Advanced Kubernetes (K8s) topics, focusing on deployments, scaling, and Helm charts. These are crucial for managing and orchestrating containerized applications in a Kubernetes environment.

1. Advanced Kubernetes Deployments

Deployments in Kubernetes manage the lifecycle of applications, including updates and rollbacks. Here are some advanced deployment features:

1.1. Rolling Updates

Rolling updates allow you to update your application with zero downtime by incrementally replacing old Pods with new ones.

Example:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp
        image: myapp:latest
        ports:
        - containerPort: 80
```

To update the image:

```
kubectl set image deployment/myapp myapp=myapp:v2
```

1.2. Blue-Green Deployments

Blue-green deployments involve maintaining two separate environments (blue and green). You switch traffic between these environments to deploy updates with minimal downtime.

Implementation Steps:

1. **Create Two Deployments:** One for each environment.

```
# Blue deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-blue
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
      color: blue
  template:
    metadata:
      labels:
        app: myapp
        color: blue
    spec:
      containers:
      - name: myapp
        image: myapp:blue

# Green deployment
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
  name: myapp-green
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
      color: green
  template:
    metadata:
      labels:
        app: myapp
        color: green
    spec:
      containers:
      - name: myapp
        image: myapp:green
```

2. **Update Service:** Point to the green deployment once it's tested.

```
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
spec:
  selector:
    app: myapp
    color: green
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

1.3. Canary Deployments

Canary deployments roll out updates to a small subset of users before a full deployment. This allows you to test new features and catch issues early.

Implementation Steps:

1. **Create Initial Deployment:**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp
        image: myapp:1.0
```

Deploy Canary Version:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-canary
spec:
  replicas: 1
  selector:
    matchLabels:
      app: myapp
      version: canary
  template:
```

```
  metadata:
    labels:
      app: myapp
      version: canary
  spec:
    containers:
    - name: myapp
      image: myapp:1.1
```

2. **Update Service to Route Traffic:**
   Use a service with labels to direct a percentage of traffic to the canary deployment.

```
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
spec:
  selector:
    app: myapp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

2. Scaling in Kubernetes

Scaling applications in Kubernetes can be achieved manually or automatically. Here's how you can manage scaling:

2.1. Manual Scaling

You can manually scale deployments up or down using `kubectl`.

Example:

```
kubectl scale deployment myapp --replicas=5
```

2.2. Horizontal Pod Autoscaler (HPA)

The HPA automatically scales the number of Pods based on observed CPU utilization or other custom metrics.

Implementation Steps:

1. **Create an HPA Resource**

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: myapp-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: myapp
  minReplicas: 2
  maxReplicas: 10
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 50
```

2. **Monitor Scaling:**
   Use `kubectl get hpa` to check the status of the HPA.

```
kubectl get hpa
```

2.3. Vertical Pod Autoscaler (VPA)

The VPA adjusts the CPU and memory requests for Pods based on their usage.

Implementation Steps:

1. **Create a VPA Resource:**

```
apiVersion: "container.autoscaling.k8s.io/v1alpha1"
kind: VerticalPodAutoscaler
metadata:
  name: myapp-vpa
spec:
  targetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: myapp
  updatePolicy:
    updateMode: "Auto"
```

2. **Monitor VPA Recommendations:**
   Check the recommended changes.
   `kubectl describe vpa myapp-vpa`

3. Helm Charts

Helm is a package manager for Kubernetes that simplifies deploying and managing applications.

3.1. Helm Chart Basics

- **Chart**: A package of pre-configured Kubernetes resources.
- **Values File**: Customizable settings for your chart.
- **Release**: An instance of a chart running in a Kubernetes cluster.

3.2. Creating a Helm Chart

1. **Initialize a New Chart:**
   `helm create mychart`

   This creates a directory structure with templates and default configurations.

2. **Edit `values.yaml`:**
   Customize the default values for your chart.
   ```
   replicaCount: 2

   image:
     repository: myapp
     tag: "latest"
   ```

3. **Update Templates:**
   Modify the templates under the `templates/` directory to match your application requirements.
4. **Install the Chart:**
   Deploy the chart to your Kubernetes cluster.
   ```
   helm install myrelease ./mychart
   ```
5. **Upgrade the Chart:**
   Update an existing release with new changes.
   ```
   helm upgrade myrelease ./mychart
   ```
6. **Rollback Changes:**
   Revert to a previous release version if needed.
   ```
   helm rollback myrelease 1
   ```
7. **List and Manage Releases:**
   ```
   helm list
   ```

3.3. Helm Chart Repositories

- **Add a Repository:**
  ```
  helm repo add stable https://charts.helm.sh/stable
  ```
- **Search for Charts:**
  ```
  helm search repo stable
  ```
- **Update Repositories:**
  ```
  helm repo update
  ```

By mastering these advanced Kubernetes features, you can efficiently manage and scale your applications while utilizing Helm for simplified deployments and management. If you need further details or specific examples, feel free to ask!