PrivacyScrub V5: Enterprise Deployment Manifest

Author: Greg Burns **Version:** 5.0 (Enterprise Edition) **Date:** November 26, 2025 **Architecture:** Microservices / GPU-Accelerated / Infrastructure as Code (IaC)

1. Project Overview and Directory Structure

PrivacyScrub V5 transitions from a monolithic Colab notebook to a professional, local development workflow. This repository implements a distributed microservices architecture deployed to Google Cloud Platform (GCP) using Terraform.

1.1 Directory Layout

Create the following structure locally:

```
privacyscrub-v5/
├── terraform/              # Infrastructure as Code (GCP Resources)
│   ├── main.tf             # Core resource definitions
│   ├── variables.tf        # Configuration variables
│   ├── outputs.tf          # Deployment outputs (URLs)
│   └── provider.tf         # GCP Provider configuration
├── services/               # Microservices Source Code
│   ├── gateway/            # API Gateway (Ingress)
│   │   ├── main.py
│   │   └── Dockerfile
│   ├── orchestrator/       # Job Lifecycle Manager
│   │   ├── main.py
│   │   └── Dockerfile
│   └── gpu-worker/         # GPU Inference Engine
│       ├── main.py
│       ├── inference.py
│       └── Dockerfile
├── scripts/                # Automation
│   └── deploy.ps1          # PowerShell Deployment Script
└── README.md
```

2. Infrastructure as Code (Terraform)

These configurations provision the necessary GCP resources: Artifact Registry, Cloud Storage, Firestore, Cloud Tasks, and Cloud Run services.

2.1 `terraform/main.tf`

```
# --- terraform/main.tf ---
# Provisions the PrivacyScrub V5 Enterprise Environment

# 1. Artifact Registry (Docker Images)
resource "google_artifact_registry_repository" "repo" {
  location      = var.region
  repository_id = "privacyscrub-v5"
  format        = "DOCKER"
}

# 2. Cloud Storage (Media Buckets)
resource "google_storage_bucket" "media_bucket" {
  name          = "${var.project_id}-media-v5"
  location      = var.region
  force_destroy = true

  # Strict 24h TTL for privacy compliance (FR-V5-05)
  lifecycle_rule {
    condition { age = 1 }
    action { type = "Delete" }
  }
}

# 3. Firestore (State Database)
resource "google_firestore_database" "database" {
  name        = "(default)"
  location_id = var.region
  type        = "FIRESTORE_NATIVE"
}

# 4. Cloud Tasks Queue (Job Orchestration)
resource "google_cloud_tasks_queue" "video_queue" {
  name     = "privacyscrub-video-queue"
  location = var.region
}

# 5. Cloud Run Services
# NOTE: Initial deployment uses a placeholder image.
# The actual application code is deployed via the build script later.
```

```
# Service A: API Gateway (Ingress)
resource "google_cloud_run_v2_service" "gateway" {
  name     = "privacyscrub-gateway"
  location = var.region
  template {
    containers {
      image = "us-docker.pkg.dev/cloudrun/container/hello" # Placeholder
      env {
        name = "GCP_PROJECT_ID"; value = var.project_id
      }
      # Orchestrator URL is injected post-deployment
    }
  }
}

# Service B: Orchestrator (Job Manager)
resource "google_cloud_run_v2_service" "orchestrator" {
  name     = "privacyscrub-orchestrator"
  location = var.region
  template {
    containers {
      image = "us-docker.pkg.dev/cloudrun/container/hello" # Placeholder
      env {
        name = "GCS_BUCKET_NAME"; value = google_storage_bucket.media_bucket.name
      }
    }
  }
}

# Service C: GPU Worker (Inference Engine)
resource "google_cloud_run_v2_service" "gpu_worker" {
  name     = "privacyscrub-gpu-worker"
  location = var.region
  launch_stage = "BETA" # Required for GPU features

  template {
    scaling {
      max_instance_count = 5
    }
    containers {
      image = "us-docker.pkg.dev/cloudrun/container/hello" # Placeholder
      resources {
        limits = {
          cpu    = "4000m"
```

```
        memory = "16Gi"
        # "[nvidia.com/gpu](https://nvidia.com/gpu)" = "1" # Uncomment if GPU quota is
approved/available
      }
    }
  }
 }
}
```

2.2 `terraform/variables.tf`

```
variable "project_id" {
  description = "The GCP Project ID"
  type        = string
}

variable "region" {
  description = "GCP Region for resources"
  type        = string
  default     = "us-central1"
}
```

2.3 `terraform/outputs.tf`

```
output "gateway_url" {
  value = google_cloud_run_v2_service.gateway.uri
}

output "orchestrator_url" {
  value = google_cloud_run_v2_service.orchestrator.uri
}
```

3. Service Implementation (Python Microservices)

3.1 API Gateway (`services/gateway/main.py`)

This service handles external client requests, authentication, and initial job creation.

```
# services/gateway/main.py
from fastapi import FastAPI, UploadFile, File, Form, HTTPException
from google.cloud import firestore, tasks_v2
```

```
import os, json, uuid

app = FastAPI(title="PrivacyScrub V5 Gateway")

# Configuration
PROJECT_ID = os.environ.get("GCP_PROJECT_ID")
ORCHESTRATOR_URL = os.environ.get("ORCHESTRATOR_URL")
QUEUE = "privacyscrub-video-queue"
REGION = "us-central1" # Consider moving to env var in prod

db = firestore.Client()
tasks_client = tasks_v2.CloudTasksClient()

@app.post("/v1/video")
async def submit_video(file: UploadFile = File(...), webhook_url: str = Form(None)):
    job_id = str(uuid.uuid4())

    # 1. Create Job Record
    db.collection("jobs").document(job_id).set({
        "status": "QUEUED",
        "webhook_url": webhook_url,
        "created_at": firestore.SERVER_TIMESTAMP
    })

    # 2. Dispatch to Orchestrator
    # In V5 Enterprise, the Gateway validates the request but offloads
    # heavy I/O (like GCS upload) to the Orchestrator via a secure internal link.
    if not ORCHESTRATOR_URL:
        raise HTTPException(500, "Orchestrator not configured")

    parent = tasks_client.queue_path(PROJECT_ID, REGION, QUEUE)
    task = {
        "http_request": {
            "http_method": tasks_v2.HttpMethod.POST,
            "url": f"{ORCHESTRATOR_URL}/internal/ingest",
            "headers": {"Content-Type": "application/json"},
            "body": json.dumps({"job_id": job_id, "filename": file.filename}).encode()
        }
    }
    tasks_client.create_task(request={"parent": parent, "task": task})

    return {"job_id": job_id, "status": "QUEUED"}

@app.get("/v1/jobs/{job_id}")
```

```
def get_status(job_id: str):
    doc = db.collection("jobs").document(job_id).get()
    if not doc.exists:
        raise HTTPException(404, "Job not found")
    return doc.to_dict()
```

3.2 Orchestrator (`services/orchestrator/main.py`)

Manages the complex lifecycle of splitting, dispatching to GPU workers, and stitching results.

```
# services/orchestrator/main.py
from fastapi import FastAPI
from pydantic import BaseModel
# ... imports for GCS, Tasks ...

app = FastAPI()

@app.post("/internal/ingest")
def ingest_video(payload: dict):
    # 1. Download or Receive Video Stream
    # 2. Analyze Duration
    # 3. Split into Chunks (FFMPEG)
    # 4. Dispatch Tasks to GPU Worker
    pass

@app.post("/internal/stitch")
def stitch_video(payload: dict):
    # 1. Verify all chunks complete
    # 2. Concatenate chunks
    # 3. Strip Metadata
    # 4. Trigger Webhook (if configured)
    pass
```

3.3 GPU Worker (`services/gpu-worker/main.py`)

The high-performance engine. Runs the multi-model stack.

```
# services/gpu-worker/main.py
# ... imports (torch, ultralytics, cv2) ...

# Model Initialization (Load to GPU)
model = YOLO('yolov8m.pt') # Use Medium/Large model on GPU
```

```
@app.post("/internal/process-chunk")
def process_chunk(payload: dict):
    # 1. Download Chunk
    # 2. Inference Loop (Batch processing on GPU)
    # 3. Object Tracking (DeepSORT)
    # 4. Redaction
    # 5. Upload Result
    pass
```

GPU Worker Dockerfile:

```
# services/gpu-worker/Dockerfile
FROM pytorch/pytorch:2.1.0-cuda12.1-cudnn8-runtime

WORKDIR /app

# Install System Deps (FFMPEG with NVDEC support requires custom build or specific repos)
RUN apt-get update && apt-get install -y ffmpeg libsm6 libxext6

# Install Python Stack
RUN pip install ultralytics moviepy opencv-python-headless google-cloud-storage fastapi
uvicorn

COPY . .

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8080"]
```

4. Deployment Automation (PowerShell)

This script automates the "Build -> Push -> Deploy" cycle, glueing the Terraform infrastructure to the application code.

`scripts/deploy.ps1`

```
# --- scripts/deploy.ps1 ---
# Usage: ./deploy.ps1 -ProjectId "my-gcp-project"

param (
    [string]$ProjectId
)
```

```
$Region = "us-central1"
$Repo = "$Region-docker.pkg.dev/$ProjectId/privacyscrub-v5"

Write-Host "🚀 Starting PrivacyScrub V5 Enterprise Deployment..." -ForegroundColor Green

# 1. Enable Required GCP APIs
Write-Host "🔌 Enabling APIs..."
gcloud services enable run.googleapis.com artifactregistry.googleapis.com
cloudtasks.googleapis.com firestore.googleapis.com --project $ProjectId

# 2. Terraform Provisioning
Write-Host "🏗️ Provisioning Infrastructure with Terraform..."
cd ../terraform
terraform init
terraform apply -var="project_id=$ProjectId" -auto-approve
cd ../scripts

# 3. Build & Push Docker Images
$Services = @("gateway", "orchestrator", "gpu-worker")

foreach ($Service in $Services) {
    Write-Host "📦 Building Service: $Service..."
    $ImageUri = "$Repo/$Service`:latest"

    # Build using the service-specific Dockerfile
    docker build -t $ImageUri -f "../services/$Service/Dockerfile" "../services/$Service"

    # Push to Artifact Registry
    docker push $ImageUri
}

# 4. Deploy/Update Cloud Run Services
# This step updates the running services with the new image we just pushed.
# We also inject the service URLs into each other for service-to-service communication.

# Get Service URLs
$OrchestratorUrl = gcloud run services describe privacyscrub-orchestrator --region $Region
--format 'value(status.url)'
$GpuWorkerUrl = gcloud run services describe privacyscrub-gpu-worker --region $Region
--format 'value(status.url)'

Write-Host "🚀 Deploying/Updating Services..."

# Deploy Gateway (Inject Orchestrator URL)
```

```
gcloud run deploy privacyscrub-gateway `
    --image "$Repo/gateway:latest" `
    --region $Region `
    --set-env-vars "ORCHESTRATOR_URL=$OrchestratorUrl"

# Deploy Orchestrator (Inject Worker URL)
gcloud run deploy privacyscrub-orchestrator `
    --image "$Repo/orchestrator:latest" `
    --region $Region `
    --set-env-vars "WORKER_URL=$GpuWorkerUrl"

Write-Host "✅ V5 Deployment Complete!" -ForegroundColor Green
```