SCC Project 2 : IAAS Tukano

Jonas Dimitrow 71867
Cornelius Wiehl 72009

Authentication

- rest endpoint ‘/rest/login’ that returns a login page with a form that asks username and password
  - submitting the form sends a post request to the application, that validates the input and then creates a cookie if successful
  - the cookie can then be used to access the blob storage
- for a successful login the user must be present in the database and the passwords need to match

Leverage our Tukano Application with Kubernetes

- In order to improve our application we dockerized the tomcat application in order to make the application usable by kubernetes pods.
- Our deployment.yaml manages the application lifecycle and covers scaling, updates, and rollbacks but also ensures a desired state of the application (maintaining a specified number of replicas)
- The service.yaml exposes the application by providing network access to Pods. It also enables load balancing and stable endpoints for communication.
- With this approach our application becomes easier and automatically scalable which is a key challenge in cloud computing
- It also makes the application more resilient and portable.
- Because the pod needs to communicate outside of the cluster we decided to use NodePort to expose the service

Deployment and Service

```yaml
apiVersion: v1
kind: Service
metadata:
  name: tukano-service
spec:
  selector:
    app: tukano
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
  type: NodePort
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: tukano-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: tukano
  template:
    metadata:
      labels:
        app: tukano
    spec:
      containers:
        - name: tukano-container
          image: yeeman/tukano-app:v1
          imagePullPolicy: Always
          ports:
            - containerPort: 8080
          volumeMounts:
            - name: tukano-storage
              mountPath: /mnt/storage
      volumes:
        - name: tukano-storage
          persistentVolumeClaim:
            claimName: tukano-pvc
```

Persistent Volume

- In order to offer **more flexibility** and **better performance** we replaced Azure Blob Storage with Local File Storage that uses a persistent volume for data handling.
- By using HostPath Volumes the data is stored on the server where Kubernetes is deployed.
- This offers better latency and therefore significant improvements in speed.
- Also we do not have additional cloud storage costs.

Persistent Volume

- Managing the volumes in Kubernetes is straightforward with yaml file
- In order to have more flexibility we used a persistent-volume yaml and a persistent-volume-claim yaml for this task allowing for finer access control
- This also enables dynamic binding, letting Kubernetes match the request to available storage automatically, supporting different storage backends.

Database

- We replaced the existing SQL database with a postgres running in a pod.
- This provides more flexibility for scaling as we can adjust the settings easily in the yaml files.
- It also uses an open-source framework and may prevent vendor lock-in.
- In order to persist the data we also used a Volume Mount.
- Since the database communicates within the cluster we used ClusterIP in the service configuration

Cache

- We also deployed the RedisCache as a separate pod.
- Again ClusterIP was used in the service configuration.
- Again this provides several benefits. Among them are scalability, flexibility, high availability and cost efficiency but also portability.