Architectural Analysis: Multi-Layered Software Stack for High-End Residential Automation and Enterprise Monitoring

1. Executive Summary

This deep-dive architectural analysis and technical feasibility study presents a comprehensive design for a multi-layered software stack tailored for high-end residential automation. The proposed system is designed to bridge the gap between robust, localized control systems (KNX/Zigbee) and scalable, enterprise-grade remote monitoring. The target environment is a hybrid topology where an "AI Edge Node"—specifically an HP EliteDesk 800 G5 running Proxmox VE—maintains autonomous operation at the client site, while telemetric data is aggregated into a centralized "Diagnostic Home Lab" for multi-tenant monitoring by the systems integrator.

The research is conducted within the development context of **Google Antigravity**, an agentic Integrated Development Environment (IDE) that fundamentally shifts the developer's role from writing syntax to orchestrating autonomous agents. Consequently, the output of this report is structured not merely as theoretical analysis but as actionable "Artifacts"—code scaffolds, configuration schemas, and deployment strategies—optimized for an agentic workflow.

The architecture addresses four critical components: a Python-based Home Assistant (HA) Add-on for backend health monitoring, a Flutter-based client interface capable of real-time AI vision overlays, a scalable InfluxDB v2 diagnostic layer utilizing the Flux functional data scripting language, and an Ansible-driven "Zero-Touch" provisioning toolkit.

Key findings from the feasibility study indicate that while the "no-client VPN" access strategy utilizing Cloudflare Tunnels offers superior User Experience (UX) and reduced support overhead compared to traditional VPNs, it necessitates a rigorous application-layer security posture, including Mutual TLS (mTLS) and granular Access Policies, to mitigate the risks associated with removing network-layer authentication.[1] Furthermore, the analysis of the "Hybrid Link" between KNX and Zigbee protocols reveals that latency optimization requires a departure from standard polling intervals in favor of event-driven, push-based state management within the Home Assistant event loop.[3]

---

2. Architectural Overview and Hardware Topology

The proposed system utilizes a split-topology design that rigorously decouples local control availability from remote diagnostic visibility. This "Edge-Local / Cloud-Diagnostic" split ensures that a catastrophic failure in internet connectivity, the cloud broker, or the central monitoring stack does not impact the resident's ability to control their physical environment.

2.1. Client-Side: The AI Edge Node

The hardware standard selected for the client-side deployment is the **HP EliteDesk 800 G5** (Small Form Factor or Mini), utilized as a bare-metal hypervisor host running **Proxmox Virtual Environment (VE)**. This selection represents a strategic deviation from the ARM-based Single Board Computers (SBCs) commonly found in hobbyist setups, such as the Raspberry Pi.

2.1.1. Compute Architecture and Virtualization Strategy

The HP EliteDesk 800 G5 typically features an Intel Core i5 or i7 processor (9th Gen). The architecture leverages this x86-64 compute power to support virtualization and heavy I/O operations that would saturate an SBC bus.

- **Proxmox VE (Hypervisor):** Proxmox is chosen for its ability to manage both Linux Containers (LXC) and Kernel-based Virtual Machines (KVM) with a unified web interface and API. This facilitates the segregation of duties.[5]
- **Home Assistant (LXC):** The automation core runs in a privileged LXC container. This is preferred over a VM to minimize overhead and allow direct access to the host's network stack, which is crucial for handling KNX multicast traffic (224.0.23.12:3671) without complex bridging or NAT traversal issues often encountered in VM networking.[4]
- **Frigate NVR (Docker in LXC/VM):** Computer vision is the most resource-intensive workload. Frigate is deployed in a Docker container (either within a nested LXC or a separate VM) with a **Google Coral Edge TPU** passed through via PCIe or USB. The Intel processor's integrated GPU (iGPU) is utilized for hardware-accelerated video decoding (Intel Quick Sync), freeing the CPU for logic processing. This architecture allows the Coral TPU to handle object detection inference at <10ms latency.[6]
- **OpenMediaVault (OMV):** OMV serves as the Network Attached Storage (NAS) layer, handling the retention of high-definition video footage. By virtualizing the NAS on the same hardware but in a separate VM, we ensure that disk I/O wait times during heavy recording do not block the Home Assistant main loop.

2.2. Connectivity Layer: The Hybrid Mesh

The connectivity layer is designed to be protocol-agnostic at the automation level, utilizing **MQTT** as the universal message bus.

- **KNX/IP Backbone:** KNX serves as the primary layer for critical infrastructure (lighting, HVAC, blinds) due to its decentralized reliability. The connection to Home Assistant is

established via a KNX/IP Router supporting multicast, or a Tunneling Interface if multicast is blocked by network equipment. The integration allows HA to act as a KNX device, reading group addresses and sending telegrams directly to the bus.[4]

- **Zigbee Retrofit Layer:** Zigbee is used for sensors and retrofit devices where running KNX bus cable is impractical. Devices are bridged via **Zigbee2MQTT** rather than the native ZHA integration. This architectural choice decouples the Zigbee mesh from Home Assistant; if HA restarts, the Zigbee mesh remains active, and messages are queued on the MQTT broker.[8]
- **Remote Telemetry (HiveMQ Cloud):** To bridge the local Edge Node with the central Diagnostic Lab, a secure MQTT bridge is established to a managed HiveMQ Cloud cluster. This ensures that telemetry data leaves the site via an outbound TLS connection, requiring no inbound firewall ports.

2.3. Installer-Side: The Diagnostic Home Lab

The integrator maintains a centralized infrastructure, referred to here as the "Diagnostic Home Lab," which acts as a multi-tenant monitoring center.

- **TIG Stack:** The core technology stack consists of **Telegraf** (collection), **InfluxDB v2** (storage), and **Grafana** (visualization).
- **InfluxDB v2:** The migration to version 2 is critical for this architecture to leverage **Flux**, a functional scripting language that allows for complex, server-side data transformation and alerting logic that was not possible with InfluxQL.[9] This database serves as the "source of truth" for the health of the entire fleet of deployed Edge Nodes.

---

3. Component 1: Home Assistant Add-on (Backend Architecture)

The first research task involves designing a Python-based backend service running as a Home Assistant Add-on. The primary objective of this component is to handle "MQTT Statestream" ingestion to monitor the health of KNX heartbeats and hybrid entities.

3.1. Design Pattern: The "Sidecar" Health Monitor

A critical design principle in high-reliability systems is that a system cannot effectively monitor its own total failure. If Home Assistant freezes, its internal automations cannot send an alert. Therefore, this component is architected as a **Sidecar Service**—a standalone process running in a separate container (Add-on) that shares the network namespace but has an independent lifecycle.

3.1.1. MQTT Statestream Integration

The component relies on the **MQTT Statestream** integration in Home Assistant. This integration publishes every state change of selected entities to the MQTT bus under a hierarchy like homeassistant/statestream/entity_id/state.[11]

- **Decoupling:** By listening to the MQTT bus rather than querying the HA API, the Sidecar Monitor operates asynchronously. It subscribes to the stream of events. If the stream falls silent (no heartbeats), the monitor detects the anomaly.
- **Configuration:**
  YAML
  ```
  # Home Assistant configuration.yaml
  mqtt_statestream:
    base_topic: homeassistant
    publish_attributes: true
    publish_timestamps: true
    include:
      entities:
        - binary_sensor.knx_system_heartbeat
        - sensor.zigbee_coordinator_availability
  ```

3.2. Python Backend Implementation

The backend logic is implemented in Python, utilizing the asyncio library for non-blocking I/O and paho-mqtt for bus communication.

3.2.1. The Heartbeat Algorithm

KNX systems often include a "cyclical send" mechanism where a device sends a telegram (e.g., True) every 60 seconds to prove it is alive. The Python script implements a "Deadman Switch" logic for these entities.

1. **Subscription:** The script subscribes to homeassistant/binary_sensor/knx_heartbeat_*/state.
2. **State Management:** It maintains an in-memory dictionary mapping entity_id to last_seen_timestamp.
3. **The Watchdog Loop:** A background coroutine wakes up periodically (e.g., every 10 seconds) to check if (current_time - last_seen_timestamp) > threshold.
4. **Alerting:** If a timeout occurs, the script publishes a JSON payload to a dedicated diagnostic topic diagnostics/{site_id}/alerts.[12]

3.2.2. S6 Overlay and Process Supervision

Home Assistant Add-ons utilize the **S6 Overlay** for process supervision. This is a critical requirement for "Enterprise Grade" reliability: if the Python script crashes, S6 automatically

restarts it.

- **Constraint:** S6 Overlay v3 requires specific configuration in the Docker container, notably setting init: false in the Add-on configuration to prevent conflicts with the Docker daemon's init process.[14]

Antigravity Agent Task:

> "Generate the file structure for a Home Assistant Add-on including the Dockerfile, config.yaml, and the S6 service definition files (run, finish) for a Python script named monitor.py."

S6 Service Run Script (rootfs/etc/services.d/monitor/run):

Bash

```
#!/usr/bin/with-contenv bashio
# Standard S6 run script for Python service
# bashio handles logging and config parsing

echo "Starting MQTT Health Monitor Sidecar..."

# Extract MQTT credentials from HA's internal service discovery
export MQTT_HOST=$(bashio::services mqtt "host")
export MQTT_USER=$(bashio::services mqtt "username")
export MQTT_PASSWORD=$(bashio::services mqtt "password")
export MQTT_PORT=$(bashio::services mqtt "port")

# Retrieve Site ID from Add-on Options
export SITE_ID=$(bashio::config "site_id")

# Execute Python in unbuffered mode to ensure logs appear immediately
exec python3 -u /usr/bin/monitor.py
```

3.3. JSON Payload Structure for Diagnostics

To support the multi-tenant requirement of the Installer-Side Lab, the data emitted by this Add-on must follow a strict schema. This allows the centralized InfluxDB to index data correctly.

Proposed Telemetry Schema:

JSON

```
{
  "site_id": "residence_alpha_01",
  "timestamp": "2023-10-27T10:00:00.123Z",
  "message_type": "heartbeat_failure",
  "data": {
    "entity_id": "binary_sensor.knx_actuator_heating",
    "protocol": "knx",
    "last_seen_seconds_ago": 125,
    "threshold_seconds": 60,
    "status": "offline"
  },
  "diagnostics": {
    "ha_uptime": 45000,
    "mqtt_connection": "connected",
    "memory_usage_mb": 150
  }
}
```

- **Rationale:** The site_id is top-level for easy routing. The data block contains the specific failure context. The diagnostics block provides environmental context (e.g., did the device fail, or is HA itself running out of memory?).[16]

---

4. Component 2: Flutter-Based Client Interface

The user interface for high-end residential clients demands performance characteristics—specifically launch speed and interaction latency—that generic web-based dashboards (like the standard Home Assistant frontend) cannot consistently provide. The requirement is a bespoke **Flutter** application compiled for iOS and Android, focusing on "Computer Vision" overlays and instant security panel response.

4.1. WebSocket Architecture for Low Latency

Unlike REST APIs, which require a new TCP handshake and header exchange for every request (polling), WebSockets allow Home Assistant to push state changes to the Flutter client

immediately over a persistent connection.

4.1.1. Authentication and Connection Handshake

The Flutter application must implement the Home Assistant WebSocket Authentication API.

1. **Connection:** The client connects to wss://<ha_url>/api/websocket.
2. **Auth Phase:** HA sends {"type": "auth_required"}. The client responds with {"type": "auth", "access_token": "..."}.
3. **Command Phase:** Upon receiving {"type": "auth_ok"}, the client can subscribe to events.
    - **Subscription:** {"id": 1, "type": "subscribe_events", "event_type": "state_changed"}.

4.1.2. State Management: BLoC Pattern

For this architecture, the **BLoC (Business Logic Component)** pattern is recommended over Provider. BLoC separates the UI from the business logic via streams, which aligns perfectly with the WebSocket stream of events. A WebSocketBloc listens to the incoming JSON stream, parses it into Dart objects (e.g., EntityState), and yields new states to the UI. This ensures that only the specific widget listening to "Living Room Light" rebuilds when that specific state changes, optimizing rendering performance.[18]

4.2. Frigate Integration: Person-Detection Overlays

A key feature of the "AI Edge Node" is the ability to visualize what the AI sees. Frigate streams video via RTSP/WebRTC but publishes object detection metadata (bounding boxes) via MQTT. Ideally, the Flutter app overlays these boxes on the live video stream.

4.2.1. The Coordinate Mapping Challenge

Frigate's MQTT topic frigate/events publishes a JSON payload containing the bounding box coordinates.

- **Payload Format:** The box is typically defined as [x_min, y_min, x_max, y_max] (or top/left/bottom/right depending on version), representing coordinates in the source video resolution.[21]
- **The Problem:** The mobile device screen has a different aspect ratio and pixel density than the camera's source stream (e.g., 1920x1080 source vs. 390x844 iPhone screen).
- **The Solution:** The Flutter app must implement a coordinate normalization and scaling logic within a CustomPainter.

4.2.2. Flutter Implementation: CustomPainter

The CustomPainter class in Flutter allows for low-level drawing on the canvas. The implementation involves:

1. **Ingestion:** Listening to the MQTT stream for frigate/events.

2. **Normalization:** Converting the source coordinates to a percentage (0.0 to 1.0) based on the known camera resolution.
    - $x_{norm} = x_{source} / width_{source}$
    - $y_{norm} = y_{source} / height_{source}$
3. **Projection:** Scaling the normalized coordinates to the current size of the video player widget.
    - $x_{canvas} = x_{norm} \times width_{widget}$
    - $y_{canvas} = y_{norm} \times height_{widget}$

Code Scaffold (frigate_overlay.dart):

Dart

```
import 'package:flutter/material.dart';

class DetectionBoxPainter extends CustomPainter {
  // Bounding boxes from MQTT: [y_min, x_min, y_max, x_max] (Example format)
  final List<List<int>> boxes;
  final Size sourceResolution; // e.g., 1920x1080

  DetectionBoxPainter({required this.boxes, required this.sourceResolution});

  @override
  void paint(Canvas canvas, Size size) {
    if (boxes.isEmpty) return;

    final paint = Paint()
      ..color = Colors.redAccent
      ..style = PaintingStyle.stroke
      ..strokeWidth = 3.0;

    // Calculate scale factors
    double scaleX = size.width / sourceResolution.width;
    double scaleY = size.height / sourceResolution.height;
```

```
    for (var box in boxes) {
      // Map coordinates to the canvas size
      // Assuming format: [top, left, bottom, right]
      Rect rect = Rect.fromLTRB(
        box * scaleX, // left
        box * scaleY, // top
        box * scaleX, // right
        box * scaleY  // bottom
      );

      canvas.drawRect(rect, paint);

      // Draw Label
      TextSpan span = TextSpan(style: TextStyle(color: Colors.white, backgroundColor:
Colors.red), text: "PERSON");
      TextPainter tp = TextPainter(text: span, textDirection: TextDirection.ltr);
      tp.layout();
      tp.paint(canvas, Offset(rect.left, rect.top - 20));
    }
  }

  @override
  bool shouldRepaint(DetectionBoxPainter oldDelegate) => true;
}
```

- **Insight:** This CustomPainter should be wrapped in a RepaintBoundary to prevent the video player (which repaints every frame) from triggering unnecessary rebuilds of the overlay layer, or vice versa.[22]

4.3. Low-Latency Security Panel

For security functions (e.g., disarming an alarm), perceived latency is critical.

- **Optimistic UI:** The App should update the UI state to "Disarmed" immediately upon the user's tap, without waiting for the server response. If the WebSocket command fails (timeout or error), the UI reverts to "Armed" and displays an error toast.
- **Direct Socket Command:** Bypass any HTTP REST endpoints. Send the call_service command directly through the active WebSocket connection. This avoids the overhead of establishing a new HTTPS connection and performing a TLS handshake, saving 200-500ms on mobile networks.[24]

---

5. Component 3: Multi-Tenant Diagnostics (InfluxDB v2)

The centralized "Diagnostic Home Lab" faces the challenge of monitoring hundreds of distinct sites (tenants) while maintaining data isolation and query performance. The shift to **InfluxDB v2** introduces **Flux**, which is pivotal for this architecture.

5.1. Multi-Tenancy Strategy: Tags vs. Buckets

There are two primary approaches to multi-tenancy in time-series databases:

1. **Bucket-per-Tenant:** Each site gets its own bucket. This offers perfect isolation and distinct retention policies but makes "global" queries (e.g., "How many sites are offline?") extremely difficult, requiring complex join() operations across hundreds of buckets.[25]
2. **Tag-Based Segregation:** All data flows into a single global_telemetry bucket, with a site_id tag distinguishing the tenants.

Recommendation: For this fleet monitoring use case, **Tag-Based Segregation** is the superior architecture.

- **Reasoning:** The primary goal is *fleet monitoring*. Flux queries are highly optimized for filtering by tags. A single query can aggregate health stats across all customers. Access policies (Tokens) in InfluxDB v2 can still restrict a specific read-token to data containing a specific tag, ensuring security is not compromised.[26]

Schema Design:

- **Bucket:** telemetry_prod
- **Measurement:** heartbeats
- **Tags:** site_id (UUID), region (e.g., us-east), plan_tier (gold/silver).
- **Fields:** status (int: 0=OK, 1=Error), latency_ms (float), cpu_load (float).

5.2. "Site Down" Alerts: The Flux Deadman Check

Detecting when a site goes offline is a "Deadman" problem—the absence of data indicates failure. In Flux, the monitor.deadman() function is designed specifically for this.

Flux Query for Global Site-Down Alert:

This query groups data by site_id and triggers an alert if any specific site has reported zero records in the last 5 minutes.

Code snippet

```
import "influxdata/influxdb/monitor"
import "influxdata/influxdb/schema"

// Define the data stream from the global bucket
data = from(bucket: "telemetry_prod")

|> range(start: -10m)
|> filter(fn: (r) => r._measurement == "heartbeat")
|> group(columns: ["site_id"]) // Crucial: Groups stream by site

// Define the alert trigger
option task = {name: "Global Site Down Monitor", every: 1m}

check = {
  _check_id: "site_down_check",
  _check_name: "Site Down Alert",
  _type: "deadman",
  tags: {},
}

// Deadman function: Triggers if no data received for 5 minutes
data

|> monitor.deadman(
  t: experimental.subDuration(d: 5m, from: now()),
  fn: (r) => r._level == "crit",
  messageFn: (r) => "CRITICAL: Site ${r.site_id} is offline. No telemetry for 5 mins."
 )
```

- **Insight:** By grouping by site_id *before* calling monitor.deadman, InfluxDB creates a separate stream for each site. The deadman check monitors each stream independently. This means a single Flux task can monitor 1,000 sites dynamically; as soon as a new site_id appears in the ingest stream, it is automatically monitored.[27]

5.3. Grafana Dashboard Logic

The Grafana dashboard utilizes **Variables** to allow the integrator to filter views without editing queries.

- **Variable $site:** Defined by the query import "influxdata/influxdb/schema" schema.tagValues(bucket: "telemetry_prod", tag: "site_id").
- **Row Repeater:** A "Site Status" row in Grafana can be configured to **Repeat** for the variable $site. This automatically generates a status panel for every client in the database, creating a "Mission Control" view that scales automatically as new customers are onboarded.[28]

---

6. Component 4: Integrator Toolkit (Automation)

To achieve the "Zero-Touch" provisioning requirement, manual configuration of Proxmox and Home Assistant must be eliminated. The toolkit relies on **Ansible** for infrastructure orchestration and **Terraform** for cloud resource management.

6.1. Ansible-Driven Proxmox Provisioning

The community.general.proxmox_lxc Ansible collection is used to orchestrate the creation of containers on the Edge Node.

6.1.1. Automating Cloudflare Tunnels

Automating Cloudflare Tunnels is complex because the standard cloudflared login command requires interactive browser authentication. To bypass this for automated provisioning, we must use the **Cloudflare API** to generate credentials beforehand.

Workflow:

1. **Terraform (Cloud):** The integrator runs a Terraform script that interacts with the Cloudflare API. It creates a new Tunnel object and exports the tunnel_token to a secure file (or Ansible Vault).
2. **Ansible (Edge):**
   - Connects to the Proxmox Host.
   - Creates a new LXC container for the tunnel.
   - Injects the tunnel_token directly into the container's systemd service configuration.
   - Starts the service.

Actionable Ansible Snippet (Role: deploy_tunnel):

YAML

```
- name: Create Cloudflared LXC Container
  community.general.proxmox_lxc:
    node: "{{ proxmox_node }}"
    vmid: 200
    hostname: "cloudflared-gateway"
    ostemplate: "local:vztmpl/debian-12-standard.tar.zst"
    netif: '{"net0":"name=eth0,ip=dhcp,bridge=vmbr0"}'
    state: started

- name: Install Cloudflared Binary
  ansible.builtin.command:
    cmd: "lxc-attach -n 200 -- curl -L -o /usr/bin/cloudflared
https://github.com/cloudflare/cloudflared/releases/latest/download/cloudflared-linux-amd64"

- name: Inject Token and Install Service
  ansible.builtin.command:
    cmd: "lxc-attach -n 200 -- cloudflared service install {{ cloudflare_tunnel_token }}"
  # The token is passed from the Terraform output
```

- **Insight:** This bypasses the need for cloudflared login. The token authorizes the tunnel to connect to the specific Cloudflare account and route traffic to the specific DNS hostname defined in the Terraform state.[30]

6.2. ETS to YAML Scaffolding

KNX programming is traditionally done in ETS (Engineering Tool Software). Manually re-entering hundreds of Group Addresses (GAs) into Home Assistant is error-prone.

- **Tool:** ets-to-homeassistant (a Ruby-based CLI tool) is identified as the optimal solution.[31]
- **Integration Strategy:**
  1. Integrator exports the project from ETS6 as a .knxproj file.
  2. **Antigravity Agent Task:** An agent is scripted to watch the project repository. When a .knxproj file is committed, the agent automatically executes: ets_to_hass --format homeass --output knx_config.yaml project.knxproj
  3. The agent then generates a Pull Request with the updated knx_config.yaml, scaffolding all new lights, switches, and sensors defined in the ETS project. This reduces configuration time from hours to seconds.

---

7. Critical Analysis: Security & Optimization

7.1. Security Implications of "No-Client VPN" (Cloudflare Tunnel)

The user query specifically requests an analysis of the "no-client VPN" model using Cloudflare Tunnels.

Assessment:

- **Convenience:** The model offers high convenience. It works behind CGNAT (Starlink/LTE) and requires no client software configuration on user devices.
- **Security Risk:** Unlike a traditional VPN (WireGuard) where the encrypted tunnel end-to-end terminates on the user's device, Cloudflare Tunnel terminates TLS at the Cloudflare Edge. This technically introduces a "Man-in-the-Middle" (MITM) risk where Cloudflare (the provider) has visibility into the traffic. Furthermore, the public endpoint is technically exposed to the internet, protected only by the authentication layer.

Hardening Strategy (Enterprise Grade):

To elevate this model to an acceptable security standard for high-end residential clients, **Mutual TLS (mTLS)** must be enforced.

1. **Private CA:** The integrator generates a private Certificate Authority (CA) within Cloudflare Zero Trust.
2. **Client Certificates:** A unique client certificate is generated for each user device (iPhone/Android).
3. **Access Policy:** A rigorous Cloudflare Access Policy is configured to **BLOCK** any request that does not present a valid client certificate signed by the private CA.
4. **Result:** Even if an attacker knows the URL and steals a username/password, they cannot access the login page without the cryptographic certificate installed on the device. This restores the "Device Identity" factor present in VPNs.[2]

7.2. Optimizing "Hybrid Link" Logic (KNX-to-Zigbee Latency)

A common performance bottleneck in hybrid stacks is the latency "bridge delay" when a KNX wall switch is pressed to toggle a Zigbee light bulb.

- **Root Cause:** This is often caused by Home Assistant *polling* the KNX bus (introducing a delay up to the scan_interval) or congestion in the Zigbee mesh.
- **Event Loop Optimization:**
  1. **Push, Don't Poll:** The KNX integration must be configured to rely entirely on knx_event (multicast push) rather than polling state addresses.
  2. **Native Exposure (knx: expose):** The knx: expose feature in Home Assistant is the critical optimization. It allows HA to bypass the standard automation engine triggers. Instead, it directly links a Home Assistant entity state to a KNX Group Address at the integration driver level.

```yaml
# configuration.yaml optimization
knx:
  expose:
    - type: binary
      address: "1/1/10" # KNX Status GA
      entity_id: light.zigbee_bulb_01
```

This ensures that as soon as the Zigbee bulb state changes, the KNX bus is updated immediately, and vice versa, with minimal processing overhead. 3. **Zigbee Grouping:** To reduce "popcorn effect" (latency where lights turn on one by one), Zigbee bulbs must be bound to a Zigbee Group. Home Assistant should send a single MQTT command to the Group ID (zigbee2mqtt/group_living_room/set), reducing the traffic on the Zigbee mesh by a factor of N (where N is the number of bulbs).[8]

---

8. Development Context: Antigravity Agentic Workflows

Development in **Google Antigravity** transforms the integrator from a "coder" to an "architect." The workflow involves defining high-level missions for the AI agents.

8.1. Sample Agent Mission: "Scaffold Diagnostics"

Instead of writing the Flux query manually, the integrator issues a prompt to the Antigravity Agent Manager:

> **Agent Mission:** "Analyze the knx_config.yaml file. Identify all binary sensors that represent security contacts. Generate a Flux query that monitors these specific entities for a 'deadman' state (no heartbeat for 2 hours) and format it as a JSON model for a Grafana Alerting panel."

8.2. Verification Artifacts

Antigravity agents generate **Artifacts**—tangible deliverables like a screenshot of the rendered Grafana dashboard or a dry-run log of the Ansible playbook.[34] The integrator reviews these artifacts to verify the logic. For example, the agent might produce a "Implementation Plan" artifact outlining the exact MQTT topics it intends to monitor before writing any Python code. This allows the integrator to catch schema errors (e.g., diagnostics/site_id vs site_id/diagnostics) before deployment.

---

9. Conclusion

This architectural analysis confirms the technical feasibility of the proposed stack. By leveraging the **HP EliteDesk G5** with **Proxmox** for robust local virtualization, the system achieves the necessary compute power for edge AI. The **Cloudflare Tunnel** architecture, when hardened with **mTLS**, provides a secure and user-friendly remote access solution that eliminates the support burden of traditional VPNs.

The "Installer-Side" requirement is met through a **Tag-Based InfluxDB v2** architecture, enabling scalable fleet monitoring via **Flux Deadman checks**. Finally, the complexity of configuration is managed through the **Antigravity** development context, where **Ansible** playbooks and **Flutter** UI scaffolding are generated by autonomous agents, transforming the systems integrator's role from manual configuration to high-level architectural orchestration.

10. Actionable Artifacts Summary

1. **HA Add-on:** monitor.py utilizing paho-mqtt and S6 overlay for watchdog monitoring.
2. **Flutter App:** BLoC-based WebSocket client with CustomPainter for normalized Frigate bounding box overlays.
3. **InfluxDB:** Tag-based schema (site_id) with Flux Deadman checks for global "Site Down" alerting.
4. **Ansible:** Playbook utilizing community.general.proxmox_lxc and Terraform-generated tokens for Zero-Touch deployment.

Works cited

1. Home Assistant Remote Access: Best Methods 2025 ..., accessed on February 17, 2026, https://smarthomescene.com/top-picks/best-home-assistant-remote-access-methods-compared/
2. Mutual TLS · Cloudflare One docs, accessed on February 17, 2026, https://developers.cloudflare.com/cloudflare-one/access-controls/service-credentials/mutual-tls-authentication/
3. Controlling Zigbee TRV via KNX is quite slow - how to improve? - #7 by farmio, accessed on February 17, 2026, https://community.home-assistant.io/t/controlling-zigbee-trv-via-knx-is-quite-slow-how-to-improve/519272/7
4. KNX - Home Assistant, accessed on February 17, 2026, https://www.home-assistant.io/integrations/knx/
5. Automation in Action: Scenarios with Proxmox and Ansible - DEV Community, accessed on February 17, 2026, https://dev.to/serafiev/automation-in-action-scenarios-with-proxmox-and-ansible-3dno
6. [Support]: Home Assistant with "WebRTC Camera" and Frigate ..., accessed on February 17, 2026, https://github.com/blakeblackshear/frigate/discussions/21015

7. Camera Person Detection with Frigate AI & Home Assistant - YouTube, accessed on February 17, 2026, https://www.youtube.com/watch?v=V8vGdoYO6-Y
8. Zigbee network optimization: a how-to guide for avoiding radio frequency interference + adding Zigbee Router devices (repeaters/extenders) to get a stable Zigbee network mesh with best possible range and coverage by fully utilizing Zigbee mesh networking - Home Assistant Community, accessed on February 17, 2026, https://community.home-assistant.io/t/zigbee-network-optimization-a-how-to-guide-for-avoiding-radio-frequency-interference-adding-zigbee-router-devices-repeaters-extenders-to-get-a-stable-zigbee-network-mesh-with-best-possible-range-and-coverage-by-fully-utilizing-zigbee-mesh-networking/515752
9. InfluxDB data source | Grafana documentation, accessed on February 17, 2026, https://grafana.com/docs/grafana/latest/datasources/influxdb/
10. Get started with Grafana and InfluxDB, accessed on February 17, 2026, https://grafana.com/docs/grafana/latest/fundamentals/getting-started/first-dashboards/get-started-grafana-influxdb/
11. MQTT Statestream - Home Assistant, accessed on February 17, 2026, https://www.home-assistant.io/integrations/mqtt_statestream/
12. Python code to publish MQTT message to mosquitto? - Home Assistant Community, accessed on February 17, 2026, https://community.home-assistant.io/t/python-code-to-publish-mqtt-message-to-mosquitto/519545
13. MQTT device tracker for heartbeating devices - Frontend - Home Assistant Community, accessed on February 17, 2026, https://community.home-assistant.io/t/mqtt-device-tracker-for-heartbeating-devices/96702
14. App configuration | Home Assistant Developer Docs, accessed on February 17, 2026, https://developers.home-assistant.io/docs/add-ons/configuration/
15. S6 Overlay for our Docker containers - Home Assistant Developer Docs, accessed on February 17, 2026, https://developers.home-assistant.io/blog/2020/04/12/s6-overlay/
16. Data model schemas - Managed integrations for AWS IoT Device Management, accessed on February 17, 2026, https://docs.aws.amazon.com/iot-mi/latest/devguide/data-model-schemas.html
17. Configure OpenTelemetry data flow endpoints in Azure IoT Operations (preview), accessed on February 17, 2026, https://learn.microsoft.com/en-us/azure/iot-operations/connect-to-cloud/howto-configure-opentelemetry-endpoint
18. Flutter State Management BLoC VS Provider | by DevMonarch - Medium, accessed on February 17, 2026, https://medium.com/@DevMonarch/bloc-vs-provider-flutter-state-management-a-detailed-comparison-5a932e9033dd
19. Best Flutter State Management Libraries 2026 | Complete Guide - Foresight

Mobile, accessed on February 17, 2026, https://foresightmobile.com/blog/best-flutter-state-management
20. State Management in Flutter: Riverpod 2.0 vs. Bloc vs. Provider - DEV Community, accessed on February 17, 2026, https://dev.to/bestaoui_aymen/state-management-in-flutter-riverpod-20-vs-bloc-vs-provider-1ogh
21. MQTT | Frigate, accessed on February 17, 2026, https://docs.frigate.video/integrations/mqtt/
22. CustomPaint class - widgets library - Dart API - Flutter, accessed on February 17, 2026, https://api.flutter.dev/flutter/widgets/CustomPaint-class.html
23. Custom UI/UX in Flutter: Drawing Widgets with CustomPainter | by Prathamesh Mali | Medium, accessed on February 17, 2026, https://medium.com/@prathamesh.dev004/custom-ui-ux-in-flutter-drawing-widgets-with-custompainter-6c60b121f785
24. Building a WebSocket Client in Flutter: From Zero to Hero | by Punith ..., accessed on February 17, 2026, https://medium.com/@punithsuppar7795/building-a-websocket-client-in-flutter-from-zero-to-hero-30d945dc7722
25. Flux query over multiple downsampled buckets with different retention policies - InfluxDB 2, accessed on February 17, 2026, https://community.influxdata.com/t/flux-query-over-multiple-downsampled-buckets-with-different-retention-policies/14652
26. Multi tenant structure - InfluxDB 2 - InfluxData Community Forums, accessed on February 17, 2026, https://community.influxdata.com/t/multi-tenant-structure/28660
27. Create monitoring checks in InfluxDB | InfluxDB OSS v2 ..., accessed on February 17, 2026, https://docs.influxdata.com/influxdb/v2/monitor-alert/checks/create/
28. Mimir / Tenants | Grafana Labs, accessed on February 17, 2026, https://grafana.com/grafana/dashboards/16021-mimir-tenants/
29. Grafana Multi-Tenant - Brett Beeson, accessed on February 17, 2026, https://brettbeeson.com.au/grafana-multi-tenant/
30. Ansible · Cloudflare One docs, accessed on February 17, 2026, https://developers.cloudflare.com/cloudflare-one/networks/connectors/cloudflare-tunnel/deployment-guides/ansible/
31. laurent-martin/ets-to-homeassistant: script to convert ETS5 ... - GitHub, accessed on February 17, 2026, https://github.com/laurent-martin/ets-to-homeassistant
32. What is mTLS? | Mutual TLS - Cloudflare, accessed on February 17, 2026, https://www.cloudflare.com/learning/access-management/what-is-mutual-tls/
33. How to improve Zigbee Latency/Reliability : r/homeassistant - Reddit, accessed on February 17, 2026, https://www.reddit.com/r/homeassistant/comments/1qyn6xl/how_to_improve_zigbee_latencyreliability/

34. Build with Google Antigravity, our new agentic development platform, accessed on February 17, 2026, https://developers.googleblog.com/build-with-google-antigravity-our-new-agentic-development-platform/