**Kubernetes Multi-Service Architecture: Comprehensive Guide**

**1. Architecture Overview**

**System Design**

[Client]

│

▼

[LoadBalancer Service (k8s-web-to-nginx)]

│

▼

[Web Deployment (Node.js)]

│

▼

[ClusterIP Service (nginx)]

│

▼

[Nginx Deployment]

**Key Components:**

1. **Web Deployment** (Node.js/Express)
   * Externally accessible via LoadBalancer
   * Makes internal requests to Nginx service
   * 3 replicas
2. **Nginx Deployment**
   * Internal only (ClusterIP service)
   * 5 replicas
   * Serves as backend for web deployment

**2. Configuration Files**

**Combined YAML Approach**

Use --- separator to combine multiple resource definitions in one file:

# k8s-web-to-nginx.yaml

apiVersion: v1

kind: Service

metadata:

name: k8s-web-to-nginx

spec:

type: LoadBalancer

selector:

app: k8s-web-to-nginx

ports:

- protocol: TCP

port: 3030

targetPort: 3000

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: k8s-web-to-nginx

spec:

replicas: 3

selector:

matchLabels:

app: k8s-web-to-nginx

template:

metadata:

labels:

app: k8s-web-to-nginx

spec:

containers:

- name: k8s-web-to-nginx

image: <your-dockerhub-username>/k8s-web-to-nginx

ports:

- containerPort: 3000

**Nginx Service Configuration**

# nginx.yaml

apiVersion: v1

kind: Service

metadata:

name: nginx

spec:

selector:

app: nginx

ports:

- protocol: TCP

port: 80

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx

spec:

replicas: 5

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx

ports:

- containerPort: 80

**3. Key Concepts**

**Service Discovery**

* Kubernetes provides DNS-based service discovery
* Services can be accessed by <service-name>.<namespace>.svc.cluster.local
* Same-namespace services can be accessed by just <service-name>

**Network Communication**

// In Node.js application

app.get('/nginx', async (req, res) => {

const response = await fetch('http://nginx');

const body = await response.text();

res.send(body);

});

**Service Types**

1. **LoadBalancer**
   * External access
   * Cloud provider integration
   * In Minikube, behaves like NodePort
2. **ClusterIP**
   * Internal-only service
   * Stable IP address within cluster
   * Default service type

**4. Deployment Process**

**Step-by-Step:**

1. Build and push custom web image:
2. docker build -t <your-dockerhub-username>/k8s-web-to-nginx .
3. docker push <your-dockerhub-username>/k8s-web-to-nginx
4. Apply configurations:
5. kubectl apply -f k8s-web-to-nginx.yaml
6. kubectl apply -f nginx.yaml
7. Verify deployment:
8. kubectl get deployments
9. kubectl get services
10. kubectl get pods
11. Access the application:
12. minikube service k8s-web-to-nginx

**5. Inter-Service Communication**

**DNS Resolution**

* Kubernetes creates DNS records for services
* nginx resolves to ClusterIP of nginx service
* Works across namespaces with FQDN

**Connection Flow:**

1. Client → LoadBalancer (port 3030)
2. Web Pod → ClusterIP (port 80)
3. Nginx Pod responds

**6. Scaling Considerations**

**Web Deployment:**

* 3 replicas for load distribution
* Stateless - can scale horizontally

**Nginx Deployment:**

* 5 replicas for high availability
* Stateless - easy to scale

**7. Best Practices**

1. **Service Naming**:
   * Use consistent, descriptive names
   * Match labels between deployments and services
2. **Port Management**:
   * Standardize port numbers
   * Document port mappings
3. **Resource Separation**:
   * Consider namespaces for different environments
   * Use NetworkPolicies for security
4. **Health Checks**:
   * Add readiness/liveness probes
   * Implement circuit breakers for inter-service calls

**8. Troubleshooting**

**Common Issues:**

1. **Connection Failures**:
   * Verify service names match exactly
   * Check pod labels match service selectors
2. **Port Mismatches**:
   * Confirm containerPort matches targetPort
   * Verify protocol (TCP/UDP)
3. **DNS Resolution**:
4. kubectl run -it --rm --image=busybox:1.28 test -- nslookup nginx
5. **Log Inspection**:
6. kubectl logs <web-pod>
7. kubectl logs <nginx-pod>

**9. Advanced Configuration**

**Adding Probes:**

livenessProbe:

httpGet:

path: /

port: 3000

initialDelaySeconds: 15

periodSeconds: 20

**Resource Limits:**

resources:

limits:

cpu: "500m"

memory: "500Mi"

requests:

cpu: "200m"

memory: "200Mi"

**10. Cleanup**

kubectl delete -f k8s-web-to-nginx.yaml

kubectl delete -f nginx.yaml

This architecture demonstrates a common Kubernetes pattern of frontend/backend services communicating through ClusterIP, providing a foundation for more complex microservices deployments.

# Kubernetes Multi-Service Architecture: Comprehensive Guide

## 1. Architecture Overview

### System Design

```

[Client]

│

▼

[LoadBalancer Service (k8s-web-to-nginx)]

│

▼

[Web Deployment (Node.js)]

│

▼

[ClusterIP Service (nginx)]

│

▼

[Nginx Deployment]

```

### Key Components:

1. \*\*Web Deployment\*\* (Node.js/Express)

- Externally accessible via LoadBalancer

- Makes internal requests to Nginx service

- 3 replicas

2. \*\*Nginx Deployment\*\*

- Internal only (ClusterIP service)

- 5 replicas

- Serves as backend for web deployment

## 2. Configuration Files

### Combined YAML Approach

Use `---` separator to combine multiple resource definitions in one file:

```yaml

# k8s-web-to-nginx.yaml

apiVersion: v1

kind: Service

metadata:

name: k8s-web-to-nginx

spec:

type: LoadBalancer

selector:

app: k8s-web-to-nginx

ports:

- protocol: TCP

port: 3030

targetPort: 3000

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: k8s-web-to-nginx

spec:

replicas: 3

selector:

matchLabels:

app: k8s-web-to-nginx

template:

metadata:

labels:

app: k8s-web-to-nginx

spec:

containers:

- name: k8s-web-to-nginx

image: <your-dockerhub-username>/k8s-web-to-nginx

ports:

- containerPort: 3000

```

### Nginx Service Configuration

```yaml

# nginx.yaml

apiVersion: v1

kind: Service

metadata:

name: nginx

spec:

selector:

app: nginx

ports:

- protocol: TCP

port: 80

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx

spec:

replicas: 5

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx

ports:

- containerPort: 80

```

## 3. Key Concepts

### Service Discovery

- Kubernetes provides DNS-based service discovery

- Services can be accessed by `<service-name>.<namespace>.svc.cluster.local`

- Same-namespace services can be accessed by just `<service-name>`

### Network Communication

```javascript

// In Node.js application

app.get('/nginx', async (req, res) => {

const response = await fetch('http://nginx');

const body = await response.text();

res.send(body);

});

```

### Service Types

1. \*\*LoadBalancer\*\*

- External access

- Cloud provider integration

- In Minikube, behaves like NodePort

2. \*\*ClusterIP\*\*

- Internal-only service

- Stable IP address within cluster

- Default service type

## 4. Deployment Process

### Step-by-Step:

1. Build and push custom web image:

```bash

docker build -t <your-dockerhub-username>/k8s-web-to-nginx .

docker push <your-dockerhub-username>/k8s-web-to-nginx

```

2. Apply configurations:

```bash

kubectl apply -f k8s-web-to-nginx.yaml

kubectl apply -f nginx.yaml

```

3. Verify deployment:

```bash

kubectl get deployments

kubectl get services

kubectl get pods

```

4. Access the application:

```bash

minikube service k8s-web-to-nginx

```

## 5. Inter-Service Communication

### DNS Resolution

- Kubernetes creates DNS records for services

- `nginx` resolves to ClusterIP of nginx service

- Works across namespaces with FQDN

### Connection Flow:

1. Client → LoadBalancer (port 3030)

2. Web Pod → ClusterIP (port 80)

3. Nginx Pod responds

## 6. Scaling Considerations

### Web Deployment:

- 3 replicas for load distribution

- Stateless - can scale horizontally

### Nginx Deployment:

- 5 replicas for high availability

- Stateless - easy to scale

## 7. Best Practices

1. \*\*Service Naming\*\*:

- Use consistent, descriptive names

- Match labels between deployments and services

2. \*\*Port Management\*\*:

- Standardize port numbers

- Document port mappings

3. \*\*Resource Separation\*\*:

- Consider namespaces for different environments

- Use NetworkPolicies for security

4. \*\*Health Checks\*\*:

- Add readiness/liveness probes

- Implement circuit breakers for inter-service calls

## 8. Troubleshooting

### Common Issues:

1. \*\*Connection Failures\*\*:

- Verify service names match exactly

- Check pod labels match service selectors

2. \*\*Port Mismatches\*\*:

- Confirm containerPort matches targetPort

- Verify protocol (TCP/UDP)

3. \*\*DNS Resolution\*\*:

```bash

kubectl run -it --rm --image=busybox:1.28 test -- nslookup nginx

```

4. \*\*Log Inspection\*\*:

```bash

kubectl logs <web-pod>

kubectl logs <nginx-pod>

```

## 9. Advanced Configuration

### Adding Probes:

```yaml

livenessProbe:

httpGet:

path: /

port: 3000

initialDelaySeconds: 15

periodSeconds: 20

```

### Resource Limits:

```yaml

resources:

limits:

cpu: "500m"

memory: "500Mi"

requests:

cpu: "200m"

memory: "200Mi"

```

## 10. Cleanup

```bash

kubectl delete -f k8s-web-to-nginx.yaml

kubectl delete -f nginx.yaml

```

This architecture demonstrates a common Kubernetes pattern of frontend/backend services communicating through ClusterIP, providing a foundation for more complex microservices deployments.