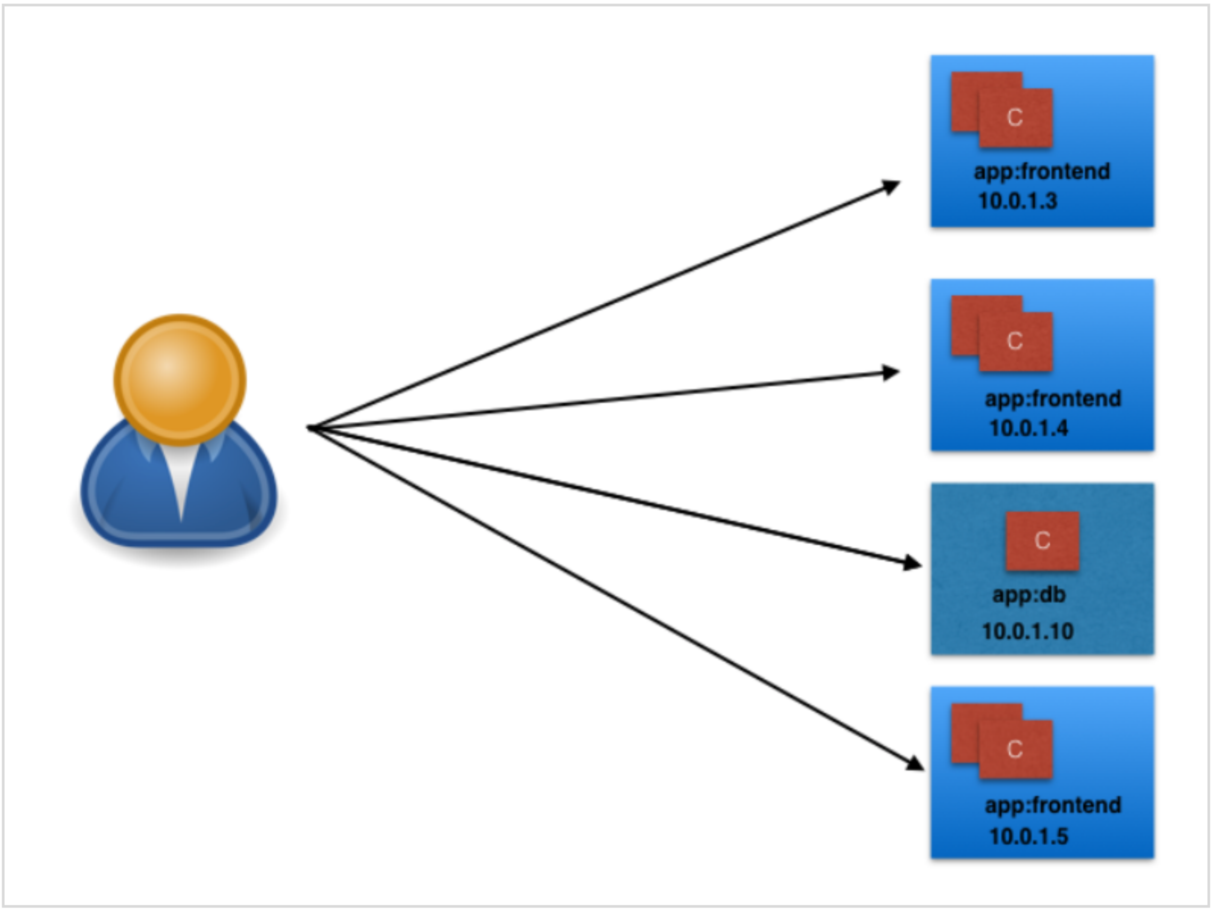
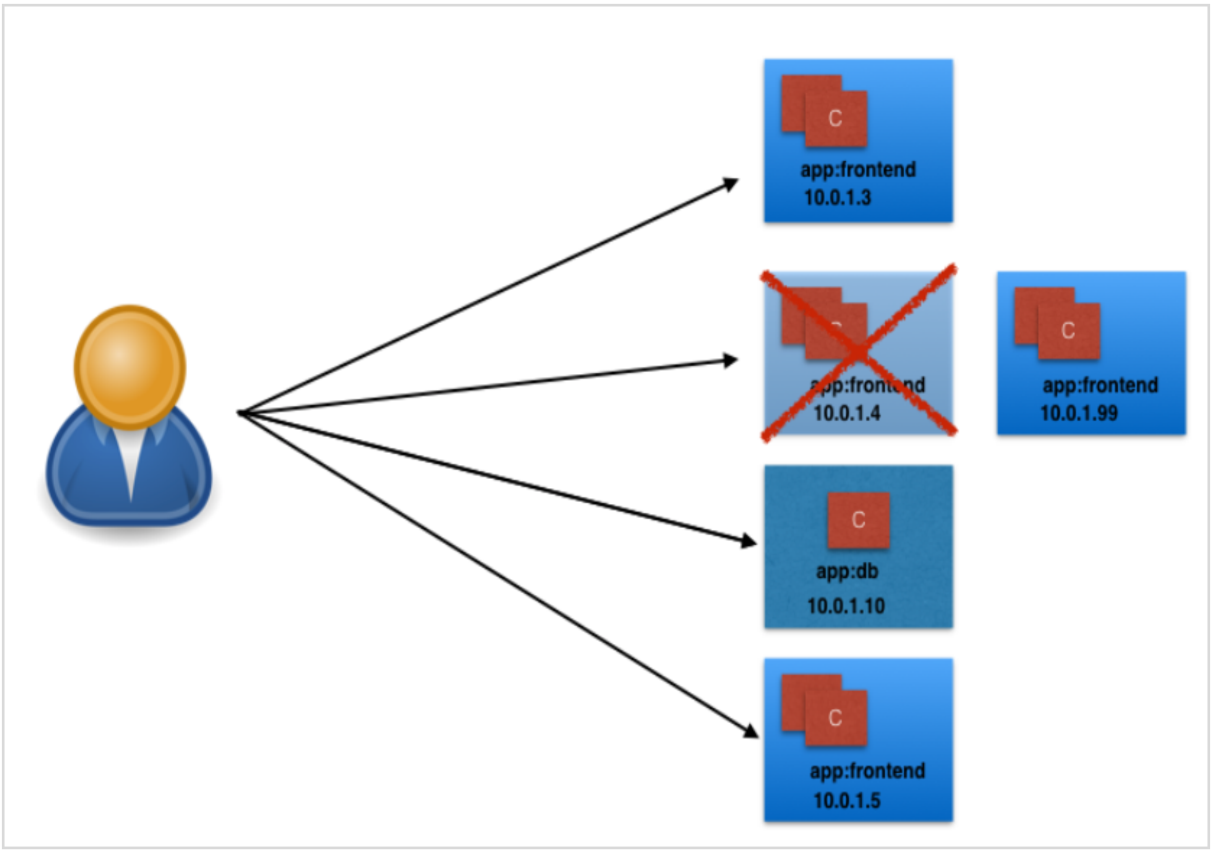
Chapter 10 Services

Connecting Users to Pods



User is connected to a Pod via its IP address

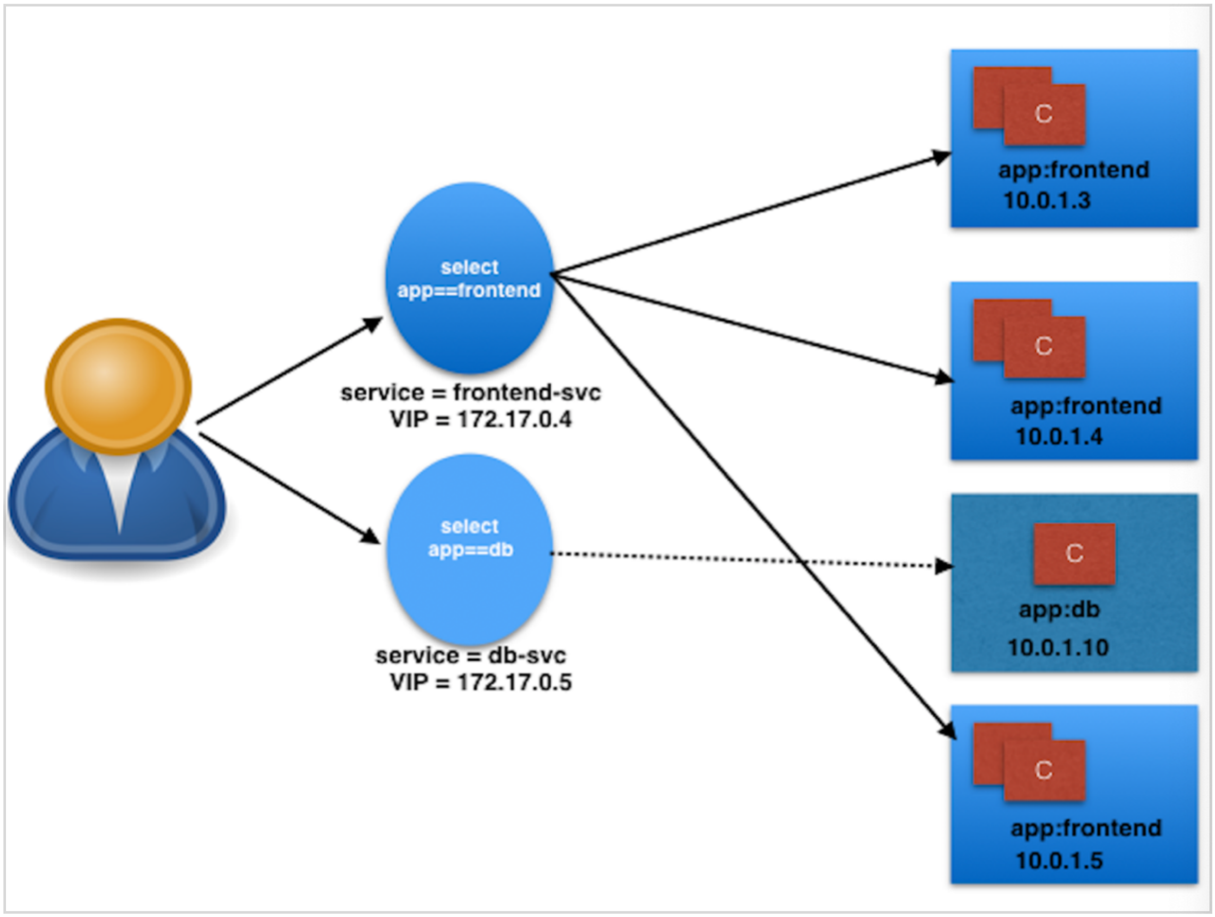


VM terminates, new one gets spun up using different IP address, user cannot connect

K8s provides a higher-level abstraction `Service`

- logically groups Pods and defines a policy to access them via `Labels` and `Selectors`

Services



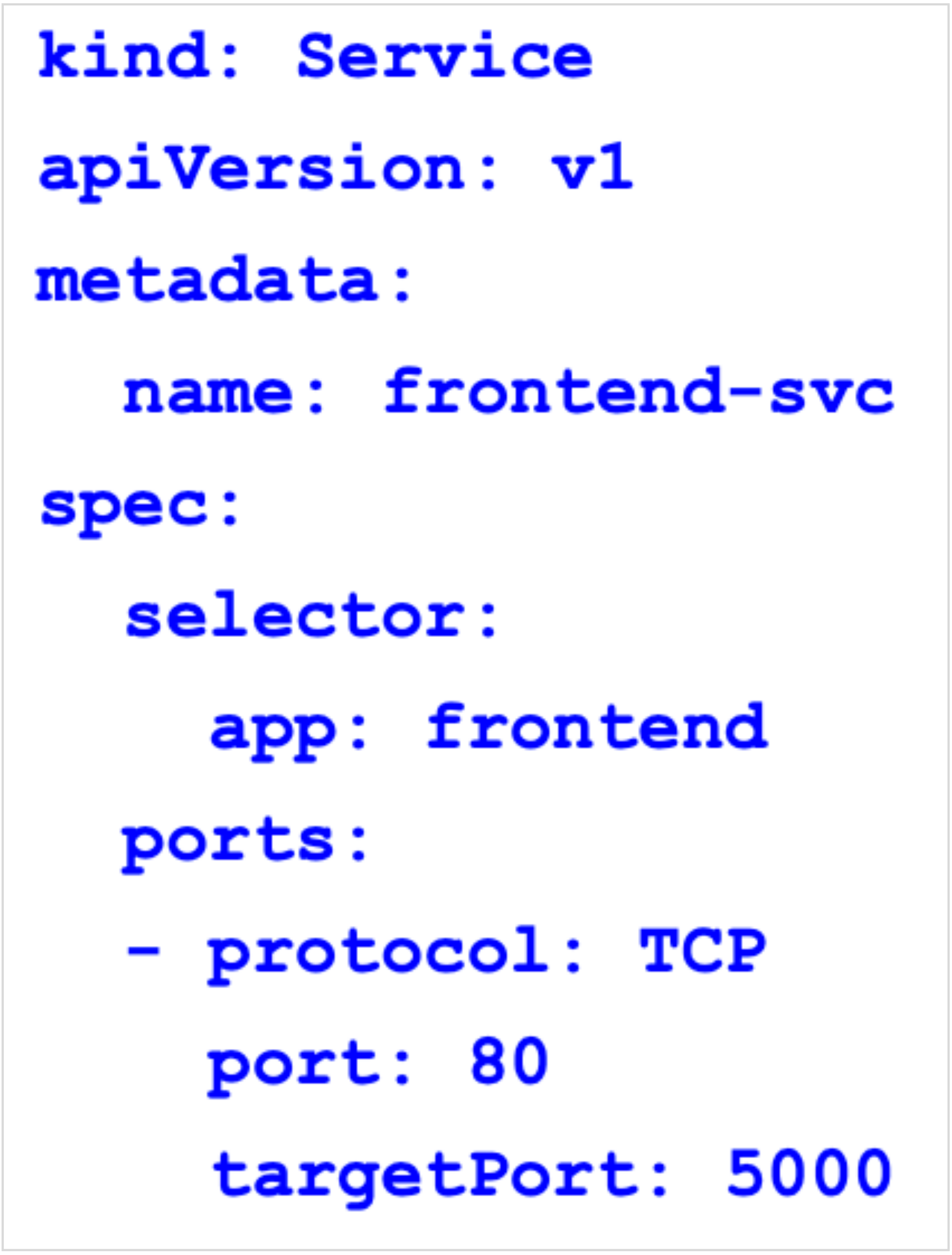
Using `Selectors`

- `app==frontend` : 3 pods
- `app==db` : 1 pod

`Service` is used to assign a name to the logical grouping

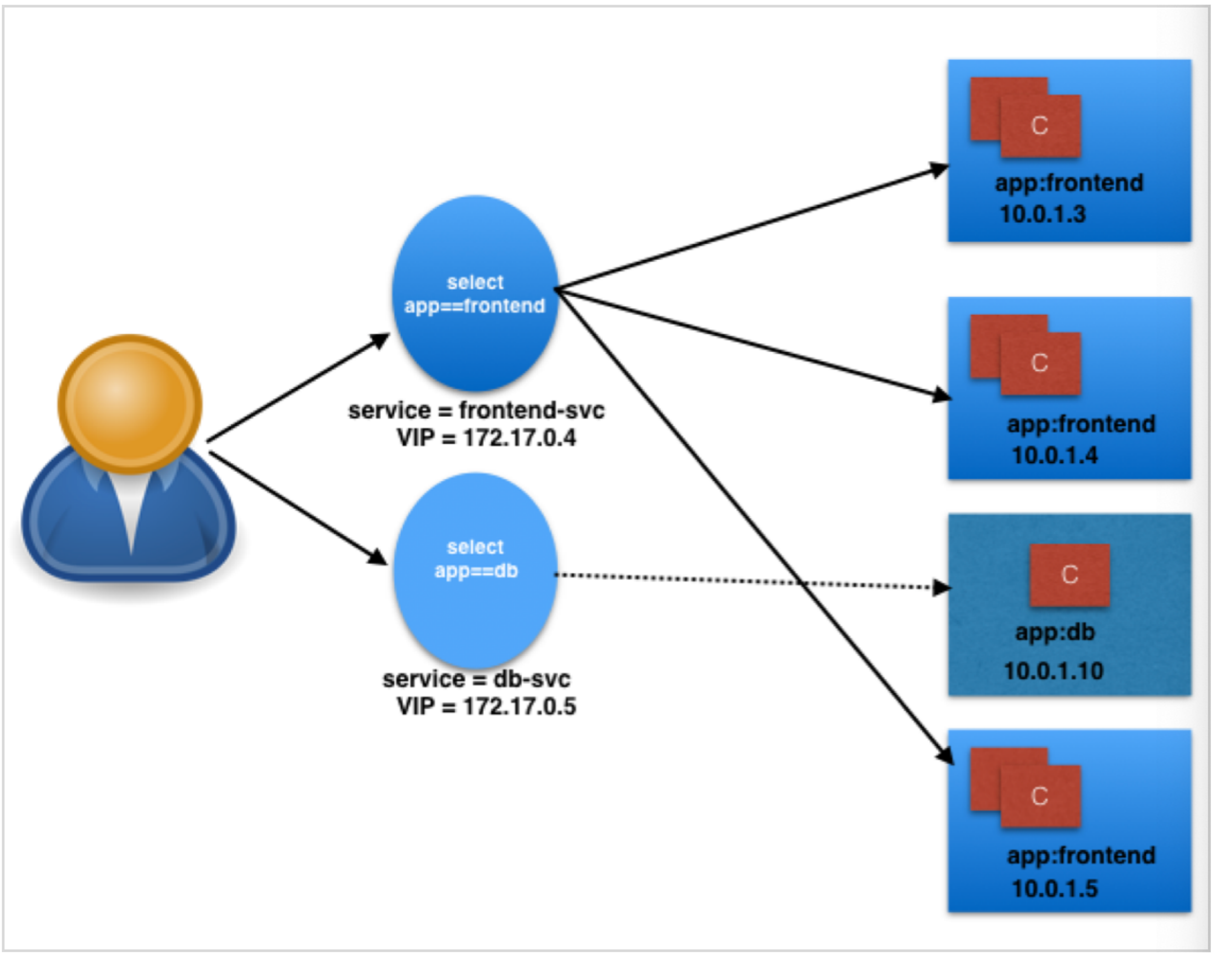
- Service `frontend-svc` names the Selector `app==frontend`

`app` is the Label key
`frontend` and `db` are Label values



```yaml
kind: Service
apiVersion: v1
metadata:
  name: frontend-svc
spec:
  selector:
    app: frontend
  ports:
  - protocol: TCP
    port: 80
    targetPort: 5000
```

`frontend-svc` Service for all Pods with Label key `app` and value `frontend`



Each Service gets a ClusterIP (routable only inside the cluster)

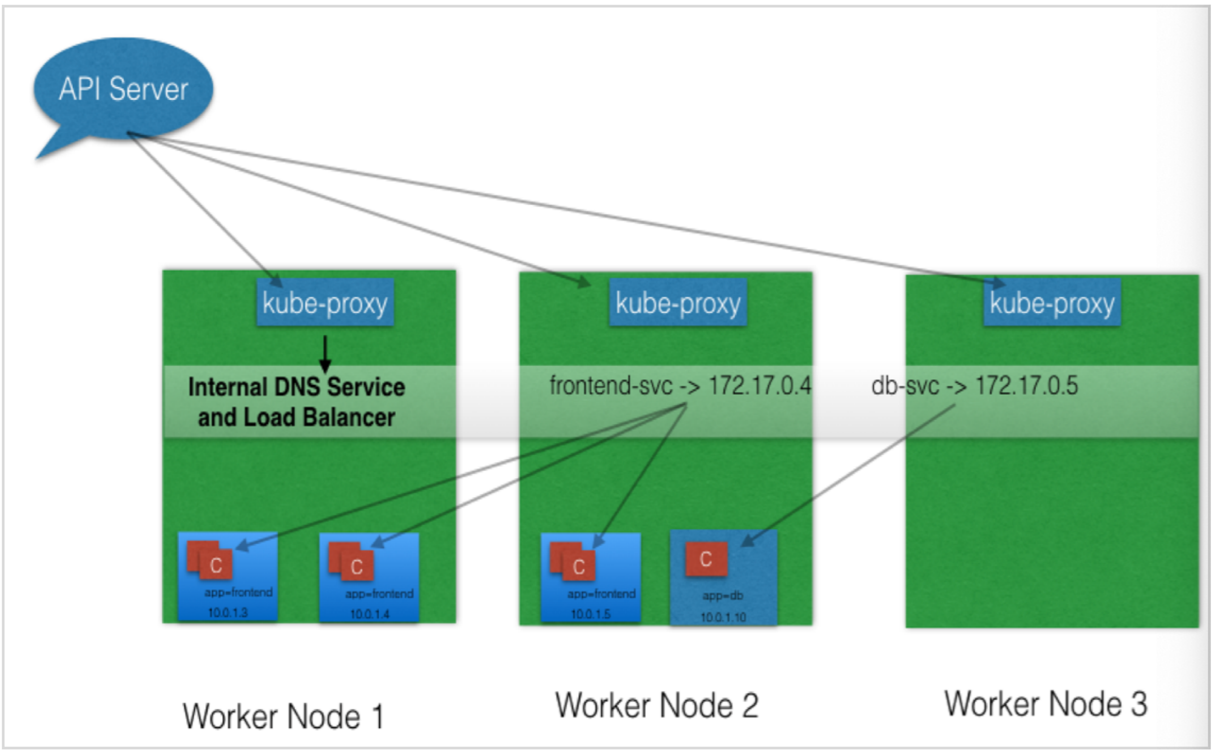
Service receives requests on port 80
Pods receive requests using targetPort 5000 by default

Service endpoint: logical set of a Pod's IP address w/ targetPort

Endpoints are created and managed automatically by the Service, not the K8s cluster admin.

Kube-Proxy

API watcher on every worker node
Watches for addition/removal of Services and endpoints



kube-proxy configures iptables rules to capture the traffic for its ClusterIP and forwards it to one of the Service's endpoints

Receive external traffic, route it internally inside the cluster based on the iptables rules

When Service is removed, kube-proxy removes the corresponding iptables rules on all nodes as well

Service Discovery

When a Pod starts on a worker node, the kubelet daemon sets env vars in the Pod for all active Services

```
REDIS_MASTER_SERVICE_HOST=172.17.0.6
REDIS_MASTER_SERVICE_PORT=6379
REDIS_MASTER_PORT=tcp://172.17.0.6:6379
REDIS_MASTER_PORT_6379_TCP=tcp://172.17.0.6:6379
REDIS_MASTER_PORT_6379_TCP_PROTO=tcp
REDIS_MASTER_PORT_6379_TCP_PORT=6379
REDIS_MASTER_PORT_6379_TCP_ADDR=172.17.0.6
```

env vars for active Service `redis-master` that exposes `port 6379` with ClusterIP

`172.17.0.6`
Note: if Service is created after Pods, they won't have env vars.

- DNS
  - add-on that creates a DNS record for each Service
  - `my-svc.my-namespace.svc.cluster.local`
  - Services within that namespace can find each other by the name all Pods in namespace `my-ns` can look up `redis-master`
  - Pods in other NS can look up `redis-master.my-ns`

ServiceType

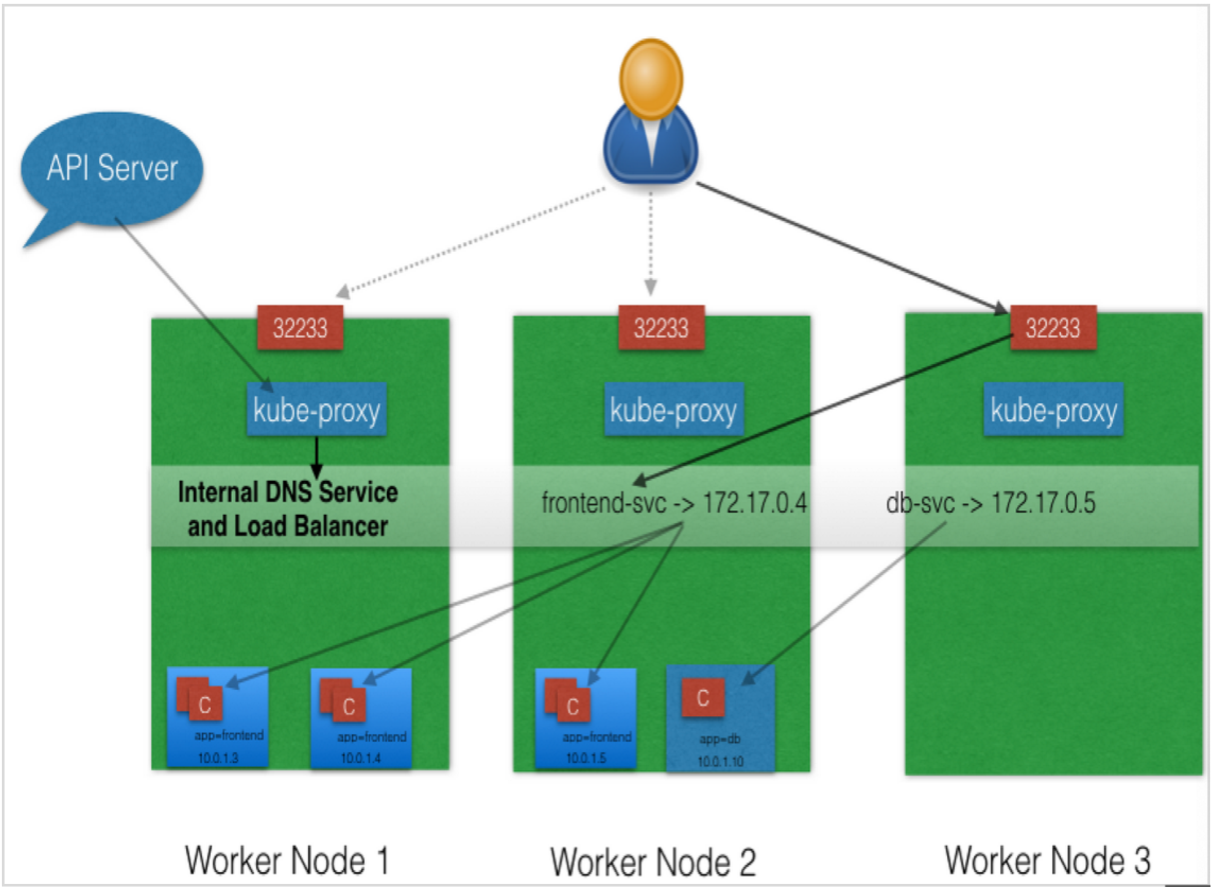
Defines accessibility scope

- cluster
- cluster + external
- maps to an entity (inside or outside of cluster)

ClusterIP (default)

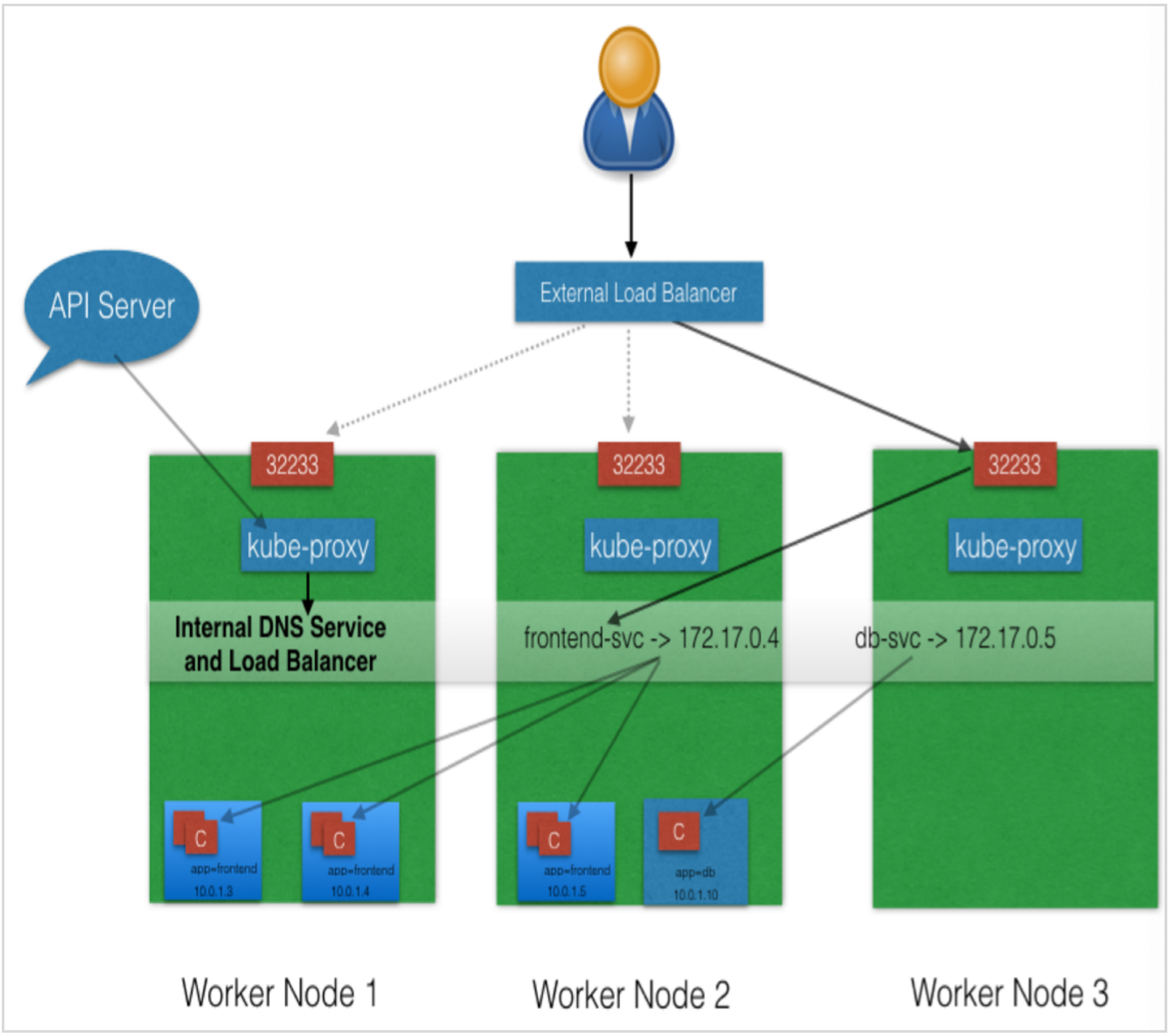
- Service receives a Virtual IP address for communication
- only accessible within the cluster

NodePort



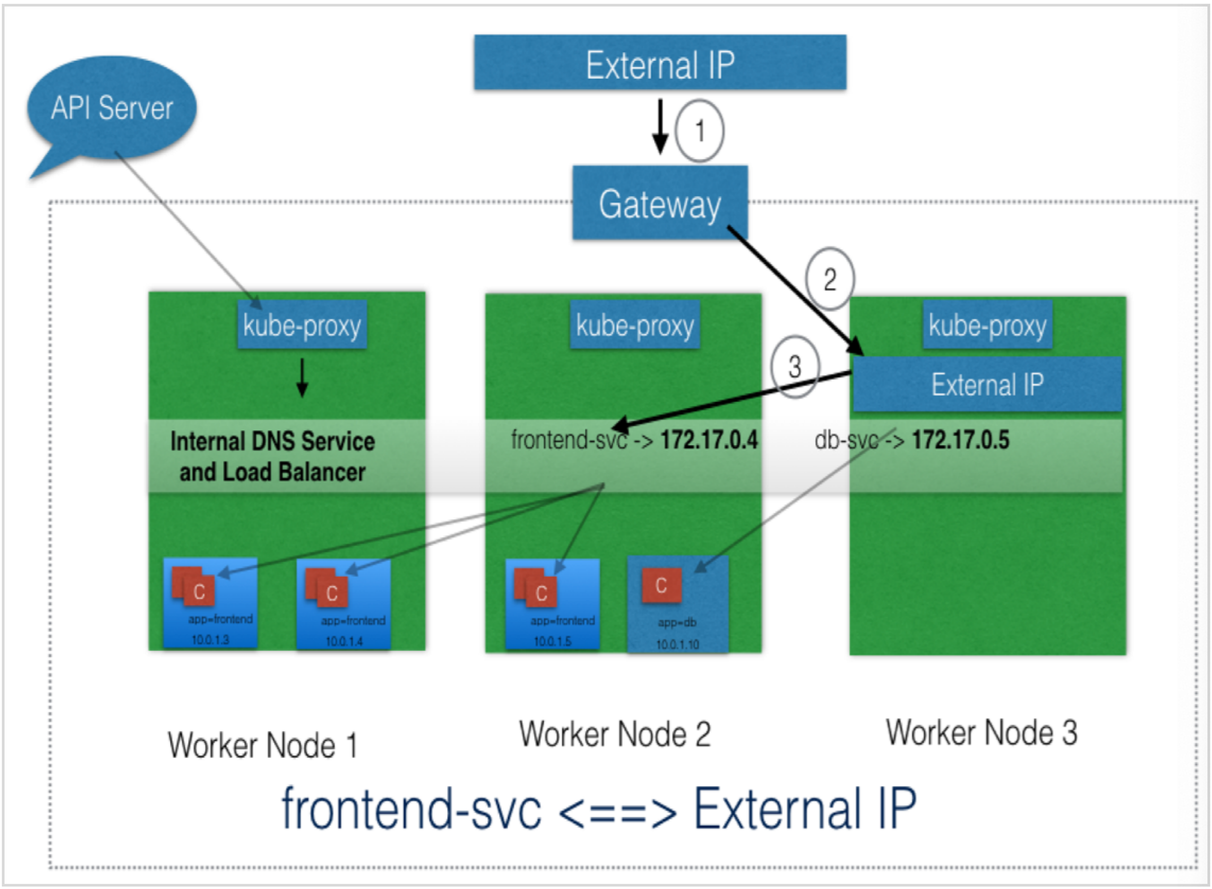
- In addition to a ClusterIP
- port dynamically picked from default range of 30000-23767
- mapped to the Service
- NodePort `32233` for Service `frontend-svc`
  - All traffic would be redirected to assigned ClusterIP `172.17.0.4`
- Easy to access externally

LoadBalancer



- NodePort & ClusterIP are automatically created
    - external LB will route to them
- Service is exposed at a static port (NodePort) on each worker node
- Only work if the infrastructure supports auto-creation of LB for K8s

ExternalIP



- A Service is mapped to an ExternalIP address if it can route to 1+ worker nodes
- Ingressed traffic via ExternalIP gets routed to one of the Service endpoints
- Not managed by K8s

ExternalName

- no Selectors and doesn't define any endpoints
- returns a `CNAME` record of an externally configured Service
- allows externally configured Services (database, etc.) available to apps inside the cluster
- apps can access the service by the name in the same namespace