# INDEX

[**INDEX 1**](#_iu9rd052ekaf)

[**Why helm? 4**](#_xken97okeaoj)

[**Install Helm 4**](#_b3md9heu8sc5)

[**Connecting HELM to K8s 4**](#_86h94kf7xgdx)

[**HELM Basics 4**](#_q51bepaj3fx2)

[Basic Commands 4](#_jz533qeh42or)

[Practicing Installation of MySQL with HELM 6](#_d9a3divtd8jm)

[Namespace-specific Installation 8](#_rlmj0nfkb57a)

[Passing custom configuration to helm 9](#_6sxtfrc7hvxl)

[Helm Upgrade 10](#_yhmwwrnp6x4e)

[Reusing Values 11](#_bx298r9qqm1q)

[Release Records 11](#_82qj15wma12k)

[**Assignment 1: Install Tomcat using custom configuration 12**](#_61kjng7cwxjj)

[**Advanced Commands 14**](#_ubcotdj5ii98)

[What happens in the background 14](#_jm8myt9djvx9)

[HELM Dry run 14](#_ec65jvrg6td5)

[Helm Template Command 14](#_63j8o12pzywk)

[Helm Get-Command 14](#_7g4ld2d5n9b5)

[Helm History 15](#_7eu9jlpb1flt)

[Helm Rollback 16](#_zgf4h7wbpgfm)

[Create Namespace during new installation 16](#_khhdyd4a19wu)

[Helm Upgrade or Install 17](#_vfqy42lsvs1u)

[Generate name for releases 17](#_nzd6v1zt1w)

[Helm Wait 18](#_r91kix2ov2h1)

[Helm Timeout 19](#_pe84rndz89ly)

[Helm Atomic Install 19](#_hnlbzdz3rcu3)

[Helm Forceful Upgrades 20](#_bkdb2w6a2jiv)

[Helm Cleanup on failure 21](#_mxginikw4lwb)

[**Assignment 2: 21**](#_y1ii0mjl18l)

[**Working with own Charts 22**](#_v0rtonx132ty)

[Create Helm Charts 22](#_8rynmv8p7ymh)

[Folder/File Structure 22](#_b1p2mtmdma54)

[Installing the chart using the custom chart 23](#_7xnuag942i9y)

[How Notes is rendered? 24](#_whnc6jurhyab)

[Understanding Files/Folders 24](#_6akddow8bb0g)

[Understand Chart.yaml 24](#_6m4qpbh1cuq9)

[Templates Folder 25](#_s571qdk3fb0i)

[Understand Helpers.tpl file 26](#_3k94g8llpyah)

[Understand Values.yaml file 27](#_kof0j13ftdpm)

[Helm Packaging 28](#_o1h3ru38m3vj)

[Package dependency update 28](#_dawd1hiwlfzf)

[Package Location 29](#_p19r63bcs7bn)

[Other Options for packaging 29](#_u1xh3owmtlby)

[Helm Lint 30](#_dclo5qc81m8o)

[Helm Ignore 31](#_wtxxco395u8)

[**Helm Template & Go Lang 31**](#_cil0gt4v8jts)

[Template Actions 31](#_ejhncqbh7a1n)

[Action element - White space removal 31](#_eul0ro3wjkjn)

[Action element - String concat 31](#_t27b8xf3hkxb)

[Template Information 32](#_4dfq9vf00ynb)

[Template Info: .Values object 32](#_kvu2dyy3ddoh)

[Template info: .Chart object 33](#_e34z9o544yzz)

[Template Info: .Release 34](#_qsclqg5cht71)

[Template Info: .Template 34](#_kikfd0ee52fi)

[Template Info: Pipelines 34](#_y3tbjbxgsr0v)

[Template Functions 35](#_jv0y6s95am9x)

[Default function 35](#_3dxysbzfbdq2)

[Upper function 35](#_uuwfhn81bxrs)

[Lower function 35](#_7hldezeew6pn)

[Quote function 36](#_oeby6oewg80h)

[Template Conditions 36](#_wk9gstffyauf)

[Conditions: IF ELSE 36](#_kdmd19iaw8kr)

[Conditions: With 37](#_g3hy3fymo04h)

[Conditions: Variables 38](#_cd3svlfw7r1l)

[Template Loops 40](#_xm6n1qiyen3e)

[Loops: For loop 40](#_9gxkmjmijegu)

[Loops: Dict types 40](#_2jx326wh1k5v)

[Debugging the Templates 40](#_h6j9t5uhleks)

[Helm Get Manifests 41](#_9a4xzuw4p981)

[Helpers.tpl file 41](#_af5rcqqqzu4h)

[Define custom Template 42](#_t95gt6ge83ff)

[Use the template 42](#_qploa3m63mhd)

[Passing Default Value 43](#_ku9g5noqwj5m)

[**Advanced Chart 44**](#_ubfmqawmnkil)

[Dependencies 44](#_cley0opk2uer)

[Dependency Version Range 45](#_xhn10eexrak7)

[Chart Lock 46](#_7dwz2mv8si7j)

[Using RepoName instead of URL 46](#_jhb8q1sfw5oy)

[Conditional Dependency 47](#_8zbytfzcx3kg)

[Multiple Conditional Dependencies 48](#_imafn4fp4s6u)

[Passing Values to Dependencies 49](#_bqn0i3aqjqsi)

[Read Values from Child Chart 51](#_p75sydy18xan)

[Explicit export and import 51](#_fkzm3dnc31jo)

[Use values that are not exported 51](#_i41c2rr2rm9o)

[Hooks 52](#_s5y0ki28kauq)

[Options: 53](#_d0n904hkjw3p)

[Create and use hook 54](#_u5wuyvgx0m13)

[Testing your chart 55](#_zblwbizghp9s)

[Perform testing on a chart 56](#_s98cponlabqb)

[**Assignment 3 56**](#_6k0iup9a4581)

[**Repositories 57**](#_qo24gt5hi452)

[How to host local chart repository 58](#_nc9xpbn1fzdq)

[Host webserver using python 59](#_v5fcuhjma4g5)

[Using the local repository 60](#_b62e5rkacabu)

[Installing using HELM PULL 61](#_ikksjp3mb8y4)

[Update Repositories 61](#_kmsohr4d3f1x)

[Github Pages for hosting your charts 62](#_azbx57efn941)

[OCI Experimental 64](#_96555os9xa68)

[**Chart Security 65**](#_47328lidz13b)

[PGP Installation 66](#_lhpof7m7a10o)

[Generate the key 66](#_ojt8fuazuba5)

[Signing Charts 68](#_o8cf1kq9r4is)

[Verify Signature of chart 69](#_q2iby8ga7t8f)

The most used helm package repository: <https://bitnami.com/stacks/helm>

# Why helm?

1. Revision history
2. Consistency

# Install Helm

There are 3 ways to install HELM

1. Using Binaries
2. Using script. Download the script and run it
3. Using package managers
   1. $ brew install helm
   2. $ choco install kubernetes-helm
   3. USING APT/YUM

Check Installation: $ helm --help

# Connecting HELM to K8s

HELM uses the same kube config file as kubectl to connect to the cluster.

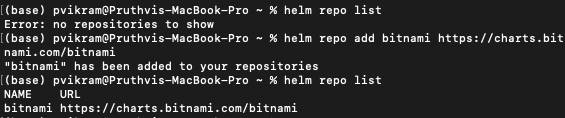
# HELM Basics

## Basic Commands

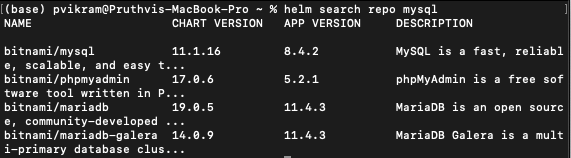
| Show repo list | $ helm repo list |
| --- | --- |
| Add new repository | $ helm repo add myrepo [https://charts.bitnami.com/bitnami](https://charts.com/bitnami) |
| Search the repo | $ helm search repo mysql |
| List all versions | $ helm search repo mysql --version By default the latest version of the chart is shown. |
| Remove repository | $ helm repo remove myrepo |
| Install a chart | $ helm install mysql myrepo/mysql mysql name is unique in a namespace. |
| Check the installation status | $ helm status mysql |
| List releases | $ helm list  $ helm list --namespace mysql By default revisions are listed from default ns. Use --namespace or -n to specify a specific namespace |
| Remove an installed release | $ helm uninstall mysql  $ helm uninstall -n mysql mysql Without namespace, the uninstallation happens from default namespace |
| To upgrade the existing repository | $ helm repo update It will connect to the remote repository and pull the latest updates. |
| To update an installed chart | $ helm upgrade mysql bitnami/mysql --values /Users/pvikram/Documents/Learning/HELM/mysql/properties.yaml |

## Practicing Installation of MySQL with HELM

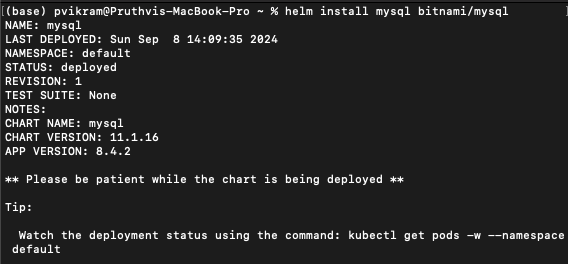
Adding repo and listing repos



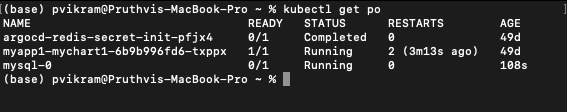
Search for mysql in repos



Installing mysql from bitnami repo

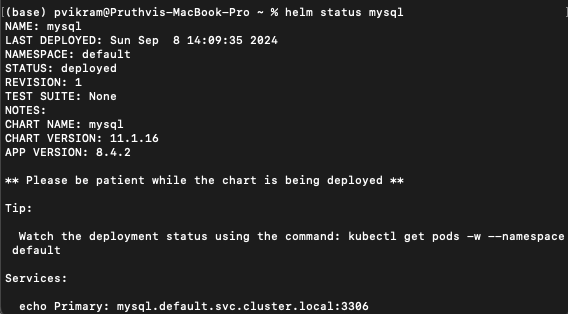


Check the status of installation

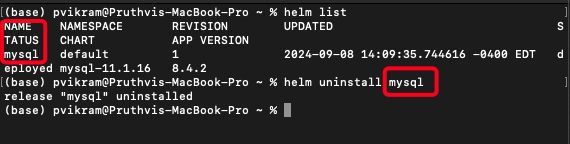


Get helm installation status

(There will be some instructions to configure mysql or any other chart. Use the below command to get the status and instructions)

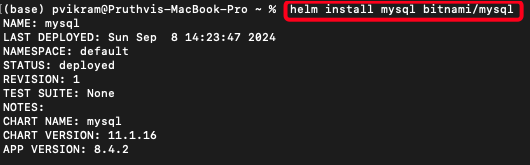


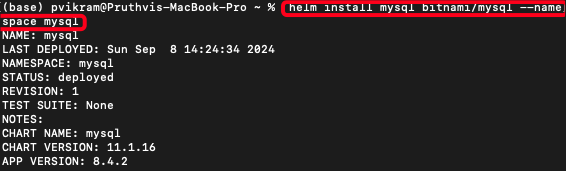
Remove installed chart/release

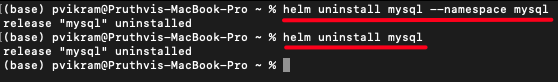


## Namespace-specific Installation

Names for installed releases must have a unique name in a namespace. Same name can be used in different namespaces.







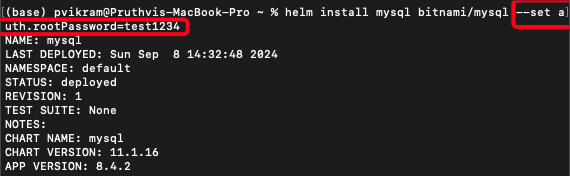
## Passing custom configuration to helm

Example: When installing mysql using helm, a password is generated by helm chart. But we can pass the configuration to take the default password.

There are two ways in which custom config is passed.

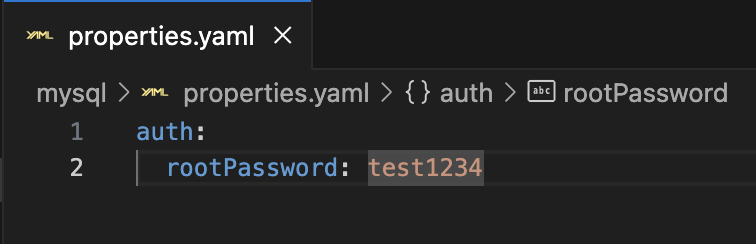
1. Using --set
2. Using YAML property file

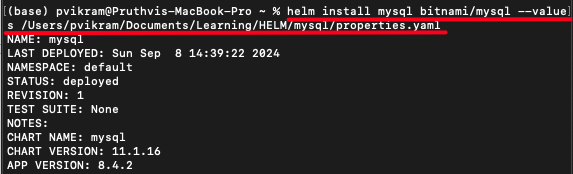
Using --set

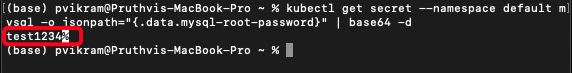


Using property file

Create a property yaml file and place all your configurations in that file. Pass the file which installing a chart.







## Helm Upgrade

Updating the repository:

$ helm repo update

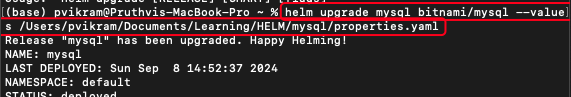
This command downloads the latest chart versions from the remote repository.

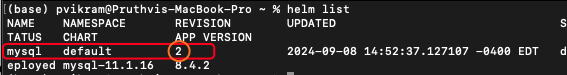
To update a chart, run the below command. Usually the commands to upgrade a particular chart is already available in the chart documentation.

$ helm upgrade mysql bitnami/mysql --values <path to values file>.

This command will upgrade the installed chart with the latest version if available. You can also see that chart revision is updated.







Helm is smart enough to send just the updates that are required. It wont send all the changes.

## Reusing Values

When performing an upgrade, helm requires same values to be passed again that were used during first installation. When same values are not passed, the helm uses the default values to perform the upgrade.

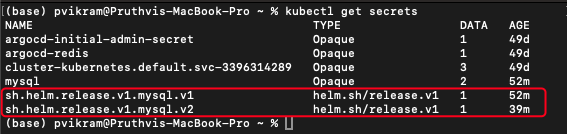
To avoid this, use --reuse-values option

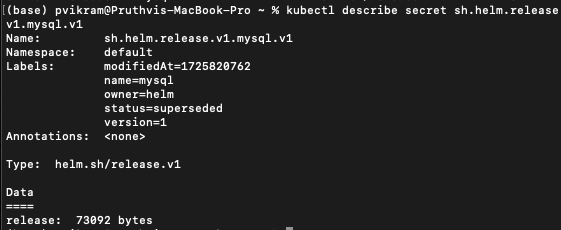
$ helm upgrade mysql bitnami/mysql --reuse-values

This command ensures that the same values are used.

## Release Records

Helm maintains release records as secrets in kubernetes. To check the records, use the kubectl get secrets command.





Even after uninstallation, you can store the history

$ helm uninstall mysql --keep-history

# Assignment 1: Install Tomcat using custom configuration

Created below custom configuration for tomcat server.

tomcatPassword: "testpassword"

service:

type: NodePort

nodePorts:

http: 30007

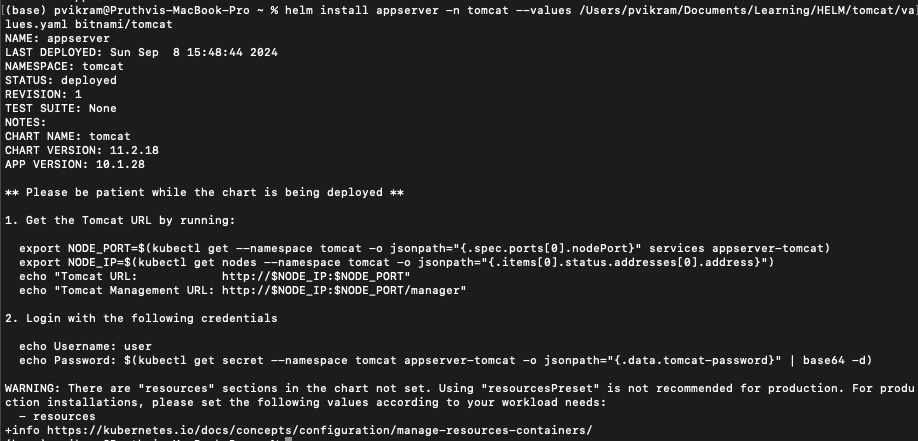
pdb:

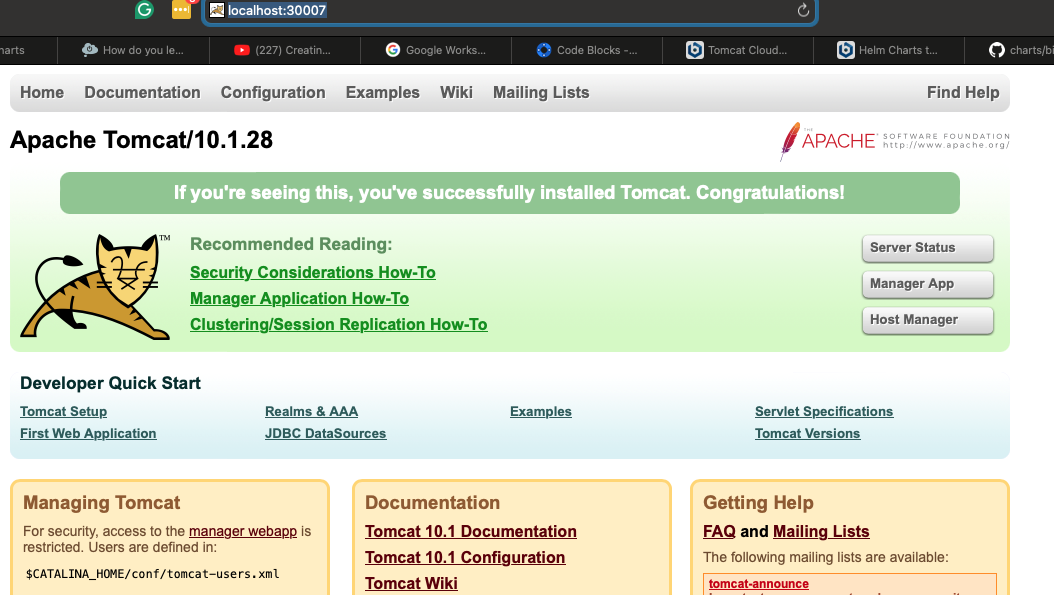
create: true

minAvailable: 2

Other configuration details can be found in the below links.  
<https://github.com/bitnami/charts/tree/main/bitnami/tomcat/#installing-the-chart>

<https://artifacthub.io/packages/helm/bitnami/tomcat/10.3.5>





# 

# Advanced Commands

## What happens in the background

Focus on what happens during the installation of a helm chart

1. Load the chart templates and dependencies (if local, else downloads the chart and loads them)
2. Parses the default values.yaml
3. Generates the final YAML files by replacing the placeholders
4. Validates the generated Yaml
5. Generates the final YAML and sends it to Kube

## HELM Dry run

The dry run helps you check if the templates, values, and everything are properly created and the final YAML is validated.

With a dry-run, 5th phase from above will not be performed.

$ helm install mydb bitnami/mysql –values <path> --dry-run

## Helm Template Command

Several projects require templates as dry-run include non-yaml format code.

$ helm template mysql bitnami/mysql --namespace mysql --values /Users/pvikram/Documents/Learning/HELM/tomcat/values.yaml

Template command does not just send yaml to kube but also doesn’t validate the code.

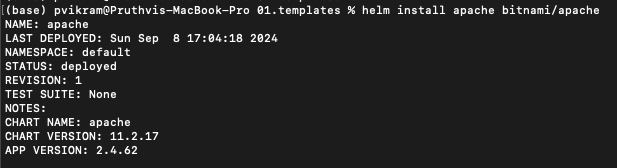
## Helm Get-Command

Helm get allows you to do few things

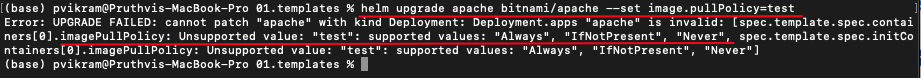
1. Get release notes: helm get notes mysql
2. Get Values used: helm get values mysql (only used values)
   1. helm get values mysql --all (Gives all values even not used.)
   2. helm get values mysql --revision 2 (Values of a special revision)
3. Get manifests: helm get manifest mysql (can be used with revision)

## Helm History

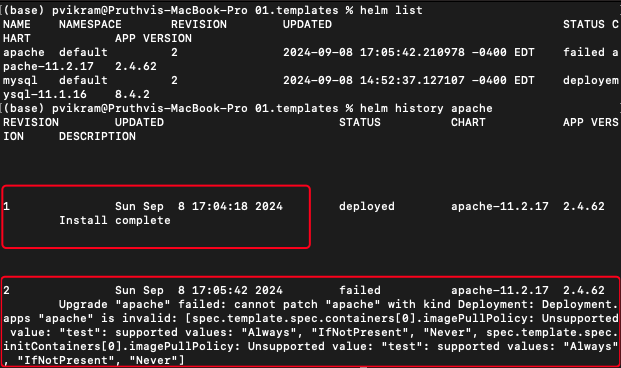
This allows you to track all changes made on the chart. It includes not only the upgrade information, and revision information, but also provides the error information. For example, if you install Apache webserver and upgrade with an invalid value, the upgrade fails and that information is also stored in the history.



Let’s try to upgrade with an invalid value



Let’s check the history

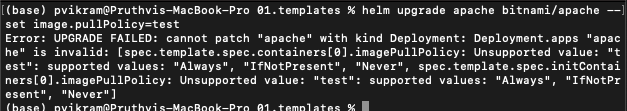


## Helm Rollback

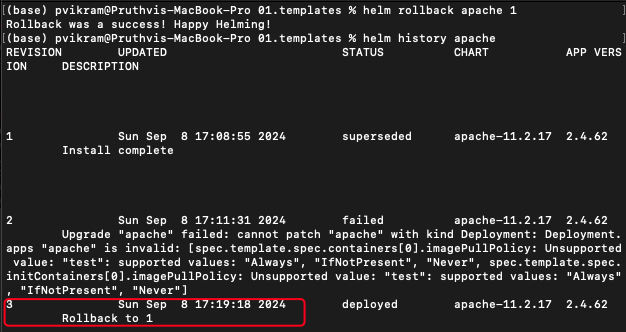
Allows you to roll back during any failure. In the following scenario, the newly installed apache server is running as expected.



Let’s upgrade it with an error



Let’s check the rollback function.

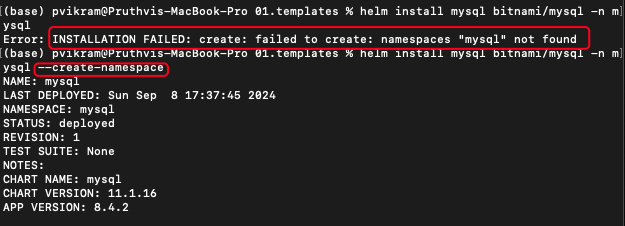


## Create Namespace during new installation

helm install by default does not create a new namespace and also fails if the namespace doesn’t exist. But, you can force the helm command to create the namespace for you.

$ helm install mysql --namespace mysql --create-namespace bitnami/mysql

This will ensure that the namespace exists if not creates it.



## Helm Upgrade or Install

In CI/CD pipeline this will be very useful. This command will allow to select between install and upgrade operations

1. If the chart is already installed, the upgrade is used.
2. If the chart is not installed, the chart is installed with out making any changes to the command.

$ helm upgrade **--install** mysql bitnami/mysql

## Generate name for releases

You can ask helm to create a name for the release by using the below option. So that you dont have to pass the name to the command. Helm also make sure that the names are not re used.

$ helm install bitnami/mysql --generate-name

You can also control the name generation using below option.

$ helm install bitnami/mysql --generate-name --name-template “webserver-**{{randAlpha 7 | lower}}**”

{{}} - Go lang expression

randAlpha - Golang function

7 - attribute value to the function

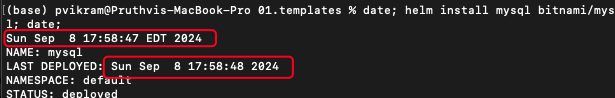
lower - Upper case is not allowed by K8s in their names. Hence this golang func allows to convert to lowercase.

## Helm Wait

By default, helm doesn’t wait for the resources to be created. Once the YAML is sent to the Kube in the 5th phase, the helm operation will end. But we can change this functionality by passing the **--wait** option.

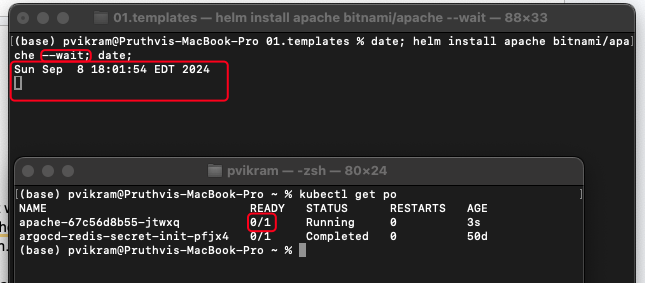
$ helm install mysql bitnami/mysql --wait

This option ensures that the resources are created and the pods are up and running.

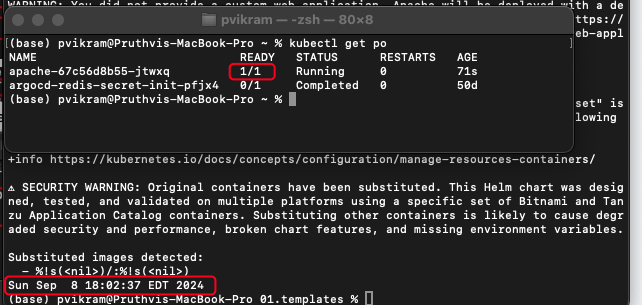




Let’s try installing with the wait option



You can compare the start and end time. Only when the requested pods are completely up and running, then only the helm operation exits.



## Helm Timeout

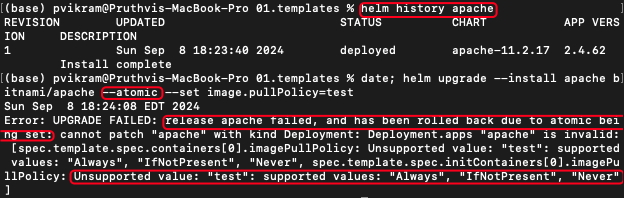
Wait option is a great way to ensure the successful creation of K8s resources. But what if the installation is held due to k8s resources not getting created successfully. Timeout option helps you to define the length of time you want to wait.

$ helm install apache bitnami/apache --wait --timeout 10m5s

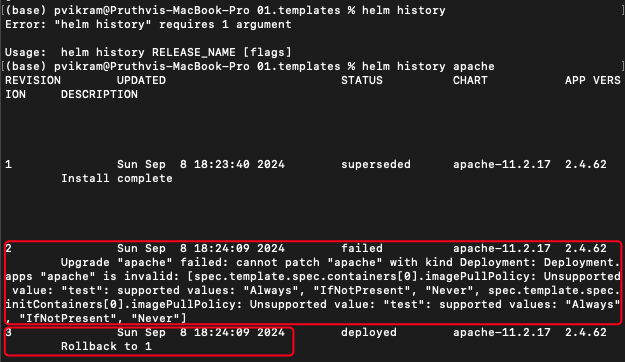
## Helm Atomic Install

Due to any reason, if the release fails, the --atomic option will roll it back to the previous successful release.

$ helm install apache bitnami/apache --atomic



The rollback information is also recorded in the helm history with a new revision which is rolling back to revision 1.



You can also use timeout and wait with atomic releases.

## Helm Forceful Upgrades

With default upgrade, the helm only restarts pods whose values are changed rather than restarting all the pods. We can change this functionality by passing the force option.

$ helm upgrade mysql bitnami/mysql --force

## Helm Cleanup on failure

During any failure, not all resources like config maps, etc are removed. You can pass --cleanup-on-failure option to remove them.

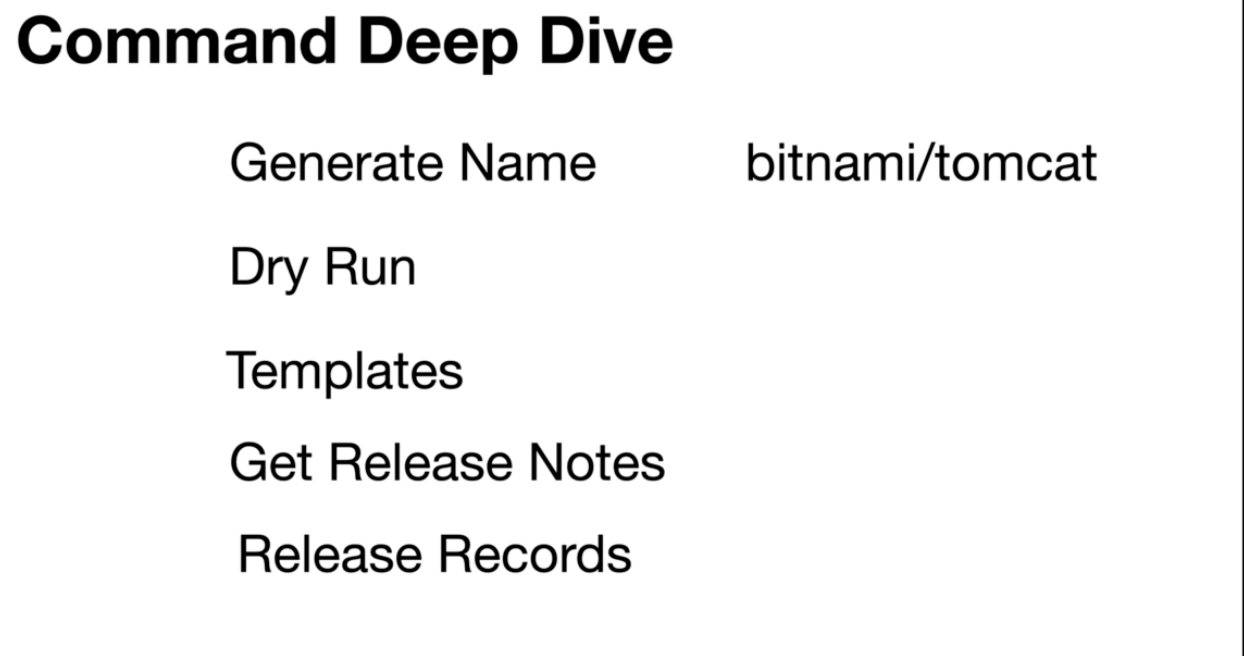
$ helm upgrade --install apache bitnami/apache --cleanup-on-failure

This will ensure the upgrade or install cleans all the resources when failed to complete successfully.

This is not recommended as cleaning all will not allow us to debug the issue.

# Assignment 2:

Install apache tomcat on the kubernetes cluster and write commands for all the below.



| Generate Name | $ helm install bitnami/tomcat --generate-name --name-template “server-{{randAlpha 10 | lower}}” If the command is failing make sure to check the double-quotations |
| --- | --- |
| Dry run | $ helm install bitnami/tomcat --generate-name --name-template “server-{{randAlpha 10 | lower}}” --dry-run If the command is failing make sure to check the double-quotations |
| Templates | $ helm template bitnami/tomcat --generate-name --name-template "server-{{randAlpha 10|lower}}" If the command is failing make sure to check the double-quotations |
| Get release notes | $ helm get notes server-hggblacjrz |
| Get release records | $ kubectl get secrets |

# Working with own Charts

## Create Helm Charts

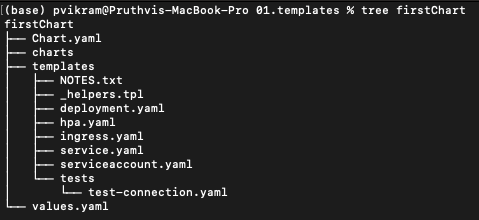
To create your own helm chart use the below command to create the template. By default, the nginx chart is used to create the structure.

$ helm create first-chart

But you can use a different chart as your starter chart.

### Folder/File Structure

By running the above command, the below structure is created under firstChart folder which is the name we passed in the command.

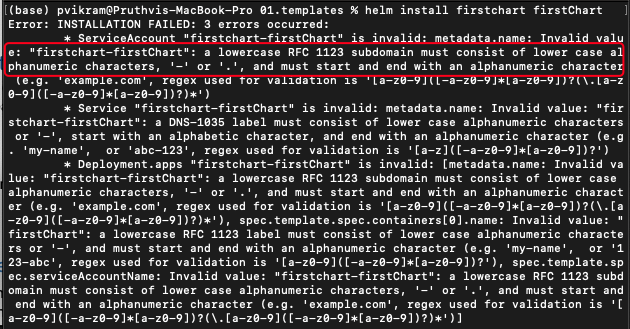


| Name | Type | Description |
| --- | --- | --- |
| Chart.yaml | File | Metadata of the chart |
| Charts | folder | Dependency charts are pulled and stored |
| templates | folder | All the yaml files to create the K8s resources are stored here |
| values.yaml | file | All the values to render the chart yaml is stored in this file |
| Notes.txt | file | Release notes for this chart is stored in this file |

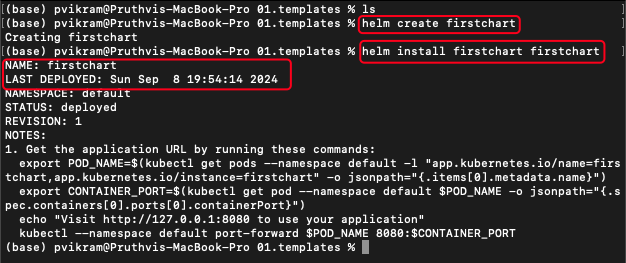
### Installing the chart using the custom chart

In the previous step we created the custom chart. Now lets install the chart.

$ helm install firstChart firstChart (Assuming that the firstChart is the name of the folder in pwd)

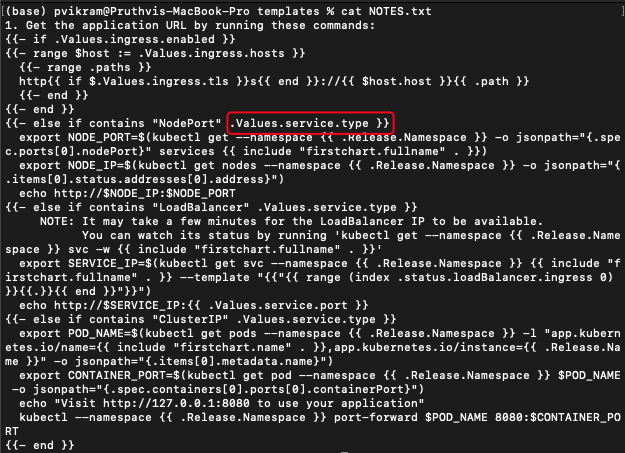


Installation failed because the kubernetes do not allow Capital letters in the names of the resources. To fix this, we need to re create the structure with firstchart instead of firstChart

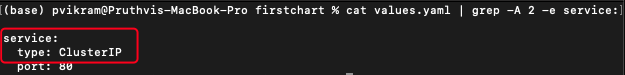


#### How Notes is rendered?

The NOTES.txt holds the information with placeholders. The placeholders are replaced with the values from values.yaml.



.Values.service.type is rendered from the value in the values.yaml



## Understanding Files/Folders

### Understand Chart.yaml

apiVersion: v2

name: firstchart

description: A Helm chart for Kubernetes

type: application

version: 0.1.0

appVersion: "1.16.0"

Chart.yaml holds the metadata of the chart. Artifacthub and other artifact hubs use the information from this file.

There are three important elements

1. apiVersion: Determines the format of the document. Helm 3.0 uses the v2 version.
2. name: Name of the chart
3. version: Version of the chart. Start from 0.1.0. When any change is made to the chart, value can be bumped.

Rest of the elements are optional.

1. description: used to describe the chart
2. type: Application / Library. When set to Library, only functions that can be used by other charts are defined. It won’t have any templates.
3. appVersion: Version of the application that is packaged through this chart.

Optional Elements:

1. icon: Link to image file remote or local
2. keywords: List of keywords that define this project. Like db, mysql, etc…
3. home: URL to the project on the internet. Like the home page, etc…
4. sources: URLs to different source information. This is an YAML array

sources:

- <http://pruthvivikram.com/helm>

- [http://pruthvivikram.com](http://pruthvivikram.com/helm)

1. maintainers:

- name: pruthvi

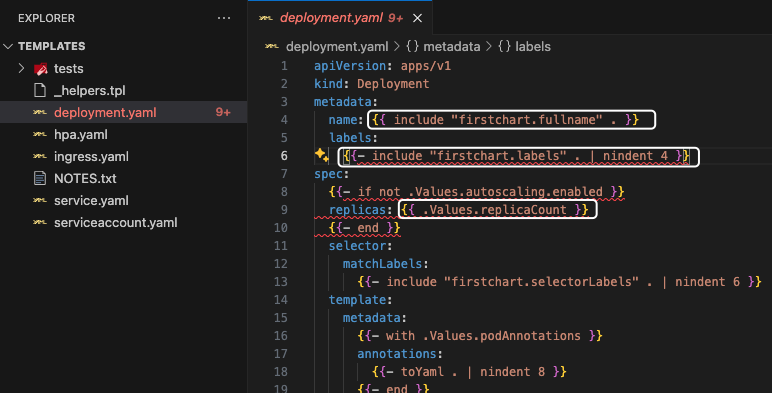
email: [test@gmail.com](mailto:test@gmail.com)

- name: vikram

email: [tes1t@gmail.com](mailto:test@gmail.com)

### Templates Folder

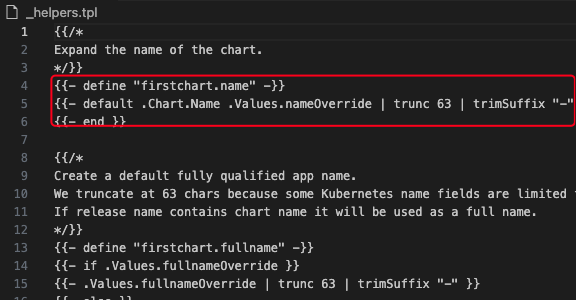
Heart of Helm charts. Yaml files are included in this folder with placeholders whose values are replaced using the values.yaml file. This is the google go templating structure. Using this templating structure, you can also specify conditions as well.



### Understand Helpers.tpl file

If you want to define any methods that can be used across all the template files, you can define them in this \_helpers.tpl file

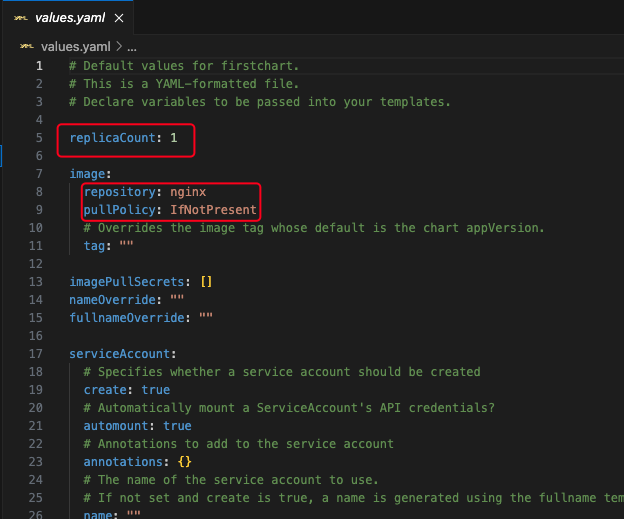
Sample function to expand the name of the chart.



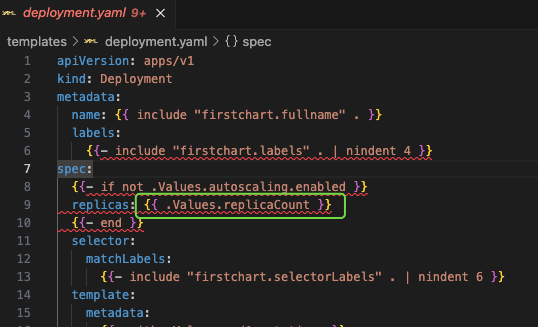
### Understand Values.yaml file

In the values.yaml default values are already present based on the chart. You can replace/change the defined values.

This is how the values are defined.



How defined values are used?

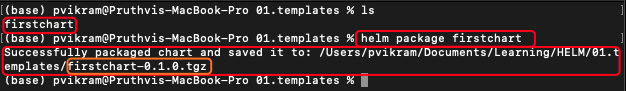


This is how the values file are referred in the deployment files.

## Helm Packaging

This is the process of packaging the created chart to repositories, hubs, etc so that it can be install across different environments like QA, Staging, Production, etc… through CI/CD pipelines, etc…

$ helm package firstchart



Being at the root level of the chart folder, run the above command. This will generate .tgz file which will have all the required files along with the version of the chart.

Helm install command also works with this .tgz file. Highlighted in orange is the format in which the command expects.

### Package dependency update

$ helm package firstchart **--dependency-update**

$ helm package firstchart **-u**

This command downloads the latest dependencies and updates the package. The latest version of the dependency charts are stored in the charts folder before packaging.

### Package Location

By default the final package (.tgz) file is stored inside the present working directory. If you want to change the location use -d flag.



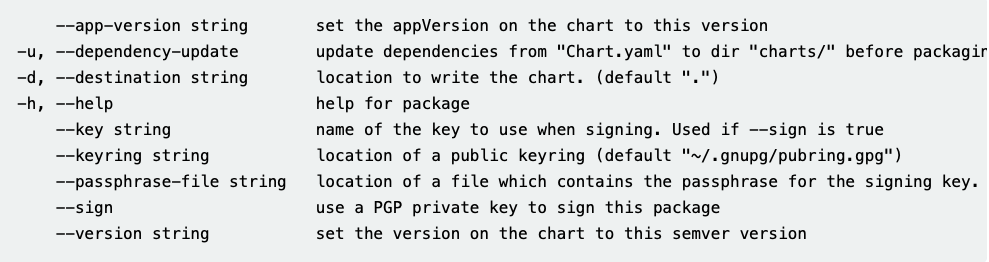
$ helm package firstchart --destination <path>

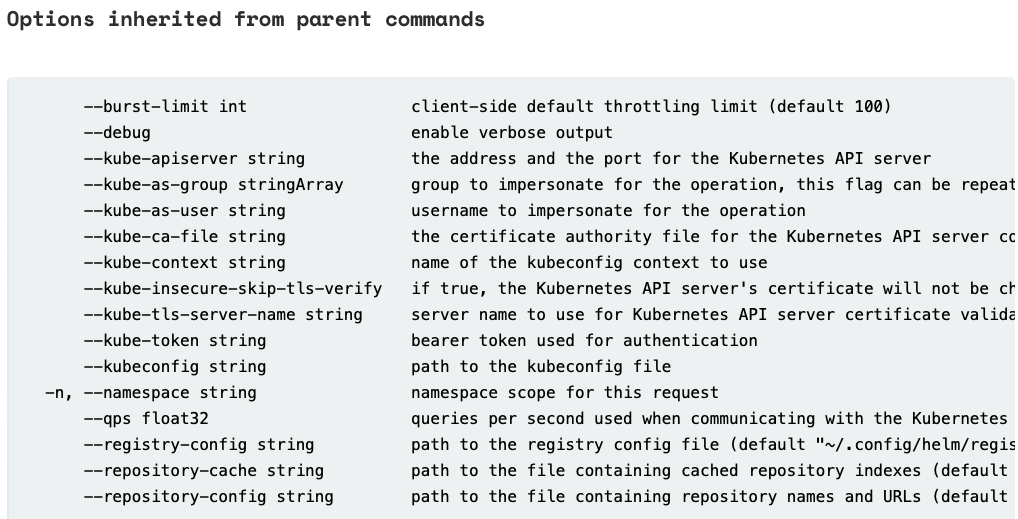
$ helm package firstchart -d <path>



### Other Options for packaging

Visit <https://helm.sh/docs/helm/helm_package/> for other packaging options.





## Helm Lint

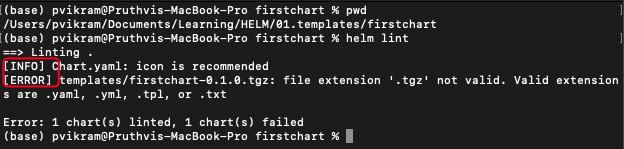
This command scans through the code and fixes syntactic and indent issues with the templates or commands.

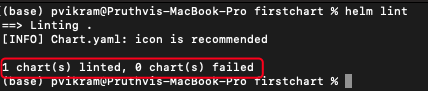
There are three types of errors

1. INFO
2. WARNING
3. ERROR

INFO and WARNING returns 0 error code.

ERROR returns a non-zero error code.





## Helm Ignore

If you want to ignore a file from packaging you can add those files or patterns in the .helmignore file

[https://helm.sh/docs/chart\_template\_guide/helm\_ignore\_file](https://helm.sh/docs/chart_template_guide/helm_ignore_file/)

# Helm Template & Go Lang

Template uses go lang to control what resources are created in clusters. Not whole golang is required to develop templates.

## Template Actions

Actions start and end with two flower brackets. We use these to define variables, invoking functions, etc. Anything outside the brackets are rendered as is. The content inside the brackets are evaluated and rendered.

{{ ACTION }}

{{ include “firstchart.fullname” . }}

### Action element - White space removal

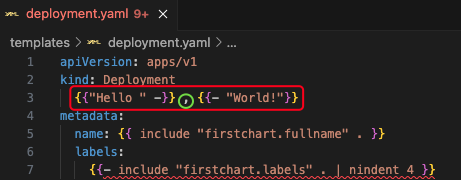
{{- include “firstchart.fullname” . -}} : - is used to remove any leading or trailing spaces. Since yaml is stricts about spaces and indentation, hyphens are used to remove additional spaces.

Only one hyphen - is required for one set of braces. Either at the end or at the start. {{- “”}} or {{“” -}}.

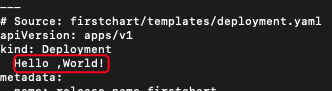
### Action element - String concat

If you have two string in action block, you can separate them using a comma.

{{- “Hello ” }}, {{ “World” -}}

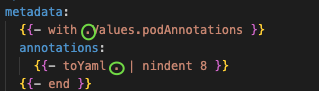






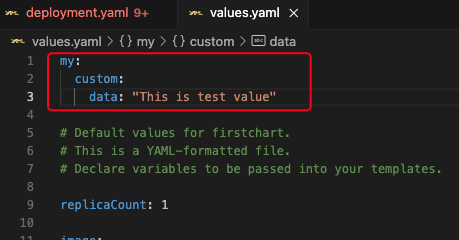
## Template Information

When helm renders, it passes all the information represented by a dot (.) object.



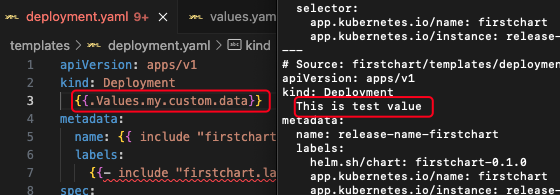
### Template Info: .Values object

Ex: The value defined in the values.yaml can be accessed using the . object



This my.custom.data value can be accessed using dot(.) object as follows.

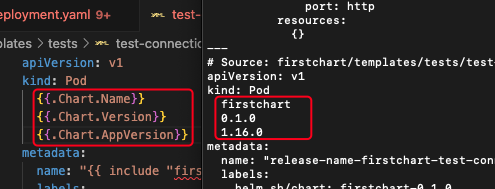
{{.Values.my.custom.data}}

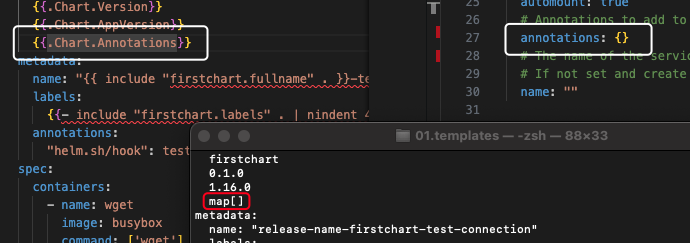


##### .Values represents the values.yaml file and its contents.

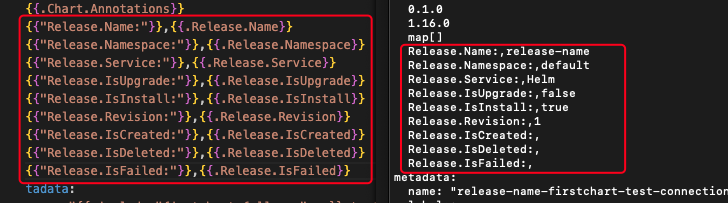
### Template info: .Chart object

You can use the chart information from charts.yaml file using .Chart object.



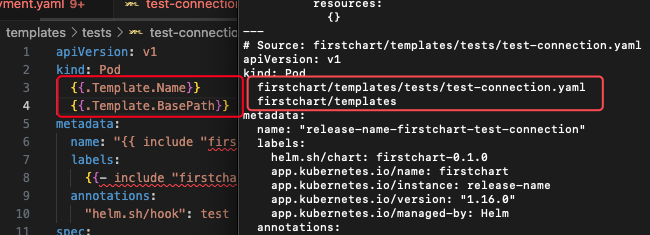


### Template Info: .Release

This will give the release information. This information can be used to display as information or in the conditional statements.

### Template Info: .Template

You can get the template information using below



### Template Info: Pipelines



Pipe symbol is used to pass the output of the previous function or variable to the next command/function

Here in the above example, toYaml . output is passed to nindent function with 8 as an attribute value.

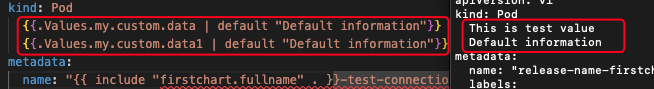
## Template Functions

Functions are used to perform set of operations. They can be used anywhere in the template file. Some of the functions are nindent, toYaml, etc.

You can find the list of functions in the below URL.

<https://helm.sh/docs/chart_template_guide/function_list/>

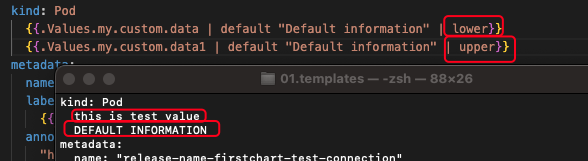
### Default function



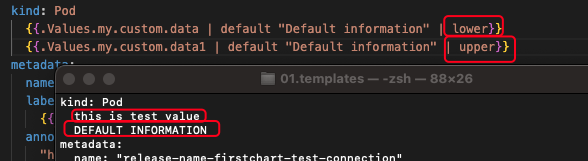
Custom.data has a value hence it is printed as “This is test value.”

Custom.data1 has no value, so it printed the “Default information” as output.

### Upper function



### Lower function

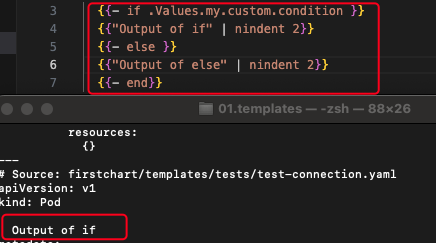


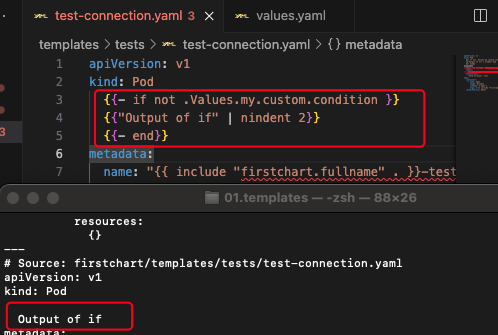
### Quote function



## Template Conditions

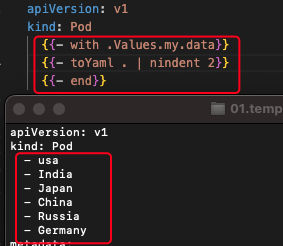
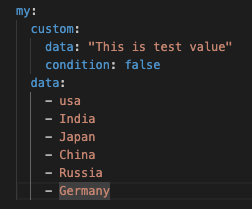
### Conditions: IF ELSE



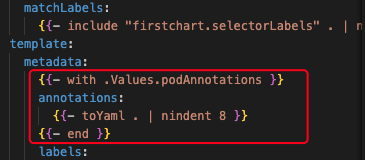


### Conditions: With

You can run each value from a Yaml array using with clause.

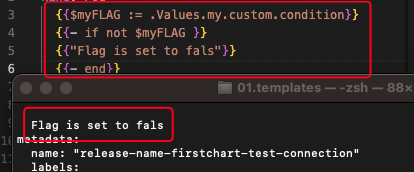


You can also use else condition with with condition. If the array is empty, then the else clause kick in.



Hare, if the podAnnotations array is present, then the annotations: is printed with each array value. If the array is empty, then the whole portion is skipped.

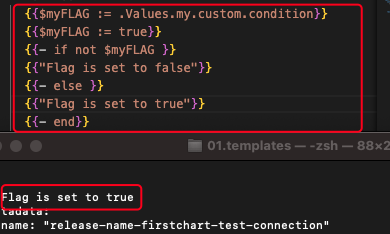
### Conditions: Variables



Variable is created in {{$VARNAME}} format.

To assign a value we use :=

To reassign a value we use =



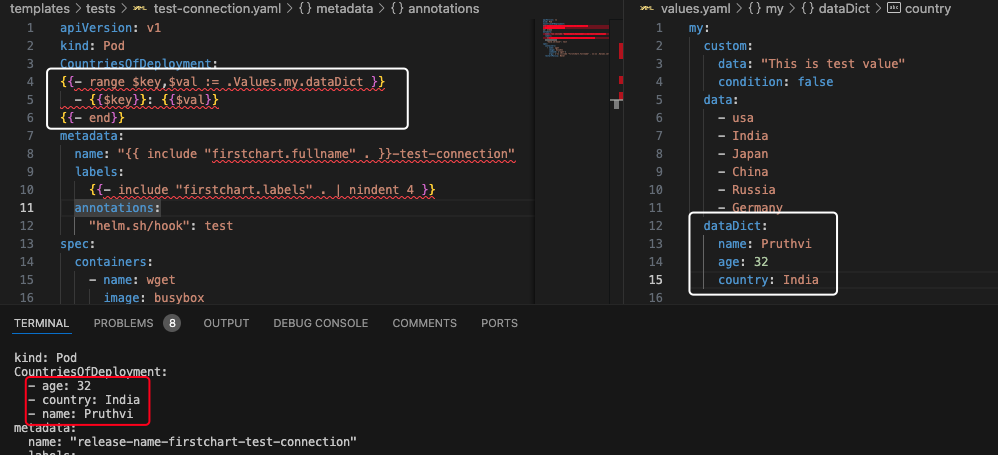
## Template Loops

### Loops: For loop



Range is used as for loop in Helm templates. Unlike with, range process one element at a time.

### Loops: Dict types



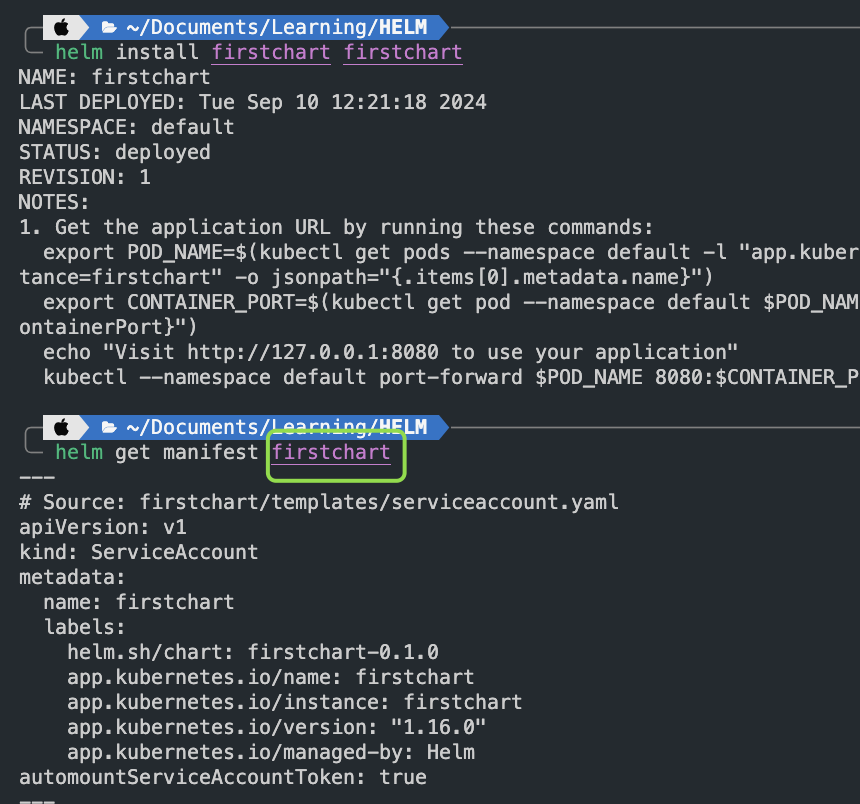
## Debugging the Templates

You can use --debug to check if there are any errors in the code or template. You can use the dry-run to validate the syntactical and semantical errors in the code.

You can also perform the same using the helm template. But, the only difference is that the template command doesn’t do the validation. For example, if you remove the **kind: Deployment** and run the helm template firstchart, It will print the data as it’s not performing any validation. But if you do the same with **helm install --dry-run firstchart**, It will fail.

## Helm Get Manifests

Very useful to get the manifest while doing the installation. The manifest sent to kubernetes after rendering all the template files can be retrieved using this command.



## Helpers.tpl file

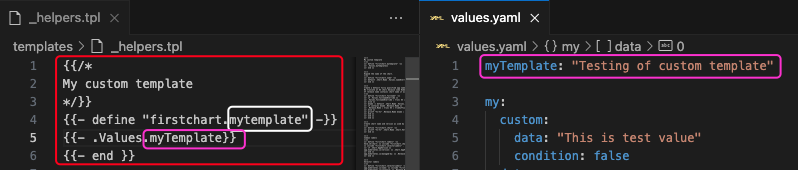
.tpl stands for template. This host several functions that are used across the templates.

{{- define “firstchart.name” -}} is the template of the defining a function.

{{- include “firstchart.name” -}} or {{- template “firstchart.name” -}} is used to reference above defined namespace.

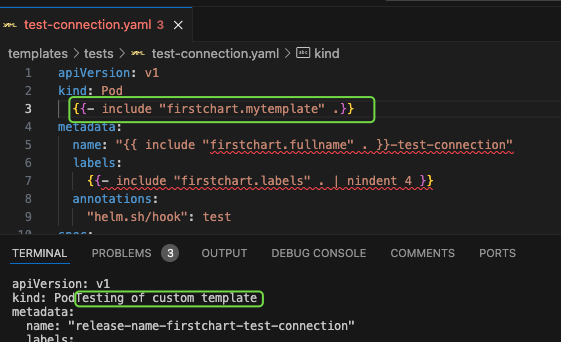
Diff is with template, you can not use output to pipe to another function, its only possible with include.

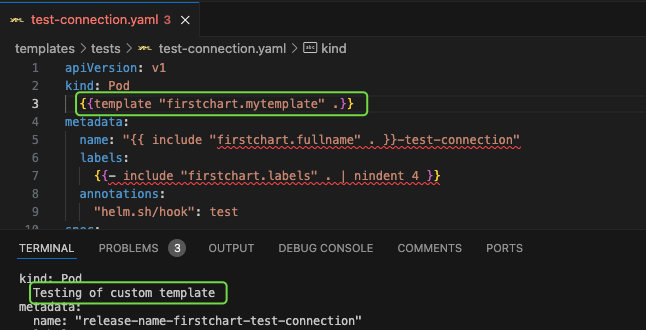
### Define custom Template



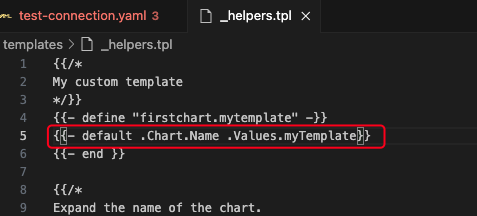
Now you can use firstchart.mytemplate anywhere. And the value is referenced from values.yaml.

### Use the template

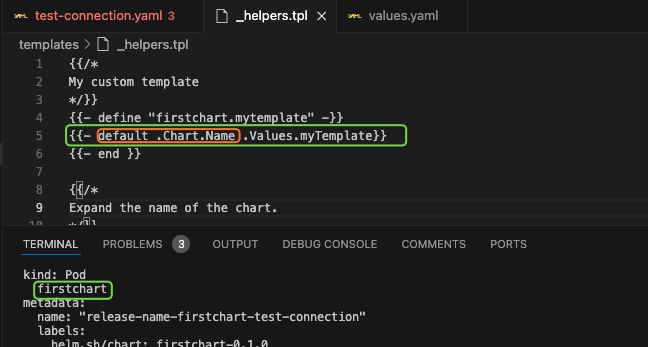




### Passing Default Value



This way we can set a default value. If the values.yaml is missing myTemplate variable and values, it will return the default value here in this case, Chart Name



# Advanced Chart

* Dependency management
* Exchanging values across dependency charts
* Helm Hooks
* How to test your chart?

## Dependencies

In the chart.yaml, add dependency section

dependencies:

* name: mysql

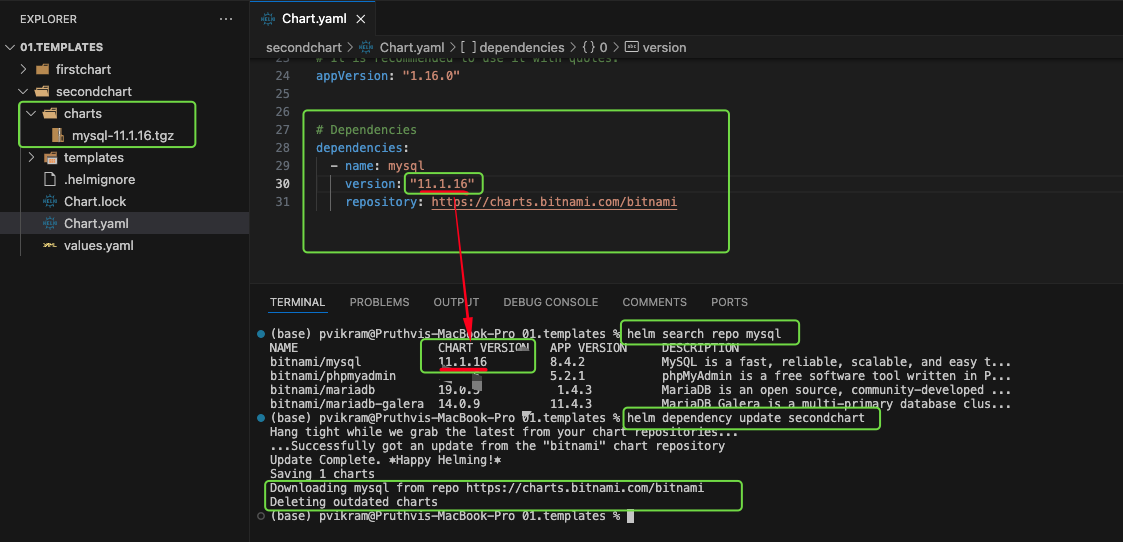
version: “8.8.6”

repository: “<https://charts.bitnami.com/bitnami>”

Now the dependencies have been defined. Let’s update the dependency

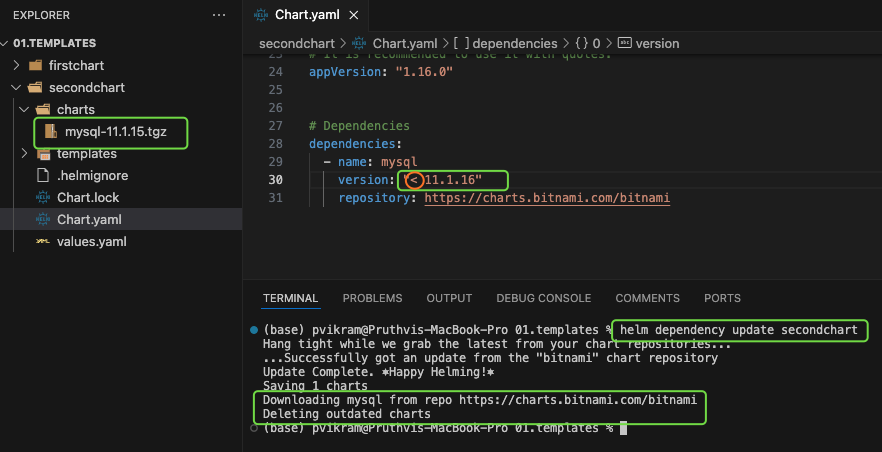
$ helm dependency update

By now, the dependency chart has been downloaded and stored in the charts folder from the central repository.



### Dependency Version Range

Instead of hardcoding the chart version in the dependency, we can mention range.

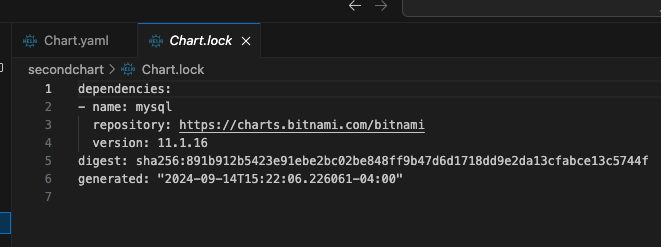


Other options

~ 1.3.4, ~ 2, ~ 2.3, >= 1.2.3, < 1.4.0, >= 2, >= 2.3, <2.4, <3

### Chart Lock

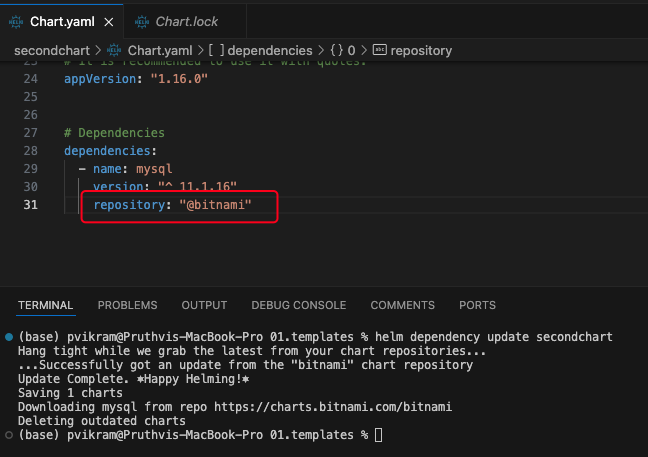
Helm also generates a lock file for charts. Its called chart.lock. This file consists of the dependency list along with the version of chart.



### Using RepoName instead of URL

If we already have a repository thats already downloaded to local, we can mention the name directly instead of URL.

“@bitnami”



### Conditional Dependency

In the values.yaml, define a property

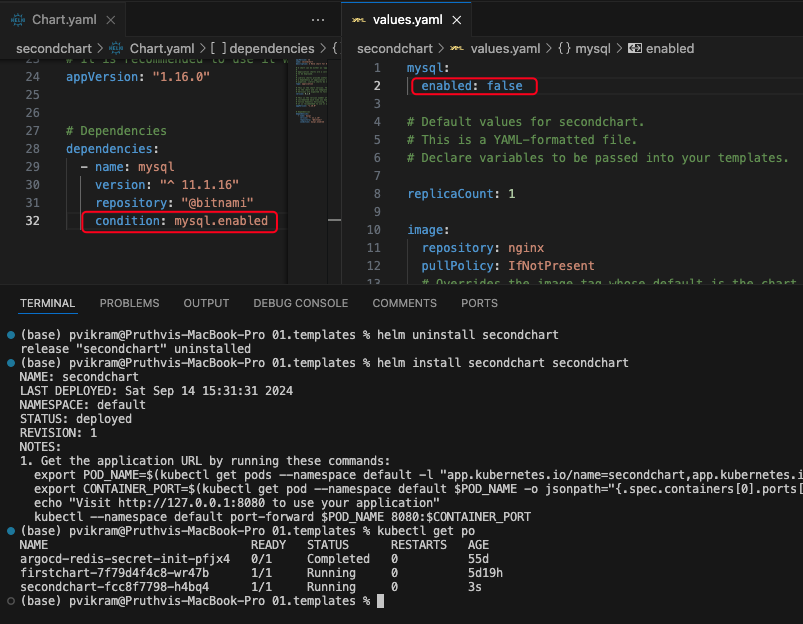
mysql:

enable: false

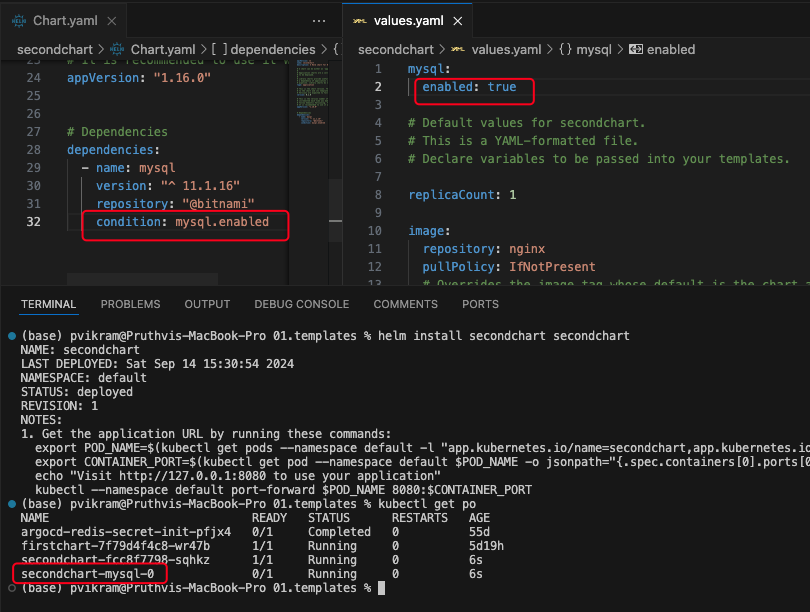
And in the chart.yaml, set a variable for dependency to check the condition

condition: mysql.enabled

Below is when the mysql is disabled



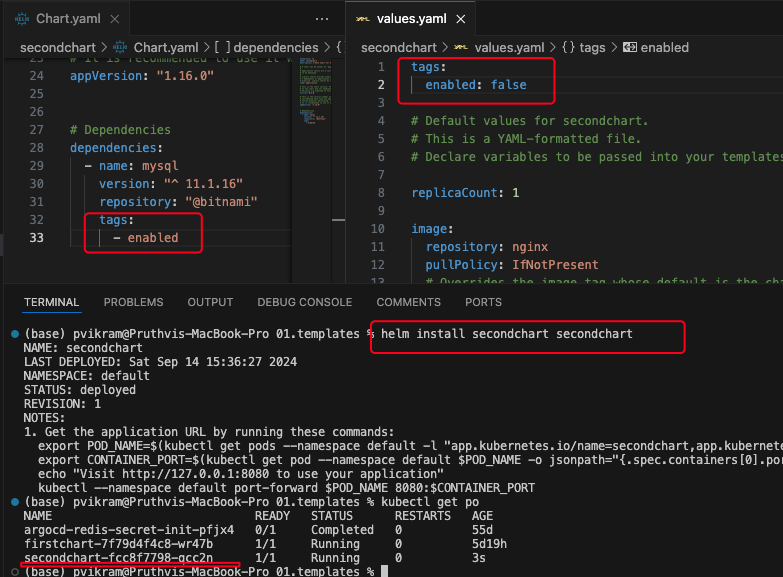
Below is when the condition is set to true.



### Multiple Conditional Dependencies

If you have multiple conditions, instead of repeating values in values.yaml you can use TAGS

In the below screenshot, tags are used to disable mysql installation.



## Passing Values to Dependencies

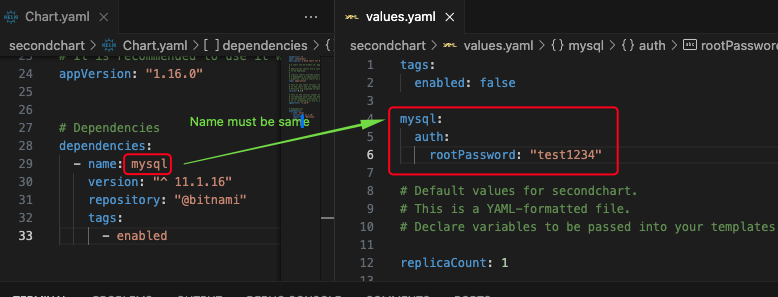
Usually for dependency charts, the default values are passed. But if you want to customize your chart, use the values.yaml and chart.yaml.

Values.yaml

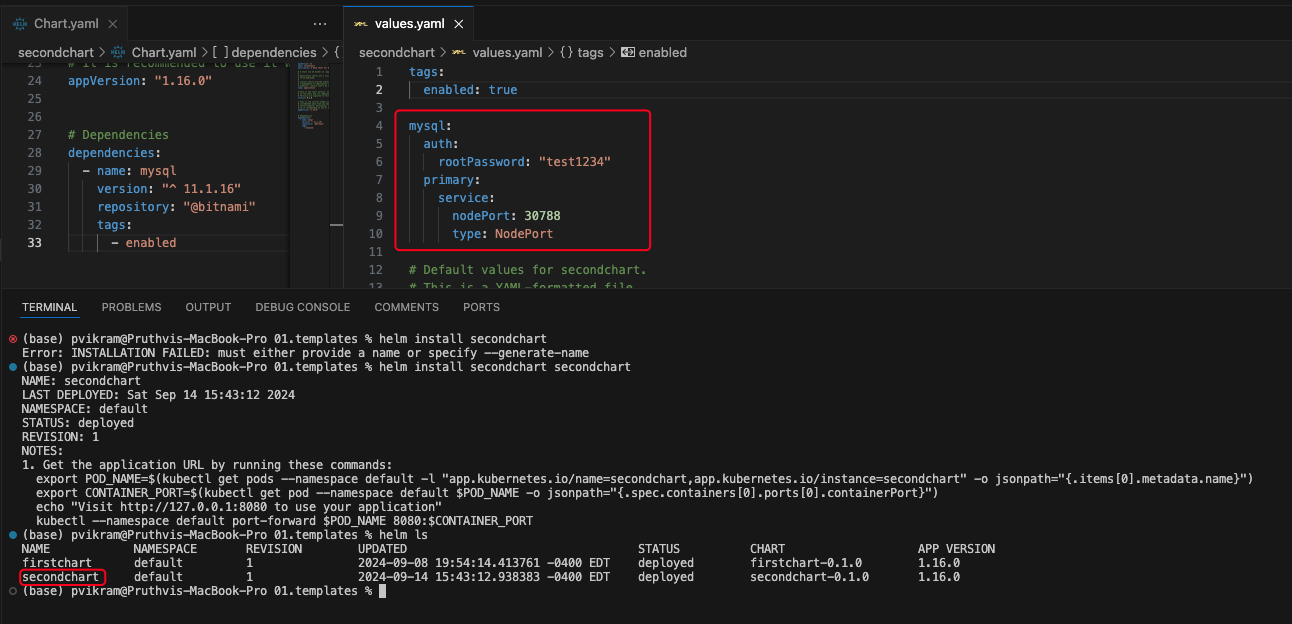
mysql:

auth:

rootPassword: test1234



Let’s test the custom values





### Read Values from Child Chart

We can use import-values element in the chart dependency to import values from child to parent chart.

### Explicit export and import

Usually the child chart exports few values explicitly. Mysql wont do that. But if the child chart’s values.yaml has a property like below

export:

service:

port: 8080

And the chart dependencies has

import-values:

* service

With this setup, the service can be used in any of the parent child’s yaml files as if they are added in the values.yaml file.

{{- .Values.service.type .}}

### Use values that are not exported

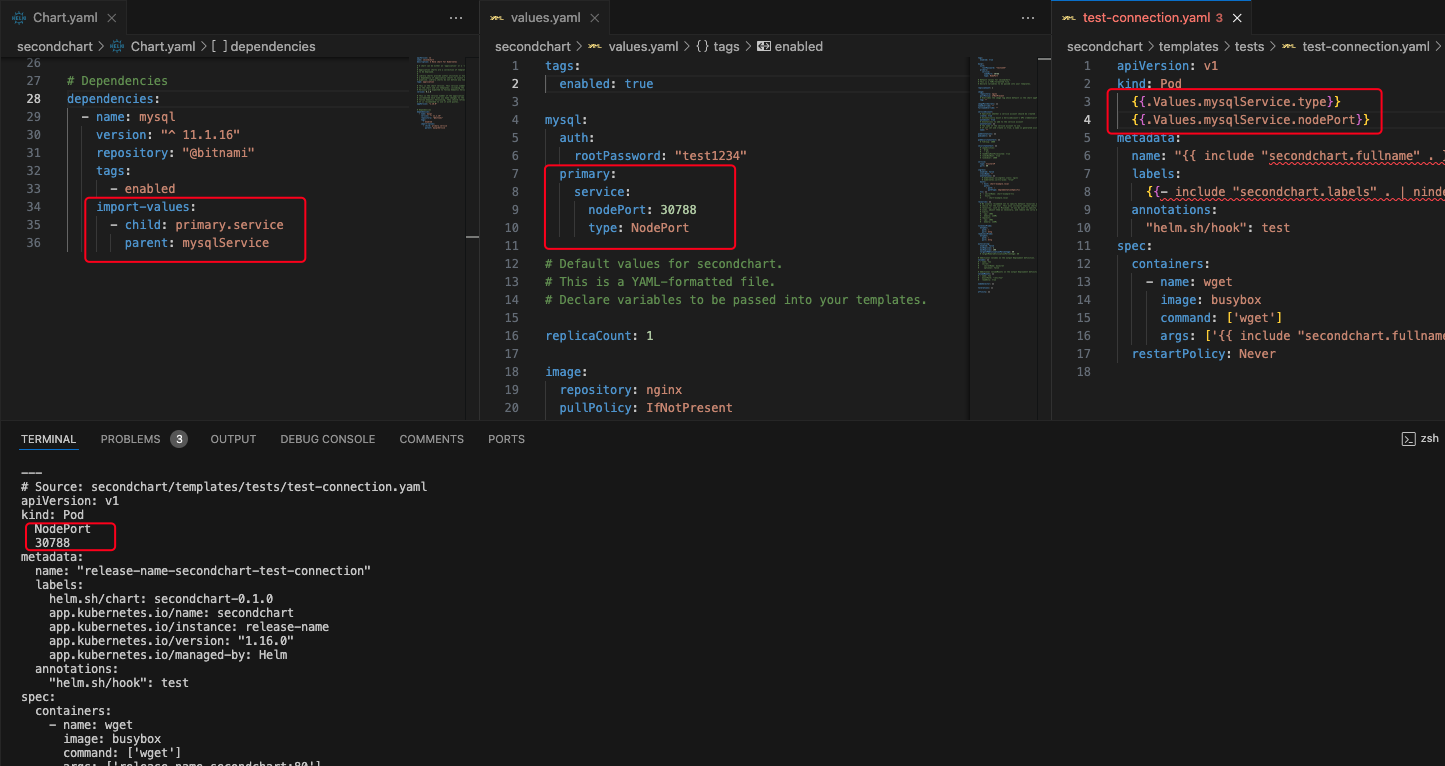
NOTE: If the child’s values.yaml file doesn’t contain export, then the child dependency doesn’t export any values. But even then we can use the values.

In chart.yaml, instead of element name, use below element configuration

import-values:

* child: primary.service
* parent: mysqlService

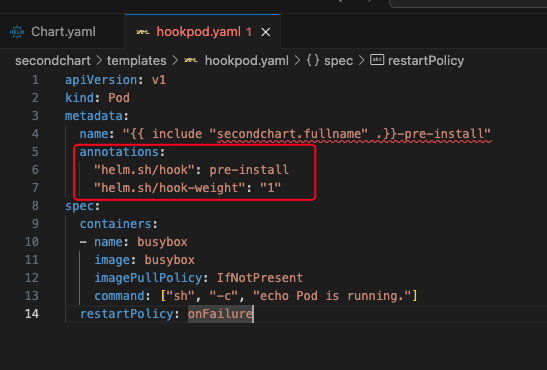
With this configuration, you can get use mysql’s service details in the parent yamls.



## Hooks

Used to take special actions during the release process. The file is created under chart’s templates folder.

hookpod.yaml is the file that can be used to create any kubernetes resources



In the above hookpod, a busy box pod is getting created. The hook is identified based on the annotations that are added in the yaml file.

helm.sh/hook is used to decide when to run the hook.   
pre-install : Before any of the other resources are created, the busy-box is created and run.

You can deploy any type Kubernetes resources but not only the pod.

### Options:

1. Pre-install : before release
2. Post-install : After installation of release
3. Pre-delete : before uninstall
4. Post-delete : After uninstall
5. Pre-upgrade
6. Post-upgrade
7. Pre-rollback : Before the rollback of any installation
8. Post-rollback
9. Test : To perform tests.

Other property

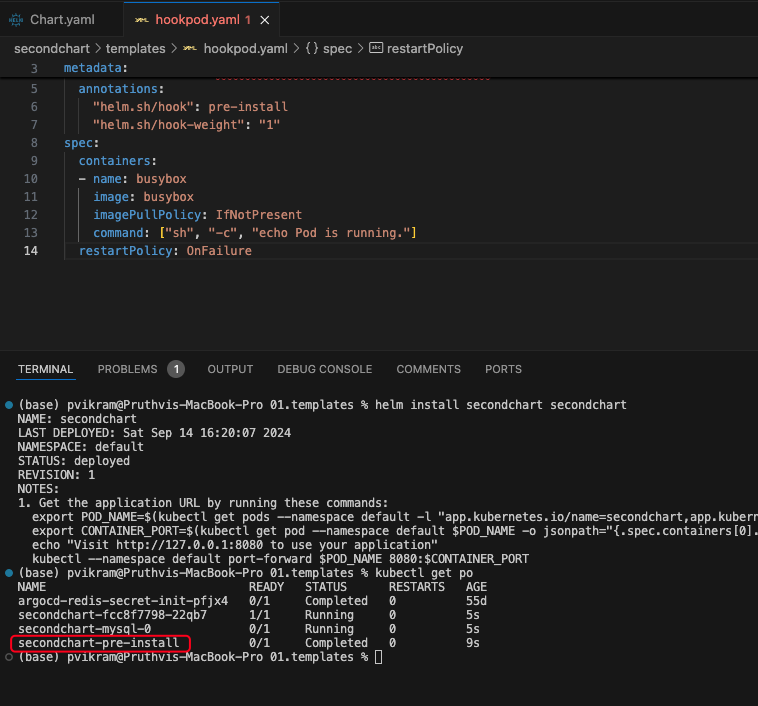
1. Before-hook-creation
2. Hook-succeeded
3. Hook-failed

Weight

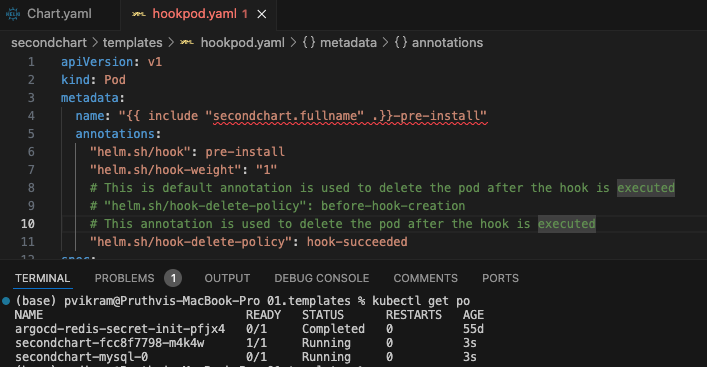
Ascending order. If you have multiple hooks, you can define this.

### 

### Create and use hook



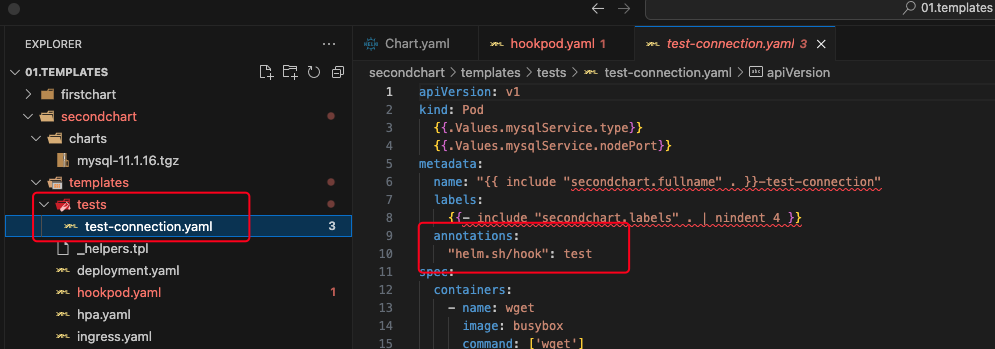
With hook-succeeded in the delete-policy, the pod is deleted when the hook is successfully created.



## Testing your chart

By default when a new chart is created, test-connection.yaml file is created.

By default a test type hook is added in the annotations.



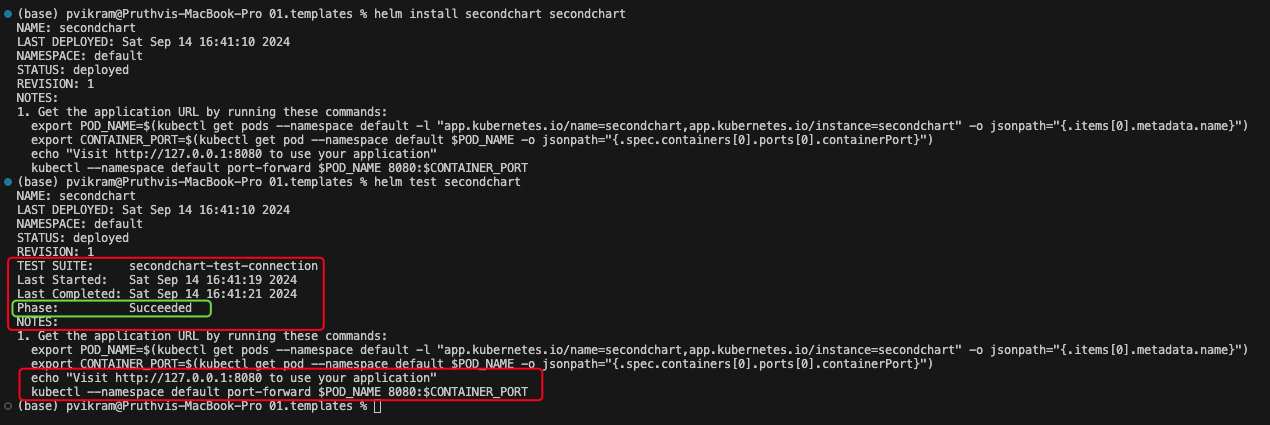
In above case, the test is run on the nginx pod, wget command is used to test the connection. If the wget command returns 0 its successful and if it returns non-zero code, then its a failure.

### Perform testing on a chart

Helm test is conducted on a release but not at the template level. So run the installation first.

Then run the below command to run the tests.

$helm test secondchart



# Assignment 3

Create a chart dependency, Package, Install the chart.

Dependency : tomcat

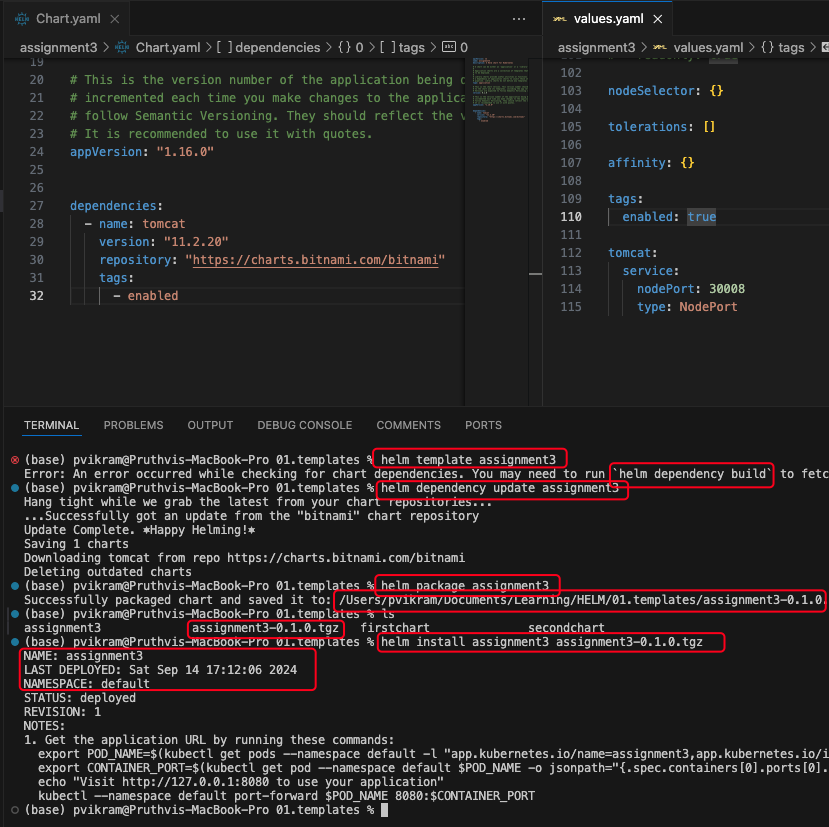
Custom values:

tomcat:

service:

type: NodePort

nodePort: 30007



# Repositories

Helm repos are simple http repositories. In this you will learn how to

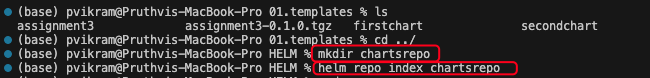
* generate indexes
* Setup local repo using Python web server
* Host a public chart repository using GitHub pages
* Launch OCI registry locally using Docker

## How to host local chart repository

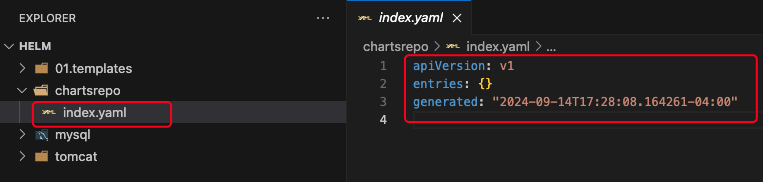
Create a folder chartsrepo

Or use command $ helm repo index chartsrepo/

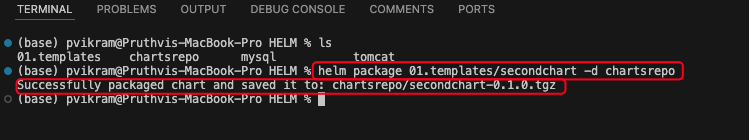
This will create index.yaml file.



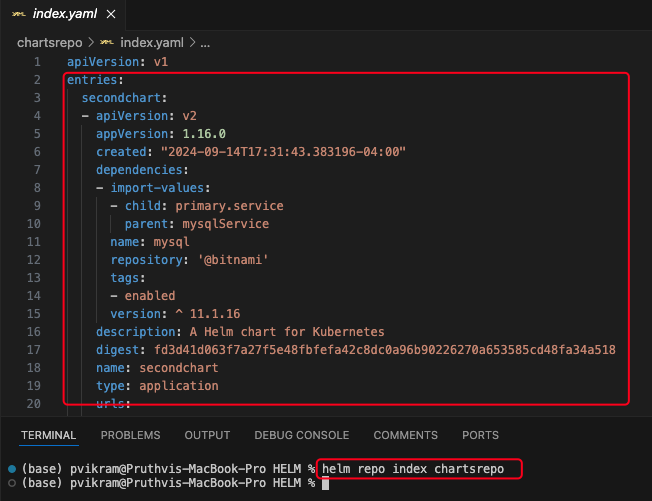
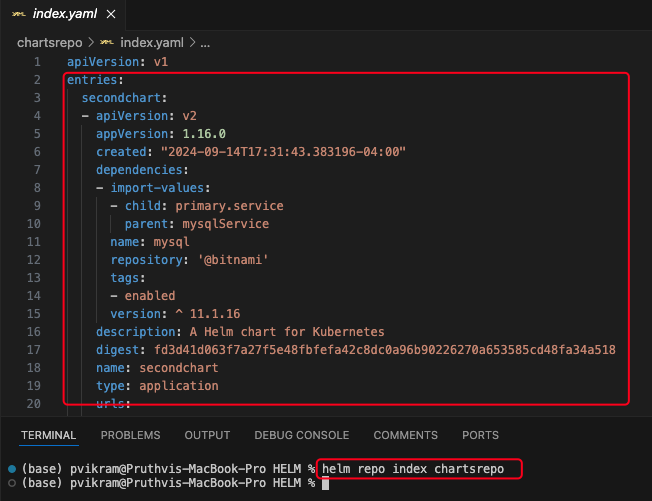
Index.yaml contents



Now package and put the chart in the chartsrepo folder.



Now update the repo index and observe the changes.



## Host webserver using python

Run the http server in the chartsrepo directory

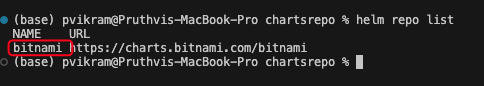
$ python3 -m http.server --bind 127.0.0.1 8080



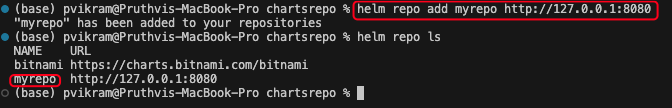
## Using the local repository

First add the local repository

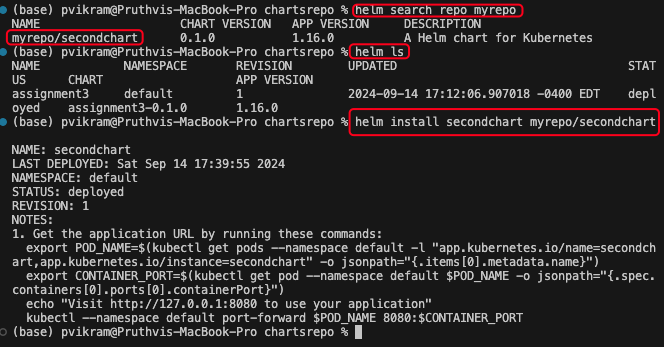
Currently we only have the bitnami repo



Add the local repo



Install from repo



## Installing using HELM PULL

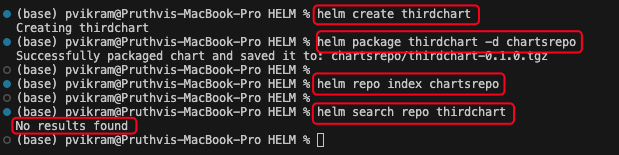
Instead of pulling the chart from the local or remote repository, we can first download the chart locally and then install it.

Pull the chart locally.

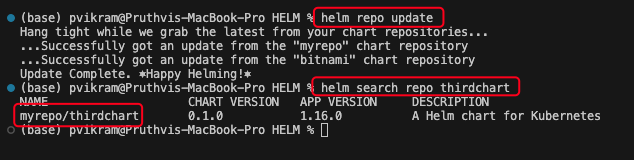


## Update Repositories

Sometimes when the new chart is added to the repositories, helm search command won’t pickup as the helm search command uses a local cache to search for the chart packages.



In order to update the local chart repo use



With this, the local cache is updated and we can search the new cache.

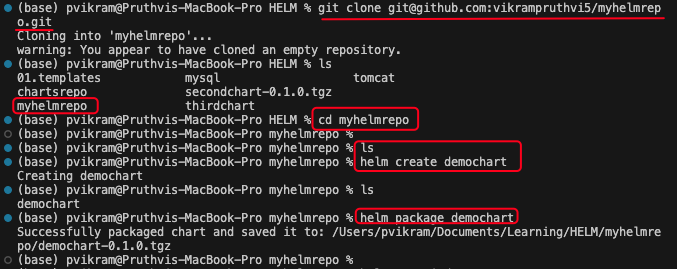
## Github Pages for hosting your charts

1. Create a new repository
2. Set access to public
3. Readme and all are optional
4. Goto command line
5. clone the repository locally
6. Create new chart in the gir repo folder
7. Create package from the chart
8. Index the chart in the same folder
9. Perform git add, commit and push

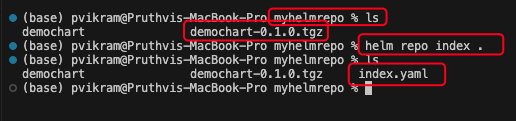
To use this git repo as pages, goto github repo settings -> pages -> select source to master

By this, github repository is configured as webserver and all the changes are published.

Once saved, the site url is generated and you can use it as a helm repository.



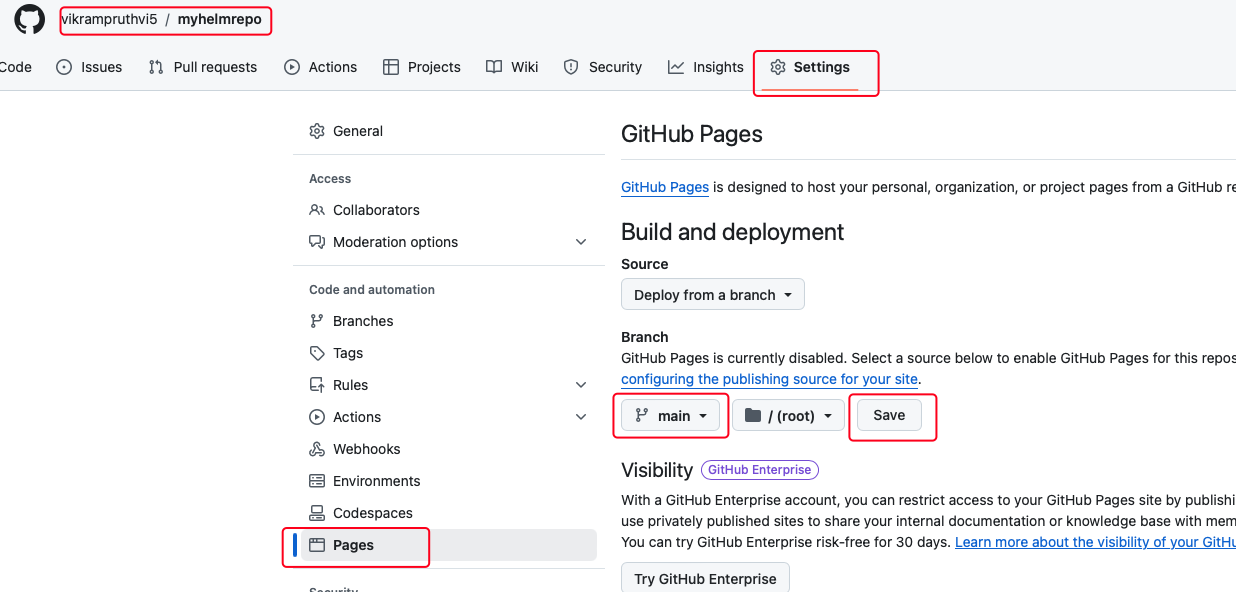
Index the repository

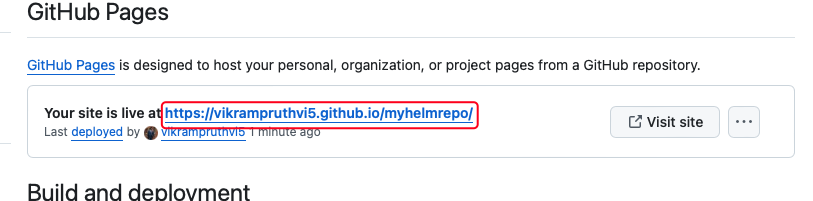


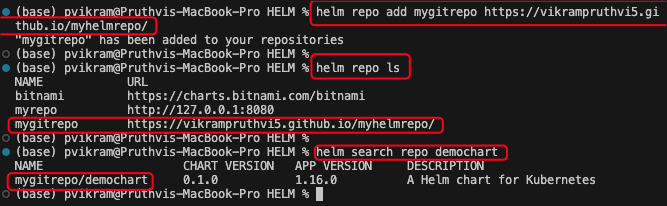
Push changes to git



Change the settings of the repository



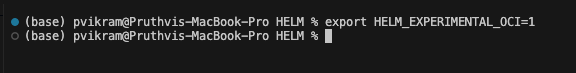
Upon saving copy the URL and add repo in your local



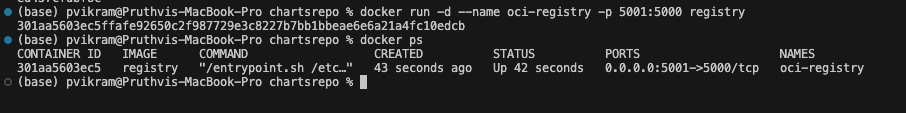
## OCI Experimental

Using the Open Container Initiative to host helm repositories.

First enable the HELM\_EXPERIMENTAL\_OCI. You can do it by configuring an environment variable.



Run local OCI registry using Docker image



Now the OCI registry is created. Let’s create the package from a chart



Commands

| Push chart | helm push ocichart-0.2.0.tgz oci://localhost:5001/helm-charts |
| --- | --- |
| Show all | helm show all oci://localhost:5001/helm-charts/ocichart --version 0.1.0 |
| Pull chart | helm pull oci://localhost:5001/helm-charts/ocichart --version 0.1.0 |
| See template | helm template myrelease oci://localhost:5001/helm-charts/ocichart –version 0.1.0 |
| Install chart | helm install myrelease oci://localhost:5001/helm-charts/ocichart –version 0.1.0 |
| Upgrade chart | helm upgrade myrelease oci://localhost:5001/helm-charts/ocichart –version 0.2.0 |
| Login to registry | helm registry login -u username oci://localhost:5001 (Only for remote registry) |
| Logout from registry | helm registry logout oci://localhost:5001  *(Only for remote registry)* |

# Chart Security



When it comes to charts, provenance and integrity are important.

Provenance : Ensuring that the chart is from the right provider

Integrity : Ensuring that the contents of the chart are not altered.

Helm uses inbuilt PGP(pretty good privacy). Where private key and public key are used.

The private key is used to encrypt the package.tgz file which creates package.tgz.prov file. Both these files are then pushed to the repository.

The public key of the provider is used by the user to ensure that the signature of the downloaded chart are same using the command

$ helm install --verify

## PGP Installation

We need a public and private key

* Download GNUPG from gnupg.org
* Perform the installation (For mac use brew install gnupg)

Run the command

Gnu privacy guard(gpg)

$ gpg –version

This is where all the keys are stored. You can use these keys to sign and verify the chart

### Generate the key

$ gpg --full-generate-key

Select the type of the key (default is RSA)

Select the size : 3072 is default

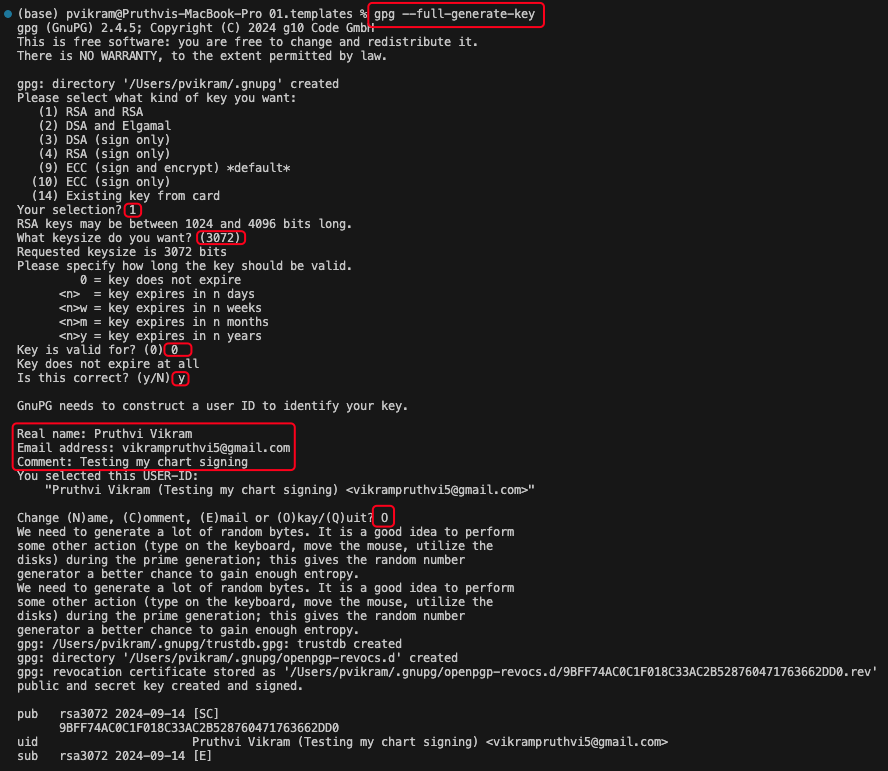
Select the expiration: 0 is for never expire

Give the real name : YOUR NAME

Email address : Email is used as alias to the key

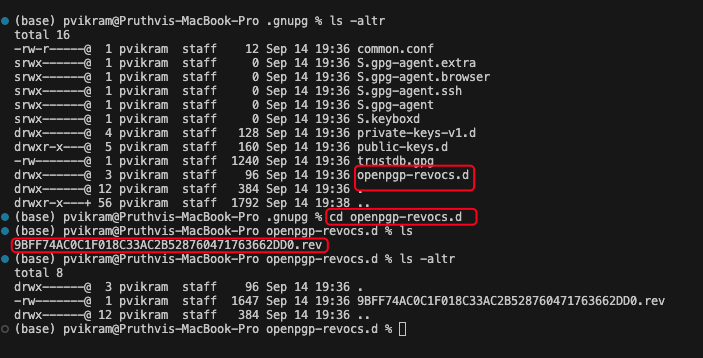
Comment : optional

Finally, you have to select the passphrase to access the keys.



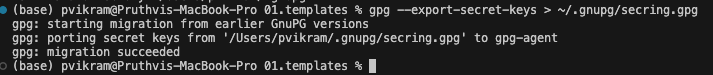
By now the keys are stored under ~/.gnupg/

As of 2021, kbx files are generated by pgp commands. But we need to convert them to .gpg files for helm to use.



Use the below command to perform the conversion.

$ gpg --export-secret-keys > ~/.gnupg/secring.gpg

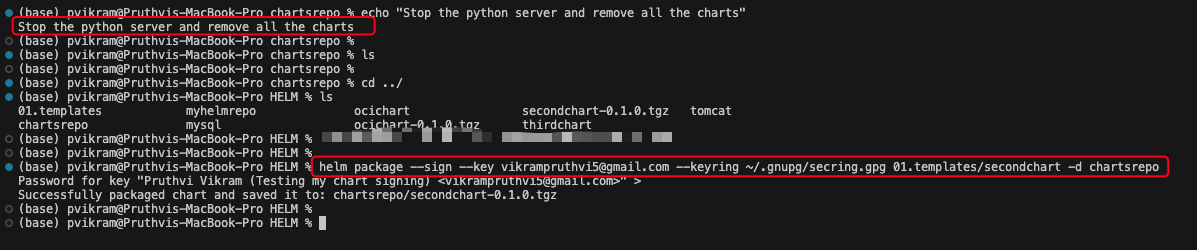


Now both public and private keys are part of the gpg file and ready to be used by helm

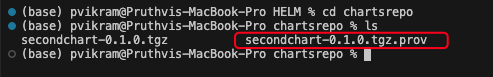
### Signing Charts

Let’s sign the chart while packaging

* Create a new chart
* Run the command to package along with signing



Two files are created.



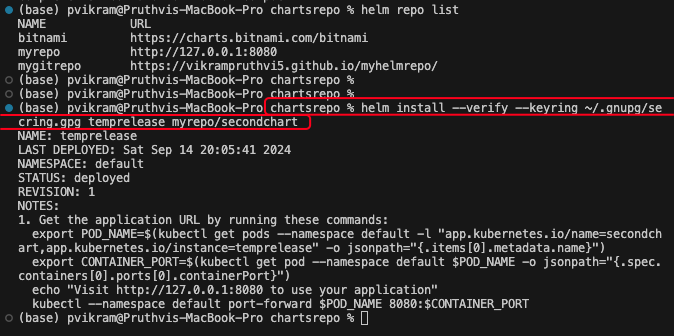
The prov file has the signing information.

### Verify the Signature of the chart



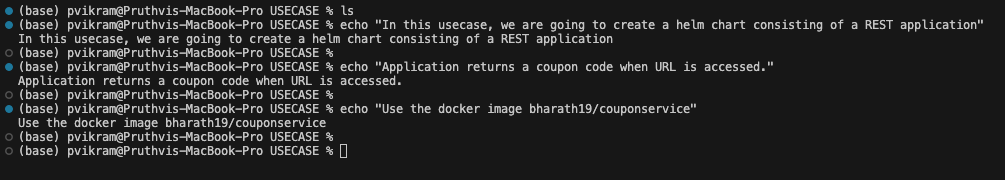
we create an index in the chart repo and start the local repository server.

With the below command, we can verify and install the chart when pulled from the repository.



If the installation fails, then it implies that the chart signature is not verified. You would see an error.

# Use Cases



For this use case, we will be performing below set of actions

* Create chart and configure image
* Update Deployment with probes
* Configure NodePort, change from default service
* Add MySQL dependency to the chart
* Pass values for the MySQL
* Configure ConfigMap

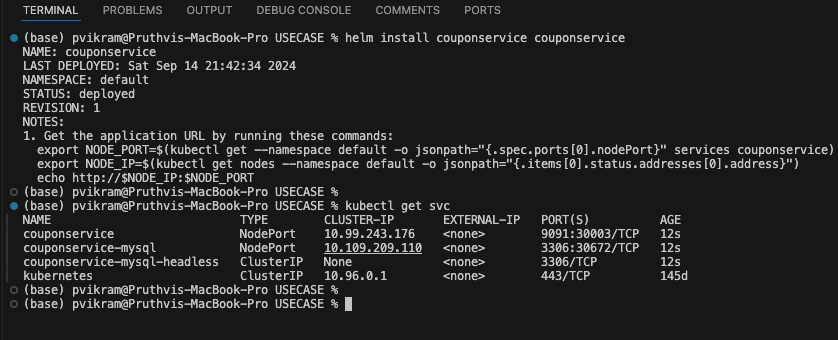
## Create chart and configure image



We dont need hpa(scaling) ingress and service account. So remove those yamls.

For all the steps, go through the commits and check the changes.

GIT URL:



# HELM Starters

Skipping the topic as it’s not important and is not usually used in corporate.