Imagine that, you need to deploy one full fledge app
which consists of frontend application and backend database

How can we restrict access of backend database
to only within the Kubernetes cluster?

ClusterIP Service

Concept

Objectives

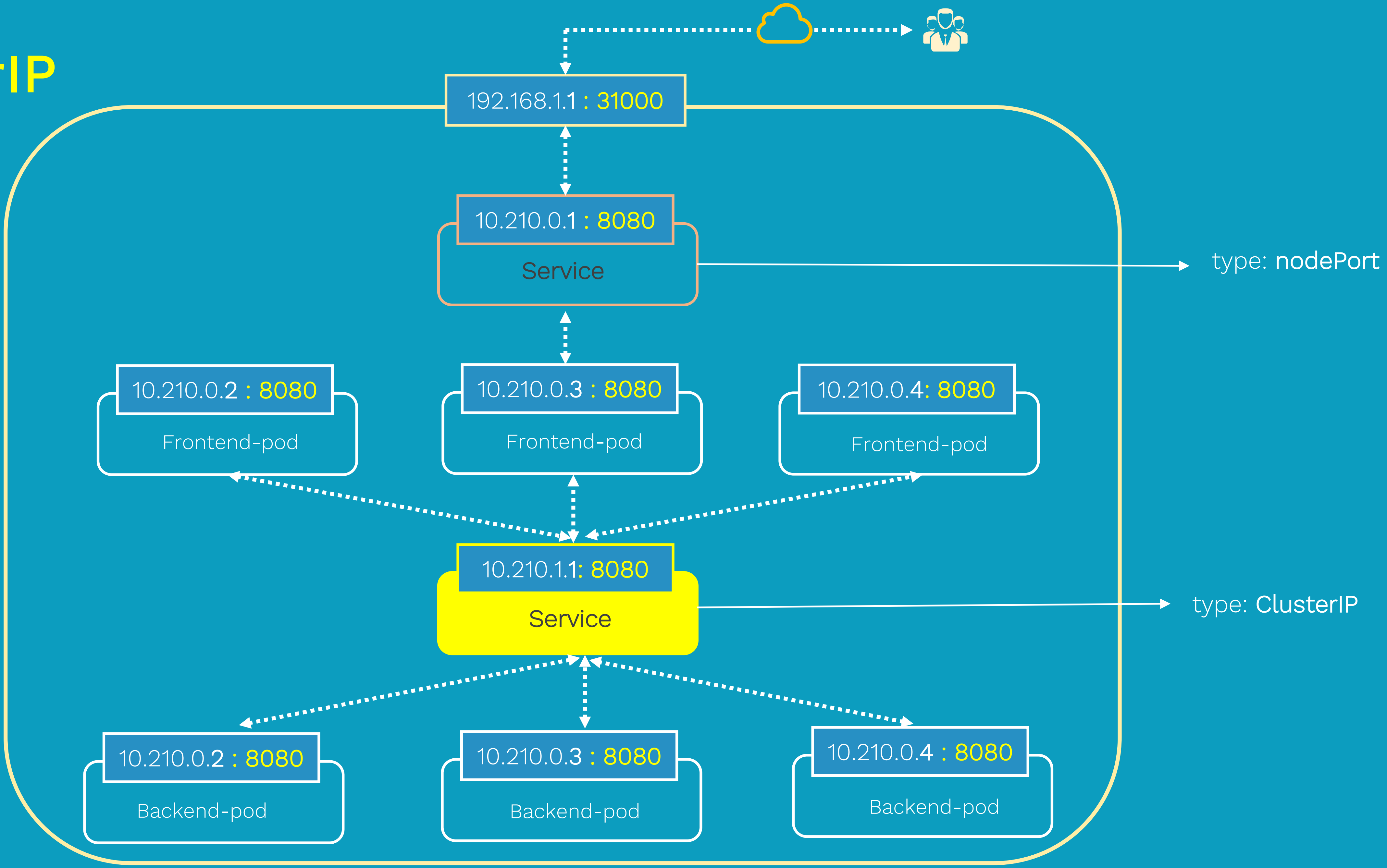
Concept

a. ClusterIP

Review Demo

a. Manifest file
b. Create and display
c. Test use cases
d. Clean up

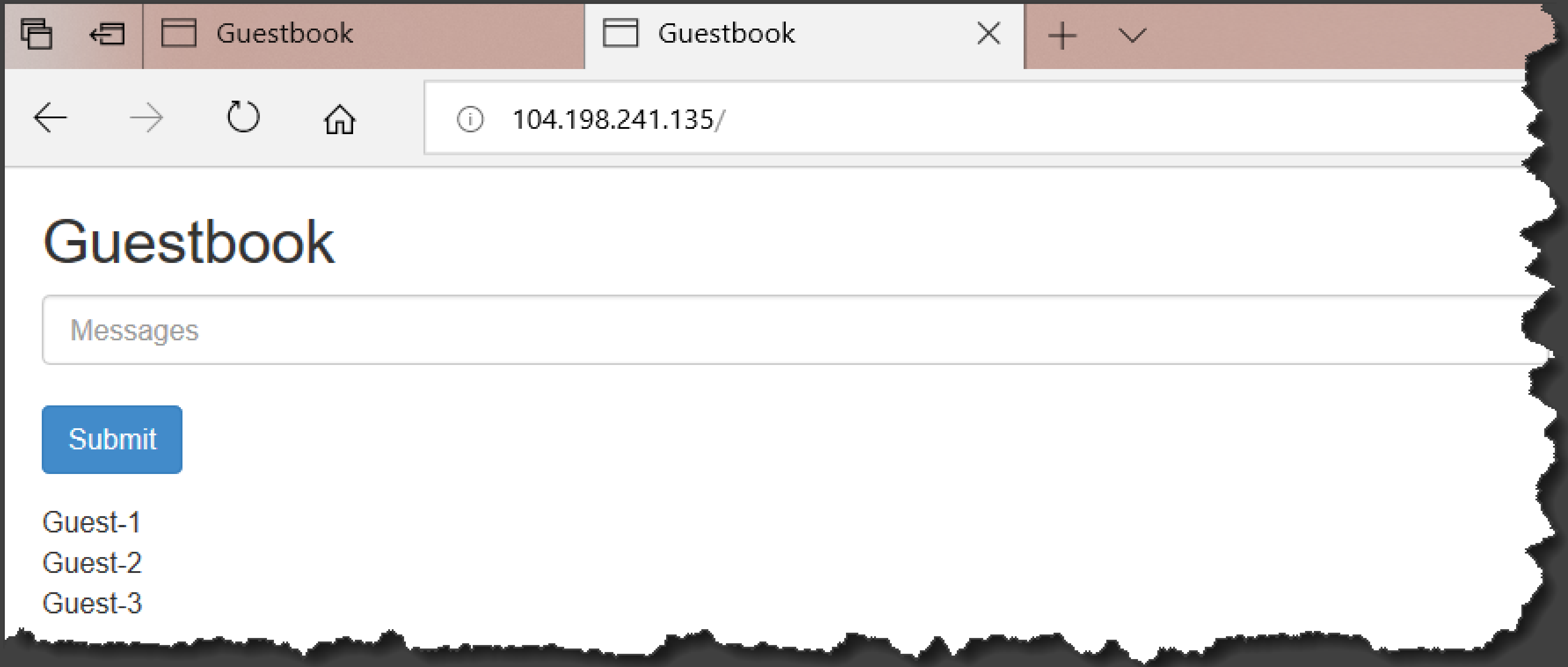
ClusterIP


192.168.1.1 : 31000
10.210.0.1 : 8080
Service
type: nodePort
10.210.0.2 : 8080
Frontend-pod
10.210.0.3 : 8080
Frontend-pod
10.210.0.4: 8080
Frontend-pod
10.210.1.1: 8080
Service
type: ClusterIP
10.210.0.2 : 8080
Backend-pod
10.210.0.3 : 8080
Backend-pod
10.210.0.4 : 8080
Backend-pod

Review

Demo

Guestbook app



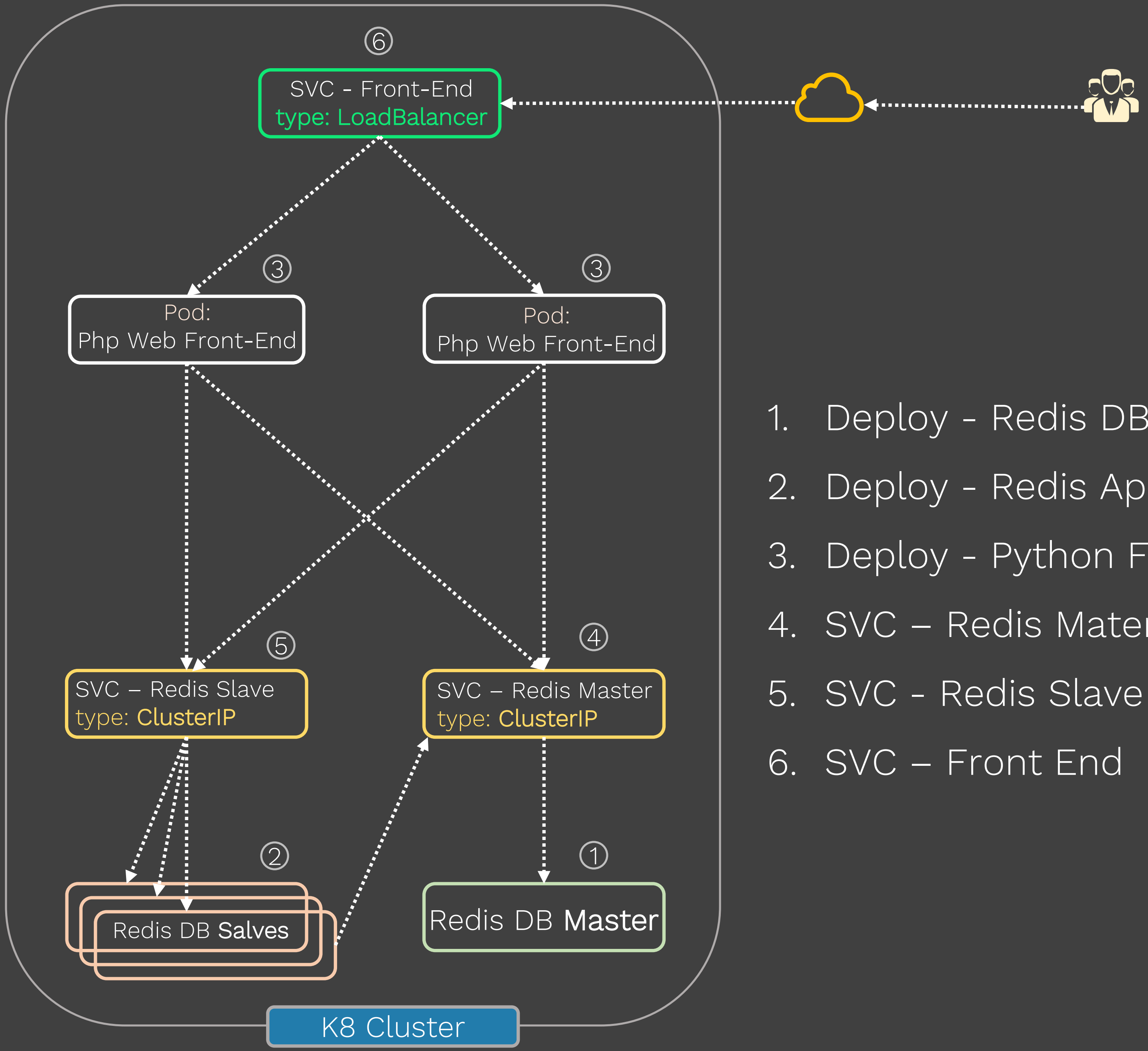
Guestbook

Messages

Submit

Guest-1
Guest-2
Guest-3

Ex: Deploying PHP Guestbook app with Redis



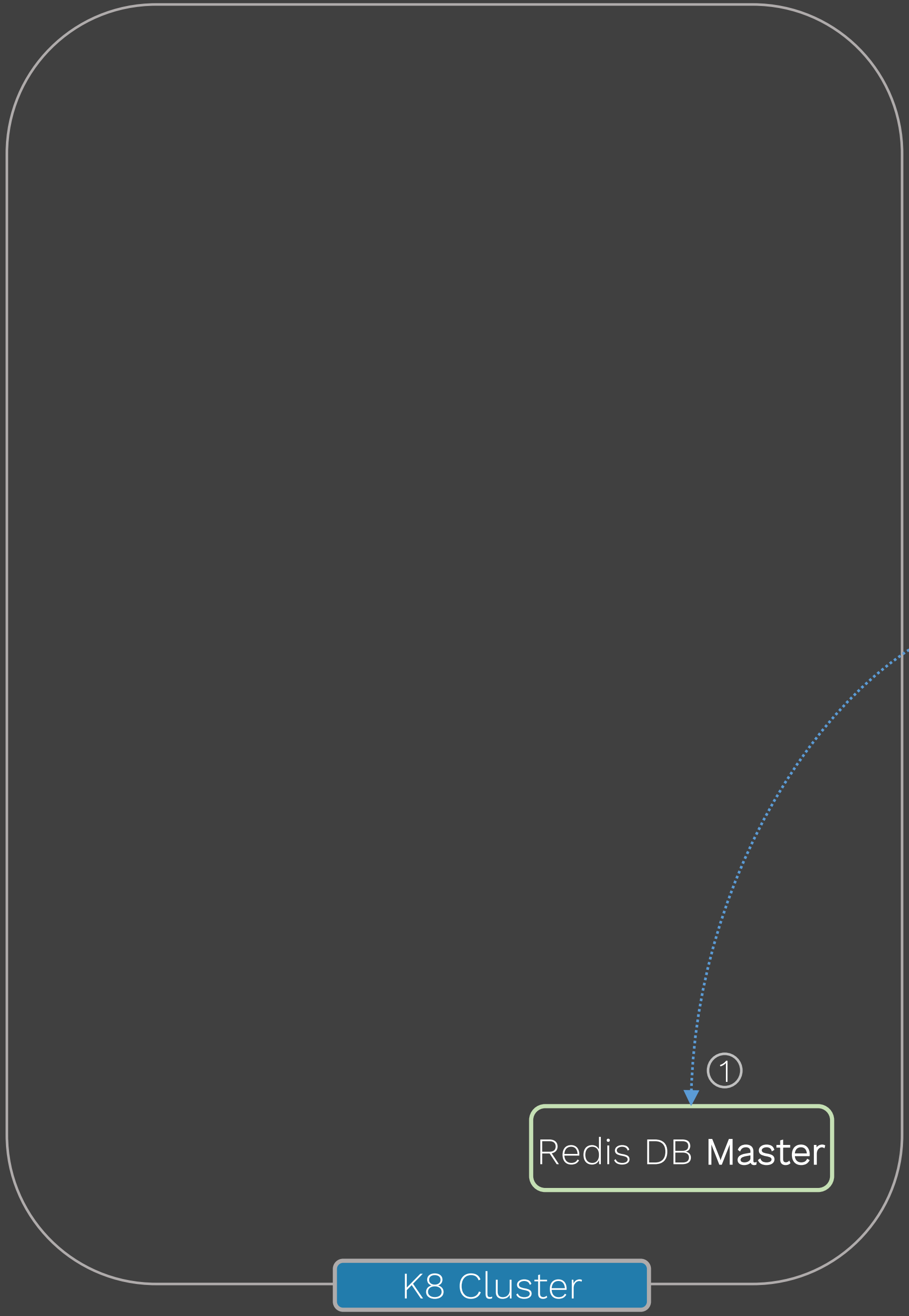
1. Deploy - Redis DB Master
2. Deploy - Redis App Slave
3. Deploy - Python Front End
4. SVC – Redis Mater
5. SVC - Redis Slave
6. SVC – Front End

srinathchalla@outlook.com

PART - 1

1. Redis Mater – Deployment
2. Redis Slave – Deployment
3. Front End – Deployment

1 Deploying 1 Redis DB Master

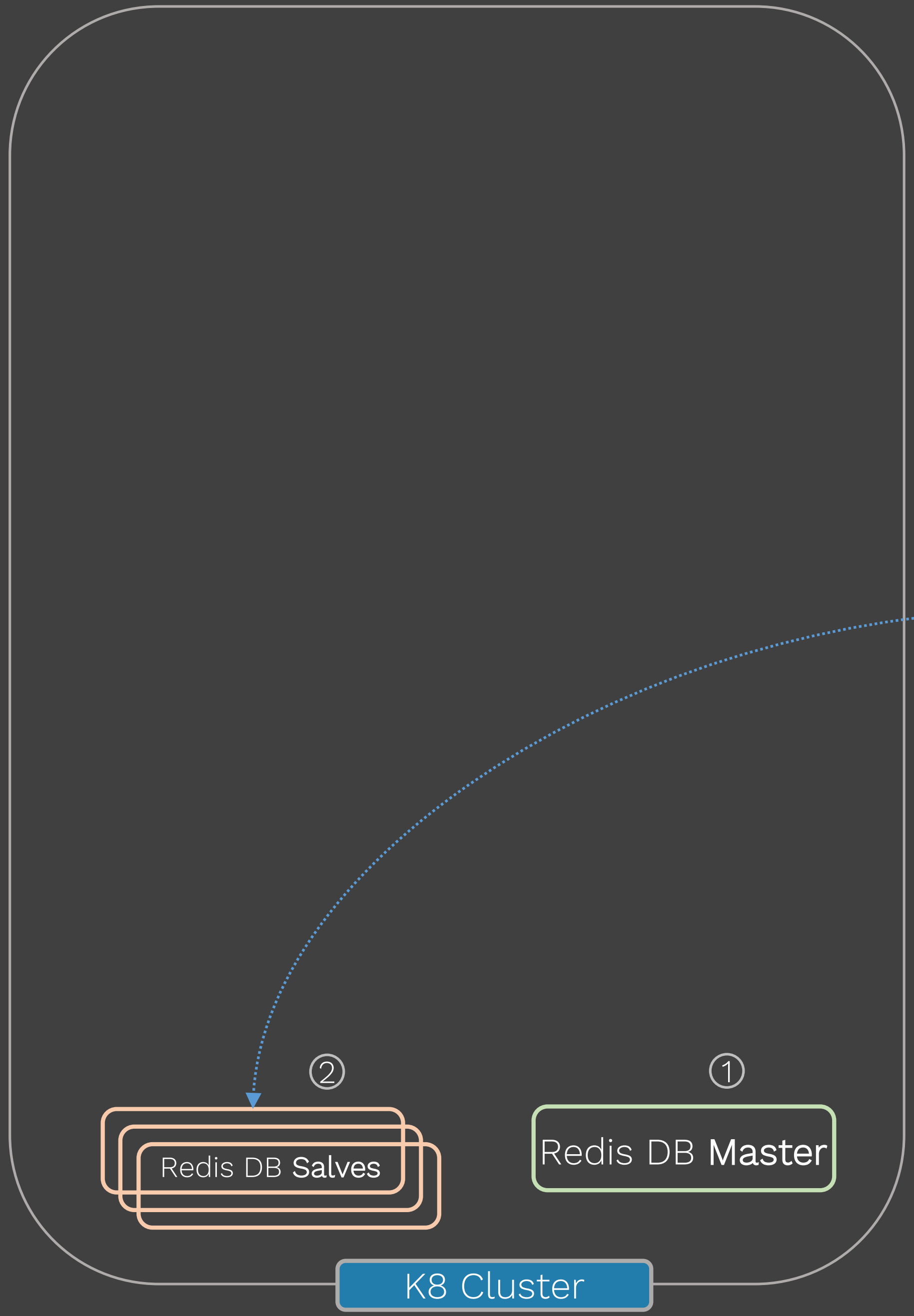


```
# redis-master-deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis-master
  labels:
    app: redis
spec:
  replicas: 1
  selector:
    matchLabels:
      app: redis
      role: master
      tier: backend
  template:
    metadata:
      labels:
        app: redis
        role: master
        tier: backend
    spec:
      containers:
      - name: master
        image: k8s.gcr.io/redis:e2e
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        ports:
        - containerPort: 6379
```

$\}$ ReplicaSet

$\}$ Pod

② Deploying 3 Redis DB Slaves

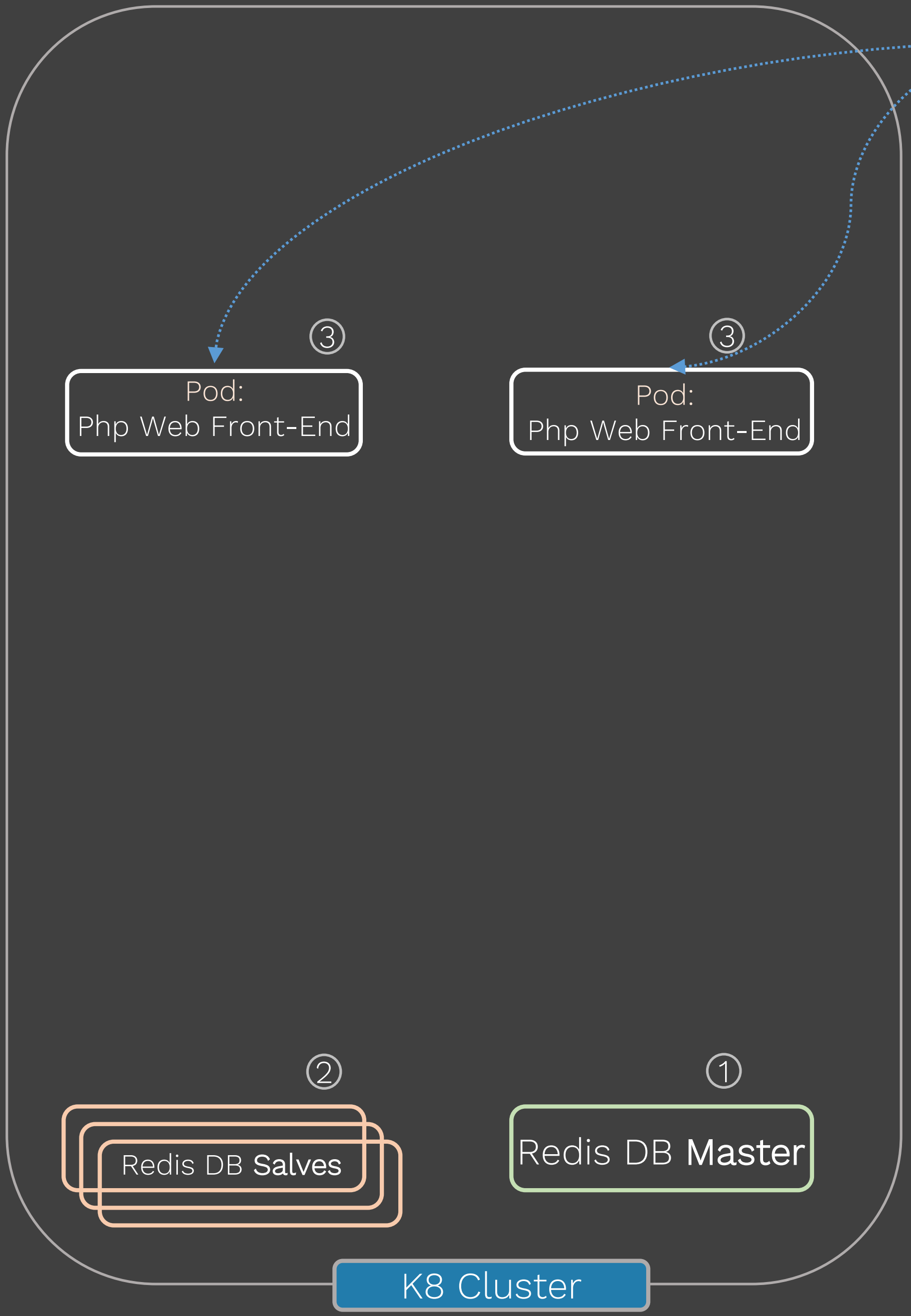


```
# redis-slave.yaml
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: redis-slave
  labels:
    app: redis
spec:
  replicas: 3
  selector:
    matchLabels:
      app: redis
      role: slave
      tier: backend
  template:
    metadata:
      labels:
        app: redis
        role: slave
        tier: backend
    spec:
      containers:
      - name: slave
        image: gcr.io/google_samples/gb-redisslave:v1
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        ports:
        - containerPort: 6379
```

ReplicaSet

Pod

③ Deploying 2 front-end PHP web pods



```
# redis-master-deployment.yaml
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1
kind: Deployment
metadata:
  name: frontend
  labels:
    app: guestbook
spec:
  replicas: 2
  selector:
    matchLabels:
      app: guestbook
      tier: frontend
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google-samples/gb-frontend:v4
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        ports:
        - containerPort: 80
```

ReplicaSet

Pod

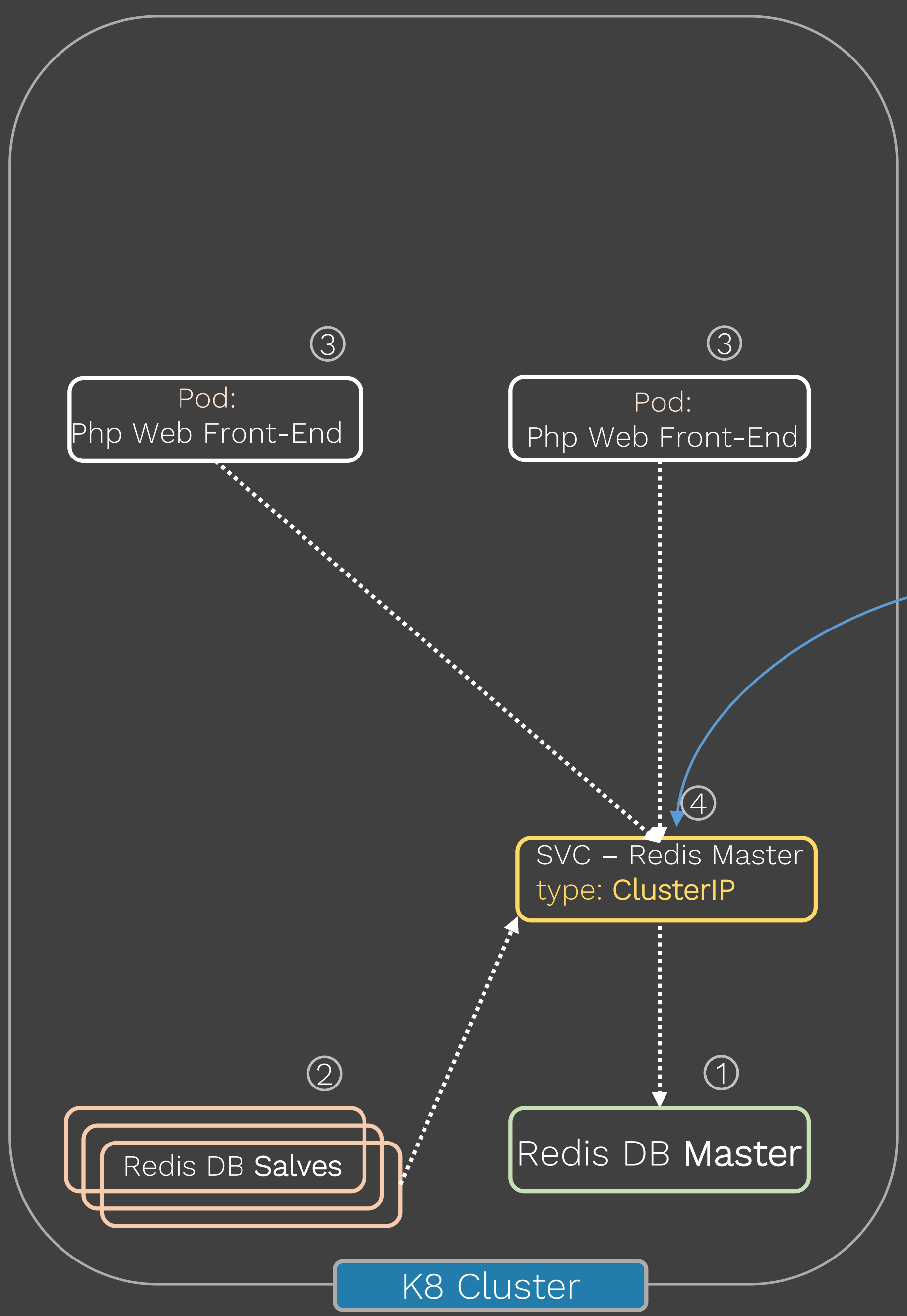
PART - 2

1. Redis Mater – SVC_ClusterIP

2. Redis Slave - SVC_ClusterIP

3. Front End - SVC_LoadBalancer

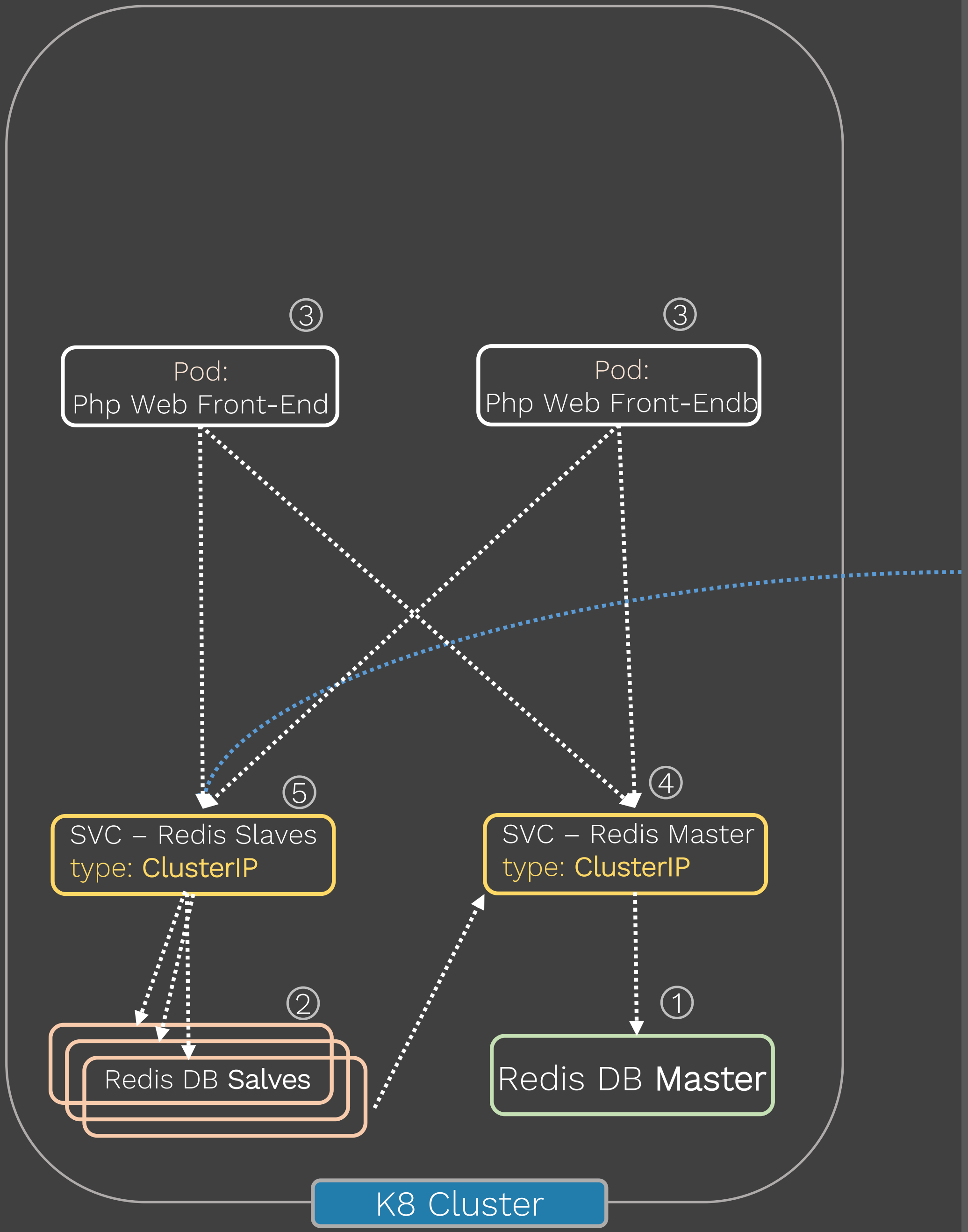
④ Creating Service for Redis Master



```yaml
# redis-master-service.yaml

apiVersion: v1
kind: Service
metadata:
  name: redis-master-svc
  labels:
    app: redis
    role: master
    tier: backend
spec:
  ports:
  - port: 6379
    targetPort: 6379
  type: ClusterIP
  selector:
    app: redis
    role: master
    tier: backend
```

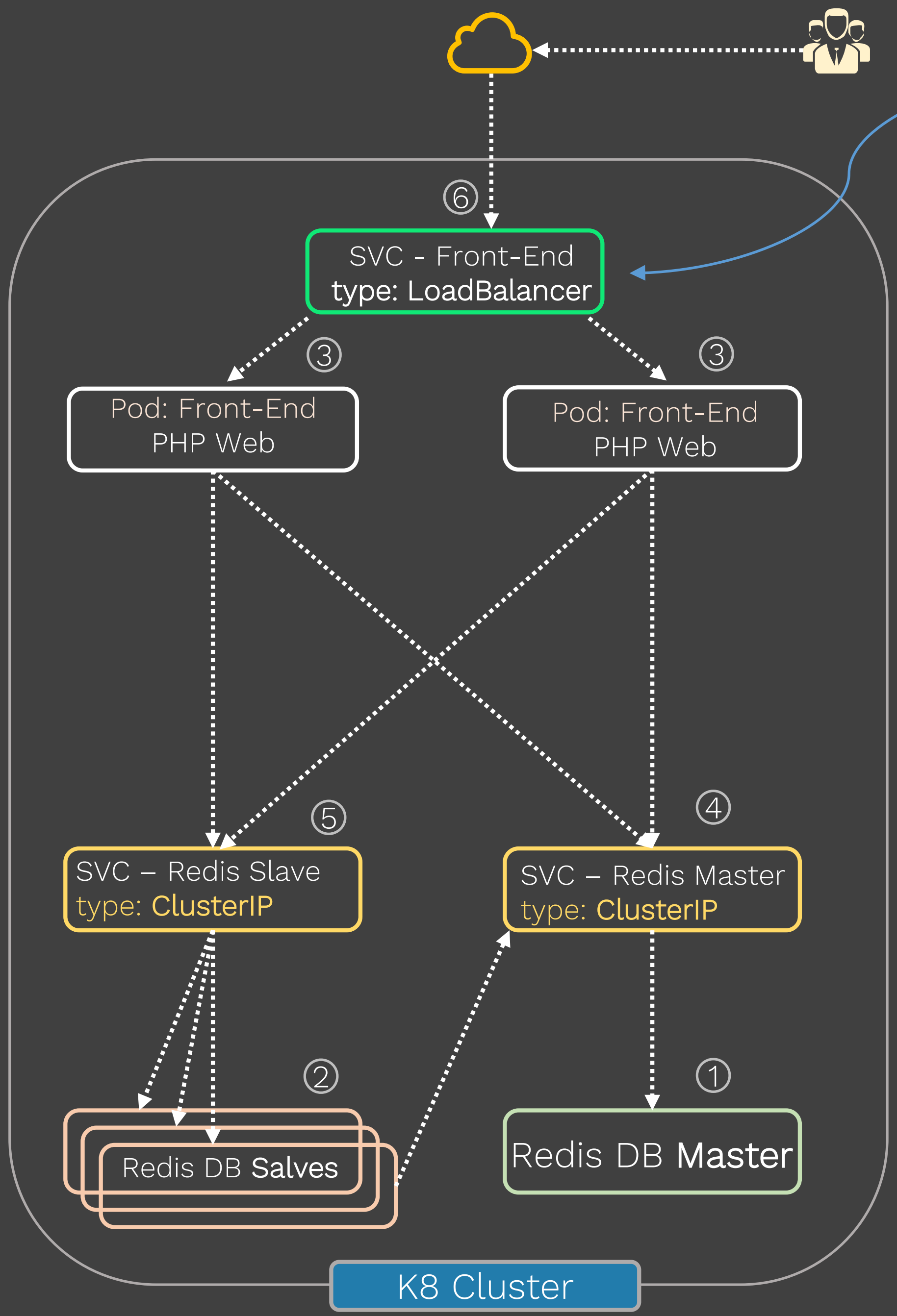
4 Creating Service for Redis DB Slaves



```yaml
# redis-master-deployment.yaml

apiVersion: v1
kind: Service
metadata:
  name: redis-slave-svc
  labels:
    app: redis
    role: slave
    tier: backend
spec:
  ports:
  - port: 6379
  type: ClusterIP
  selector:
    app: redis
    role: slave
    tier: backend
```

⑥ Creating Service for front-end apps



```yaml
# redis-master-deployment.yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend-svc
  labels:
    app: guestbook
    tier: frontend
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: guestbook
    tier: frontend

# NOTE: if your cluster doesn’t support LB, use type as NP #
type: NodePort
```

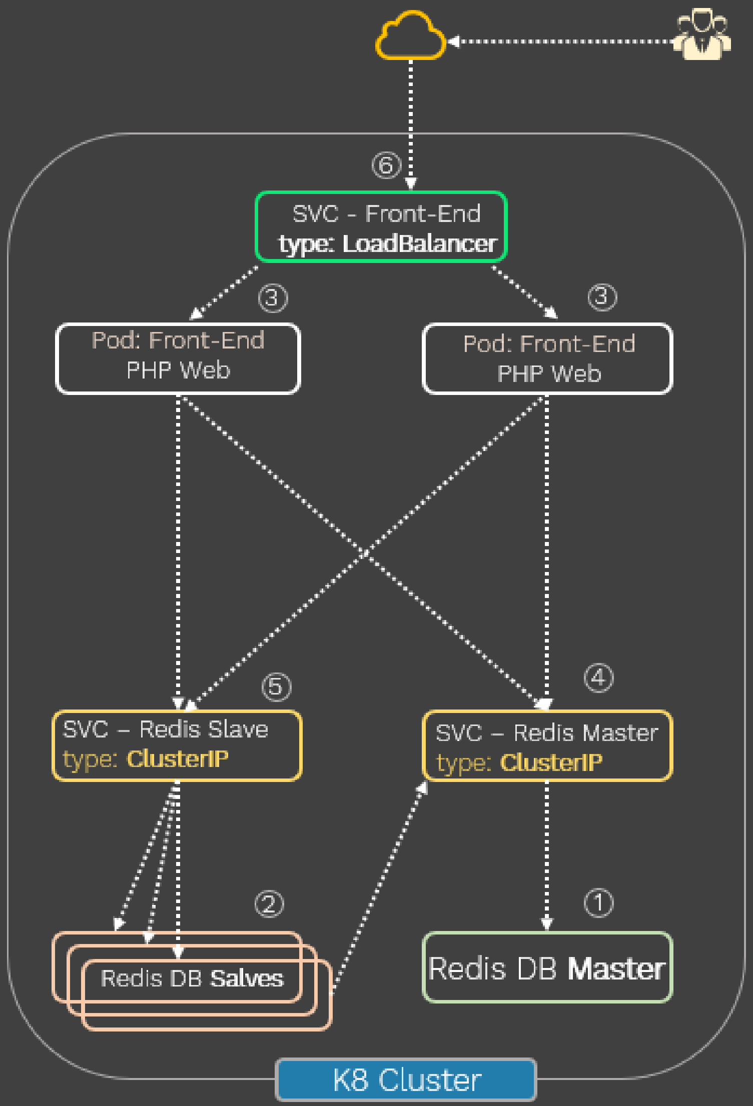
Services –Display

```
srinath@master:~ $ kubectl get po -l tier=backend
NAME                          READY     STATUS    RESTARTS   AGE
redis-master-585798d8ff-bx5z6   1/1       Running   0          1h
redis-slave-5dfddd78f5-4f7gc    1/1       Running   0          1h
redis-slave-5dfddd78f5-7z84b    1/1       Running   0          1h
redis-slave-5dfddd78f5-bwg74    1/1       Running   0          1h
```

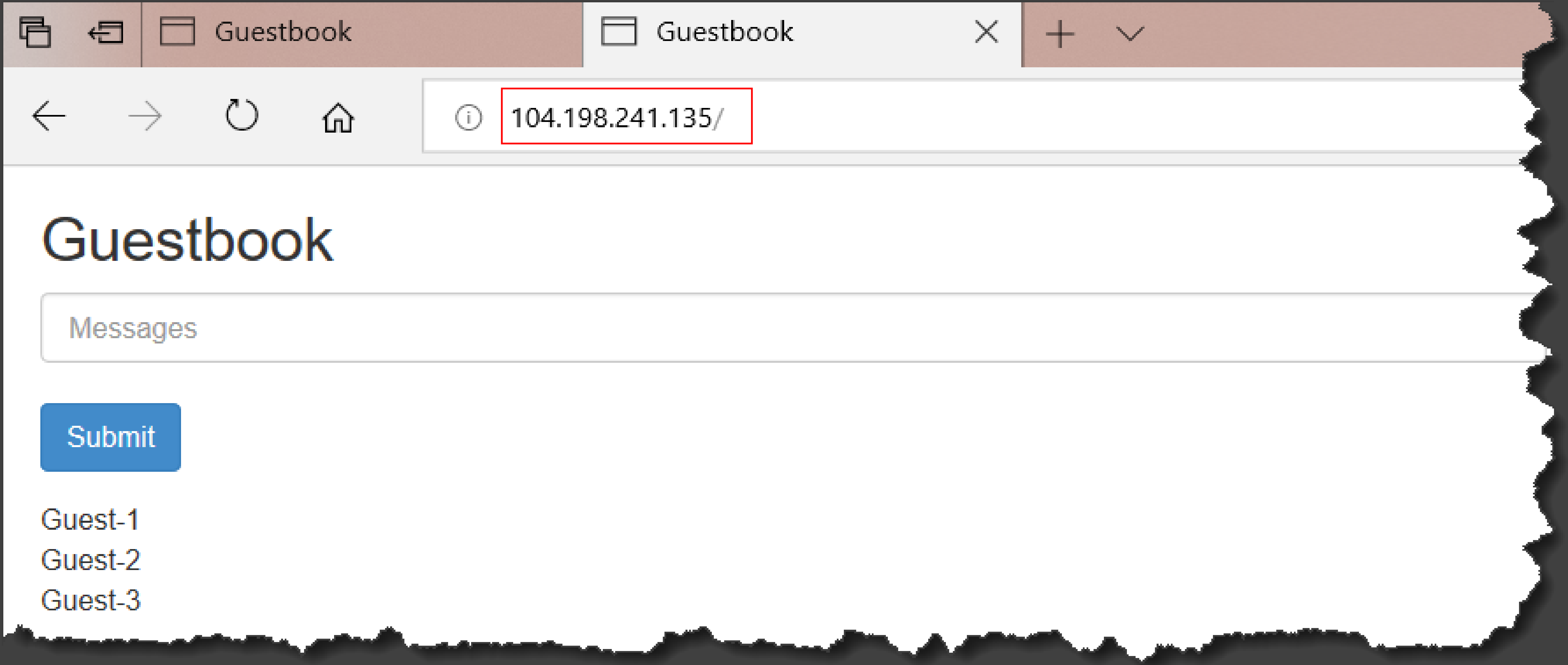
```
srinath@master:~ $ kubectl get po –l tier=frontend
NAME                   READY     STATUS    RESTARTS   AGE
frontend-d5d5947-7skmx   1/1       Running   0          1h
frontend-d5d5947-vfc9w   1/1       Running   0          1h
```

```
srinath@master:~ $ kubectl get svc -l tier=backend
NAME               TYPE        CLUSTER-IP      EXTERNAL-IP    PORT(S)     AGE
redis-master-svc   ClusterIP   10.11.240.185   <none>         6379/TCP    11m
redis-slave-svc    ClusterIP   10.11.252.193   <none>         6379/TCP    10m
```

```
srinath@master:~ $ kubectl get svc –l tier=frontend
NAME       TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)        AGE
frontend   LoadBalancer   10.11.250.88   104.198.241.135   80:31612/TCP   8m
```



Services – Accessing Guestbook app using LoadBalancer IP



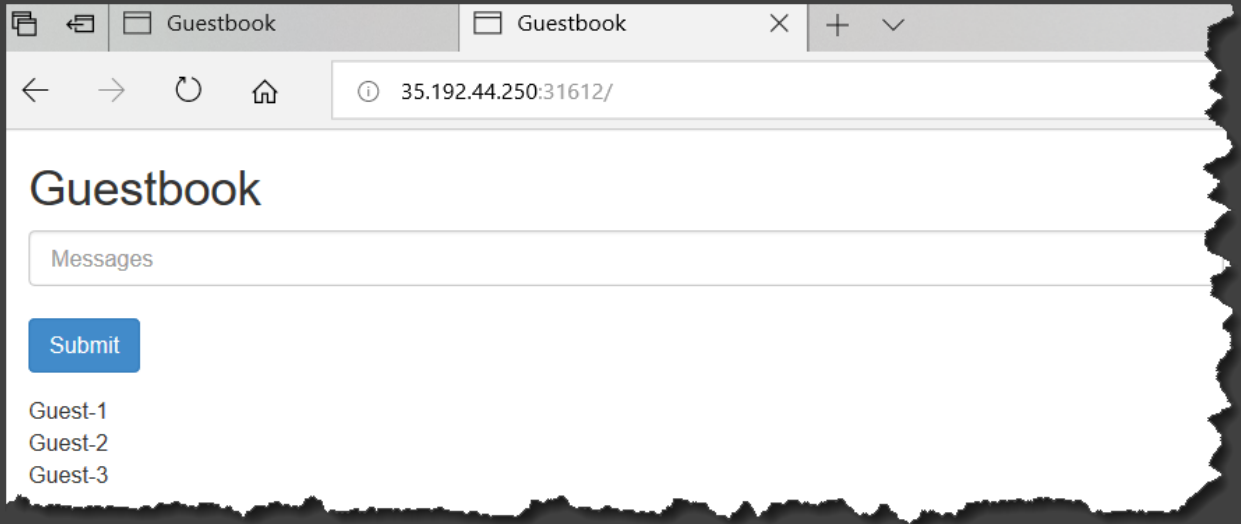
Guestbook

Guest-1
Guest-2
Guest-3

Services – Accessing Guestbook app using Node IP

```
srinath@master:~   kubectl get no -o w
NAME                                       STATUS     ROLES     AGE        VERSION         EXTERNAL-IP       OS-IMAGE
KERNEL-VERSION   CONTAINER-RUNTIME
gke-cluster-1-default-pool-203d9e7c-2t9h   Ready      <none>    2d         v1.9.7-gke.6    35.192.44.250     Container-
Optimized OS from Google   4.4.111+         docker://17.3.2
gke-cluster-1-default-pool-203d9e7c-fqr0   Ready      <none>    2d         v1.9.7-gke.6    35.226.89.109     Container-
Optimized OS from Google   4.4.111+         docker://17.3.2
gke-cluster-1-default-pool-203d9e7c-q5mj   Ready      <none>    2d         v1.9.7-gke.6    104.154.72.168    Container-
Optimized OS from Google   4.4.111+         docker://17.3.2
```

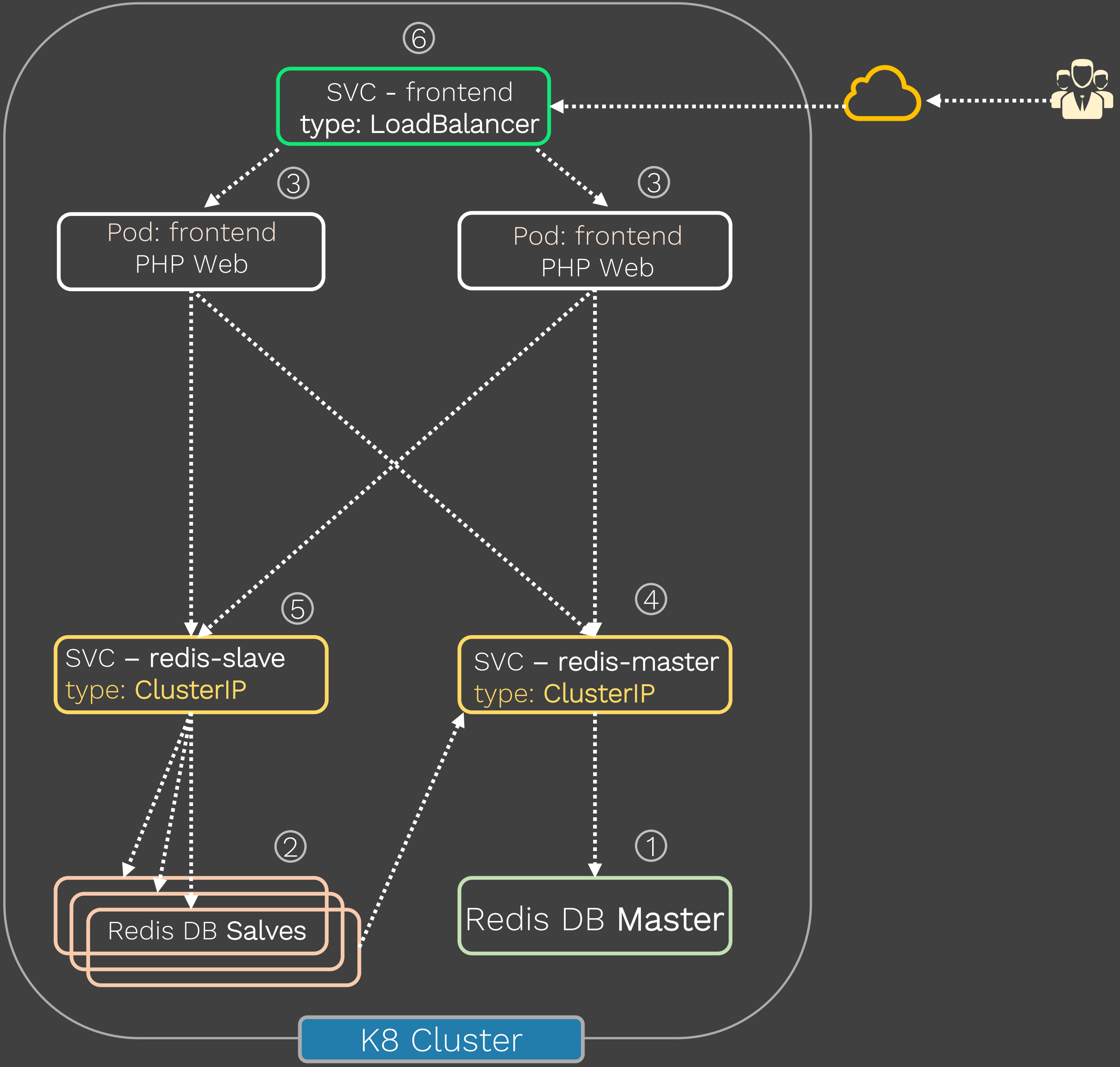
```
srinath@master:~ $ kubectl get svc -l tier=frontend
NAME       TYPE           CLUSTER-IP     EXTERNAL-IP       PORT(S)        AGE
frontend   LoadBalancer   10.11.250.88   104.198.241.135   80:31612/TCP   8m
```



Guestbook

Guest-1
Guest-2
Guest-3

Recap



1. Redis DB Master
2. Redis App Slave
3. Frontend App
4. SVC-Redis DB Master
5. SVC-Redis DB Slave
6. SVC – Front End

Summary

Concept

a. ClusterIP - Overview

Review Demo

a. Manifest file
b. Create and display
c. Test use cases
d. Clean up

Coming up...

Demo

ClusterIP