Software Design Specification

Author(s): Adel Ali

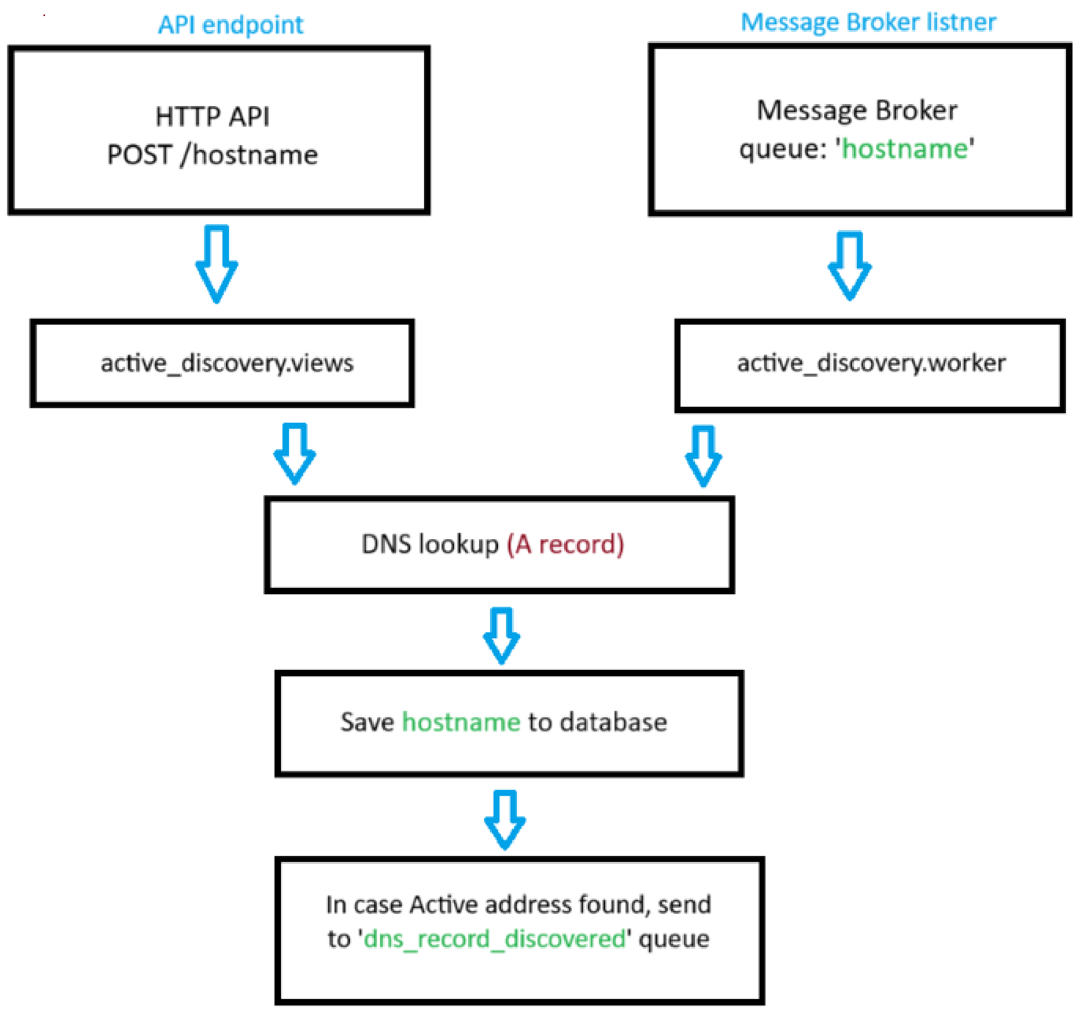
Date: 04-06-2025

Introduction

This document shows the high level design of the hostname_discovery project, explains how different components interact with each other and the possibility to add new functionalities in the future.

1. High level design and system architecture

The system is a hostname discovery and processing service built mainly with Django, Celery, and RabbitMQ. It allows for both HTTP-based and message broker-based reporting of hostnames, performs DNS lookups, and publishes results via a message broker in case an active hostname found.



1.1 Flow explanation

Two Entry Points:

- **HTTP API:**
  Clients send a HostDiscovered message (containing a hostname and source) to the /hostname endpoint.
  Django receives the request, validates the data, and queues a Celery task for background processing.

- **Message Broker (RabbitMQ):**
  External systems publish HostDiscovered messages directly to the “hostname” queue in RabbitMQ.
  The broker listener script (consumer) picks up these messages and triggers the same Celery task for processing.

Task Processing

- **Celery Tasks:**
  The Celery worker receives the task (triggered by either entry point).
  Then:
  - The worker performs a DNS lookup for the provided hostname.
  - If successful, it updates the database with the hostname and its resolved IP, with has_record field set to true.
  - Then result is published as a DNSRecordDiscovered message to the dns_record_discovered queue in RabbitMQ.
  - If not successful, it updates the database with the hostname with empty IP, with has_record field set to false. And the system logs the error.

2. System Components

2.1 Components description

- **Django Web Server:**
  - Hosts the HTTP API endpoint (`/hostname`) for submitting hostnames.
- **Message Broker (RabbitMQ):**
  - Receives `HostDiscovered` messages.
  - Used for both reporting hostnames and publishing results.
- **Celery:**
  - Processes hostnames asynchronously (DNS lookup, result publishing).
- **Broker Listener (consumer):**
  - Standalone script that listens to the `hostname` queue and triggers Celery tasks.
- **Database (SQLite by default):**
  - Stores hostname records and lookup results.

2.2 Component Interactions

- **HTTP API Entry Point:**
  - Clients POST hostnames to `/hostname`.
  - Django validates and queues a Celery task for processing.
- **Broker Entry Point:**
  - External systems publish `HostDiscovered` messages to the `hostname` queue.
  - The broker listener picks up messages and triggers Celery tasks.
- **Task Processing:**
  - Celery workers receive hostname tasks, perform DNS lookups, and update the database.
  - Results are published to the `dns_record_discovered` queue.
- **Result Publishing:**
  - Results are available for consumers via RabbitMQ.

3. Tips and tricks

3.1 To add a new feature, extend the Task Logic:

- Modify the Celery task (`process_reported_host`) to include a new step after DNS lookup for any new functionality needed.
- Can also add a new field to the (`Hostname`) model in case additional information is needed to be stored.

3.2 Complex or Tricky Parts

- **Opening connection:**
  It can cause overhead to the system to open connection with the message broker upon each request which is a heavy process, so better to check first if there is an open connection and reuse it. This is handled in `MessageQueue` class.

- **Database scalability:**
  Currently the default django development database sqlite3 is used, for more scalable and reliable production datasets Postgres or MySql can be used instead. (or MongoDB for no sql databases in case of sharding needed, Cassandra)

- **Environment Setup:**
  Running RabbitMQ, Celery, and Django together, especially on Windows, can be tricky due to process management and permission issues, will provide steps to run the system in a readme file.