# Extending Kubernetes LoadBalancer Using CRDs

IBM **Developer**

Wei Huang (IBM, wei.huang1@ibm.com)

Srini Brahmaroutu (IBM, srbrahma@us.ibm.com)

- sig-scheduling

- sig-testing
- sig-storage

## Agenda

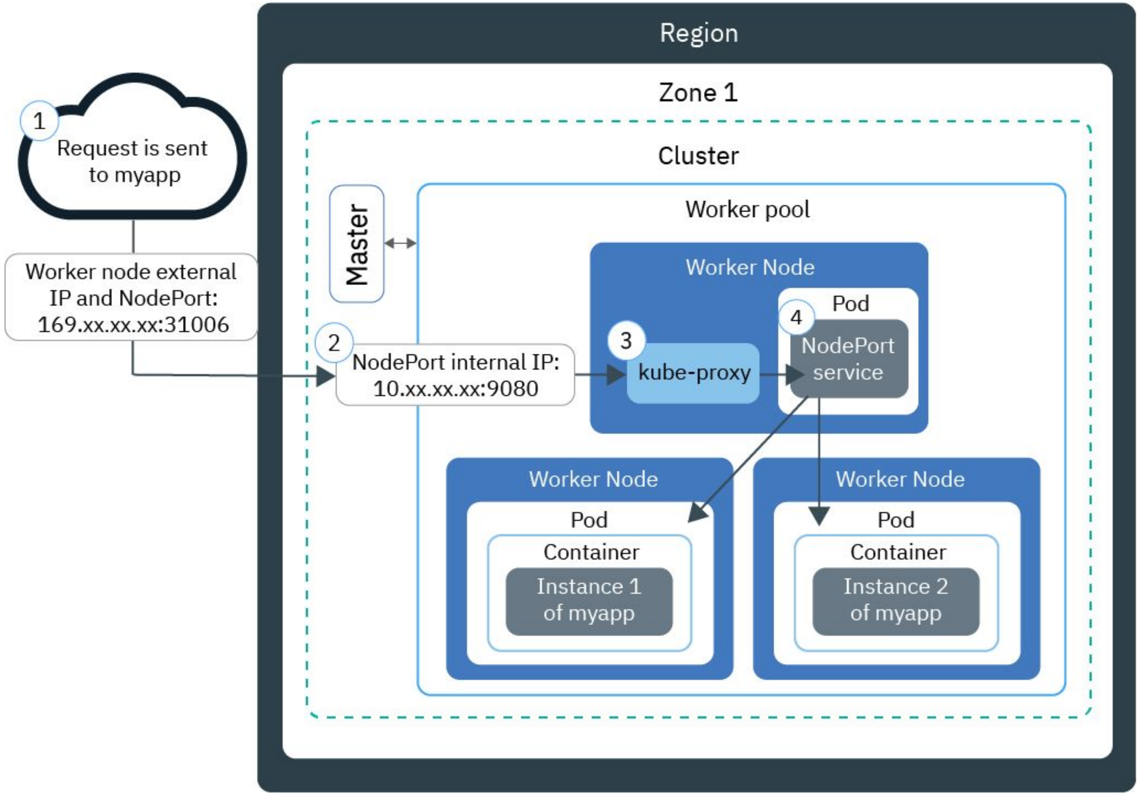
- How to Expose Kubernetes Workloads Externally
- Background / Motivations
- Shared LoadBalancer
- Demos
- Design / Implementation Details

## How to Expose Kubernetes Workloads Externally

# Kubernetes Basics - NodePort Service

## NodePort

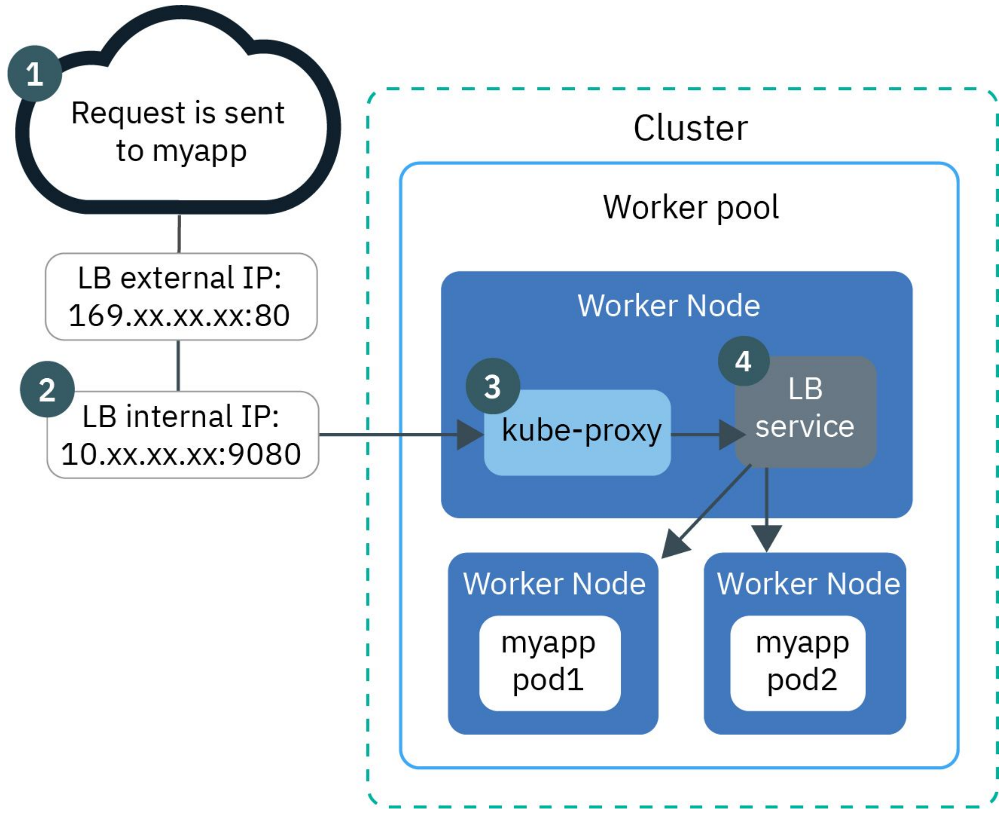
- Worker nodes needs to have a public/external IP
- Ports opened on *all* worker nodes
- Ports range from 30000 to 32767



# Kubernetes Basics - LoadBalancer Service

## LoadBalancer

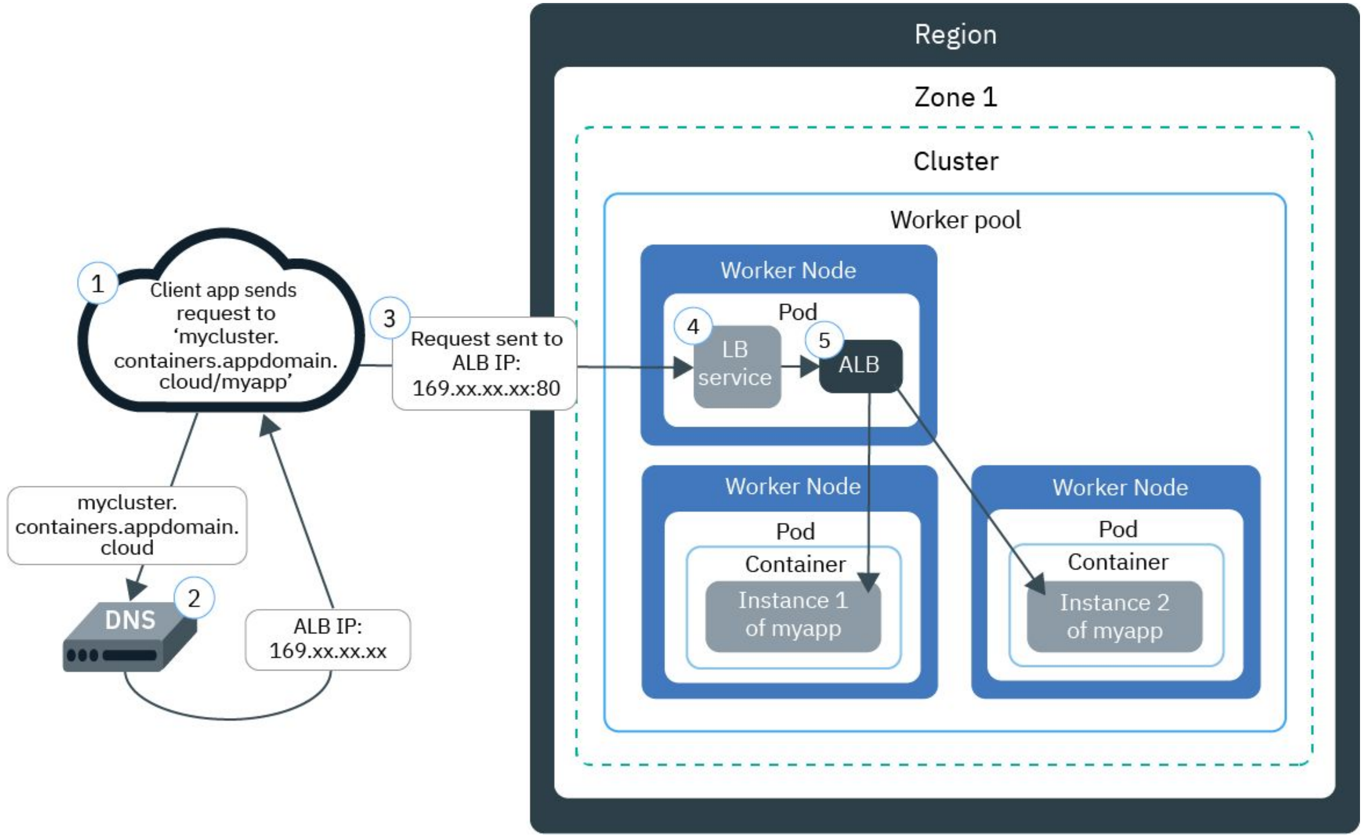
- EKS (Amazon)
- IKS (IBM)
- GKE (Google)
- AKS (Azure)
- ...



# Kubernetes Basics - Ingress

## Ingress

- L7
- Nginx/Envoy/...



IBM **Developer**

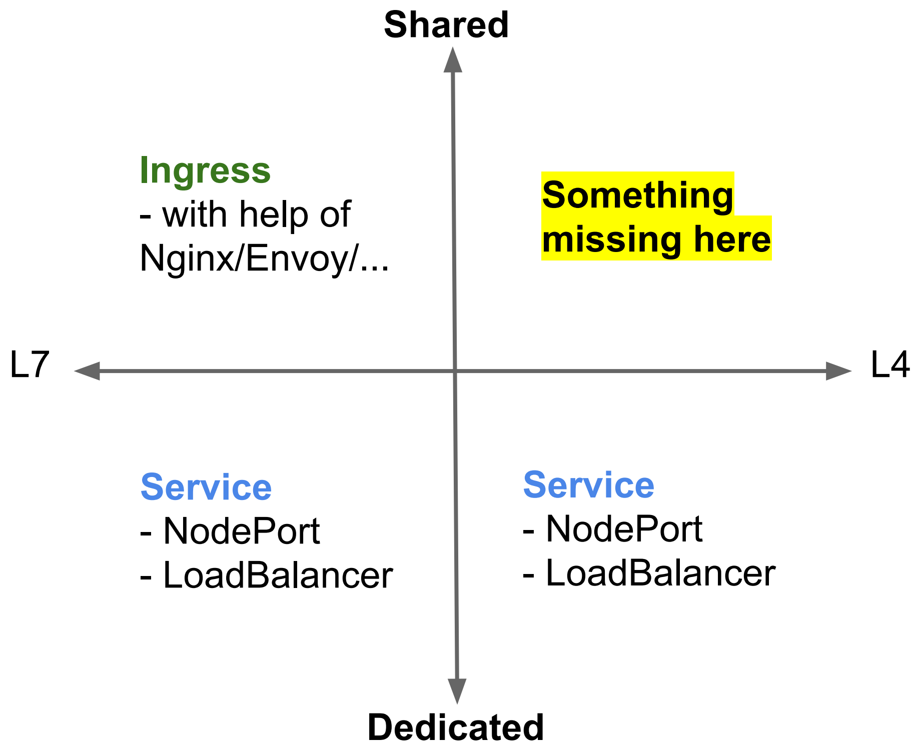
# Ways to Expose K8s Apps Externally

**Service** - for both L4/L7 traffic

- Type NodePort
- Type LoadBalancer

**Ingress** - for L7 traffic

- Shared via ingress controller

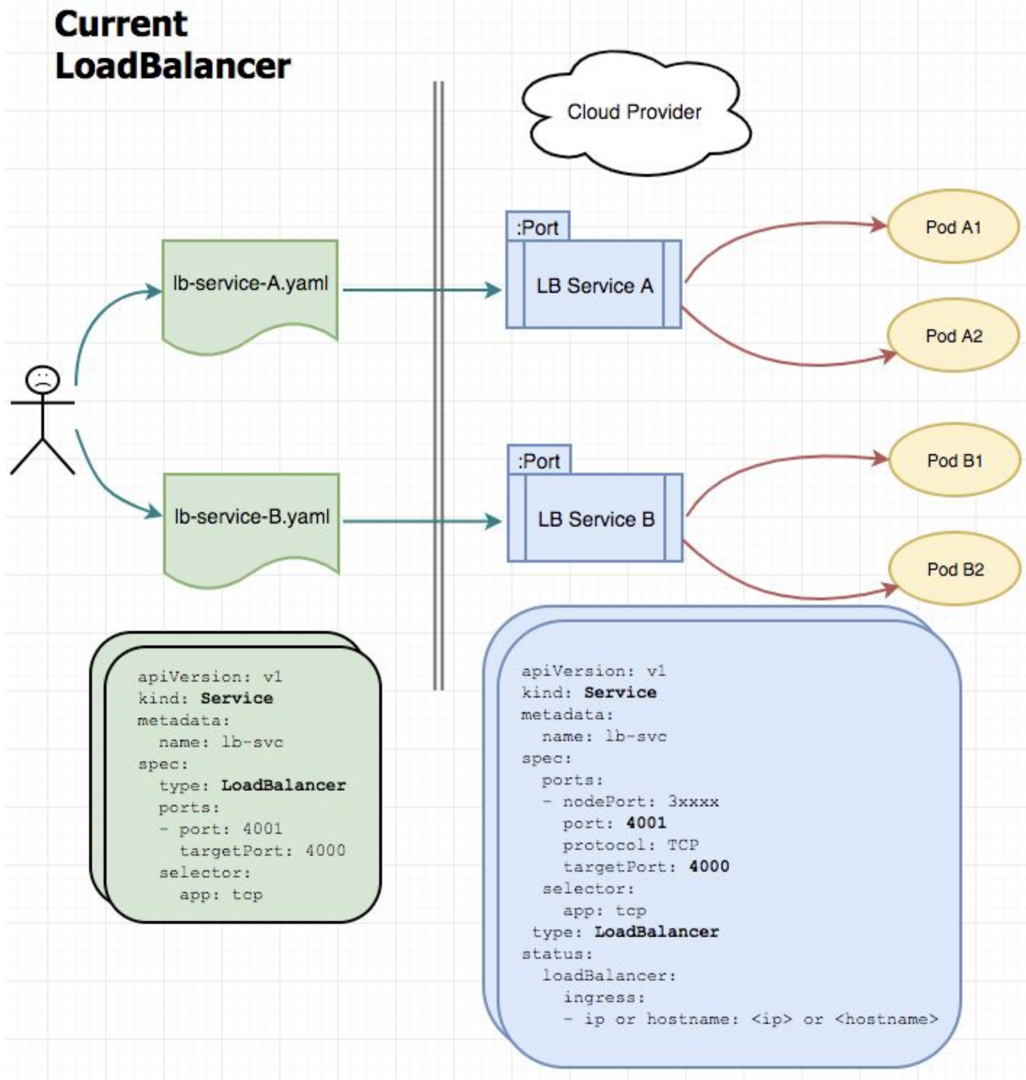


## Background / Motivations

## Background

An internal business requirement from an internal team.

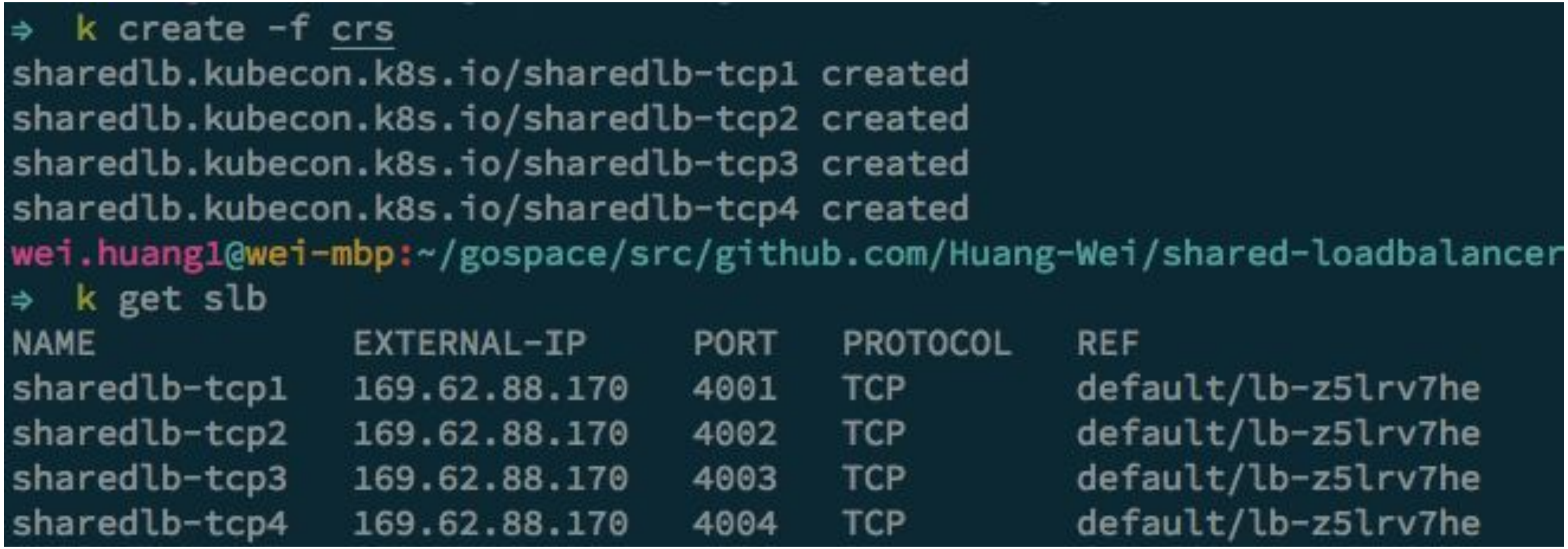
- Two JDBC services (TCP)
- One data transferring service (UDP)
- One web console service (HTTP)



```
apiVersion: v1
kind: Service
metadata:
  name: lb-svc
spec:
  type: LoadBalancer
  ports:
  - port: 4001
    targetPort: 4000
  selector:
    app: tcp
```

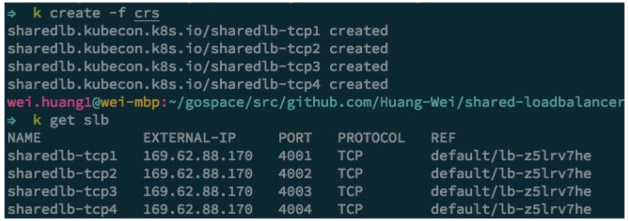
```
apiVersion: v1
kind: Service
metadata:
  name: lb-svc
spec:
  ports:
  - nodePort: 3xxxx
    port: 4001
    protocol: TCP
    targetPort: 4000
  selector:
    app: tcp
 type: LoadBalancer
status:
  loadBalancer:
    ingress:
    - ip or hostname: <ip> or <hostname>
```

## Expected Goal

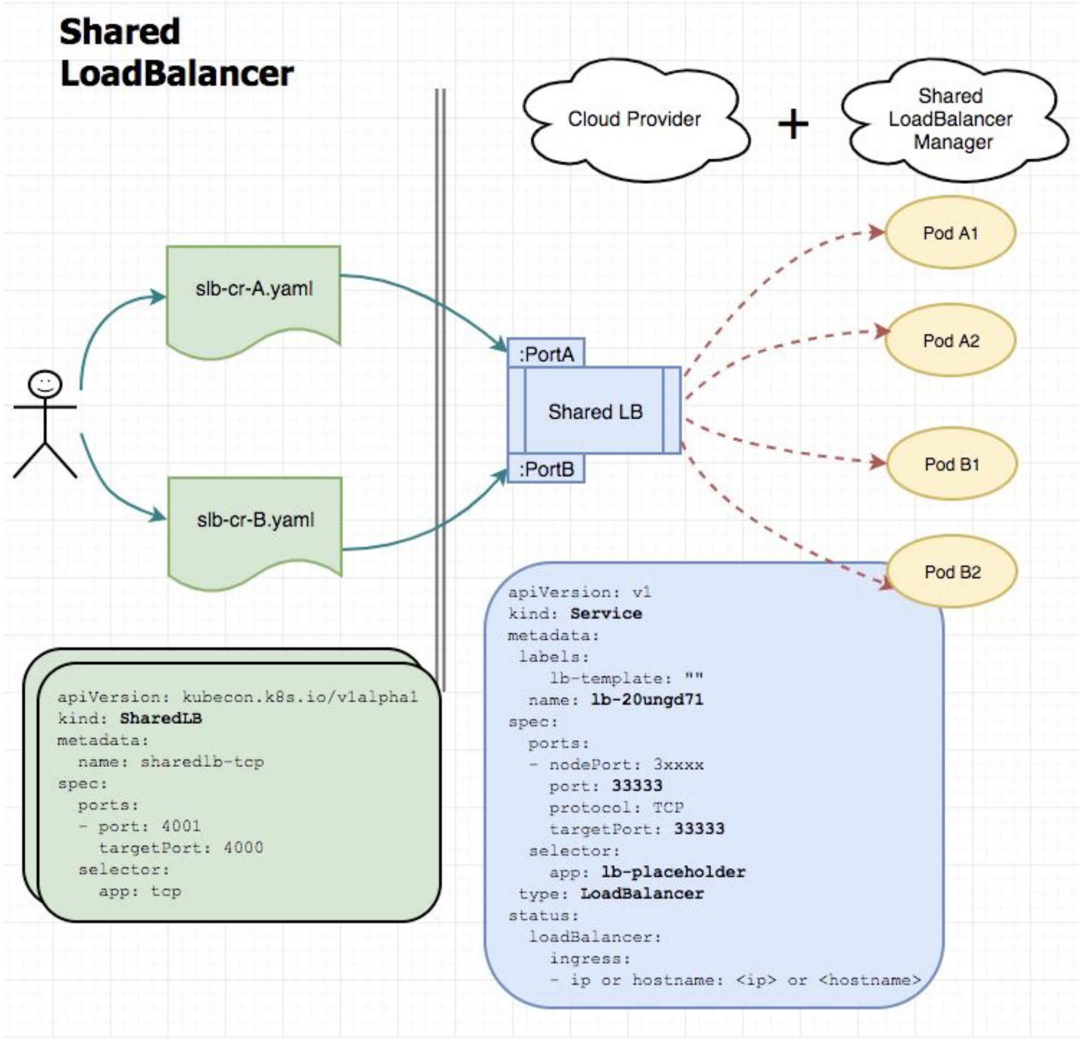


```
➜  k create -f crs
sharedlb.kubecon.k8s.io/sharedlb-tcp1 created
sharedlb.kubecon.k8s.io/sharedlb-tcp2 created
sharedlb.kubecon.k8s.io/sharedlb-tcp3 created
sharedlb.kubecon.k8s.io/sharedlb-tcp4 created
wei.huang1@wei-mbp:~/gospace/src/github.com/Huang-Wei/shared-loadbalancer
➜  k get slb
NAME            EXTERNAL-IP     PORT   PROTOCOL   REF
sharedlb-tcp1   169.62.88.170   4001   TCP        default/lb-z5lrv7he
sharedlb-tcp2   169.62.88.170   4002   TCP        default/lb-z5lrv7he
sharedlb-tcp3   169.62.88.170   4003   TCP        default/lb-z5lrv7he
sharedlb-tcp4   169.62.88.170   4004   TCP        default/lb-z5lrv7he
```

# Expected Goal (cont.)



```
➜  k create -f crs
sharedlb.kubecon.k8s.io/sharedlb-tcp1 created
sharedlb.kubecon.k8s.io/sharedlb-tcp2 created
sharedlb.kubecon.k8s.io/sharedlb-tcp3 created
sharedlb.kubecon.k8s.io/sharedlb-tcp4 created
wei.huang1@wei-mbp:~/gospace/src/github.com/Huang-Wei/shared-loadbalancer
➜  k get slb
NAME            EXTERNAL-IP     PORT   PROTOCOL   REF
sharedlb-tcp1   169.62.88.170   4001   TCP        default/lb-z5lrv7he
sharedlb-tcp2   169.62.88.170   4002   TCP        default/lb-z5lrv7he
sharedlb-tcp3   169.62.88.170   4003   TCP        default/lb-z5lrv7he
sharedlb-tcp4   169.62.88.170   4004   TCP        default/lb-z5lrv7he
```



```
apiVersion: kubecon.k8s.io/v1alpha1
kind: SharedLB
metadata:
  name: sharedlb-tcp
spec:
  ports:
  - port: 4001
    targetPort: 4000
  selector:
    app: tcp
```

```
apiVersion: v1
kind: Service
metadata:
 labels:
    lb-template: ""
  name: lb-20ungd71
spec:
  ports:
  - nodePort: 3xxxx
    port: 33333
    protocol: TCP
    targetPort: 33333
  selector:
    app: lb-placeholder
 type: LoadBalancer
status:
  loadBalancer:
    ingress:
    - ip or hostname: <ip> or <hostname>
```

IBM **Developer**

## Motivations

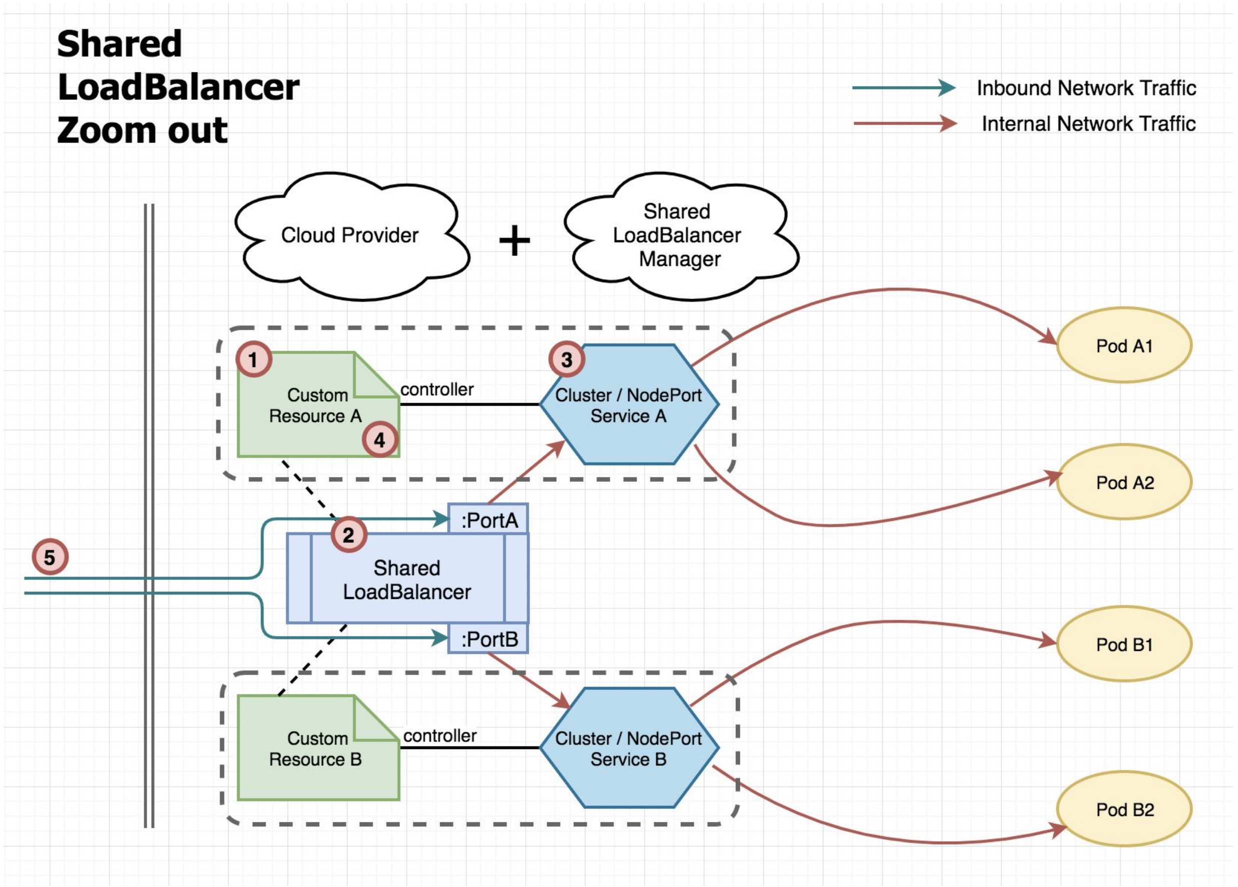
- Cost effective
- User friendly
- Minimum operation efforts
- Reusing existing Kubernetes assets (don’t reinvent wheel)
- Consistent with Kubernetes roadmap

## Shared LoadBalancer (SLB)

## Problem Analysis

1. How to open **additional** ports (and firewall rules) on the “Shared” LoadBalancer
2. How to **associate** the ports with backing pods
3. How to give **accessing info** back to end-user

# Shared LoadBalancer Internals



## Demos

## Design / Implementation Details

## Design Considerations

1. Using CRD as the facade to end-user, ~~instead of Service with annotation~~.
2. Namespaced CRD ~~vs. Clustered CRD~~.
3. Create real LB on demand~~, or prepare placeholder LBs in a pool~~.
4. Make “N” configurable (how many requests one LB can share with)
5. Adopt best practices of CRD controller - controllerRef, finalizers, etc.
6. 1 controller goroutine(worker) for the reconcile loop

## Summary

| | EKS (Amazon) | IKS (IBM) | GKE (Google) | AKS (Azure) |
|---|---|---|---|---|
| **Core Extension Solution** | NodePort Service | Cluster Service with "externalIP" | NodePort Service | NodePort Service |
| **SDK Authentication** | aws_access_key_id<br>aws_secret_access_key | APIKEY<br>*(only needed when adding portable ip quota)* | oauth2<br>*(gcloud auth application-default login)* | Service principle and Role<br>*(az ad sp create-for-rbac)* |
| **Forward Rule Firewall Rule** | Use SDK to operate | Auto Managed | Use SDK to operate | Use SDK to operate |
| **Accessing method** | <Hostname>:<Port> | <IP>:<Port> | <IP>:<Port> | <IP>:<Port> |
| **Limitations** | UDP not supported<br>Latest version is 1.10 | N/A | Random incoming port not supported*<br>Ephemeral IP => Static IP | N/A |

## Thinking in Kubernetes Way

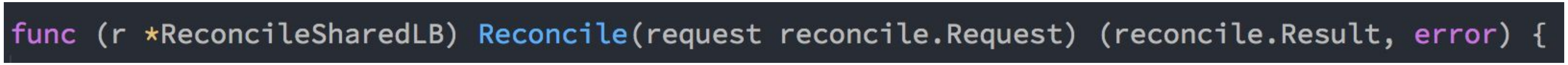
1. Abstraction/Orchestration problem
    a. LoadBalancer <-> Pause Container
    b. {CR Obj, Internal Service} pair <-> Regular Container
2. Scheduling problem
    a. Ports
    b. Resource Requests/Limits
    c. LeastRequested vs. MostRequested
    d. {Anti}-Affinity, Topology Aware
3. Avoid reinventing wheels
    a. Essense of Service on different types
    b. Understand controller loop and CRD design rational

## Thanks!

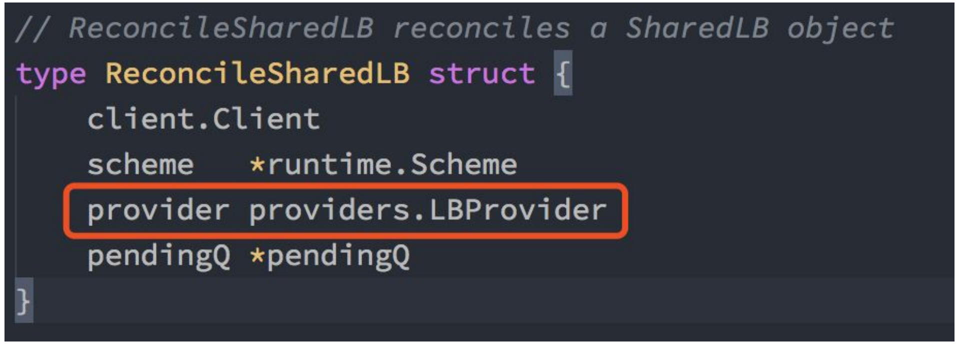
## Q&A

- [github.com/Huang-Wei/shared-loadbalancer](https://github.com/Huang-Wei/shared-loadbalancer)
- Github: @Huang-Wei / @brahmaroutu
- Slack: @Huang-Wei / @srbrahma
- Twitter: @hweicdl / @brahmaroutu

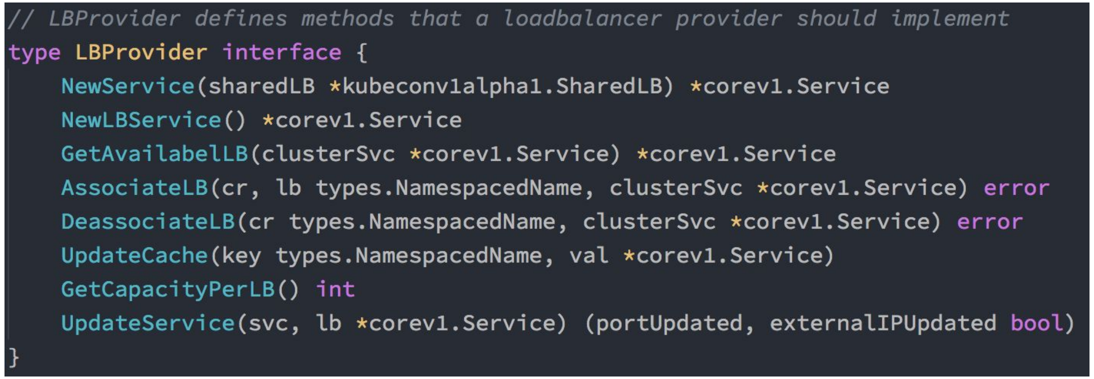
## Some Code Snippets



```go
func (r *ReconcileSharedLB) Reconcile(request reconcile.Request) (reconcile.Result, error) {
```



```go
// ReconcileSharedLB reconciles a SharedLB object
type ReconcileSharedLB struct {
	client.Client
	scheme   *runtime.Scheme
	provider providers.LBProvider
	pendingQ *pendingQ
}
```



```go
// LBProvider defines methods that a loadbalancer provider should implement
type LBProvider interface {
	NewService(sharedLB *kubeconv1alpha1.SharedLB) *corev1.Service
	NewLBService() *corev1.Service
	GetAvailabelLB(clusterSvc *corev1.Service) *corev1.Service
	AssociateLB(cr, lb types.NamespacedName, clusterSvc *corev1.Service) error
	DeassociateLB(cr types.NamespacedName, clusterSvc *corev1.Service) error
	UpdateCache(key types.NamespacedName, val *corev1.Service)
	GetCapacityPerLB() int
	UpdateService(svc, lb *corev1.Service) (portUpdated, externalIPUpdated bool)
}
```

## Backup: CRD Practices

1. Use CRD built-in features
    a. validation
    b. shortNames
    c. additionalPrinterColumns
    d. controllerRef
    e. finalizers
2. CRD controller
    a. kubebuilder
    b. internal cache
    c. reconcile upon IndexKey (namespace/name) of a changed object

# Changes in User's View

**Before/Input:** **N** service yamls (with type LoadBalancer, and src/dst port info)

**Before/Output:** **N** publicly accessible {ip/hostname, port} pairs

**After/Input:** **N** custom resource yamls (w/ or w/o src port info)

**After/Output:** **1** publicly accessible {ip/hostname, (random) port} pair

(N is configurable)

## Future Considerations

1. Get feedback (sig-cloudprovider, sig-network, interested users)
2. Support on more cloudproviders or even baremetal (e.g. metal-lb)
3. More testings and CI