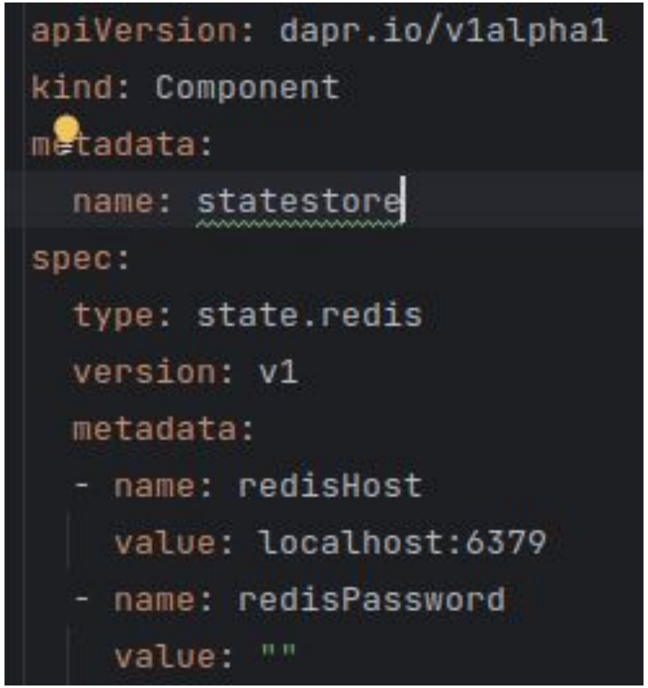
Simple distributed file storage with flask and dapr

Riccardo Benedetti
Leon Harz

Use Case and libraries

- Building an distributed data storage with dapr state management
- Utilizing a redis database provided by dapr
- For client Server communication we chose REST (Flask)
- Dapr and redis run as docker containers
- First we adapted the dapr app, in order to save and retrieve files
- Instead of storing the files directly we moved the storing part to the Dapr which saves the files in a the redis db

Dapr Set up

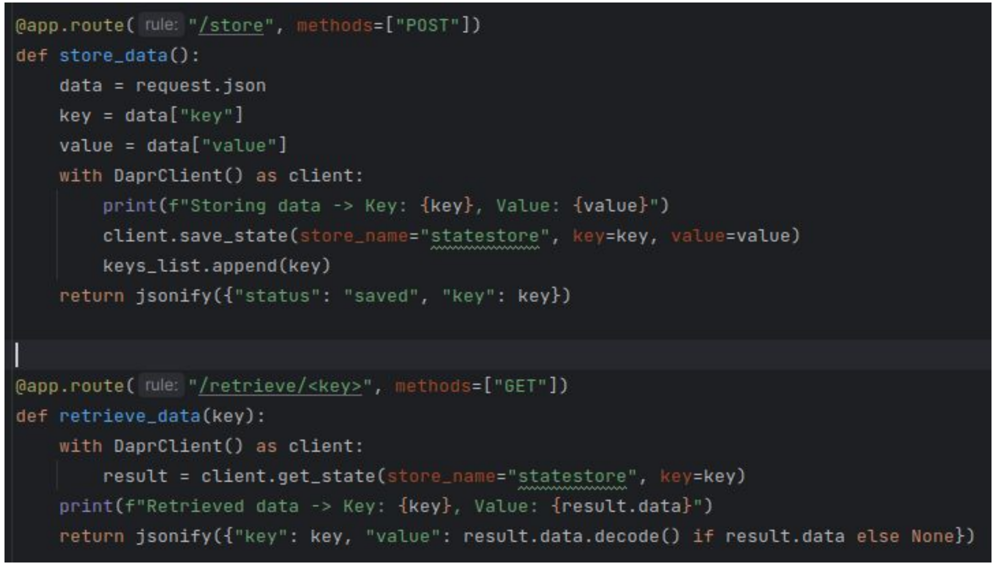


```yaml
apiVersion: dapr.io/v1alpha1
kind: Component
metadata:
  name: statestore
spec:
  type: state.redis
  version: v1
  metadata:
  - name: redisHost
    value: localhost:6379
  - name: redisPassword
    value: ""
```

- specifies api version and kind
- metadata_name allows the application to talk to this component
- spec defines the connection to the redis instance used by the component

Building a distributed data storage as a stateful service

Snippets from server

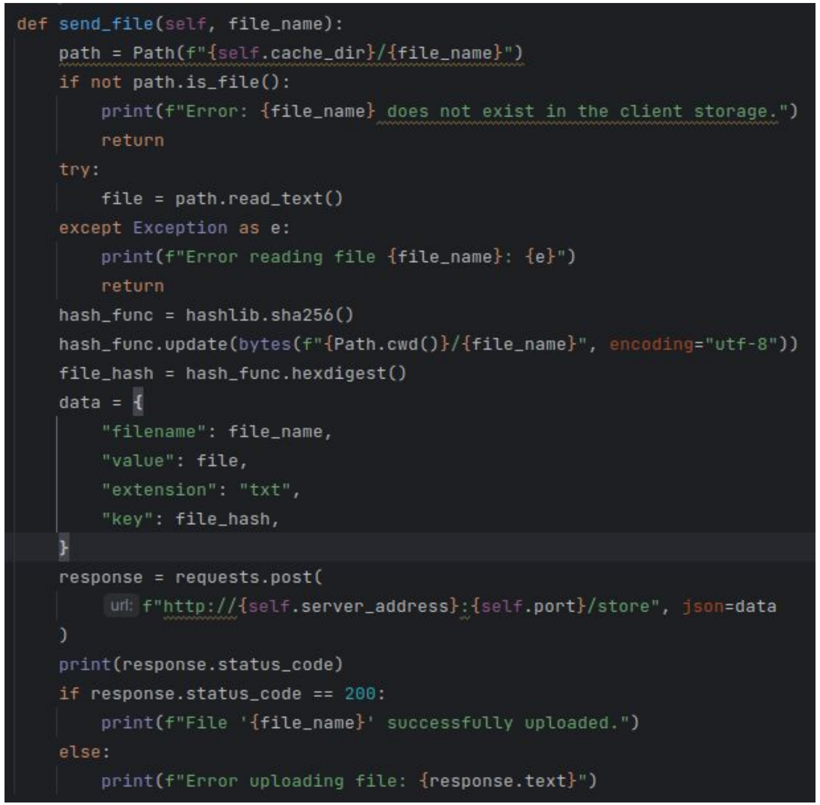


```
@app.route( rule: "/store", methods=["POST"])
def store_data():
    data = request.json
    key = data["key"]
    value = data["value"]
    with DaprClient() as client:
        print(f"Storing data -> Key: {key}, Value: {value}")
        client.save_state(store_name="statestore", key=key, value=value)
        keys_list.append(key)
    return jsonify({"status": "saved", "key": key})


@app.route( rule: "/retrieve/<key>", methods=["GET"])
def retrieve_data(key):
    with DaprClient() as client:
        result = client.get_state(store_name="statestore", key=key)
    print(f"Retrieved data -> Key: {key}, Value: {result.data}")
    return jsonify({"key": key, "value": result.data.decode() if result.data else None})
```

- store_data, is called by client using http protocol
- the request object contains key(URI) and the data of the file
- Utilizing the Dapr client these values are stored in a redis database
- retrieve_data loads the data given a key from the db and sends it to the client via http

Client Implementation



```python
def send_file(self, file_name):
    path = Path(f"{self.cache_dir}/{file_name}")
    if not path.is_file():
        print(f"Error: {file_name} does not exist in the client storage.")
        return
    try:
        file = path.read_text()
    except Exception as e:
        print(f"Error reading file {file_name}: {e}")
        return
    hash_func = hashlib.sha256()
    hash_func.update(bytes(f"{Path.cwd()}/{file_name}", encoding="utf-8"))
    file_hash = hash_func.hexdigest()
    data = {
        "filename": file_name,
        "value": file,
        "extension": "txt",
        "key": file_hash,
    }
    response = requests.post(
        url: f"http://{self.server_address}:{self.port}/store", json=data
    )
    print(response.status_code)
    if response.status_code == 200:
        print(f"File '{file_name}' successfully uploaded.")
    else:
        print(f"Error uploading file: {response.text}")
```

- send_file creates and URI from filename and path
- Builds a request object containing URI, file content etc.
- Use http to make the request
- receive_file queries the URI to the server
- If server has an db entry with this query, the client receives the file

Strengths and limitations

- Dapr allows to quickly change the database system
- Application/Code is not directly connected to underlying systems
- This allows easier maintenance and better scalability
- Using kubernets makes the architecture easily scalabe