Smart IoT Bolt Resource Catalog

Technical Documentation

1. Introduction

The Resource Catalog is a critical core component in the Smart IoT Bolt for Pipelines system. It serves as the central registry and discovery mechanism that enables the system's microservice architecture to function effectively. This document explains how the Resource Catalog works and details its functions in the context of the overall project.

2. System Context

Within the Smart IoT Bolt platform, the Resource Catalog plays several key roles:

- **Service Discovery**: Allows microservices to locate each other without hardcoded endpoints
- **Device Registry**: Maintains an inventory of all Smart IoT Bolts and their status
- **Configuration Hub**: Stores system-wide settings and thresholds
- **Status Monitoring**: Tracks the operational status of all components

The Resource Catalog is the first service all other microservices contact upon initialization to discover the system's topology and their required endpoints.

3. Architecture Overview

The Resource Catalog follows a layered architecture with the following structure:

Copy

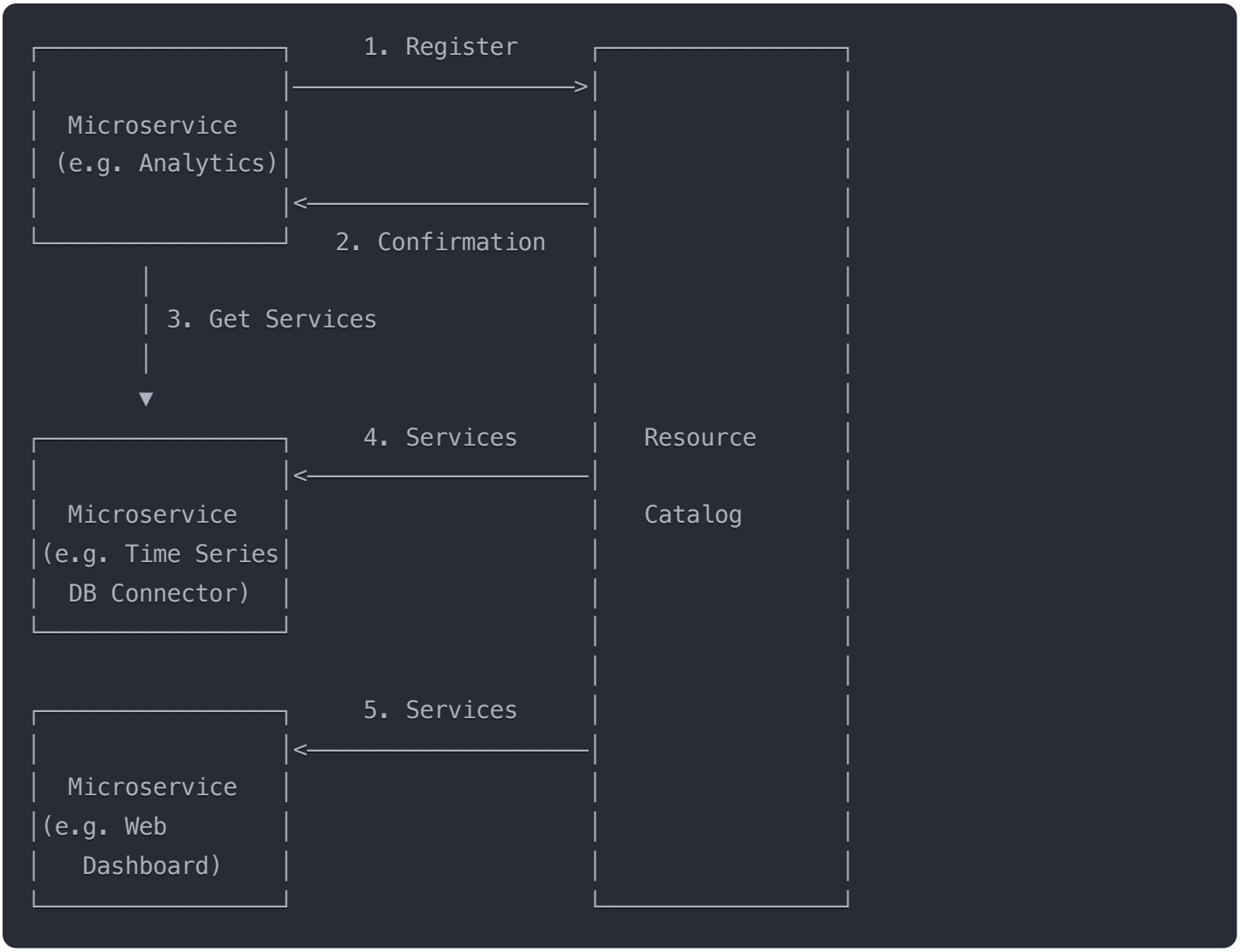
```
MS_ResourceCatalog
├── Controllers Layer (REST API Endpoints)
│   └── CatalogController
├── Services Layer (Business Logic)
│   └── RegistryService
├── Models Layer (Data Structures)
│   ├── Service
│   └── Device
└── Persistence Layer
    └── JSON Storage File
```

Key architectural characteristics:

3.1 Data Flow Diagrams

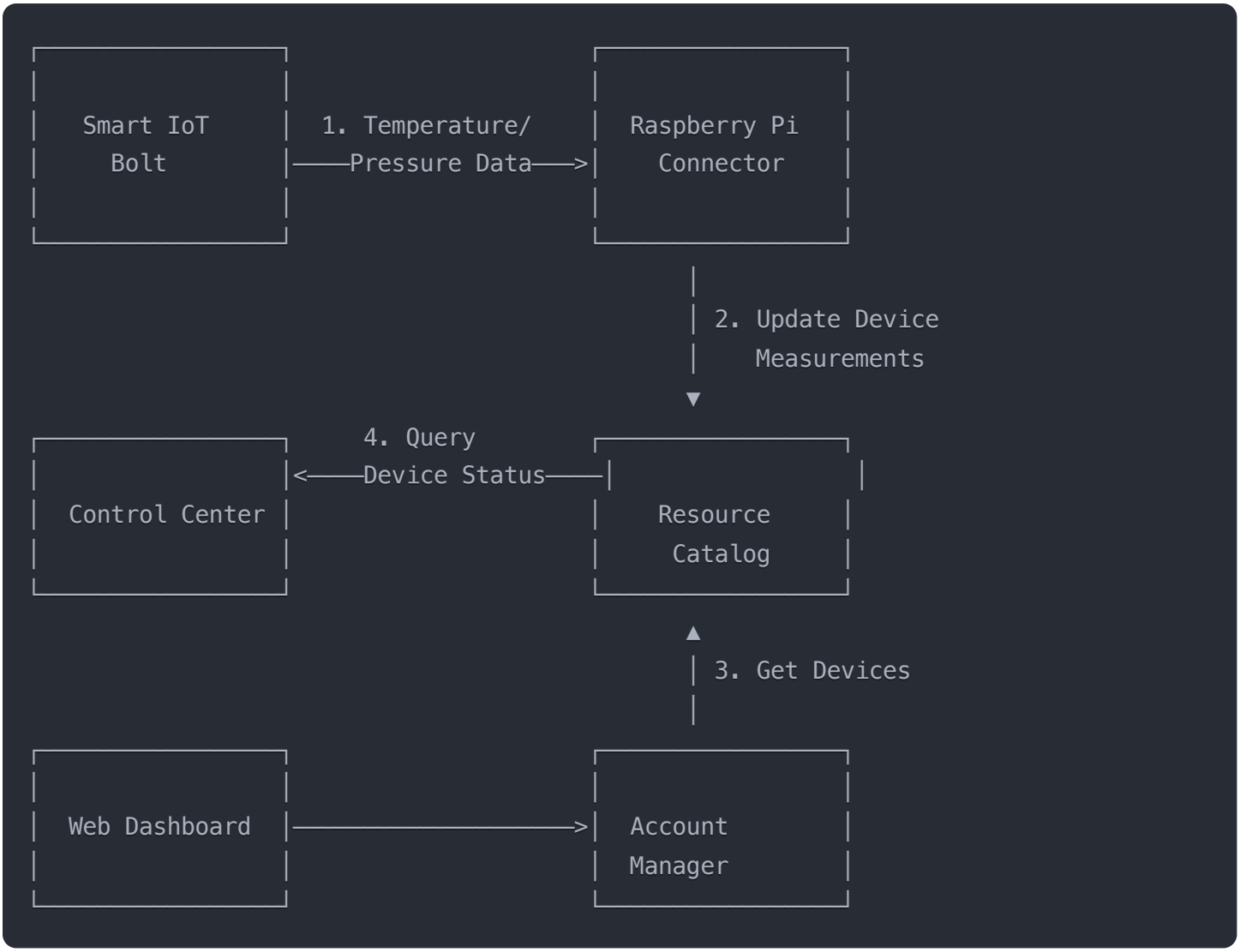
Service Registration and Discovery Flow:

Copy



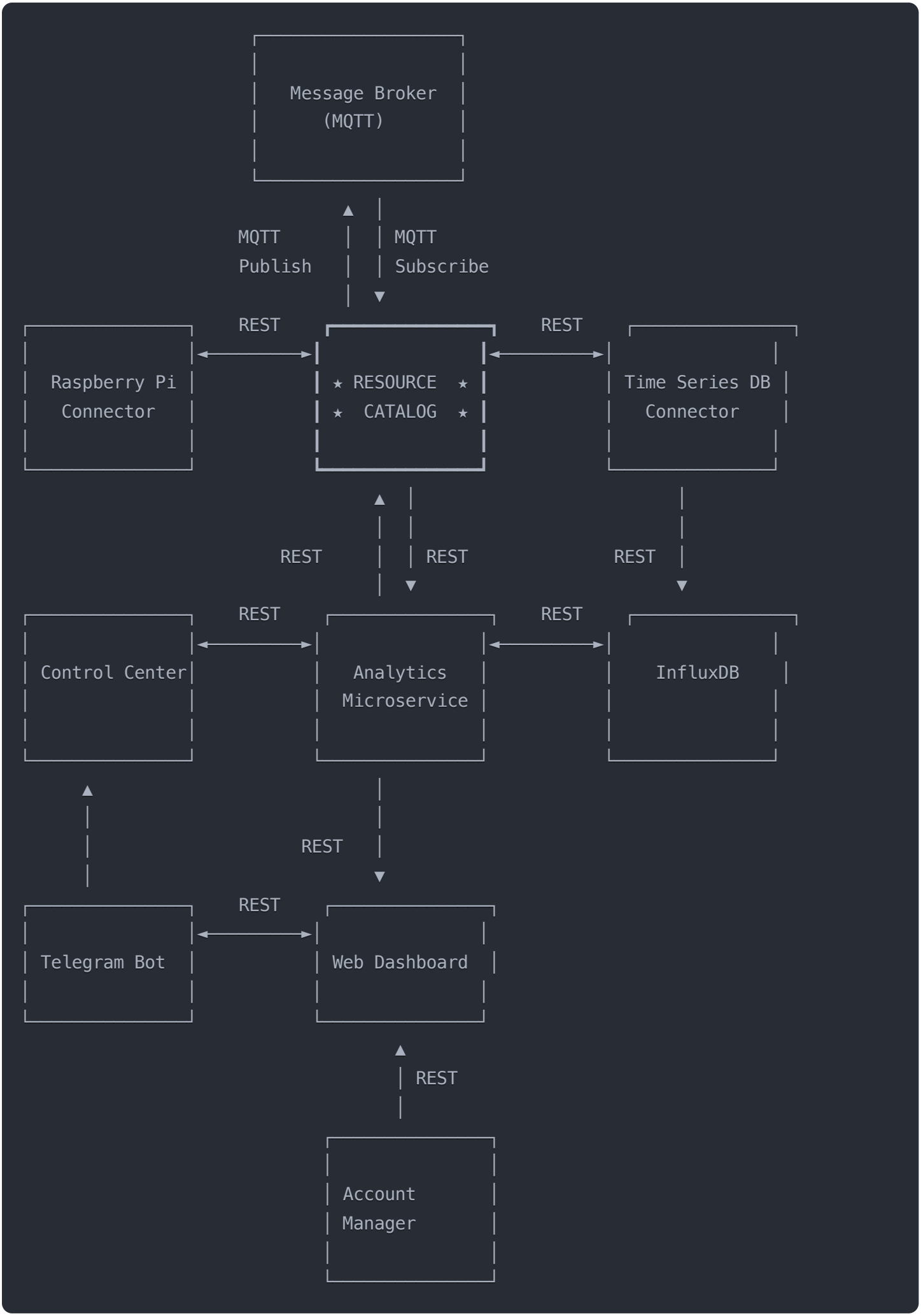
Device Data Update Flow:

Copy



System-Wide Data Flow:

Copy


Message Broker
(MQTT)
MQTT
Publish
MQTT
Subscribe
REST
REST
Raspberry Pi
Connector
★ RESOURCE ★
★ CATALOG ★
Time Series DB
Connector
REST
REST
REST
REST
REST
Control Center
Analytics
Microservice
InfluxDB
REST
REST
Telegram Bot
Web Dashboard
REST
Account
Manager

- **Models Layer**: Defines the data structures for services and devices
- **Services Layer**: Implements the business logic for registry management
- **Controllers Layer**: Exposes the RESTful API endpoints
- **Persistence Layer**: Handles data storage and retrieval

4. Key Components and Functions

4.1 Models

4.1.1 Service Model (models/service.py)

The Service model represents registered microservices in the system.

Key Functions:

- `to_dict()`: Serializes service data for API responses
- `from_dict()`: Creates a Service object from JSON data

Context in Project:
This model stores information about all microservices in the system, including:

- The Control Center that makes decisions about valve operations
- The Analytics Microservice that predicts pipeline failures
- The Time Series DB Connector that manages historical data
- The Web Dashboard and Telegram Bot that interact with users

4.1.2 Device Model (models/device.py)

The Device model represents IoT devices in the system, primarily the Smart IoT Bolts.

Key Functions:

- `to_dict()`: Serializes device data for API responses
- `from_dict()`: Creates a Device object from JSON data
- `update_measurements()`: Updates sensor readings (temperature, pressure)
- `update_status()`: Updates device operational status

Context in Project:
This model stores information about all IoT devices, including:

- Smart IoT Bolts that monitor pipeline conditions
- Valve actuators that control pipeline flow
- Temperature and pressure sensors

4.2 Registry Service (services/registry_service.py)

The RegistryService implements the core business logic for managing the service and device registries.

Key Functions:

4.2.1 Service Registry Functions

- `register_service(service_data)`: Registers a new microservice in the system
  - **Context**: When a new microservice comes online (e.g., Analytics, Control Center), it registers itself with the Catalog to become discoverable by other services.
- `update_service(service_id, service_data)`: Updates an existing service's details
  - **Context**: Microservices periodically update their status to indicate they are still active.
- `get_service(service_id)`: Retrieves a specific service's details
  - **Context**: The Control Center might use this to locate the Analytics Microservice when it needs prediction data.
- `get_all_services()`: Retrieves all registered services
  - **Context**: The Web Dashboard uses this to show system-wide status of all components.

4.2.2 Device Registry Functions

- `register_device(device_data)`: Registers a new Smart IoT Bolt or other device
  - **Context**: When a new Smart IoT Bolt is deployed, the Raspberry Pi Connector registers it with the Catalog.
- `update_device(device_id, device_data)`: Updates device details
  - **Context**: Used when device configurations change, such as location or associated services.
- `update_device_measurements(device_id, measurements)`: Updates sensor readings
  - **Context**: The Raspberry Pi Connector regularly updates pressure and temperature readings from Smart IoT Bolts.
- `update_device_status(device_id, status)`: Updates a device's operational status
  - **Context**: Indicates whether a Smart IoT Bolt is active, inactive, or in an error state.
- `get_device(device_id)`: Retrieves a specific device's details
  - **Context**: The Control Center uses this to check the status of specific Smart IoT Bolts.
- `get_all_devices()`: Retrieves all registered devices
  - **Context**: The Web Dashboard uses this to show the status of all Smart IoT Bolts.
- `get_devices_by_type(device_type)`: Retrieves devices of a specific type
  - **Context**: The Analytics Microservice might request only temperature sensors for a specific analysis.

4.2.3 System Management Functions

- `load_data()`: Loads registry data from persistent storage
  - **Context**: On startup, the Resource Catalog restores its state from the last saved state.
- `save_data()`: Saves registry data to persistent storage
  - **Context**: After any change to the registry, data is persisted to survive restarts.
- `_cleanup_stale_entries()`: Removes or marks inactive services/devices
  - **Context**: Ensures the registry accurately reflects the current system state by identifying components that have stopped updating their status.

4.3 Catalog Controller (controllers/catalog_controller.py)

The CatalogController exposes the RESTful API endpoints that other microservices use to interact with the registry.

Key Functions:

- `services()`: Endpoint to retrieve all registered services
  - **Context**: Used by all microservices during initialization to discover the system topology.
- `service(service_id)`: Endpoint to retrieve a specific service
  - **Context**: Used when a service needs to communicate with a specific other service.
- `register_service()`: Endpoint to register a new service
  - **Context**: Called by each microservice when it first comes online.
- `update_service(service_id)`: Endpoint to update a service's details
  - **Context**: Microservices call this regularly to maintain their "active" status.
- `devices()` and `device(device_id)`: Endpoints to retrieve device information
  - **Context**: The Web Dashboard, Telegram Bot, and Analytics services use these to monitor the system.
- `register_device()` and `update_device()`: Endpoints to manage device registration
  - **Context**: The Raspberry Pi Connector uses these to keep device information current.
- `update_device_measurements(device_id)`: Endpoint to update sensor readings
  - **Context**: The Raspberry Pi Connector uses this to report new temperature and pressure readings.

5. Workflow Examples

5.1 Service Registration Workflow

1. When the Analytics Microservice starts up:
   - It prepares its service information (endpoints, description)

- It makes a POST request to `/register_service` on the Resource Catalog
- The Resource Catalog registers the service and returns confirmation
- The Analytics service can now be discovered by other services

2. The Analytics service then:
   - Makes a GET request to `/services` to discover other services
   - Finds the Time Series DB Connector to retrieve historical data
   - Finds the Web Dashboard and Telegram Bot to send alerts

5.2 Device Data Update Workflow

1. A Smart IoT Bolt detects a pressure change:
   - The Raspberry Pi Connector receives the new reading
   - It makes a POST request to `/update_device_measurements/{bolt_id}` with the new data
   - The Resource Catalog updates its registry

2. Simultaneously:
   - The Raspberry Pi Connector publishes the reading to the MQTT broker
   - The Time Series DB Connector stores it in InfluxDB
   - The Control Center evaluates if action is needed
   - The Analytics service analyzes for potential issues

6. Integration with Other System Components

6.1 Integration with Message Broker

While the Resource Catalog primarily uses REST, it indirectly connects to the MQTT message flow:

- Services discover MQTT topics and connection details through the Catalog
- The Catalog maintains a mapping between devices and their associated MQTT topics

6.2 Integration with Control Center

The Control Center heavily relies on the Resource Catalog to:

- Locate sensors and actuators to monitor and control
- Access threshold settings and control parameters
- Determine device-to-service relationships

6.3 Integration with Web Dashboard

The Web Dashboard uses the Resource Catalog to:

- Authenticate and authorize users by interfacing with the Account Manager

- Discover service endpoints for data visualization
- Show system topology and component status

7. Error Handling and Resilience

The Resource Catalog implements several resilience mechanisms:

- **Thread Safety**: Uses locks to prevent concurrent modification issues
- **Automatic Cleanup**: Identifies and marks stale services/devices
- **Persistence**: Regularly saves state to recover from crashes
- **Exception Handling**: Wraps all operations in try-except blocks

8. Security Considerations

The Resource Catalog's security model includes:

- **Basic Authentication**: For initial API access control
- **Service Validation**: Verifying the authenticity of service registrations
- **Separation of Concerns**: Clear boundaries between system components

In future phases, this will be enhanced with:

- Firebase Authentication for more robust security
- Role-based access control for different user types

9. Conclusion

The Resource Catalog serves as the central nervous system of the Smart IoT Bolt platform. It enables the loosely coupled microservice architecture while maintaining system coherence. By providing dynamic service discovery and device management, it eliminates hardcoded dependencies and supports the system's scalability and resilience goals.

Without the Resource Catalog, the Smart IoT Bolt system would require extensive reconfiguration when components change, and services would be unable to locate each other dynamically. Its role is essential to the flexible, maintainable architecture that allows the platform to grow and adapt to changing industrial monitoring needs.