**Kubernetes Inter-Service Communication: Deep Dive**

**1. Service-to-Service Communication Architecture**

**System Components:**

[Client] → [LoadBalancer Service (k8s-web-to-nginx:3030)]

→ [Web Pod (Node.js)]

→ [ClusterIP Service (nginx:80)]

→ [Nginx Pod]

**Key Observations:**

* Web service exposes two endpoints:
  + / - Returns "Hello from web server"
  + /nginx - Proxies request to nginx service
* Nginx service is internal-only (ClusterIP)
* DNS resolution enables service discovery

**2. DNS Resolution in Kubernetes**

**CoreDNS Service:**

* Kubernetes' internal DNS server
* Automatically creates DNS records for Services
* Format: <service-name>.<namespace>.svc.cluster.local

**Verification Commands:**

# Check DNS resolution from inside a pod

kubectl exec <web-pod> -- nslookup nginx

# Expected output:

Server: 10.96.0.10

Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local

Name: nginx

Address 1: 10.96.123.45 nginx.default.svc.cluster.local

**DNS Configuration Files:**

# View CoreDNS config

kubectl -n kube-system get configmap coredns -o yaml

**3. Network Traffic Flow**

**Request Journey:**

1. Client → minikube service k8s-web-to-nginx (LoadBalancer)
2. Web Pod receives request at /nginx
3. Web Pod makes HTTP request to http://nginx
4. DNS resolves nginx to ClusterIP (e.g., 10.96.123.45)
5. Request routed to one of 5 nginx pods
6. Response travels back through chain

**Network Visualization:**

Client → NodePort → Service → Web Pod → ClusterIP → Nginx Pod

**4. Practical Verification Methods**

**Testing Connectivity:**

# From inside web pod:

kubectl exec <web-pod> -- curl http://nginx

# Expected output: Nginx welcome page

**Port Forwarding for Debugging:**

# Access nginx directly (bypass web service)

kubectl port-forward svc/nginx 8080:80

# Then visit http://localhost:8080

**5. Key Configuration Details**

**Web Service YAML Highlights:**

# k8s-web-to-nginx.yaml

apiVersion: v1

kind: Service

metadata:

name: k8s-web-to-nginx # Used for DNS resolution

spec:

type: LoadBalancer

ports:

- port: 3030 # External port

targetPort: 3000 # Container port

**Nginx Service YAML Highlights:**

# nginx.yaml

apiVersion: v1

kind: Service

metadata:

name: nginx # DNS name used in web app

spec:

type: ClusterIP # Internal only

ports:

- port: 80 # Service port

targetPort: 80 # Container port

**6. Scaling Considerations**

**Web Deployment:**

* 3 replicas for load distribution
* Each can independently call nginx service

**Nginx Deployment:**

* 5 replicas for high availability
* Service load-balances across all pods

**Verification:**

# Watch how requests distribute

kubectl get pods -o wide

for i in {1..10}; do curl <minikube-ip>:<node-port>/nginx; done

**7. Advanced Networking Concepts**

**Service Mesh Potential:**

* Could add Istio/Linkerd for:
  + Circuit breaking
  + Retries
  + Timeouts
  + Metrics

**Network Policies:**

# Restrict nginx access to only web pods

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: allow-web-to-nginx

spec:

podSelector:

matchLabels:

app: nginx

ingress:

- from:

- podSelector:

matchLabels:

app: k8s-web-to-nginx

**8. Troubleshooting Guide**

**Common Issues:**

1. **DNS Resolution Fails**:
   * Check CoreDNS pods are running
   * Verify service exists in same namespace
2. **Connection Refused**:
   * Verify targetPort matches container port
   * Check pods are running and ready
3. **Wrong Response**:
   * Verify service selectors match pod labels
   * Check endpoint list: kubectl get endpoints

**Diagnostic Commands:**

# Check service endpoints

kubectl get endpoints nginx

# Check pod connectivity

kubectl run -it --rm --image=alpine test -- sh

> apk add curl

> curl http://nginx

**9. Best Practices**

1. **Service Naming**:
   * Use DNS-compliant names (lowercase, hyphens)
   * Be consistent across environments
2. **Port Standards**:
   * Document all port mappings
   * Use standard ports where possible (80, 443)
3. **Connection Handling**:
   * Implement retries with exponential backoff
   * Set reasonable timeouts
4. **Observability**:
   * Add logging for inter-service calls
   * Monitor success/failure rates

**10. Cleanup Procedure**

# Delete all resources

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

kubectl delete -f nginx.yaml

# Verify cleanup

kubectl get all

This architecture demonstrates Kubernetes' powerful service discovery and networking capabilities, enabling microservices to communicate seamlessly while maintaining loose coupling and scalability.

# Kubernetes Inter-Service Communication: Deep Dive

## 1. Service-to-Service Communication Architecture

### System Components:

```

[Client] → [LoadBalancer Service (k8s-web-to-nginx:3030)]

→ [Web Pod (Node.js)]

→ [ClusterIP Service (nginx:80)]

→ [Nginx Pod]

```

### Key Observations:

- Web service exposes two endpoints:

- `/` - Returns "Hello from web server"

- `/nginx` - Proxies request to nginx service

- Nginx service is internal-only (ClusterIP)

- DNS resolution enables service discovery

## 2. DNS Resolution in Kubernetes

### CoreDNS Service:

- Kubernetes' internal DNS server

- Automatically creates DNS records for Services

- Format: `<service-name>.<namespace>.svc.cluster.local`

### Verification Commands:

```bash

# Check DNS resolution from inside a pod

kubectl exec <web-pod> -- nslookup nginx

# Expected output:

Server: 10.96.0.10

Address 1: 10.96.0.10 kube-dns.kube-system.svc.cluster.local

Name: nginx

Address 1: 10.96.123.45 nginx.default.svc.cluster.local

```

### DNS Configuration Files:

```bash

# View CoreDNS config

kubectl -n kube-system get configmap coredns -o yaml

```

## 3. Network Traffic Flow

### Request Journey:

1. Client → `minikube service k8s-web-to-nginx` (LoadBalancer)

2. Web Pod receives request at `/nginx`

3. Web Pod makes HTTP request to `http://nginx`

4. DNS resolves `nginx` to ClusterIP (e.g., 10.96.123.45)

5. Request routed to one of 5 nginx pods

6. Response travels back through chain

### Network Visualization:

```

Client → NodePort → Service → Web Pod → ClusterIP → Nginx Pod

```

## 4. Practical Verification Methods

### Testing Connectivity:

```bash

# From inside web pod:

kubectl exec <web-pod> -- curl http://nginx

# Expected output: Nginx welcome page

```

### Port Forwarding for Debugging:

```bash

# Access nginx directly (bypass web service)

kubectl port-forward svc/nginx 8080:80

# Then visit http://localhost:8080

```

## 5. Key Configuration Details

### Web Service YAML Highlights:

```yaml

# k8s-web-to-nginx.yaml

apiVersion: v1

kind: Service

metadata:

name: k8s-web-to-nginx # Used for DNS resolution

spec:

type: LoadBalancer

ports:

- port: 3030 # External port

targetPort: 3000 # Container port

```

### Nginx Service YAML Highlights:

```yaml

# nginx.yaml

apiVersion: v1

kind: Service

metadata:

name: nginx # DNS name used in web app

spec:

type: ClusterIP # Internal only

ports:

- port: 80 # Service port

targetPort: 80 # Container port

```

## 6. Scaling Considerations

### Web Deployment:

- 3 replicas for load distribution

- Each can independently call nginx service

### Nginx Deployment:

- 5 replicas for high availability

- Service load-balances across all pods

### Verification:

```bash

# Watch how requests distribute

kubectl get pods -o wide

for i in {1..10}; do curl <minikube-ip>:<node-port>/nginx; done

```

## 7. Advanced Networking Concepts

### Service Mesh Potential:

- Could add Istio/Linkerd for:

- Circuit breaking

- Retries

- Timeouts

- Metrics

### Network Policies:

```yaml

# Restrict nginx access to only web pods

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: allow-web-to-nginx

spec:

podSelector:

matchLabels:

app: nginx

ingress:

- from:

- podSelector:

matchLabels:

app: k8s-web-to-nginx

```

## 8. Troubleshooting Guide

### Common Issues:

1. \*\*DNS Resolution Fails\*\*:

- Check CoreDNS pods are running

- Verify service exists in same namespace

2. \*\*Connection Refused\*\*:

- Verify targetPort matches container port

- Check pods are running and ready

3. \*\*Wrong Response\*\*:

- Verify service selectors match pod labels

- Check endpoint list: `kubectl get endpoints`

### Diagnostic Commands:

```bash

# Check service endpoints

kubectl get endpoints nginx

# Check pod connectivity

kubectl run -it --rm --image=alpine test -- sh

> apk add curl

> curl http://nginx

```

## 9. Best Practices

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

- Use DNS-compliant names (lowercase, hyphens)

- Be consistent across environments

2. \*\*Port Standards\*\*:

- Document all port mappings

- Use standard ports where possible (80, 443)

3. \*\*Connection Handling\*\*:

- Implement retries with exponential backoff

- Set reasonable timeouts

4. \*\*Observability\*\*:

- Add logging for inter-service calls

- Monitor success/failure rates

## 10. Cleanup Procedure

```bash

# Delete all resources

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

kubectl delete -f nginx.yaml

# Verify cleanup

kubectl get all

```

This architecture demonstrates Kubernetes' powerful service discovery and networking capabilities, enabling microservices to communicate seamlessly while maintaining loose coupling and scalability.