UDACITY

PROJECT SPECIFICATION

UdaConnect

Architecture Design

CRITERIA	MEETS SPECIFICATIONS
Use the UdaConnect business requirements to determine which communication protocol and technologies to use for the project	Each module includes a 1-2 sentence justification for the module design choice either as a label on the design diagram or in a separate document The justifications indicate that the decisions about the chosen protocols and technologies are based on business requirements including required scale. Your supervisor wants to be able to launch this project in 2 weeks with a limited budget. At the same time, the project needs to be able to scale to handle large volumes of location data being ingested. The project is designed as an MVP and does not include any unnecessary features • Cost and development time are minimized • Services should run in containers • Design should be able to handle ingress of a large volume of data
Create an architecture diagram for the UdaConnect Project that accurately displays the relationship between the modules	Architecture diagram shows the design of the system as individual services. It should show the relationship between the frontend, API's, databases, and Kafka. Arrows or lines connect the individual services to represent request/response relationships between services. All of the necessary services and protocols are included as modules in the architectural design

Microservice Development

CRITERIA	MEETS SPECIFICATIONS
Deploy distinct services as microservices	• Services have their own `Dockerfile`'s • Services are deployable with Kubernetes through Kubernetes `Deployment`'s and `Service`'s.
Refactor existing code to microservices	• Kubernetes deployments use Docker images built from the students' final solution and not from the starter code.

Message Passing

CRITERIA	MEETS SPECIFICATIONS
Apply message passing techniques to enable communication between services using Kafka	• Project implements a Kafka queue in a container in a `Dockerfile` or Kubernetes deployment file. This can be configured manually in a base image or using a pre-built Kafka Docker image. • `requirements.txt` file installs the `kafka-python` Kafka library. • `kafka` deployment runs successfully without errors.
Apply message passing techniques to enable communication between services using gRPC	• Project contains a `*.proto` file showing that they mapped a message and service into a protobuf format. The `*.proto` file should have at least one `message` and one `service` declared in it. • Code should contain a `*_pb2` and `*_pb2_grpc` file generated from the `*.proto` file. • Project contains a gRPC client. If using Python with the standard `grpc` library, code should open a gPRC channel. • Project contains a gPRC host. The standard `gprc` library code should contain a `grpc.server()` instantiation. • `requirements.txt` file should define the `grpcio` package.

CRITERIA	MEETS SPECIFICATIONS
Apply message passing techniques to enable communication between services using REST API's	• Project either created a new API endpoint(s) or a modification to existing Flask API. • All new API endpoints use proper HTTP request types. • `GET` or `DELETE` request does not contain an HTTP payload • REST API's have live Swagger documentation in an external library (or manually written an OpenAPI spec.

Code Quality

CRITERIA	MEETS SPECIFICATIONS
Write effective code	Project runs without error Code is appropriately documented. Comments are concise, relevant and not excessive Code is neatly written with proper indenting. Python code follows PEP 8 guidelines.

Suggestions to Make Your Project Stand Out!

1. Implement an additional message passing strategy.
2. Extend the behavior of the current API by adding another endpoint.
3. Improve frontend application by making it look more attractive or implementing other backend API's.
4. Deploy the application to a hosted environment.