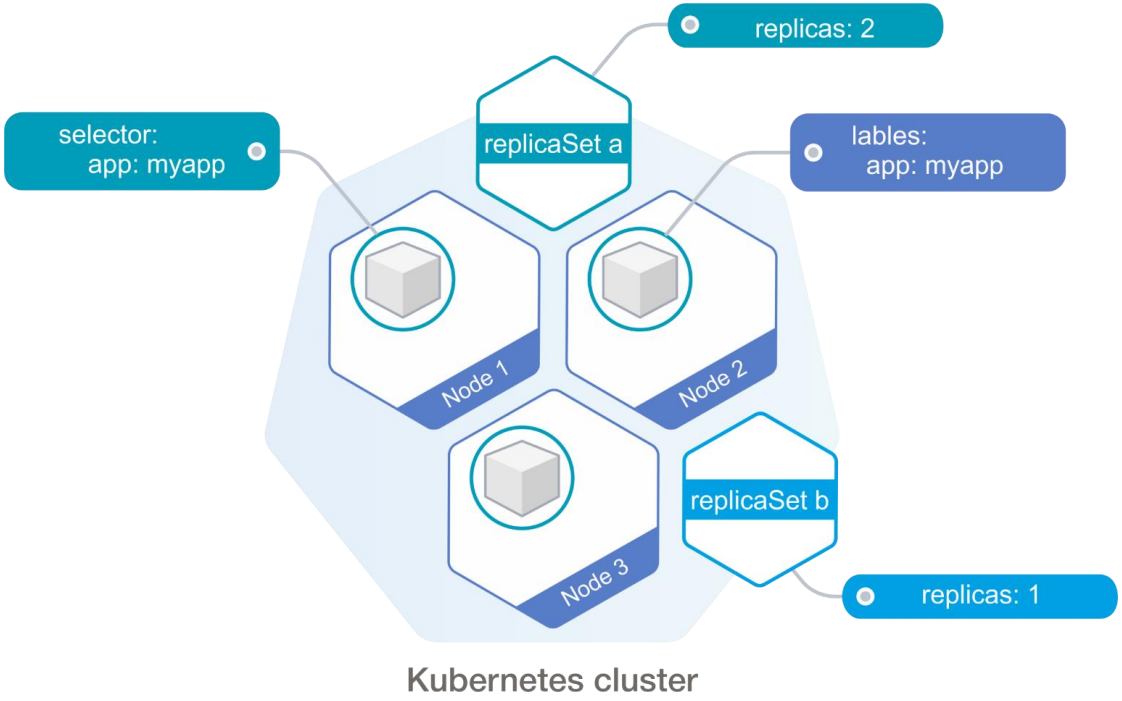
СЛЁРМ + Southbridge

Health Check. Публикация сервисов и приложений.

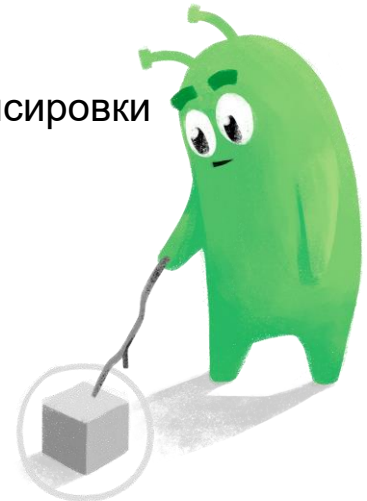
Марсель Ибраев
СТО Слёрм

Deployment



Probes

- Liveness Probe
  - Контроль за состоянием приложения во время его **жизни**
  - Исполняется постоянно
- Readiness Probe
  - Проверяет, **готово ли** приложение принимать трафик
  - В случае неудачного выполнения, приложение убирается из балансировки
  - Исполняется постоянно
- Startup Probe
  - Проверяет, **запустилось ли** приложение
  - Исполняется при старте



Способы публикации

Service: L3 OSI, NAT, kube-proxy

```
-A KUBE-MARK-MASQ -j MARK --set-xmark 0x4000/0x4000
-A KUBE-NODEPORTS -p tcp -m comment --comment "s000000/np2:http"
-m tcp --dport 30029 -j KUBE-MARK-MASQ

-A KUBE-NODEPORTS -p tcp -m comment --comment "s000000/np2:http"
-m tcp --dport 30029 -j KUBE-SVC-2F3FOG2AWAH5Y5PC
```

Ingress: L7 OSI, HTTP и HTTPS, nginx, envoy, traefik, haproxy

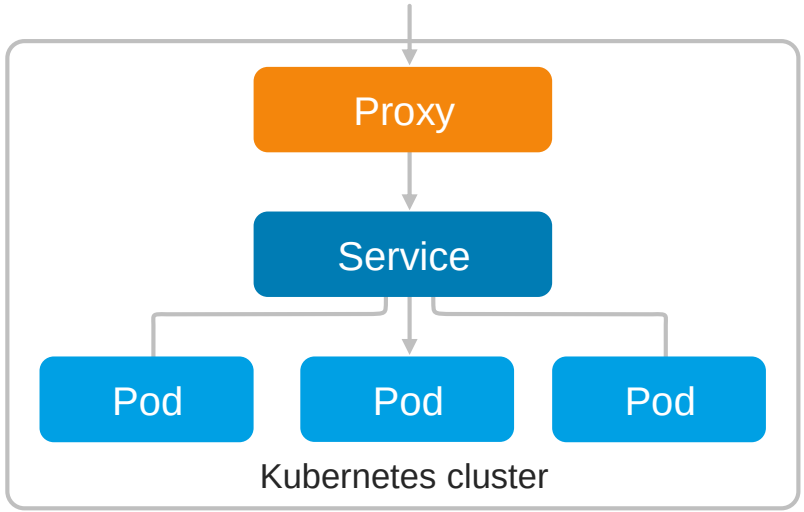
```
server {
  server_name slurm.io  ssl on;
  location {
     proxy_pass http://backend;
  }
}
```

Kubernetes Service

- ClusterIP
- NodePort
- LoadBalancer
- ExternalName
- ExternalIPs

ClusterIP

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: my-app
  type: ClusterIP
  ports:
  - name: http
    port: 80
    targetPort: 80
    protocol: TCP
```

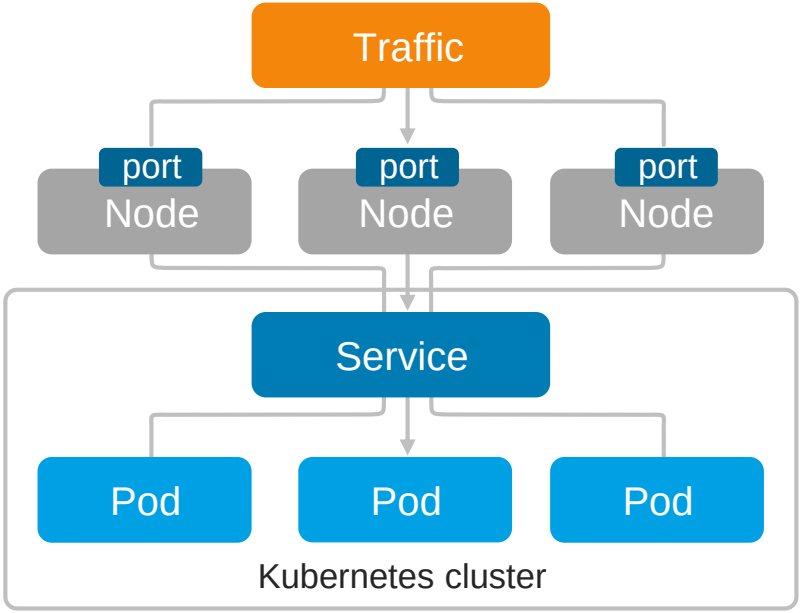



```
kubectl proxy --port=8080  http://localhost:8080/api/v1/proxy/
namespacesdefault/services/my-service:http/

kubectl port-forward service/my-service 10000:80
```

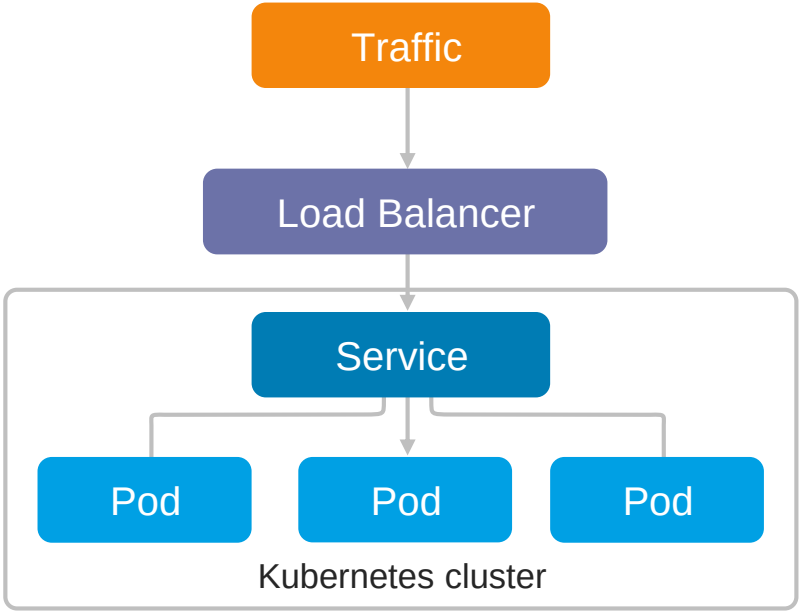
NodePort

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service-np
spec:
  selector:
    app: my-app
  type: NodePort
  ports:
  - name: http
    port: 80
    targetPort: 80
    protocol: TCP
```



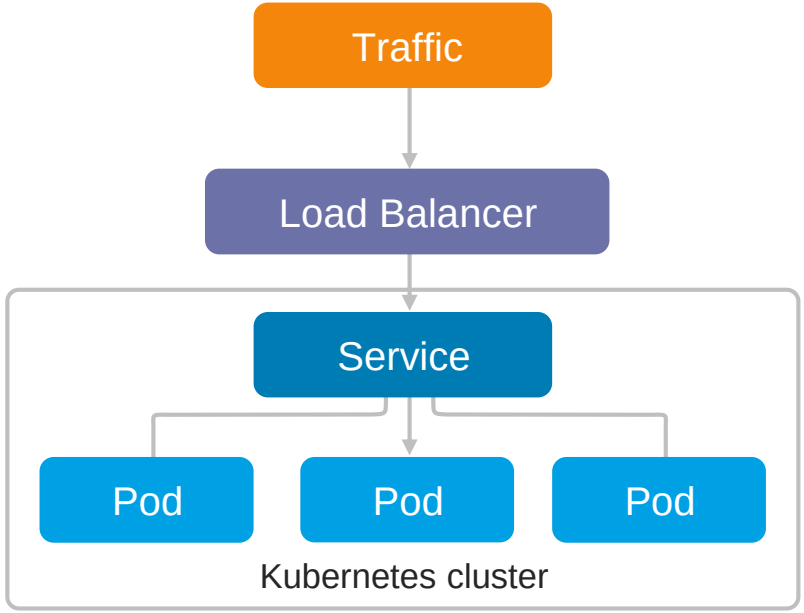
LoadBalancer

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service-lb
spec:
  selector:
    app: my-app
  type: LoadBalancer
  ports:
  - name: http
    port: 80
    targetPort: 80
    protocol: TCP
```



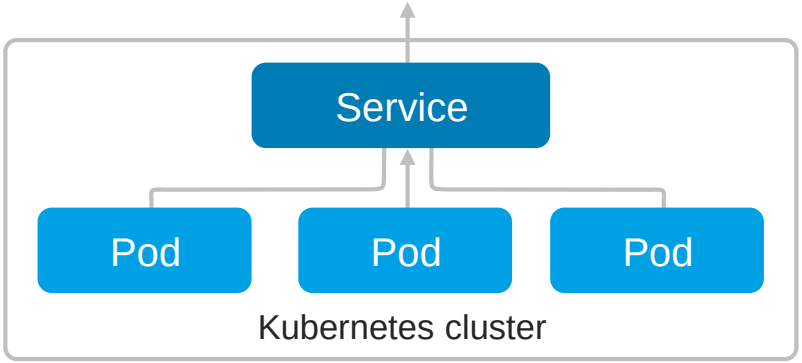
LoadBalancer static IP

```
apiVersion: v1
kind: Service
metadata:
  name: my-service-lb
spec:
  selector:
    app: my-app
  type: LoadBalancer
  loadBalancerIP:
  "1.1.1.1"
  ports:
  - port: 80
    targetPort: 80
```



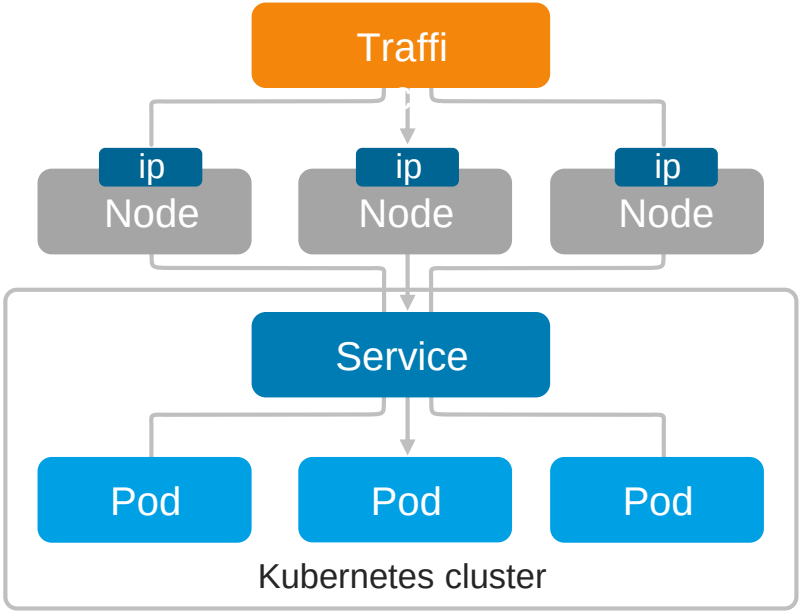
ExternalName

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: ExternalName
  externalName: example.com
```



ExternalIPs

```
apiVersion: v1
kind: Service
metadata:
  name: myservice
spec:
  selector:
    app: my-app
  ports:
  - name: http
    port: 80
    targetPort: 80
    protocol: TCP
  externalIPs:
  - 80.11.12.10
```



Kubernetes Service

- ClusterIP
- NodePort
- LoadBalancer
- ExternalName
- ExternalIPs

Headless

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: my-app
  ClusterIP: none
  ports:
  - name: http
    port: 80
    targetPort: 80
    protocol: TCP
```

```
⚡ kubectl exec pod-1 -- nslookup my-service
Server:      10.0.0.10
Address:    10.0.0.10#53

Name:    my-service.default.svc.cluster.local
Address: 10.0.12.5
Name:    my-service.default.svc.cluster.local
Address: 10.0.12.6
Name:    my-service.default.svc.cluster.local
Address: 10.0.12.7
```

Service – это в итоге какой-то прокси?

Service

```
-A KUBE-SERVICES
  -d 1.1.1.1/32
  -p tcp
  -m comment --comment "mynamespace/myservice:http cluster
  IP"
  -m tcp --dport 80
-j KUBE-SVC-UT6A43GJFBEDBO3V
```

Service

```
-A KUBE-SERVICES
  -d 1.1.1.1/32
  -p tcp
  -m comment --comment "mynamespace/myservice:http cluster
  IP"
  -m tcp --dport 80
-j KUBE-SVC-UT6A43GJFBEDBO3V


-A KUBE-SVC-UT6A43GJFBEDBO3V
  -m comment --comment "mynamespace/myservice:http"
  -m statistic
    --mode random --probability 0.50000000000
-j KUBE-SEP-MMYWB6DZJI4RZ5CQ

-A KUBE-SVC-UT6A43GJFBEDBO3V
  -m comment --comment "mynamespace/myservice:http"
-j KUBE-SEP-J33LX377GA3DLDWM
```

Service

```
-A KUBE-SVC-UT6A43GJFBEDBO3V
  -m comment --comment "mynamespace/myservice:http"
-j KUBE-SEP-J33LX377GA3DLDWM



-A KUBE-SEP-J33LX377GA3DLDWM
  -p tcp
  -m comment --comment "mynamespace/myservice:http"
  -m tcp
-j DNAT
  --to-destination 10.102.3.49:80
```

Service

```
-A KUBE-SVC-UT6A43GJFBEDBO3V
  -m comment --comment "mynamespace/myservice:http"
  -m statistic
    --mode random --probability 0.50000000000
-j KUBE-SEP-MMYWB6DZJI4RZ5CQ


-A KUBE-SEP-MMYWB6DZJI4RZ5CQ
  -p tcp
  -m comment --comment "mynamespace/myservice:http"
  -m tcp
-j DNAT
  --to-destination 10.102.0.93:80
```

Service

```
$ kubectl get po --namespace=mynamespace -o wide
```

pod-1	1/1	Running	0	6h	10.102.3.49
pod-2	1/1	Running	0	6h	10.102.0.93

Service

- Статический IP

Service

- Статический IP
- DNS имя в kube-dns на этот IP

  (myservice.mynamespace.svc.cluster.local)

Service

- Статический IP
- DNS имя в kube-dns на этот IP

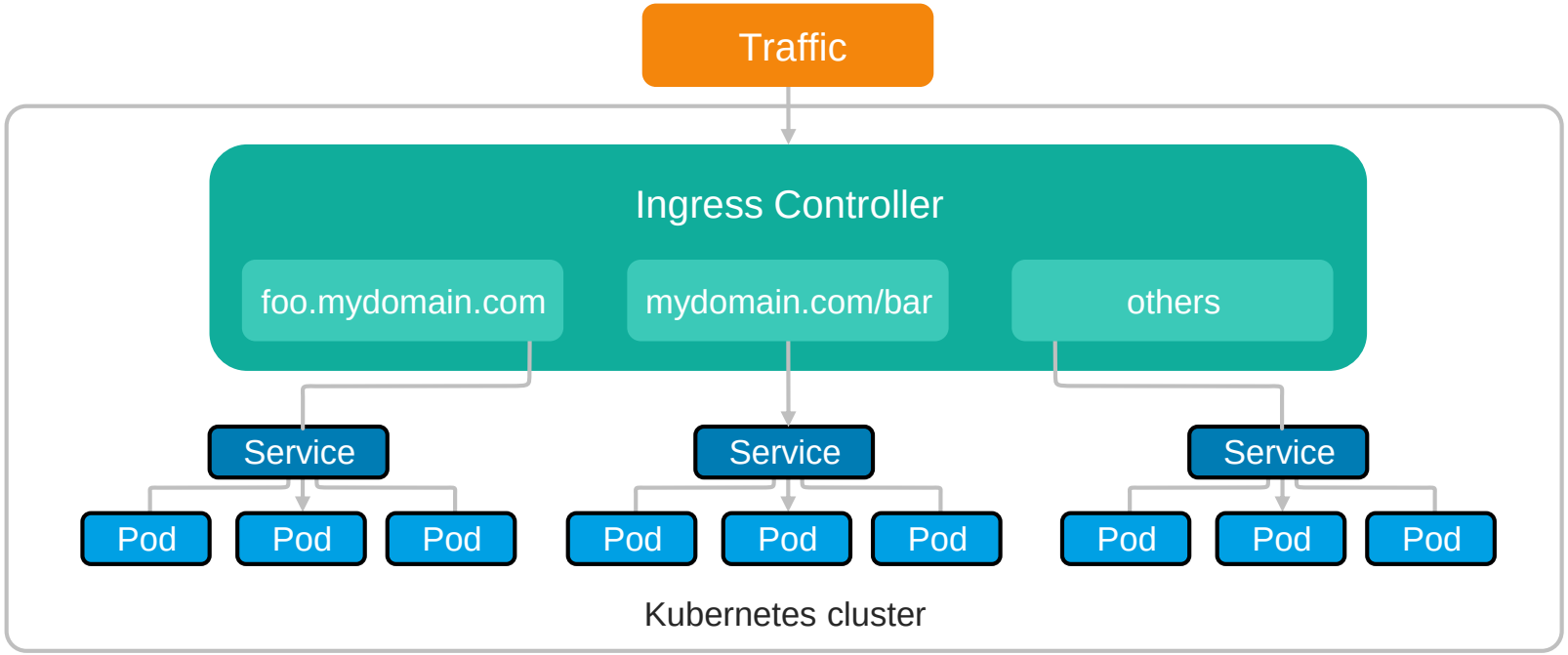
  (myservice.mynamespace.svc.cluster.local)
- Правила iptables для роутинга

Service

- Статический IP
- DNS имя в kube-dns на этот IP

  (myservice.mynamespace.svc.cluster.local)
- Правила iptables для роутинга
- Service - это не прокси!

Ingress


Traffic
Ingress Controller
foo.mydomain.com
mydomain.com/bar
others
Service
Service
Service
Pod
Pod
Pod
Pod
Pod
Pod
Pod
Pod
Pod
Kubernetes cluster

Ingress

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
  annotations:
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
spec:
  rules:
  - host: foo.mydomain.com
    http:
      paths:
      - pathType: Prefix
        path: “/”
        backend:
          service:
            name: my-service
            port:
              number: 80
```

HOST: foo.mydomain.com

Ingress

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
  annotations:
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
spec:
  rules:
  - host: foo.mydomain.com
    http:
      paths:
      - pathType: Prefix
        path: “/”
        backend:
          service:
            name: my-service
            port:
              number: 80
```

HOST: foo.mydomain.com

Ingress

Указываем сертификат в Ingress

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: tls-ingress
spec:
  tls:
  - hosts:
    - sslfoo.com
    secretName: secret-tls
```

Создаем секрет с сертификатом

```
apiVersion: v1
data:
  tls.crt: base64 encoded cert
  tls.key: base64 encoded key
kind: Secret
metadata:
  name: secret-tls
  namespace: default
type: kubernetes.io/tls
```

```
kubectl create secret tls ${CERT_NAME} --key ${KEY_FILE} --cert ${CERT_FILE}
```

Подключаем в Ingress

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: hostname-ru
  namespace: default
spec:
  acme:
    config:
    - domains:
      - hostname.ru
      - www.hostname.ru
      http01:
        ingress: ""
        ingressClass: nginx
  secretName: hostname-ru-tls
  commonName: hostname.ru
  dnsNames:
  - hostname.ru
  - www.hostname.ru
```

```
issuerRef:
    name: letsencrypt
    kind: ClusterIssuer
```

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name:  tls-ingress
  annotations:
    kubernetes.io/tls-acme:"true"
```

ИЛИ

```
    certmanager.k8s.io/cluster-
issuer:  letsencrypt
```

Спасибо!

СЛЁРМ

slurm.io

Southbridge

southbridge.io