**Kubernetes Declarative Configuration: Comprehensive Guide**

**1. Imperative vs Declarative Approaches**

**Imperative Approach**

* Uses direct commands like:
* kubectl create deployment k8s-web-hello --image=<image>
* kubectl expose deployment k8s-web-hello --type=LoadBalancer --port=3000
* Quick for testing but not reproducible
* Hard to version control

**Declarative Approach**

* Uses YAML configuration files
* Applied with:
* kubectl apply -f deployment.yaml
* kubectl apply -f service.yaml
* Reproducible and versionable
* Recommended for production

**2. Deployment Configuration File**

**Sample deployment.yaml:**

apiVersion: apps/v1

kind: Deployment

metadata:

name: k8s-web-hello

spec:

replicas: 5

selector:

matchLabels:

app: k8s-web-hello

template:

metadata:

labels:

app: k8s-web-hello

spec:

containers:

- name: k8s-web-hello

image: <your-dockerhub-username>/k8s-web-hello

ports:

- containerPort: 3000

resources:

limits:

cpu: "500m"

memory: "500Mi"

**Key Sections Explained:**

1. **apiVersion**: API group and version
   * For deployments: apps/v1
2. **kind**: Resource type
   * Deployment, Service, Pod, etc.
3. **metadata**: Identifying information
   * name: Unique name for the resource
   * labels: Key-value pairs for organization
4. **spec**: Desired state
   * replicas: Number of pod instances
   * selector: How the deployment finds pods to manage
   * template: Pod template specification
5. **template.spec**: Pod specification
   * containers: List of containers in the pod
     + image: Container image to use
     + ports: Exposed container ports
     + resources: CPU/memory limits/requests

**3. Service Configuration File**

**Sample service.yaml:**

apiVersion: v1

kind: Service

metadata:

name: k8s-web-hello

spec:

type: LoadBalancer

selector:

app: k8s-web-hello

ports:

- protocol: TCP

port: 3030

targetPort: 3000

**Key Sections Explained:**

1. **apiVersion**:
   * For services: v1
2. **spec.type**: Service type
   * ClusterIP (default)
   * NodePort
   * LoadBalancer
3. **spec.selector**: Matches pods to expose
   * Must match pod labels from deployment
4. **spec.ports**: Port mapping
   * port: Service port
   * targetPort: Container port
   * protocol: TCP or UDP

**4. Applying Configuration Files**

**Basic Commands:**

# Create/update resources

kubectl apply -f deployment.yaml

kubectl apply -f service.yaml

# View resources

kubectl get deployments

kubectl get services

kubectl get pods

# Delete resources

kubectl delete -f deployment.yaml

kubectl delete -f service.yaml

**Scaling with Declarative Approach:**

1. Edit deployment.yaml:
2. spec:
3. replicas: 5
4. Reapply:
5. kubectl apply -f deployment.yaml

**5. Kubernetes Documentation Structure**

**Finding API References:**

1. Visit [kubernetes.io](https://kubernetes.io)
2. Go to Documentation → Reference → Kubernetes API
3. Select resource type (e.g., "Deployment")

**Key Documentation Sections:**

* **apiVersion**: Required API group/version
* **kind**: Resource type
* **metadata**: Name, labels, annotations
* **spec**: Required configuration
* **status**: Read-only fields (managed by Kubernetes)

**6. Best Practices**

1. **Version Control**: Store YAML files in Git
2. **Modularity**: Separate files for different resources
3. **Comments**: Add explanations for complex configurations
4. **Validation**: Use kubectl apply --dry-run=client -f file.yaml
5. **Diff Checking**: Use kubectl diff -f file.yaml

**7. Common Configuration Patterns**

**Environment Variables:**

env:

- name: ENV\_VAR\_NAME

value: "value"

**Liveness/Readiness Probes:**

livenessProbe:

httpGet:

path: /health

port: 3000

initialDelaySeconds: 15

periodSeconds: 20

**Volume Mounts:**

volumeMounts:

- name: config-volume

mountPath: /etc/config

volumes:

- name: config-volume

configMap:

name: app-config

**8. Troubleshooting**

**Common Issues:**

1. **Selector Mismatch**: Service selector must match pod labels
2. **Port Conflicts**: Verify port mappings
3. **Image Pull Errors**: Check image name/tag
4. **Resource Limits**: Adjust CPU/memory if pods crash

**Debugging Commands:**

# View detailed resource info

kubectl describe deployment/k8s-web-hello

# Check pod logs

kubectl logs <pod-name>

# View events

kubectl get events

**9. Advanced Topics**

**Kustomize for Environment-Specific Configs:**

kubectl apply -k overlays/production

**Helm Charts:**

Package multiple Kubernetes resources into a single deployable unit

**Operators:**

Custom controllers for complex applications

This declarative approach provides a robust, maintainable way to manage Kubernetes resources, especially in production environments where reproducibility and version control are essential.

# Kubernetes Declarative Configuration: Comprehensive Guide

## 1. Imperative vs Declarative Approaches

### Imperative Approach

- Uses direct commands like:

```bash

kubectl create deployment k8s-web-hello --image=<image>

kubectl expose deployment k8s-web-hello --type=LoadBalancer --port=3000

```

- Quick for testing but not reproducible

- Hard to version control

### Declarative Approach

- Uses YAML configuration files

- Applied with:

```bash

kubectl apply -f deployment.yaml

kubectl apply -f service.yaml

```

- Reproducible and versionable

- Recommended for production

## 2. Deployment Configuration File

### Sample deployment.yaml:

```yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: k8s-web-hello

spec:

replicas: 5

selector:

matchLabels:

app: k8s-web-hello

template:

metadata:

labels:

app: k8s-web-hello

spec:

containers:

- name: k8s-web-hello

image: <your-dockerhub-username>/k8s-web-hello

ports:

- containerPort: 3000

resources:

limits:

cpu: "500m"

memory: "500Mi"

```

### Key Sections Explained:

1. \*\*apiVersion\*\*: API group and version

- For deployments: `apps/v1`

2. \*\*kind\*\*: Resource type

- `Deployment`, `Service`, `Pod`, etc.

3. \*\*metadata\*\*: Identifying information

- `name`: Unique name for the resource

- `labels`: Key-value pairs for organization

4. \*\*spec\*\*: Desired state

- `replicas`: Number of pod instances

- `selector`: How the deployment finds pods to manage

- `template`: Pod template specification

5. \*\*template.spec\*\*: Pod specification

- `containers`: List of containers in the pod

- `image`: Container image to use

- `ports`: Exposed container ports

- `resources`: CPU/memory limits/requests

## 3. Service Configuration File

### Sample service.yaml:

```yaml

apiVersion: v1

kind: Service

metadata:

name: k8s-web-hello

spec:

type: LoadBalancer

selector:

app: k8s-web-hello

ports:

- protocol: TCP

port: 3030

targetPort: 3000

```

### Key Sections Explained:

1. \*\*apiVersion\*\*:

- For services: `v1`

2. \*\*spec.type\*\*: Service type

- `ClusterIP` (default)

- `NodePort`

- `LoadBalancer`

3. \*\*spec.selector\*\*: Matches pods to expose

- Must match pod labels from deployment

4. \*\*spec.ports\*\*: Port mapping

- `port`: Service port

- `targetPort`: Container port

- `protocol`: TCP or UDP

## 4. Applying Configuration Files

### Basic Commands:

```bash

# Create/update resources

kubectl apply -f deployment.yaml

kubectl apply -f service.yaml

# View resources

kubectl get deployments

kubectl get services

kubectl get pods

# Delete resources

kubectl delete -f deployment.yaml

kubectl delete -f service.yaml

```

### Scaling with Declarative Approach:

1. Edit deployment.yaml:

```yaml

spec:

replicas: 5

```

2. Reapply:

```bash

kubectl apply -f deployment.yaml

```

## 5. Kubernetes Documentation Structure

### Finding API References:

1. Visit [kubernetes.io](https://kubernetes.io)

2. Go to Documentation → Reference → Kubernetes API

3. Select resource type (e.g., "Deployment")

### Key Documentation Sections:

- \*\*apiVersion\*\*: Required API group/version

- \*\*kind\*\*: Resource type

- \*\*metadata\*\*: Name, labels, annotations

- \*\*spec\*\*: Required configuration

- \*\*status\*\*: Read-only fields (managed by Kubernetes)

## 6. Best Practices

1. \*\*Version Control\*\*: Store YAML files in Git

2. \*\*Modularity\*\*: Separate files for different resources

3. \*\*Comments\*\*: Add explanations for complex configurations

4. \*\*Validation\*\*: Use `kubectl apply --dry-run=client -f file.yaml`

5. \*\*Diff Checking\*\*: Use `kubectl diff -f file.yaml`

## 7. Common Configuration Patterns

### Environment Variables:

```yaml

env:

- name: ENV\_VAR\_NAME

value: "value"

```

### Liveness/Readiness Probes:

```yaml

livenessProbe:

httpGet:

path: /health

port: 3000

initialDelaySeconds: 15

periodSeconds: 20

```

### Volume Mounts:

```yaml

volumeMounts:

- name: config-volume

mountPath: /etc/config

volumes:

- name: config-volume

configMap:

name: app-config

```

## 8. Troubleshooting

### Common Issues:

1. \*\*Selector Mismatch\*\*: Service selector must match pod labels

2. \*\*Port Conflicts\*\*: Verify port mappings

3. \*\*Image Pull Errors\*\*: Check image name/tag

4. \*\*Resource Limits\*\*: Adjust CPU/memory if pods crash

### Debugging Commands:

```bash

# View detailed resource info

kubectl describe deployment/k8s-web-hello

# Check pod logs

kubectl logs <pod-name>

# View events

kubectl get events

```

## 9. Advanced Topics

### Kustomize for Environment-Specific Configs:

```bash

kubectl apply -k overlays/production

```

### Helm Charts:

Package multiple Kubernetes resources into a single deployable unit

### Operators:

Custom controllers for complex applications

This declarative approach provides a robust, maintainable way to manage Kubernetes resources, especially in production environments where reproducibility and version control are essential.