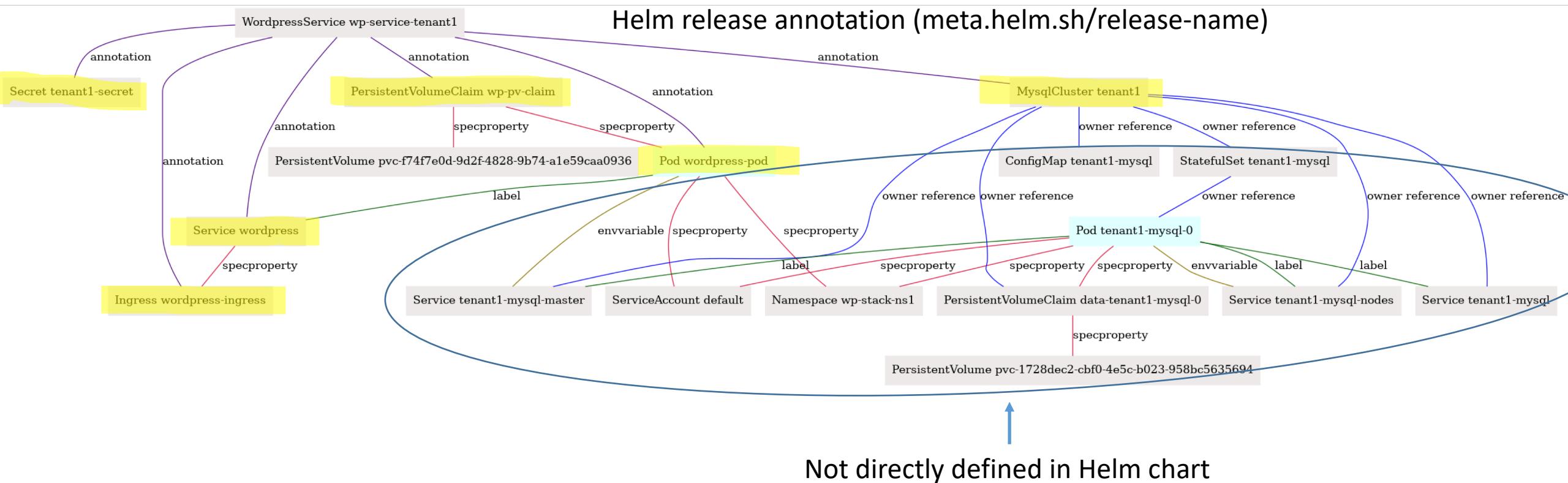


Spec attributes are  
attributes from  
values.yaml



# Tenant1 resource topology

**kubectl connections WordpressService wp-service-tenant1 wp-stack-ns1**



KubePlus discovers Kubernetes resource relationships based on: **ownerReferences, labels, annotations and spec properties**

# Verify policies – cpu/memory

## Policy Input

```
policy:  
podconfig:  
  limits:  
    cpu: 200m  
    memory: 2Gi  
  requests:  
    cpu: 100m  
    memory: 1Gi  
nodeSelector: values.nodeName
```

Provided when WordpressService was registered

## CPU/Memory requests and limits on two pods in the tenant1 stack

```
$ kubectl get pods tenant1-mysql-0 -n wp-stack-ns1 -o json | jq -r '.spec.containers[0].resources'  
{  
  "limits": {  
    "cpu": "200m",  
    "memory": "2Gi"  
  },  
  "requests": {  
    "cpu": "100m",  
    "memory": "1Gi"  
  }  
}  
  
1  
  
$ kubectl get pods wordpress-pod -n wp-stack-ns1 -o json | jq -r '.spec.containers[0].resources'  
{  
  "limits": {  
    "cpu": "200m",  
    "memory": "2Gi"  
  },  
  "requests": {  
    "cpu": "100m",  
    "memory": "1Gi"  
  }  
}  
  
2
```

# Verify policies - nodeName

```
apiVersion: platformapi.kubeplus/v1alpha1
kind: WordpressService
metadata:
  name: wp-service-tenant1
  namespace: wp-stack-ns1
spec:
  namespace: wp-stack-ns1
  tenantName: tenant1
  nodeName: gke-cluster-4-default-pool-dacc3ab3-1x4v
```

## Pods running on the specified node

- 3    \$ kubectl get pods tenant1-mysql-0 -n wp-stack-ns1 -o json | jq -r '.spec.nodeName'  
gke-cluster-4-default-pool-dacc3ab3-1x4v
- 4    \$ kubectl get pods wordpress-pod -n wp-stack-ns1 -o json | jq -r '.spec.nodeName'  
gke-cluster-4-default-pool-dacc3ab3-1x4v

# Tenant1 consumption metrics

```
kubectl metrics cr WordpressService wp-service-tenant1 wp-stack-ns1
```

Pretty

```
-----  
Kubernetes Resources created:  
    Number of Sub-resources: -  
    Number of Pods: 2  
        Number of Containers: 8  
        Number of Nodes: 1  
Underlying Physical Resources consumed:  
    Total CPU(cores): 23m  
    Total MEMORY(bytes): 422Mi  
    Total Storage(bytes): 2Gi  
    Total Network bytes received: 90728729.0  
    Total Network bytes transferred: 129797673.0  
-----
```



Prometheus

# Comparison

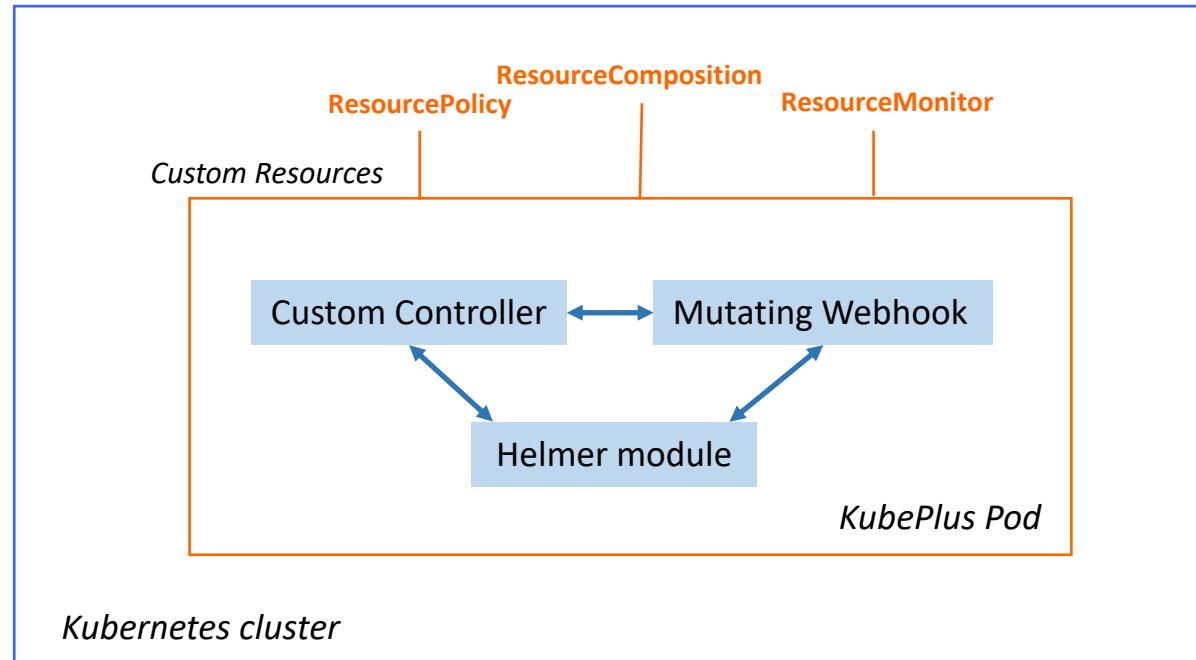
## KubePlus CRD for CRD & helm-operator from Operator framework

CRD for CRD approach allows us to have a single Operator for managing multiple APIs from different Helm charts. With helm-operator a new Operator is created for every Helm chart.

## KubePlus CRD for CRD & OAM

KubePlus is designed with tenant-centric approach and OAM is designed with app-centric approach. KubePlus currently takes Helm chart as the application input and can evolve later to take more complex application definition inputs such as OAM.

# KubePlus – Under the hood



KubePlus  
kubectl plugins

Visualize Kubernetes resource relationships

- CRUD Operations on ResourceComposition: Custom Controller
- CRUD Operations on new Custom API: Mutating Webhook + Helmer
- Policy Enforcement: Mutating Webhook
- Metrics:
  - CPU/Memory/Storage: Pod-level specs
  - Network: cAdvisor
- Resource relationship graphs: ownerReferences, labels, annotations, spec properties

# Conclusion

KubePlus enables creating multi-tenant application stacks from Helm charts declaratively, with policy controls, resource monitoring and topology visualization.

Check out KubePlus:

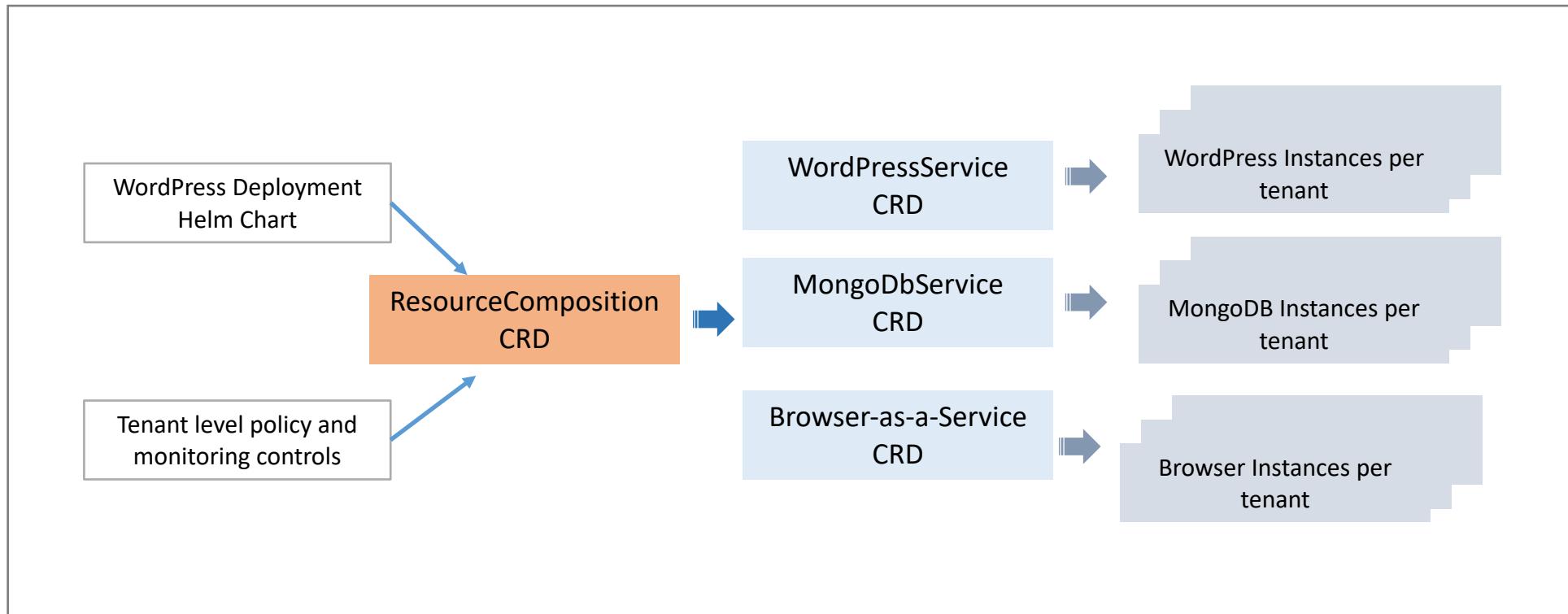
- KubePlus GitHub: <https://github.com/cloud-ark/kubeplus>
- KubePlus Documentation: <https://cloud-ark.github.io/kubeplus/docs/html/html/index.html>
- Platform-as-Code: <https://cloudark.io/platform-as-code>

Looking for feedback from the community.

# Questions?

# KubePlus

Open-source framework to create platform APIs declaratively

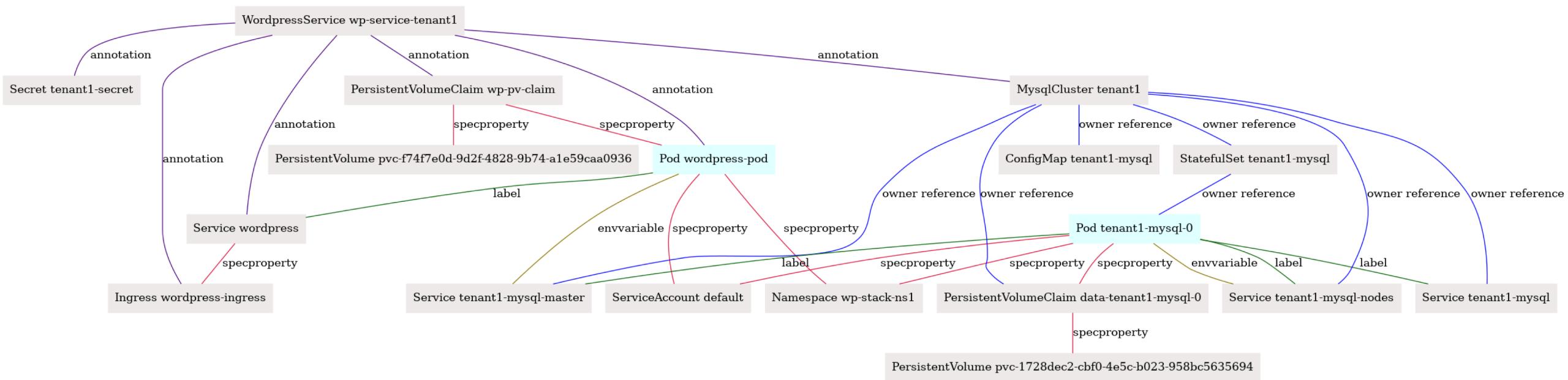


CRD for CRDs

# Tenant1 resource topology

Resource relationship graph

kubectl connections WordpressService wp-service-tenant1 wp-stack-ns1



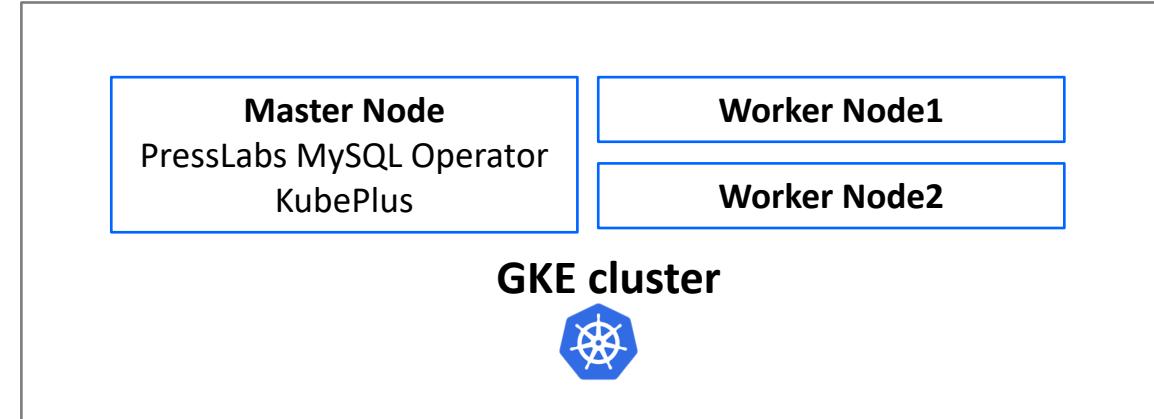
KubePlus discovers Kubernetes resource relationships based on ownerReferences, labels, annotations and spec properties

# Demo scenario

Build a WordPress-as-a-service



- Wordpress Pod
- MysqlCluster CR



Tenant-level policies that we want to enforce in this setup.

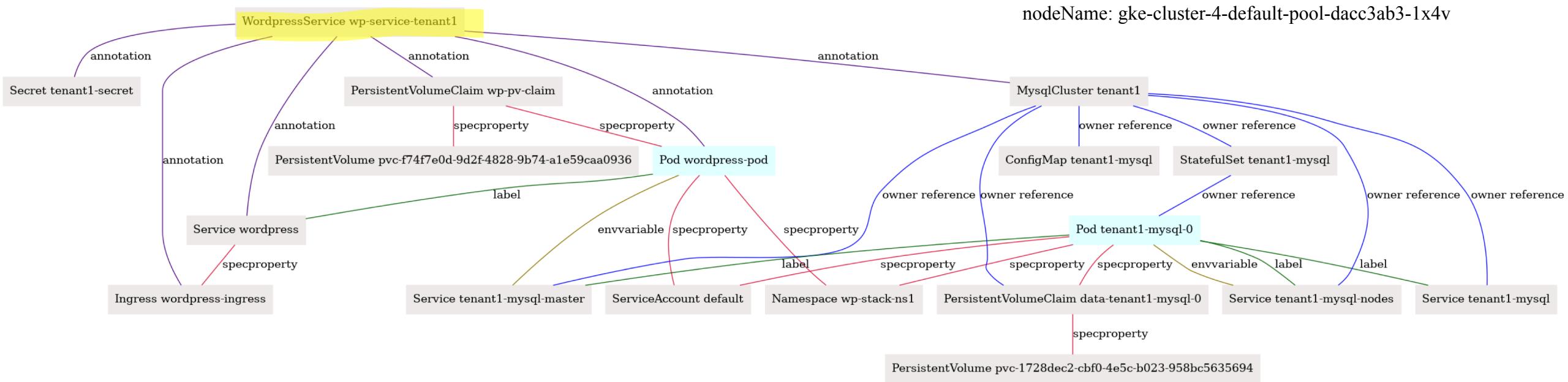
- Every Pod in Wordpress Helm chart should be defined with specific requests and limits for cpu and memory resources
- Wordpress stacks for different tenants should be deployed on different Worker nodes

In Kubernetes these are achieved by setting following attributes in the Pod Spec:

- resource requests/limits
- nodeSelector

# Tenant1 resource topology

## Resource relationship graph



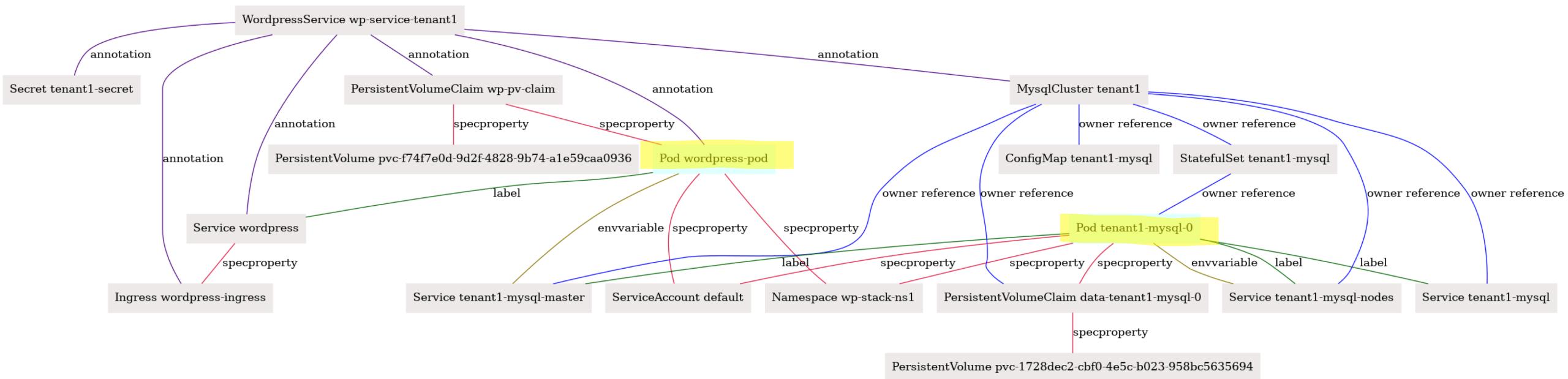
```

apiVersion: platformapi.kubeplus/v1alpha1
kind: WordpressService
metadata:
  name: wp-service-tenant1
  namespace: wp-stack-ns1
spec:
  namespace: wp-stack-ns1
  tenantName: tenant1
  nodeName: gke-cluster-4-default-pool-dacc3ab3-1x4v

```

# Tenant1 resource topology

## Resource relationship graph



# CRD for CRD - Create WordpressService

Create a sample WordPress Service

