

Crossbow Delivery Service

Link to the deployed application:

<http://35.200.136.252:8080/>

Just click on the link and get started! To see detailed steps, please go to page 2

Setting up the application:

Below are the steps, if you wish to run the application locally

Step 1: Open Docker Desktop, Open a terminal in it (lower right corner)

Step 2: Clone the repo

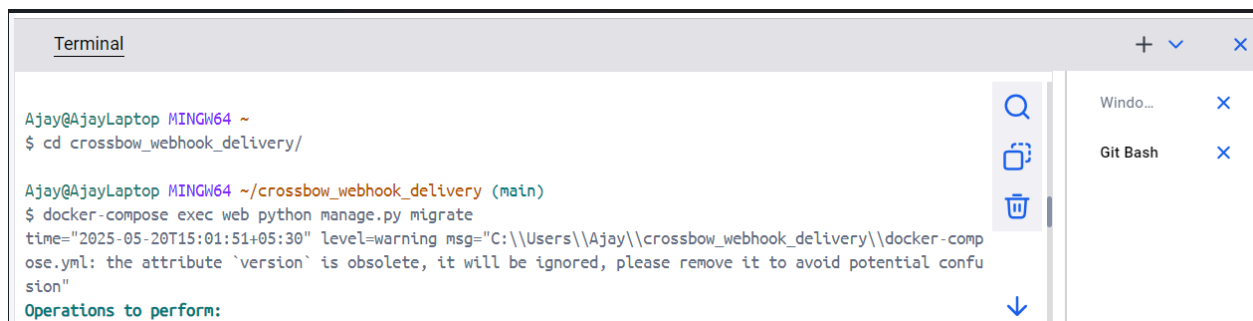
git clone https://github.com/vedanshbvb/crossbow_webhook_delivery.git
cd crossbow_webhook_delivery

Step 3: Build and Run with Docker Compose

[docker-compose up --build](#)

Step 4: Open git bash in the terminal window and run the migrations

[cd crossbow_webhook_delivery](#)
[docker-compose exec web python manage.py migrate](#)



```
Terminal
Ajay@AjayLaptop MINGW64 ~
$ cd crossbow_webhook_delivery/

Ajay@AjayLaptop MINGW64 ~/crossbow_webhook_delivery (main)
$ docker-compose exec web python manage.py migrate
time="2025-05-20T15:01:51+05:30" level=warning msg="C:\\Users\\Ajay\\crossbow_webhook_delivery\\docker-comp
ose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confu
sion"
Operations to perform:
```

Step 5: Access the Application

Open your web browser and go to: <http://localhost:8080>

Step 6: Stopping the Application

To stop the application, press `Ctrl+C` in the terminal where docker-compose is running, or open a new terminal and run: [docker-compose down](#)

Architecture Choices:

a) Framework:

- Django (5.2.1) as the main web framework
- Django REST Framework for API endpoints
- SQLite for development (with PostgreSQL configuration commented out for production)

- Redis as the message broker and result backend for Celery

b) Async Task/Queueing System:

- Celery for handling asynchronous webhook deliveries
- Redis as the message broker and result backend
- Exponential backoff retry strategy with 5 maximum retries
- Retry delays: 10s, 30s, 1m, 5m, 15m (progressive backoff)

c) Retry Strategy:

- Maximum of 5 retry attempts
- Progressive backoff with increasing delays
- Detailed logging of retry attempts and failures
- Automatic cleanup of old delivery logs after 72 hours

Database Schema and Indexing Strategies:

a) Models:

- Subscription Model:
 - Foreign key to User
 - Unique subscription_id
 - target_url for webhook delivery
 - Optional secret_key for signature verification
 - event_types as comma-separated list
 - Indexes on (user, subscription_id) and created_at
- DeliveryLog Model:
 - Foreign key to Subscription
 - attempt_number for tracking retries
 - status field for delivery state
 - http_status_code and error_details for debugging
 - event_type for tracking
 - Indexes on (subscription, timestamp), status, and event_type

b) Indexing Strategy:

- Composite indexes for common query patterns
- Timestamp-based indexing for log cleanup operations
- Status-based indexing for monitoring and analytics
- Event type indexing for filtering and reporting

Using the application:

To see video explanation:

<https://drive.google.com/drive/folders/19xUze09-YtprMSHmp77FV18-F736Bmio?usp=sharing>

Step-1: Login and Registration:



Login

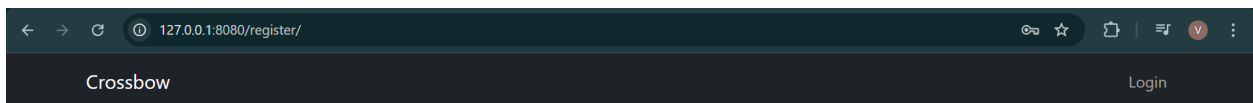
Username

Password

Login

Don't have an account? [Register here](#)

If you do not have an account, go to the registration page and register your account



Register

Username

Email

Password

Confirm Password

Register

Already have an account? [Login here](#)

Once you register, you'll be automatically logged in

127.0.0.1:8080/facilities/

Crossbow

Facilities Logout

View Logs

Your User ID
ID: 18 [Copy](#)
Keep this ID handy for creating subscriptions

Create Subscription
User ID

Target URL

Event Types (comma-separated)

[Create Subscription](#)

Send Payload
Subscription ID

Event Type

[Send Payload](#)

Check Delivery Status
Delivery ID

[Check Status](#)

View Attempts for Delivery ID
Delivery ID

[View](#)

View Attempts for Sub ID
Subscription ID

[View](#)

Step-2: Creating subscription

Copy your user id and create a subscription

You can write comma separated events for which you want to receive notifications

Create Subscription

User ID

Target URL

Event Types (comma-separated)

[Create Subscription](#)

Create Subscription

User ID

Target URL

Event Types (comma-separated)

Create Subscription

Subscription Created

Subscription ID: 43

Copy

Use this ID to send payloads

Step-2: Sending Payload:

Copy Subscription id and also a single event for which the notification is being sent.

Send Payload

Subscription ID

Event Type

Send Payload

Payload Queued for Delivery

Delivery ID: 54

Copy

Use this ID to check delivery status

Step-3: Checking Delivery Status:

Copy Delivery id and enter

Check Delivery Status

Delivery ID

Check Status

Status: Success

Step-4: View attempt for a specific Delivery id:

Copy Delivery id and enter

View Attempts for Delivery ID

Delivery ID

View

Recent Attempts for Delivery ID 54

Attempt #	Status	HTTP Status	Event Type	Timestamp	Error Details
1	Success	200	driver.reached	2025-05-19T11:54:07.471428Z	-

Step-5: View attempt for a specific Subscription id:

Copy Subscription id and enter

View Attempts for Sub ID

Subscription ID

View

Recent Attempts for Subscription ID 43

Attempt #	Status	HTTP Status	Event Type	Timestamp	Error Details
1	Success	200	driver.reached	2025-05-19T11:54:07.471428Z	-

Step-6: View Logs:

Click on view logs button in top right corner

Your User ID

Copy

Keep this ID handy for creating subscriptions

User ID

18

Target URL

<http://httpbin.org/post>

Event Types (comma-separated)

```
user.updated, order.created, driver.reached
```

[Create Subscription](#)

Subscription ID

Subscription ID

43

Event Type

```
driver.reached
```

Send Payload

Payload Queued for Delivery

Delivery ID: 54 [Copy](#)

Delivery ID

Delivery ID

54

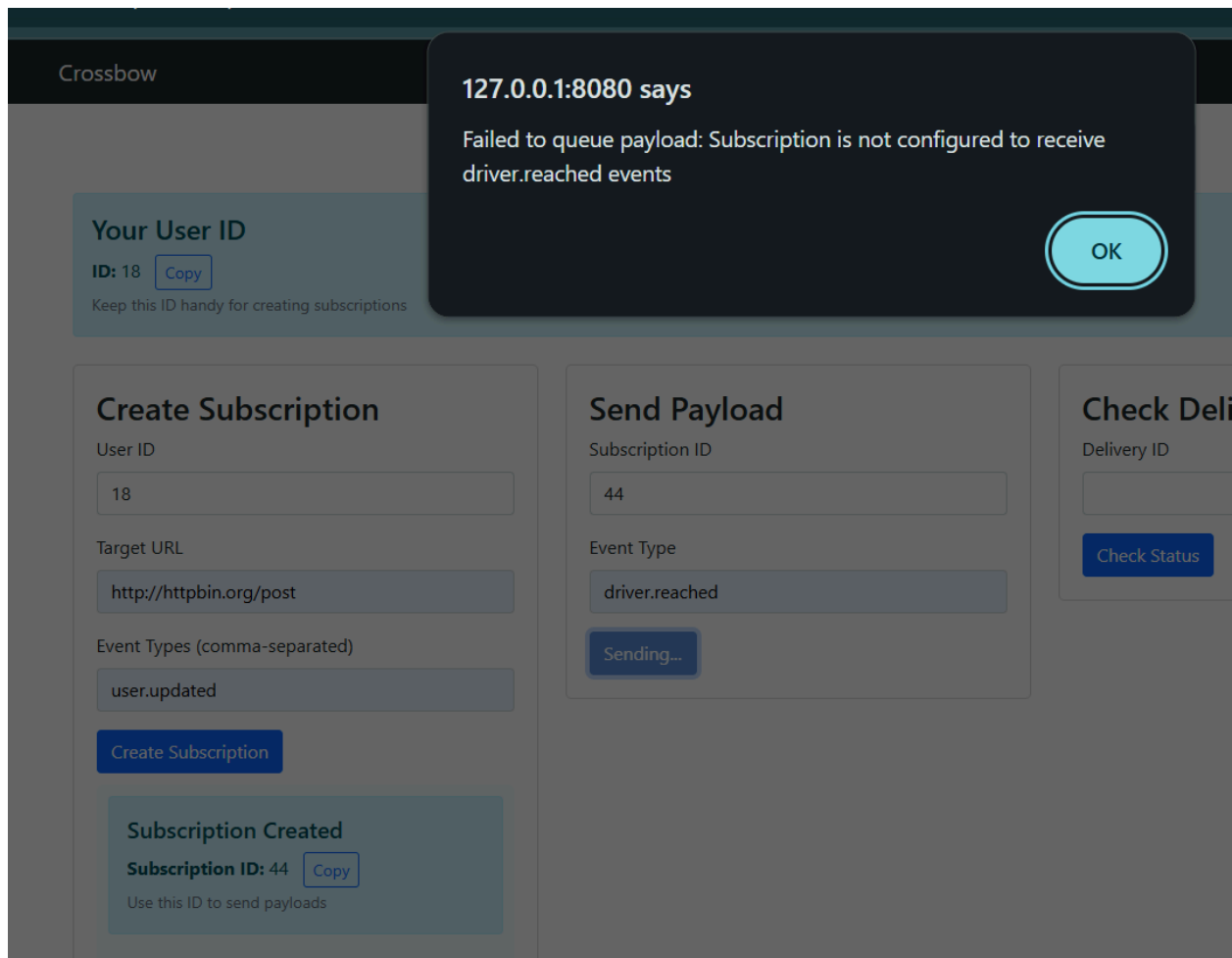
[Check Status](#)

Status: Success

[← Back to Facilities](#)

Delivery ID	Subscription ID	Target URL	Timestamp	Attempt #	Outcome	HTTP Status	Error Details
54	43	http://httpbin.org/post	May 19, 2025, 11:54 a.m.	1	Success	200	-

In case the event type for the payload is not supported by the subscription, user is shown an error message:



Monthly Cost Estimation for Google Compute Engine (GCE) Free Tier Deployment

Assumptions:

- **5000 webhooks/day** = ~150,000 webhooks/month
- **1.2 delivery attempts per webhook** = ~180,000 total attempts/month
- **Average webhook payload size** = 1KB
- **Average processing time per attempt** = 2 seconds
- **GCE instance type** = e2-micro(2 vCPUs, 1 GB RAM)
- **Operating System**: Debian-based (eligible for free tier)
- **Deployment runs 24x7**

- **Single VM instance**

Cost Breakdown:

a) Google Compute Engine (GCE):

- **e2-micro instance** is covered under GCP Free Tier (1 instance per month, 30 days x 24 hours = 720 hours/month)
 - **Free tier includes:** Approx Rs. 25,000 credits

The Always Free tier for Google Compute Engine is only available in select U.S. regions (us-west1, us-central1, us-east1). Since the current deployment is in asia-south2 (Delhi), the free tier does not apply.

- **e2-micro VM (24x7 usage):**
 - $\$0.0076/\text{hour} \times 730 \text{ hours/month} \approx \mathbf{\$5.55/\text{month}}$
- **Persistent Disk (Standard):**
 - $\$0.04/\text{GB/month} \times 10 \text{ GB} = \mathbf{\$0.40/\text{month}}$
- **Egress (network outbound):**
 - First 1 GB/month is free
 - Estimated usage: 180,000 requests \times 1KB = ~180MB/month $\Rightarrow \mathbf{\$0}$

Total Estimated Monthly Cost (Post-Free Trial):
 $\approx \$5.95/\text{month}$

Current Cost (During Free Trial):

- VM: \$0 (under free trial credits)
- Disk: \$0 (assuming <30 GB)
- Egress: \$0 (assuming <1 GB)
- **Total: \$0/month**

Architecture Assumptions:

a) Infrastructure:

- Google Compute Engine (GCE) e2-micro instance used for deployment
- Redis used for message queuing (hosted on same VM or managed externally)
- PostgreSQL for persistent storage (hosted locally or managed via Cloud SQL)

b) Performance:

- Average webhook processing time: 2 seconds
- Average payload size: 1KB
- 72-hour retention for delivery logs
- Up to 5 retry attempts with exponential backoff

c) Security:

- Webhook signature verification is optional (configurable)
- CORS is currently set to allow all origins (development only)
- Session-based authentication via tokens; secured with HTTPS

d) Scalability:

- Single e2-micro VM can handle current workload (~180k delivery attempts/month)
- Redis and PostgreSQL are expected to handle moderate message and data throughput
- System provides reliability through retry mechanisms and persistent logging
- For higher scale, migration to Cloud Run or autoscaled GCE setup is possible

Conclusion:

Under the current deployment region (asia-south2), the Always Free tier for GCE does **not** apply. However, the cost remains relatively low:

- **Current Cost (Free Trial Active):** \$0/month
- **Expected Cost (Post-Trial): ~\$5.95/month**, assuming 1 e2-micro VM, ~10 GB disk, and <1 GB outbound egress

This architecture remains efficient and cost-effective for a webhook delivery system with retry logic, performance monitoring, and moderate scale capabilities.