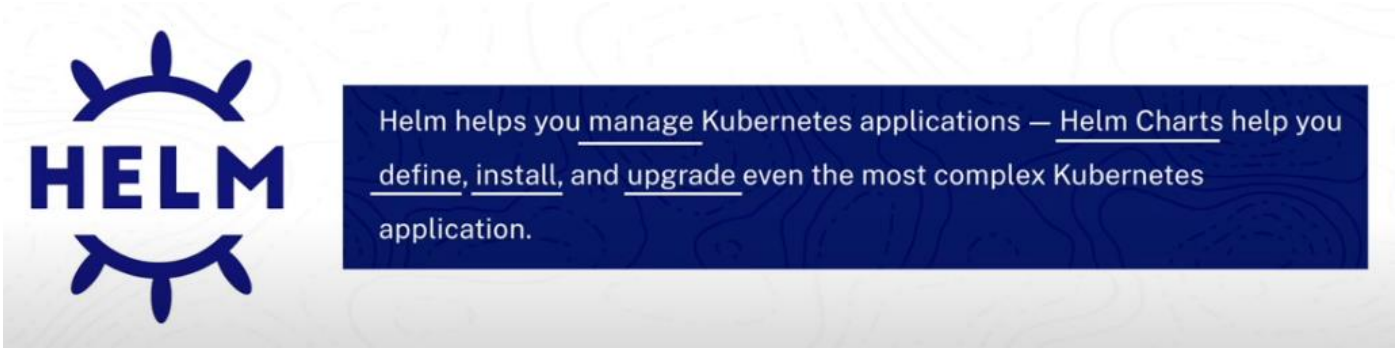
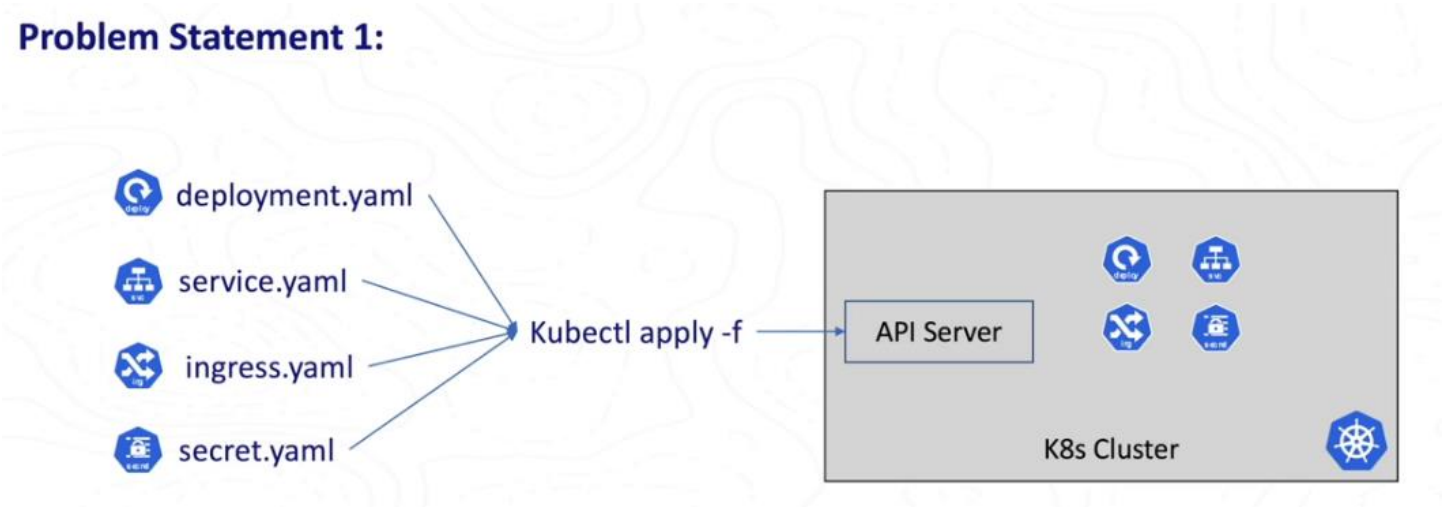
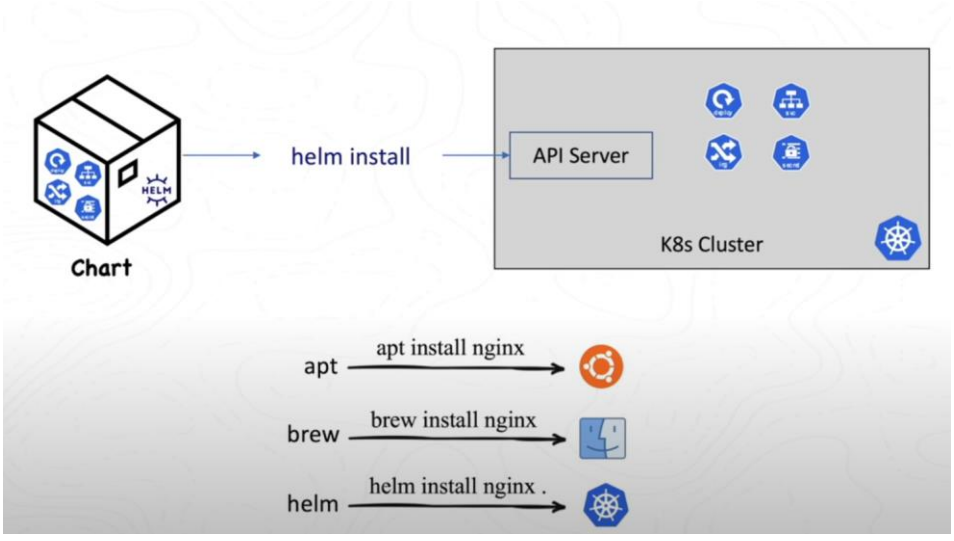
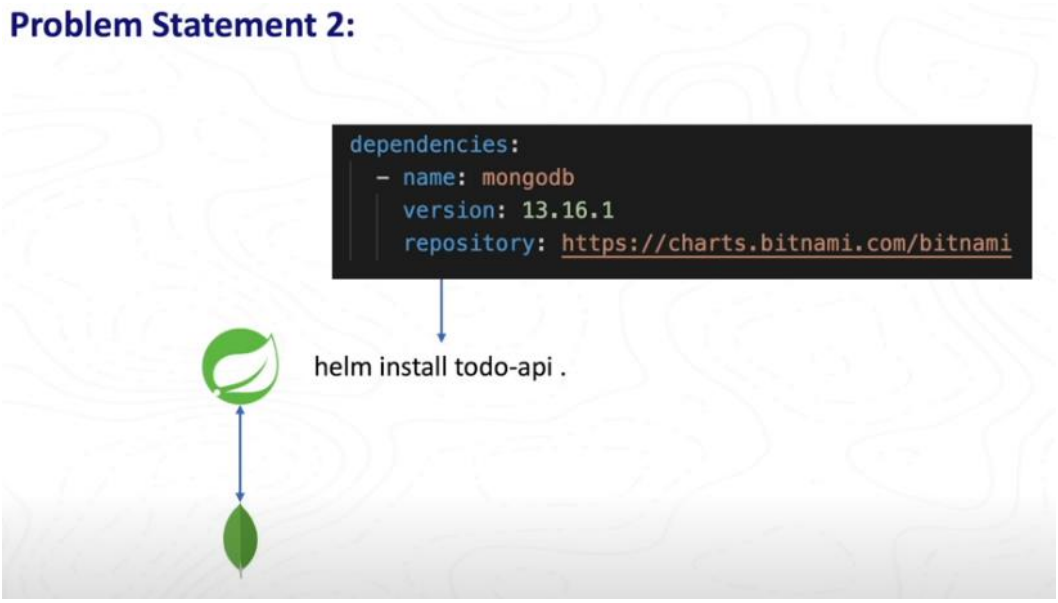
Helm





The traditional approach to deploying applications in Kubernetes requires multiple manifest files, such as Deployment, Service, Ingress, and Secrets. Managing and applying these files individually using **kubectl apply -f** becomes tedious and complex as the number of resources grows. Helm simplifies this by packaging all manifest files into a single bundle called a **Helm Chart**, allowing deployments with just one command—similar to how package managers like apt or brew work. While manually grouping manifests in a directory is possible, Helm offers additional benefits beyond just organization, which will be explored further.



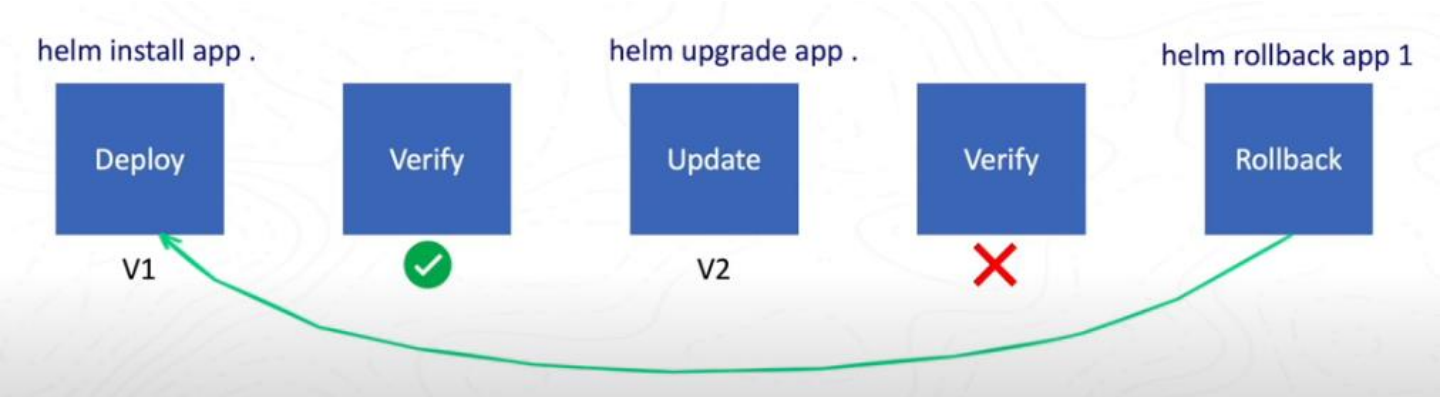


If our Spring Boot application depends on MongoDB, deploying it on a fresh Kubernetes cluster would typically require setting up MongoDB first, followed by the Spring Boot application. However, Helm simplifies dependency management by automatically handling application dependencies. We only need to specify that our Spring Boot application depends on MongoDB, and Helm ensures MongoDB is deployed first before deploying the main application, making the process seamless and hassle-free.

Another challenge with plain Kubernetes manifests is managing environment-specific configurations. For example, a hardcoded MongoDB URI must be modified for different environments like **Dev, QA, and Prod**, requiring separate manifest copies for each. This approach is tedious and error-prone. **Helm templates** solve this problem by replacing hardcoded values with **placeholders**, allowing dynamic configuration. Values can be supplied from external files like **values.yaml**, making deployments more flexible and manageable across multiple environments.

Problem statement 4 –

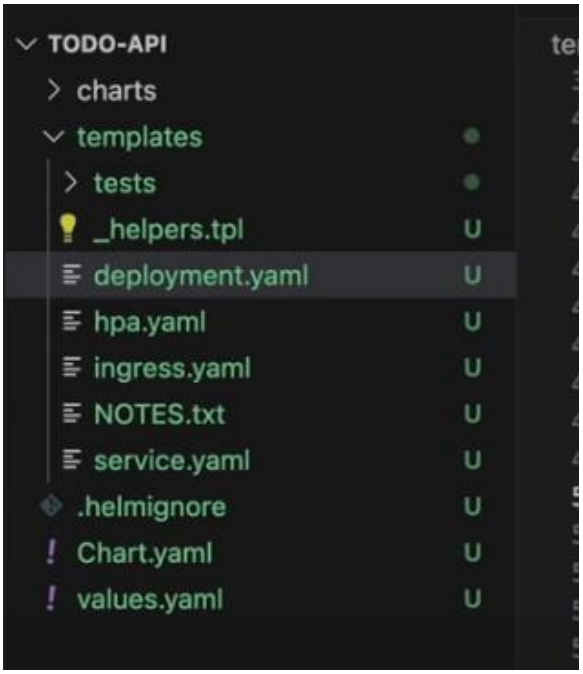
Another challenge with plain Kubernetes manifests is version management and rollbacks. Suppose we deploy an application as **version 1**, verify its functionality, and later update it to **version 2**. If an issue arises in version 2, rolling back can be difficult, as **kubectl** allows rollbacks for Deployments and StatefulSets but not for resources like **Secrets**. Helm simplifies this by creating **releases** and storing them as Kubernetes secrets. Each update generates a new **revision**, enabling easy rollbacks with a single command. Additionally, Helm allows packaging and sharing manifests effortlessly, making deployment and management more efficient. With its powerful features and seamless rollbacks, Helm significantly enhances Kubernetes application management.



How to use –

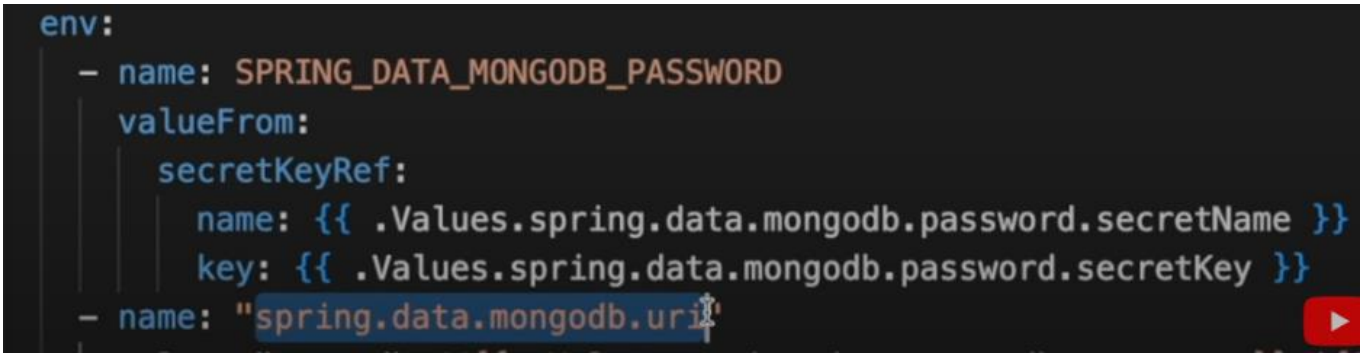
Install helm and run command: helm create <chart_name>

This will create directory structure for helm.

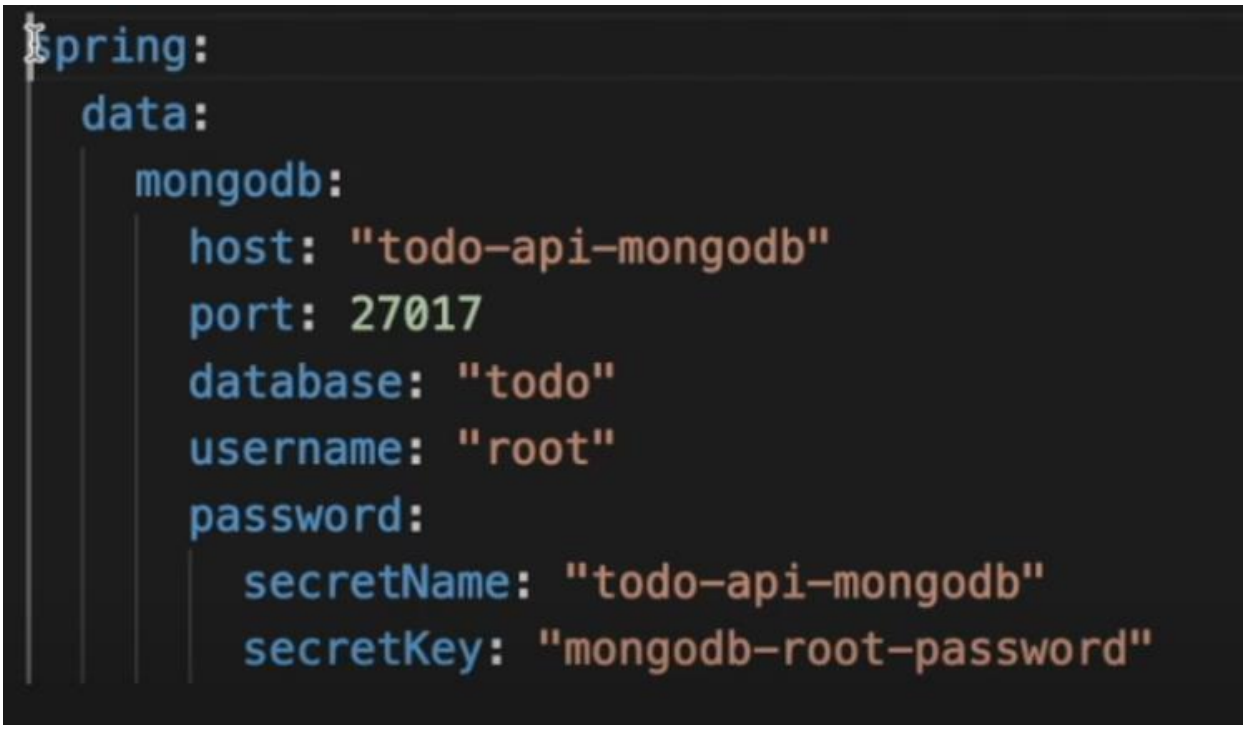


How to add and use values from values.yaml?

Usage -



values.yaml



Helm chart :-

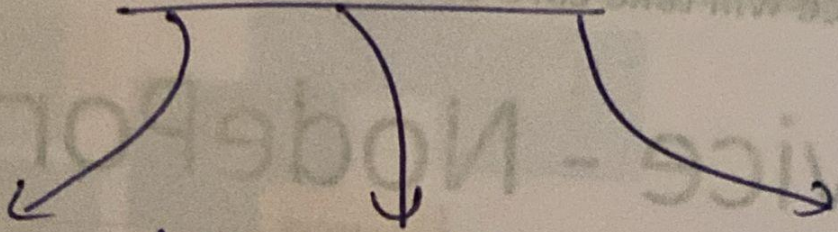


Chart.yaml

→ Metadata file contains versions and all.

templates

This dir contains manifests files written by us.

values.yaml

This file contains vars which can be used in manifests.

→ To apply all the manifests :-

```
helm install my-app ./<helm_dir_name>
```

(we will observe all manifests are applied.)

→ To remove deployment :-

```
helm uninstall my-app
```

Important helm commands –

```
# Install a Helm chart
helm install <release-name> <chart-name>

# Upgrade an existing release
helm upgrade <release-name> <chart-name>

# Rollback to a previous version
helm rollback <release-name> <revision-number>

# Uninstall a Helm release
helm uninstall <release-name>

# List all installed releases
helm list

# View the revision history of a release
helm history <release-name>

# Show the default values of a Helm chart
helm show values <chart-name>

# Package a Helm chart (create a .tgz file)
helm package <chart-directory>

# Search for a chart in the configured Helm repositories
helm search repo <chart-name>

# Add a Helm repository
helm repo add <repo-name> <repo-url>


# Update Helm repositories to fetch the latest chart versions
helm repo update

# Lint a Helm chart to check for errors
helm lint <chart-directory>

# Render Helm templates without deploying them
helm template <release-name> <chart-name>

# Check Helm client and server versions
helm version
```