GITGOOD
Help gitGood, gitBetter

DEPLOYMENT DIAGRAM DESCRIPTION

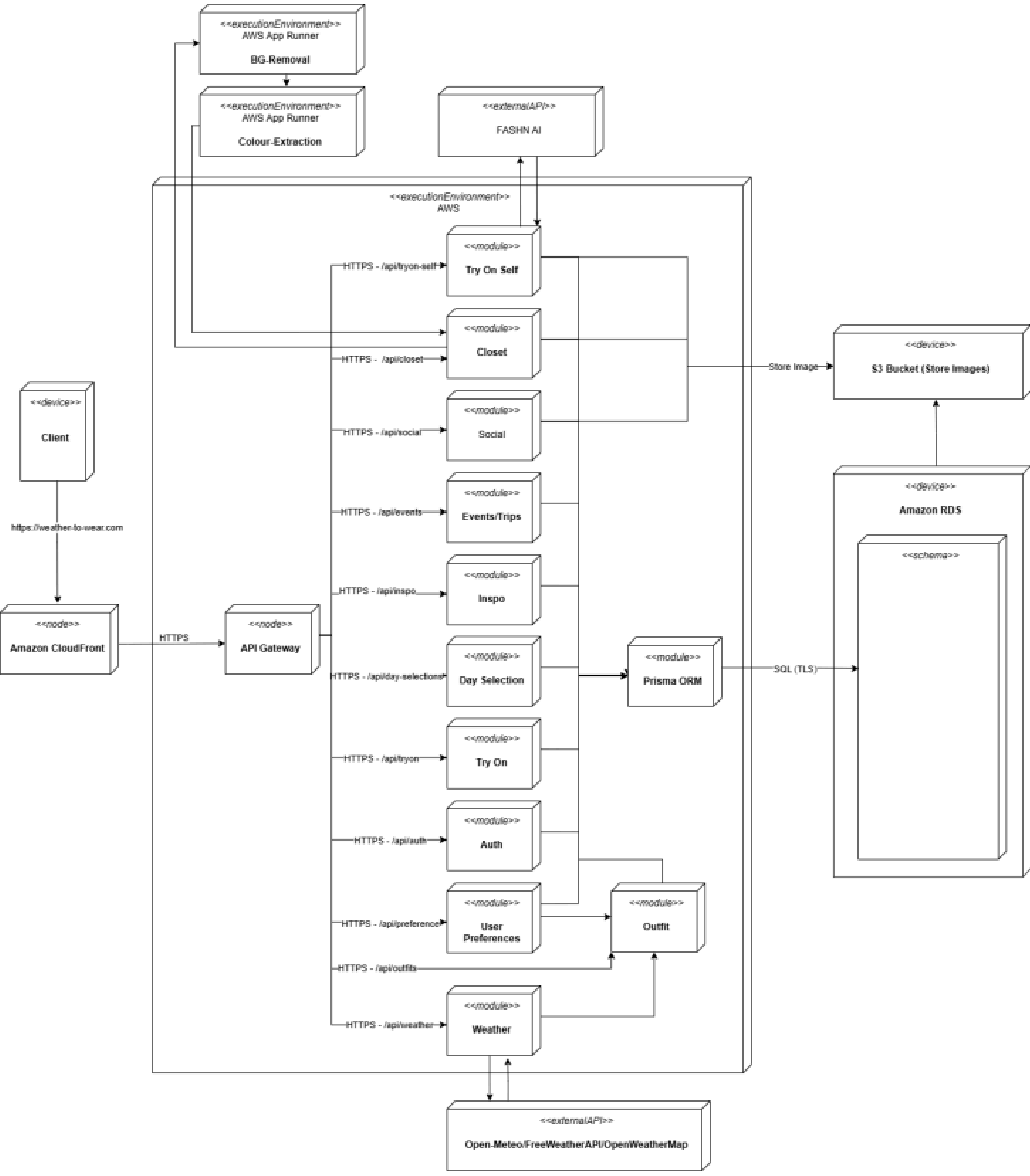
DEMO 3

WEATHER TO WEAR

EPI-USE

KYLE LIEBENBERG
DIYA BUDHIA
ALISHA PERUMAL
IBRAHIM SAID
TAYLOR SERGEL

Table of Contents



<<executionEnvironment>>
AWS App Runner
BG-Removal
<<executionEnvironment>>
AWS App Runner
Colour-Extraction
<<externalAPI>>
FASHN AI
<<executionEnvironment>>
AWS
<<module>>
Try On Self
<<module>>
Closet
<<module>>
Social
<<module>>
Events/Trips
<<module>>
Inspo
<<module>>
Day Selection
<<module>>
Try On
<<module>>
Auth
<<module>>
User Preferences
<<module>>
Outfit
<<module>>
Weather
<<module>>
Prisma ORM
HTTPS - /api/tryon-self
HTTPS - /api/closet
HTTPS - /api/social
HTTPS - /api/events
HTTPS - /api/inspo
HTTPS - /api/day-selections
HTTPS - /api/tryon
HTTPS - /api/auth
HTTPS - /api/preference
HTTPS - /api/outfits
HTTPS - /api/weather
<<device>>
Client
https://weather-to-wear.com
<<node>>
Amazon CloudFront
HTTPS
<<node>>
API Gateway
Store Image
<<device>>
S3 Bucket (Store Images)
<<device>>
Amazon RDS
<<schema>>
SQL (TLS)
<<externalAPI>>
Open-Meteo/FreeWeatherAPI/OpenWeatherMap

Legend

- <<node>> – execution environment or managed service (e.g., App Runner, CloudFront, RDS).
- <<component>> – deployable software unit (e.g., “Weather to Wear API”).
- <<artifact>> – binary/config/data item (e.g., Docker image, React build, secrets, user images).
- <<interface>> – interaction contract (REST/HTTPS endpoints).

Nodes, Components, and Artifacts

CLIENT EDGE

- Client <<node>>
- Browser/PWA that requests the frontend over HTTPS and calls the backend REST API.

FRONTEND DELIVERY

- Amazon CloudFront – Frontend CDN <<node>>
    - HTTPS <<interface>> to the client.
    - Serves the React PWA from the S3 origin using Origin Access Control (OAC).
- Amazon S3 – Frontend Bucket <<node>>
    - Build React App (build/) <<artifact>> uploaded by CI/CD.
    - Public access blocked; only CloudFront reads via OAC.
- GitHub Actions (OIDC) – Deploy Frontend <<node>>
    - Build Artifact <<artifact>> (React build output) synced to the frontend S3 bucket; CloudFront invalidation completes the release.

BACKEND COMPUTE

- AWS App Runner – Backend API <<node>>
    - Weather to Wear API (Node.js + Express + Prisma) <<component>>
    - REST API v1 /api/* <<interface>> exposed to the client over HTTPS.
    - Docker image weather-backend:prod <<artifact>> pulled from ECR.
    - Reads secrets at start-up; runs prisma migrate deploy before serving traffic (ensures schema parity).
    - Environment (plain): PORT, NODE_ENV, S3_BUCKET_NAME, S3_REGION, UPLOADS_CDN_DOMAIN, BG_REMOVAL_URL, COLOR_EXTRACT_URL.

IMAGE PROCESSING MICROSERVICES

- AWS App Runner – bg-removal (public) <<node>>
  - Background Removal Service (U^2-Net) <<component>>
  - HTTP POST /remove-bg <<interface>>
  - Docker image bg-removal:prod <<artifact>>.
- AWS App Runner – color-extract (public) <<node>>
  - Color Extraction Service (KMeans) <<component>>
  - HTTP POST /extract-colors <<interface>>
  - Docker image color-extract:prod <<artifact>>.

CONTAINER REGISTRY & SECRETS

- Amazon ECR <<node>>
  - OCI Images (tags: prod) <<artifact>> for backend and both microservices.
- AWS Secrets Manager <<node>>
  - Secrets <<artifact>>: DATABASE_URL, JWT_SECRET, and weather API keys.
  - Backend fetches via GetSecretValue at boot.

PRIVATE NETWORK & DATA STORES

- VPC: w2w-prod-vpc <<node>>
  - S3 Gateway Endpoint <<node>> for private S3 egress (saves NAT cost/latency).
  - App Runner VPC Connector <<node>> provides private connectivity to RDS on port 5432/TLS.
  - Private Subnets <<node>> host:
    - Amazon RDS PostgreSQL <<node>> (instance w2w-postgres-prod, port 5432, no public endpoint).
      - RDS Automated Backups <<artifact>>.
- Amazon CloudFront – Uploads CDN <<node>>
  - HTTPS <<interface>> for serving user images globally (via OAC to S3).
- Amazon S3 – Uploads Bucket <<node>>
  - User Images (PNG/JPEG/WebP) <<artifact>>, private with Block Public Access ON.
  - Backend writes via IAM; CloudFront reads via OAC.

Runtime Communication Paths

1. **PWA delivery**
   a. Client → CloudFront (Frontend) <<interface:HTTPS>> → S3 Frontend Bucket via OAC.
   b. The PWA is entirely static; API base URL is injected at build time.
2. **User API traffic**
   a. Client → App Runner – Backend API <<interface:REST /api/v1>>.
   b. Requests pass through the Application Layer (routing/validation/auth) and into domain services.
3. **Secrets retrieval**
   a. Backend API → Secrets Manager (GetSecretValue for DATABASE_URL, JWT_SECRET) at boot.
4. **Database access**
   a. Backend API → App Runner VPC Connector → RDS PostgreSQL on 5432/TLS (private).
   b. Prisma performs queries via Repositories in the Persistent Layer.
5. **Media write path (uploads)**
   a. Backend API (after processing) → S3 Uploads Bucket (Put/Delete via IAM).
   b. Stored object keys are returned as CDN URLs to the client.
6. **Media read path (images)**
   a. Client → CloudFront (Uploads) <<interface:HTTPS>> → S3 Uploads via OAC.
   b. Buckets remain private; no public ACLs/policies.
7. **Image processing pipeline**
   a. Backend API → bg-removal POST /remove-bg → color-extract POST /extract-colors → return metadata/bytes → S3 write → persist DB record.
8. **Container images**
   a. App Runner services pull Docker images <<artifact>> from ECR during deployment.
9. **Migrations on start**
   a. Backend container runs prisma migrate deploy at boot against RDS to ensure the live schema matches code (e.g., isTrip column).

Security Controls & Trust Boundaries

- **Transport security**: TLS/HTTPS on all external edges (CloudFront and App Runner).
- **Private data plane:** RDS in private subnets; reachable only via App Runner VPC Connector ENIs.
- **Least-privilege IAM:**
  - Backend role: read specific Secrets; write to the uploads bucket path; pull from ECR.
  - Microservices: only what they require (no DB access).
- **Private S3:** Both buckets have Block Public Access ON; CloudFront OAC is the only read path.
- **AuthN/AuthZ:** JWT for protected routes; RBAC for admin operations.
- **CORS:** allow-list the frontend origin only.
- **Config secrets:** stored in Secrets Manager; non-sensitive config via environment variables.

Availability, Scalability, and Performance

- **Stateless services** (backend & microservices) on **App Runner** support horizontal scale and **rolling deployments** with health checks.
- **CDN** terminates global image traffic, reducing API load and user-perceived latency.
- **RDS** automated backups; Multi-AZ can be enabled when budget permits.
- **Caches** (in-process, bounded TTL) reduce dependency on external weather providers and speed up repeated reads.

Observability & Operations

- App Runner logs for build/deploy/runtime; application logs to stdout/stderr.
- Optional CloudFront access logs to S3.
- Database monitoring via RDS Performance Insights/metrics.
- Alarms on latency/5xx/error budget; synthetic checks canaries during deploys.

CI/CD Touchpoints

- Frontend pipeline (GitHub Actions): build React → sync to S3 Frontend → CloudFront invalidation.
- Backend & microservices: build images → push to ECR → App Runner StartDeployment (or auto-deploy on new image tag).
- Migrations run on container start to keep RDS schema in sync.

Assumptions & Constraints

- RDS is not publicly accessible.
- All S3 buckets are private; access is via CloudFront OAC (reads) and backend IAM (writes).
- Microservices are public endpoints but callable only by the backend (documented; consider token/IP allow-list if needed).
- No message queue yet; the pipeline is synchronous today but the boundary allows adding SQS later without changing public APIs.

Failure Modes & Degradation Paths

- **Image service outage:**
  - backend returns a clear error or degrades to storing the original image; user can retry.
- **Weather provider failure:**
  - backend uses a fallback provider and/or cached summaries.
- **RDS transient errors:**
  - short retries with jitter on read-only paths; write paths remain idempotent to prevent duplication.
- **CDN miss or S3 latency:**
  - served on origin fetch; object then cached at the edge.