Pour répondre à votre demande, voici un exemple fonctionnel d'un chart Helm principal contenant un serveur web **Nginx**, et d'un subchart gérant un serveur **PHP**. Les deux partagent un volume persistant pour stocker les fichiers PHP. Ils communiquent via un service exposé.

Je vais structurer ce projet en deux parties : le chart principal appelé web et le subchart php.

Structure du projet

Voici une vue d'ensemble de la structure des fichiers et dossiers :

```
web/
├── charts/
│   └── php/ (subchart)
├── templates/
│   └── deployment.yaml (Nginx)
│   └── service.yaml (Nginx)
├── values.yaml
└── Chart.yaml
```

Le subchart php contiendra également ses propres fichiers deployment.yaml et service.yaml.

Étape 1 : Chart principal (web)

Fichier web/Chart.yaml

```yaml
apiVersion: v2
name: web
description: Chart principal pour le serveur web Nginx
type: application
version: 0.1.0
appVersion: "1.0.0"
dependencies:
  - name: php
    version: 0.1.0
    repository: "file://charts/php"
```

Fichier web/values.yaml

replicaCount: 1

```yaml
service:
  type: NodePort
  port: 80
  nodePort: 30000

php:
  replicaCount: 1
  service:
    port: 9000
```

```
sharedVolume:
  storageSize: 1Gi
```

Fichier web/templates/deployment.yaml (Nginx)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Release.Name }}-nginx
spec:
  replicas: {{ .Values.replicaCount }}
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
        volumeMounts:
        - mountPath: /usr/share/nginx/html
          name: shared-volume
      volumes:
      - name: shared-volume
        persistentVolumeClaim:
          claimName: {{ .Release.Name }}-shared-pvc
```

Fichier web/templates/service.yaml (Nginx)

```
apiVersion: v1
kind: Service
metadata:
  name: {{ .Release.Name }}-nginx-service
spec:
  type: {{ .Values.service.type }}
  ports:
  - port: {{ .Values.service.port }}
    targetPort: 80
    nodePort: {{ .Values.service.nodePort }}
  selector:
    app: nginx
```

Fichier web/templates/pvc.yaml (Volume partagé)

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: {{ .Release.Name }}-shared-pvc
spec:
  accessModes:
  - ReadWriteMany
  resources:
```

```
    requests:
      storage: {{ .Values.sharedVolume.storageSize }}
```

Étape 2 : Subchart (php)

Fichier web/charts/php/Chart.yaml

```
apiVersion: v2
name: php
description: Subchart pour le serveur PHP
type: application
version: 0.1.0
appVersion: "1.0.0"
```

Fichier web/charts/php/values.yaml

```
replicaCount: 1

service:
  port: 9000
```

Fichier web/charts/php/templates/deployment.yaml (PHP)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Release.Name }}-php
spec:
  replicas: {{ .Values.replicaCount }}
  selector:
    matchLabels:
      app: php
  template:
    metadata:
      labels:
        app: php
    spec:
      containers:
      - name: php
        image: php:8.1-fpm
        ports:
        - containerPort: 9000
        volumeMounts:
        - mountPath: /var/www/html
          name: shared-volume
      volumes:
      - name: shared-volume
        persistentVolumeClaim:
          claimName: {{ .Release.Name }}-shared-pvc
```

Fichier web/charts/php/templates/service.yaml (PHP)

```
apiVersion: v1
kind: Service
metadata:
```

```
  name: {{ .Release.Name }}-php-service
spec:
  type: ClusterIP
  ports:
  - port: {{ .Values.service.port }}
    targetPort: 9000
  selector:
    app: php
```

Étape 3 : Déploiement

1. Créez le chart principal :

```
helm create web
```

2. Ajoutez le subchart dans web/charts :

```
helm create web/charts/php
```

3. Déployez le chart principal :

```
helm install web-release ./web
```

4. Vérifiez les ressources :

```
kubectl get all
```

5. Testez l'application en accédant au NodePort :

```
http://<NodeIP>:30000
```