Programming & APIs

Automating Cisco Enterprise Solutions

KNOWLEDGE CHEATSHEET

RELIABLE AUTOMATION LOOP

Intent Desired state → **Request** Auth + schema → **Observe** Status + data → **Validate** Result + drift

01 Programming and API foundations

SDN and API interaction patterns

- **Software-defined networking** separates control decisions from packet forwarding. A controller exposes network-wide intent and policy; devices still provide forwarding hardware and local functions. Overlays run over an underlying transport network.
- **Northbound APIs** connect applications to controllers. **Southbound interfaces** connect controllers to managed devices. Intent describes the desired outcome; the controller translates it into supported network actions.
- **REST** models resources through URIs and HTTP methods with stateless requests. **RPC** invokes a named operation with parameters; RPC is not restricted to a particular transport or device type.
- **Synchronous calls** wait for a response. **Asynchronous operations** return an acknowledgment or job identifier before work finishes; track queued, running, completed and failed states. Async client code and server-side background jobs are separate mechanisms.

Python environments and structured data

- **Virtual environments** isolate a project's interpreter environment and installed packages. Create one with `python -m venv .venv`, activate it and install dependencies there. Record requirements for repeatable setup; a venv does not emulate another OS.
- **JSON objects** become Python dictionaries; arrays become lists. Access nested data with `d['data'][0]['host-name']`: quote dictionary keys and use integer list indexes. Iterate over every returned item when collecting all devices or interfaces.
- **json.loads()** parses a JSON string; **json.dumps()** serializes a Python object to a string. **json.load()** and **json.dump()** work with file objects. JSON uses true, false and null; Python uses True, False and None.
- **Functions** return values explicitly. Assign the result before using it, pass required objects as arguments and distinguish local variables from outer scope. A loop needs an iterable; test empty lists and missing keys as well as the normal response.
- **XML** elements are identified by namespace URI and local name. Prefix spelling is arbitrary when bound correctly. Match namespaces when filtering or parsing NETCONF responses; a visually similar unqualified element can address a different node.

Programming & APIs

Automating Cisco Enterprise Solutions

KNOWLEDGE CHEATSHEET

RELIABLE AUTOMATION LOOP

Intent Desired state → **Request** Auth + schema → **Observe** Status + data → **Validate** Result + drift

Git workflow and Jinja2 templates

- **Git lifecycle:** `git init`, `git add`, `git commit`, configure a remote, then `git push -u origin <branch>`. Staging selects the content for the next commit; a commit records local history and a push transfers commits to a remote.
- **Feature branches:** `git switch -c feature` creates and selects a branch. To integrate it into development, select development and run `git merge feature`. Review and resolve conflicts before committing the merge; inspect `git status` and the diff.
- **Jinja2** uses `{{ value }}` for substitution and `{% for ... %}` / `{% if ... %}` for control flow. Generate repeated VLAN, SVI and access-port stanzas from inventory rather than copying device-specific text.
- **Template filters** transform input: `default` supplies missing values, `title` formats names and `selectattr` selects records. Keep booleans typed consistently, filter enabled VLANs deliberately and render both shutdown states before applying configuration.

HTTP requests, errors and safe retries

- **GET** reads, **POST** creates or invokes an action, **PUT** replaces the addressed resource, **PATCH** modifies selected content and **DELETE** removes it. The API contract defines the payload and behavior; do not infer action verbs from the resource name.
- **200** indicates success, **201** creation, **202** accepted work, **204** success without a body and **206** partial content. Acceptance is not proof that an asynchronous configuration deployment completed.
- **400** suggests invalid input; **401** missing or invalid authentication; **403** denied authorization; **404** an unavailable resource or path; **429** rate limiting. For **5xx**, inspect server and dependency failures, including database connectivity.
- **Requests:** use a Session for cookies and connection reuse, `params=` for query values and `json=` for JSON payloads. Set a timeout and call `response.raise_for_status()` before processing success data. Handle timeouts and authentication failures per device so successful results survive.
- **Response validation:** check status, Content-Type and body before `response.json()`. A JSONDecodeError can mean HTML, an empty response or malformed JSON. Honor Retry-After for throttling, use bounded backoff and follow pagination; verify an action's outcome before repeating writes.
- **Authentication** differs by platform: Basic credentials, bearer tokens, API keys, session cookies and XSRF tokens are not interchangeable. Refresh expired credentials, protect secrets, verify TLS certificates and avoid logging tokens or shared secrets.

Programming & APIs / YANG & Protocols

Automating Cisco Enterprise Solutions

KNOWLEDGE CHEATSHEET

RELIABLE AUTOMATION LOOP

Intent Desired state → **Request** Auth + schema → **Observe** Status + data → **Validate** Result + drift

AI assistance and bounded network tools

- **Intent parsing** extracts desired outcomes and constraints from text. Preprocessing normalizes inputs before analysis. AI-generated code still needs syntax, schema, topology and operational checks; manipulated telemetry or unchecked output can produce unsafe changes.
- **Traffic analytics** measures flows for troubleshooting and optimization. Threat analysis may combine signatures with anomaly detection. Topology simulations expose potential failure points; configuration audits identify drift. Validate predicted behavior against observed network state.
- **MCP tools** expose named functions with typed inputs and structured results to an AI client. With FastMCP, register functions as tools and run the server using the client's expected transport, such as stdio. Keep diagnostic prints off the protocol's stdout stream.
- **Network utility tools:** `ipaddress.ip_network()` supplies network and broadcast properties; interpret host counts with IPv4 /31 and /32 exceptions. `socket.gethostbyname()` resolves an IPv4 address; `socket.gethostbyaddr()` performs reverse lookup. Validate CIDRs and handle DNS failures explicitly.

02 YANG, NETCONF and RESTCONF

Read YANG models and instance data

- **YANG** defines data hierarchy, types, constraints, keys and operations. A **module** has its own namespace; a **submodule** contributes to a module. Data instances in XML or JSON must follow the supported model, including augmentations.
- **Container** groups child nodes; **list** holds repeated entries, usually identified by keys; **leaf** holds one value; **leaf-list** holds repeated scalar values. A list of interfaces is distinct from the container that encloses it.
- **Tree notation:** `rw` marks configuration data and `ro` read-only state; `?` marks optional nodes, `*` repeated nodes and brackets identify list keys. A trailing `!` identifies a presence container. Read indentation to reconstruct parent-child paths.
- **Native models** expose vendor-specific features. **IETF models** provide standardized schemas; **OpenConfig** provides operator-driven vendor-neutral schemas. Both standardized approaches need device support; a vendor-neutral model does not guarantee every platform implements every node.

YANG & Protocols

Automating Cisco Enterprise Solutions

KNOWLEDGE CHEATSHEET

MODEL TO INTERFACE

- **YANG** — Schema
- **NETCONF** — XML RPCs
- **RESTCONF** — HTTPS data
- **Capabilities** — Supported nodes

- **Interface modeling:** ietf-interfaces defines interface entries and names; ietf-ip augments interfaces with IP configuration; iana-if-type supplies interface-type identities. Match the list key, augmentation namespace, address prefix and enable state to the actual schema.
- **OpenConfig configuration** commonly separates intended `config` from reported `state`. For BGP, distinguish local AS under global configuration from each keyed neighbor's peer AS. Discover installed modules and supported features before constructing paths or payloads.

NETCONF sessions and datastores

- **NETCONF** exchanges XML RPCs, commonly over SSH on port 830. The hello exchange advertises capabilities. Enable NETCONF, provide authorized credentials and validate the SSH host key; successful SSH CLI access alone does not establish NETCONF access.
- **get-config** retrieves configuration from a specified datastore; **get** retrieves configuration and state data. With ncclient, use `get_config(source='running')` for running configuration and `get(filter=...)` for operational interface or routing data.
- **edit-config** changes a target datastore with an XML configuration payload. Merge preserves unspecified content; replace can remove content omitted from the addressed subtree. Use namespace-correct subtree or XPath filters and inspect RPC errors.
- **Candidate**, **validate**, **commit**, **confirmed commit** and **rollback-on-error** depend on advertised capabilities. Lock the relevant datastore to coordinate writers, validate before commit where supported and always release locks. Do not assume every device has a candidate datastore.
- **copy-config** copies between supported sources and targets. Running and startup are separate datastores where supported; changing running does not necessarily persist a reboot. Retrieve both and normalize XML before a unified diff to avoid formatting-only drift.

RESTCONF paths, payloads and replacement

- **RESTCONF** exposes YANG data and operations through HTTPS, normally under `/restconf/data` and `/restconf/operations`. It supports XML and JSON encodings; NETCONF RPC messages use XML. RESTCONF does not expose NETCONF's explicit candidate-lock-commit workflow.
- **Media types:** use `application/yang-data+json` or `application/yang-data+xml`. Accept requests the response encoding; Content-Type declares the request body's encoding. For Basic authentication, the Authorization value begins with Basic; Base64 is encoding, not encryption.
- **Resource paths** include module-qualified top-level nodes and list-instance keys, such as `/restconf/data/ietf-interfaces:interfaces/interface=Loopback1`. URI-encode special characters in keys. A wrong module, namespace or full path can cause missing data or a 404.

YANG & Protocols / IOS XE & Telemetry

Automating Cisco Enterprise Solutions

KNOWLEDGE CHEATSHEET

DEVICE AUTOMATION PATHS

- **Day 0** Bootstrap
- **Configure** Desired state
- **Telemetry** Push updates
- **Verify** Operational state

- **JSON payloads** follow YANG containers, lists and qualified names. Send a dictionary with `json=payload`, or a serialized JSON string with the correct Content-Type. Match the payload root to the addressed container, list or leaf.
- **PUT replacement** targets the addressed resource; omitted children there may disappear. PATCH merges selected changes according to supported patch semantics. For hostname compliance, read first, compare, update the precise hostname leaf and verify the result.

03 IOS XE automation and telemetry

CLI automation and on-box execution

- **Netmiko** provides device-aware SSH CLI sessions. `send_command()` collects show output; `send_config_set()` applies a command sequence; `send_config_from_file()` reads commands from a file. Save configuration explicitly when persistence is required.
- **Compliance loop:** collect the current value, compare it with desired state, change only a mismatch and verify afterward. Use independent exception handling for each router, close sessions and retain successful device results when other connections fail.
- **pexpect** automates interactive prompts with expect/send sequences. Terminal-server console access requires matching the actual login, privilege and configuration prompts. Telnet carries unencrypted traffic; confine it to an approved isolated bootstrap path.
- **Guest Shell Python** can call IOS XE CLI helpers for show output and configuration. `cli()` returns output; `clip()` prints output. Choose the supported execution or configuration helper for the command, handle failures and record health-check results to an allowed file location.
- **EEM** runs event-driven applets or policies, such as matching configuration-change syslog and executing privileged save commands. Account for AAA permissions and prompt behavior; avoid a trigger/action cycle that repeatedly invokes the same applet.

Ansible and Terraform desired state

- **Ansible** uses inventory, variables and YAML playbooks to run ordered tasks. Network automation is generally agentless, using SSH or APIs. Configure the connection type, network OS and privilege escalation required by the selected module.
- **IOS modules:** use command modules for show output, configuration modules for CLI changes and resource modules such as `cisco.ios.ios_l3_interfaces` for structured intent. Preserve indentation, supply required parameters and register results for later checks.
- **State matters:** resource-module `merged` preserves unrelated configuration, while replaced or overridden states can remove it. Older modules may use `state: present` or absent. Follow the selected module's schema rather than assuming all modules share the same state values.

IOS XE & Telemetry

Automating Cisco Enterprise Solutions

KNOWLEDGE CHEATSHEET

DEVICE AUTOMATION PATHS

- **Day 0** — Bootstrap
- **Configure** — Desired state
- **Telemetry** — Push updates
- **Verify** — Operational state

- **Loops** apply a task to inventory data, commonly with `loop: '{{ intlist }}'`; older playbooks use with_items. Jinja expressions substitute variables. For an HTTP task, provide the correct method, headers and body_format, then use the registered JSON result.
- **Terraform** uses providers, declarative resources and state to compare desired and managed infrastructure. Review plan before apply and protect state credentials. Both Terraform and Ansible help standardize configurations; idempotency depends on the provider, module and task design.

Day-0 provisioning mechanisms

- **ZTP** bootstraps an unconfigured device using network-delivered provisioning information. DHCP supplies addressing and boot information; the device retrieves a script or configuration over a supported transfer protocol. IOS XE workflows can run Python through Guest Shell.
- **PnP** coordinates discovery, claim, site assignment and provisioning through a controller. For Catalyst Center onboarding, identify the device by serial and product ID, import it into PnP and claim it with the required site and deployment information.
- **iPXE** performs network boot and image acquisition. On supported switches, `boot ipxe forever switch 1` retries network boot rather than relying on a one-time attempt. Plan DHCP, image-server reachability and a trusted image source together.
- **cloud-init** applies first-boot configuration on supported virtual devices. write_files can stage startup CLI and runcmd can invoke supported merge commands. Device-specific integration determines how the file becomes running configuration and how it is saved.

Model-driven telemetry subscriptions

- **Model-driven telemetry** pushes selected modeled data to a collector. A subscription specifies the stream or datastore, filter, update policy, encoding and transport. Precise subscriptions reduce repeated polling work and improve the timeliness of troubleshooting data.
- **Periodic** publication sends snapshots at the requested interval, useful for counters and trend analysis. **On-change** publication reports supported data changes, useful for state transitions. Counters, filters and transports do not all support both policies.
- **Dynamic subscriptions** are established at runtime, often through a NETCONF establish-subscription RPC, and depend on that session. **Configured subscriptions** are device configuration and can initiate delivery to a configured receiver independently of a subscriber session.

IOS XE & Telemetry / Catalyst Center & ISE

Automating Cisco Enterprise Solutions

KNOWLEDGE CHEATSHEET

CONTROLLER WORKFLOW

Authenticate Token → **Resolve** Device + site IDs → **Submit** Intent action → **Track** Task + outcome

- **YANG-push** describes subscription behavior; **OpenConfig** describes data models. For the documented IOS XE yang-push on-change workflow, set dampening-period to 0. Other native telemetry implementations may support dampening intervals; inspect the device's subscription capabilities.
- **Delivery troubleshooting:** check receiver address and port, source interface or VRF, routing, ACLs, transport and collector readiness. Verify the exact XPath and XML namespaces. Select an encoding the collector supports, such as JSON or GPB where available, and confirm subscription status.

04 Catalyst Center and ISE integration

Controller authentication and resource identity

- **Catalyst Center Intent APIs** expose northbound automation. Common resources begin `/dna/intent/api/v1/`. Authenticate with `POST /dna/system/api/v1/auth/token`, then include the returned token in **X-Auth-Token** over HTTPS. Renew it when it expires.
- **Inventory:** GET network-device resources, retain returned device IDs and use documented filters or detail searches. Display names, IP addresses, MAC addresses and controller UUIDs are different identifiers; use the identifier required by the endpoint.
- **Sites** form an area/building/floor hierarchy. Query `/dna/intent/api/v1/site` to resolve a known site name or hierarchy to its ID before assigning devices. Filter on the complete hierarchy when names repeat at different locations.
- **Discovery** needs a supported discovery type and matching target information, plus credentials for the intended access methods. Validate mandatory payload fields, including names where required. Device reachability, discovery completion and inventory synchronization are separate checks.

Asynchronous tasks and Command Runner

- **Asynchronous changes** commonly return a task ID and status URL. Poll the task until completion or failure, inspect progress and error details, then retrieve the resulting resource. A successful submission does not prove discovery or provisioning succeeded.
- **Command Runner sequence:** resolve device IDs, POST `/dna/intent/api/v1/network-device-poller/cli/read-request`, poll `/dna/intent/api/v1/task/{taskId}`, extract the result file ID and GET `/dna/intent/api/v1/file/{fileId}`.

Catalyst Center & ISE

Automating Cisco Enterprise Solutions

KNOWLEDGE CHEATSHEET

CONTROLLER WORKFLOW

Authenticate Token → **Resolve** Device + site IDs → **Submit** Intent action → **Track** Task + outcome

- **Command Runner** executes supported read-only CLI commands. Check command allow/block lists and per-device results. Use supported configuration templates or provisioning APIs for changes instead of sending configuration commands to a read-only runner.
- **Provisioning workflows** combine inventory, sites, templates, wireless profiles and fabric resources. Resolve required IDs, match the JSON schema and track deployment tasks. Device packages and multivendor extensions provide supported third-party integration; feature coverage varies by device package.

Assurance, synchronization and event delivery

- **Assurance** covers client, device and site health, issues and application experience. Issues APIs provide affected resources and remediation context. Path trace helps inspect a traffic path; software image management handles image distribution and activation.
- **Health snapshots** require the correct resource scope, identifier and timestamp units. An endpoint expecting epoch milliseconds cannot use seconds unchanged. Configuration-sync status describes controller/device alignment; compare intended and observed configuration before remediation.
- **Event subscriptions** select event IDs and delivery endpoints. A webhook normally sends an HTTPS POST with a JSON event body. Configure the callback URL, connector method and required authentication; parse event type and affected resource before routing alerts.
- **Integration bundles** can forward network events to REST endpoints for external dashboards or incident systems. Register the intended bundle and subscription, validate receiver reachability and acknowledge promptly. Queue downstream work and deduplicate retries using event identity.

ISE ERS and TrustSec objects

- **ISE ERS** exposes supported identity and policy resources over HTTPS. Enable the service, use an authorized API account and send the required JSON media types. Treat credentials and RADIUS shared secrets as sensitive input.
- **Network access devices** are ERS networkdevice resources. A device definition includes identity, IP address/mask information and authentication settings such as the RADIUS shared secret. Match the JSON root and nested fields to the ERS resource schema.
- **Security Group Tags** classify endpoints or groups for TrustSec policy. ERS sgt resources create and manage these group objects. Retain returned IDs for later policy relationships; the tag value and object identifier serve different purposes.
- **SGACLs** describe group-based traffic permissions. ERS sgacl resources carry ACL definitions; policy bindings determine which source and destination groups use them. Creating an SGT or SGACL object alone does not establish the complete segmentation policy.

SD-WAN Manager

Automating Cisco Enterprise Solutions

KNOWLEDGE CHEATSHEET

TEMPLATE DEPLOYMENT

Session Cookie + XSRF → **Inputs** Device variables → **Attach** Template action → **Verify** Per-device status

05 SD-WAN Manager automation

Session authentication and API requests

- **SD-WAN Manager APIs** commonly use `/dataservice` resources. In the session-based workflow, POST credentials to `/j_security_check`, retain the session cookie, retrieve `/dataservice/client/token` and send the XSRF value as **X-XSRF-TOKEN** where required.
- **Session reuse** keeps authentication cookies and security tokens together. A login response containing an HTML form is not a valid inventory JSON response. Verify authentication before querying, refresh expired sessions and validate the controller certificate.
- **Device inventory** supplies identifiers for overlay devices and WAN edges. Keep UUID, system IP, hostname and serial separate. A monitoring endpoint's deviceId must use the identifier that endpoint expects, rather than an arbitrary inventory field.
- **Request construction:** put deviceId and other documented filters in the correct query or path location. Use `params=` to encode local/remote color and device filters. Process the response's data list safely and archive structured JSON for later reporting.

Templates, policy and deployment status

- **Feature templates** describe individual configuration features; **device templates** assemble them for target devices. Before attachment, obtain the per-device input variables and validate mandatory values. Preserve the API's expected booleans and new/edited-template indicators.
- **Attachment workflow:** submit the documented POST with template and device input, retain the returned process or action ID and poll action status with that identifier. Inspect per-device success and failure details; the device ID does not replace the process ID.
- **Detach to CLI mode** is a separate mode-change action from deleting a template definition. Confirm affected device IDs, submit the supported CLI-mode request and verify completion before making direct CLI changes.
- **Centralized policies** define overlay behavior and require an activation workflow. Read the existing policy, alter only the intended routing preference and validate references before activation. Exported WAN-edge bootstrap configuration supports Day-0 onboarding and may contain sensitive provisioning information.

SD-WAN Manager / Meraki

Automating Cisco Enterprise Solutions

KNOWLEDGE CHEATSHEET

TEMPLATE DEPLOYMENT

Session Cookie + XSRF → **Inputs** Device variables → **Attach** Template action → **Verify** Per-device status

Monitoring and certificate lifecycle

- **Control connections and OMP** describe control-plane relationships and route exchange. **BFD** monitors overlay tunnel liveness. Collect state and uptime to distinguish a failed transport tunnel from a failed controller connection.
- **Monitoring APIs** expose interface counters and errors, hardware health, BGP routes, BFD sessions and application-aware routing statistics. Distinguish real-time operational state from historical statistics; apply the required device, time range, color and VPN filters.
- **Certificate management** includes CSR generation and supported signed-certificate installation or distribution. A CSR requests signing; a trusted CA issues the certificate. Separate controller/device identity certificates from the Manager web-server certificate workflow.
- **Troubleshooting sequence:** check HTTP status and body, API path, session/token validity and permissions before parsing fields. Check controller/backend availability for server failures. Reboot, certificate installation and policy deployment are actions whose completion must be verified.

06 Meraki cloud automation

Organizations, networks and device onboarding

- **Dashboard API** uses HTTPS at `api.meraki.com` under `/api/v1`. API-key clients send **X-Cisco-Meraki-API-Key**. Scope access to the intended organization and retain organization, network and device identifiers separately.
- **Network inventory and creation:** GET or POST `/organizations/{organizationId}/networks`. Creation requires a name and productTypes array, such as switch, wireless, appliance or camera. Set tags and timeZone as needed; retain the created network ID.
- **Network actions:** POST `/networks/{networkId}/bind` with configTemplateId to bind a configuration template; POST the network's split action to separate a combined network where supported. DELETE the network resource to remove it; these operations have different effects.
- **Device onboarding:** claim the serial into the target network, then update device properties such as name and address through the device resource. Claiming is distinct from assigning metadata or setting management addressing. Organization administrators are managed at organization scope.

Meraki

Automating Cisco Enterprise Solutions

KNOWLEDGE CHEATSHEET

CLOUD MANAGEMENT SCOPE

- **Organization** — Access + inventory
- **Network** — Product settings
- **Device** — Serial + metadata
- **Events** — Alerts + health

Appliance, wireless and firmware configuration

- **MX VLAN workflow:** enable VLAN support through the appliance VLAN settings resource, then create or update VLAN objects with ID, subnet and appliance IP. Creation commonly uses POST to the collection; updating an existing VLAN uses PUT to its resource.
- **Site-to-site VPN** settings live under the network's appliance VPN resources. Supply the intended mode and subnet participation, plus hub settings when applicable. Verify the complete returned configuration so an update does not drop required VPN relationships.
- **Wireless SSIDs** are addressed by network and SSID number. For enterprise authentication, configure the supported WPA/802.1X mode, RADIUS server address, port and secret. SSID configuration, authentication policy and ISE authorization must agree.
- **Firmware scheduling** updates a network's firmware-upgrade resource with the intended product's next upgrade time and target version ID. Appliance, switch and wireless products have separate settings. Enumerate eligible networks and verify each scheduled result before the change window.
- **Configuration settings** also cover switch-port descriptions, network SNMP and alert recipients. Use the endpoint's actual schema: community-based read access differs from user-based credentials, and alert destination changes differ from the external webhook receiver code.

Webhooks, health and rate control

- **Meraki webhooks** deliver event JSON to a registered HTTP server. Configure the destination and relevant alerts. Validate the shared secret where provided, inspect event type and severity, then extract device identity and occurrence time from the payload.
- **Python receivers** use an HTTP route accepting POST and parse the request JSON. Acknowledge promptly and handle retries idempotently. Route only relevant critical events to an external messaging API using that service's access token and destination identifier.
- **Webhook history** helps audit deliveries over a documented time range. **Organization device statuses** report device availability; summarize by status and model, retain timestamps and distinguish stale monitoring data from a newly observed outage.
- **HTTP 429** means the API rate limit was exceeded. Respect Retry-After, pace organization-wide loops and use supported batch operations or SDK retry controls. Follow pagination when listing networks, devices or status records so reports cover the entire organization.

Meraki

Automating Cisco Enterprise Solutions

KNOWLEDGE CHEATSHEET

CLOUD MANAGEMENT SCOPE

- **Organization** — Access + inventory
- **Network** — Product settings
- **Device** — Serial + metadata
- **Events** — Alerts + health

External captive portal workflows

- **EXCAP** hosts an external captive portal. The client is redirected to the portal with context; the portal presents the splash or authentication flow, returns the client through the supplied authorization path and then redirects to the allowed destination.
- **Click-through** acknowledges splash terms; **sign-on** requires authentication. Keep the portal redirect flow distinct from the management API used to configure SSIDs and splash settings. Preserve and validate supplied return URLs and client context.

Location services and MV Sense

- **Scanning APIs** export Wi-Fi and BLE observations for location analytics, mapping and wayfinding. Receiver validation and observation processing are separate steps. Treat location as an estimate; AP placement, the environment and client orientation affect accuracy.
- **MV Sense** exposes camera analytics such as detections, occupancy and light measurements. REST requests retrieve supported snapshots or historical information; **MQTT** uses broker-based publish/subscribe topics for live analytics delivery.
- **MQTT topics** identify the camera and measurement stream, including light-level and zone analytics. Light measurements are not a REST resource merely because the topic resembles a path. Match the message schema and handle reconnects before interpreting occupancy or lux values.