ITI8801 · Cloud Architectures and DevOps

Practice 8 · Assignment 8: a Helm chart, configuration and autoscaling

Siim Rebane · TalTech · Autumn 2026

What you build

Your Assignment 7 project, changed in four ways:

Part	Written by you
a **Helm chart**	the manifests of your service and its database as templates, with a values file per environment; the pipeline deploys with `helm upgrade --install`
a **ConfigMap**	made by the chart; your service reads a setting from it and reports it at `GET /info`
a **HorizontalPodAutoscaler**	in staging only: between 2 and 5 pods, by CPU utilisation
a **PodDisruptionBudget**	in staging only: one pod always stays

The two environments, their host names, the pipeline's five jobs and the Assignment 5 API stay as they are in Assignment 7.

The chart and the releases

Requirement	Detail
a chart in the repository	a directory with `Chart.yaml`, `values.yaml` and `templates/`
one values file per environment	for dev and for staging; they hold what differs
the deploy jobs	`deploy-dev` and `deploy-staging` run `helm upgrade --install`; no `kubectl apply` of manifests in the pipeline
two releases	the objects of dev belong to one Helm release, the objects of staging to another, both from the same chart
the image tag	set by the pipeline for every deploy: the commit's short SHA, as before
the database passwords	the Secrets you made by hand stay as they are; no Secret manifest in the chart

The chart contains everything of an environment: the Deployment, its Service, the Ingress, the ConfigMap, the database's StatefulSet and Service, and for staging the HPA and the PDB.

The configuration

Your service gets one new endpoint:

```
GET /info        →  200   {"environment": "<value>"}
```

Requirement	Detail
the value	the content of an environment variable of your service, e.g. `ENVIRONMENT`
where it comes from	a ConfigMap that the chart creates from the values; the Deployment reads it with `envFrom` or `configMapKeyRef` (a volume alone sets no variable)
in dev	`{"environment": "dev"}`
in staging	`{"environment": "staging"}`
the image	the same in both environments: the value is not in the image and not a literal `value:` in the Deployment template

The JSON may hold more fields.

Scaling and disruptions, in staging

Object	Requirements
a **HorizontalPodAutoscaler**, `autoscaling/v2`	for staging's Deployment; `minReplicas: 2`; `maxReplicas` between 3 and 5; one metric: CPU utilisation with a target you choose
the Deployment's containers	a CPU request, so that utilisation can be computed
the Deployment's manifest	no fixed `replicas` when the HPA manages it
a **PodDisruptionBudget**	selects staging's service pods; `minAvailable: 1`

Dev has neither: the same chart with another values file leaves them out.

Under load the HPA must act: while 20 requests at a time arrive at staging's host for 90 s, the HPA raises the Deployment above 2 replicas. Your CPU request and your target decide whether it does.

The end state, checked

- the Assignment 5 API answers at both host names, and `GET /info` answers `dev` and `staging`
- the Deployments, Services, Ingresses, ConfigMaps and StatefulSets of both environments carry Helm's label and release annotations; dev and staging are two releases
- each Deployment reads its ConfigMap
- staging has an HPA with a current utilisation, and a PDB that allows one disruption; dev has neither
- during the load the HPA's wanted replicas rise above 2
- the commits that run in dev and staging have green pipelines, as in Assignment 7
- the repository holds the chart and no credential

The validator

`https://iti8801.digivader.com/lab8-validator/` : your student code and your GitLab project.

Box	Green when
API	the A5 API test and `GET /info` answer at both hosts; `environment` is `dev` and `staging`
Releases	the Deployment, Service, Ingress, ConfigMap and StatefulSet behind each host carry Helm's label and release annotation, dev and staging as two releases; the repository has a chart (`Chart.yaml` , `values.yaml` , `templates/` with a Deployment) and a values file per environment; `deploy-dev` and `deploy-staging` run `helm upgrade` and no `kubectl apply` ; no Secret manifest, kubeconfig or token in the repository
Configuration	each Deployment reads a ConfigMap of its release; the ConfigMap holds the value that `/info` answers
Scaling	staging's HPA: `autoscaling/v2` , min 2, max 3 to 5, one metric, CPU utilisation with a current value; a CPU request on every container of the Deployment; dev has no HPA. Then the load: 20 requests at a time to staging's host for up to 90 s; green when the HPA wants more than 2 replicas. Not sent while the HPA still wants more than 2 from an earlier load
Disruption	staging's PDB selects the service pods, `minAvailable: 1` , allows a disruption now; dev has none
Pipeline	as in Assignment 7: the project is private, named `iti8801-2026…` , with `Siim.Rebane` as a Developer member; the commit dev runs went build, test, scan, deploy-dev on `main` ; the commit staging runs came through a tag pipeline without a build; staging runs dev's image repository

On the walkthrough's service, 20 requests at a time were 80 to 130 requests per second, and the HPA wanted 4 replicas within 30 s. The validator sends the load once per check, runs one check per namespace at a time, and creates and changes nothing in your namespace or your project.

The quota

The HPA can raise staging to 5 service pods. Everything in the namespace still shares one quota:

Resource	Quota	Staging at 5 pods needs, with the sizes of the walkthrough's service
CPU requests	500m	5 × 25m + 50m for the database
CPU limits	6	5 × 250m + 250m
memory limits	6Gi	5 × 256Mi + 512Mi
pods	20	6

A pod the quota refuses is not created: the HPA shows 5 replicas and `kubectl get pods` shows fewer. `kubectl describe quota` shows the sums.

Your Assignment 6 objects may be deleted once your Assignment 6 check is PASSED; they hold 1500m of CPU limits with the sizes of practice 6.

What to submit, and when

- The validator's code of a check that is **PASSED**. Paste it into Moodle.
- Deadline: Sunday next week, 23:59, as for every assignment.
- A check is stored with its result: what you change in the project or the namespace afterwards does not change it.

Walkthrough

The steps in order, with the commands and what each should print. The outputs come from the test namespace `00007test` and its project. Replace what is yours (`<…>`).

Step 0 · Helm

System	Command
macOS	`brew install helm`
Windows	`winget install Helm.Helm`
Linux	the script or the binary from `helm.sh/docs/intro/install`

```
$ helm version --short
v4.3.0+gbec5b06
```

Helm reads the same kubeconfig as `kubectl`: with `KUBECONFIG` set as in practice 6, `helm list` answers with the releases of your namespace, at first none.

The pipeline's image `alpine/k8s:1.36.5` contains Helm 4.3. Helm 3, the previous version and the one most tutorials still show, works with the same chart; it calls `--rollback-on-failure` `--atomic`, and that is the only difference you will meet.

Step 1 · the endpoint

Add `GET /info` to your service: it answers with the value of an environment variable.

```
@app.get("/info")
def info():
    return {"environment": os.environ.get("ENVIRONMENT", "")}
```

Add a line for it to `tests/smoke.sh`, commit, push. The pipeline builds, tests, scans and deploys to dev as before.

The variable is not set anywhere yet: `/info` answers with an empty value. The next steps set it through the chart.

Step 2 · the chart's frame

```
chart/
├── Chart.yaml
├── values.yaml
└── templates/
values-dev.yaml
values-staging.yaml
```

```yaml
# chart/Chart.yaml
apiVersion: v2
name: <a name for your application>
description: <one line>
version: 0.1.0
```

```yaml
# chart/values.yaml
image:
  repository: <your image repository>
  tag: ""
host: ""
replicas: 1
config:
  ENVIRONMENT: ""
autoscaling:
  enabled: false
  minReplicas: 2
  maxReplicas: 5
  cpuUtilization: 60
```

The names of the values are yours to choose; the templates of the next steps use these.

Step 3 · the templates

Move your manifests from `k8s/` into `chart/templates/`, one file per object or group, and replace the placeholders:

In the manifests of Assignment 7	In the templates
`$ENV`	`{{ .Release.Name }}`
`$HOST`	`{{ required "host is not set" .Values.host }}`
`$IMAGE`	`{{ .Values.image.repository }}:{{ required "image.tag is not set" .Values.image.tag }}`
`replicas: $REPLICAS`	the three lines below

```
spec:
  {{- if not .Values.autoscaling.enabled }}
  replicas: {{ .Values.replicas }}
  {{- end }}
```

The release will be named `dev` or `staging`, so `api-{{ .Release.Name }}` gives the names the objects already have, and `secretKeyRef: {name: db-{{ .Release.Name }}, key: password}` finds the Secrets you made for Assignment 7.

The database's volume claim is named after the StatefulSet. With the same StatefulSet name as before, the new StatefulSet finds the old claim and its data.

Step 4 · the ConfigMap

`chart/templates/configmap.yaml`:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: api-{{ .Release.Name }}
data:
  {{- range $key, $value := .Values.config }}
  {{ $key }}: {{ $value | quote }}
  {{- end }}
```

In the Deployment template, the container reads every key as an environment variable, and the pod template carries the ConfigMap's checksum:

```
  template:
    metadata:
      labels:
        app: api-{{ .Release.Name }}
      annotations:
        checksum/config: {{ include (print $.Template.BasePath "/configmap.yaml") . | sha256sum }}
    spec:
      containers:
      - name: service
        envFrom:
        - configMapRef:
            name: api-{{ .Release.Name }}
```

The annotation changes when a value changes, so the Deployment replaces its pods and the new ones read the new value.

Step 5 · the HPA and the PDB

`chart/templates/hpa.yaml`, both objects under one condition:

```
{{- if .Values.autoscaling.enabled }}
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: api-{{ .Release.Name }}
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: api-{{ .Release.Name }}
  minReplicas: {{ .Values.autoscaling.minReplicas }}
  maxReplicas: {{ .Values.autoscaling.maxReplicas }}
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: {{ .Values.autoscaling.cpuUtilization }}
---
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: api-{{ .Release.Name }}
spec:
  minAvailable: 1
  selector:
    matchLabels:
      app: api-{{ .Release.Name }}
{{- end }}
```

Step 6 · the values files, and a look at the result

```yaml
# values-dev.yaml
host: <code>-dev.k8s.digivader.com
config:
  ENVIRONMENT: dev
```

```yaml
# values-staging.yaml
host: <code>-staging.k8s.digivader.com
config:
  ENVIRONMENT: staging
autoscaling:
  enabled: true
```

```
$ helm lint chart -f values-dev.yaml --set image.tag=0fa0b979
==> Linting chart
[INFO] Chart.yaml: icon is recommended

1 chart(s) linted, 0 chart(s) failed
$ helm template dev chart -f values-dev.yaml --set image.tag=0fa0b979 | less
$ helm template staging chart -f values-staging.yaml --set image.tag=0fa0b979 | grep '^kind:'
```

Read the rendered manifests of both: dev has a `replicas:` line and no HPA; staging has the HPA and the PDB and no `replicas:` line. `helm template` contacts no cluster.

Step 7 · dev becomes a release

Helm does not take over objects that `kubectl apply` created:

```
$ helm upgrade --install dev chart -f values-dev.yaml --set image.tag=0fa0b979
Error: unable to continue with install: Service "db-dev" in namespace "000007test" exists and cannot be imported
  into the current release: invalid ownership metadata; label validation error: missing key
  "app.kubernetes.io/managed-by": must be set to "Helm"; …
```

The error names the first object it meets. Every object of dev that `kubectl` made has to go, the ConfigMap too if you made one by hand. Not the volume claim, not the Secret: the data is in the claim, and the new StatefulSet, same name, picks it up again.

```
$ kubectl delete deployment,service,ingress api-dev
$ kubectl delete statefulset,service db-dev
$ kubectl delete configmap api-dev
$ kubectl get pvc data-db-dev-0
NAME            STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
data-db-dev-0   Bound    pvc-2ba786ef-88a7-4310-8c7e-8915fcb390cf   1Gi        RWO            local-path     35m
```

Install, with the tag of an image that exists in your registry:

```
$ helm upgrade --install dev chart -f values-dev.yaml --set image.tag=0fa0b979 --wait --timeout 2m
Release "dev" does not exist. Installing it now.
NAME: dev
STATUS: deployed
REVISION: 1
$ helm list
NAME   NAMESPACE    REVISION   UPDATED                STATUS     CHART
dev    000007test   1          2026-10-05 23:02:26   deployed   boardgames-0.1.0
$ curl -s http://000007test-dev.k8s.digivader.com/health
{"status":"ok"}
```

Step 8 · the pipeline deploys with Helm

```
.deploy:
  stage: deploy
  image: alpine/k8s:1.36.5
  script:
    - helm upgrade --install $ENV chart -f values-$ENV.yaml --set image.tag=$CI_COMMIT_SHORT_SHA --wait --timeout 2m

deploy-dev:
  extends: .deploy
  variables: { ENV: dev }
  environment: { name: dev, url: "http://<code>-dev.k8s.digivader.com" }
  rules:
    - if: $CI_COMMIT_BRANCH == "main"
```

`deploy-staging` likewise, with `ENV: staging` and its rule. Delete the directory `k8s/`, commit the chart, the values files and the changed `.gitlab-ci.yml`, push.

```
$ helm upgrade --install $ENV chart -f values-$ENV.yaml --set image.tag=$CI_COMMIT_SHORT_SHA --wait --timeout 2m
Release "dev" has been upgraded. Happy Helming!
STATUS: deployed
REVISION: 2
Job succeeded
```

```
$ curl -s http://000007test-dev.k8s.digivader.com/info
{"environment":"dev"}
```

Step 9 · staging becomes a release

Delete staging's objects that `kubectl` made, as in Step 7, with `-staging` in the names, and the HPA and the PDB if you applied them by hand. Then tag the commit whose pipeline on `main` is green:

```
$ kubectl delete deployment,service,ingress api-staging
$ kubectl delete statefulset,service db-staging
$ kubectl delete configmap,hpa,pdb api-staging
$ git tag v2.0
$ git push origin v2.0
```

After the tag pipeline:

```
$ helm list
NAME      NAMESPACE    REVISION   UPDATED                   STATUS     CHART
dev       000007test   2          2026-10-05 20:04:20 UTC   deployed   boardgames-0.1.0
staging   000007test   1          2026-10-05 20:05:39 UTC   deployed   boardgames-0.1.0
$ curl -s http://000007test-staging.k8s.digivader.com/info
{"environment":"staging"}
$ kubectl get hpa,pdb
NAME                                                REFERENCE                TARGETS        MINPODS   MAXPODS   REPLICAS   AGE
horizontalpodautoscaler.autoscaling/api-staging   Deployment/api-staging   cpu: 20%/60%   2         5         2          32m

NAME                                       MIN AVAILABLE   MAX UNAVAILABLE   ALLOWED DISRUPTIONS   AGE
poddisruptionbudget.policy/api-staging   1               N/A               1                     32m
```

`TARGETS` shows `<unknown>` for the first minute. Staging does not answer between the deletion and the end of the tag pipeline.

Step 10 · see it scale

A pod in your namespace that requests staging's Service in four endless loops:

```
$ kubectl run load --image=busybox:1.37 --restart=Never -- sh -c \
   'for i in 1 2 3 4; do (while true; do wget -q -O /dev/null http://api-staging/api/<resource>; done) & done; wait'
pod/load created
$ kubectl get hpa api-staging          # repeated every 15 s
NAME          REFERENCE                TARGETS         MINPODS   MAXPODS   REPLICAS   AGE
api-staging   Deployment/api-staging   cpu: 22%/60%    2         5         2          70s
api-staging   Deployment/api-staging   cpu: 680%/60%   2         5         2          85s
api-staging   Deployment/api-staging   cpu: 884%/60%   2         5         4          100s
api-staging   Deployment/api-staging   cpu: 778%/60%   2         5         5          116s
$ kubectl top pods -l app=api-staging
$ kubectl delete pod load --now
```

`busybox` is a 5 MB image with a shell and `wget`; `--restart=Never` makes a plain pod that is not started again when its command ends. `kubectl get hpa api-staging --watch` prints a line at every change of the object, until Ctrl-C. The first reaction comes 45 to 60 s after the load began. After the load pod is deleted the HPA waits 5 minutes before it removes pods.

You see	Look at
`TARGETS` stays `<unknown>`	the containers' CPU request; `kubectl describe hpa`
utilisation rises, `REPLICAS` rises, pods do not	`kubectl describe quota`; the load pod itself counts
utilisation stays under the target	your service needs little CPU for this load: a lower CPU request or a lower target; `kubectl top pods` shows the use

Step 11 · the validator

- `https://iti8801.digivader.com/lab8-validator/` : your student code and your project.
- Six boxes. The Scaling box sends load to staging for up to 90 s and watches the HPA; start a check when staging is at 2 replicas and no load pod runs: 5 minutes after your own load, or after a check. A check takes about two minutes.
- Paste the code of the **PASSED** check into Moodle.

```
$ helm list
$ helm history staging
$ kubectl get all,configmap,hpa,pdb -l app.kubernetes.io/managed-by=Helm
```

show the two releases, the revisions of one, and every object Helm manages.

When something does not work

You see	Look at
`exists and cannot be imported into the current release`	an object with that name was not made by Helm: delete it (Step 7); never the volume claim
`Error: … <message of a required value>`	a value is missing: the values file, the `--set` in the job
`error converting YAML to JSON`	`helm template … --debug` prints the rendered text with the broken line; usually the indentation around an `if` or `range`
`UPGRADE FAILED: … context deadline exceeded`	the new pods did not become ready in time: `kubectl get pods`, `kubectl describe pod`
a release in the status `failed` or `pending-upgrade`	`helm history <release>`, then `helm rollback <release> <revision>` to the last `deployed` one
`/info` answers an empty value	the ConfigMap's key against the variable your service reads; `kubectl exec deployment/api-dev -- env`
the pods are not replaced after a changed value	the checksum annotation is missing in the pod template
after every deploy staging drops to 1 pod for a moment	a `replicas:` line reaches the Deployment while the HPA manages it

What stays where

Thing	Lives in
the chart, the values files, `.gitlab-ci.yml`, the source	the repository
what differs between dev and staging	`values-dev.yaml` and `values-staging.yaml`
the image tag of a deploy	given by the pipeline with `--set`, recorded in the release
the history of each release	Secrets of type `helm.sh/release.v1` in your namespace; do not delete them
the database passwords	the Secrets `db-dev` and `db-staging`, made by hand, untouched by Helm
the databases' data	the volume claims, untouched by `helm upgrade` and by `helm uninstall`



Questions