Introduction to Helm and the problems that it solves

In the previous section, I gave a very quick introduction about Helm.

Now, inside this lecture, let us try to understand more details about Helm.

**If someone asks you what is Helm, you can simply reply saying that Helm is a package manager for Kubernetes.**

**The main objective of Helm is to help developers and DevOps team members to manage Kubernetes projects and deployments by offering a more efficient approach in handling the Kubernetes manifest files.**

Regardless of how many microservices you have inside your microservice network, Helm is going to make your life easy without helm like we discussed previously, we need to maintain lot many Kubernetes manifest files. Without helm like we discussed previously, we need to maintain all the Kubernetes manifest files like for deployment, service, config map, for each microservice that you are going to deploy inside your microservice.

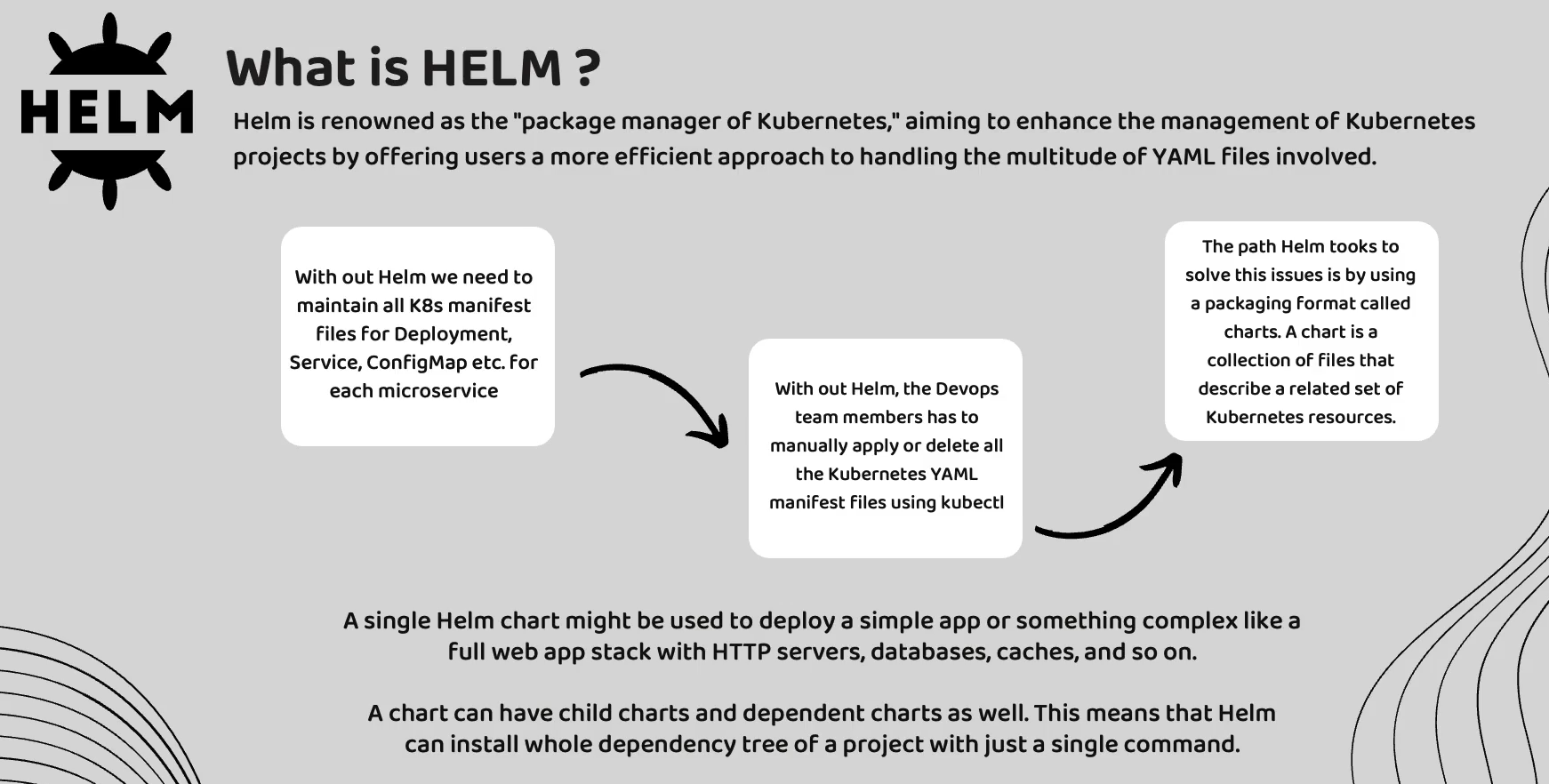
And apart from maintaining these Kubernetes manifest files, your DevOps team members also they should manually apply or delete these Kubernetes manifest files with the help of **kubectl** commands.

But like I said, if we use Helm, we can overcome these challenges.

And here you may have question like how Helm is going to help us to overcome these challenges.

For the same helm is going to follow a **packaging format called Charts**.

Inside Helm, a chart is a collection of files that describes a related set of Kubernetes resources.



Suppose inside your microservice network, if you have 50 microservices, you can club all these 50

microservices related manifest files into a single component called Chart inside Helm.

So, using the same single chart, we can deploy a simple application or any kind of complex application which can include Http servers, REST APIs, databases, supporting components like cache.

So regardless of how complex your microservice network is, you can always try to create a chart specific to your organization.

And with that, the maintenance of your Kubernetes deployments is going to be super easy.

And one more beauty of this helm charts is, a chart can have a **child charts** or **dependent charts** as well, just like in Java, how we can have parent class or child class or dependent classes very similarly

inside helm charts also, we can maintain the dependent charts and child charts in case if you have dependency on other third-party components, you can try to add the helm charts of those components into your organization charts.

This way you can install whole dependency tree of your project, including your own microservices and dependent components with just a single command.

And like I said, Helm is a package manager for Kubernetes.

And here you may have a question.

**What is a package manager?**

**Package manager** is a component that can help you in installing or uninstalling or upgrading your software packages. Just like how we have helm for Kubernetes, very similarly, we also have other famous package managers inside the industry.

*For example,* ***Pip*** *is a package manager that can help you to install any Python packages.*

*And very similarly for JavaScript, we have a package manager with the name* ***NPM****.*

*Using this NPM package manager, we can install any kind of JavaScript libraries like Angular, React.*

*All these libraries we can easily install and set up inside our local system with the help of this NPM package manager.*

Very similarly, ***helm*** **is the best way to find, share and use software’s built for Kubernetes**.

I hope this is clear.

Now let me try to explain you very quickly on how Helm is going to solve all the challenges that we

have discussed. Without Helm like we discussed, we need to maintain separate Kubernetes manifest files for all the microservices inside a project.

For example, think like I have accounts, loans, and cards, microservice. If I want to access them,

either using external traffic or with the help of internal traffic, I should expose them with the help

of Service object inside Kubernetes.

For the same, I need to create three different Kubernetes manifest files, one for account-service,

and the second one is for the loan-service and the third one for the card-service.

So, if you try to closely observe these manifest files, the skeleton of these files is almost same or

static except a few dynamic values, like the very first element **apiVersion:** which is v1 is going

to be static for all the files.

The same applies for **kind:** object post that will be having **metadata** **name** inside the metadata name,

we are going to have a dynamic value which is specific to this microservice.

Post that and a **spec** and **selector**.

The app name is going to be the dynamic value and the type also

is going to be a dynamic value based upon our requirements.

We can use LoadBalancer or ClusterIP, but coming to the **protocol**, TCP is going to be a static value

and the **port** and **targetPort** will have dynamic values which are specific to each microservice.

So apart from these few dynamic values, the whole skeleton or template of the Kubernetes service manifest file is going to be the same.

This is where Helm is going to come into picture and it will try to make our life easy.

Now, let me quickly show you how Helm is going to approach this problem.

So, whenever we are using Helm, first, we are going to create a single **template.yaml** file for the Service object regardless of how many microservices you have inside your cluster, you are always going to have a single **template.yaml** file like you can see here.



This template yaml files will have static values along with the static syntax structure.

If you closely observe this **helm service template file** at some places it is trying to accept the dynamic values.

For example, under the metadata name it is going to accept the dynamic value with the help of name: {{ .Values.serviceName }}

type: {{ .Values.service.type }}

targetPort: {{ .Values.service.targetPort }}

So, whatever you mention inside this double curly brace is going to represent a dynamic value which

we are going to **inject at runtime whenever we are trying to deploy our microservices into Kubernetes cluster.**

For example, think like we have this helm service template file. As a next step, what I can do is, I can try to provide a **values.yaml** file which is specific to accounts microservice.

So inside this values.yaml I'm going to maintain all the key and values you see inside my values.yaml



The very first property that I am trying to create here is **deploymentlabel: accounts**

The same value that we have defined inside the values.yaml is going to be applied to the service

template file at runtime and behind the scenes helm is going to automatically create the account microservice specific service manifest file.

Very similarly, you just have to provide the values.yaml for other microservices.

With that, you will always have a single template file which is going to be maintained by the helm.

But as a developer or as a DevOps team member, we should provide the values.yaml for each microservice.

And behind the scenes, using these values that we have defined, the helm is going to generate the Kubernetes manifest files at runtime for all your microservices.

**On top of that, we do not have to execute these Kubernetes manifest files manually.**

Whenever we are using Helm, we are going to achieve all the deployment of our microservice with a single command.

So, by following this approach, Helm is going to solve these problems.

**What are the problems that helm try to solve?**

**1st problem solved**

The very first advantage, I would say as Helm is going to support packaging of your Kubernetes manifest files into a single helm chart, the same helm chart can be distributed into a public repository or a private repository you can share with others.

So, it is up to you how you want to leverage that helm chart.

This is very similar to our Java code.

We can store that in a centralized location if needed

we can share our code with others.

Very similarly, we are going to maintain the helm charts.

That is why we are calling Helm as a package manager for Kubernetes.

**2nd problem solved**

And the next problem that Helm solves is it is going to make developers or DevOps team members life easy because it is going to support easier installation.

With the help of Helm, you can always deploy or upgrade or rollback or uninstall your entire microservice application into your Kubernetes cluster with a single command.

No need to run any manual commands with the help of Kubectl, and at last Helm is going to support,

release or version management as well, which means whenever needed, you can roll back your entire Kubernetes cluster to the previous working state with a single command.

Whereas with Kubernetes manifest files like we discussed previously, we can only roll back a particular microservice.

What if you want to roll back your entire Kubernetes cluster. In such scenarios, helm is going to help us.

Installing Helm

Install Chocolatey

<https://chocolatey.org/install>

<https://helm.sh/docs/intro/install/>

Installing a sample Helm Chart

Now, inside this lecture, I am going to show you the power of hell by taking a sample chart available

from the official documentation.

So, before we try to explore the official documentation of hell on how to install a chart of hell, first,

we need to make sure the local Kubernetes cluster is up and running.



We can confirm by going to the Docker dashboard inside the Docker dashboard, you should be able to see a message which is Kubernetes is running.

So, this confirms the cluster is running.

Very similarly you can also run any of the Kubectl command, for example, I can try to run a command which

**> kubectl get services**

So, this will give me the list of services inside my local Kubernetes cluster.

As of now, we have only one service which is related to Kubernetes itself.



So, this confirms our local Kubernetes cluster is working fine.

As a next step, I am going to run a helm command, which

**> helm ls**

So, this is going to list all the releases or all the installations that we have done into the Kubernetes

cluster with the help of Helm.

So, if I try to run as of now, you can see there is no helm installations because as of now we have

not installed any chart into the Kubernetes cluster.

**How my helm can connect to the Kubernetes cluster**?

Because without connecting to the Kubernetes cluster, my helm cannot show this output.

So, the answer to this question is, Helm is going to look for Kubernetes cluster connection details inside your local system.

So, whenever you try to connect to a Kubernetes cluster inside your local system, it is going to make

an entry inside your system.

For example, inside my scenario, I can go to folder

**C:\Users\niles\.kube**

If I try to open this, there is a config file, so if I try to open this file, you will be able to see

all the connection details that my kubectl right now is using to connect to my Kubernetes cluster.

So, you can see right now it is connected to a Docker desktop cluster.

So, the same connection details Helm also is going to leverage whenever it wants to interact with a Kubernetes cluster.

I hope this is clear.

If you are using Windows Operating system, you will be able to find **C:\Users\niles\.kube**

Now let us go and explore on how to setup a sample helm chart inside our local system.

So that will give some good exposure to the helm.

And what are the advantages of Helm?

So, for the same inside this website where we explored about the installation of helm into the local

system, just under this installing helm, we also have one more option which is using Helm.

<https://helm.sh/docs/intro/using_helm/>

So let me click on that post that here there is some good introduction on how to use helm.

<https://helm.sh/docs/intro/using_helm/#helm-search-finding-charts>

You can read if you are interested, but I want to directly go to the section where we have

**’helm search’** command.

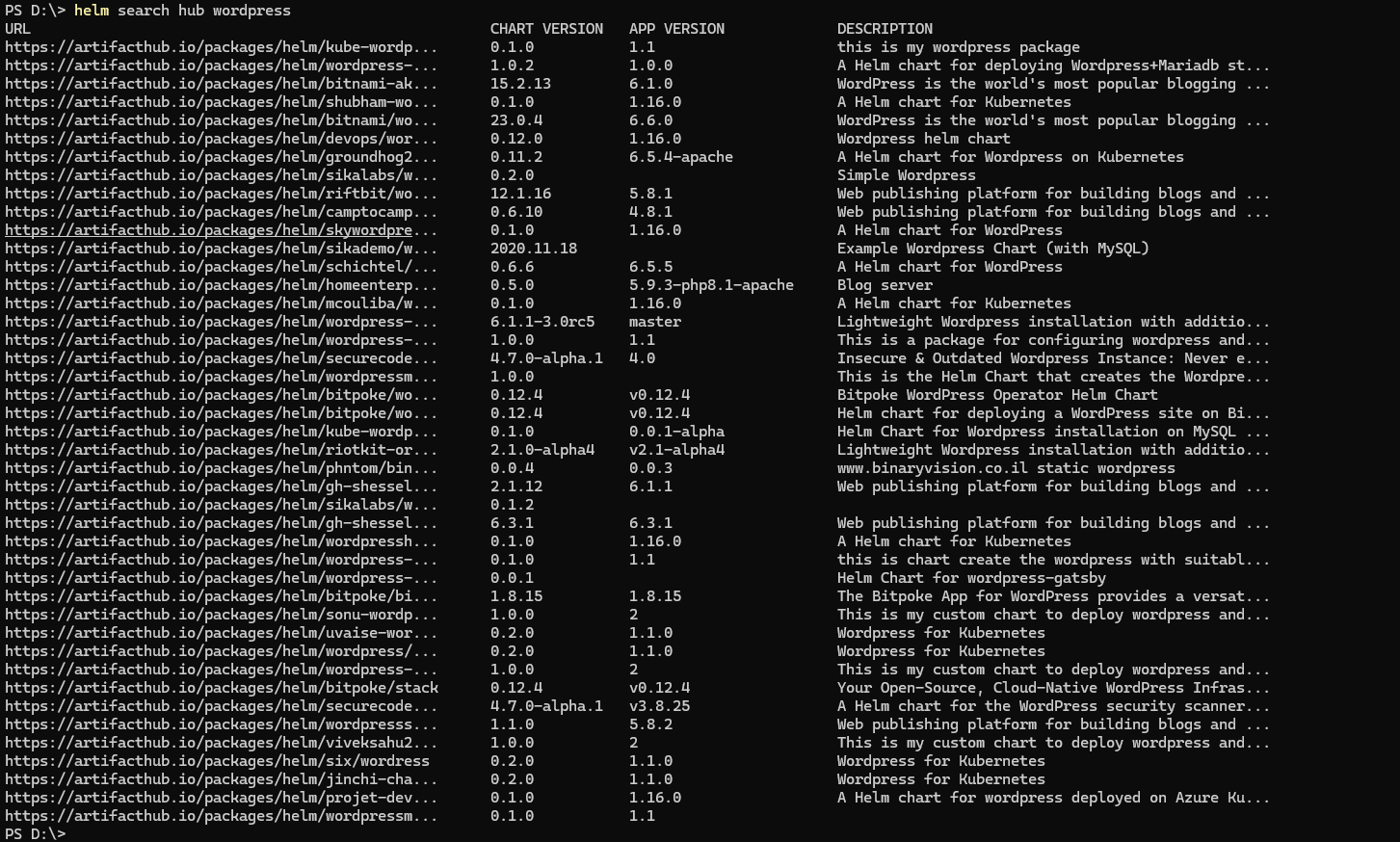
Helm comes with a powerful search command.

Like I said before, there is a good amount of community for Helm where a lot many companies, organizations, and open-source developers, they build a lot of helm charts which you can search with the help of helm search.

For example, here we have a command which is   
**> helm search hub wordpress**

Think like you want to deploy a wordpress website into your Kubernetes cluster. For the same,

first, we need to search if there are any wordpress **related charts available inside the public repositories**.



As soon as I enter, you will be able to see all the repositories where we have a chart with the name wordpress.

So, you can see a lot many places we are able to see wordpress.

So that is why we can see all of them in the output.

Here also, you can see there is a statement saying that

*“The above searches for all wordpress charts on Artifact Hub”*   
So, there are many repositories output that we receive.

So, we have received many repository details based upon our helm search command.

So, we need to choose one of the repositories to install the wordpress helm chart.

For the same, let us go to the website here.

<https://helm.sh/docs/intro/using_helm/#helm-install-installing-a-package>

If you can scroll down, there is a helm install section here they have mentioned on how to install

the WordPress chart available under the Bitnami repo.

> helm install happy-panda bitnami/wordpress

**How to add bitnami repo?**

Like I said, Bitnami is a famous repository which maintains production ready helm charts. So, we can

use the same Bitnami repository to install this WordPress chart.

But before we try to install this command, we need to make sure we have added the Bitnami repo details inside our local system.

So how to add that for the same you can take this bitnami and search inside the Google saying that bitnami helm chart installation.

<https://docs.bitnami.com/kubernetes/faq/get-started/install-configure-helm/>

So, if you try to search with this, you will get the results inside the Google search engine.

And here you can click on this very first link, which is install the chart. So here you can see the very first command they have given is how to add the Bitnami repo to the local helm.

Here I am trying to run the command and this will add the bitnami repo into our local system.

**> helm repo add bitnami https://charts.bitnami.com/bitnami**



Now we can safely run this helm install command.

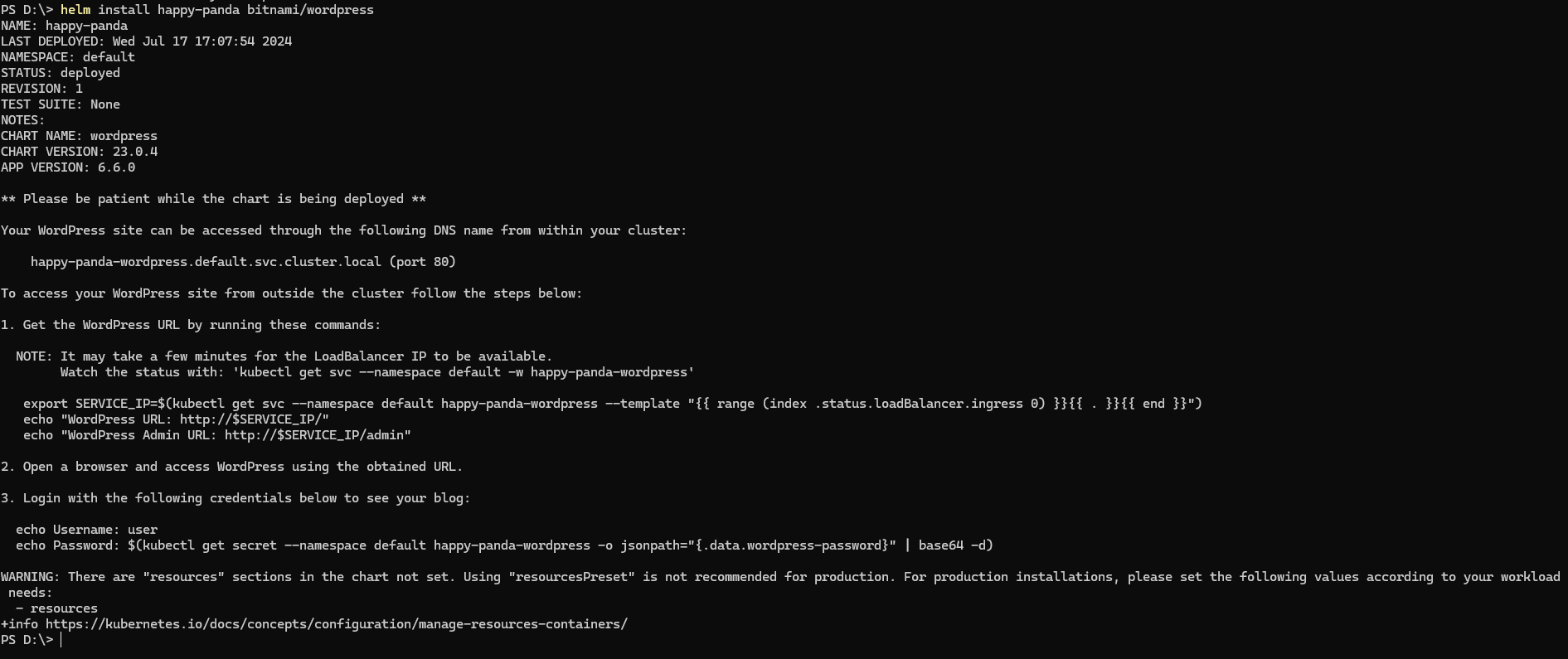
**> helm install happy-panda bitnami/wordpress**

So, the command is helm install and what is the **installation name or the release name** that you want to give? So here they have given the name as happy-panda. The name can be anything and post that bitnami indicates **what is the repository and this repo** which is **bitnami**.

What is the **helm chart**? The helm chart name is wordpress.

So, if I try to run this command, my helm is going to install the chart available inside the Bitnami

repo.



So here if you try to read the very first point, they highlighted that you can get the WordPress URL

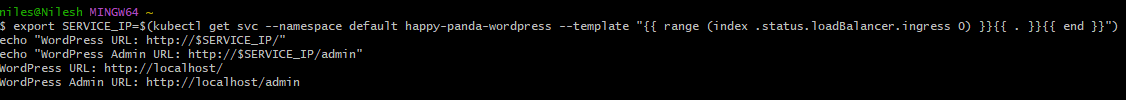
by running these commands and these are the commands.

But before that we have a note saying that it may take few minutes for the load balancer IP to be available. So, we need to be patient and we need to give 1 or 2 minutes for this to get complete.

So, to understand what is the URL of WordPress, we can try to copy this entire command and execute the same inside our terminal.

To execute this command, in Windows 11, you have a few options, including using the Windows Subsystem for Linux (WSL), Git Bash, or a similar Unix-like environment. Here is how you can do it using WSL and Git Bash:

**> wsl --install**

****

**export SERVICE\_IP=$(kubectl get svc --namespace default happy-panda-wordpress --template "{{ range (index .status.loadBalancer.ingress 0) }}{{ . }}{{ end }}")**

**echo "WordPress URL: http://$SERVICE\_IP/"**

**echo "WordPress Admin URL:** [**http://$SERVICE\_IP/admin**](http://$SERVICE_IP/admin)**"**

And this will give you an output saying that the WordPress URL is http localhost,

whereas if you want to log in as an admin, you need to access the URL, which is http localhost/admin

So let me go to the browser and try to access the same.

First, I will try to access the normal URL.

So here I am trying to access and you can see I got a sample WordPress web page.



So, what is WordPress famous for? To build the blog website. So that is why we can see some sample blog website. If you are not able to access it, please wait for 1 to 2 minutes.

You can also confirm the deployment status inside the dashboard.

\eazybank-kubernetes-manifests\kubernetes-dashboard> **kubectl apply -f .\dashboard-adminuser.yaml**

serviceaccount/admin-user created

\eazybank-kubernetes-manifests\kubernetes-dashboard> **kubectl apply -f .\dashboard-rolebinding.yaml**

clusterrolebinding.rbac.authorization.k8s.io/admin-user unchanged

\eazybank-kubernetes-manifests\kubernetes-dashboard> **kubectl -n kubectl -n kubernetes-dashboard create token admin-user**

\eazybank-kubernetes-manifests\kubernetes-dashboard> **kubectl -n kubernetes-dashboard create token admin-user**

eyJhbGciOiJSUzI1NiIsImtpZCI6IlNQOHJNUmdsY1dhbXp3UUt1ZDB6cE9MTWRYd0xhNmltLWVrVjhSTkVrZ2cifQ.eyJhdWQiOlsiaHR0cHM6Ly9rdWJlcm5ldGVzLmRlZmF1bHQuc3ZjLmNsdXN0ZXIubG9jYWwiXSwiZXhwIjoxNzIxMjIyOTQ3LCJpYXQiOjE3MjEyMTkzNDcsImlzcyI6Imh0dHBzOi8va3ViZXJuZXRlcy5kZWZhdWx0LnN2Yy5jbHVzdGVyLmxvY2FsIiwia3ViZXJuZXRlcy5pbyI6eyJuYW1lc3BhY2UiOiJrdWJlcm5ldGVzLWRhc2hib2FyZCIsInNlcnZpY2VhY2NvdW50Ijp7Im5hbWUiOiJhZG1pbi11c2VyIiwidWlkIjoiZmQ2ZGYzYzMtZjY5OC00OWQ0LThmM2EtNzMyMDkyYzM2NDAzIn19LCJuYmYiOjE3MjEyMTkzNDcsInN1YiI6InN5c3RlbTpzZXJ2aWNlYWNjb3VudDprdWJlcm5ldGVzLWRhc2hib2FyZDphZG1pbi11c2VyIn0.nXbLmaD\_OfG7OtGLNI3NoYpS7XY9Yx0YPdWSdEo\_NtRNp\_eql5rsodNVt-NHrEJrZY-wTOygVn4BjizPQBk7PcBb0sMDtR9erRcR35sKvHycxXjNlT\_STxikk3mAul0cqNkVGBKoBsI1eNszzPCTs2Xiv031OebPw3K3Cutwno2JjBalERilJ5RD0tcgDFG\_rBjN9aiNoNEr7zDMJiW23CXRG\_QbUwkQ3wj3pEqdSZO1D5on6zFbftfLWzEZmhzLZwPXNi\_lNcRDoyOUjEWoA8YzuzbcBIigTyBs7\_zygODrkZFCsgsW6sTIjXa0abaPPfFXJiWvjEisVqbWlIESpw

\eazybank-kubernetes-manifests\kubernetes-dashboard> **kubectl apply -f .\long-lived-secret-dashboard-token.yaml**

secret/admin-user created

So as of now you can see whatever helm chart that we have installed.

It did a lot of work behind the scenes.

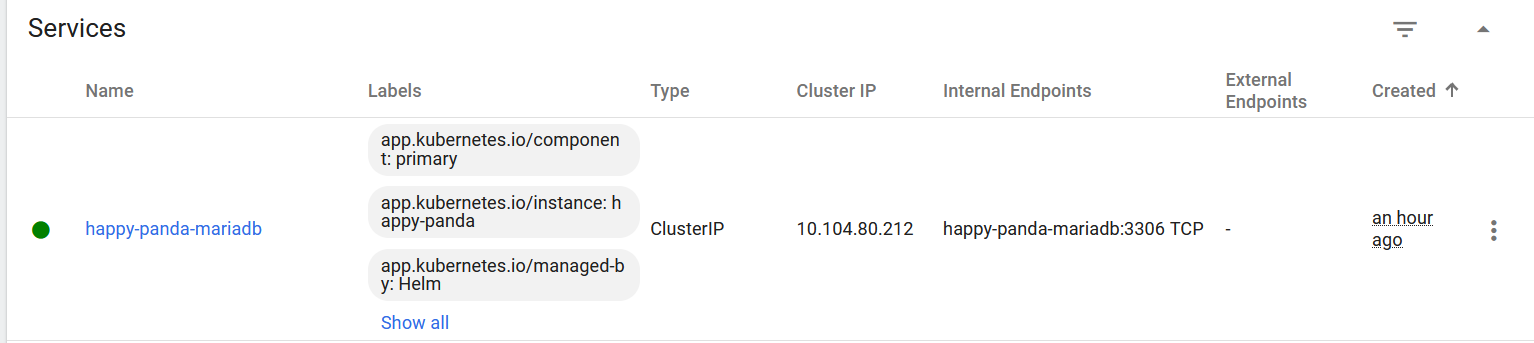
First, you can check and deployments.

There is 1 deployment happened and similarly, under the pods you will be able to see 2 pods.

One is related to the MariaDB, which my blog website is going to use behind the scenes.

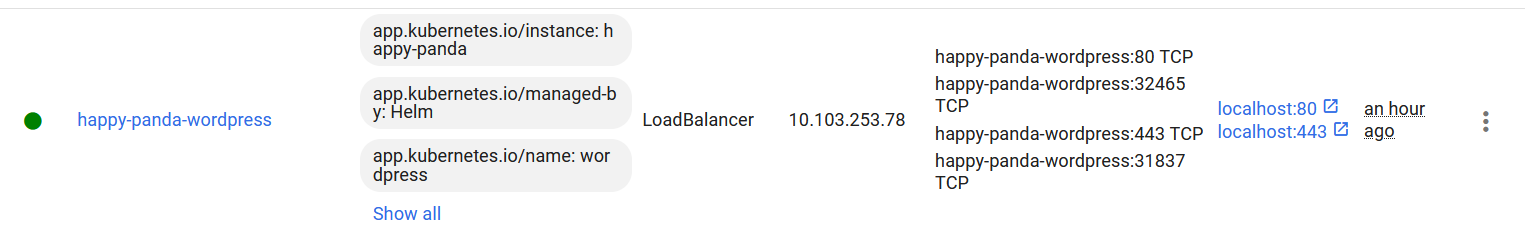
And the second part is related to the WordPress website itself and we will be having replica set details and similarly, you will be able to see the service details.

So, my MariaDB is exposed as cluster IP because I do not have any requirement to access from outside of the cluster. My WordPress website is going to access it internally within the cluster.



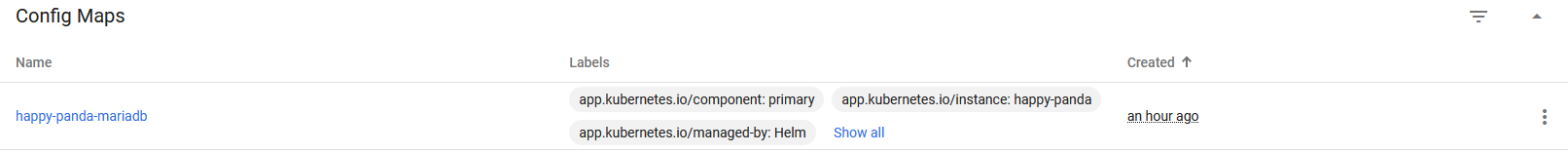
Whereas the WordPress website is deployed as LoadBalancer service type because we need to access it from outside of the cluster.

That is why we have the LoadBalancer service type here.



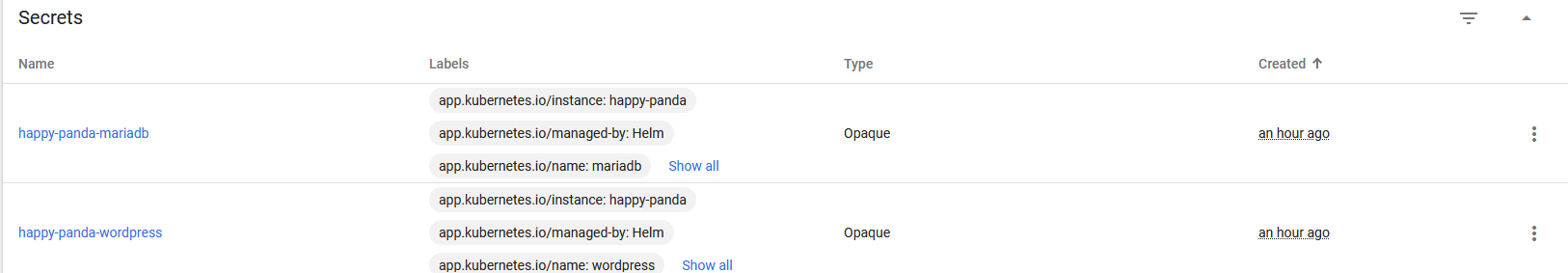
Very similarly, you can go and check what are the config maps created.

So here this is a config map which has all these environment property details



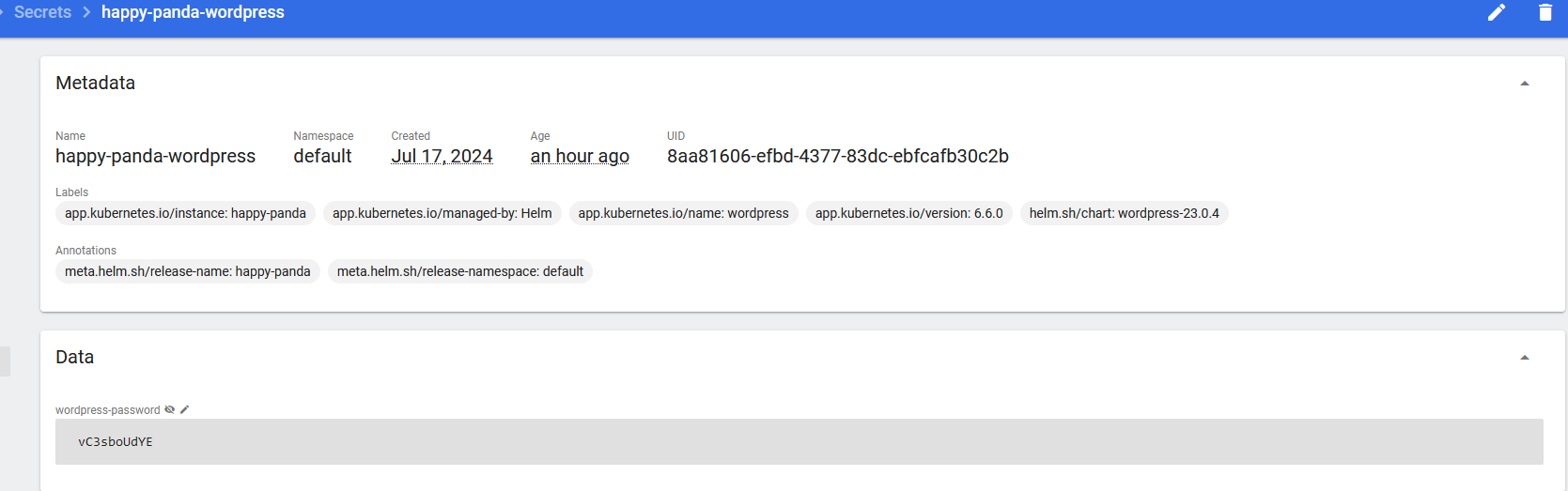
and similarly you can

go to secrets and here there are two secrets one with the name happy-panda-wordpress and the other one related to the database.



So, if I try to click on this WordPress secret, so here we have a password that we can use to access

the website as an admin.

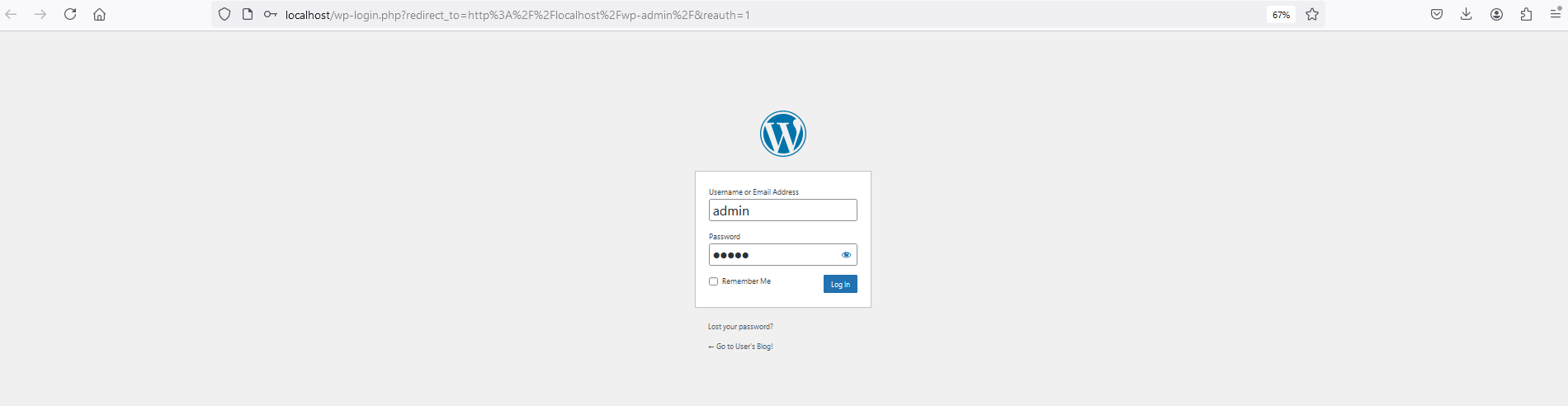


So, this way behind the scenes, my helm chart did a lot of job for me to set up this website with production ready standards.

So now let me try to access as an admin for the same,

I just need to access path http://localhost/admin,

but it is going to ask me the username and password, password we already saw previously.



This is going to be the password.

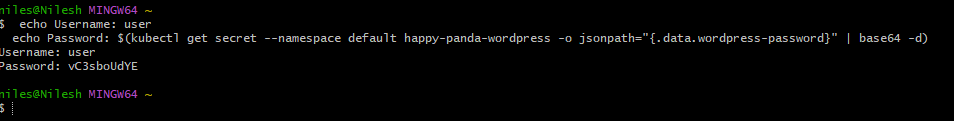
We can also try to get the username and password from the instructions that we have received on the terminal. So, inside the terminal you can see under the third point they have highlighted how to get the username and password so you can try to take this command and run the same inside your terminal.

And this will give what is the username.

Username is user and this is the password.

$ echo Username: user

$ echo Password: $(kubectl get secret --namespace default happy-panda-wordpress -o jsonpath="{.data.wordpress-password}" | base64 -d)



So let me copy the password.

Here the username is going to be user and password

I have pasted and post that I am going to click on this login.

Now you can see I am able to access my WordPress website as an admin.

Here I can create new blogs or new articles as an admin of this WordPress website.

So, the point that I want you to highlight here is did you see how easily I can set up all these

WordPress websites very easily with the help of Helm chart, I just ran a single command which

**> helm install command**

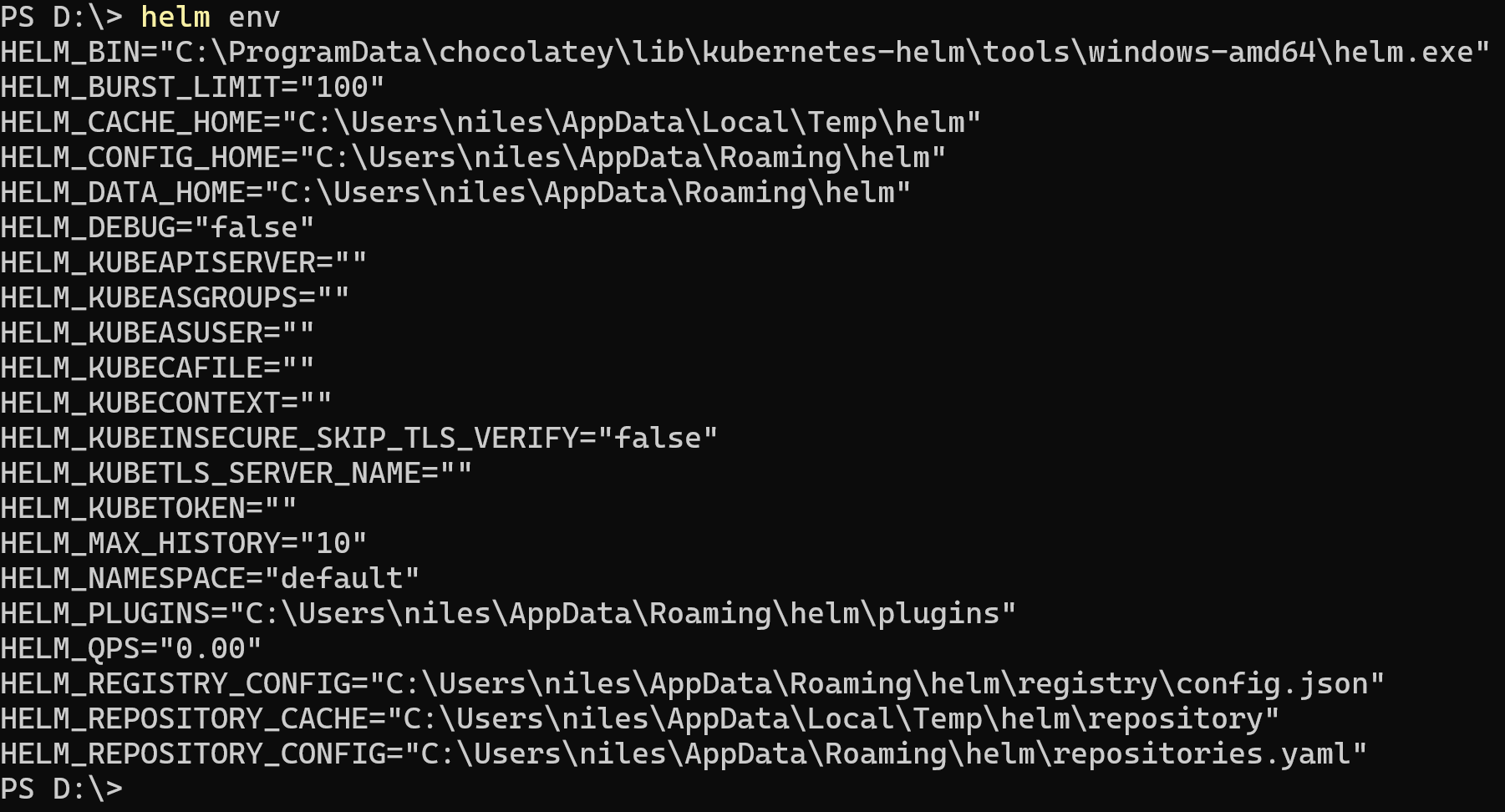
and behind the scenes it ran so many Kubernetes manifest files.

So, do you want to see what are all the files that it has executed?

So, for the same we need to understand where the helm chart inside your local system is saved.

So, to understand the same, we need to run the command which

**> helm env**

****

So, this will give you output about your health.

Here I just wanted to highlight you that all our helm charts are going to be saved under these



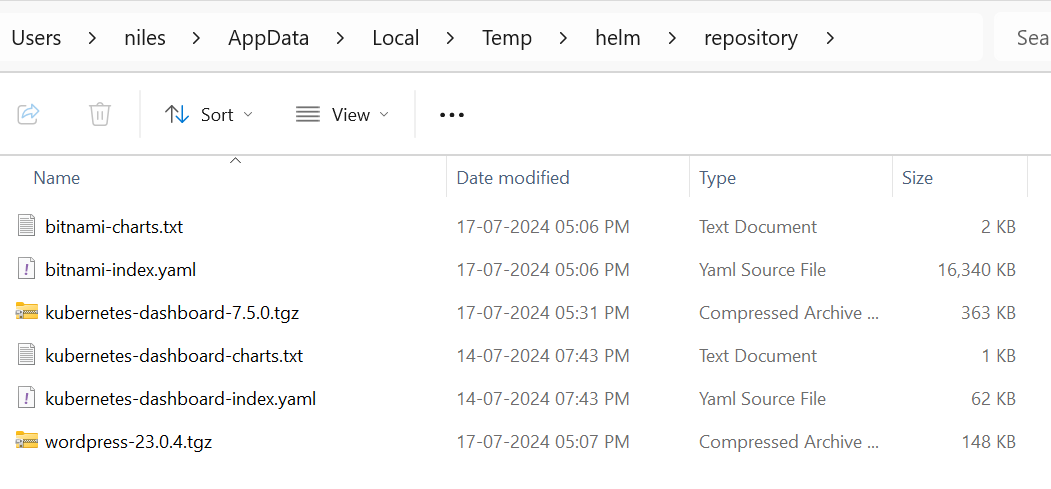
So, let me copy this path.

I will try to access the same.

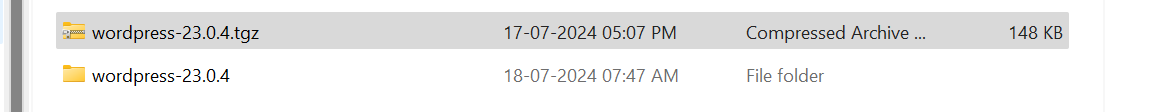
So, inside this helm folder we have another folder

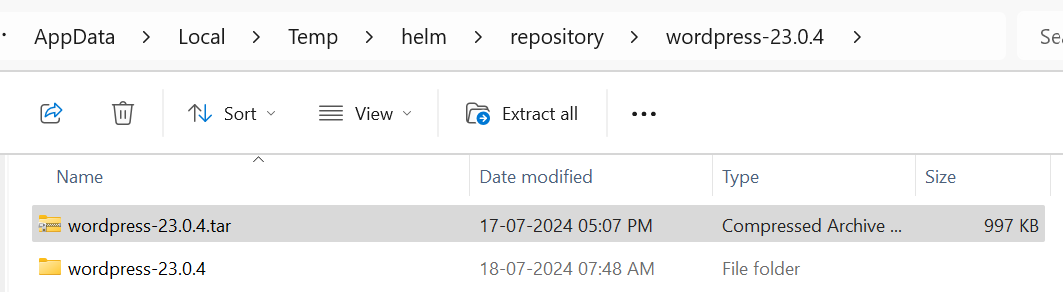
C:\Users\niles\AppData\Local\Temp\helm\**repository**

And here we have a helm chart with the name wordpress.



So, let us try to extract this compressed file.

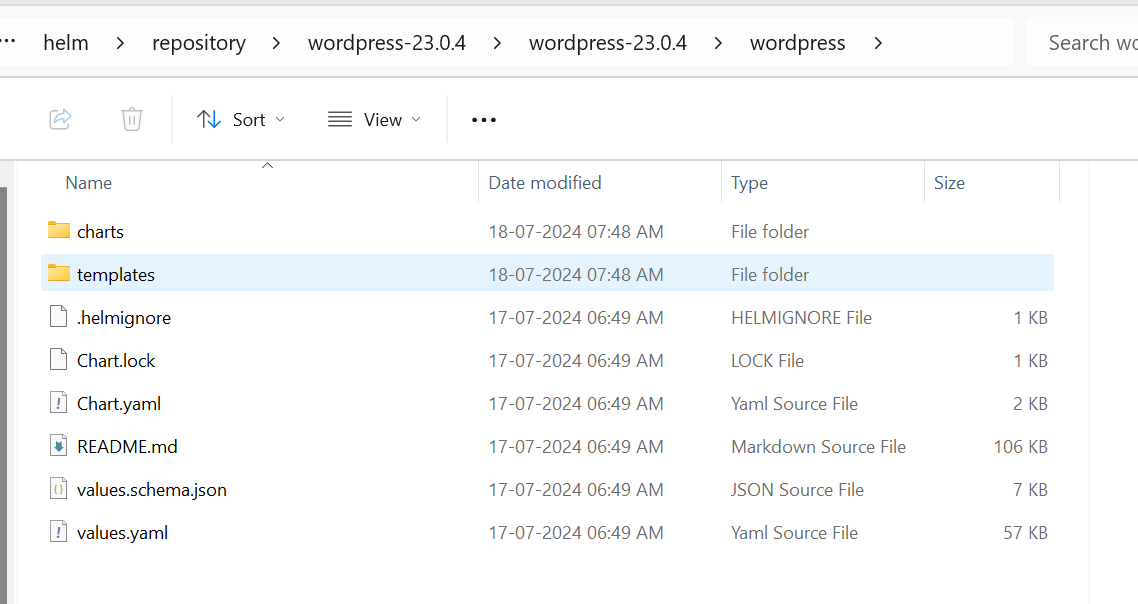




So, inside my local system I extracted this by double clicking on this file.

So, if you try to open this WordPress, you will be able to see the list of files available inside the

helm chart with the name WordPress.



So, let us try to understand more details about the structure of helm charts in the next lecture.

In the same process, I am going to show you all the Kubernetes manifest files that my helm ran behind the scenes to set up the WordPress website.

Understanding Helm Chart structure

Creating our own Helm chart and template files

Creating Helm chart for Accounts microservice

Creating Helm chart for other microservices

Creating Helm Chart for Dev, QA and Prod environment

Demo of helm template command

Install Keycloak in Kubernetes Cluster using Helm Chart

Install Kafka in Kubernetes Cluster using Helm Chart

Install Prometheus in Kubernetes Cluster using Helm Chart

Install Grafana Loki and Tempo in Kubernetes Cluster using Helm Chart

Install Grafana in Kubernetes Cluster using Helm Chart

Install eazybank microservices in Kubernetes Cluster using Helm Chart

Demo of helm upgrade command

Demo of helm history and rollback commands

Demo of helm uninstall command

Quick revision of important helm commands