SCALING OF THE K8S(LAST UHUHUHUHUHU)

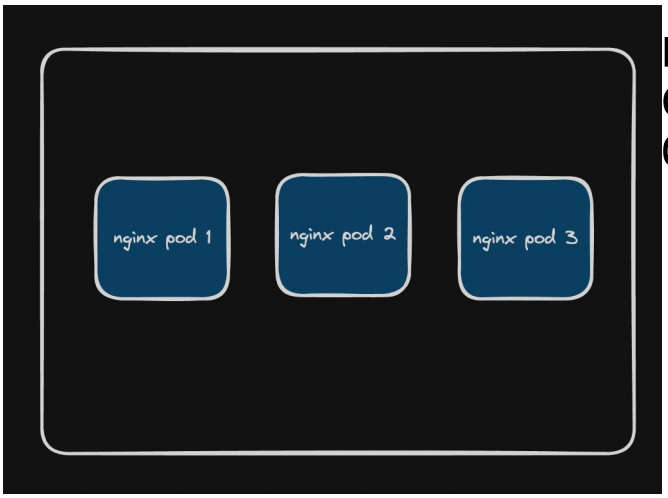
HORIZONTAL POD AUTOSCALER

Ref - https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/

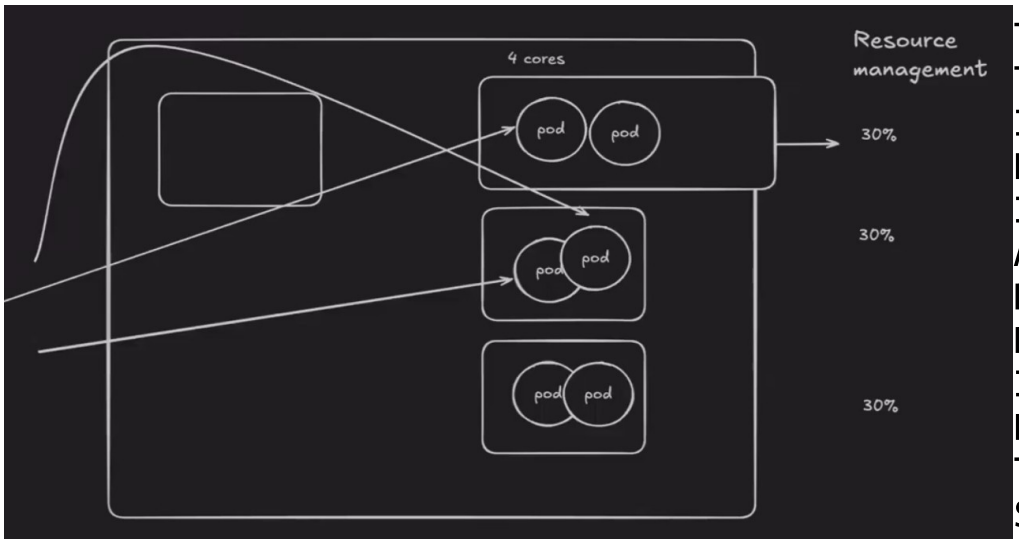
A Horizontal Pod Autoscaler (HPA) is a Kubernetes feature that automatically adjusts the number of pod replicas in a deployment, replica set, or stateful set based on observed metrics like CPU utilisation or custom metrics.
This helps ensure that the application can handle varying loads by scaling out (adding more pod replicas) when demand increases and scaling in (reducing the number of pod replicas) when demand decreases.

Horizontal scaling

As the name suggests, if you add more pods to your cluster, it means scaling horizontally. Horizontally refers to the fact that you havent increased the resources on the machine.



NODE JS IS SINGLE THREADED SO ONE CORE CAN RUN ONE NODE JS PROCESS ONLY



THIS IS HOW YOU DO IT THE RESOURCE IS MANAGED IN THIS ACROSS MULTIPLE PODS WHEN THE USAGE INCREASES AND DECREASE ALSO WHEN THE USAGE DECREASES AND LIKE IF A NODE AS LIKE 3 CORE CPU IT SHOULD MAX RUN THREE PODS OR IF IT RUN MORE THEN IT WILL CONTEXT SWITCH WHICH WILL LEAD TO POOR PERFORMANCE

VERTICAL SCALING IS LIKE INCREASE THE COMPUTATION POWER OF A MACHINE LIKE MAKING IT FROM 4 CORES TO 8 CORES
BUT ITS NOT EASY WITHOUT STOPPING THE MACHINE

THERE ALSO EXIST A VERTICAL POD AUTOSCALER WILL STUDY IN RESOURCE MANAGEMENT

Architecture

Kubernetes implements horizontal pod autoscaling as a control loop that runs intermittently (it is not a continuous process) (once every 15s)

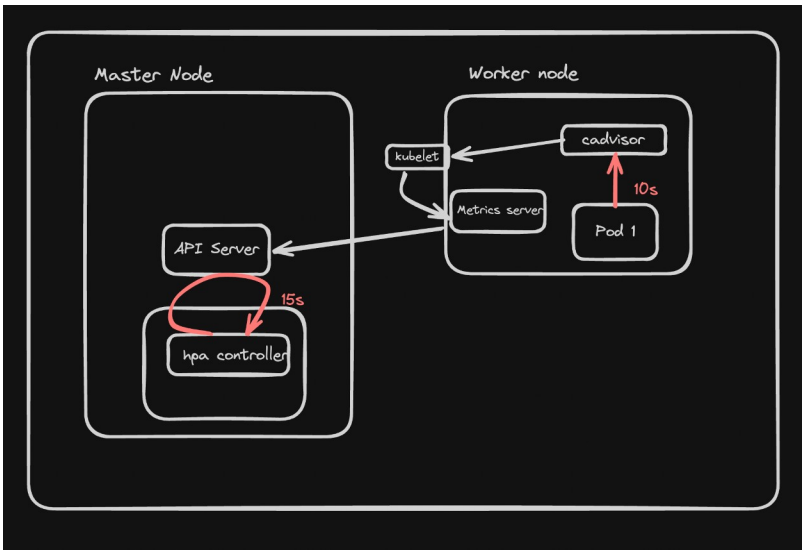
- cadvisor - https://github.com/google/cadvisor

(USES THE OPEN SOURCE PROJECT TO SHOW THE METRICS OF THE POD AND K8S JUST USES IT AND PROVIDE A GOOD API)

- Metrics server - The Metrics Server is a lightweight, in-memory store for metrics. It collects resource usage metrics (such as CPU and memory) from the kubelets and exposes them via the Kubernetes API (Ref - https://github.com/kubernetes-sigs/metrics-server/issues/237)

THIS WE HAVE APPLY IN THE SYSTEM TO ACCESS THE USAGE OF THE SYSTEM

kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
(THIS WILL HELP TO GET ALL THE INFO OF THE PODS)



or
Apply from here - https://github.com/100xdevs-cohort-2/week-28-manifests

- Try getting the metrics

kubectl top pod -n kube-system
(THIS COMMAND WILL HELP US SHOW THE USAGE OF RESOURCES)
kubectl top nodes -n kube-system

Sample request that goes from hpa controller to the API server

GET https://338eb37e-2824-4089-8eee-5a05f84fb85e.vultr-k8s.com:6443/apis/metri

IMAGINE YOU HAVE A OPTION TO CHOOSE BETWEEN 1 BIG SERVER OR 10 SERVERS IN WHICH THE COMPUTATION IS DIVIDED WITH THE SAME PRICE SO CHOOSE THE 10 MACHINES BCOZ THE RESOURCES ARE DIVIDED, LB IS HAPPENING, AND IF LIKE 3 MACHINES GO DOWN 7 OF THEM ARE STILL RUNNING

NOW CREATING A K8S CLUSTER AND WE CAN CREATE OFFLINE ALSO
kind create cluster --config clusters.yml -n local

which starts a k8s cluster with one master node and 2 worker nodes

MANY OF THE TIME IT HAPPENS IN CIVO THE MASTER NODE CRASHES 💀💀 THEN YOU ARE SCREWED

so now the k8s cluster is ready so just copy the config file to the /.kube/config
cd ~/.kube
cp new_config config
kubectl get pods

applying this
kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml

kubectl run nginx --image=nginx
kubectl get pods(nginx pod running)

kubectl top pods
output

```
NAME    CPU(cores)   MEMORY(bytes)
nginx   0m           3Mi
```

kubectl top nodes

now to test the metric we are adding a cpu heavy code
using deployment.yml(creates the pods and all) and service.yml(export the pods on a lb to the world so that we can hit the url) for now they both are in the same file only seperate by ---

codes:
manifest.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cpu-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: cpu-app
  template:
    metadata:
      labels:
        app: cpu-app
    spec:
      containers:
      - name: cpu-app
        image: 100xdevs/week-28:latest
        ports:
        - containerPort: 3000
---
apiVersion: v1
kind: Service
metadata:
  name: cpu-service
spec:
  selector:
    app: cpu-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 3000
  type: LoadBalancer
```

kubectl apply -f .\manifest.yml
abh locally use nhi kar skte kyuki lb kaha sae laoge internet exposed wala

now like checking through the kubectl get pods(pods are running)

now using kubectl top pods(both the pods are idle)

now using the kubectl get svc(we get the external ip which we run many times)

now when we do kubectl top pods

```
> test-manifests-scaling kubectl top pods
NAME                              CPU(cores)   MEMORY(bytes)
cpu-deployment-54c4d866d8-9xffw   1m           62Mi
cpu-deployment-54c4d866d8-jzjd9   1m           34Mi
> test-manifests-scaling kubectl top pods
NAME                              CPU(cores)   MEMORY(bytes)
cpu-deployment-54c4d866d8-9xffw   837m         64Mi
cpu-deployment-54c4d866d8-jzjd9   1m           34Mi
```

837m means 0.8 usage out of 1 of a cpu

GREAT QUESTION
WHY ONLY SINGLE CPU IS USED THE MOST?
BCOZ IN THIS THERE IS SOME STICKINESS WHICH MEANS IF YOU ARE SENDING REQUEST FROM A SINGLE BROWSER IT WILL ONLY TAKE IN ONE POD AND IF YOU SEND REQUEST FROM DIFFERENT BROWSER TOO THAN THERE WILL BE SPIKE IN BOTH THE PODS

```
> test-manifests-scaling kubectl top pods
NAME                                CPU(cores)   MEMORY(bytes)
cpu-deployment-54c4d866d8-9xffw     1m           64Mi
cpu-deployment-54c4d866d8-jzjd9     1m           36Mi
> test-manifests-scaling kubectl top pods
NAME                                CPU(cores)   MEMORY(bytes)
cpu-deployment-54c4d866d8-9xffw     961m         64Mi
cpu-deployment-54c4d866d8-jzjd9     251m         36Mi
```

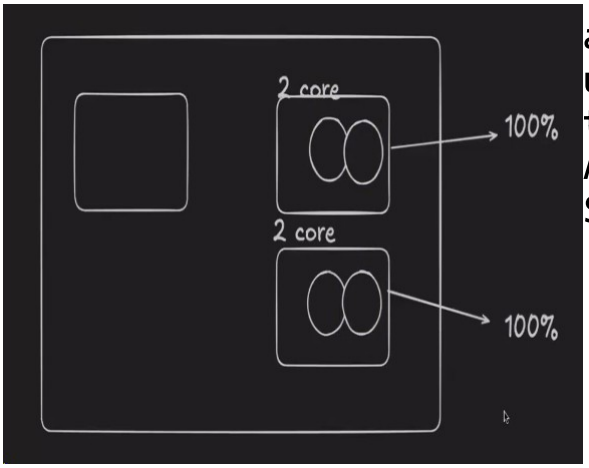
NOW SEE IN THIS BOTH THE CORES ARE USED IN THIS

THIS IS DUE TO PROCESS STICKINESS, CORE STICKINESS

AND ALSO LIKE IN A SAME BROWSER IF YOU SEND AFTER SOME TIME THE REQUEST IT MAY TRANSFER TO THE SECOND BROWSER

```
↦ test-manifests k top nodes
NAME                          CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
class-metrics-cluster-6923b   1000m        52%    840Mi           27%
class-metrics-cluster-6923g   32m          1%     748Mi           24%
class-metrics-cluster-6923w   977m         51%    913Mi           29%
```

now the issue is the worker are used just 50% not 100% ? bcoz node js is single threaded



as its a 2 core machine so one pod can only use 1 core so for that we need to autoscale to 2 pods in each node
AND EVERY 15 SECONDS THE METRICS ARE SCRAPED

TWO APPROACHES TO FIX THIS
1. manually updating the replicas in the manifest file from 2 to 4
2. creating a horizontal pod autoscaler

LIKE TOP IS USED TO SEE THE METRICS IN THE MAC MACHINE
SAME KUBECTL TOP NODES HELPS TO SEE IN NODES ITS USAGE

now before doing the horizontal pod autoscaler

Resource requests and limits

Ref - https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
When you specify a Pod, you can optionally specify how much of each resource a container needs. The most common resources to specify are CPU and memory (RAM).
There are two types of resource types

Resource requests

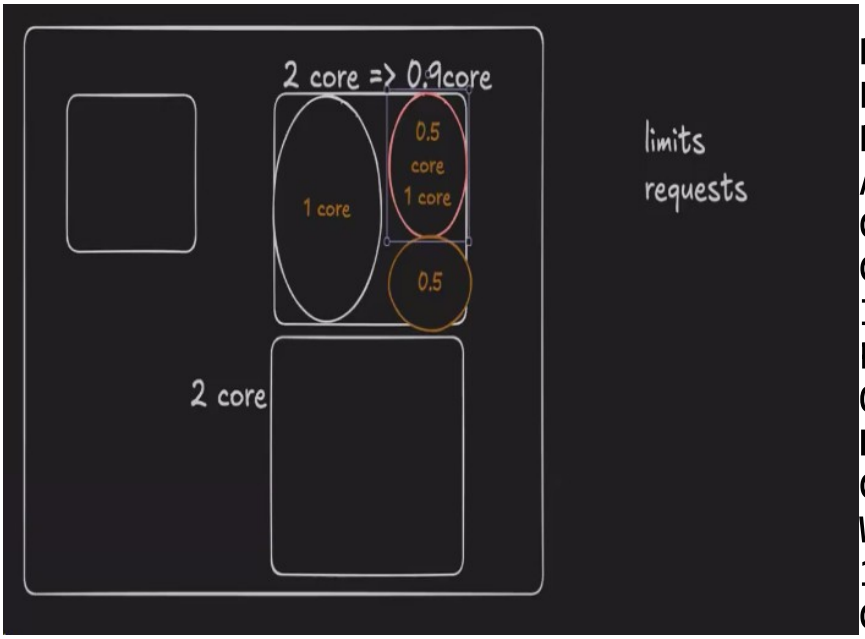
The kubelet reserves at least the *request* amount of that system resource specifically for that container to use.

Resource limits

When you specify a resource *limit* for a container,
the kubelet enforces those limits so that the running container is not allowed to use more of that resource than the limit you set.

Difference b/w limits and requests

If the node where a Pod is running has enough of a resource available, it's possible (and allowed) for a container to use more resource than its request for that resource specifies. However, a container is not allowed to use more than its resource limit.



REQUEST MEANS A EG FOR A NODE POD COMES AND REQUEST THAT IT NEEDS 1 MIN CORE TO START AND ANOTHER ONE COMES WANT 0.5 CORE AND WHEN A ANOTHER ONE COMES AND WANT 0.6 CORE THEN IT WILL SEND TO ANOTHER NODE FOR THAT REQUEST THIS PROCESS OF ASSIGNING IS CALLED AS **REQUEST**, AND RIGHT NOW EACH CORE HAS A MIN BUT NO MAXIMUM WHICH MEANS THE 0.5 CAN USE 1.5 CORE DUE TO WHICH THE 1 CORE MACHINE WILL STARVE AND WILL NOT START TO FIX THAT WE CAN ASSIGN A LIMIT TO EACH POD FOR THAT WE USE THE **RESOURCE LIMITS**
LIKE THE 0.5 CORE POD CAN MAX USE 1 CORE ONLY

30% CPU usage on a single threaded Node.js app

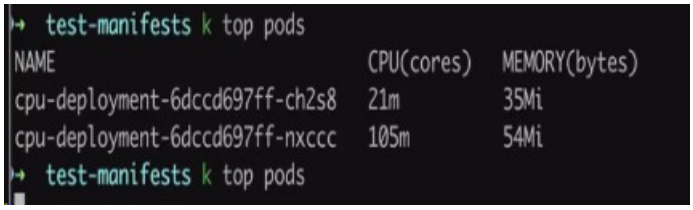
Update the spec from the last slide to decrease the CPU usage. Notice that the CPU doesnt go over 30% even though this is a Node.js app where it can go up to 100%

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cpu-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: cpu-app
  template:
    metadata:
      labels:
        app: cpu-app
    spec:
      containers:
      - name: cpu-app
        image: 100xdevs/week-28:latest
        ports:
        - containerPort: 3000
        resources:
          requests:
            cpu: "100m"
          limits:
            cpu: "300m"
```

IN THIS WE HAVE MENTIONED LIKE THE REQUEST AND THE LIMITS FOR THE POD changes this in the manifest.yml file with the deployment part

AND NOW IF WE HIT THE URL IT WILL BE VERY SLOW BCOZ THE MAX A POD CAN USE IS 0.3 OF 1 CORE CPU

THIS WILL HELP IN NOT HOG ONE CPU FOR LIKE ONE HEAVY REQ



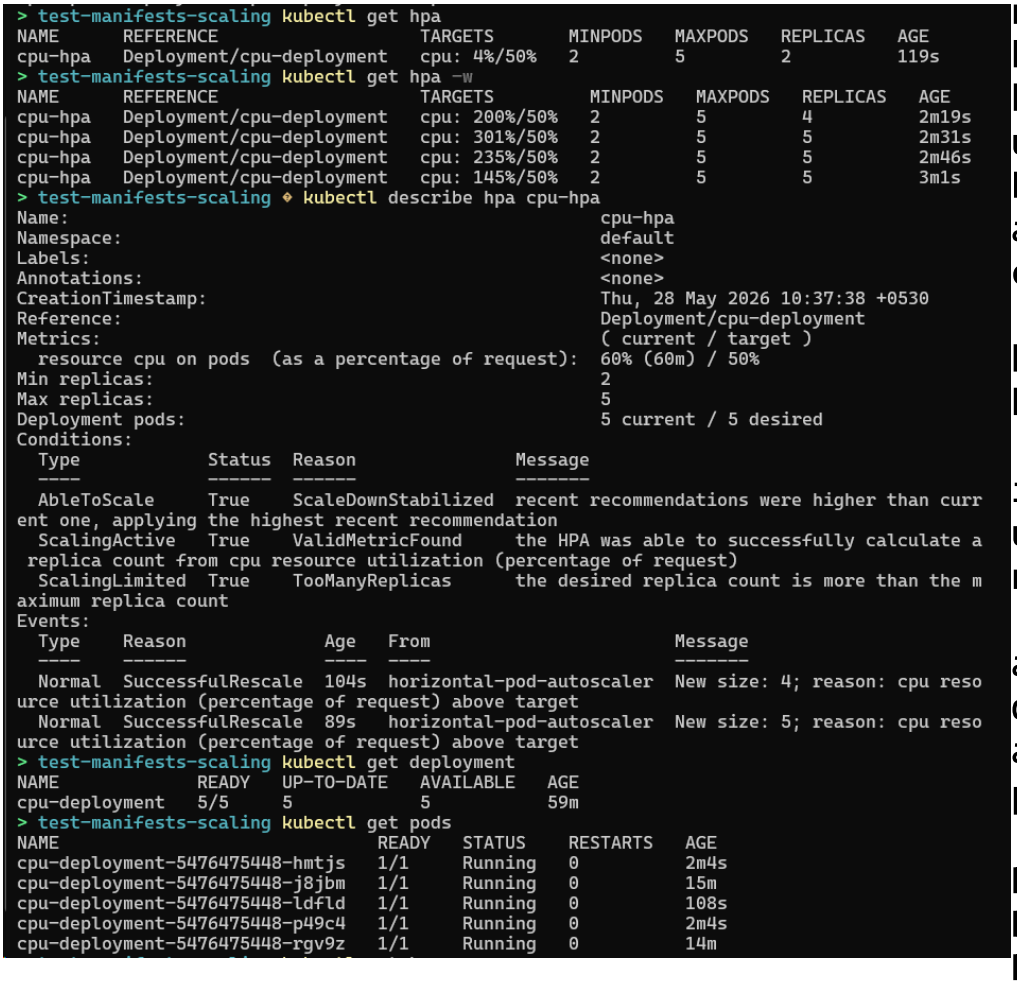
NOW WHAT averageUtilization: 50 MEANS?
This means the limit which we gave in request for the cpu the 50% above that that mean 50 for this one as the total requested was 100 for the above example

NOW CREATING A HPA HPA.YML

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: cpu-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: cpu-deployment
  minReplicas: 2
  maxReplicas: 5
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50
```

kubectl apply -f hpa.yml

kubectl get hpa



now after creating the hpa we started to hit the hpa and you can see the usage is VERY MUCH above FROM 50% means it should autoscale so when we check that using

kubectl desribe hpa cpu-hpa

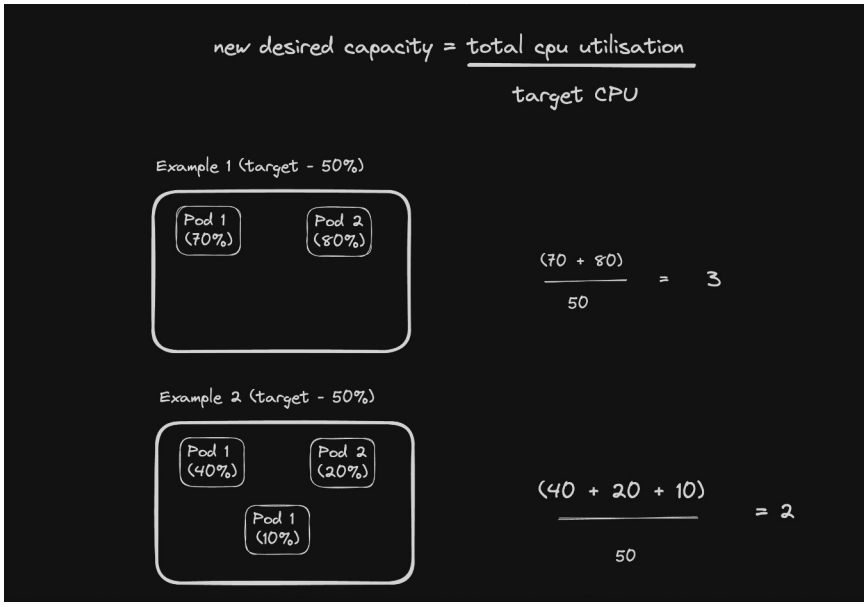
it shows that due to cpu utilization the size is now 4 to 5

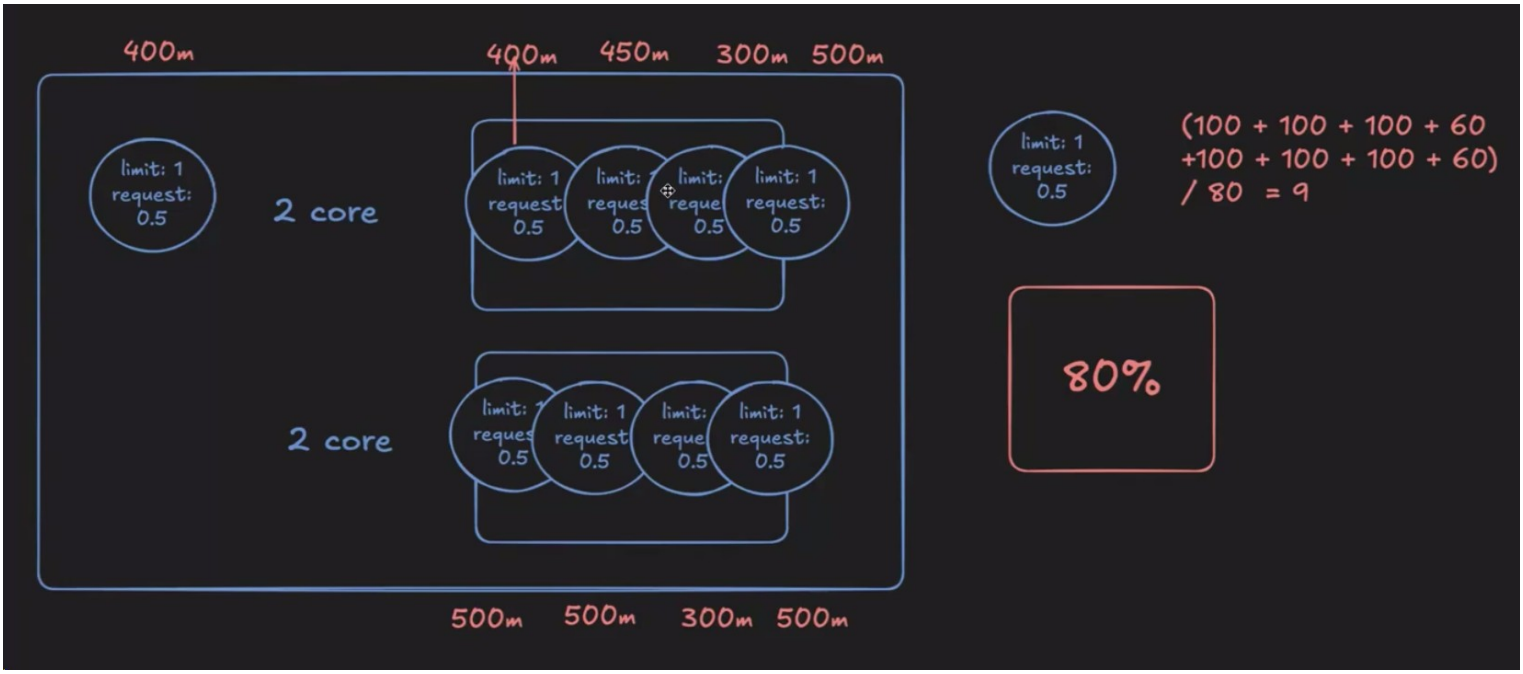
and u can see in the deployment also the pods are now 5 and running pods are also 5 now

NOW AS THE USAGE DECREASED THE PODS WILL RETURN TO 4 AGAIN

You can scale up/down based on multiple metrics.
If either of the metrics goes above the threshold, we scale up
If all the metrics go below the threshold, we scale down

Formula for scaling up





UNDERSTAND THROUGH THIS LIKE IF THE USAGE OF ALL THE CPU WHILE ADDING AND FIND THE PERCENTAGE GOES ABOVE THE MENTIONED USAGE IN THE HORIZONTAL POD SCALER THEN ITS REQUIRES SCALING LIKE IN THIS ALSO IT WAS AROUND 82% WHICH MEANS IT REQUIRES AUTO SCALING

AND WHEN THE TRAFFIC GOES AWAY THE POD AUTOSCALE DOWN TO LESS PODS

GITOPS AND ARGOCD

What is gitops

GitOps is a modern approach to managing infrastructure(ADDING NEW RESOUCES, CREATING A VM) and application deployment(CREATING PODS AND SERVICES) using Git as the **single source of truth**. It applies DevOps practices—like version control, CI/CD, and automation—to infrastructure.

Core Concepts of GitOps:

1. **Git as the Source of Truth**:

   All infrastructure and deployment configurations are stored in a Git repository. This includes Kubernetes manifests, Helm charts, Terraform files, etc.

2. **Declarative Descriptions**:

   Desired system state is described in a declarative way (e.g., YAML files for Kubernetes).

3. **Automated Reconciliation**:

   A GitOps agent (like Argo CD or Flux) continuously watches the Git repo and automatically applies changes to the live environment to match the desired state.

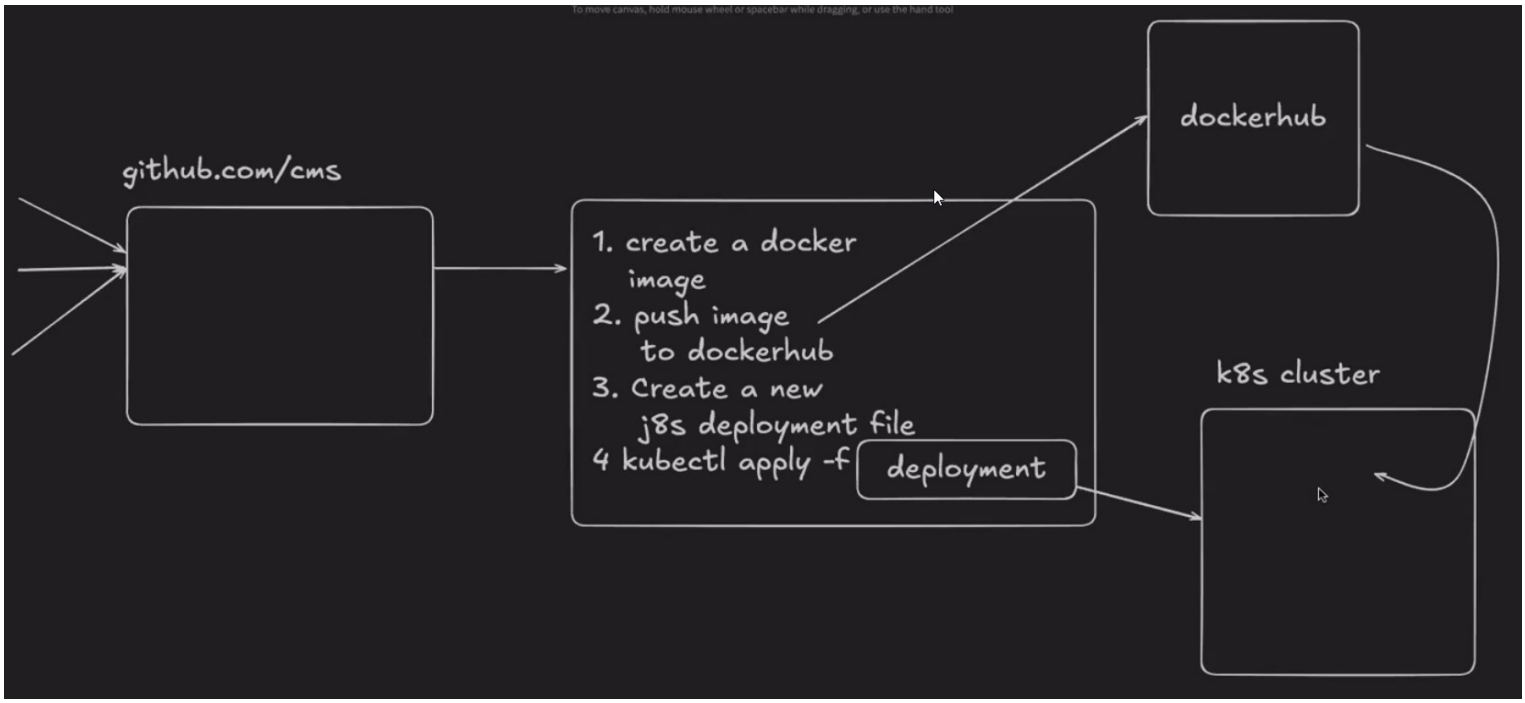
4. **Pull-Based Deployment**:

   Instead of pushing changes from a CI/CD pipeline, GitOps tools pull updates from the Git repository, making the system more secure and observable.

5. **Audit and Rollback**:

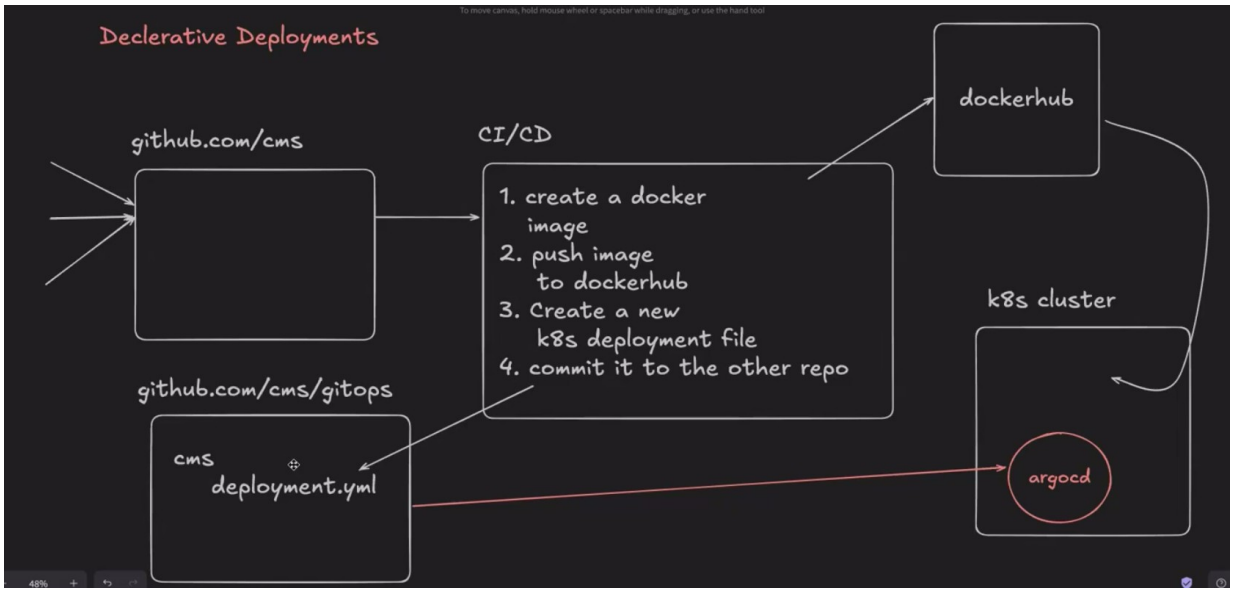
   Every change is tracked via Git commits, enabling easy auditing and rollback by reverting commits.

THE BIG PROBLEM IS WHEN YOU ARE WORKING WITH THE VM YOU DON’T KNOW WHO IS DOING WHAT CHANGES WHEN AND WHERE, WHICH DEV SSH INTO THE MACHINE ETC
NO VISIBILITY HOW YOUR INFRA CODE IS CHANGING



BY FIRST PRINCIPLE HOW WE USED TO DEPLOY?
LIKE IN THIS WE CREATE A DCOKER FILE PUSH TO DOCKERHUB THEN CREATE THE YML FILES AND THEN APPLY IT AND WHEN WE MAKE CHANGES IN THE FILE THEN WE CHANGE THE URL OF THE YML FILE OF DEPLOYMENT.YML AND THEN WE PUSH THE NEW CODE AND IT ALSO ENSURES IF ANY BAD CODE IS GIVE ROLLING UPDATE WILL NOT BE DONE AND ALSO BUT BUT BUT (**THIS IS CALLED AS A PUSH BASE MODEL**)
THERE IS NOT LOG OF THE CHANGES WHICH ARE DONE

FULL WORKFLOW EXPLAINED WITH GITOPS



imagine 2 repos one with the code and other one with the yml files only which the k8s access

now in this imagine someone made some changes so than it runs on the github workflows

```yaml
name: Continuous Deployment (Prod)

on:
  push:
    branches: [ "main" ]

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
        with:
          fetch-depth: 0

      - name: Docker login
        uses: docker/login-action@v2
        with:
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKER_SECRET }}

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v2

      - name: Build and push school image
        uses: docker/build-push-action@v4
        with:
          context: .
          file: Dockerfile
          push: true
          tags: 100xdevs/school:${{ github.sha }}

      - name: Clone staging-ops repo, update image tags, and push tags
        env:
          PAT: ${{ secrets.PAT }}
        run: |
          git clone https://github.com/code100x/staging-ops.git
          cd staging-ops

          sed -i 's|image: 100xdevs/school:.*|image: 100xdevs/school:${{ github.sha }}|' prod/school/deployment.yml

          git config user.name "GitHub Actions Bot"
          git config user.email "actions@github.com"
          git add .
          git commit -m "Update school server image tags to ${{ github.sha }}"
          git push https://${PAT}@github.com/code100x/staging-ops.git main
```

now first the docker login is done and the docker new file is made with the sha of the github commit and then pushed to the dockerhub
then the yml file repo is cloned and then the docker file name is changed in that file and then the commit is pushed in that repo also

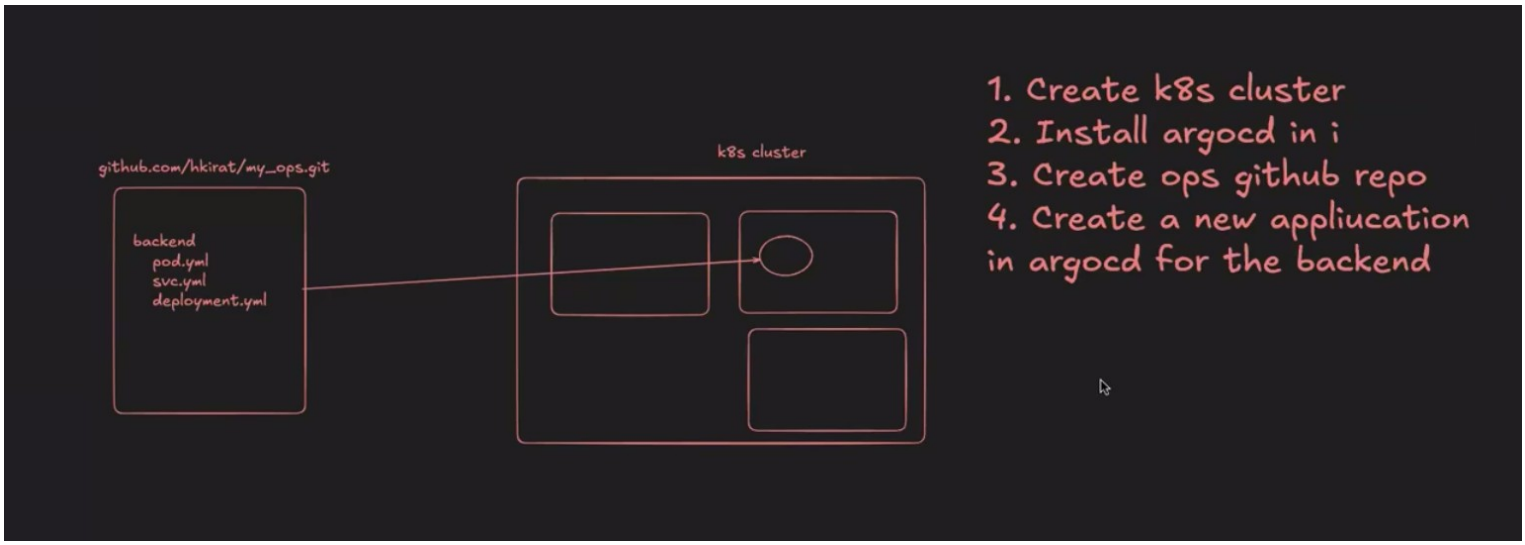
THIS IS LIKE WE HAVE DESCRIBED THE **OPS FILE USING GIT**
THIS IS CALLED AS **DECLARATIVE DEPLOYMENT(THIS IS LIKE WE HAVE DECLARED WHAT TYPE OF FILE WE WANT AND HOPING THAT THE K8S WILL CREATE THAT)**

NOW IN THE K8S SOMETHING IS PRESENT WHICH IS PULLING EVERY 5-10 SEC NEW DATA FROM THE OPS REPO(WHICH IS CALLED AS ARGOCD GITHUB PROVIDER) SAME AS ARGO CD THERE IS SOMETHING CALLED AS FLUX

REACT IS A DECLARATIVE UI(ISME BHI HAAM DECLARE KARDETE HAI KAISA DIKHNA CHAIYE AND WOH FHIR DIKH JATA HAI)

ADV:
GIT LOGS ARE PRESENT, UNDO THE CHANGES, ETC

ACTUALLY LEARNING ABOUT THE GITOPS AND ARGOCD



1. CREATE A FRESH K8S CLUSTER and name is gitops
copy the config file

2. installing the argocd (it just means applying a very big manifest file)
kubectl create namespace argocd
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml

or also you can use DO marketplace

in the cluster settings directly install the argocd from the marketplace with a simple install button

3. created a ops repo

https://github.com/PDGamerSG/argo-deployment

4.now what we want is to access the argocd for that we can either use the ui or the commands
the commands are:
Install argocd CLI (not required but recommended)

```
brew install argocd
```

- Access the argocd ui

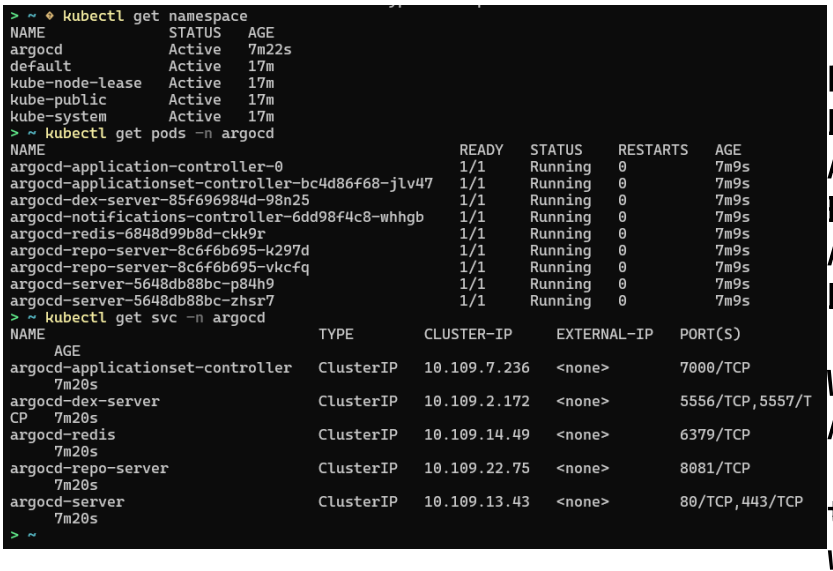
```
kubectl port-forward svc/argocd-server -n argocd 8080:443
```

- Get the argocd password

```
kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}"
```

- Debugging

```
kubectl logs -n argocd deployment/argocd-application-controller
```



BUT WE ARE USING SOMETHING ELSE LIKE YOU CAN SEE IN THIS THE ARGOCD EXPOSES A SERVER WHICH EXPOSED FE ALSO SO WE WANT TO ACCESS THAT WE CAN USE LOADBALANCER BUT WHY TO USE THAT

WE CAN USE SOMETHING NEW CALLED AS **PORT FORWARDING**

the argocd-server is the one which we want to port forward too

```
kubectl port-forward svc/argocd-server -n argocd 8080:443
```

use this command to port forward

now at localhost:8080
you can see the argocd ui

NOW IN THIS WE CAN TELL ARGOCD ISS REPO SAE CODE PULL KARNA HAI

username: admin
password: use this command
kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}"

now this password is in base64 encoded because its stored in the secrets.yml file but in configfile we store plain only
V0E0d3JISzBOcFhlTGpiZw==
convert that again use this website
https://www.base64decode.org/

don't base64encode use base64decode.org
OR USE

echo "pass" | base64 --decode

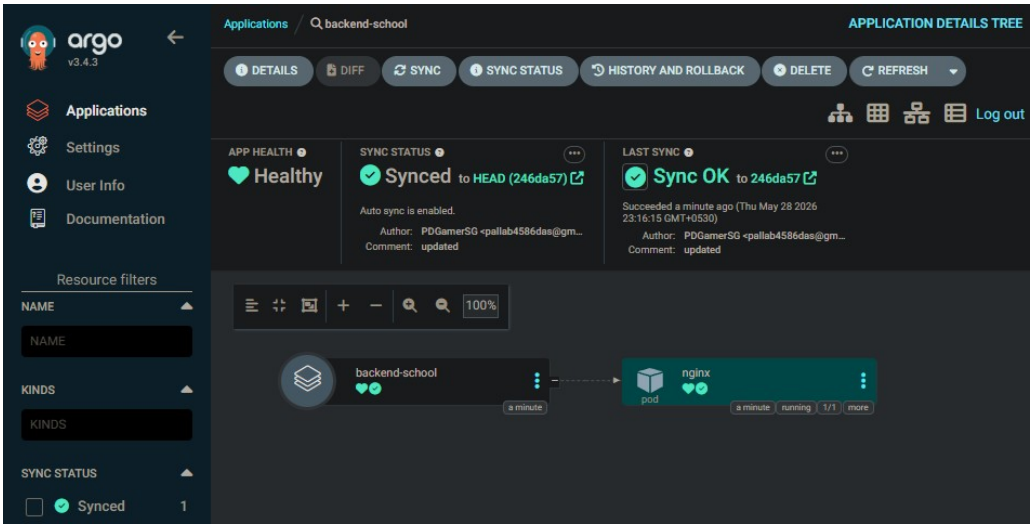
now you can login and see the UI

now create the project
choose name
backend-school
project name == default
sync policy == automatic (AND MANUAL MEANS DEV HAS TO CLICK AND APPROVE)
then there are some options
ENABLE AUTO SYNC ✅
PRUNE RESOURCES ✅ (this means if the yml repo is deleted the project will also be deleted)
SELF HEAL✅

NOW PUT THE REPO URL
PATH IS THE ROOT FOLDER FOR NOW PUT DOT .
CLUSTER URL == https://kubernetes.default.svc
NAMESPACE = DEFAULT
DIRECTORY RECURSE == IF THERE ARE OTHER FOLDERS INSIDE THE REPO THAT FILES WILL ALSO BE APPLIED

NOW CREATE

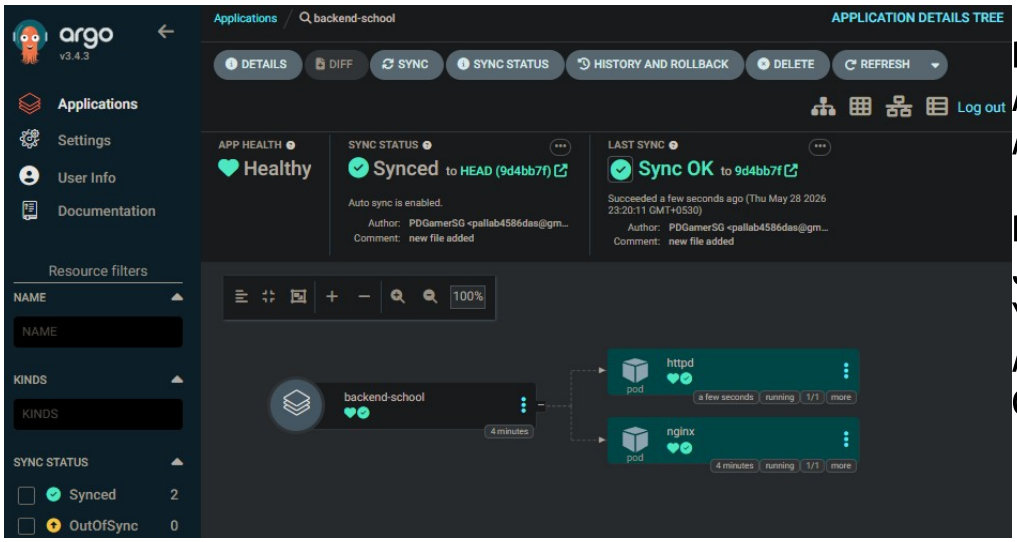
NOW CLICK ON HEALTHY
IT WILL SHOW WHAT ALL THINGS ARE RUNNING



now if you check in the kubectl get pods it will show nginx is running which automatically ran using the argo cd DAMNNNNNN

NOW IF WE MAKE CHANGES IN THE YML FILE

NOW IN THE ARGO CD CLICK ON SYNC TO GET THE NEW FILE



NOW YOU CAN SEE WE ADDED A NEW CONTAINER AND ITS ADDED

NOW ITS LIKE WE HAVE TO JUST MAKE CHANGES IN THE YML FILE AND AUTOMATICALLY THE CHANGES WILL HAPPEN

```
> argo-deployment git(main) kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
httpd   1/1     Running   0          89s
nginx   1/1     Running   0          5m26s
```

ALSO THIS SHOWS THAT

JUST THE USE OF THIS IS THE GIT COMMITS WILL KEEP A TRACK LIKE WHAT ALL CHANGES WERE DONE AT WHAT TIME

AND ALSO LIKE IF ONE REPO HAS TO PUSH SOMETHING TO ANOTHER REPO USING THE CI CD PIPELINE WE HAVE TO CREATE SOME TYPE OF PAT FILE AND THEN USE IT

CREATING A WHOLE WORKFLOW OF GITOPS

npx create-next-app@latest
create the next js application

now create a repo and push the code
and now create .github/workflows folder

now create a prob.yml and copy this

```
name: Continuous Deployment (Prod)

on:
  push:
    branches: [ "main" ]

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
        with:
          fetch-depth: 0

      - name: Docker login
        uses: docker/login-action@v2
        with:
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKER_SECRET }}

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v2

      - name: Build and push school image
        uses: docker/build-push-action@v4
        with:
          context: .
          file: Dockerfile
          push: true
          tags: pdgamersg/todo-app-week-39:${{ github.sha }}
```

now add the docker secrets in the github in the yml file
github
settings/secret and variables/actions
create secrets for the docker username, and secret

from the settings add these secrets

also create a docker file because the workflow needs that

A QUESTION LIKE JS, NODE IS SINGLE THREADED AND GOLANG IS MULTI THREADED WHICH MEANS IT CAN RUN A LONG TASK FASTER BY DIVIDING THE WORK INTO MULTIPLE CORES WHEREAS THE NODEJS CANNOT DO THAT AS IT CAN ONLY USE SINGLE CORE AT ONCE

CONCURRENCY MEANS IT CAN DO 2 DIFFERENT THINGS TOGETHER BUT IN A SINGLE CORE ONLY, LIKE AEK CHIJ THODI RUN HUI FHIR THODI RUN HUI JS IS CONCURRENT
WINDOWS RUNS THE APPLICATIONS CONCURRENTLY NOT PARRELELLY OR OTHERWISE WOULD HAVE REQUIRED LIKE 50 CORES

NOW ON THIS url the docker file is present

https://hub.docker.com/r/pdgamersg/todo-app-week-39

NOW TO DO IT MANUALLY JUST GO TO THE https://github.com/PDGamerSG/argo-deployment AND JUST ADD THIS ABOVE DOCKER LINK TO THAT

and the ARGO CD will automatically deploy that

NOW TO REMOVE THAT MANUAL STEP WE HAVE TO MAKE SOME CHANGES IN THE PROD.YML

```
    - name: Clone staging-ops repo, update image tags, and push tags
      env:
        PAT: ${{ secrets.PAT }}
      run: |
        git clone https://github.com/PDGamerSG/argo-deployment
        cd argo-deployment

        sed -i 's|image: 100xdevs/todo-app-week-39:.*|image: 100xdevs/todo-app-week-39:${{ github.sha }}|' manifest.yml

        git config user.name "GitHub Actions Bot"
        git config user.email "actions@github.com"
        git add .
        git commit -m "Update school server image tags to ${{ github.sha }}"
        git push https://${PAT}@github.com/pdgamersg/argo-deployment.git main
```

now for this we have to create a PAT(personal access token) for that go to first the account settings and then the developer settings and then the PAT TOKEN CLASSIC

create a token for the argo repository
and add that in the todo app secret

and then push the code
the actions success

```
> ~ kubectl logs -f nginx

> my-app@0.1.0 start
> next start

▲ Next.js 16.2.6
- Local:         http://localhost:3000
- Network:       http://10.108.1.105:3000
✓ Ready in 244ms
```

running now lets go

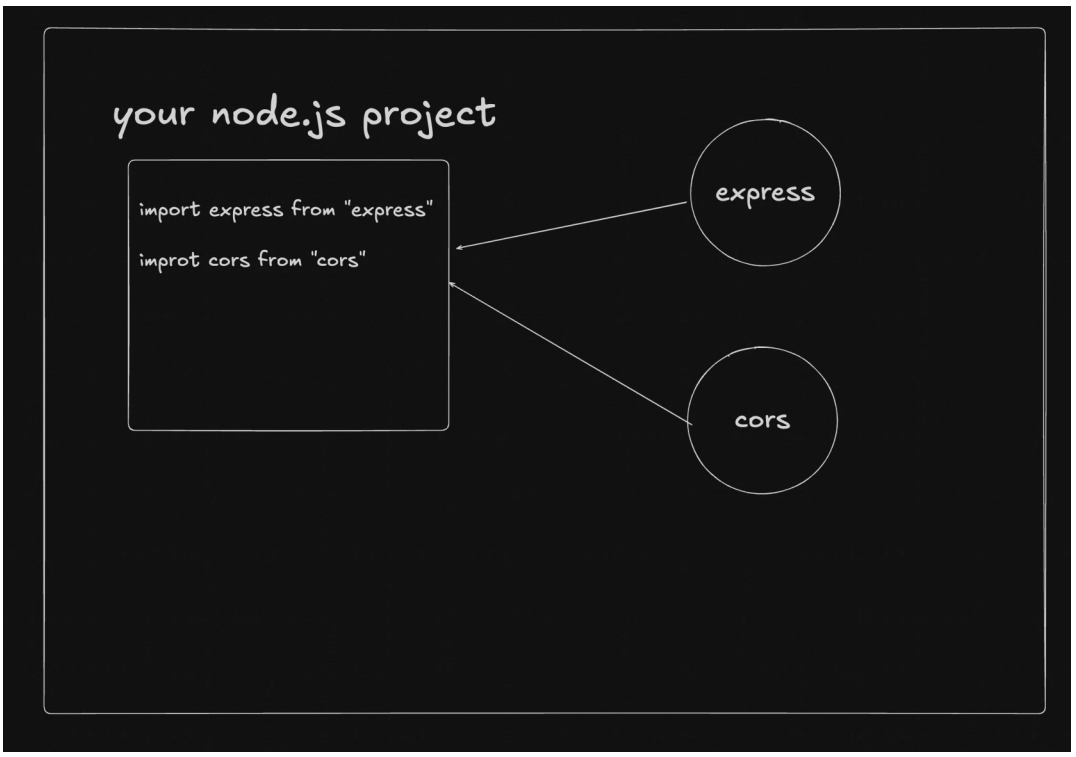
FOR LIKE ONCE YOU HAVE TO CHANGE THE URL IN THE MANIFEST.YML IN THE ARGO DEPLOYMENT REPO AND THEN ALSO IT STARTS TO CHANGE THE NAME

HELM

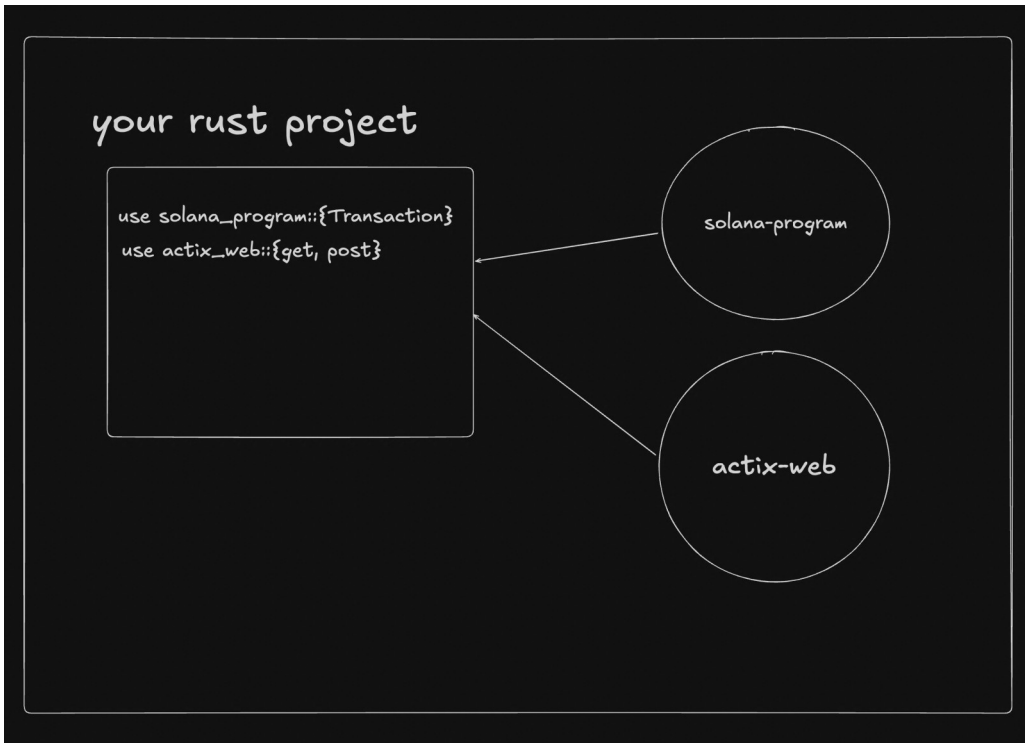
```
choco install kubernetes-helm
 cat .\clusters.yml
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
  - role: control-plane
  - role: worker
  - role: worker
kind delete cluster -n local
kind create cluster --config .\clusters.yml -n local
```

ITS LIKE A WAY OF MAKING THE LIFE EASY LIKE EARLIER IF WE HAVE TO START LIKE THE POSTGRESS WE HAVE TO CREATE SO MANY THINGS OURSELF SO TO FIX THAT WE CAN JUST USE THAT AND JUST MAKE SOME CHANGES TO THE CONFIG FILE AS PER OUR USAGE

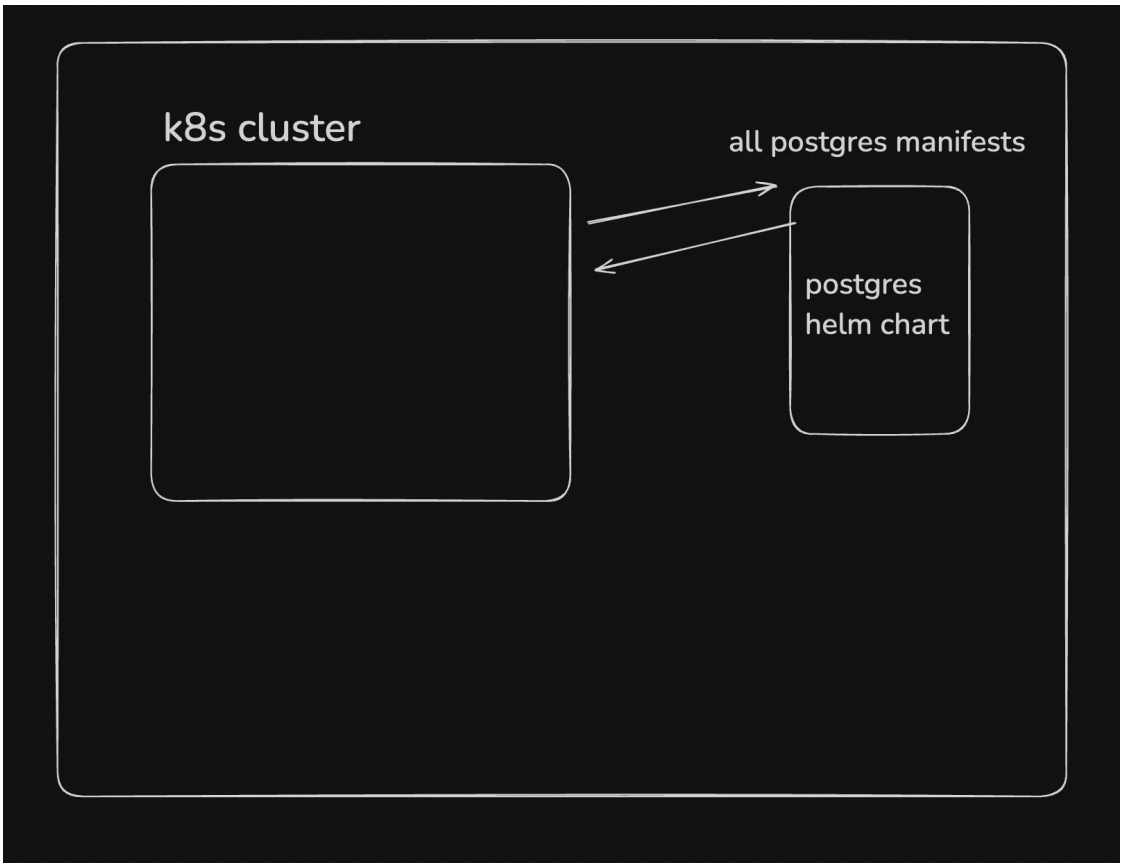
What is package management
We've used npm in the past. If lets you use packages that other people have written
Node.js
npmnpm: express



Rust
https://docs.rs/actix-web/latest/actix_web/



Why package management in k8s?
To start postgres in your k8s cluster, you will have to create
A deployment
A secret
A service
A PVC
A namespace
That file would look something like this
Wouldn’t it be nice if we can get all the postgres manifests from somewhere?



Experiment

1. Create a new k8s cluster
2. Install the helm cli
3. Install the postgres helm chart

```
helm install postgres
oci://registry-1.docker.io/bitnamicharts/postgresql
```

1. To see the files being applied locally, run

```
helm pull
oci://registry-1.docker.io/bitnamicharts/postgresql --untar
```

Chart registries

1. Bitnami - https://charts.bitnami.com/
2. Artifacts hub - https://artifacthub.io/

BEFORE REACT JS EJS WAS USED

TEMPLATE ARE USED TO CHANGE THE VALUES IN THE HELM CHARTS

Go templates

What are Go Template? (similar to ejs in Node.js)

Go Templates are a simple and powerful templating engine in the Go programming language. They are designed for creating text-based output (like YAML, HTML, etc.) by embedding Go logic inside placeholders. In Helm, Go Templates are used to generate Kubernetes YAML files dynamically based on input values and template logic.

- Go code to fill a templated HTML file –

```
package main

import (
        "html/template"
        "log"
        "os"
)

// User struct to hold individual user data
type User struct {
        Username string
        Gender   string  // "male" or "female"
}

// PageData struct with an array of users
type PageData struct {
        Title string
        Heading string
        Users []User
}

func main() {
        // Parse the template file
        tmpl, err := template.ParseFiles("template.html")
        if err != nil {
                log.Fatalf("error parsing template: %v", err)
        }

        // Define the data to pass to the template
        data := PageData{
                Title:   "User List",
                Heading: "List of Users",
                Users: []User{
                        {Username: "John Doe", Gender: "male"},
                        {Username: "Jane Smith", Gender: "female"},
                        {Username: "Alex Lee", Gender: "male"},
                        {Username: "Emily Davis", Gender: "female"},
                },
        }

        // Open a file to write the rendered HTML
        outputFile, err := os.Create("output.html")
        if err != nil {
                log.Fatalf("error creating output file: %v", err)
        }
        defer outputFile.Close()

        // Execute the template with the provided data
        err = tmpl.Execute(outputFile, data)
        if err != nil {
                log.Fatalf("error executing template: %v", err)
        }

        log.Println("Template rendered successfully, check the 'output.html' file!")
}
```

- Template file

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{{ .Title }}</title>
</head>
<body>
    <h1>{{ .Heading }}</h1>
    <ul>
        {{ range .Users }}
            <li>
                {{ if eq .Gender "male" }}
                    Mr. {{ .Username }}
                {{ else if eq .Gender "female" }}
                    Mrs. {{ .Username }}
                {{ else }}
```

```
                {{ .Username }}  <!-- If no gender is provided, show just the name -->
            {{ end }}
        </li>
    {{ else }}
        <li>No users found</li>
    {{ end }}
</ul>
</body>
</html>
```

- Go code to serve the html

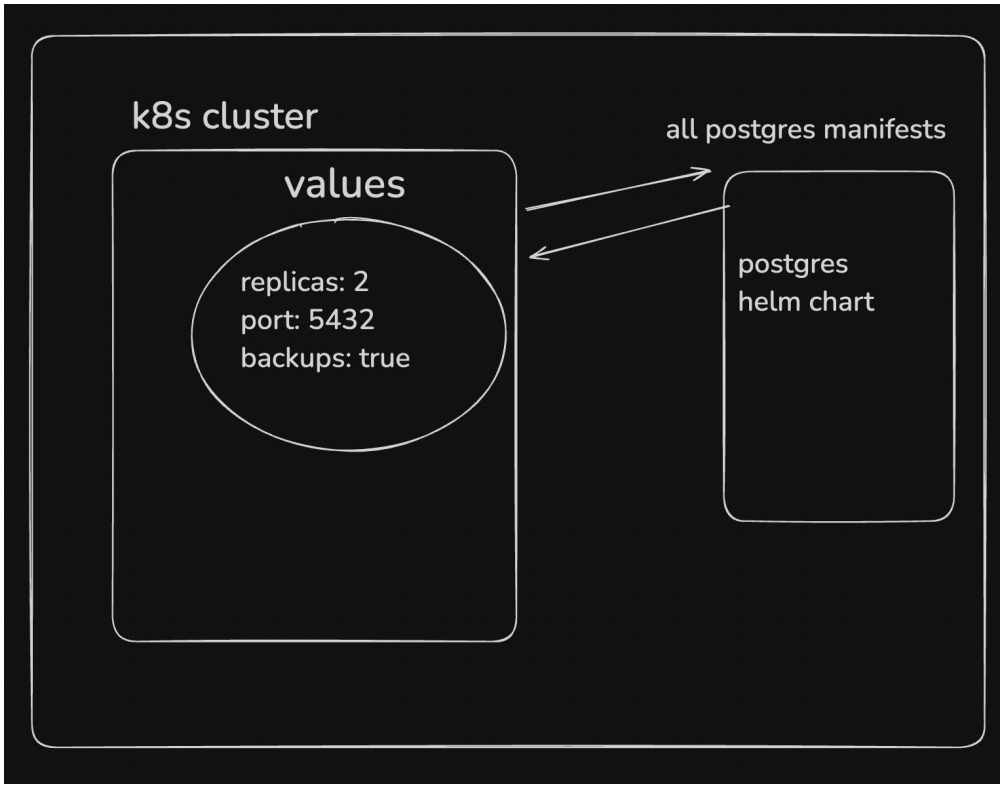
```
package main

import (
	"html/template"
	"log"
	"net/http"
)

// User represents a user in the system
type User struct {
	Username string
	Gender   string // "male", "female", or others
}

// PageData holds the data passed into the template
type PageData struct {
	Title   string
	Heading string
	Users   []User
}

func main() {
	// Load the HTML template once at startup
	tmpl, err := template.ParseFiles("template.html")
	if err != nil {
		log.Fatalf("Failed to parse template: %v", err)
	}

	// HTTP handler function
	http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
		// Example data
		data := PageData{
			Title:   "User Directory",
			Heading: "Welcome to the User Directory",
			Users: []User{
				{Username: "John Doe", Gender: "male"},
				{Username: "Jane Smith", Gender: "female"},
				{Username: "Alex Kim", Gender: "male"},
				{Username: "Emily Nguyen", Gender: "female"},
				{Username: "Jordan Lee", Gender: "non-binary"},
			},
		}

		// Render the template to the response
		err := tmpl.Execute(w, data)
		if err != nil {
			http.Error(w, "Template execution error", http.StatusInternalServerError)
			log.Println("Template error:", err)
		}
	})

	// Start HTTP server
	log.Println("Serving on <http://localhost:8080> ...")
	log.Fatal(http.ListenAndServe(":8080", nil))
}
```

Basic Syntax

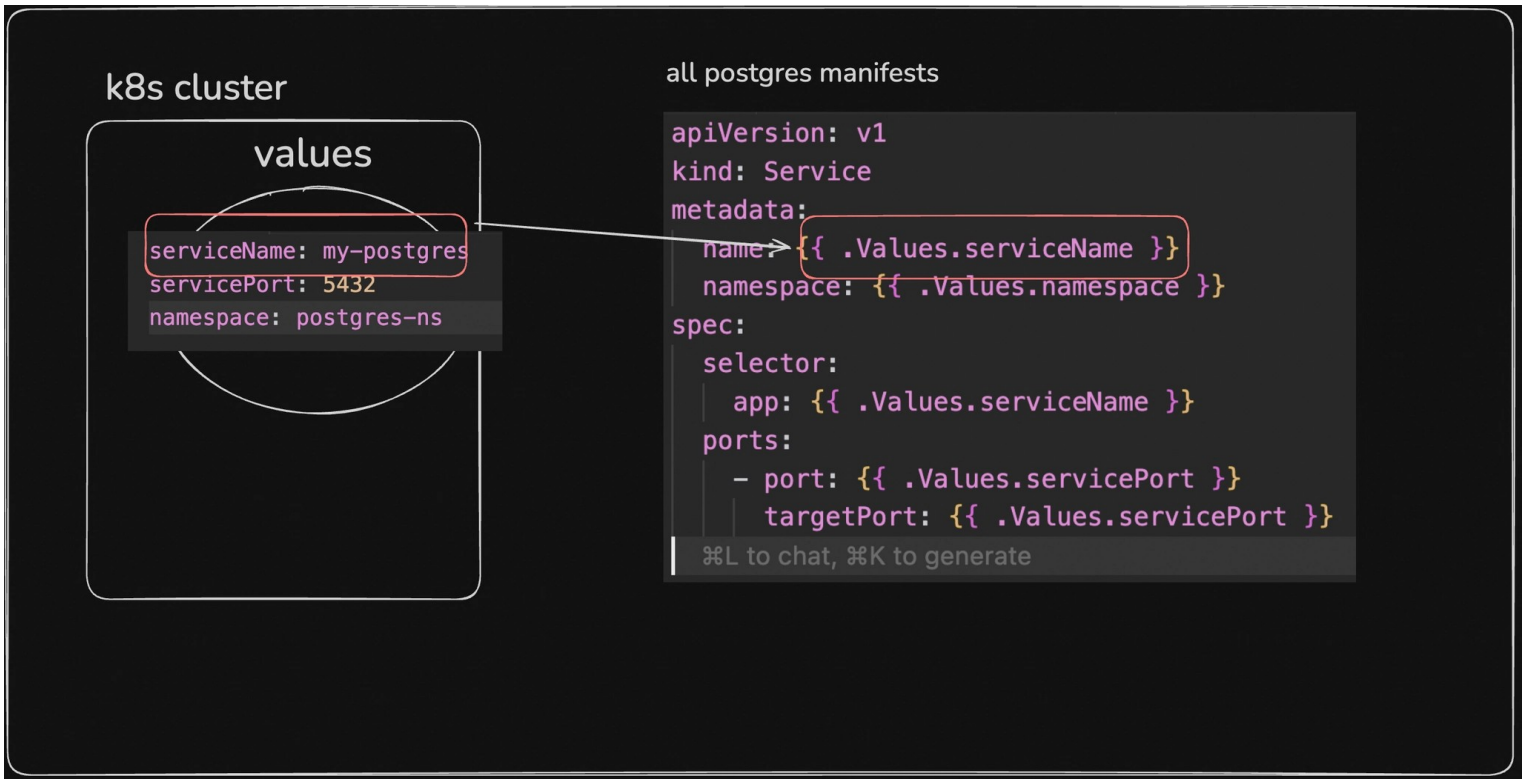
In Helm, templates are defined using the following structure:

- **Start of a Template**: {{ ... }}
- **Accessing Variables**: {{ .SomeVariable }}
- **Control Flow**:
    - {{ if .Condition }} ... {{ else }} ... {{ end }}
    - {{ range .List }} ... {{ end }}

Why templates?
There cant be a generic chart (package) that you can create that caters to everyones need.
Hence, when a chart is created, most values are injected in via templates



More technically



Creating your own chart

Chart.yaml

This file is the **metadata manifest** for your Helm chart.

For example

```
apiVersion: v2
name: backend-app
description: A Helm chart for deploying backend of our app
version: 0.1.0
appVersion: "1.16.0"
```

WHY DOUBLE QUOTES FOR THE APP VERSION??

BECAUSE THE VALUE OF IF ITS LIKE 2.2.10 THEN GOLANG WILL TAKE IT AS 2.2.1 SO FOR THAT WE ADD THEM IN DOUBLE QUOTES TO MAKE IT A STRING

values.yaml

This file contains **default values** for your chart's configuration — like image tags, replica counts, or resource limits.

For example

```
replicaCount: 2


image:
  repository: nginx
  tag: "1.21"
  pullPolicy: IfNotPresent


service:
  type: ClusterIP
  port: 80
```

templates/ folder

This is where the **actual Kubernetes resource files** (Deployment, Service, etc.) are defined — but written using Go **templating syntax**.

deployment.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Values.serviceName }}
spec:
  replicas: {{ .Values.replicaCount }}
  template:
    spec:
      containers:
        - name: my-app
          image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
```

Creating our own postgres chart

- Initialise a helm chart

  mkdir my-postgres-chart

  cd my-postgres-chart

  helm create .

- Remove the default templates

  rm -rf templates/*

- Update Chart.yml

  apiVersion: v2

  name: my-postgres

  description: A simple PostgreSQL Helm chart

  version: 0.1.0

  appVersion: "15"

- Create values.yml

  postgresUser: user

  postgresPassword: password

  postgresDatabase: mydb

  namespace: postgres

  image:

      repository: postgres

      tag: "15"

- Create templates/deployment.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Values.name }}
  namespace: {{ .Values.namespace }}
spec:
  replicas: 1
```

```
  selector:
    matchLabels:
      app: {{ .Values.name }}
  template:
    metadata:
      labels:
        app: {{ .Values.name }}
    spec:
      containers:
        - name: postgres
          image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
          env:
            - name: POSTGRES_USER
              value: {{ .Values.postgresUser }}
            - name: POSTGRES_PASSWORD
              value: {{ .Values.postgresPassword }}
            - name: POSTGRES_DB
              value: {{ .Values.postgresDatabase }}
          ports:
            - containerPort: 5432
```

- Create templates/0-namespace.yml

```
apiVersion: v1
kind: Namespace
metadata:
  name: {{ .Values.namespace }}
```

- Create templates/service.yml

```
apiVersion: v1
kind: Service
metadata:
  name: {{ include "my-postgres.fullname" . }}
  namespace: {{ .Values.namespace }}
spec:
  selector:
    app: {{ include "my-postgres.name" . }}
  ports:
    - port: 5432
      targetPort: 5432
```

- Install the chart

```
helm install postgres-chart .
```

- Change something

```
helm upgrade postgres-chart .
```

Explore the pods and svc in the postgres namespace

- Debug the helm chart

```
helm template postgres .
```

Assignment

1. Should you use my-postgres.name or my-postgres.fullname for the spec.selector.app ?
2. How can you start two different apps?
3. Can you make the ports templated as well?

ITS JUST LIKE CREATE A TEMPLATE AND JUST CREATE THE VALUES FILE AND APPLY AGAIN

CMD TO USE THAT

```
helm install postgres-chart4 .
```

Each time change this

with the values

```
helm install posgresc-chart4 . --values values-cms.yml
```

Why would you do this?

1. Using helm charts - Make it easy to install things like postgres, nginx ingress controller, caddy ingress controller etc
2. Creating helm charts - Re-usability

Ref -

1. https://github.com/code100x/staging-ops/blob/main/prod/dailycode/deployment.yml
2. https://github.com/code100x/staging-ops/blob/main/prod/cms/deployment.yml
3. https://github.com/code100x/staging-ops/blob/main/prod/photo-ai/deployment.yml

Assignment

Create a helm chart to create a deployment for a backend, and create 3 different Charts

Hint

```
helm install app1 ./chart -f photo-ai-backend-values.yaml
```

```
helm install app2 ./chart -f probo-backend-values.yaml
```

```
helm install app3 ./chart -f chess-backend-values.yaml
```

Tomorrow

1. Helm with argocd
2. Publishing your charts
3. Chart dependencies

THIS IS USED TO REMOVE THE REUSING OF THE DATA