Merging Everything into the Same YAML Definition

In this lesson, we will merge all the definitions explored until now into a single YAML file and then create objects using that file.

WE'LL COVER THE FOLLOWING

Looking into the Merged File

Consider this lesson a short intermezzo. We'll merge the definitions we used in this chapter into a single YAML file.

You already had a similar example before, so there's no need for lengthy explanations.

```
cat deploy/go-demo-2.yml
```

The **output** is as follows.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: go-demo-2-db
  labels:
    type: db
    service: go-demo-2
    vendor: MongoLabs
spec:
  selector:
    matchLabels:
      type: db
      service: go-demo-2
  strategy:
    type: Recreate
  template:
```

```
    metadata:
      labels:
        type: db

        service: go-demo-2
        vendor: MongoLabs
    spec:
      containers:
      - name: db
        image: mongo:3.3

---

apiVersion: v1
kind: Service
metadata:
  name: go-demo-2-db
spec:
  ports:
  - port: 27017
  selector:
    type: db
    service: go-demo-2

---

apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: go-demo-2-api
  labels:
    type: api
    service: go-demo-2
    language: go
spec:
  replicas: 3
  selector:
    matchLabels:
      type: api
      service: go-demo-2
  template:
    metadata:
      labels:
        type: api
        service: go-demo-2
        language: go
    spec:
      containers:
      - name: api
        image: vfarcic/go-demo-2
        env:
        - name: DB
          value: go-demo-2-db
        readinessProbe:
          httpGet:
            path: /demo/hello
            port: 8080
          periodSeconds: 1
        livenessProbe:
          httpGet:
            path: /demo/hello
            port: 8080
```

```
---

apiVersion: v1
kind: Service
metadata:
  name: go-demo-2-api
spec:
  type: NodePort
  ports:
  - port: 8080
  selector:
    type: api
    service: go-demo-2
```

The Differences

If you start searching for differences with the previous definitions, you will find a few.

- The `minReadySeconds`, `progressDeadlineSeconds`, `revisionHistoryLimit`, and `strategy` fields are removed from the `go-demo-2-api` Deployment.
- We used them mostly as a way to demonstrate their usage. But, since Kubernetes has sensible defaults, we omitted them from this definition.
- You'll also notice that there are two Services even though we created only one in this chapter. We did not need the `go-demo-2-api` Service in our examples since we didn't need to access the API. But, for the sake of completeness, it is included in this definition.
- Finally, the strategy for deploying the database is set to `recreate`. As explained earlier, it is more suited for a single-replica database, even though we did not mount a volume that would preserve the data.

Creating Objects with the Merged File

Let's create the objects defined in `deploy/go-demo-2.yml`. Remember, with `--save-config` we're making sure we can edit the configuration later. The alternative would be to use `kubectl apply` instead.

```
kubectl create \
    -f deploy/go-demo-2.yml \
    --record --save-config

kubectl get -f deploy/go-demo-2.yml
```

The **output** of the latter command is as follows.

```
NAME                DESIRED UP-TO-DATE AVAILABLE AGE
deploy/go-demo-2-db 1       1          1         15s

NAME              TYPE      CLUSTER-IP EXTERNAL-IP PORT(S)   AGE
svc/go-demo-2-db ClusterIP 10.0.0.125 <none>      27017/TCP 15s

NAME                  DESIRED  UP-TO-DATE AVAILABLE AGE
deploy/go-demo-2-api 3        3          3         15s

NAME               TYPE     CLUSTER-IP EXTERNAL-IP PORT(S)        AGE
svc/go-demo-2-api NodePort 10.0.0.57  <none>      8080:31586/TCP 15s
```

All four objects (two Deployments and two Services) were created successfully.

In the next lesson, we can move on and explore ways to update multiple objects with a single command.