Detailed Handout: Helm – Kubernetes Package Manager

# 1. Introduction to Helm

Helm is the package manager for Kubernetes. It simplifies the deployment and management of applications by packaging Kubernetes manifests into reusable, versioned units called Charts.  
  
Why Helm?  
- Kubernetes applications often require multiple YAML files (Deployment, Service, ConfigMap, etc.).  
- Managing these files manually can be repetitive and error-prone.  
- Helm allows you to:  
 - Package YAML manifests into a Chart  
 - Deploy with a single command  
 - Manage versions, rollbacks, and upgrades  
  
Think of Helm as the “apt” or “yum” for Kubernetes.

# 2. Helm Core Concepts

- Chart → A package of Kubernetes manifests (like an app installer).  
- Release → An instance of a Chart deployed to a Kubernetes cluster.  
- Repository → A storage location for Charts (like Docker Hub for Helm Charts).  
- Values.yaml → A configuration file that customizes a chart without editing manifests.

# 3. Installing Helm

On Linux/Mac  
curl https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 | bash  
  
On Windows (Chocolatey)  
choco install kubernetes-helm  
  
Verify Installation  
helm version

# 4. Basic Helm Workflow

1. Search for a Chart  
 helm search repo nginx  
  
2. Install a Chart  
 helm install my-nginx bitnami/nginx  
  
3. List Installed Releases  
 helm list  
  
4. Upgrade a Release  
 helm upgrade my-nginx bitnami/nginx --set service.type=LoadBalancer  
  
5. Rollback a Release  
 helm rollback my-nginx 1  
  
6. Uninstall a Release  
 helm uninstall my-nginx

# 5. Creating Your First Helm Chart

1. Create a new chart:  
 helm create mychart  
  
2. Customize values.yaml:  
 replicaCount: 2  
 image:  
 repository: nginx  
 tag: "1.21"  
 service:  
 type: ClusterIP  
 port: 80  
  
3. Deploy the chart:  
 helm install myapp ./mychart  
  
4. Verify:  
 kubectl get pods,svc

# 6. Understanding values.yaml

- Default configuration for a chart.  
- Allows parameterization.  
  
Example:  
replicaCount: 3  
image:  
 repository: myapp  
 tag: "1.0.0"  
  
Override values at install:  
helm install myapp ./mychart --set replicaCount=5  
  
Override with a file:  
helm install myapp ./mychart -f custom-values.yaml

# 7. Helm Repositories

- Add a repo:  
 helm repo add bitnami https://charts.bitnami.com/bitnami  
  
- Update repos:  
 helm repo update  
  
- Search charts:  
 helm search repo mysql

# 8. Intermediate Features

Templating  
Helm uses Go templates. Example in deployment.yaml:  
spec:  
 replicas: {{ .Values.replicaCount }}  
 containers:  
 - name: {{ .Chart.Name }}  
 image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"  
  
Hooks  
- Pre-install, Post-install, Pre-upgrade, Post-upgrade events.  
  
Dependencies  
Defined in Chart.yaml:  
dependencies:  
 - name: redis  
 version: 14.8.8  
 repository: https://charts.bitnami.com/bitnami  
  
Chart Lifecycle Management  
- helm history → Shows release history.  
- helm rollback → Reverts to a stable version.

# 9. Best Practices

- Use semantic versioning for charts.  
- Keep application code and charts in separate repos.  
- Always define default values in values.yaml.  
- Use linting before deploying:  
 helm lint ./mychart  
- Test charts in staging before production.

# 10. Hands-On Lab Exercises

Lab 1: Deploy NGINX using Helm  
1. Add Bitnami repo:  
 helm repo add bitnami https://charts.bitnami.com/bitnami  
2. Install NGINX:  
 helm install webserver bitnami/nginx  
3. Verify pod and service creation.  
  
Lab 2: Customize Deployment  
1. Create custom-values.yaml:  
 replicaCount: 3  
 service:  
 type: LoadBalancer  
2. Install with custom values:  
 helm install custom-nginx bitnami/nginx -f custom-values.yaml  
  
Lab 3: Create Your Own Chart  
1. Run helm create mychart.  
2. Modify values.yaml to use nginx.  
3. Deploy:  
 helm install myapp ./mychart  
4. Upgrade with new replica count.

# 11. Summary

- Helm simplifies application packaging and deployment in Kubernetes.  
- Charts → Package; Releases → Deployed instances.  
- Supports versioning, rollback, and templating.  
- Helps standardize and accelerate Kubernetes application management.  
  
By mastering Helm basics and intermediate features, teams can reduce deployment complexity, improve consistency, and enable scalable GitOps workflows.