In this module, we are adjusting the resources and preparing the deployment.

You should have your Git Repository with the kubernetes resources ready and the resources deployed.

Task 1: Adjusting Resources and Enabling Autoscaling

You just deployed the kubernetes resources. However, you do have any resource associated to the pods. As a consequence, the schedule has no knowledge how to distribute the load plus the pods are not covered by any kind of QoS-guideline (Please read https://kubernetes.io/docs/tasks/configure-pod-container/quality-service-pod/ for more information).

1. Adapt your kubernetes resources by adding limits and resources to the deployment of each pod:
2. Decide what QoS you want to implement.
3. Find the correct resources. `kubectl top` helps you out
4. Adapt your deployment-configs: https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/ tells you where to adapt.
5. Apply the settings via `kubectl apply -f FILE`

You will see that the rolling deployment takes place where a new pod is started while the other is still running. The moment, the traffic can be switched, the old pods are going to be terminated. Bear in mind that it takes double resources why you should only patch one pod at one time.

You may use additional tooling such as the Vertical Pod Autoscaler(https://learn.microsoft.com/en-us/azure/aks/vertical-pod-autoscaler) and/or Goldilocks (https://goldilocks.docs.fairwinds.com/) to define the resources if you want to…you can make based on your assumption as well.

Task 2: Generate Helm-Resources

Since we do not want to repeat ourselves, we use Helm for templating and generating resource for different stages. Your existing kubernetes resources are good boilerplate code for developing a helm chart for the chatbots. Develop one as start for implementing deployment strategies and stages:

1. Create a new helm chart e.g. with `helm create chatops-helm`.
2. Make a git-repository out of it. `git init` helps in the folder. Create a remote-repository and bind the local one to the remote repository: `git remote add origin GITURL`
3. Develop the chart. Think about what elements might differ especially when you deploy the application multiple times. Think about the chapter of configuration externalization. Variable

configuration that might differ between stages should be templated. The initialized helm chart should give you a smooth ramp up regarding externalizing configurations.
Tips for developing:
- Use an IDE with syntax highlighting, templating and checking of the yaml. IntelliJ with the kubernetes plugin does the job.
- Let `helm lint` run in the project: it checks the chart while developing it.
- Use `helm template` for generating the k8s-resources out of the template to compare with the k8s-resources

4. When done, deploy it via `helm install RELEASENAMETOCHOOSE . -n NAMESPACE.`
   When not successful, you can uninstall it via `helm uninstall RELEASENAMETOCHOOSE -n` NAMESPACE.
   If you want to apply an update, upgrade the chart with `helm upgrade FORMERRELEASENAME . -n NAMESPACE`

Task 3: Deploy and enable ArgoCD.

To overcome the need to trigger helm each time something changes in the Git, we use ArgoCD to pull changes from the Git: With this approach, we can listen directly to the helm chart.

You can install argocd directly via helm:

```
kubectl create ns argocd
helm repo add argo https://argoproj.github.io/argo-helm
helm install argocd argo/argo-cd -n argocd
```

After the creation, make a port forwarding from the `argocd-server` - instance to localhost.

The user is admin while the password can be retrieved via

```
kubectl get secret argocd-initial-admin-secret -o
jsonpath="{.data.password}" -n argocd | base64 -d
```

To access Argocd, you have two possibilities:

1. Create a port forward

2. Switch the Service to type “LoadBalancer”. Refer to https://argo-cd.readthedocs.io/en/stable/getting_started/#3-access-the-argo-cd-api-server .

After login, you can register your helm chart with the following steps:

1. Go to Settings and register a repository. The repository should be your helm-repository. Create a link to this repository, you might create a new token similar to the pull secret in your gitlab-subgroup.

2. Create a namespace for the deployment e.g. `kubectl create ns chatbots-helm`

3. Create an app from within argocd. Choose the given namespace and the repo.