

AUTOLEARN

Rosario Scalia

Agenda

- Definizione Progetto
- Business-Logic
- Persistenza
- Presentazione
- Requisiti Progettuali
- Design Pattern
- Stack Software
- Bozza Architettura

Definizione Progetto

- Applicativo Distribuito che permette la gestione del ciclo di vita di un Modello di Machine Learning
- **Fasi del Ciclo di Vita Gestite:** Training ed Evaluation
- **Modelli a Catalogo:**
 - 1) Logistic Regressor
 - 2) Naive-Bayes Classifier
 - 3) ...
- Progetto simile nella filosofia ad Azure Custom Vision

Business-Logic

- Visualizza Catalogo Modelli e Dataset
- Training di un Modello di ML
 - 1) Selezione Dataset
 - 2) Selezione Modello
- Calcolo Misure di Valutazione su uno Specifico Modello già trainato
- Salvataggio Training nello Storage *Permanente*
- Visualizza Info Training (training della sessione attuale)
- Visualizza Info Esperimenti Passati

Persistenza

- I dati degli Esperimenti verranno archiviati in un apposito DB
- **Dati Esperimenti:**
 - 1) Timestamp Commit Sessione
 - 2) Dati di Training (parametri appresi, settaggi di training scelti...)
 - 3) Dati di Evaluation (prestazioni modello secondo una serie di misure di valutazione)
- I dati di Sessione verranno archiviati in un apposito DB
- **Dati Sessione:**
 - 1) Timestamp Creazione Sessione
 - 2) Dati di Training (parametri appresi, settaggi di training scelti...)
 - 3) Dati di Evaluation (prestazioni modello secondo una serie di misure di valutazione)

Presentazione

- Command Line Interface
- Client comunica col Server attraverso l'invocazione di una serie di Procedure Remote Messe a disposizione da quest'ultimo
- **Tecnologia Comunicazione: REST**
- **Modalità Invocazione API-REST:** Sincrona e Asincrona

Requisiti Progettuali – Machine Learning

→ I dataset devono risiedere sul Backend

→ **Modelli di ML disponibili a Catalogo:**

- 1) Regressore Logistico
- 2) Decision Tree
- 3) Random Forest
- 4) SVM
- 5) Naive Bayes

→ **Misure di Valutazione:**

- 1) Precision
- 2) Recall

Requisiti Progettuali - Architettura

- Il Backend sarà a Microservizi
- Ogni Microservizio espone un REST End-Point
- Le comunicazioni Sincrone avverranno attraverso chiamate ai REST End-Point
- Le comunicazioni Asincrone avverranno attraverso Message Broker
- La Logica associata ad ogni chiamata REST verrà gestita da un apposito Processo, generato al momento della chiamata
- Ogni Microservizio sarà costituito da 3 *Tipologie* di Processi Differenti:
 - **REST-WORKER**, processo che resta in ascolto per chiamate al REST End-Point
 - **EVENT-WORKER**, processo che resta in ascolto ,asincronicamente, in merito a nuovi eventi associati ad un task specifico (*ES. Comunicazione Asincrona fra Microservizi*)
 - **TASK-WORKER**, processo che prende in carico il task associato alla chiamata di un'API REST
- I Microservizi possono avere più istanze parallele in esecuzione, dato che il Web Server sfrutta il *parallelismo* offerto dalle moderne CPU Multi-Core

Requisiti Progettuali – Gestione Sessione

- Il sistema conserva dei dati di sessione che il Client può decidere di confermare in futuro
- La conferma di una Sessione implica la sua scrittura nello Storage permanente e la sua successiva rimozione
- I dati di sessione verranno mantenuti su un apposito Database
 - **DATABASE SESSION STATE**

Requisiti Progettuali - Sicurezza

- Client e Server utilizzano uno schema di Crittografia Simmetrico per scambiarsi Informazioni Confidenziali
- La chiave crittografica sarà una chiave a 128 bit generata dall'Algoritmo AES
- Le informazioni Confidenziali delle comunicazioni sono i Dati di Sessione

Requisiti Progettuali – Monitoraggio

- Lo stato passato dell'applicazione deve essere *ricostruibile*
- Ogni Comunicazione fra Microservizi deve essere *segnala ed archiviata*
- Ogni Comunicazione Client -----> Backend deve essere *segnala ed archiviata*
- **Tipologia Archiviazione:**
 - 1) Memoria
 - 2) Event Store

Requisiti Progettuali – Modularità

- *A progetto finito*, deve essere “semplice” ampliare il catalogo attuale di Modelli
- *A progetto finito*, deve essere “semplice” ampliare il catalogo attuale delle Misure di Valutazione

Design Pattern

→ **Comunicazione Distribuita**

- 1) Remote Proxy
- 2) Forward-Receiver
- 3) Remote Facade
- 4) Data Transfer Object

→ **Gestione Sessione**

- 1) Session State

→ **Gestione Monitoraggio**

- 1) Event Sourcing

Stack Software

Service	Component
SERVER	<u>Python</u>
CLIENT	<u>Python</u>
MULTI-PROCESSING	<u>multiprocessing</u>
THREADING	<u>threading</u>
ASYNC-PROCESSING	<u>asyncio</u> <u>concurrent.futures</u>
IPC	<u>multiprocessing.Queue</u>
ITCC	<u>janus</u>
ICC	<u>asyncio.Queue</u>
HTTP SYNC REQ	<u>requests</u>
HTTP ASYNC REQ	<u>aiohttp</u>
SERIALIZATION ENG	<u>json + pickle</u>
CRYPTOGRAPHIC LIB	<u>cryptographic.fernet</u>
MACHINE LEARNING	<u>scikit-learn</u>
SOFTWARE VERSIONING	<u>git</u>

Stack Software

Service	Component
SERVER	<u>Python</u>
CLIENT	<u>Python</u>
MULTI-PROCESSING	<u>multiprocessing</u>
THREADING	<u>threading</u>
ASYNC-PROCESSING	<u>asyncio</u> <u>concurrent.futures</u>
IPC	<u>multiprocessing.Queue</u>
ITCC	<u>janus</u>
ICC	<u>asyncio.Queue</u>
HTTP SYNC REQ	<u>requests</u>
HTTP ASYNC REQ	<u>aiohttp</u>
SERIALIZATION ENG	<u>json + pickle</u>
CRYPTOGRAPHIC LIB	<u>cryptographic.fernet</u>
MACHINE LEARNING	<u>scikit-learn</u>
SOFTWARE VERSIONING	<u>git</u>

Service	Component
REST END-POINT	<u>fastAPI</u>
WEB SERVER	<u>uvicorn</u>
MESSAGE BROKER	<u>aio-pika</u> { rabbitMQ }
EVENT STORE	<u>aiokafka</u>
DATA STORE	<u>motor</u> { mongoDB }
TEST	<u>pytest</u> <u>unittest</u>
DOC GEN	<u>fastAPI</u> { OpenAPI } <u>sphinx</u> <u>pdoc3</u>
DEPLOY ENV	<u>docker</u>

Bozza Architettura

CLIENT

CATALOG
{REST API}

TRAINING
{REST API}

EVALUATION
{REST API}

SESSION
{REST API}

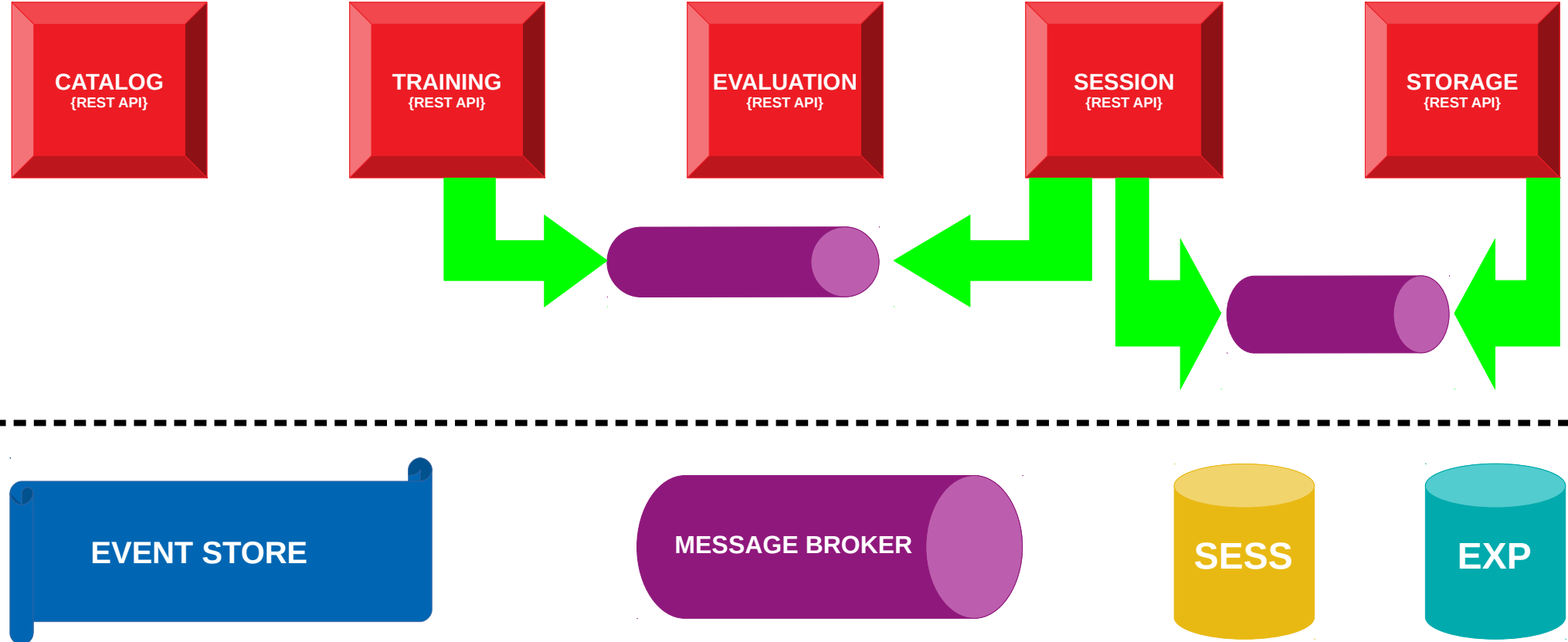
STORAGE
{REST API}

EVENT STORE

MESSAGE BROKER

SESS

EXP



FINE

Rosario Scalia