CampusWell — Paso a paso (backend + ingesta + despliegue)

Guía práctica para levantar los **5 microservicios**, datos masivos, **ingesta a S3 + Glue + Athena**, **Swagger**, **docker-compose**, **balanceo con Nginx** en 2 VMs y **Frontend (Amplify)** que consume 2+ métodos por MS.

---

0) Prerrequisitos locales (o en una EC2 de laboratorio)

- **Docker** y **Docker Compose v2**
- **Java 21** + Maven (para Spring Boot)
- **Node.js 20** + **Nest CLI** (`npm i -g @nestjs/cli`)
- **Python 3.11+** (recomendado 3.12) + pip
- **AWS CLI v2** configurado (para S3/Athena)

---

1) Estructura de repositorio (monorepo sugerido)

```
campuswell/
  .env
  docker-compose.yml
  nginx/
    nginx.conf
  psych-svc/            # Spring Boot + Postgres
  sports-svc/           # FastAPI + MySQL
  habits-svc/           # NestJS + MongoDB
  aggregator-svc/       # FastAPI (sin BD)
  analytics-svc/        # FastAPI + Athena
  ingest/
    ingest-psych/       # Python → S3 (Postgres)
    ingest-sports/      # Python → S3 (MySQL)
    ingest-habits/      # Python → S3 (Mongo)
  seed/
    seed-psych.sql
    seed-sports.sql
    seed-habits.js
  frontend/             # (opcional) app React para Amplify
```

1.1) `.env` (ejemplo)

```
# Postgres (psych)
POSTGRES_DB=campuswell
POSTGRES_USER=campus
POSTGRES_PASSWORD=campus
```

```
# MySQL (sports)
MYSQL_DATABASE=campuswell
MYSQL_USER=campus
MYSQL_PASSWORD=campus
MYSQL_ROOT_PASSWORD=root

# Mongo (habits)
MONGO_URL=mongodb://mongo:27017/campuswell

# JWT
JWT_SECRET=supersecret

# Athena/Glue
AWS_REGION=us-east-1
ATHENA_DB=campuswell
ATHENA_OUTPUT=s3://tu-bucket/athena-output/
S3_RAW_BUCKET=s3://tu-bucket/raw/
```

---

2) Microservicio 1 — psych-svc (Spring Boot + PostgreSQL)

Objetivo: estudiantes y citas psicológicas.

2.1) `pom.xml` (mínimo)

```
<project ...>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.campuswell</groupId>
  <artifactId>psych-svc</artifactId>
  <version>0.0.1</version>
  <properties>
    <java.version>21</java.version>
    <spring.boot.version>3.3.2</spring.boot.version>
  </properties>
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-dependencies</artifactId>
        <version>${spring.boot.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
```

```
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
      <groupId>org.postgresql</groupId>
      <artifactId>postgresql</artifactId>
      <scope>runtime</scope>
    </dependency>
    <dependency>
      <groupId>org.springdoc</groupId>
      <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
      <version>2.5.0</version>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
</project>
```

2.2) `application.yml`

```
server:
  port: 8081
spring:
  datasource:
    url: jdbc:postgresql://postgres:5432/${POSTGRES_DB}
    username: ${POSTGRES_USER}
    password: ${POSTGRES_PASSWORD}
  jpa:
    hibernate:
      ddl-auto: update
    properties:
      hibernate.dialect: org.hibernate.dialect.PostgreSQLDialect
```

2.3) Entidades

```
// Student.java
@Entity
public class Student {
  @Id @GeneratedValue private Long id;
  private String name; private String email; private String career; private
```

```
String cohort;
  // getters/setters
}

// Appointment.java
@Entity
public class Appointment {
  @Id @GeneratedValue private Long id;
  private Long studentId; private String psychologist; private
java.time.OffsetDateTime date;
  @Enumerated(EnumType.STRING) private Status status; // PENDING, CONFIRMED,
CANCELLED
  public enum Status { PENDING, CONFIRMED, CANCELLED }
  // getters/setters
}
```

2.4) Repos y Controller (ejemplo rápido)

```
@RestController
@RequestMapping("/api")
public class PsychController {
  private final StudentRepo students; private final AppointmentRepo appts;
  public PsychController(StudentRepo s, AppointmentRepo a){ this.students=s;
this.appts=a; }

  @PostMapping("/students") public Student create(@RequestBody Student s){
return students.save(s);}
  @GetMapping("/students/{id}") public Student one(@PathVariable Long id)
{return students.findById(id).orElseThrow();}
  @GetMapping("/students/{id}/history") public List<Appointment>
hist(@PathVariable Long id){return appts.findByStudentId(id);}
  @PostMapping("/appointments") public Appointment ap(@RequestBody
Appointment a){return appts.save(a);}
  @GetMapping("/health") public Map<String,String> health(){return
Map.of("status","ok");}
}
```

2.5) Dockerfile

```
FROM eclipse-temurin:21-jre
WORKDIR /app
COPY target/psych-svc-0.0.1.jar app.jar
EXPOSE 8081
ENTRYPOINT ["java","-jar","/app/app.jar"]
```

---

3) Microservicio 2 — sports-svc (FastAPI + MySQL)

Objetivo: eventos/inscripciones deportivas y culturales.

3.1) `requirements.txt`

```
fastapi
uvicorn[standard]
sqlalchemy
pymysql
python-multipart
```

3.2) `main.py` (mínimo funcional)

```
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from sqlalchemy import create_engine, text
import os

app = FastAPI(title="sports-svc")
DB_URL=f"mysql+pymysql://{os.getenv('MYSQL_USER')}:
{os.getenv('MYSQL_PASSWORD')}@mysql:3306/{os.getenv('MYSQL_DATABASE')}"
engine = create_engine(DB_URL, pool_pre_ping=True)

class EventIn(BaseModel):
    name:str; type:str; date:str; location:str

@app.get("/events")
def list_events(type: str | None = None):
    q = "SELECT id,name,type,date,location FROM events" + (" WHERE type=:t"
if type else "")
    with engine.begin() as cn: rows = cn.execute(text(q), {"t": type} if type
else {}).mappings().all()
    return list(rows)

@app.post("/registrations")
def add_registration(student_id:int, event_id:int):
    with engine.begin() as cn:
        cn.execute(text("INSERT INTO registrations(student_id,event_id)
VALUES (:s,:e)"),{"s":student_id,"e":event_id})
    return {"ok": True}

@app.get("/health")
def health(): return {"status":"ok"}
```

3.3) Dockerfile

```
FROM python:3.12-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
EXPOSE 8082
CMD ["uvicorn","main:app","--host","0.0.0.0","--port","8082"]
```

---

4) Microservicio 3 — habits-svc (NestJS + MongoDB)

Objetivo: hábitos (sueño, ejercicio, ánimo) por estudiante.

4.1) Instalación inicial

```
cd habits-svc
nest new habits-svc --package-manager npm
npm i @nestjs/mongoose mongoose @nestjs/config
```

4.2) Esquema/Controller (resumen)

```
// habits.schema.ts
import { Schema } from 'mongoose';
export const HabitSchema = new Schema({
  studentId: Number,
  sleepHours: Number,
  exerciseMinutes: Number,
  mood: Number,
  date: { type: Date, default: Date.now }
});

// habits.controller.ts
@Get('habits/:studentId')
async list(@Param('studentId') id:number){ return
this.habitsModel.find({studentId: id}).limit(200); }
@Post('habits')
async create(@Body() dto:any){ return new this.habitsModel(dto).save(); }
@Get('health') getHealth(){ return {status:'ok'} }
```

4.3) `main.ts` (CORS y puerto)

```
const app = await NestFactory.create(AppModule, { cors: true });
await app.listen(8083, '0.0.0.0');
```

4.4) Dockerfile

```
FROM node:20-alpine
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build
EXPOSE 8083
CMD ["node","dist/main.js"]
```

---

5) Microservicio 4 — aggregator-svc (FastAPI, sin BD)

Objetivo: orquestar consultas y recomendaciones.

5.1) `requirements.txt`

```
fastapi
uvicorn[standard]
httpx
python-jose[cryptography]
```

5.2) `main.py` (llamadas con timeout/retry)

```
from fastapi import FastAPI, Depends
import httpx, os

app = FastAPI(title="aggregator-svc")
PSY=os.getenv('PSYCH_BASE','http://psych-svc:8081')
SPO=os.getenv('SPORTS_BASE','http://sports-svc:8082')
HAB=os.getenv('HABITS_BASE','http://habits-svc:8083')
client = httpx.Client(timeout=5.0)

@app.get('/wellbeing/{student_id}/overview')
def overview(student_id:int):
    s = client.get(f"{PSY}/api/students/{student_id}").json()
    h = client.get(f"{PSY}/api/students/{student_id}/history").json()
    r = client.get(f"{HAB}/habits/{student_id}").json()
    return {"student": s, "appointments": h, "habits": r}

@app.post('/wellbeing/recommendation')
def rec(student_id:int):
    habits = client.get(f"{HAB}/habits/{student_id}").json()
    avg_mood = sum([x.get('mood',0) for x in habits])/max(1,len(habits))
    events = client.get(f"{SPO}/events?type=sport").json()
    suggestion = events[0] if events else None
```

```
    return {"avg_mood": avg_mood, "suggested_event": suggestion}

@app.get('/health')
def health(): return {"status":"ok"}
```

5.3) Dockerfile (igual que sports, puerto 8080)

```
FROM python:3.12-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
EXPOSE 8080
CMD ["uvicorn","main:app","--host","0.0.0.0","--port","8080"]
```

---

6) Microservicio 5 — analytics-svc (FastAPI + Athena)

Objetivo: exponer KPIs vía Athena (datos en S3).

6.1) `requirements.txt`

```
fastapi
uvicorn[standard]
boto3
```

6.2) `main.py` (Athena client)

```
from fastapi import FastAPI
import boto3, os, time

app = FastAPI(title='analytics-svc')
athena = boto3.client('athena', region_name=os.getenv('AWS_REGION'))
DB=os.getenv('ATHENA_DB'); OUT=os.getenv('ATHENA_OUTPUT')

SQL_STRESS = """
SELECT date_trunc('week', date) wk, count(*) confirmed
FROM psych_appointments
WHERE status='CONFIRMED'
GROUP BY 1 ORDER BY 1
"""

@app.get('/analytics/stress-trends')
def stress():
    q = athena.start_query_execution(QueryString=SQL_STRESS,
QueryExecutionContext={'Database':DB},
ResultConfiguration={'OutputLocation':OUT})
```

```
    qid = q['QueryExecutionId']
    while True:
        s=athena.get_query_execution(QueryExecutionId=qid)['QueryExecution']
['Status']['State']
        if s in ('SUCCEEDED','FAILED','CANCELLED'): break
        time.sleep(1)
    if s!='SUCCEEDED': return {'error': s}
    res=athena.get_query_results(QueryExecutionId=qid)
    rows = [[c.get('VarCharValue') for c in r['Data']] for r in
res['ResultSet']['Rows']][1:]
    return {'rows': rows}

@app.get('/health')
def health(): return {"status":"ok"}
```

6.3) Dockerfile

```
FROM python:3.12-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
EXPOSE 8084
CMD ["uvicorn","main:app","--host","0.0.0.0","--port","8084"]
```

---

7) Bases de datos y seeds

7.1) Esquemas iniciales (MySQL y Postgres)

MySQL (sports)

```
CREATE TABLE IF NOT EXISTS events (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(120), type VARCHAR(40), date DATETIME, location VARCHAR(120)
);
CREATE TABLE IF NOT EXISTS registrations (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  student_id BIGINT, event_id BIGINT,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

Postgres (psych) – gestionado por JPA (`ddl-auto=update`).

7.2) Seed de ejemplo

Postgres `seed-psych.sql`

```
INSERT INTO student (name,email,career,cohort) VALUES
 ('Ana','ana@u.edu','CS','2022'),('Luis','luis@u.edu','EE','2021');
```

MySQL `seed-sports.sql`

```
INSERT INTO events(name,type,date,location) VALUES
 ('Yoga en el parque','sport','2025-09-20 09:00:00','Parque A'),
 ('Taller de Teatro','culture','2025-09-22 16:00:00','Aula 3');
```

Mongo `seed-habits.js`

```
db.habits.insertMany([
 {studentId:1, sleepHours:7.5, exerciseMinutes:20, mood:3, date:new Date()},
 {studentId:1, sleepHours:6.0, exerciseMinutes:0, mood:2, date:new Date()},
 {studentId:2, sleepHours:8.0, exerciseMinutes:30, mood:4, date:new Date()}
])
```

---

8) `docker-compose.yml` (multi-servicio + BDs)

```
version: "3.9"
services:
  postgres:
    image: postgres:16
    environment:
      POSTGRES_DB: ${POSTGRES_DB}
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
    ports: ["5432:5432"]
    healthcheck: { test: ["CMD-SHELL","pg_isready -U $$POSTGRES_USER"],
interval: 10s, timeout: 5s, retries: 5 }

  mysql:
    image: mysql:8
    environment:
      MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
      MYSQL_DATABASE: ${MYSQL_DATABASE}
      MYSQL_USER: ${MYSQL_USER}
      MYSQL_PASSWORD: ${MYSQL_PASSWORD}
    command: ["--default-authentication-plugin=mysql_native_password"]
    ports: ["3306:3306"]
    healthcheck: { test: ["CMD","mysqladmin","ping","-h","localhost"],
interval: 10s, timeout: 5s, retries: 5 }

  mongo:
    image: mongo:7
    ports: ["27017:27017"]
```

```
  psych-svc:
    build: ./psych-svc
    environment: [POSTGRES_DB=${POSTGRES_DB}, POSTGRES_USER=$
{POSTGRES_USER}, POSTGRES_PASSWORD=${POSTGRES_PASSWORD}]
    depends_on: [postgres]
    ports: ["8081:8081"]

  sports-svc:
    build: ./sports-svc
    environment: [MYSQL_DATABASE=${MYSQL_DATABASE}, MYSQL_USER=$
{MYSQL_USER}, MYSQL_PASSWORD=${MYSQL_PASSWORD}]
    depends_on: [mysql]
    ports: ["8082:8082"]

  habits-svc:
    build: ./habits-svc
    environment: [MONGO_URL=${MONGO_URL}]
    depends_on: [mongo]
    ports: ["8083:8083"]

  aggregator-svc:
    build: ./aggregator-svc
    environment:
      - PSYCH_BASE=http://psych-svc:8081
      - SPORTS_BASE=http://sports-svc:8082
      - HABITS_BASE=http://habits-svc:8083
      - JWT_SECRET=${JWT_SECRET}
    depends_on: [psych-svc, sports-svc, habits-svc]
    ports: ["8080:8080"]

  analytics-svc:
    build: ./analytics-svc
    environment: [AWS_REGION=${AWS_REGION}, ATHENA_DB=${ATHENA_DB},
ATHENA_OUTPUT=${ATHENA_OUTPUT}]
    ports: ["8084:8084"]
```

Arranque local:

```
docker compose build
# Cargar seeds (opcional, después de que BD esté ready)
docker exec -i $(docker ps -qf name=mysql) mysql -u${MYSQL_USER} -p$
{MYSQL_PASSWORD} ${MYSQL_DATABASE} < seed/seed-sports.sql
psql postgresql://$POSTGRES_USER:$POSTGRES_PASSWORD@localhost:
5432/$POSTGRES_DB -f seed/seed-psych.sql
mongo --host localhost:27017 --eval "load('seed/seed-habits.js')"

docker compose up -d
```

---

9) Balanceador Nginx (2 VMs)

Asumiendo **VM1** y **VM2** corren `docker compose` con los 5 MS. El **Nginx (LB)** expone público el **aggregator** y opcionalmente `analytics`.

9.1) `nginx/nginx.conf`

```
worker_processes auto;
events { worker_connections 1024; }
http {
  upstream aggregator_pool {
    server vm1.private:8080;
    server vm2.private:8080;
  }
  upstream analytics_pool {
    server vm1.private:8084;
    server vm2.private:8084;
  }
  server {
    listen 80;
    location /analytics/ { proxy_pass http://analytics_pool/; }
    location / { proxy_pass http://aggregator_pool/; proxy_set_header Host
$host; proxy_set_header X-Forwarded-For $remote_addr; }
  }
}
```

Comandos en el LB

```
sudo apt-get update && sudo apt-get install -y nginx
sudo cp nginx/nginx.conf /etc/nginx/nginx.conf
sudo nginx -t && sudo systemctl restart nginx
```

---

10) Ingesta → S3 + Glue + Athena

10.1) Contenedores de ingesta (pull total)

ingest-psych/main.py

```
import os, csv, boto3, psycopg
S3=os.getenv('S3_RAW_BUCKET')
conn=psycopg.connect(f"dbname={os.getenv('POSTGRES_DB')}
user={os.getenv('POSTGRES_USER')} password={os.getenv('POSTGRES_PASSWORD')}
host=postgres")
cur=conn.cursor();
cur.execute("SELECT id,student_id,psychologist,date,status FROM appointment")
with open('/tmp/appointments.csv','w',newline='') as f:
```

```
  w=csv.writer(f);
w.writerow(['id','student_id','psychologist','date','status']);
w.writerows(cur.fetchall())
boto3.client('s3').upload_file('/tmp/appointments.csv', S3.split('://')
[1].split('/')[0], 'raw/psych/appointments/appointments.csv')
```

(Análogo para MySQL y Mongo: export a CSV/JSONL y subir a `raw/sports` *y* `raw/habits` *)*

10.2) Glue + Athena

1. Crear **DB Glue** `campuswell` y **crawlers** para `s3://.../raw/psych`, `.../raw/sports`, `.../raw/habits`.
2. Ejecutar crawlers → tablas en catálogo.
3. En **Athena**, set `Query results` a `ATHENA_OUTPUT`.
4. Consultas de ejemplo (crear vistas si la rúbrica lo pide):

```
-- Vista 1: citas confirmadas por semana
CREATE OR REPLACE VIEW vw_stress AS
SELECT date_trunc('week', date) wk, count(*) confirmed
FROM psych_appointments WHERE status='CONFIRMED'
GROUP BY 1;

-- Vista 2: promedio ánimo vs ejercicio
CREATE OR REPLACE VIEW vw_activity_mood AS
SELECT studentId, avg(exerciseMinutes) ex_avg, avg(mood) mood_avg
FROM habits GROUP BY studentId HAVING count(*)>10;
```

---

11) Swagger y Health

- **psych-svc**: `/swagger-ui.html` y `/api/*`
- **sports-svc**: `/docs`
- **habits-svc**: configurar SwaggerModule si deseas (`/api`)
- **aggregator-svc**: `/docs`
- **analytics-svc**: `/docs`
- Todos: `/health`

---

12) Frontend (Amplify)

- App React con páginas:
- Perfil bienestar (consume `GET /wellbeing/{id}/overview`, `POST /wellbeing/recommendation`).
- Psicología (consume `GET /students/{id}`, `POST /appointments`).
- Deportes/Cultura (consume `GET /events`, `POST /registrations`).
- Hábitos (consume `POST /habits`, `GET /habits/{id}`).
- Analytics (consume `GET /analytics/stress-trends`, `GET /analytics/activity-impact`).

13) Operación: comandos útiles

```
# construir e iniciar todo
docker compose build && docker compose up -d

# logs de un servicio
docker compose logs -f aggregator-svc

# parar y limpiar
docker compose down -v
```

14) Checklist de rúbrica (cumplimiento)

- 5 MS: 3 con BD (Postgres/MySQL/Mongo), 1 orquestador sin BD, 1 analítico con Athena.
- Datos masivos: seeds + Faker (amplía los scripts si necesitas llegar a 20k+).
- Ingesta completa: 3 contenedores → S3; Glue + 4+ queries y 2 vistas.
- Swagger en todos; healthchecks.
- Despliegue con docker-compose en 2 VMs + Nginx LB.
- Front en Amplify consumiendo ≥2 métodos por MS.

Sugerencia: si estás en AWS Academy, corre ingestas y despliegue desde una EC2 con el rol permitido; destruye recursos al final para evitar costos.