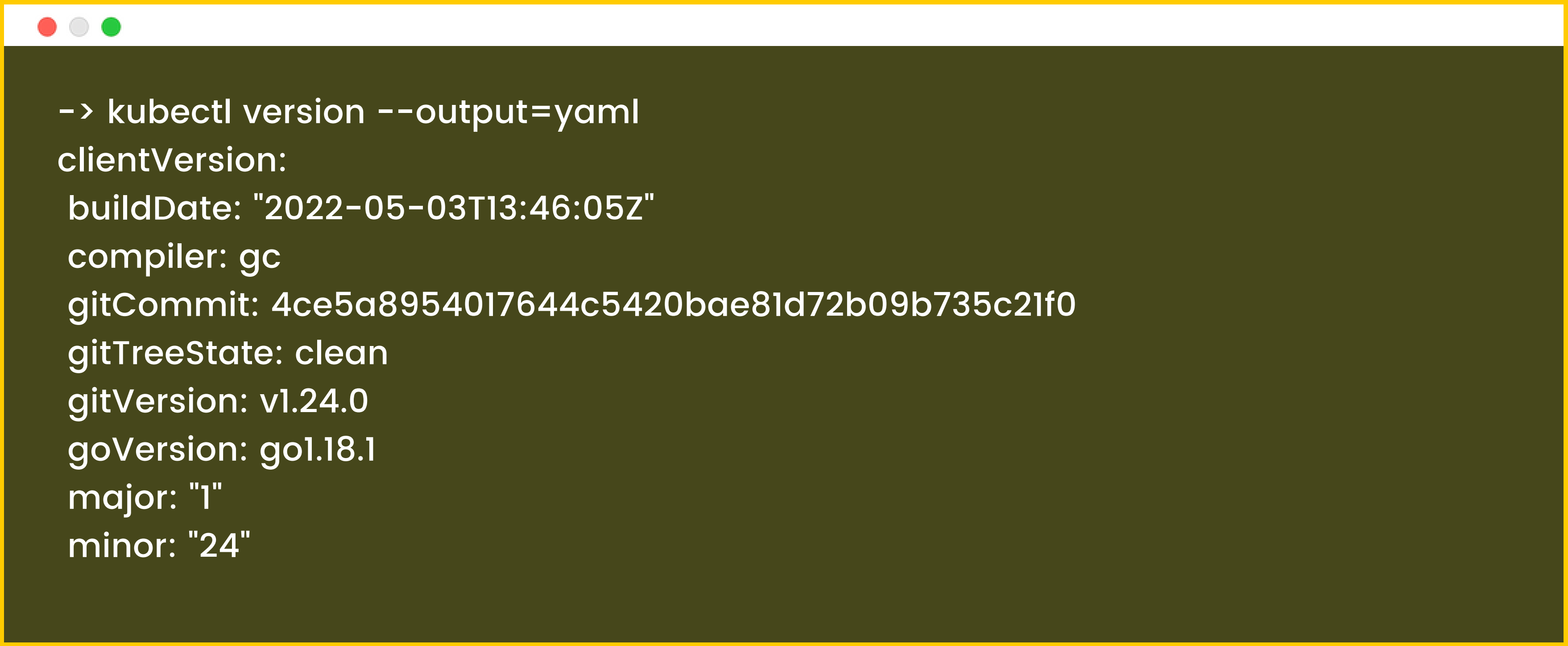
Kubernetes Best Practices

Created by Akshay Suresh

@akshayvsuresh

Upgrade your Kubernetes version

You may obtain new features, bug fixes, and security updates by running Kubernetes in its most recent version. Older Kubernetes versions might leave your system vulnerable, and you could have a difficult time getting support for them. Upgrade to the most recent stable release, unless you have a good reason to keep an older version.

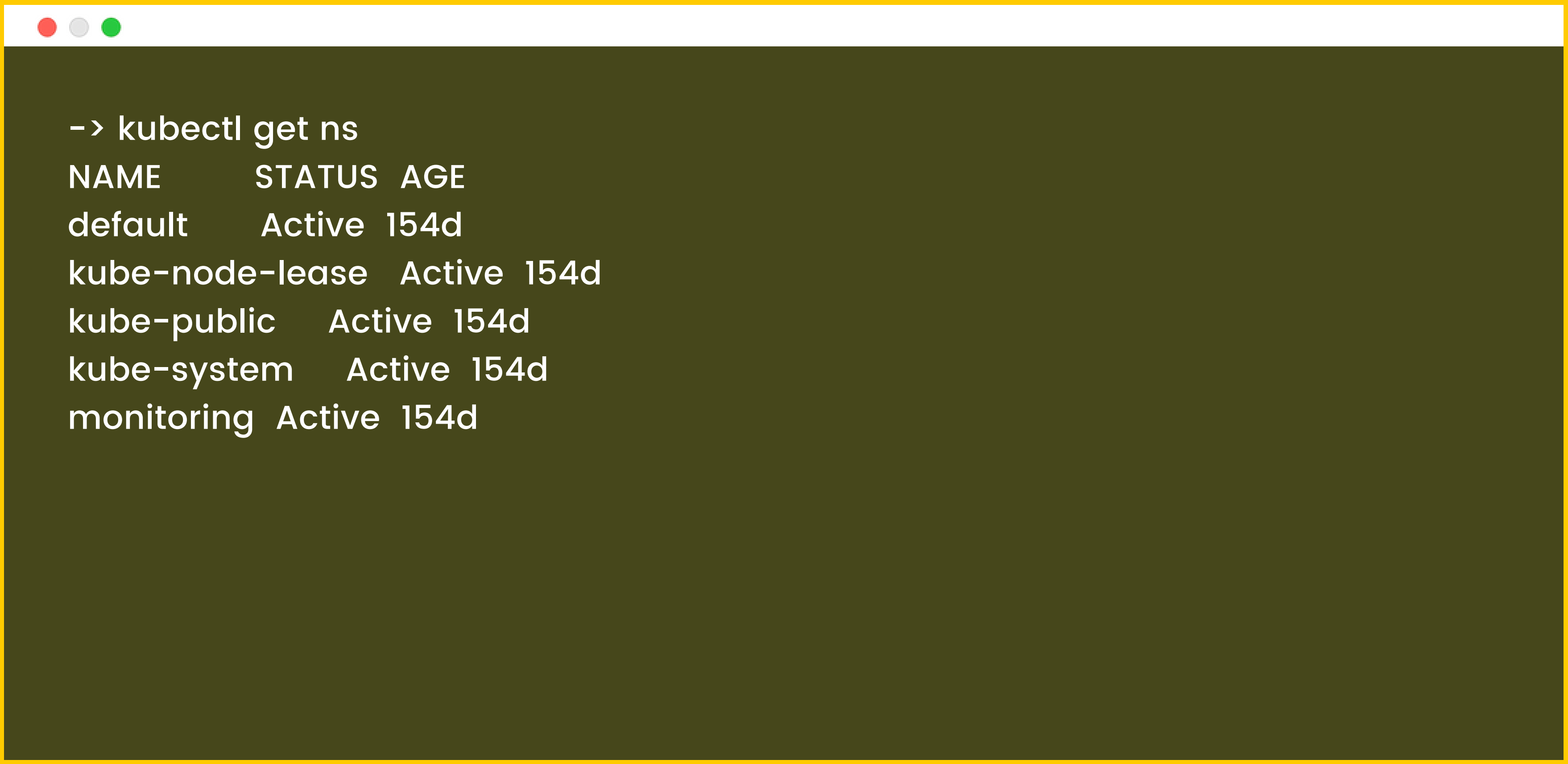


```
-> kubectl version --output=yaml
clientVersion:
 buildDate: "2022-05-03T13:46:05Z"
 compiler: gc
 gitCommit: 4ce5a8954017644c5420bae81d72b09b735c21f0
 gitTreeState: clean
 gitVersion: v1.24.0
 goVersion: go1.18.1
 major: "1"
 minor: "24"
```

@akshayvsuresh

Using Namespaces

Using namespaces, you may logically separate different domains or functions inside your cluster. You may provide policies and access rules for each namespace after it has been configured. Namespaces make container management easier and lower the possibility of widespread failure.

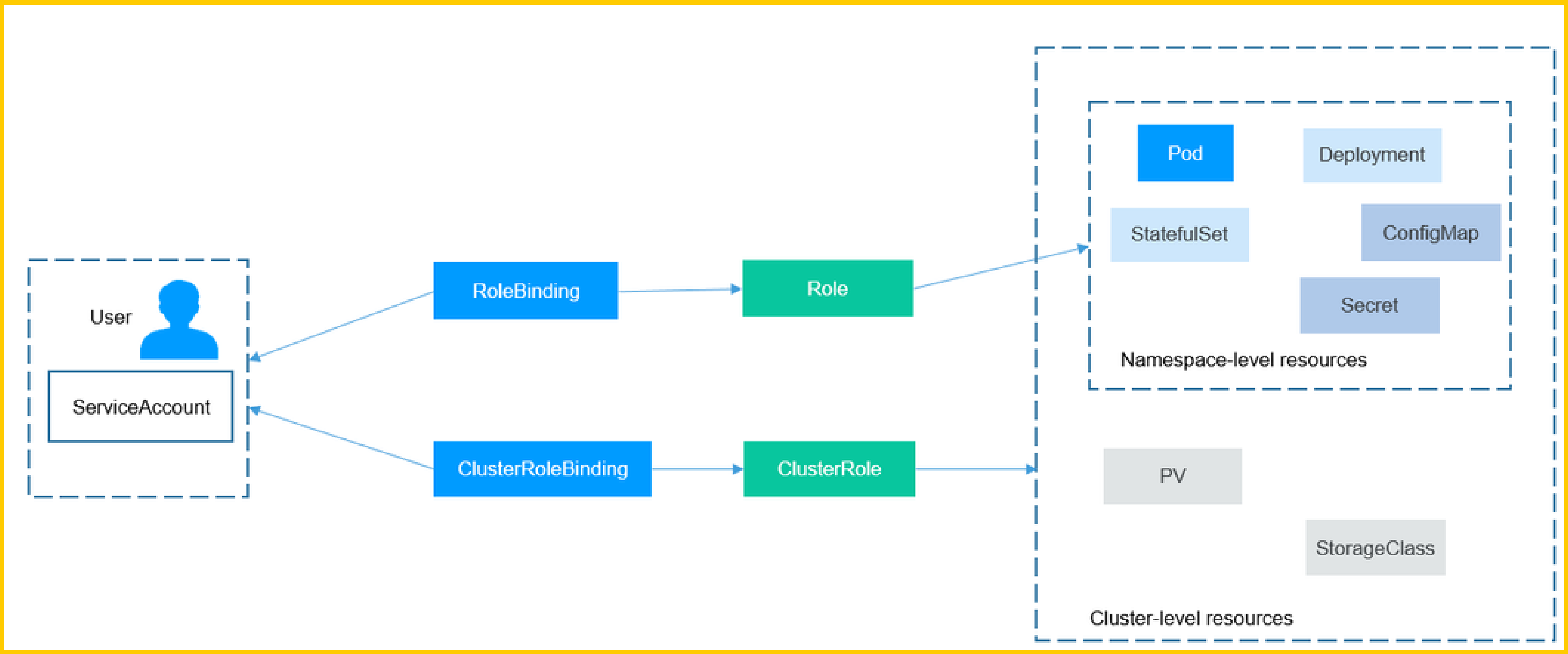


```
-> kubectl get ns
NAME        STATUS  AGE
default      Active  154d
kube-node-lease  Active  154d
kube-public    Active  154d
kube-system    Active  154d
monitoring  Active  154d
```

@akshayvsuresh

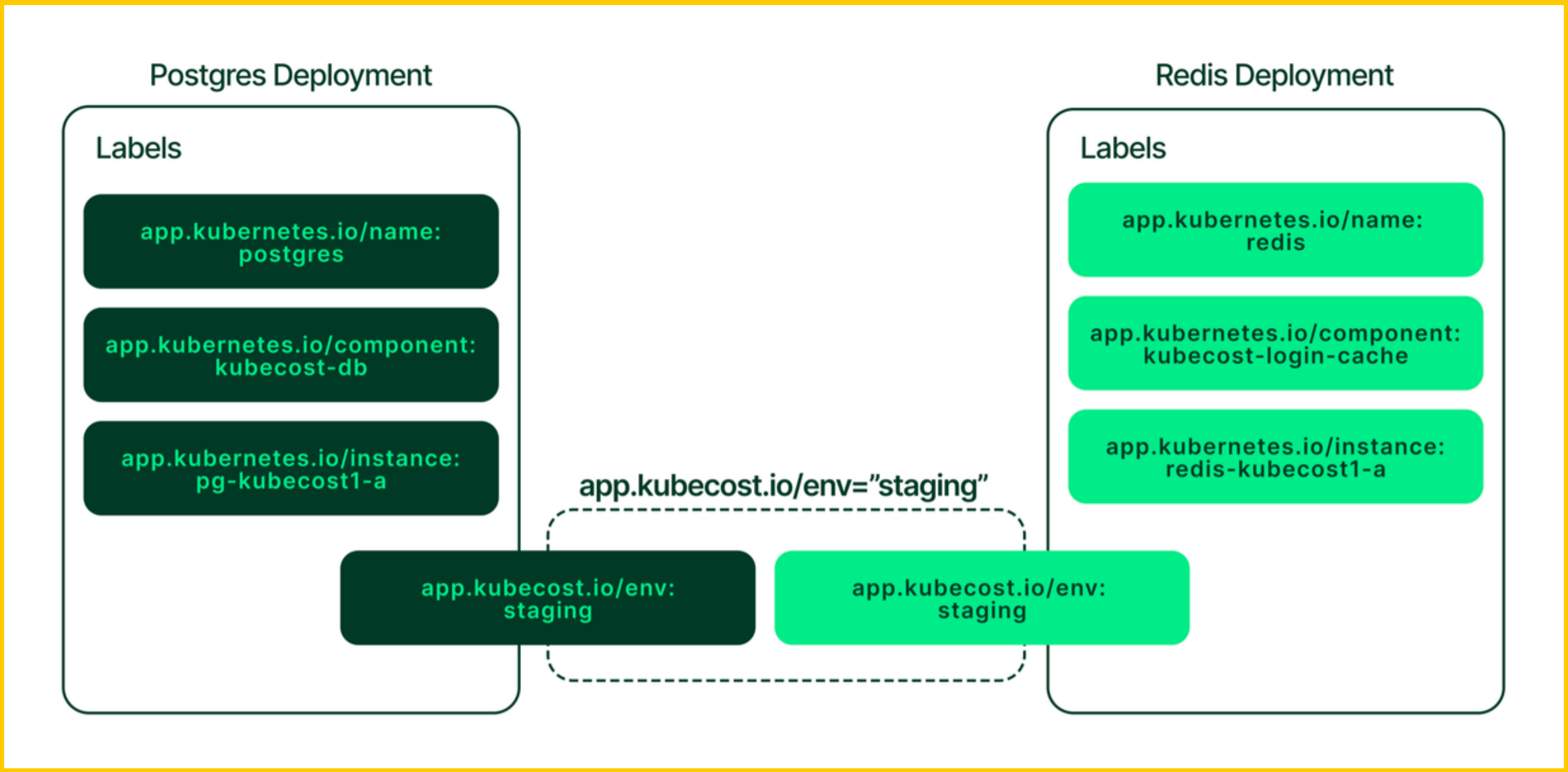
Using RBAC

RBAC makes it simpler to manage the permissions and access granted to your users or service accounts, reducing the possibility of mistakes. Kubernetes allows you to set up roles within namespaces or across the cluster and define their permissions. Accounts can then be linked to those roles, ensuring they only have the permissions granted by those roles.



Using Labels

Labels in Kubernetes allow you to attach key-value pairs to K8s objects. As an orchestration platform, Kubernetes lets you define objects to maintain an abstraction layer around your individual clusters and their states. As your infrastructure grows, you'll end up with a growing number of objects as well. Labels make it easier to manage these objects. At a base level, you can use them to define and track metadata. For instance, you could use a label to track who a pod's owner is.



Postgres Deployment
Labels
app.kubernetes.io/name: postgres
app.kubernetes.io/component: kubecost-db
app.kubernetes.io/instance: pg-kubecost1-a
Redis Deployment
Labels
app.kubernetes.io/name: redis
app.kubernetes.io/component: kubecost-login-cache
app.kubernetes.io/instance: redis-kubecost1-a
app.kubecost.io/env="staging"
app.kubecost.io/env: staging
app.kubecost.io/env: staging

@akshayvsuresh

Using Readiness & Liveness

Readiness probe: Ensures a given pod is up-and-running before allowing the load to get directed to that pod. If the pod is not ready, the requests are taken away from your service until the probe verifies the pod is up.
Liveness probe: Verifies if the application is still running or not. This probe tries to ping the pod for a response from it and then check its health. If there is no response, then the application is not running on the pod. The liveness probe launches a new pod and starts the application on it if the check fails.

```
apiVersion: v1
kind: Pod
metadata:
 name: test
spec:
 containers:
- image: nginx
name: probe-test
livenessProbe:
       httpGet:
         path: /ping
         port: 8080
```

```
apiVersion: v1
kind: Pod
metadata:
 name: test
spec:
 containers:
- image: nginx
name: probe-test
readinessProbe:
       httpGet:
         path: /ping
         port: 8080
```

@akshayvsuresh

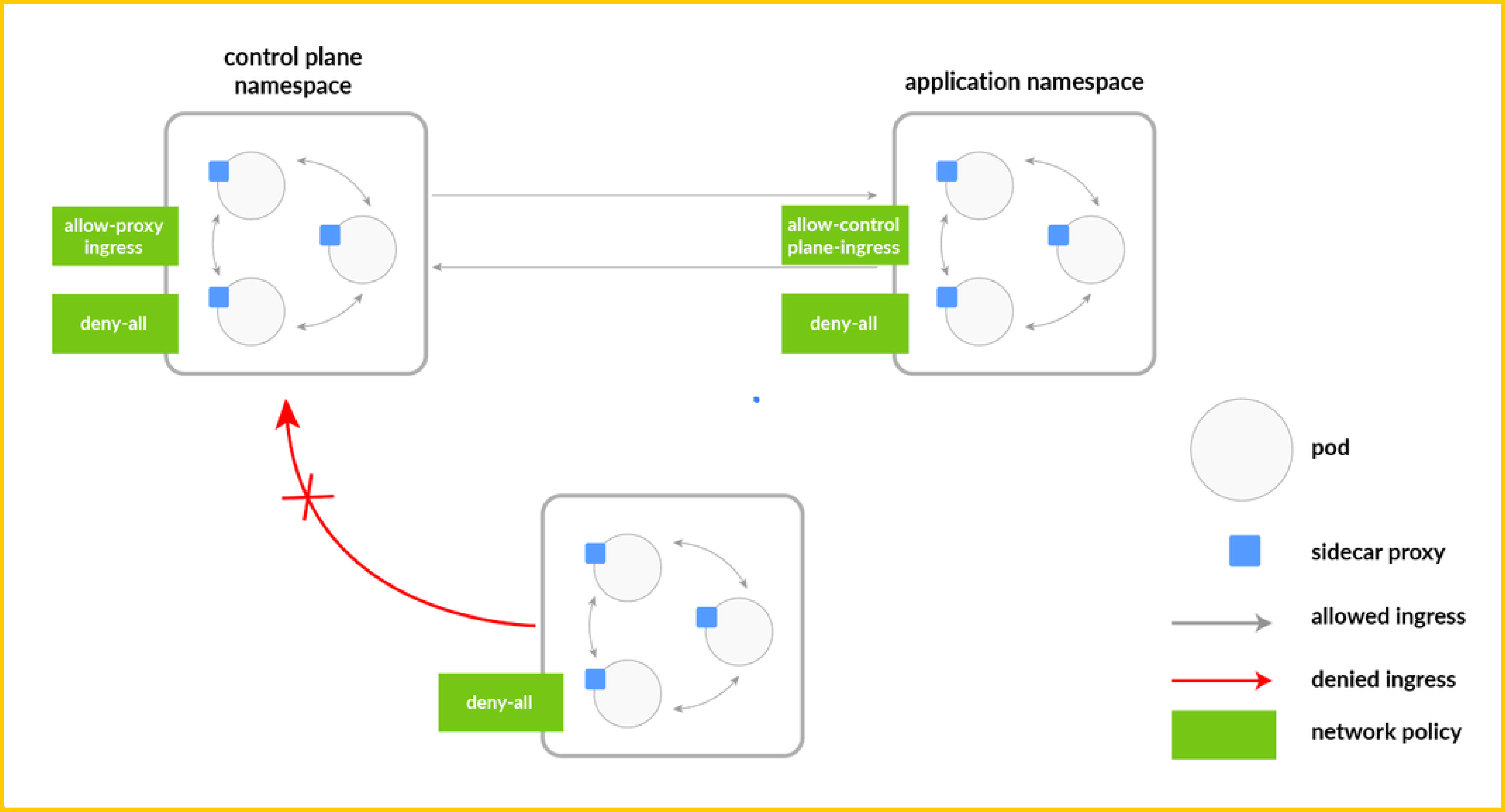
Using Requests and Limits

Resource requests: specify the minimum amount of resources a container can use
Resource limits: specify the maximum amount of resources a container can use.
For both requests and limits, It’s typical to define CPU in millicores and memory is in megabytes or mebibytes.
Containers in a pod do not run if the request of resources made is higher than the limit you set.

```
containers:
- name: devContainer2
   image: ubuntu
   resources:
      requests:
         memory: "128Mi"
         cpu: "400m"
      limits:
         memory: "256Mi"
         cpu: "800m"
```

Using Network Policies

Though it may seem safe enough to allow your containers to communicate with any other service behind your firewall, this can pose a risk if malicious actors gain entry into your system.
By default, your containers should deny any traffic unless it is from specifically allowed sources.



Akshay Suresh

DevOps Engineer @ Rubrik

Senior DevOps Engineer @ Hashedin by Deloitte

YCombinator SUS'21

@akshayvsuresh

akshayvsuresh27@gmail.com

Thank You!

@akshayvsuresh