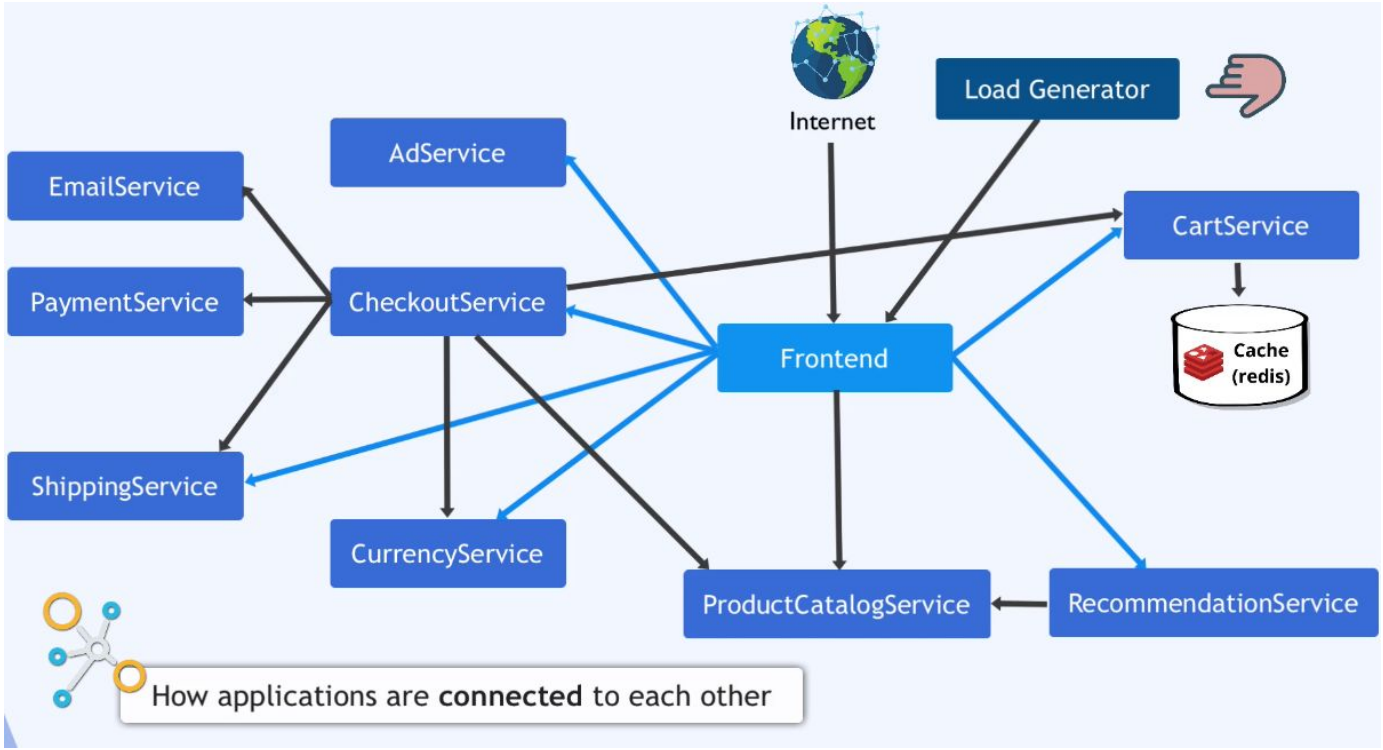
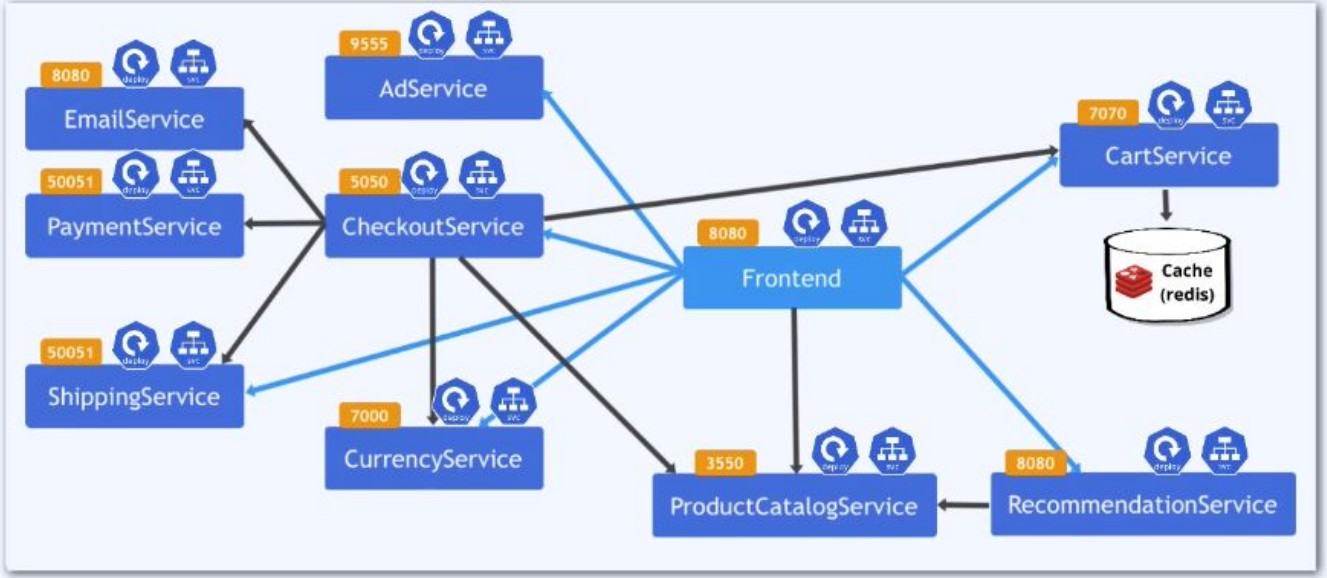
Deploy microservice app into k8s cluster



how applications are connected to each other
image names for each microservice
environment variables
port
deploy ms into single spaces

Create **Deployment** Config for **each Microservice**

Create **Service** Config for **each Microservice**



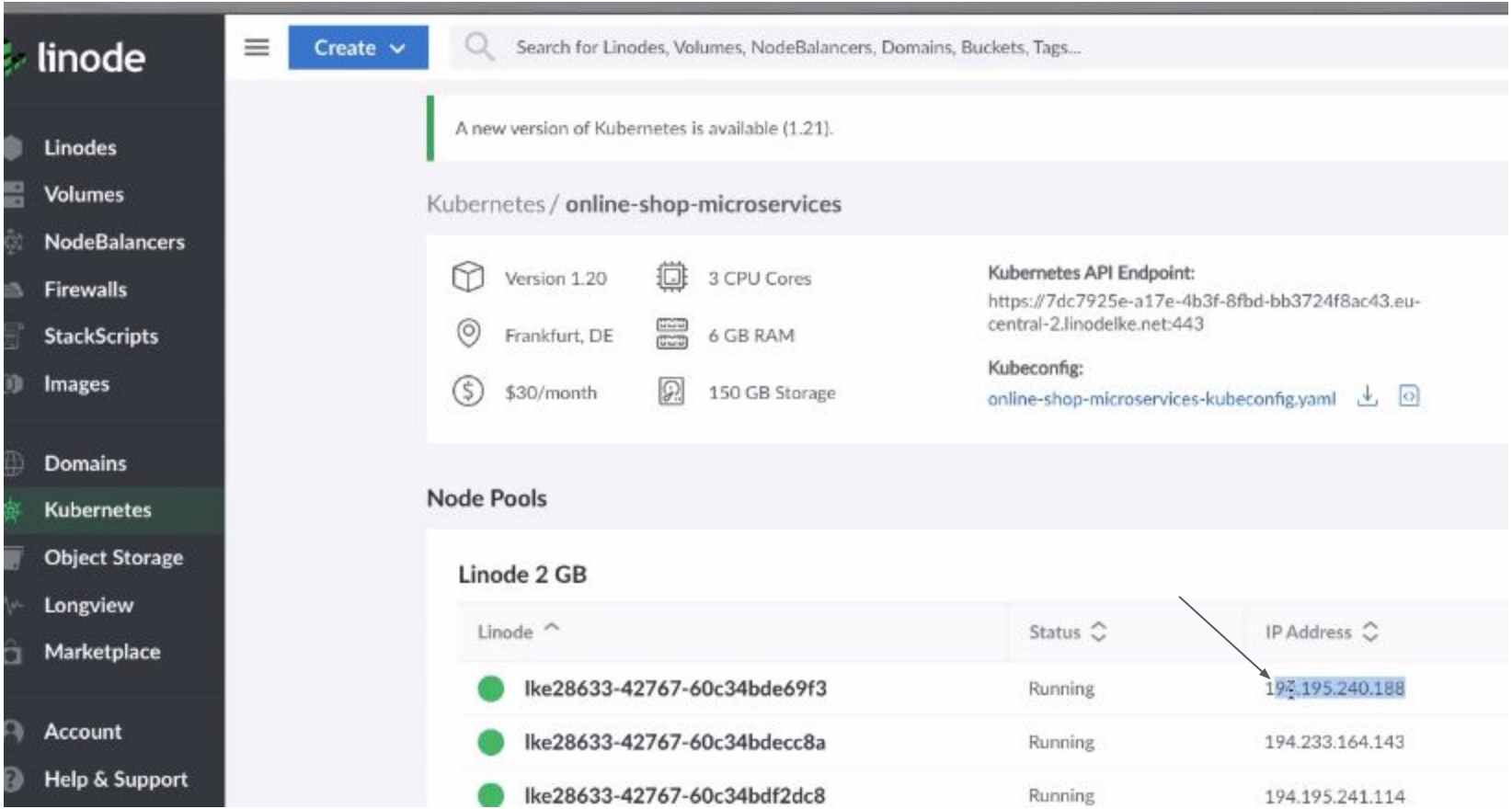
1. create a config file
   a. skeleten
2. speed up k8s cluster in linode
   a. download the kubeconfig file
3. set permission of kubeconfig file

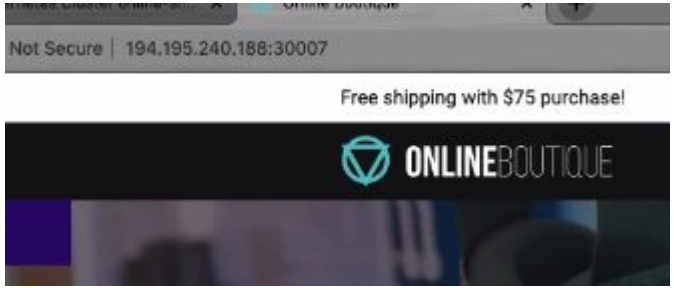
```
[\W]$ chmod 400 ~/Downloads/online-shop-kubeconfig.yaml
[\W]$ ls -l ~/Downloads/online-shop-kubeconfig.yaml
-r--------@ 1 nanajanashia  staff  2764 Jun 16 09:54 /Users
config.yaml
```

4. variable

```
[\W]$ export KUBECONFIG=~/Downloads/online-shop-microservices-kubeconfig.yaml
```

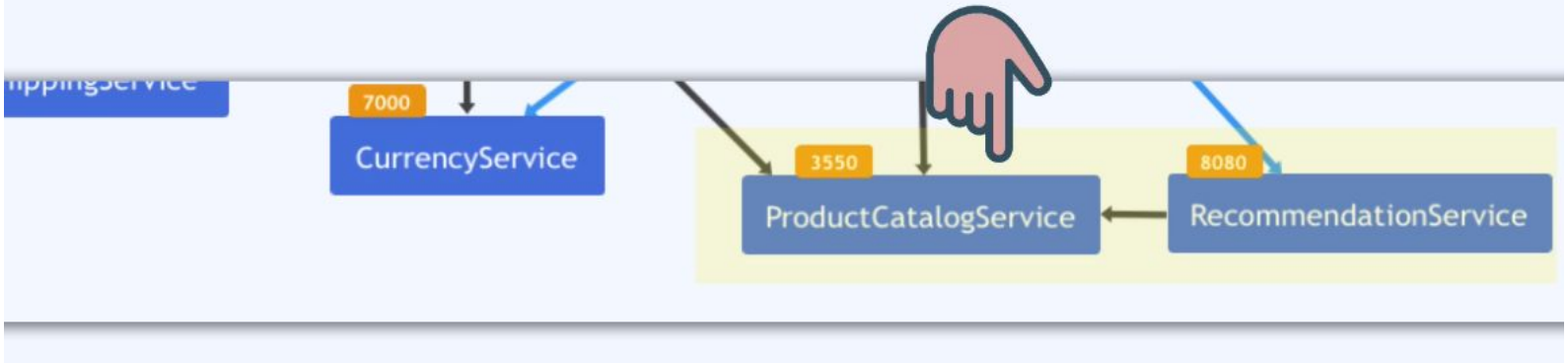
5. kubectl get node check if the cluster right configured
6. kubectl create ns microservices
7. kubeclt apply -f config.yaml -n microservices
8. kubectl get pod -n microservices
9. ubectl get svc -n microservices


linode
Create
Search for Linodes, Volumes, NodeBalancers, Domains, Buckets, Tags...
Linodes
Volumes
NodeBalancers
Firewalls
StackScripts
Images
Domains
Kubernetes
Object Storage
Longview
Marketplace
Account
Help & Support
A new version of Kubernetes is available (1.21).
Kubernetes / online-shop-microservices
Version 1.20
3 CPU Cores
Frankfurt, DE
6 GB RAM
$30/month
150 GB Storage
Kubernetes API Endpoint:
https://7dc7925e-a17e-4b3f-8fbd-bb3724f8ac43.eu-central-2.linodelke.net:443
Kubeconfig:
online-shop-microservices-kubeconfig.yaml
Node Pools
Linode 2 GB
Linode
Status
IP Address
lke28633-42767-60c34bde69f3
Running
194.195.240.188
lke28633-42767-60c34bdecc8a
Running
194.233.164.143
lke28633-42767-60c34bdf2dc8
Running
194.195.241.114


Not Secure | 194.195.240.188:30007
Free shipping with $75 purchase!
ONLINEBOUTIQUE

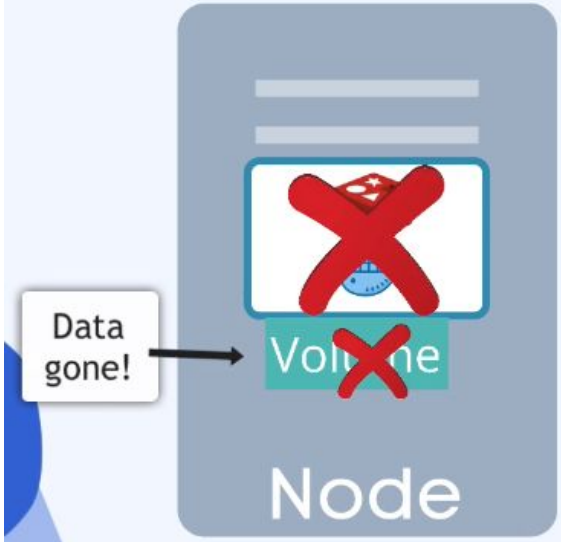
Recommendation Service needs to know the endpoint of ProductCatalogService

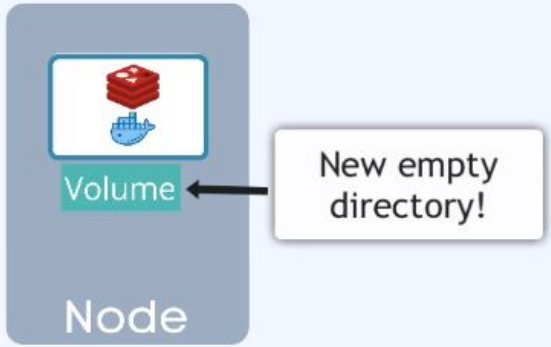
Endpoint = Service Name + Service Port



emptyDir

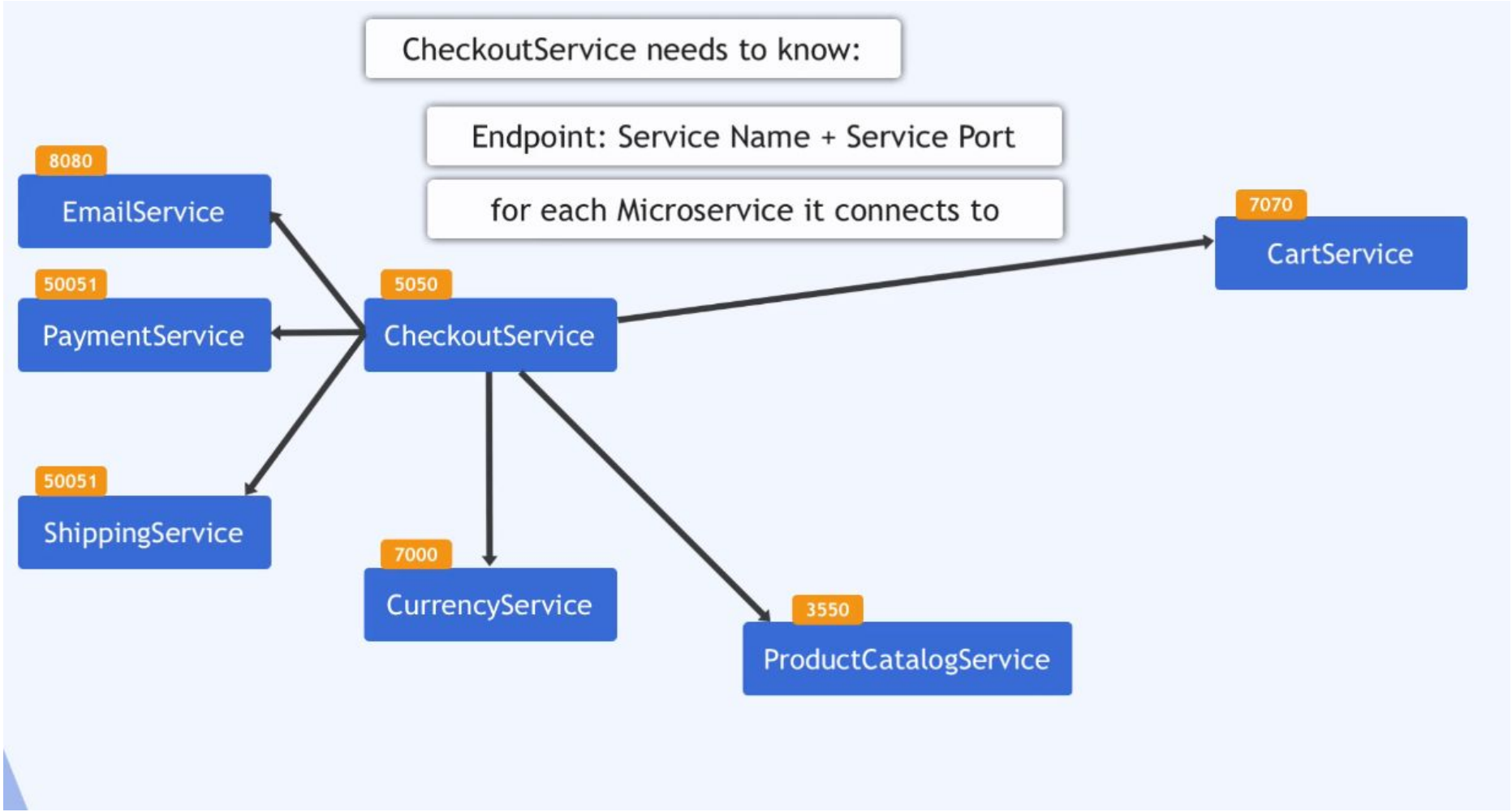
- Is initially empty
- First created when a Pod is assigned to a Node
- **Exists as long as the Pod is running**





emptyDir

- Is initially empty
- First created when a Pod is assigned to a Node
- **Exists as long as the Pod is running**
- Container crashing does NOT remove a Pod from a Node
- Therefore, **data safe across container crashes!**


CheckoutService needs to know:
Endpoint: Service Name + Service Port
for each Microservice it connects to
8080
EmailService
50051
PaymentService
50051
ShippingService
5050
CheckoutService
7070
CartService
7000
CurrencyService
3550
ProductCatalogService

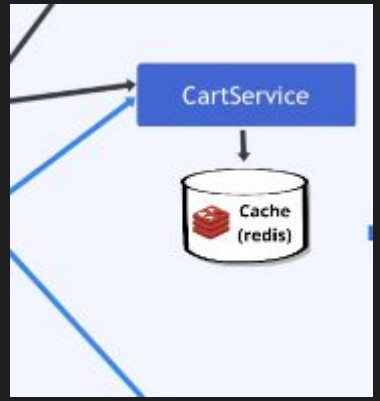
```
88                port = os.Getenv("PORT")
89        }
90
91        svc := new(checkoutService)
92        mustMapEnv(&svc.shippingSvcAddr, "SHIPPING_SERVICE_ADDR")
93        mustMapEnv(&svc.productCatalogSvcAddr, "PRODUCT_CATALOG_SERVICE_ADDR")
94        mustMapEnv(&svc.cartSvcAddr, "CART_SERVICE_ADDR")
95        mustMapEnv(&svc.currencySvcAddr, "CURRENCY_SERVICE_ADDR")
96        mustMapEnv(&svc.emailSvcAddr, "EMAIL_SERVICE_ADDR")
97        mustMapEnv(&svc.paymentSvcAddr, "PAYMENT_SERVICE_ADDR")
98
```

```
349          ports:
350          - containerPort: 5050
351          env:
352          - name: PORT
353            value: "5050"
354          - name: PRODUCT_CATALOG_SERVICE_ADDR
355  ---
356  apiVersion: v1
357  kind: Service
358  metadata:
359    name: checkoutservice
360  spec:
361    type: ClusterIP
```

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: emailservice
spec:
  selector:
    matchLabels:
      app: emailservice
  template:
    metadata:
      labels:
        app: emailservice
    spec:
      containers:
        - name: service
          image: gcr.io/google-samples/emailservice
          ports:
            - containerPort: 8080
          env:
            - name: PORT
              value: "8080"
```

```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: emailservice
spec:
  type: ClusterIP
  selector:
    app: emailservice
  ports:
    - protocol: TCP
      port: 5000
      targetPort: 8080
```

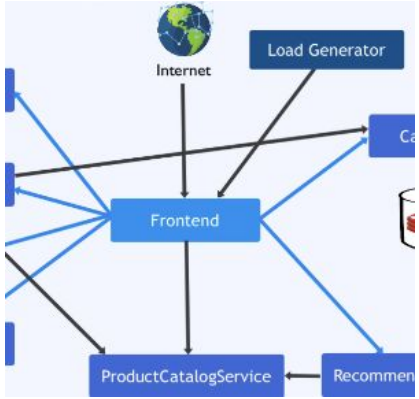
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cartservice
spec:
  selector:
    matchLabels:
      app: cartservice
  template:
    metadata:
      labels:
        app: cartservice
    spec:
      containers:
        - name: service
          image: gcr.io/google-samples/cartservice
          ports:
            - containerPort: 7070
          env:
            - name: PORT
              value: "7070"
            - name: REDIS_ADDR
              value: "redis-cart:6379"
```




```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis-cart
spec:
  selector:
    matchLabels:
      app: redis
  template:
    metadata:
      labels:
        app: redis
    spec:
      containers:
        - name: redis
          image: redis:alpine
          ports:
            - containerPort: 6379
          volumeMounts:
            - mountPath: /data
              name: redis-data
      volumes:
        - name: redis-data
          emptyDir: {}
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
spec:
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
        - name: service
          image: gcr.io/google-samples/frontend
          ports:
            - containerPort: 8080
          env:
            - name: PORT
              value: "8080"
            - name: SHIPPING_SERVICE_ADDR
              value:  "shippingservice:50051"
            - name: PRODUCT_CATALOG_SERVICE_ADDR
              value: "productcatalogservice:3550"
            - name: CART_SERVICE_ADDR
```

```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: frontend
spec:
  type: NodePort
  selector:
    app: frontend
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
      nodePort: 30007
---
```



best practices

1. define image version
2. liveness
3. readness

when the container is not running
kubectl logs checkout
disable functions

config.yaml × ! tmp.yaml

config.yaml > {} spec > {} template > {} spec > [ ] containers > {} 0 > [ ] env

```
66        - name: service
67          image: gcr.io/google-samples/microservices-demo/recommendati
68          ports:
69          - containerPort: 8080
70          env:
71          - name: PORT
72            value: "8080"
73          - name: PRODUCT_CATALOG_SERVICE_ADDR
74            value: "productcatalogservice:3550"
75          - name: DISABLE_TRACING
76            value: "1"
77          - name: DISABLE_PROFILER
78            value: "1"
79          - name: DISABLE_DEBUGGER
80            value: "1"
81          readinessProbe:
```

temp — nanajanashia@Nanas-MacBook-Pro — -zsh — 98×26

...Bootcamp/temp

```
[\W]$ kubectl get pod
NAME                                     READY   STATUS    RESTARTS   AGE
emailservice-5787b7575-jhtrk             1/1     Running   0          36s
productcatalogservice-54d7fc98bb-nj7f2   1/1     Running   0          36s
recommendationservice-68956c9949-2z5md   1/1     Running   0          36s
[\W]$
```


config.yaml > {} spec > {} template > {} spec > [ ] containers > {} 0 > {} readinessProbe > {} exec > [ ] command
23 value: "8080"
2) TCP probes
Kubelect makes probe connection at the node, not in the pod
26 - name: DISABLE_P
27 value: "1"
28 readinessProbe:
29 periodSeconds: 5
30 exec:
31 command: ["/bin/grpc_health_probe", "-addr=:8080"]
32 livenessProbe:
33 periodSeconds: 5
34 exec:
35 command: ["/bin/grpc_health_probe", "-addr=:8080"]
36 ---

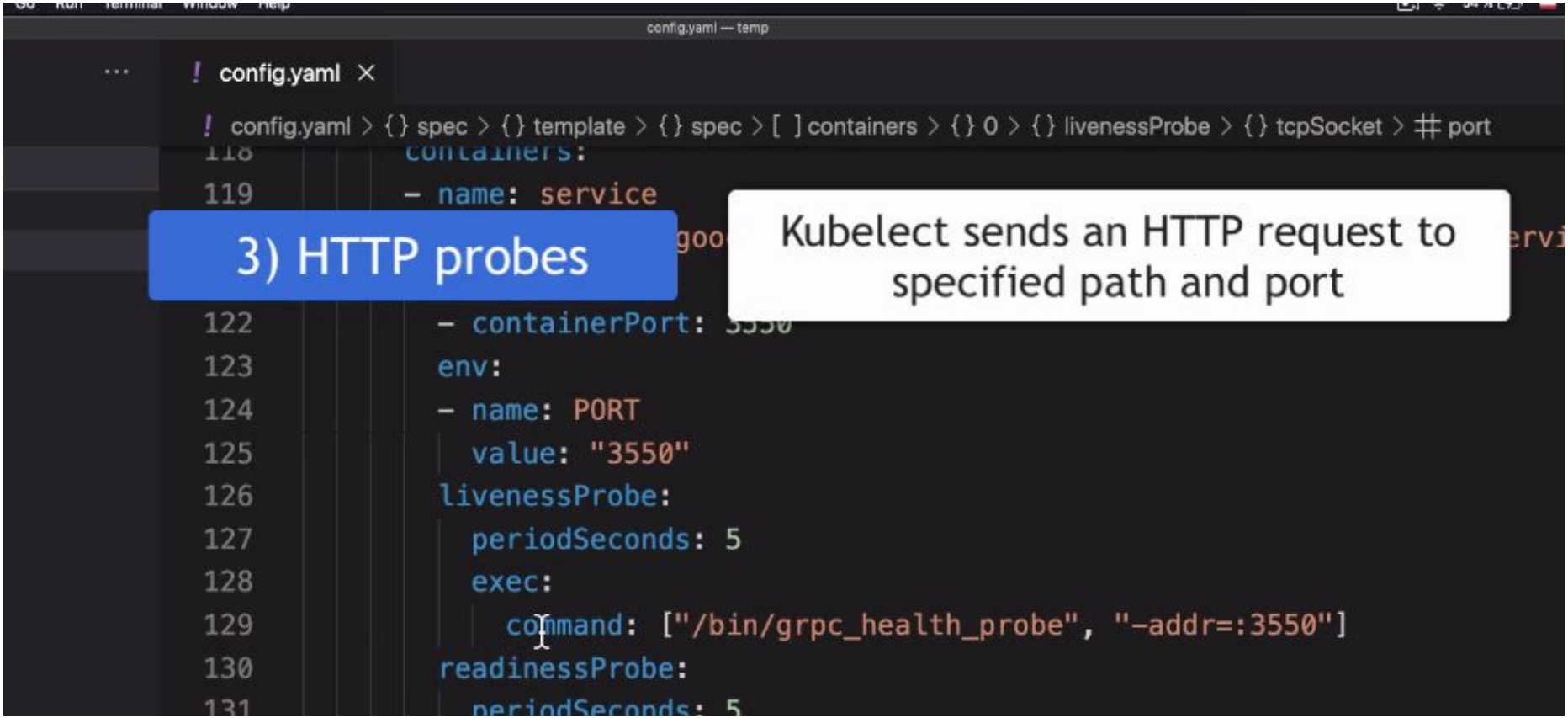
config.yaml

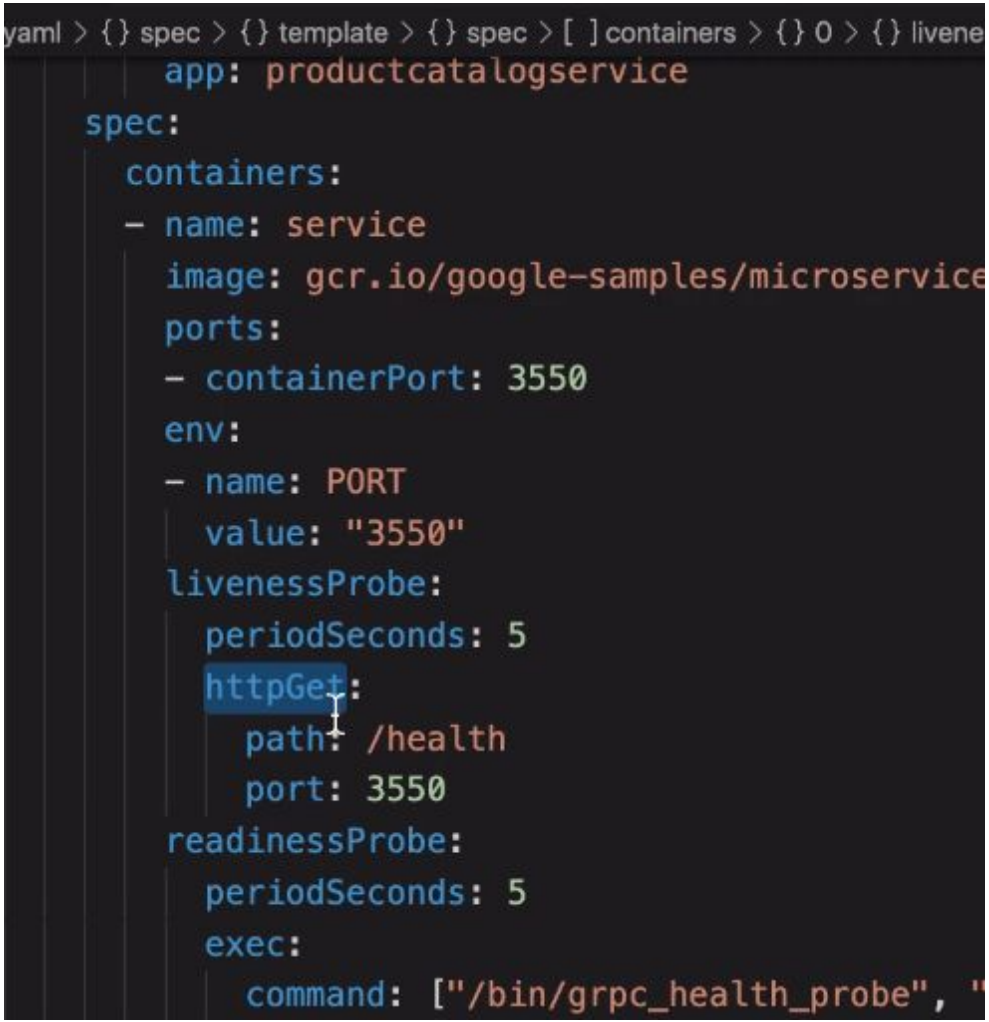
config.yaml > {} spec > {} template > {} spec > [ ] containers > {} 0 > {} livenessProbe > {} tcpSocket

```yaml
  template:
    metadata:
      labels:
        app: redis-cart
    spec:
      containers:
      - name: redis
        image: redis:alpine
        ports:
        - containerPort: 6379
        readinessProbe:
          periodSeconds: 5
          tcpSocket:
            port: 6379
        livenessProbe:
          periodSeconds: 5
          tcpSocket:
            port: 6379
        volumeMounts:
        - name: redis-data
```

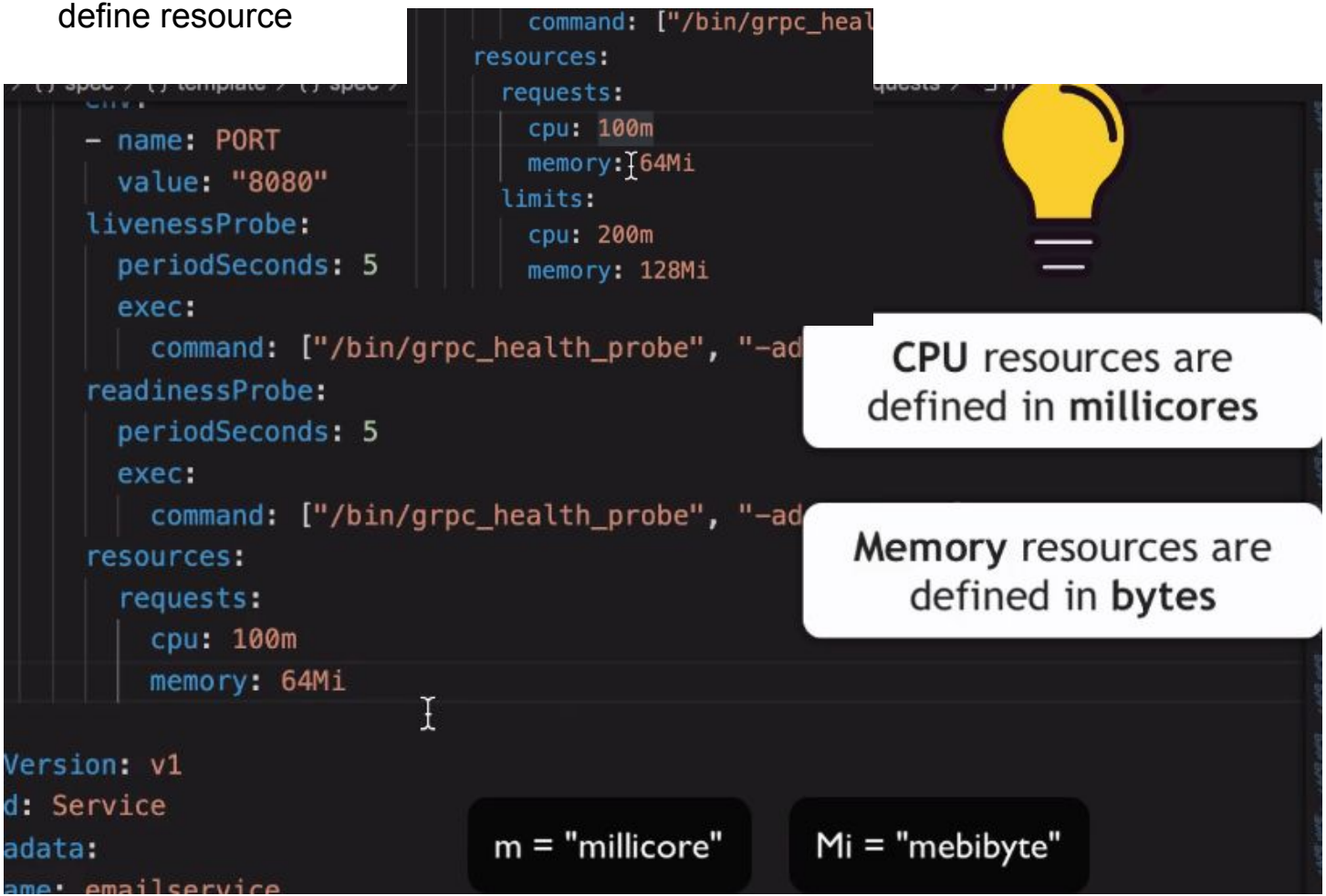
methods:
command
tcpsocket
http

wait 5 seconds before performing probe

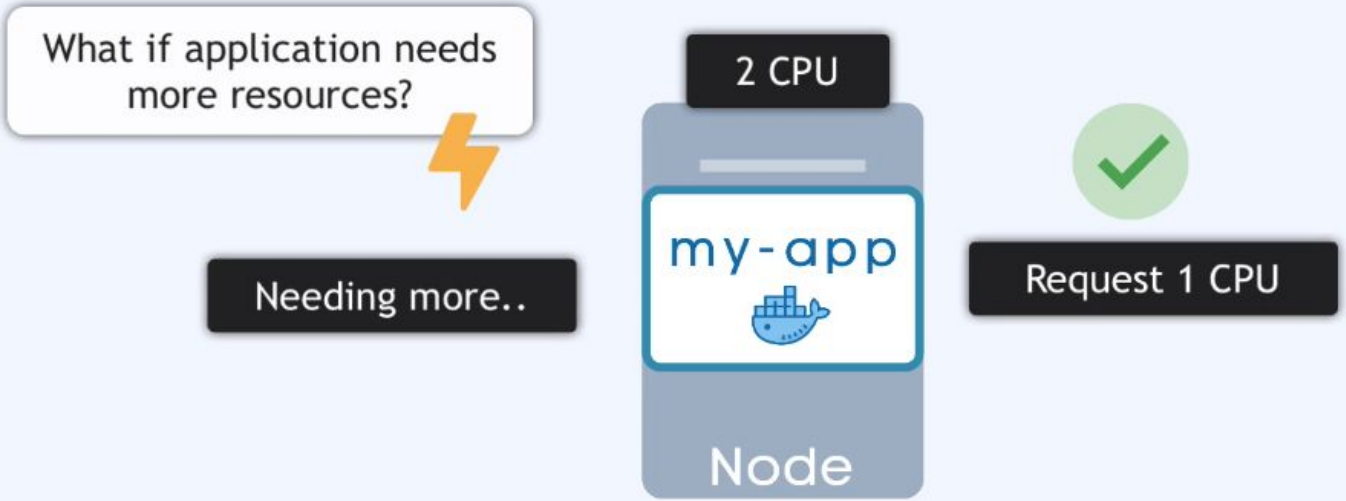

config.yaml — temp
config.yaml
config.yaml > {} spec > {} template > {} spec > [ ] containers > {} 0 > {} livenessProbe > {} tcpSocket > # port
containers:
- name: service
- containerPort: 3550
env:
- name: PORT
value: "3550"
livenessProbe:
periodSeconds: 5
exec:
command: ["/bin/grpc_health_probe", "-addr=:3550"]
readinessProbe:
periodSeconds: 5
3) HTTP probes
Kubelect sends an HTTP request to specified path and port


yaml > {} spec > {} template > {} spec > [ ] containers > {} 0 > {} livene
app: productcatalogservice
spec:
containers:
- name: service
image: gcr.io/google-samples/microservice
ports:
- containerPort: 3550
env:
- name: PORT
value: "3550"
livenessProbe:
periodSeconds: 5
httpGet:
path: /health
port: 3550
readinessProbe:
periodSeconds: 5
exec:
command: ["/bin/grpc_health_probe", "

define resource

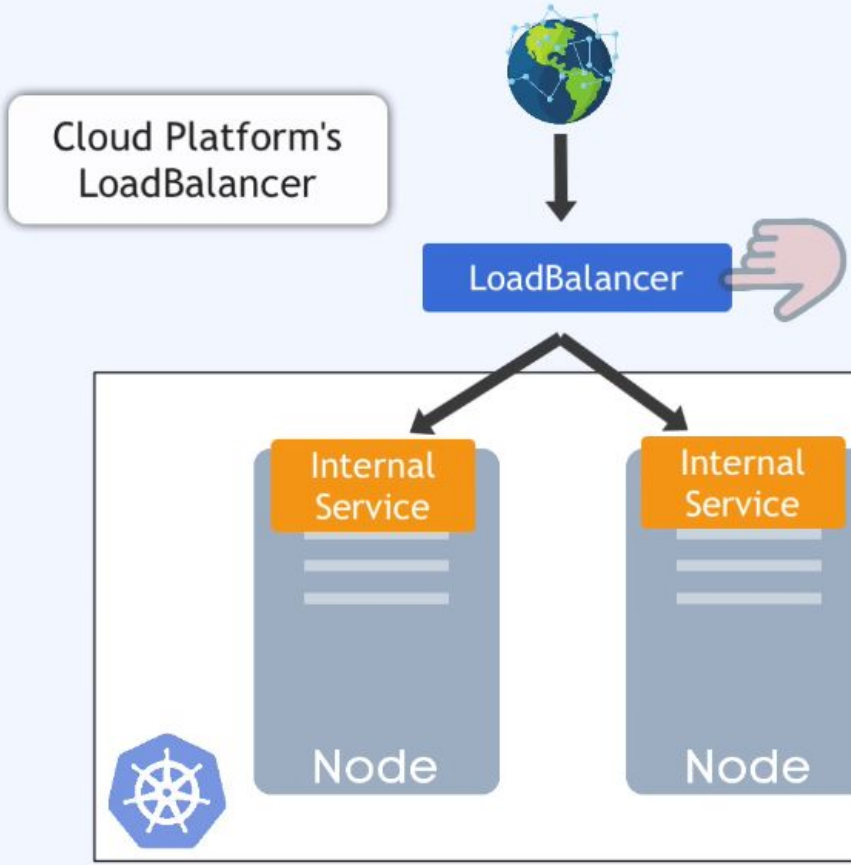


Why Resource Limits are important?



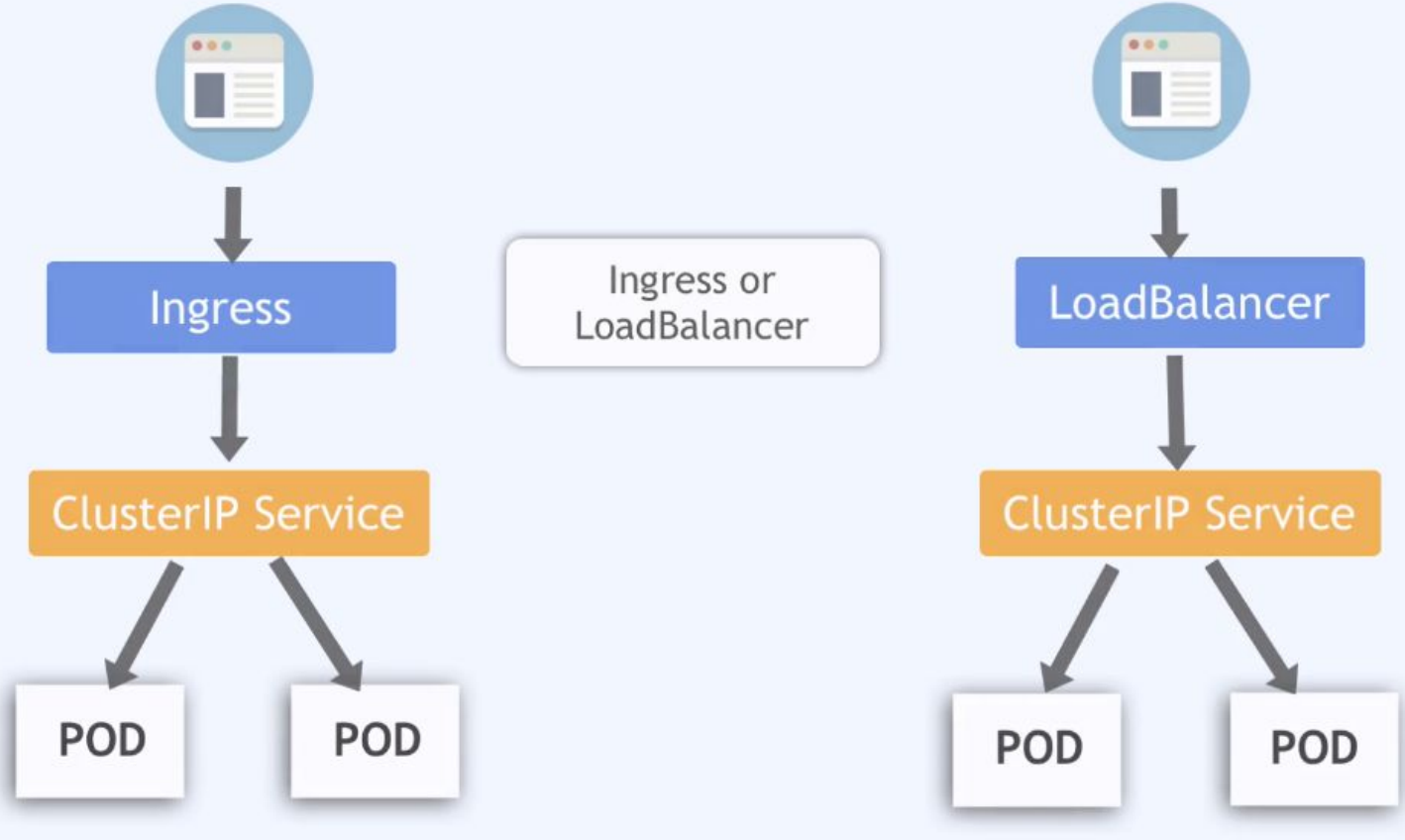
- Container will consume more than the requested resources
- If not limited, **container** could **consume all the Node's resources**

Why it's a Bad Practice?




```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: frontend-external
spec:
  type: LoadBalancer
  selector:
    app: frontend
  ports:
  - name: http
    port: 80
    targetPort: 8080
```

Ingress as alternative



Why it's important?

- **Custom Identifier** for your components

1. Group Pods with Labels

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: emailservice
spec:
  selector:
    matchLabels:
      app: emailservice
  template:
    metadata:
      labels:
        app: emailservice
    spec: ⋯
```

2. Reference in Service Component

```yaml
apiVersion: v1
kind: Service
metadata:
  name: emailservice
spec:
  type: ClusterIP
  selector:
    app: emailservice
  ports:
  - protocol: TCP
    port: 5000
    targetPort: 8080
```