 Images are collected (on-board), processed (off-board), and stored as static data.

 Metadata and spatial data are stored in PostgreSQL with PostGIS.

 nginx handles user requests and routes them to:

* Django (for UI and metadata queries)
* TileStache (for map tile rendering)

 External maps are fetched from public tile services.

 GeoKettle handles background ETL jobs to load and transform new spatial datasets.

Work flow

**1. Image Collection**

**A. On-Board Image Collection**

* Field devices (likely mounted inside tunnels) are capturing images in real time.
* These images are either stored locally on the device or temporarily held for transfer.

**B. Off-Board Image Processing**

* Images are moved from field devices to a processing workstation via:
  + **Removable Media** (USB drives, SD cards, etc.)
  + **SFTP Upload** (over a secure network connection)

**🔵 2. Static Image Storage**

* The processed images are stored in **local disk-based storage**.
* This serves as the source for serving tiles or inspection images via the web interface.

**🔵 3. Application Layer**

**A. Web Server: nginx**

* Acts as the **HTTP reverse proxy**.
* It handles incoming requests from TunnelView users over the internet.
* Routes traffic to appropriate internal services (like Django or TileStache).

**B. Application Servers: gunicorn + Django**

* **Gunicorn** is a Python WSGI HTTP server that hosts the web applications.
* Runs the **Django-based TunnelView app** that:
  + Interacts with the backend PostgreSQL/PostGIS database
  + Serves image metadata, map overlays, spatial search, etc.

**🔵 4. Map Tile Server: TileStache**

* A Python-based tile server used to:
  + Serve pre-rendered map tiles
  + Overlay spatial data (from PostGIS or GeoKettle outputs)
* TileStache responds to map tile requests from frontend users.

**🔵 5. Cache Layer: memcached**

* Temporarily holds map data and image metadata in memory.
* Boosts performance by reducing database hits for frequently accessed data.

**🔵 6. Database Layer: PostgreSQL + PostGIS**

* This is the central spatial database powering the system.
* Organized into:
  + **TunnelView Application Data**: User interaction data, configs
  + **Map Data**: Spatial coordinates, features
  + **GeoKettle Metadata Repository**: Stores ETL job metadata

**🔵 7. ETL Tool: GeoKettle**

* A graphical ETL tool specialized for geospatial data.
* Performs:
  + Extraction of spatial datasets from sources like Oracle Spatial (T-MAP CANKERS)
  + Transformation using spatial logic
  + Loading data into the PostGIS database or TileStache

**🔵 8. External Map Sources**

These are third-party map tile providers:

* **OpenStreetMap**
* **USGS Topographic Maps**
* **MapQuest OSM**

They are fetched:

* Dynamically by the web frontend
* Or cached locally for performance

**🔵 9. Users: TunnelView Users (Cloud)**

* These are the **end users** (engineers, inspectors, analysts) accessing TunnelView through a browser.
* They:
  + View map overlays
  + Inspect tunnel images
  + Search spatial features

nginx manages routing of requests to Django for metadata or TileStache for maps.

**PHASE 1: Pre-Migration – Planning & Assessment**

**🔹 1. Review Existing Architecture**

* Understand all current components:
  + Web (nginx, Django/Gunicorn)
  + TileStache
  + PostgreSQL/PostGIS
  + Memcached
  + GeoKettle
  + File-based image processing and storage

**🔹 2. Prerequisites / Admin Setup**

| **Requirement** | **Purpose** |
| --- | --- |
| Azure Subscription | Needed to deploy all services |
| Resource Group | Logical container for resources |
| Azure AD Tenant | For managing users and identities |
| Network Planning (VNet, NSG) | Define security and connectivity model |
| Region Selection | Based on user base and data residency |

**🔹 3. Tools & Accounts Setup**

* Install **Azure CLI / Azure PowerShell**
* Set up **Azure DevOps / GitHub Actions** (optional for CI/CD)
* Setup **Storage Explorer** for Blob access
* Get access to **Azure Portal** and relevant RBAC roles:
  + Owner/Contributor
  + Storage Blob Data Contributor
  + PostgreSQL DB Contributor

**✅ PHASE 2: Core Azure Services Setup**

**🔹 4. Setup Storage**

* **Azure Blob Storage**
  + Create containers: /images, /tiles, /uploads
  + Configure lifecycle policies (optional)
  + Use Blob replication if cross-region needed

**🔹 5. Setup PostgreSQL**

* **Azure Database for PostgreSQL – Flexible Server**
  + Enable **PostGIS extension**
  + Import your existing PostgreSQL schema & data

**🔹 6. Deploy Web App**

* Choose one:
  + **Azure App Service (Linux)** – for simple deployment of Django + Gunicorn
  + **Azure Kubernetes Service (AKS)** – for scalable microservices (if using containers)
* Create CI/CD pipeline to deploy Django source code

**🔹 7. Reverse Proxy / Traffic Manager**

* Use **Azure Front Door** or **Azure Application Gateway**
  + For SSL termination, routing, load balancing

**🔹 8. Replace Memcached**

* Use **Azure Cache for Redis**
  + Enable Geo-replication if needed
  + Update app code to use Redis Python client

**✅ PHASE 3: ETL, Image Processing, and External Integration**

**🔹 9. Replace GeoKettle**

* Use **Azure Data Factory** for ETL pipelines
  + Build pipelines that pull spatial data from external systems
  + Use **Azure Functions / Python / FME** for custom spatial transformations

**🔹 10. Image Upload Pipeline**

* **IoT Route**: Use **Azure IoT Hub + IoT Edge**
* **Offline Upload**: Use **Storage Explorer** or **SFTP to Blob**
* Trigger **Azure Functions / Azure Batch** for post-upload image processing

**🔹 11. TileStache Containerization**

* Containerize **TileStache** using Docker
* Deploy via:
  + **Azure App Service for Containers**
  + OR **Azure Container Instances**
  + OR run it inside **AKS**

**✅ PHASE 4: Security, Access & Monitoring**

**🔹 12. Identity & Access**

* Integrate **Azure Active Directory** with your Django app
  + Use MSAL / OAuth2 for auth
  + Enforce RBAC in Azure for resource access

**🔹 13. Logging & Monitoring**

* Enable:
  + **Azure Monitor**
  + **Log Analytics Workspace**
  + **App Insights** for Django + Function Apps

**🔹 14. Backup & Disaster Recovery**

* **Enable automatic backups** for PostgreSQL
* Set up **geo-redundant storage (GRS)** for Blob
* Use **Azure Site Recovery** (optional for full failover)

**✅ PHASE 5: Testing & Go-Live**

**🔹 15. Smoke Testing**

* Validate:
  + Web app loading tiles from Blob
  + Queries to PostgreSQL PostGIS
  + ETL flows from Data Factory
  + Authentication with Azure AD

**🔹 16. User Acceptance Testing**

* Have real Tunnel View users interact with the system
* Validate map rendering, image loads, queries, etc.

**🔹 17. Go-Live & Optimization**

* Cutover DNS to Azure Front Door
* Monitor usage and tune:
  + Redis cache TTL
  + Blob access tiers (Hot/Cool/Archive)
  + PostgreSQL performance tiers

**💡 OPTIONAL ENHANCEMENTS**

| **Feature** | **Azure Service** |
| --- | --- |
| Auto-scaling | App Service / AKS HPA |
| CDN for maps/tiles | Azure CDN (edge caching) |
| Security Scanning | Microsoft Defender for Cloud |
| Geo-fencing | Azure Maps APIs + IP Restrictions |

**Checklist: What to Discuss with the Application Team Before Azure Migration**

**🔹 1. Understand the Current System Behavior**

Ask the team to **demonstrate the current application end-to-end**, including:

* 🔍 How users log in and interact
* 📷 How images are uploaded and processed
* 🗺️ How maps/tiles are generated and served
* 🗃️ How data is queried and where it’s stored (DB, filesystem, etc.)

**Key Question Examples:**

* Can you walk me through the full workflow from image ingestion to web user access?
* How is each component currently connected (web → app → DB → file system)?
* What manual steps are involved in current image upload or processing?

**🔹 2. Gather Infrastructure & Capacity Metrics**

You need to right-size Azure resources based on **existing usage**.

Ask for:

* 📈 Current CPU/RAM/disk usage for:
  + Web/App Servers (nginx, Django)
  + DB Server (PostgreSQL + PostGIS)
  + TileStache Server
* 📊 Peak traffic and concurrent users
* 🧠 Memory/cache usage by Memcached
* 🖼️ Image storage growth per month (in GB/TB)

**Key Question Examples:**

* What are the server specs (CPU/RAM) for each component?
* How many users access the system daily/weekly?
* How much image data do you store annually?
* What’s the largest job you process (in terms of image size or tiles)?

**🔹 3. Understand Dependencies & Integration**

Understand every external/internal system the application connects to.

Ask for:

* 🔗 Data sources used by GeoKettle (e.g., Oracle Spatial, APIs)
* 🔄 Any internal or external APIs called
* 🔐 Any third-party services requiring authentication or licensing
* 🌐 Map tile providers (e.g., OpenStreetMap, MapQuest)

**Key Question Examples:**

* What external map services or data sources are currently used?
* Do any components rely on a licensed third-party tool?
* Are any systems calling Tunnel View APIs?

**🔹 4. Review Application Code & Deployment**

Understand how the application is currently managed:

Ask for:

* 🛠️ Source code access (GitHub, GitLab, etc.)
* ⚙️ Deployment method (manual, CI/CD?)
* 🔄 Frequency of code changes or feature updates

**Key Question Examples:**

* How is the current deployment done — scripted, CI/CD, or manual?
* Are you using any containerized environments?
* Is the code Python 3.x compatible for Azure App Service or Functions?

**🔹 5. Confirm Software Stack Versions**

Knowing software versions helps determine Azure compatibility.

Ask for versions of:

* Django
* Gunicorn
* TileStache
* PostgreSQL/PostGIS
* GeoKettle
* Memcached

**Key Question Examples:**

* Are all software components updated and supported?
* Any legacy libraries or packages that might break during migration?

**🔹 6. Authentication & Access Model**

Azure AD integration and IAM are core to the new environment.

Ask for:

* 🔐 How users authenticate now
* 🛂 Roles and permissions defined within the app
* 🎯 Any need for SSO, multi-tenancy, or conditional access

**Key Question Examples:**

* How is user authentication handled today?
* Is there a requirement for integrating with enterprise SSO (e.g., Azure AD)?
* What user roles exist (admin, viewer, editor, etc.)?

**🔹 7. Backup, Logging & DR Requirements**

You must align backup and monitoring needs with Azure services.

Ask for:

* 🕒 Current backup schedule for DB, image data
* 📜 Logging system used (if any)
* 🌪️ Disaster recovery expectations

**Key Question Examples:**

* How often are backups taken and where are they stored?
* Do you need log retention for audit or compliance?
* What’s your acceptable downtime or RTO/RPO?

**🔹 8. Expected Migration Timeline & Environment Staging**

Plan the actual migration with clarity.

Ask:

* 🧪 Whether staging/UAT environments exist
* 📆 Target dates for production cutover
* 🚦 Readiness for a parallel run if needed

**Key Question Examples:**

* Do you have dev/staging/prod environments? Do we migrate all?
* When is the ideal time window for the go-live?
* Do you want a rollback plan or phased migration?

**🔹 9. Azure Resource Access During Migration**

Ask for:

* 🔐 Whether app team needs access to Azure Portal, Logs, or DB
* 👥 Who should be added as contributors/viewers?

**🔹 10. Prepare a RACI Matrix**

* Define **who is Responsible**, **Accountable**, **Consulted**, and **Informed** for each migration task:
  + Data migration
  + App deployment
  + Testing
  + Monitoring
  + Authentication

**Meeting Agenda: Application Team Discovery for Azure Migration**

**Objective:**

To gather complete technical and operational context of the existing Tunnel View system before provisioning cloud resources in Azure.

**Agenda Items:**

1. **Introduction & Migration Goals**
   * Define objectives of Azure migration
   * Set scope (full or phased migration)
2. **Current System Walkthrough**
   * End-to-end demo: from image capture to user interaction
   * Show how data moves between components
3. **Resource Utilization & Capacity**
   * Discuss current CPU, RAM, storage, DB size, and network usage
   * Understand peak usage and growth trends
4. **Application Dependencies & Integrations**
   * Identify internal/external APIs, map sources, and ETL systems
   * Confirm any licensed or proprietary data sources
5. **Code & Deployment Overview**
   * Deployment process, CI/CD, source code repository access
   * Version control systems and branching strategies
6. **Authentication & User Access**
   * Current auth mechanisms and role-based access
   * Evaluate feasibility of Azure AD or federated login integration
7. **Monitoring, Logging, and Backup Practices**
   * Current tools, frequency, formats, and storage locations
   * Expectations for Azure-native replacements (e.g., Monitor, Log Analytics)
8. **Environment Layout and Migration Readiness**
   * Dev/stage/prod setup, freeze periods, test plans
   * Compatibility concerns for existing environments on Azure
9. **Resource Access During and After Migration**
   * Identify app team members who need Azure access
   * Define RBAC roles and access boundaries
10. **Next Steps and Action Items**

* Review open questions
* Assign owners for migration blockers and dependencies

**Questionnaire for Application Team**

**System Workflow**

* Can you walk through the user flow from image ingestion to viewing?
* What steps are automated vs manual in the workflow?
* Which components are mission-critical and time-sensitive?

**Infrastructure**

* What are the specs of your current app and DB servers?
* What is your current average and peak traffic load?
* How much data (image/tiles) is processed or uploaded monthly?
* Do you have monitoring metrics available (CPU/RAM/disk/network)?

**Software Stack**

* What versions of Django, Gunicorn, PostgreSQL/PostGIS, TileStache, Memcached are you using?
* Are there legacy libraries or unsupported tools?
* Are these components under active maintenance?

**ETL / Data Integration**

* What data sources does GeoKettle interact with?
* What transformations are applied to spatial data?
* Do you use FME or other spatial tools?

**Image Handling**

* How are images uploaded? Real-time or batch?
* Where are they stored, and how is metadata managed?
* Are image formats standardized?

**Auth & Access**

* How are users authenticated? Any SSO?
* What user roles are defined?
* Do you plan to integrate with Azure AD?

**Backups & Logs**

* What is your current DB and file backup policy?
* Where are logs stored? How long are they retained?
* Are logs structured or flat files?

**Environments & Deployment**

* Do you have dev/test/prod environments?
* How often is code deployed?
* Are blue-green or canary deployments in use?

**Monitoring**

* What monitoring and alerting tools are used now?
* What metrics are considered critical?

**Migration Preferences**

* Preferred downtime window for go-live?
* Can we run in parallel before switch-over?
* Are you comfortable with phased migration (e.g., DB first, app later)?