# Ghid de Implementare (Deployment) pe Google Cloud

Acest document descrie procesul de a implementa aplicația "Sparky's Workshop Factory" pe infrastructura Google Cloud, folosind arhitectura noastră factorizată.

### **Pasul 0: Pregătirea Mediului Local**

Înainte de a începe, asigurați-vă că aveți următoarele:

1. **Google Cloud SDK (gcloud CLI):** Instalat și configurat pe mașina dumneavoastră.
2. **Un Proiect Google Cloud:** Creat, cu facturarea activată.
3. **API-uri Activate:** Asigurați-vă că API-urile "Cloud Functions", "Cloud Build", "Vertex AI" și "Cloud Storage" sunt activate în proiectul dumneavoastră.

### **Pasul 1: Implementarea Backend-ului (Funcțiile Cloud)**

Vom lua fiecare serviciu de backend și îl vom transforma într-o Funcție Cloud. Să luăm ca exemplu funcția de generare a textului.

**Structura fișierelor pentru funcție:**

/generate-text-function  
 |- main.py  
 |- requirements.txt

**1.1. Codul Funcției (main.py):** Acesta este un exemplu de cod Python care primește o cerere, cheamă API-ul Vertex AI (Gemini) și returnează rezultatul.

import functions\_framework  
from flask import jsonify  
import vertexai  
from vertexai.generative\_models import GenerativeModel  
  
# Inițializare Vertex AI  
vertexai.init(project="YOUR\_PROJECT\_ID", location="us-central1")  
model = GenerativeModel("gemini-1.0-pro")  
  
@functions\_framework.http  
def generate\_text\_handler(request):  
 # Setarea header-elor CORS pentru a permite cereri de la browser  
 headers = {  
 'Access-Control-Allow-Origin': '\*' # Într-o aplicație reală, restricționați la domeniul dvs.  
 }  
  
 if request.method == 'OPTIONS':  
 # Gestionarea cererilor pre-flight CORS  
 headers = {  
 'Access-Control-Allow-Origin': '\*',  
 'Access-Control-Allow-Methods': 'POST',  
 'Access-Control-Allow-Headers': 'Content-Type',  
 'Access-Control-Max-Age': '3600'  
 }  
 return ('', 204, headers)  
  
 # Procesarea cererii POST  
 request\_json = request.get\_json(silent=True)  
 if not request\_json or 'prompt' not in request\_json:  
 return (jsonify({"error": "Prompt lipsă"}), 400, headers)  
  
 prompt = request\_json['prompt']  
   
 try:  
 response = model.generate\_content(  
 f"Creează un text scurt și creativ pentru copii, bazat pe ideea: '{prompt}'"  
 )  
 return (jsonify({"text": response.text}), 200, headers)  
 except Exception as e:  
 return (jsonify({"error": str(e)}), 500, headers)

**1.2. Dependențele (requirements.txt):**

functions-framework==3.\*  
google-cloud-aiplatform>=1.38

**1.3. Comanda de Implementare (din terminal):** Navigați în directorul /generate-text-function și rulați:

gcloud functions deploy generate-text-function \  
--gen2 \  
--runtime=python311 \  
--region=us-central1 \  
--source=. \  
--entry-point=generate\_text\_handler \  
--trigger-http \  
--allow-unauthenticated

După finalizare, veți primi un **URL de trigger**. Salvați acest URL! Acesta este endpoint-ul API pentru funcția dumneavoastră.

**Repetați acest proces pentru celelalte funcții de backend** (generate-image, generate-package), fiecare în propriul său director și cu propriul său cod.

### **Pasul 2: Implementarea Frontend-ului (React App)**

Vom găzdui aplicația noastră React pe Cloud Storage, o soluție eficientă și ieftină pentru site-uri statice.

**2.1. Compilarea Aplicației:** În directorul proiectului React, rulați comanda de build:

npm run build

Aceasta va crea un director build/ (sau dist/) care conține toate fișierele HTML, CSS și JavaScript statice.

**2.2. Crearea unui Bucket în Cloud Storage:** Creați un bucket nou, care va găzdui site-ul. Este o bună practică să-i dați un nume asociat domeniului (ex: www.montessori-workshops.org).

gcloud storage buckets create gs://NUME\_BUCKET\_UNIC --uniform-bucket-level-access

**2.3. Încărcarea Fișierelor:** Încărcați conținutul directorului build/ în bucket:

gcloud storage cp -r build/\* gs://NUME\_BUCKET\_UNIC

**2.4. Setarea Permisiunilor Publice:** Faceți bucket-ul accesibil public pe internet:

gcloud storage buckets add-iam-policy-binding gs://NUME\_BUCKET\_UNIC --member=allUsers --role=roles/storage.objectViewer

**2.5. Setarea Paginii Principale:**

gcloud storage buckets update gs://NUME\_BUCKET\_UNIC --web-main-page-suffix=index.html

Acum, aplicația dumneavoastră React este live și accesibilă la un URL public de forma https://storage.googleapis.com/NUME\_BUCKET\_UNIC/index.html.

### **Pasul 3: Conectarea Finală (Configurarea Frontend-ului)**

Ultimul pas este să-i spunem aplicației React unde să găsească funcțiile de backend.

**3.1. Variabile de Mediu în React:** În directorul src/ al aplicației React, creați un fișier .env.production:

REACT\_APP\_GENERATE\_TEXT\_URL="URL-ul funcției generate-text-function obținut la pasul 1.3"  
REACT\_APP\_GENERATE\_IMAGE\_URL="URL-ul funcției generate-image-function"  
REACT\_APP\_GENERATE\_PACKAGE\_URL="URL-ul funcției generate-package-function"

**3.2. Actualizarea Serviciului API:** Modificați sparkyService.js (sau echivalentul) pentru a folosi aceste variabile:

const textApiUrl = process.env.REACT\_APP\_GENERATE\_TEXT\_URL;  
const imageApiUrl = process.env.REACT\_APP\_GENERATE\_IMAGE\_URL;  
  
// Exemplu de apel  
async function generateText(prompt) {  
 const response = await fetch(textApiUrl, {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json' },  
 body: JSON.stringify({ prompt })  
 });  
 // ... restul logicii  
}

**3.3. Re-implementarea Frontend-ului:** Rulați din nou pașii 2.1 și 2.3 pentru a actualiza aplicația live cu noile URL-uri de API.

### **Felicitări!**

Ați implementat cu succes întreaga arhitectură în cloud. Acum aveți:

* Un frontend React rapid, găzduit global.
* Un set de micro-servicii de backend scalabile și serverless.
* O bază de date Firestore.

Toate acestea lucrează împreună pentru a oferi experiența completă "Sparky's Workshop Factory" oricui, oriunde în lume. Următorul pas ar fi configurarea unui domeniu personalizat (DNS) pentru a direcționa www.montessori-workshops.org către bucket-ul Cloud Storage.