SPARC-P PubApp Deployment Guide (Pixel Streaming / Server-Side Rendering)

Overview

This notebook provides step-by-step instructions for deploying SPARC-P on **UF RC PubApps** using **server-side rendering** for thin clients. Instead of serving Unity WebGL assets to the browser, Unity runs on the PubApps GPU and streams rendered video/audio to the browser through WebRTC.

Key Changes from the WebGL Deployment

1. **Audio2Face removed** to free GPU memory (~2GB VRAM reclaimed).
2. **Unity Linux Server Build** runs in a container on PubApps and performs server-side rendering.
3. **Signaling Server** is added as a lightweight Node.js container for WebRTC negotiation.
4. **Single L4 VRAM budget** is enforced across all runtime services.

1.0 Prerequisites

1.1 Required Allocations

Before deploying to PubApps, ensure you have:

1. **PA-Instance allocation** ($300/year) - Request via HiPerGator Service Purchase Form
2. **PA-GPU allocation** - Single NVIDIA L4 (24GB VRAM)
3. **Completed Risk Assessment** - See Risk Assessment Documentation

1.2 PubApp Instance Setup

After purchasing a PA-Instance, open a support ticket to provision your instance:

- Instance accessible via SSH from HiPerGator
- Project account provisioned (commonly `SPARCP`)
- Default resources: 1x L4 (24GB), 2 vCPUs, 16GB RAM, 1TB `/pubapps` storage

1.3 Strict L4 VRAM Budget (24GB)

Use this budget to prevent OOM and unstable services:

- **LLM (vLLM, `gpt-oss-20b`, 4-bit quantized)**: ~13GB
- **Unity Linux Server Rendering**: ~3.5GB
- **Riva TTS/ASR (Embedded config)**: ~3GB
- **System/driver/runtime overhead buffer**: ~2-3GB

Enforcement guidance:

- Start services in this order: Riva -> LLM -> Signaling -> Unity
- Verify with `nvidia-smi` after each startup
- If usage exceeds budget, reduce Unity quality profile and/or LLM KV cache limits

1.4 Architecture Overview (Server-Side Rendering)

```
┌──────────────────────────────────────────────────────────────────────────┐
│                              Public Browser                              │
│                   (thin client, no local Unity/WebGL)                    │
└────────────────────────────────┬─────────────────────────────────────────┘
                                 │ HTTPS/WSS
┌────────────────────────────────▼─────────────────────────────────────────┐
│                                 NGINX                                    │
│                  TLS termination + reverse proxy routing                 │
└────────────────┬───────────────────────────────────────────────┬─────────┘
                 │                                               │
        /signal /ws                                           /api/*
                 │                                               │
┌────────────────▼──────────────┐                    ┌───────────▼─────────┐
│ Node.js Signaling Container   │                    │ FastAPI + LLM (vLLM)│
│ WebRTC session negotiation    │                    │ gpt-oss-20b 4-bit   │
└────────────────┬──────────────┘                    └───────────┬─────────┘
                 │                                               │
                 │ WebRTC setup                                  │ localhost
                 │                                               │
┌────────────────▼───────────────────────────────────────────────▼──────────┐
│                        Unity Linux Server Container                       │
│          (server-side rendering + Render Streaming package)               │
│                   local calls to Riva + backend services                  │
└────────────────────────────────┬──────────────────────────────────────────┘
                                 │
                       ┌─────────▼─────┐
                       │ Riva Server   │
                       │ ASR + TTS     │
                       └───────────────┘
```

Data flow: Browser connects to the **Signaling Server** -> signaling negotiates WebRTC -> browser receives A/V stream rendered by **Unity container** -> Unity exchanges local requests with **FastAPI/LLM** and **Riva**.

2.0 Transfer Trained Models from HiPerGator

2.1 Sync Models to PubApps Storage

```
# On HiPerGator
rsync -avz --progress \
  /blue/jasondeanarnold/SPARCP/trained_models/ \
  SPARCP@pubapps-vm.rc.ufl.edu:/pubapps/SPARCP/models/
```

2.2 Verify Model Transfer

```
ssh SPARCP@pubapps-vm.rc.ufl.edu
ls -lh /pubapps/SPARCP/models/
# Expected: CaregiverAgent/, C-LEAR_CoachAgent/, SupervisorAgent/
```

3.0 Setup Conda Environment (Backend + vLLM)

3.1 Install Conda

```
ssh SPARCP@pubapps-vm.rc.ufl.edu
cd ~
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
bash Miniconda3-latest-Linux-x86_64.sh -b -p ~/miniconda3
~/miniconda3/bin/conda init bash
source ~/.bashrc
conda --version
```

3.2 Create Runtime Environment

```
cd /pubapps/SPARCP
conda env create -f environment_backend.yml -p
/pubapps/SPARCP/conda_envs/sparc_backend
conda activate /pubapps/SPARCP/conda_envs/sparc_backend
python -c "import torch, fastapi; print(torch.__version__)"
```

4.0 Build and Stage Containers

4.1 Required Runtime Images

Build or pull these images before service enablement:

1. `sparc/llm-backend:latest` (FastAPI + vLLM)
2. `sparc/unity-server:latest` (Unity Linux server build with Render Streaming)
3. `sparc/signaling-server:latest` (Unity Render Streaming signaling server)
4. `nvcr.io/nvidia/riva/riva-speech:2.16.0-server`

4.2 Reference Build Commands

```
podman build -f Dockerfile.mas -t sparc/llm-backend:latest .
podman build -f Dockerfile.unity-server -t sparc/unity-server:latest .
podman build -f Dockerfile.signaling -t sparc/signaling-server:latest .
podman pull nvcr.io/nvidia/riva/riva-speech:2.16.0-server
```

5.0 Podman + Systemd Orchestration

5.1 Create Runtime Directories

```
mkdir -p /pubapps/SPARCP/{models,riva_models,logs}
mkdir -p ~/.config/containers/systemd
```

5.2 Define `avatar.pod` (Ports + GPU-aware services)

```
# ~/.config/containers/systemd/avatar.pod
[Unit]
Description=SPARC-P Avatar Pod (Pixel Streaming)
After=network-online.target

[Pod]
PodName=sparc-avatar
PublishPort=8000:8000
PublishPort=8080:8080
PublishPort=3478:3478/udp
PublishPort=49152-49200:49152-49200/udp
PublishPort=49152-49200:49152-49200/tcp

[Install]
WantedBy=default.target
```

- `8000`: FastAPI backend
- `8080`: Signaling HTTP/WebSocket
- `3478/udp`: STUN baseline (if used)
- `49152-49200`: WebRTC media candidate range (adjust per RC network policy)

5.3 Riva Container Service

```
# ~/.config/containers/systemd/riva-server.container
[Unit]
Description=SPARC-P Riva Speech Server
After=avatar-pod.service
Requires=avatar-pod.service

[Container]
Pod=sparc-avatar.pod
Image=nvcr.io/nvidia/riva/riva-speech:2.16.0-server
ContainerName=riva-server
Volume=/pubapps/SPARCP/riva_models:/data:Z
PublishPort=50051:50051
Device=nvidia.com/gpu=all
Environment=NVIDIA_VISIBLE_DEVICES=all
Exec=/opt/riva/bin/riva_server --riva_model_repo=/data/models
```

```
[Service]
Restart=always
TimeoutStartSec=300

[Install]
WantedBy=default.target
```

5.4 Backend/LLM Container Service

```
# ~/.config/containers/systemd/sparc-backend.container
[Unit]
Description=SPARC-P FastAPI + vLLM Backend
After=avatar-pod.service riva-server.service
Requires=avatar-pod.service riva-server.service

[Container]
Pod=sparc-avatar.pod
Image=sparc/llm-backend:latest
ContainerName=sparc-backend
Volume=/pubapps/SPARCP/models:/pubapps/SPARCP/models:Z
Environment=MODEL_ID=gpt-oss-20b
Environment=QUANTIZATION=4bit
Environment=RIVA_SERVER=localhost:50051
Device=nvidia.com/gpu=all
Exec=uvicorn main:app --host 0.0.0.0 --port 8000 --workers 1

[Service]
Restart=always
RestartSec=10

[Install]
WantedBy=default.target
```

5.5 Signaling Server Container Service

```
# ~/.config/containers/systemd/signaling-server.container
[Unit]
Description=SPARC-P Unity Render Streaming Signaling Server
After=avatar-pod.service
Requires=avatar-pod.service

[Container]
Pod=sparc-avatar.pod
Image=sparc/signaling-server:latest
ContainerName=signaling-server
Environment=HTTP_PORT=8080
Environment=PUBLIC_HOST=sparc-p.rc.ufl.edu
Exec=node server.js --httpPort 8080
```

```
[Service]
Restart=always
RestartSec=5

[Install]
WantedBy=default.target
```

5.6 Unity Server Container Service (GPU via CDI)

```
# ~/.config/containers/systemd/unity-server.container
[Unit]
Description=SPARC-P Unity Linux Server (Render Streaming)
After=avatar-pod.service signaling-server.service sparc-backend.service riva-
server.service
Requires=avatar-pod.service signaling-server.service sparc-backend.service riva-
server.service

[Container]
Pod=sparc-avatar.pod
Image=sparc/unity-server:latest
ContainerName=unity-server
Device=nvidia.com/gpu=all
Environment=SIGNALING_URL=ws://localhost:8080
Environment=BACKEND_URL=http://localhost:8000
Environment=RIVA_URL=localhost:50051
Exec=/app/SPARC-P.x86_64 -logFile /dev/stdout -batchmode -force-vulkan

[Service]
Restart=always
RestartSec=5

[Install]
WantedBy=default.target
```

5.7 Enable Services

```
systemctl --user daemon-reload
systemctl --user enable --now avatar-pod
systemctl --user enable --now riva-server
systemctl --user enable --now sparc-backend
systemctl --user enable --now signaling-server
systemctl --user enable --now unity-server
```

6.0 Configure NGINX Reverse Proxy

6.1 Ticket Request Template

```
Subject: NGINX Configuration for SPARC-P Pixel Streaming (PubApps)

Body:
Please configure reverse proxy for SPARC-P server-side rendering deployment:

1. SSL certificate for sparc-p.rc.ufl.edu
2. Proxy routes:
   - /api/ -> http://localhost:8000/
   - /signal/ -> http://localhost:8080/
3. WebSocket upgrade enabled for /signal/
4. UF Shibboleth SSO enabled for access control
5. Preserve headers for WebRTC signaling endpoints
```

6.2 Reference NGINX Config

```
server {
    listen 443 ssl http2;
    server_name sparc-p.rc.ufl.edu;

    ssl_certificate /path/to/cert.pem;
    ssl_certificate_key /path/to/key.pem;

    location /api/ {
        proxy_pass http://localhost:8000/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    location /signal/ {
        proxy_pass http://localhost:8080/;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
    }
}
```

7.0 Deployment Checklist

7.1 Pre-Deployment

- [ ] PubApps instance provisioned
- [ ] Models transferred to `/pubapps/SPARCP/models/`

- [ ] Conda backend environment created
- [ ] Runtime images available (backend, unity-server, signaling, riva)
- [ ] UF RC risk assessment completed

7.2 Service Deployment

- [ ] `avatar-pod` active
- [ ] `riva-server` active
- [ ] `sparc-backend` active
- [ ] `signaling-server` active
- [ ] `unity-server` active
- [ ] All services configured for auto-start

7.3 Validation

- [ ] `https://sparc-p.rc.ufl.edu/api/health` responds
- [ ] `/signal/` endpoint negotiates WebRTC sessions
- [ ] Browser receives Unity-rendered stream
- [ ] Unity can call backend and Riva on localhost
- [ ] `nvidia-smi` confirms runtime fits 24GB budget

8.0 Monitoring and Maintenance

8.1 Service Commands

```
systemctl --user status avatar-pod riva-server sparc-backend signaling-server
unity-server
journalctl --user -u unity-server -n 100 -f
journalctl --user -u signaling-server -n 100 -f
journalctl --user -u sparc-backend -n 100 -f
journalctl --user -u riva-server -n 100 -f
```

8.2 Resource Monitoring

```
nvidia-smi
free -h
df -h /pubapps/SPARCP
```

8.3 VRAM Guardrail Checks

```
# Ensure aggregate stays within single L4 budget
nvidia-smi --query-compute-apps=pid,process_name,used_memory --format=csv
```

9.0 Security and Compliance

1. **Authentication**: UF Shibboleth SSO
2. **Transport Security**: TLS 1.2+
3. **Data Handling**: Session state managed through approved services, no PHI persistence in container logs
4. **Access Control**: SSH key-based project access for operators
5. **Compliance**: Follow UFIT RC PubApps Policy

10.0 Troubleshooting

10.1 Common Issues

Issue: Unity stream connects but no video

```
journalctl --user -u unity-server -n 100
journalctl --user -u signaling-server -n 100
# Verify signaling URL and WebRTC ports are open per RC policy
```

Issue: GPU memory exceeded

```
nvidia-smi
# Reduce Unity quality settings / frame rate
# Lower LLM KV cache or max context configuration
```

Issue: Browser cannot negotiate WebRTC

```
curl -i https://sparc-p.rc.ufl.edu/signal/
# Verify NGINX WebSocket upgrade config and signaling server health
```

Issue: Backend unavailable from Unity container

```
curl http://localhost:8000/health
# Check sparc-backend service logs and model load status
```

11.0 Summary

This deployment variant replaces client-side WebGL with server-side rendering optimized for thin clients:

1. Unity runs on PubApps GPU in a Linux server container.
2. Browser sessions connect through a Node signaling service for WebRTC negotiation.

3. FastAPI + LLM + Riva remain local services for dialog, inference, and speech.
4. Audio2Face is removed to maintain single-L4 VRAM stability.
5. Runtime is orchestrated with Podman + systemd user services on PubApps.

Next steps:

1. Build and publish the Unity Linux server and signaling images.
2. Enable services in order and validate WebRTC signaling and stream quality.
3. Tune Unity and model runtime settings to keep total VRAM within the 24GB budget.