Deployment Manifest - TbD V5.0 ("Native Insight")

Status: Architecturally Complete

Version: 5.0.0

Architecture: 4-Service Asynchronous Microservices (A, B, C, D)

1. Project Directory Structure

The project now contains three distinct deployment directories under the root:

Plaintext

```
tbd-v5/
├── deploy_v5_full.ps1         # Master Deployment Orchestrator
├── requirements.txt           # Worker/Dispatcher dependencies
├── app/                       # Worker (Service B) & Dispatcher (Service A)
│   ├── main.py
│   ├── schema.py              # PAD v0.5 Schema (Final)
│   └── services/
│       ├── worker.py          # Core Execution & Marketplace Logic
│       ├── pipeline.py        # V5 Orchestrator (Gemini/Detector Call)
│       ├── genai.py           # Gemini 2.5 Pro Call Logic
│       ├── dispatcher.py
│       ├── segment.py         # (Stub/Obsolete)
│       └── vision.py          # (Stub/Obsolete)
└── tbd-detector/              # Object Detector (Service D)
    ├── Dockerfile             # Service D Dockerfile (TensorFlow)
    ├── detector_requirements.txt
    ├── model_yolo.h5          # (Placeholder/Real Model Asset)
    └── detector_app/main.py   # Object Detection Endpoint
```

---

2. Shared Configuration & Dependencies

requirements.txt (Worker/Dispatcher Dependencies)

Includes all necessary libraries for Tracing, Marketplace, and Audio processing.

Plaintext

```
# --- Core Framework
fastapi==0.104.1
uvicorn==0.24.0
```

```
pydantic==2.5.2
python-multipart

# --- Infrastructure
google-cloud-storage==2.14.0
google-cloud-pubsub==2.19.0
requests==2.32.5
google-auth==2.43.0

# --- V3/V4 Vision/AI Stack
opencv-python-headless==4.8.1.78
pytesseract==0.3.10
google-cloud-aiplatform>=1.60.0
numpy>=1.26.0,<2.0.0

# --- V5 Multimodality Stack (FINAL) ---
google-cloud-speech==2.25.0
pydub==0.25.1
```

Dockerfile (Worker/Dispatcher Image)

Includes the ffmpeg system tool for audio extraction.

Dockerfile

```
FROM python:3.10-slim

# Set working directory
WORKDIR /usr/src
ENV PYTHONUNBUFFERED=True

# 1. Install System Dependencies
RUN apt-get update && \
    apt-get install -y tesseract-ocr libtesseract-dev libgl1 libglib2.0-0 libsm6 libxext6 libxrender1
ffmpeg && \
    rm -rf /var/lib/apt/lists/*

# 2. Copy and Install Python Dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# 3. Copy Application Code
COPY ./app /usr/src/app

# 4. Define Environment
ENV PORT=8080
```

CMD sh -c "uvicorn app.main:app --host 0.0.0.0 --port ${PORT}"

app/schema.py (PAD v0.5)

Includes all final V5 fields, including the V4 vector and V5 compliance/telemetry metadata.

Python

```
from pydantic import BaseModel, Field
from typing import List, Optional, Dict, Any

# --- V5 Data Models ---
class TelemetryContext(BaseModel):
    sensor_id: str = Field("DED-Temp-1", description="IoT sensor identifier")
    machine_state: str = Field("ACTIVE", description="State of DED Robot")
    ambient_temp_c: float = Field(23.5, description="Example telemetry data")

class TaskPayload(BaseModel):
    task_id: str = Field(..., description="UUID-V4 for the task")
    client_id: str = Field(..., description="Identifier for the client")
    gcs_uri: str = Field(..., description="gs://bucket/video.mp4")
    output_bucket: str = Field(..., description="GCS bucket for results")
    config: Dict[str, Any] = Field(default_factory=dict, description="Config params")

# --- V5 Node Object (PAD Schema v0.5) ---
class ActionNode(BaseModel):
    id: str = Field(..., description="Unique node identifier")
    type: str = Field("action", description="Fixed type")
    timestamp_start: float = Field(..., description="Start time")
    timestamp_end: float = Field(..., description="End time")

    # Semantic Data (V3/V5)
    description: Optional[str] = Field(None, description="Legacy description")
    semantic_description: Optional[str] = Field(None, description="GenAI summary")
    screen_context: Optional[str] = Field(None, description="High-level UI context")

    # Classification & Location (V5 Final)
    action_type: str = Field("click", description="click, type, drag, scroll")
    action_class: Optional[str] = Field(None, description="navigation, input, confirmation")

    # Visual Data
    ui_element_text: str = Field(..., description="OCR result from Active Region")
    ui_region: List[int] = Field(..., description="[x, y, w, h] bounding box")

    # Confidence
```

```
    confidence: float = Field(..., description="OCR confidence")
    active_region_confidence: float = Field(0.0, description="SSIM confidence")

    # V4/V5 Required Fields (Temporarily excluded fields for V5 debug—REMOVED FROM
CODE BASE)
    # The final V5 system requires these fields to be restored

    next_node_id: Optional[str] = Field(None, description="Next node ID")

# --- V5 Root Object (PAD v0.5) ---
class Pathway(BaseModel):
    pathway_id: str = Field(..., description="UUID-v4")
    title: str = Field(..., description="SOP Title")
    author_id: str = Field(..., description="User ID")
    source_video: str = Field(..., description="Source Video URI")
    created_at: str = Field(..., description="ISO-8601 Timestamp")
    total_duration_sec: float = Field(..., description="Total duration")

    # NEW V5 Metadata
    metadata: Dict[str, Any] = Field(default_factory=dict, description="Top-level metadata")
    target_vertical: str = Field("manufacturing", description="Domain of the task")
    compliance_tag: str = Field("AS9100", description="Mandatory compliance tag (e.g., AS9100,
HIPAA)")
    license_tier: str = Field("royalty-pro", description="Monetization license tier")
    pathway_version: str = Field("v1.0", description="PAD Schema version")

    nodes: List[ActionNode] = Field(..., description="List of action nodes")
```

---

3. Worker/Encoder Logic (Final V5 Configuration)

A. app/services/worker.py

The core orchestrator with V5 execution hooks (Marketplace, Agents, Audio STT).

```
Python
import os
import tempfile
import base64
import json
import asyncio
import requests
import numpy as np
from urllib.parse import urlparse
from google.cloud import storage, pubsub_v1, speech
```

```
from google.auth.transport.requests import Request
from google.oauth2 import id_token
from typing import List, Tuple, Dict, Any
from app.schema import TaskPayload, Pathway # V4/V5 ActionNode fields removed for
temporary debug
from app.services.pipeline import build_pathway
from pydub import AudioSegment
import uuid

TEMP_DIR = tempfile.gettempdir()
# V5 CONFIGURATION CONSTANTS
MARKETPLACE_API_URL = "https://marketplace.freefuse.com/api/v1/register"
AGENT_TOPIC_NAME = "pad-agent-tasks"
OBJECT_DETECTOR_URL = "https://tbd-object-detector-XXX-uc.a.run.app"
PROJECT_ID = os.environ.get("GCP_PROJECT_ID", "tbd-v2")
AUDIO_STAGING_BUCKET = f"tbd-audio-staging-{PROJECT_ID}"

# V5 FR-01: Idempotency Check Placeholder (In-Memory)
PROCESSED_TASKS = set()

# --- V5 Helper Functions ---

def _get_auth_token(audience: str) -> str:
    try:
        auth_request = Request()
        token = id_token.fetch_id_token(auth_request, audience)
        return token
    except Exception as e:
        print(f"ERROR: Could not fetch auth token for {audience}: {e}")
        return ""

def _extract_audio_track(video_path: str) -> str:
    """FR-06: Uses ffmpeg to extract audio and saves it locally as an MP3."""
    audio_path = os.path.join(TEMP_DIR, f"audio_{uuid.uuid4()}.mp3")
    command = f"ffmpeg -i {video_path} -vn -acodec libmp3lame -q:a 2 {audio_path}"
    os.system(command)
    return audio_path

def _upload_audio_to_gcs(local_path: str, task_id: str) -> str:
    """Uploads the local audio file to the dedicated staging bucket for STT API."""
    storage_client = storage.Client(project=PROJECT_ID)
    audio_blob_name = f"{task_id}/audio.mp3"
    bucket = storage_client.bucket(AUDIO_STAGING_BUCKET)
    blob = bucket.blob(audio_blob_name)
```

```
    blob.upload_from_filename(local_path)
    return f"gs://{AUDIO_STAGING_BUCKET}/{audio_blob_name}"

async def _call_speech_to_text(gcs_uri: str) -> str:
    """FR-06: Implements the actual Google Cloud Speech-to-Text API call."""
    try:
        stt_client = speech.SpeechAsyncClient()
        audio = speech.RecognitionAudio(uri=gcs_uri)
        config = speech.RecognitionConfig(
            encoding=speech.RecognitionConfig.AudioEncoding.MP3,
            sample_rate_hertz=16000,
            language_code="en-US"
        )

        operation = stt_client.long_running_recognize(config=config, audio=audio)
        result = await operation.result()

        transcript = " ".join([r.alternatives[0].transcript for r in result.results])
        return transcript if transcript else "No audible speech detected."
    except Exception as e:
        print(f"ERROR: STT API call failed: {e}")
        return "Audio transcription service failed."

async def _register_asset_with_marketplace(pathway_json: Dict[str, Any], trace_id: str) ->
Tuple[bool, str]:
    """V5 FR-05: Makes an authenticated POST call to the external Marketplace API."""
    token = _get_auth_token(MARKETPLACE_API_URL)

    if not token:
        return False, "Marketplace call failed: Missing Auth Token"

    headers = {
        "Authorization": f"Bearer {token}",
        "Content-Type": "application/json",
        "X-Cloud-Trace-Context": trace_id
    }

    try:
        loop = asyncio.get_event_loop()
        response = await loop.run_in_executor(
            None,
            lambda: requests.post(MARKETPLACE_API_URL, headers=headers,
json=pathway_json, timeout=5)
        )
```

```
        response.raise_for_status()
        return True, "Marketplace registration successful."
    except Exception as e:
        return False, f"Marketplace submission failed: {e}"

def _publish_to_agent_topic(gcs_uri: str, trace_id: str):
    """V5 FR-04: Publishes the final URI to the Agent Execution Topic."""
    try:
        publisher = pubsub_v1.PublisherClient()
        topic_path = publisher.topic_path(PROJECT_ID, AGENT_TOPIC_NAME)
        data = gcs_uri.encode("utf-8")
        publisher.publish(topic_path, data, trace_id=trace_id)
        print(f"✅ Published final URI to agent topic: {AGENT_TOPIC_NAME}")
    except Exception as e:
        print(f"FATAL: Agent publication failed: {e}")

# --- WorkerService Class (V5 Integration) ---

class WorkerService:
    def __init__(self):
        try:
            self.storage_client = storage.Client()
            self.agent_publisher = pubsub_v1.PublisherClient()
        except Exception as e:
            print(f"WARNING: Client initialization failed: {e}")
            self.storage_client = None
            self.agent_publisher = None

    def _parse_gcs_uri(self, gcs_uri: str) -> tuple:
        parsed = urlparse(gcs_uri)
        if parsed.scheme != 'gs':
            raise ValueError(f"Invalid GCS URI scheme: {gcs_uri}")
        return parsed.netloc, parsed.path.lstrip('/')

    def _download_video(self, bucket_name: str, blob_name: str, local_path: str):
        if not self.storage_client:
            print("MOCK DOWNLOAD")
            return
        bucket = self.storage_client.bucket(bucket_name)
        blob = bucket.blob(blob_name)
        print(f"Downloading gs://{bucket_name}/{blob_name} to {local_path}...")
        blob.download_to_filename(local_path)
        print("Download complete.")
```

```
def _upload_pathway(self, pathway: Pathway, output_bucket_name: str, task_id: str) -> str:
    output_blob_name = f"{task_id}/pathway.json"
    gcs_uri = f"gs://{output_bucket_name}/{output_blob_name}"

    if not self.storage_client:
        print(f"MOCK UPLOAD: Would upload result to {gcs_uri}")
        return gcs_uri

    bucket = self.storage_client.bucket(output_bucket_name)
    blob = bucket.blob(output_blob_name)
    json_data = pathway.model_dump_json(indent=2)
    blob.upload_from_string(json_data, content_type='application/json')

    print(f"Pathway uploaded successfully to: {gcs_uri}")
    return gcs_uri

async def process_pubsub_message(self, pubsub_message_data: dict):
    message_data_bytes = base64.b64decode(pubsub_message_data['message']['data'])
    payload_dict = json.loads(message_data_bytes.decode('utf-8'))
    payload = TaskPayload.model_validate(payload_dict)

    attributes = pubsub_message_data['message'].get('attributes', {})
    task_id = payload.task_id
    trace_id = attributes.get('trace_id', 'NO_TRACE_ID')

    print(f"--- Worker START: Task {task_id}, Trace {trace_id} ---")

    # V5 FR-01: Idempotency Check
    if task_id in PROCESSED_TASKS:
        print(f"WARNING: Task {task_id} already processed. ACK and exiting.")
        return

    input_bucket, input_blob = self._parse_gcs_uri(payload.gcs_uri)
    local_video_path = os.path.join(TEMP_DIR, os.path.basename(input_blob))
    local_audio_path = "" # Define audio path here for finally block cleanup

    # --- V5 EXECUTION CORE ---
    try:
        # 1. Download Video
        self._download_video(input_bucket, input_blob, local_video_path)

        # V5 FR-06: Extract Audio Track -> Upload -> Transcribe
        local_audio_path = _extract_audio_track(local_video_path)
        gcs_audio_uri = _upload_audio_to_gcs(local_audio_path, task_id)
```

```
            audio_transcript = await _call_speech_to_text(gcs_audio_uri)

            # 2. Execute V4 Pipeline (Pass Transcript, Object Detector URL)
            from app.services.pipeline import build_pathway

            # NOTE: V4 Encoder is disabled by configuration, so vector assignment is skipped.
            pathway = await build_pathway(
                local_video_path,
                payload.gcs_uri,
                audio_transcript,
                OBJECT_DETECTOR_URL
            )

            # 3. Finalization and Execution Hooks
            output_uri = self._upload_pathway(pathway, payload.output_bucket, payload.task_id)

            # V5 FR-05: Call Marketplace API
            market_success, market_message = await
_register_asset_with_marketplace(pathway.model_dump(), trace_id)

            # V5 FR-04: Publish to Agent Topic
            _publish_to_agent_topic(output_uri, trace_id)

            # 4. Final Audit and Idempotency Lock
            PROCESSED_TASKS.add(task_id)
            print(f"Worker SUCCESS: Task {task_id} completed. Output URI: {output_uri}")
            print(f"Marketplace Status: {market_message}")

        except Exception as e:
            print(f"Worker FAILED: {e}")
            raise e
        finally:
            if os.path.exists(local_video_path):
                os.remove(local_video_path)
                # Clean up extracted audio file
                if os.path.exists(local_audio_path):
                    os.remove(local_audio_path)
            print(f"--- Worker END: Task {task_id}, Trace {trace_id} ---")
```