HELM

Helm is written in google go language

<https://razorops.com/blog/introduction-to-helm-3-the-package-manager-for-kubernetes>

If u want to see sample deployment.yaml file,where u can fetch data from values.yaml file then refer below

<https://github.com/bitnami/charts/blob/main/bitnami/nats/templates/deployment.yaml>

Generally in projects we write charts but here we can see all the predefined charts <https://github.com/bitnami/charts/tree/main/bitnami>

|  |  |
| --- | --- |
| To see all functions list | <https://helm.sh/docs/chart_template_guide/function_list/>  <https://helm.sh/docs/chart_template_guide/functions_and_pipelines/> |
|  |  |

1. **What is HELM**

* Helm is a package manager for kubernetes where package means it is a collection of template yaml files like like secret.yml, deployment.yaml file, values.yml , chart.yml

Helm is going to prepare a consolidated template yaml file by replacing the placeholders present in each template yml files with values present in values.yml

and that yml will be passed to kubernetes or ocp to create objects (pod, rs, deployment ) in kubernetes cluster

In kubernetes, we will create multiple yaml files right for each kubernetes object like –

1 yaml file for Deployment object

1 yaml file for Pod

1 yaml file for Service-and 1 for each sub type right,

1 yaml file for Ingress

In kubernetes to execute each file we use command “kubectl –f apply.yaml”. – Then that particular obj will be created

Since helm is a package manager –

Internally helm will combine all those template yamls together & prepares a single yaml file by replacing all places holder ${} with values present in values.yml and it will execute that file using “kubectl apply –f <consolidated yaml file name>” – then all resources will be created in kub ns

If u run 1 command all those objects mentioned in that file will be executed and objects will be created in kubernetes cluster

* Helm Chart – helm chart is the package of files – which contains all templates like secret.yml, deployment.yaml file, values.yml , chart.yml
* install the charts – means executing those template yml files and creating objects in ns
* helm only deals with charts

In summary helm is a powerful tool for deploying and managing kubernetes apps by executing helm templates in kub cluster

1. Helm architecture

|  |  |
| --- | --- |
| Template files with ${} | Generally we will place all the yaml files like deployment.yaml, configmap.yaml, secret.yaml –  Note these yaml file will not contain hard coded values- if we hardcode then we should maintain separate yaml files for each env , so alternatively each yaml file contain ${} symbol and that value will be fetched from values.yaml file |
| values.yml file | we will maintain separate values.yaml files for each environment  so that values from each file will be replaced in template file , as the template file contains ${}  But values.yaml file alone will be under root folder-not inside templates folder |
|  |  |

As mentioned above we have many yaml files, like deployment.yaml, pod.yaml, secret.yaml, configmap.yaml will be packaged and executed as one.

HELM as a package manager will package all the yaml files and create single helm chart

3)Helm advantages

and everything is versioned means it will maintain versions

helm will install the chart

|  |  |
| --- | --- |
| single shot deployment with single command which will replace all placeholders ${} with values passed from yml file | “helm install <desired chart name> <downloaded chart name>”  All the template yaml files (deployment.yaml,pod,yaml,rs,service.yaml,hpa.yaml,configmap, secret, serviceaccount which are present in the templates folder) will be executed and all objects will be deployed in single shot  if helm is not there, if 10 resource/templates files are there then we should execute 10 files  that to we write template files with place holders like ${} we should manually replace all placeholders with env spec file  which is difficult process  even if all 10 ymls executed successfully, in future if u want to revert to this successful version, its not possible, because we create objects indivudually |
| single shot uninstall | means with single command all objects will be uninstalled from cluster  ex:- uninstall will delete all related objects to that chart like it will delete deployment obj, rs obj, cm object .. all related objects  with out helm means we should manually delete each object one by one |
| rollback | suppose when ur prod deployment fails, you can easily rollback to the previous working version very easily if u use helm by  helm rollback <chart name> <version number to which u wanted to rollback>  if helm is not used,  in case if u changed db credentials, while rollback u should know old password again we should re-create the secret object with old password, we should re-create the old configmap (before doing installation we should manually copy the configmap obj contents) you should know the image name and while reverting again we should write the deployment object with old image name  again we should delete all old objects, and execute file by file to create object by object which is very difficult  for rollback purposes this is very great |

1. Helm Setup & Basic commands

helm –-help to get basic help

If u are giving any input values to in the command, then u should pass double hyphen

kubectl config set-context --current --namespace=default

this is not only here, this is in all languages

predefined chart names

famous chart names - bitnami/mysql or bitnami/apache

#### **Helm setup**

After downloading the helm, we should add it to path “so that helm --version”

|  |  |
| --- | --- |
| install helm and add it to path | Install helm from <https://github.com/helm/helm/releases>  “helm version” check the installation status using this |
| Helm needs kubernetes cluster or ocp cluster  to execute the yaml file, bec it will execute that objects definitions and keep that resources In kubernetes cluster only na  helm either needs kubernetes cluster info or openshift cluster info & all that cluster login info /token is maintained in “C:\Users\Manideep\.kube\config” file | |
| login to ocp cluster – I logged as 30 days free trail OCP cluster  login <https://oauth-openshift.apps.rm2.thpm.p1.openshiftapps.com/oauth/token/display>  and get token,    this config file needs to be placed in mentioned path | oc login --token=sha256~POkfaHJJil5vjrDUdfraFZjJr-WoSNRAbVh2DE5mqzc --server=https://api.rm2.thpm.p1.openshiftapps.com:6443  once u executed above command, that token info will be updated in “C:\Users\Manideep\.kube\config” file then onwards u can type “helm ls” |

Commands

note :- helm only deals with charts

|  |  |
| --- | --- |
| here we are dealing with repo, hence repo keyword is mandatory, generally since helm only deals with charts, when u are dealing with charts like installing chart, removing charts  then u don’t need to specify “chart” keyword  Generally in projects we write charts but here we can see all the predefined charts <https://github.com/bitnami/charts/tree/main/bitnami> | |
| Add chart repo – only if u add u can download the chart later-  In real time we will not add chart repo because we will have charts locally written – for demo purposes  we are adding remote charts | helm repo add bitnami <chart repo>  helm repo add bitnami <https://charts.bitnami.com/bitnami>  after adding type “helm repo list” – now u can see the repo list |
| list all chart repo | helm repo list or helm repo ls  to see all the locally added chart repositories |
| search | ex:- you can see apache related charts as “helm search repo apache”  “helm search repo mysql” all mysql related charts |
| remove | helm repo remove bitnami |
| template – this command will generate the consolidate template file with all values by replacing the place holders with values –but it will not execute template (means it wont create any object, it will just generate the yml file) | helm template <custom name> <repo name/ > --values = <values file>  helm template mydb <bitnami/mysql> --values = <values file location >  Note:- to exec this command we don’t need the cluster login also - but I didn’t tried by logging out  If helm code is in local  helm install <desired chart name > <folder name which contains all charts>  E:\ 3.HELM-charts>helm install c1 samplechart1  usecase:- in local if u keep on changing some code in template yml files, then if u want to just compile without creating objects then we should use this template command,  Because local debugging is best before executing this script in environment, so locally if u want to exec this script then prefer this |
|  |  |

|  |  |  |
| --- | --- | --- |
| helm install <chart name> <location> --dry-run | helm lint <chart name> <location> | helm template <chart name> <location> --values.yml |
| This will verify if the template is as per spec of not & this will also generate template by replacing the placeholders  ex:-  apiVersion: apps/v1  metadata:  here kind is missing this dryrun command will verify missing kind and it will errorout  whereas helm template/ helm lint command will not give any error | both lint and template will check for yml syntaxes but these will never validate the written definition against schema | this template command will just replace the placeholders {} with values present in values.yml  this cmnd doesnot validate the definition agains api spec, it will just replace the contents |

Installation –dryrun

|  |  |
| --- | --- |
| install downloaded chart –means  all template files will be execute & kub objects related to that image will be created in ns  Template files like – deployment.yml, secret.yml, configmap.yml..  internally helm will combine all those template yamls together & prepares a single yaml file by replacing all places holder ${}with values present in values.yml and it will execute that file using  “kubectl apply –f <consolidated yaml file name>” – then all resources will be created in kub ns | helm install <desired chart name> <chart location – repo name/ chart name >  or – if u don’t want to give chart name /if u want helm to genereate the name use below command  helm install <chart location – repo name/ chart name > --generate-name  helm install mydb bitnami/mysql  this chart contains mysql image name  installing a chart in different namespace - helm install mydb bitnami/mysql –n <existing ns name>  now the same chart will be installed in another ns  --  since helm only deals with charts  we don’t need to specially add chart in the command name |
| installing a local chart – this is useful when we want to deploy the code into env from local  in mvn pom.xml = in helm chart.yml is the main file | helm install <desired chart name > <folder name which contains all charts>  E:\ 3.HELM-charts>helm install c1 samplechart1  here in this I have a folder named “samplechart1” which contains chart.yml file |
| install or upgrade – if chart is not there it will install automatically  if present then it will upgrade | helm upgrade –-install <desired chart name> <– repo name/ chart name > |
| install same image with diff chart name  helm install <custom chart name> <chart repo/chart name> | helm install mywebserver bitnami/apache  helm install manimysql bitnami/mysql  C:\Users\Manideep>helm ls  NAME NAMESPACE REVISION STATUS CHART APP VERSION  manidb vv-manideep-dev 1 deployed mysql-12.2.1 8.4.3  manimysql vv-manideep-dev 1 deployed mysql-12.2.1 8.4.3 |
| install charts with custom properties file | helm install <desired chart name> <chart location – repo name/ chart name > --values <yaml file location>  this is most useful when we are passing env spec file |
| To see installed charts – with chart name | helm list or helm ls  C:\Users\Manideep>helm list  NAME NAMESPACE REVISION STATUS CHART APP VERSION  mydb vv-manideep-dev 1 deployed **mysql-12.2.1** 8.4.3 |
| install with dry run -dry run means-  helm will parse all those yml, it will replace the place holders  and check whether template files are in line with kub spec or not  but it will not at all execute that yaml it will just check syntax | helm install <desired chart name> <chart location – repo name/ chart name > --dry-run  or  helm upgrade <desired chart name> <chart location – repo name/ chart name > --dry-run  in prod we should do the dry run to just check if all objects are correct asper spec or not   * the advantage here is after doing dry run as it will parse yml file, after parsing It will give us the final consolidated yml file- we can see that consolidated yml file   Note:- both dryrun and helm template command will generate final consolidated yml file but prefer helm template command, because when we use dry run command- the generated yml file contains non yaml syntax too , so better use helm template command |
| To See the release notes | helm get notes <chart name>  helm get notes manidb |
| To see the user supplied values-  while installing chart we provide values.yml file location right   * this will tell what values are supplied to chart   we would have installed as helm install <desired chart name> <chart location – repo name/ chart name > --values <yaml file location>   * this cmnd will only tell us what values has been supplied to the chart from an ext yaml file like prd/values.yml file * if u want to see all values use “ helm get manifest” * if u want to see pod values – “oc rsh podname” and go into pod and type “env” | // we can see all the values passed to that chart for that particular revision too used in debugging purposes  //suppose for 4th release if u want to see what values have been passed to the chart  helm get values <installed chart name>  helm get values <installed chart name> --all  helm get values <installed chart name> --all -–revision <rev number> -- u can see all the values passed to the chart for that specific revision  generally in prod we do upgrade many times, if u want to see what values passed to the chart in last time , we can see using revision number   * this is especially used in debugged purposes, after someone installing chart, we can see what values have been passed to that chart – its like going inside pod and checking env var “oc rsh podname” “env”   ex:-  C:\Users\Manideep>helm get values manidb  USER-SUPPLIED VALUES:  null  C:\Users\Manideep>helm get values manidb --revision 2  USER-SUPPLIED VALUES:  null |
| To see the full template definition which was passed to cluster for any revision u can see, | helm get manifest <installed chart name> --revision <rev-number>  helm get manifest manidb --revision 2  ex:- for debugging purpose, if that chart installation is failed, if u want to debug and see what values have been passed and to see what is the full template file then we should use this command |
| wait or timeout – helm by default whenever kubectl receives the manifest file it will declare installation as success – it wont wait for pod, service obj creation and all | helm install <desired chart name> --wait --timeout 5m10s  ex:- helm install bitnami/apache --generate-name --wait --timeout 5m20s  so if u want helm to wait till obj creation then use flag –-wait  this will wait default for 5 m, if u want to wait more time then mention it under --timeout |
| Atomic install – means always successul version must be there  if current upgrade/installation is success it will go keep that, if it is failed then it will autorollback to previous helm successful release | helm install <desired chart name> bitnami/apache –-atomic  helm install maniapache bitnami/apache –atomic  helm upgrade --install maniapache bitnami/apache --atomic |
| see here when u create a chart  all the resources are grouped under the chart name | C:\Users\Manideep>helm ls  NAME NAMESPACE REVISION STATUS CHART APP VERSION  manidb vv-manideep-dev 2 deployed mysql-12.2.1 8.4.3 |

Analogy

If u want to see current pod info values- then you can go inside pod( “oc rsh <pod name>“ ) and type “env”, then u can see all the values passed to that pod

or u can go to even configmap and see the current values

but the problem is when u are debugging, if u check pod and configmap u can see only current values, but u cant see the old values, if u use “helm get values -–revision 2“

Then u can even see 3rd release values (helm get values <chart name> -–revision 3) and full chart (manifest) info for any release (this option is mostly used for debugging purposes)

Uninstall & upgrade

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| uninstall existing chart-  all objects like- deployment, config, secret obj(which contains related info & related secret obj), rs, pod,svc,route objs will be deleted from ns  with single shot,  without helm means we have to delete each obj one by one which is horrible and history will not be maintained, in future we cant rollback | helm uninstall <installed chart name>  C:\Users\Manideep>helm uninstall manimysql  release "manimysql" uninstalled   |  |  | | --- | --- | | Before uninstall | after uninstall | | C:\Users\Manideep>oc get service  NAME TYPE  hello-world ClusterIP  manidb-mysql ClusterIP  manidb-mysql-headless ClusterIP  manimysql ClusterIP  manimysql-headless ClusterIP  modelmesh-serving ClusterIP  C:\Users\Manideep>helm uninstall manimysql  release "manimysql" uninstalled | C:\Users\Manideep>oc get service  NAME TYPE  hello-world ClusterIP  manidb-mysql ClusterIP  manidb-mysql-headless ClusterIP  modelmesh-serving ClusterIP  see after uninstall 2 service objects named “manimysql” has been gone |   Generally if u uninstall, even the secret objects (which contains all release info), history for that charts also will be deleted, but if u want to keep those secret objects & history then use flag  if history is also deleted then in future u cant rollback, so if u want to preserve history then u keep flag  helm uninstall manimysql -–keep-history  with above command history & secret object will be preserved for future rollback purposes  Why to save history? – because in future if u want to rollback to a certain version , then its possible only when u have history, so while uninstalling be cautious and to preserve history or not |

|  |  |
| --- | --- |
| Before uninstall | after uninstall |
| before uninstall we have both secret and history  C:\Users\Manideep>oc get secret  NAME TYPE DATA AGE  sh.helm.release.v1.manidb.v1 helm.sh/release.v1 1 39h  sh.helm.release.v1.manidb.v2 helm.sh/release.v1 1 28h  C:\Users\Manideep>helm ls  NAME NAMESPACE REVISION STATUS CHART APP VERSION  manidb vv-manideep-dev 2 deployed mysql-12.2.1 8.4.3  C:\Users\Manideep>helm uninstall manidb --keep-history  release "manidb" uninstalled  now we have uninstalled the chart by maintaining the history | see here, even after we uninstall we can see history, secret object  with that history we can happily rollback  C:\Users\Manideep>helm ls  NAME NAMESPACE REVISION UPDATED STATUS CHART APP VERSION  C:\Users\Manideep>helm history manidb  REVISION STATUS CHART APP VERSION DESCRIPTION  1 superseded mysql-12.2.1 8.4.3 Install complete  2 uninstalled mysql-12.2.1 8.4.3 Uninstallation complete  C:\Users\Manideep>oc get secret  NAME TYPE DATA AGE  sh.helm.release.v1.manidb.v2 helm.sh/release.v1 1 28h  user-profile Opaque 3 14d |

|  |  |
| --- | --- |
| helm upgrade –  in real time this is used mostly in prod  because after 1st time installation, second time onwards we do upgrade command only  Note:- for every upgrade helm will create a secret object and it will keep all those upgradation/release-what has been changed that info will be kept in secret object  These are called release records  C:\Users\Manideep>oc get secret  NAME TYPE DATA AGE  sh.helm.release.v1.manidb.v1 helm.sh/release.v1 1 11h  sh.helm.release.v1.manidb.v2 helm.sh/release.v1 1 24m | helm upgrade <installed chart name> <chart repo/chart name>  helm upgrade <installed chart name> <chart repo/chart name> --values <values.yml file location>  helm upgrade <installed chart name> <chart repo/chart name> --values <values.yml file location> --dry-run  usecase:- after installing 1st release, 2nd time we will have newer image name placed in deployment.yml  now with helm upgrade, we are just updating the chart as content in template.yml file is changed  C:\Users\Manideep>helm ls  NAME NAMESPACE REVISION STATUS CHART APP VERSION  manidb vv-manideep-dev 1 deployed mysql-12.2.1 8.4.3  //you must give only installed chart name  C:\Users\Manideep>helm upgrade manidb2 bitnami/mysql  Error: UPGRADE FAILED: "manidb2" has no deployed releases  C:\Users\Manideep>helm upgrade manidb bitnami/mysql  Release "manidb" has been upgraded. Happy Helming!  NAME: manidb  LAST DEPLOYED: Sun Jan 12 08:04:39 2025  NAMESPACE: vv-manideep-dev  STATUS: deployed  REVISION: 2 |
| to see the status of chart  like – to see how many times chart has upgrade (revision num) all these chart info | helm status <installed chart name>  helm status manidb |

History & rollback

|  |  |
| --- | --- |
| history- u can see what happened like upgraded each time-  when u uninstall a chart make sure you maintain the history , else u cant rollback  helm uninstall manimysql -–keep-history | helm history <chart-name>  C:\Users\Manideep>helm history manidb  REVISION UPDATED STATUS CHART APP VERSION DESCRIPTION  1 Sat Jan 11 20:39:08 2025 superseded mysql-12.2.1 8.4.3 Install complete  2 Sun Jan 12 08:04:39 2025 deployed mysql-12.2.1 8.4.3 Upgrade complete |
|  | helm rollback <chart name> <version number to which version u wanted to move>  C:\Users\Manideep>helm history manidb  REVISION STATUS CHART APP VERSION DESCRIPTION  1 superseded mysql-12.2.1 8.4.3 Install complete  2 uninstalled mysql-12.2.1 8.4.3 Uninstallation complete  //Here we wanted to rollback to version 2  C:\Users\Manideep>helm rollback manidb 2  Rollback was a success! Happy Helming!  here, since we uninstall by maintaining history, we are now able to rollback |
|  |  |
|  |  |
|  |  |
|  |  |

Important commands

|  |  |
| --- | --- |
| in prod we want features like rollback | helm upgrade --install <existing chart name> <chart location>–atomic  helm upgrade --install maniapache bitnami/apache -–atomic  & since every we use same command either install or upgrade – best alternate is use this  autorollback is possible with atomic keyword |
| to check charts syntaxes | helm install <chart name> <location> --dry-run  This will verify if the template is as per spec of not & this will also generate template by replacing the placeholders  --both helm lint and helm template will check for yml syntaxes but these will never validate the written definition against schema  helm lint <current folder name>  helm lint samplechart1 |
| To see the full template definition which was passed to cluster for any revision u can see,  or to see all values  or to see the variables of a pod | helm get manifest <installed chart name> --revision <rev-number>  helm get manifest manidb --revision 2  ex:- for debugging purpose, if that chart installation is failed, if u want to debug and see what values have been passed and to see what is the full template file then we should use this command   * if u want to see full chart def and values use “ helm get manifest”   if u want to see pod values – “oc rsh podname” and go into pod and type “env” |

Charts & its commands

|  |  |
| --- | --- |
| Create charts | helm create <desired chart name>  ex: helm create c1  it will internally use nginx chart  now all template files are kept in “c1” folder |
| to check charts syntaxes (I mean to check syntax in any yml file – values.yml, deployment.yml) | helm lint <current folder name>  helm lint samplechart1 |
|  |  |
|  |  |

Values.yml

|  |  |
| --- | --- |
|  | here this values.yml file is the default values.yml file like application.properties file  you can have env specific yaml files also (like application-dev.yml / env specific yml file) |

Datatypes

here also we string

|  |  |
| --- | --- |
| String | {{ $myFlag :=”manideep”}} // Here := means 1st time initialization  {{$myFlag = “santu” }} // if no colon then it is re-assigning |
|  |  |

Syntaxes

|  |  |
| --- | --- |
| assigning variable  tcs:  goodAccount: wellsFargo  wellsFargo:  count: 20000  isActive: true | while declaring you must stick to initial type  after initializing if u are re-assigning a variable with different data type, that data will be ignored  {{ $name:= .Values.tcs.wellsFargo.isActive}}  {{ $name = "1234"}}   * see here while declaring we declared name as boolean, now we are re-assigning to boolean with string – this is not acceptable –but somehow its working |
| | 🡪 pipe symbol means  kind: Service  metadata:  name: {{ include "samplechart1.fullname" . }}  labels:  {{- include "samplechart1.labels" . | nindent 4 }} | generally pipe will transfer water from one place to another  here also same thing, pipe will pass the output of 1 function to another function    here we are passing this output to nindent4 function  here nindent 4 means it is a function takes the content , and it will place content after 4 spaces from starting |

usage of hyphen & spacing

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Ensure to keep – hyphen or proper space in every line – else formatting issues will come   |  |  | | --- | --- | | with hyphen | best **is keep hyphen in every line even for var declaration or remove hyphen in all** | | apiVersion: apps/v1  kind: Deployment  {{- $name := "1234" }}  {{- $name }}  metadata: | result:  apiVersion: apps/v1  kind: Deployment1234  metadata:  here since hyphen is there in line 3, it removed space and joined with above line | | * means remove extra space , this can be placed either in front or at last * Generally by default data will come in next line   {{- with .Values.tcs.wellsFargo.technologies}} here 1st hyphen means , instead of current line data will come in 1 line above |
| |  |  | | --- | --- | | with proper space |  | | apiVersion: apps/v1  kind: Deployment  {{ $name := "1234" }}  {{- $name }}  metadata: | result:  apiVersion: apps/v1  kind: Deployment  1234  metadata:  here since we removed hyphen the data 1234 came in next line , but its mandatory to print 1234 with some space, else error will come | | printing single value at after some colums  apiVersion: apps/v1  kind: Deployment  {{ $name := "1234" }}  {{$name }}  metadata: | result  apiVersion: apps/v1  kind: Deployment  1234  metadata:  explanation:- since we removed hyphen in line line 3,4 extra space will stay as it is  only thing is if u print at starting it will expect key:value, but if u are printing single val then print after some 3 columns space  hence line 4 I printed by giving some space | | printing key values at starting  apiVersion: apps/v1  kind: Deployment  {{ $name := "1234" }}  {{$name }} : {{ "Val" }}  metadata: | result  apiVersion: apps/v1  kind: Deployment  1234 : Val  metadata:  //since in 3rd line we didn’t use hyphen space came as it is |   bro | to print any variable name just use {{$<existed variable name>}}  ex: {{- $name }}   |  |  | | --- | --- | | without proper space | result | | apiVersion: apps/v1  kind: Deployment  {{ $name := "1234" }}  {{- $name }} | E:\study related\my git hub -new\JavaAllSubjectsNotes\6.devops\3.HELM-charts>helm template samplechart1  Error: YAML parse error on samplechart1/templates/deployment.yaml: error converting YAML to JSON: yaml: line 4: could not find expected ':'  this Is wrong as per yml syntax, because |   so summary is line 4: could not find expected ':'  this error will come due to improper spacing in above line   |  |  | | --- | --- | | printing single value at starting :  if u print at starting col we need to print as key : value else it will error out | if u want to print only single value then print after few space like print at 3rd col | | apiVersion: apps/v1  kind: Deployment  {{ $name := "1234" }}  {{$name }}  metadata: | result  deployment.yaml: error converting YAML to JSON: yaml: line 5: could not find expected ':'  this is because we tried to print at 1st col | |  |  | |

|  |  |
| --- | --- |
| here {{}} is like ${} in other languages- fetch something from  what ever you write outside {{}} those will just be considered as text  whatever you write inside {{}} will be evaluated  analogy:- this is same like in react, <> {all js code } </> in react also all js code must be kept inside braces {} then only it will be detected, here also if u want to evaluate any expression then use this | even if u want to declare any variable / write loops / use with keywords  any code u want to evaluate by go it must be written inside {{}} |
| . here dot means root object  it is like namespace | here the meaning of dot changes as per context   1. Generally, dot means root object 2. When u are using dot in with keyword 🡪 . means current element |
| .Values means values.yml file / Values object which represents values.yml file | apiVersion: apps/v1  kind: Deployment  {{.Values.image.repository}}  {{.Release}}  {{.Chart.Version}}  sometimes if small case didn’t work then use caps case |
| {{.Chart}}means chart file /chart object |  |
| {{.Release}} means release object  {{.Release.Name}}  {{ .Release.Namespace}} | assumption  class Release {  name, namespace, isInstall, isUpgrade, Service  } |
|  |  |
|  |  |

Sample function names

|  |  |
| --- | --- |
| upper & lower  metadata:  name: {{ include "samplechart1.fullname" . }}  labels:  {{- include "samplechart1.labels" . | nindent 4 | upper }} | here these functions takes arguments and converts the case |
| quote | this quote function will add quotes |
| toYaml | {{- with .Values.tcs.wellsFargo.technologies}}  {{- toYaml . | nindent 10}}  {{- end }} |

Keywords & operators

keywords in helm are 🡪 and , or

Helm –if-else

|  |  |
| --- | --- |
| Note:- for testing purposes u can write any helm code in deployment.yml file , if u write this code in other it may not work properly  apiVersion: apps/v1  kind: Deployment  {{.Values.image.repository}}  {{.Release}}  {{.Chart}}  {{.Chart.Version}}  hello | |
| note:- end keyword is mandatory even for if-else or for with |  |
| apiVersion: apps/v1  kind: Deployment  {{- if .Values.tcs.wellsFargo.isActive }}  {{ "Bro Account is active" }}  {{- else}}  {{"Account is disabled"}}  {{- end}}  metadata: | note:- while writing ensure, you are following the proper indentation in each line |
| {{- if .Values.autoscaling.enabled }}  apiVersion: autoscaling/v2  kind: HorizontalPodAutoscaler  metadata:  name: {{ include "samplechart1.fullname" . }}  labels:  // Here in values.yml if autoscaling is enabled then only this hpa object will be created  else it will not even be created at all | ingress.yml  {{- if .Values.ingress.enabled -}}  {{- $fullName := include "samplechart1.fullname" . -}}  {{- $svcPort := .Values.service.port -}}  {{- if and .Values.ingress.className (not (semverCompare ">=1.18-0" .Capabilities.KubeVersion.GitVersion)) }}  {{- if not (hasKey .Values.ingress.annotations "kubernetes.io/ingress.class") }}  {{- $\_ := set .Values.ingress.annotations "kubernetes.io/ingress.class" .Values.ingress.className}}  {{- end }}  {{- end }}  {{- if semverCompare ">=1.19-0" .Capabilities.KubeVersion.GitVersion -}}  apiVersion: networking.k8s.io/v1  //Here also if ingress is enabled then only this obje will be created |
|  |  |
|  |  |
|  |  |

With, range keywords

|  |  |
| --- | --- |
| here the meaning of dot changes as per context   1. Generally, dot means root object 2. When u are using dot in with, range keywords 🡪then . means current element   The advantage of range keyword is range keywords can loop even on map/ dictionary | tcs:  goodAccount: wellsFargo  wellsFargo:  count: 20000  isActive: true  technologies:  - java  - Dot net  - openshift |
| apiVersion: apps/v1  kind: Deployment  {{- if .Values.tcs.wellsFargo.isActive }}  {{ "Bro Account is active" }}  {{- else}}  {{"Account is disabled"}}  {{- end}}  {{- with .Values.tcs.wellsFargo.technologies }}  {{ . }}  {{- end }}  metadata: | output:  # Source: samplechart1/templates/deployment.yaml  apiVersion: apps/v1  kind: Deployment  Bro Account is active  [java Dot net openshift]  metadata:  error scene   |  |  | | --- | --- | | {{- with .Values.tcs.wellsFargo.technologies }}  {{ . }}  {{ end }} | it will error because if u try to print single value you must print after some space  here it will error because I tried to print at start column  ex:- line 5: could not find expected ':' | |
| \_\_ printing with toYml function  apiVersion: apps/v1  kind: Deployment  {{- with .Values.tcs.wellsFargo.technologies}}  {{- toYaml . | nindent 10}}  {{- end }} | # Source: samplechart1/templates/deployment.yaml  apiVersion: apps/v1  kind: Deployment  - java  - Dot net  - openshift |

|  |  |
| --- | --- |
| map as sub object to other | each entry key value pair as root object |
| range  apiVersion: apps/v1  kind: Deployment  countries of Deployment are :  {{- range $key, $name := .Values.tcs.amex }}  {{$key}} : {{$name}}  {{- end }}  metadata: | apiVersion: apps/v1  kind: Deployment  {{- range $key, $name := .Values.tcs.amex }}  {{$key}} : {{$name}}  {{- end }}  metadata: |
| apiVersion: apps/v1  kind: Deployment  countries of Deployment are :  location : Deccanpark  onsite : yesitsavailable  metadata:  Note:- to print as key value pairs, we should enclose all those key value pairs in some object like here I have enclosed them in “countries of Deployment are :”  here this sub object must be tied to parent root object | result  apiVersion: apps/v1  kind: Deployment  location : Deccanpark  onsite : yesitsavailable  metadata:  Note:- either this map should be a tied to a parent object or  individual entries should be tied globally |

1. Create a helm project

Chart contains kubernetes yaml file

helm create <desired-helm-project folder name>

helm create helloworld

if u execute it, it will create a folder called “helloworld” where it contains charts..

1. Values.yaml

If u want to fetch anything from values.yaml being in any other file ,we have to use

“{{ .Values.hostAliases }}” whereas Values. Refers to values.yaml file

Sample values.YAML



1. Install-uninstall all the yaml files

In simple words-helm will execute all those yaml files and creates kubernetes objects as mentioned as per

Yaml files in the namespace.

Single yaml file

helm install <release-name> <foldername where all files are present>

Multiple yaml files

when multiple yaml files are present 1 per environment basis, then we should install as below

in case of multiple yaml files, we have to tell which yaml file and where is that yaml file

helm install <release-name> <folder-name> ---values ./<foldername>/<values-\*.yaml>

helm install v1 proj1 --values proj1/values-dev.yaml

or by mentioning the current directory

**helm install r1 proj1** -**-values ./proj1/values-dev.yaml**

Uninstall the installed,

If u uninstall all the deployed objects present in that namespace will be uninstalled or deleted

helm uninstall <release-name>

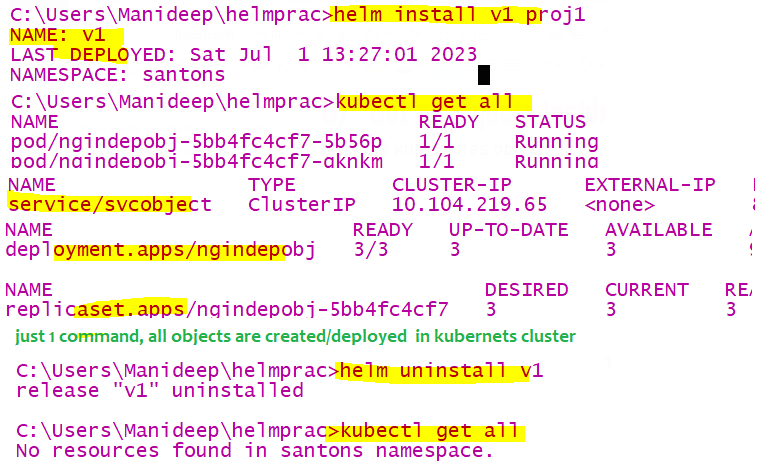
When u install many kubernetes objects like pod,deployment,service objects would have got created,

If u want to uninstall all those objects at once then use this command

Some releases will be another namespace, so u have to go to that namespace and delete else, u can use –n flag

**helm uninstall v5 -n manins**

**helm uninstall v4 -n manins**



1. Upgrade the image name

example project:- for simplicity purpose, keep only deployment,hpa,service yaml files and remove everything

when u made any changes like updating the image name in deployment.yaml or if u made any changes in any yaml

if u want to uninstall all objects and create all objects again **,then prefer upgrade**

Let’s say if u modified the project and then no need to helm install, u should do

helm upgrade <existing-release-name> <folder-name>

helm upgrade <release-name> <folder-name> ---values ./<foldername>/<values-\*.yaml>

helm upgrade r1 proj1 --values ./proj1/values-dev.yaml

lets say u upgrade it and u want to see what is the revision made ,then type “helm history r1”

1. List out all helm release

|  |  |
| --- | --- |
| To list out all helm releases | helm list |
| if u want to get all releases (some release build number) from all namespaces | helm list -A |
|  |  |
|  |  |

1. Get version/revision history

helm history <release-name>

helm history v1

1. Rollback to older release

helm rollback <release-name>

helm rollback r1