When we deploy applications to K8s cluster, we may need to create multiple services. Very large application containing more than 40 microservices, and each service contains multiple K8s resources, such as Service , Deployment , StatefulSet , etc. The same application needs to be deployed in different environments, such as test, qa, uat, prod environment, etc., that is to say, the configuration of the application is also different.

For a large-scale application, it is very cumbersome and inefficient to deploy K8s resources one by one based on YAML files, and these YAML files are extremely complicated to maintain and prone to errors.

So, is there a more efficient way? For example, like a Docker image, all the K8s resource files required by the application are packaged together, and the application is deployed and managed as a whole through this package, thereby reducing the complexity of application deployment and maintenance.

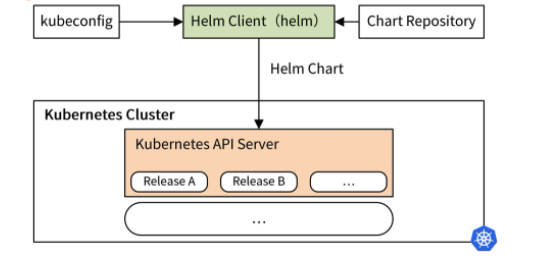
The answer is YES. You can manage K8s YAML files through the Helm Chart package, and create and manage your applications based on the Chart package through the helm command.

# What is Helm

Helm is currently the de facto standard for K8s service orchestration. It is a package manager for K8s, similar to pip for Python and yum for centos . The Helm Chart package contains a series of K8s resource definition in YAML format, as well as the configurations.

Helm also provides a helm command-line tool, which can create an application based on the Chart package. Application publishers can package applications, manage application dependencies, manage application versions, and publish applications to software repositories through Helm; for users, they do not need to write complex application deployment files after using Helm. Find, install, upgrade rollback and uninstall applications becomes very convenient.

The latest version of Helm is v3, and Helm3 builds on the core features of Helm2. Its architecture looks like the following:



* The core is the Helm client and the Helm Chart package.
* The helm command can download the Helm Chart package from repository, read the kubeconfig file and construct HTTP requests for kube-apiserver REST API interface.
* By calling the REST API interface provided by K8s, all the K8s resource defined in YAML format contained in the Chart package are created in K8s.

# Three Helm Basic Concepts

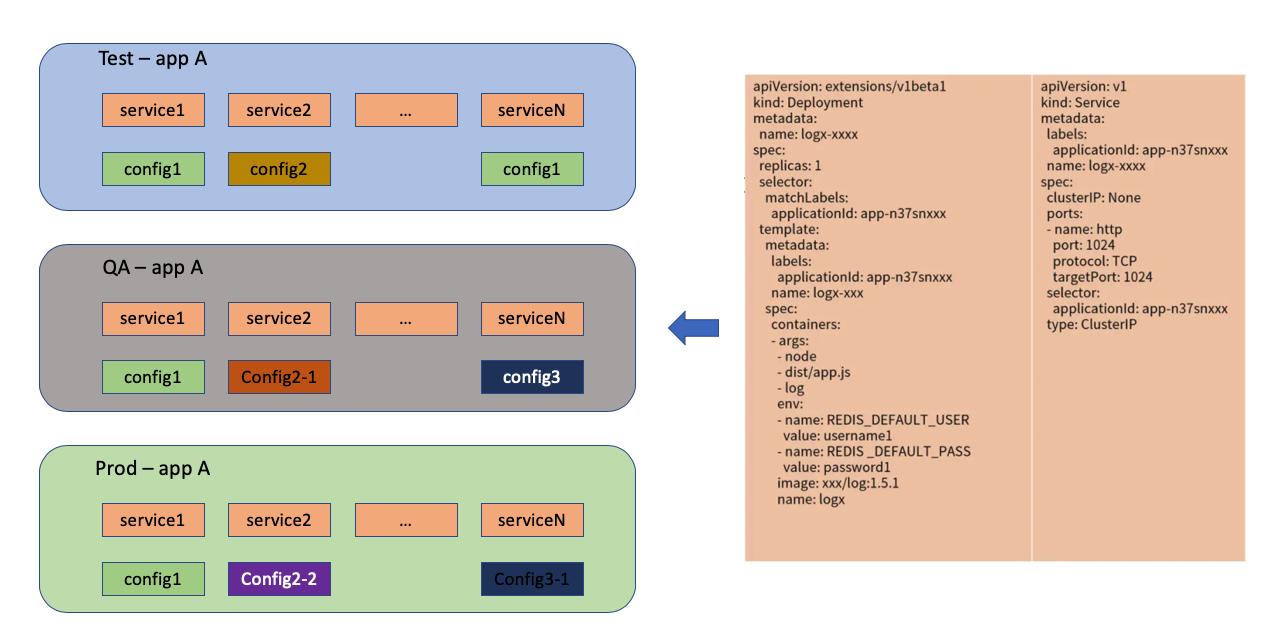
To learn and use Helm, you must understand the three basic concepts in Helm, and all operations of Helm are basically carried out around these concepts.

* **Chart:** Represents a Helm package. It contains all the resource definition files in YAML format needed to run an application, tool or service.
* **Repository:** It is a place to store and share Helm Charts, similar to Github repository where source code is stored, and Docker repository where images are stored.
* **Release:** It is an instance of Chart running in K8s. A Chart can usually be installed multiple times in the same K8s cluster. Every installation creates a new Release.

# Why Use Helm

Now you have a certain understanding of Helm , here I will talk about why you should use Helm .

Let’s first take a look at the traditional application deployment model:



In the above deployment model, we have three environments: test, qa and prod environment. Each environment deploys an application **A**. Application **A** contains multiple services, and each service contains its own configuration. Some configurations between different services are shared, such as configuration A.

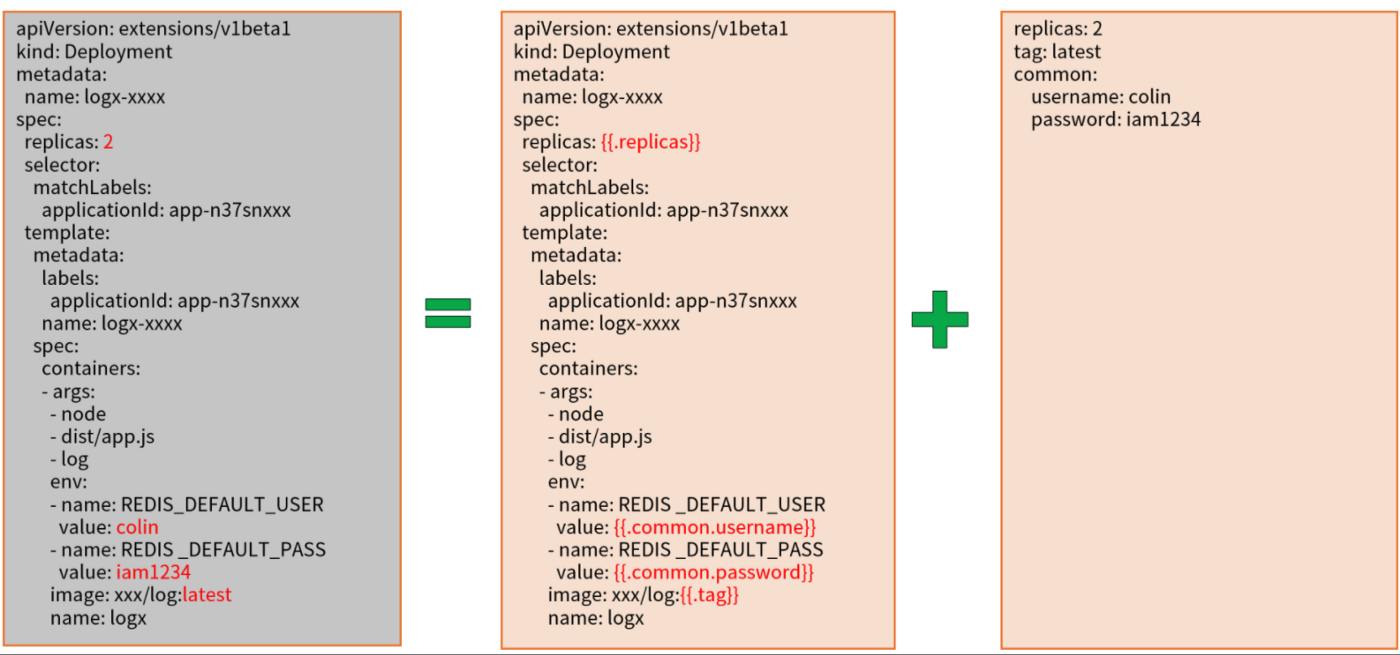
Each service is defined and created by a complex K8s YAML format file. It can be seen that maintaining these YAML format files in a traditional way and creating applications with different configurations in different environments is a very difficult task.

And as microservice grow larger, they face the following challenges:

* The increase in the number of services increases the difficulty of management and is a challenge to operation and maintenance deployment.
* As the number of services increases, service dependencies become more complex, and the management of service dependencies becomes more difficult.
* In terms of environmental information management, it becomes more difficult to rapidly deploy a complex application in a new environment.

So, a better way is needed to maintain and manage these YAML files, and Helm can help us solve these problems.

In Helm, it can be understood that it mainly contains two types of files: template files and configuration files. There are usually multiple template files and one configuration file. Helm template provides powerful template rendering capabilities and Helm can render the values in the configuration file into the template file, and finally generate a resource definition file. For example:



In the above example, application configurations are stored in the configure file:

replicas: 2  
tag: latest  
common:  
 username: srini  
 password: ab1234

So in Helm, deploying an application can be simplified as Chart template (multiple services) + Chart configuration -> application , as shown in the following figure:

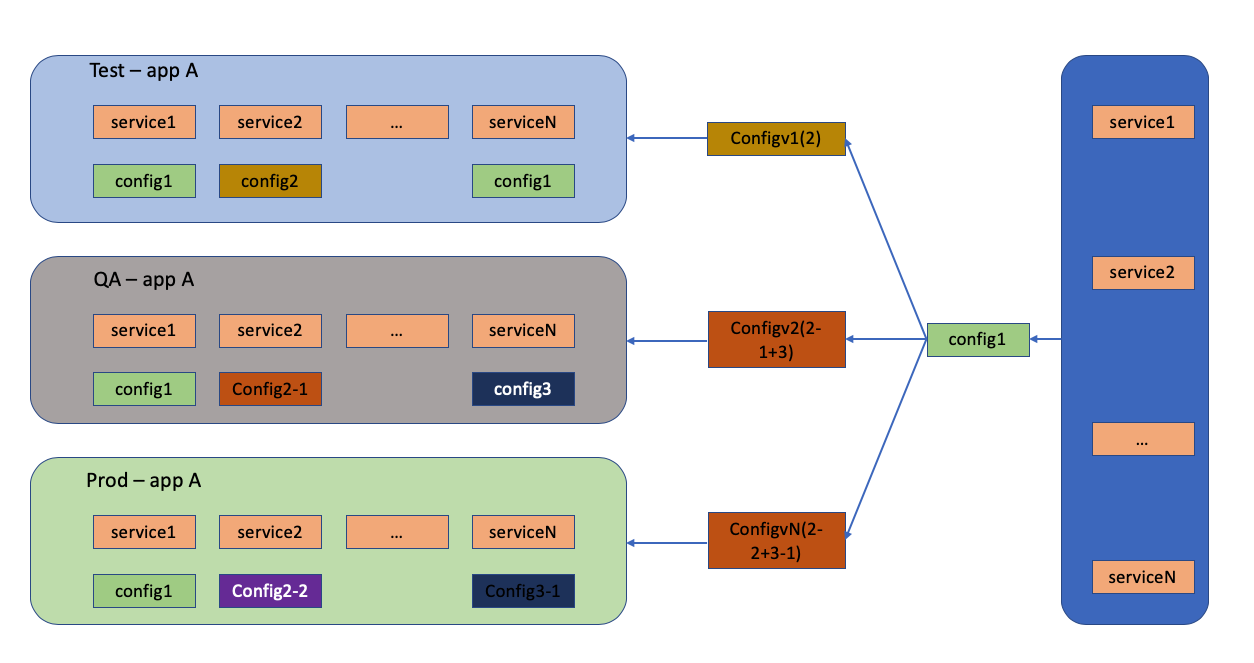


Chart templates are written only once for an application and can be used repeatedly. When deploying, you can specify different configurations to deploy applications in different environments, or deploy applications with different configurations in the same environment.