Routewren complete guide

Drivers, IT staff and developers at a delivery firm that uses Routewren

Routewren shows your vans on a live map and tells your own software when a van arrives. This one guide has three parts. Read the part that is yours.

For drivers is part 1. It shows how to run a shift in the Routewren Driver app.

For IT is part 2. It sets up a new depot, an API key and a webhook.

For developers is part 3. It follows one location ping through the pipeline.

Contents

1. For drivers
   - 1.1 Start your shift
   - 1.2 Deliver a stop
   - 1.3 Break and end of the route

2. For IT
   - 2.1 Before you start
   - REQUIRED The depot code cannot be changed
   - 2.2 Create the depot
   - 2.3 Set the depot rules
   - 2.4 Create an API key
   - 2.5 Connect a webhook
   - 2.6 Check the setup

3. For developers
   - 3.1 The big picture
   - 3.2 Send pings
   - 3.3 What the ingest API ignores
   - 3.4 Arrived and delivered
   - 3.5 Events and signatures
   - 3.6 Retries and the live map
   - 3.7 Numbers to remember

Change log

1. For drivers

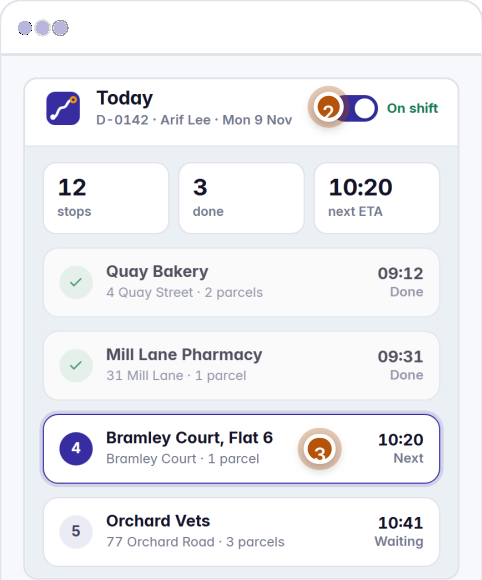
You use the **Routewren Driver** app on your phone. It shows your stops for the day and tells Routewren where your van is.

Driver app › **Today**

1.1 Start your shift

> **Warning**
> Never share your PIN. Set the phone location to **Always**. If it is not, your van disappears from the map when the screen is off.

1. Open **Routewren Driver**. Log in with the depot code, your driver ID, for example `D-0142`, and your 4-digit PIN.
2. Check that **On shift** is switched on.
3. Look at the list. The blue card is your next stop. Each stop has a time and a status: **Next**, **Waiting** or **Done**.

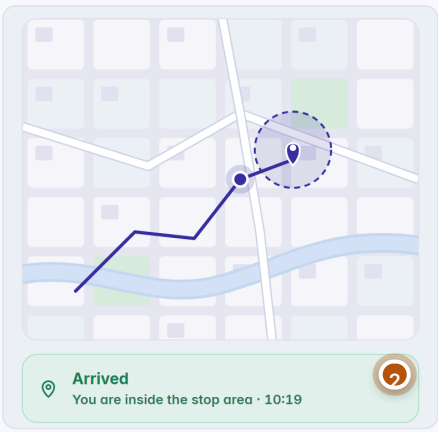


Your stops for today

1.2 Deliver a stop

1. Tap your next stop. The card shows the address, the customer's first name and the number of parcels.

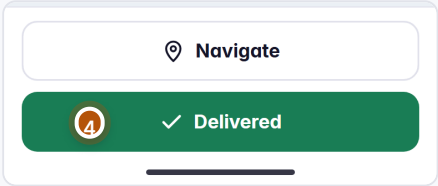
2. Tap **Navigate** and drive to the stop. When the van is close enough, a green **Arrived** card shows by itself.



The green Arrived card

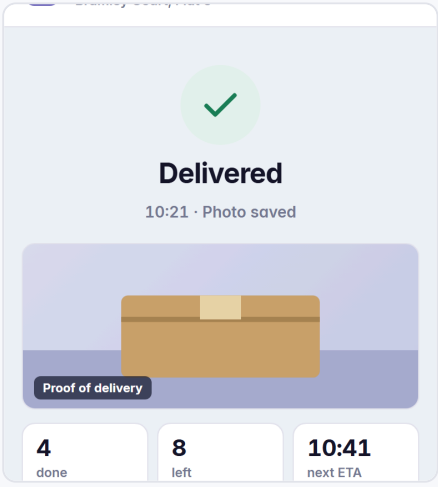
3. Hand over the parcel.

4. Tap **Delivered**.



Tap Delivered

5. Take a photo if the app asks. The depot decides if a photo is needed.



The stop is delivered

6. Tap **Next stop**.

1.3 Break and end of the route

> For a break, switch **On shift** off. The app sends no location while you are on break. At the last stop the app says **Route complete**.

2. For IT

A depot is a place where vans start and finish the day. Routewren needs one depot for each. This part sets up the depot, its rules, an API key and a webhook, in that order.

Console › **Depots**

2.1 Before you start

You need	Example	Note
A Console login	your work e-mail, `mei@example.com`	Ask your admin for a login
The depot name	`Northside Depot`	Shown on the map and in reports
The depot code	`nth-depot-01`	The API and the driver login use it
The time zone	`Asia/Kuala_Lumpur`	Picked from a list
The street address	`12 Dock Lane, Northside Industrial Park`	Routewren finds the map pin from it
A server address for webhooks	`https://orders.example.com/hooks/routewren`	Must start with `https`

REQUIRED

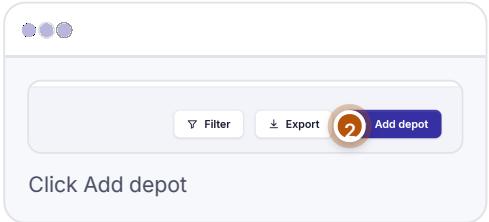
Use lowercase letters, digits and hyphens, up to 24 characters. The API and the driver login use the code. A wrong code means a new depot, so check it twice.

The Console has a **Live** and a **Test** switch at the top right. Test has fake vans and sends nothing real, and an amber strip shows under the top bar. A depot exists in both modes, but the keys are different. Make your keys and webhooks in Test first.

2.2 Create the depot

Console › Depots › **Add depot**

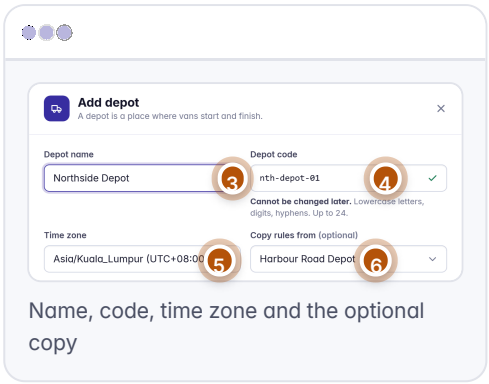
1. Click **Depots** in the left menu.
2. Click **Add depot** at the top right.



Click Add depot

3. Type the depot name, `Northside Depot`.

4. Type the depot code, `nth-depot-01`. A green tick means the code is free.
5. Pick the time zone, `Asia/Kuala_Lumpur`.
6. Open **Copy rules from** and pick `Harbour Road Depot`. This is optional.



Name, code, time zone and the optional copy

Then set the address and create the depot.

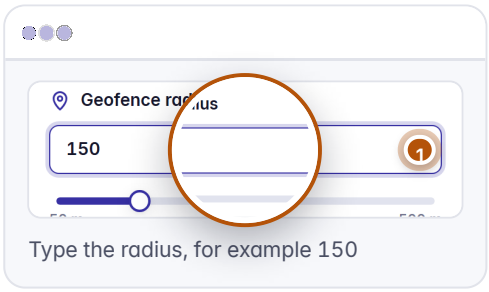
1. Type the street address.
2. Look at the map pin. Drag it if it is in the wrong place.
3. Click **Create depot**. The depot opens on its **Rules** page.

2.3 Set the depot rules

Depots › Northside Depot › **Rules**

The geofence radius decides when a van counts as arrived. The default is 150 m.

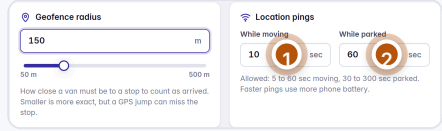
1. Type the radius, from 50 to 500.



Type the radius, for example 150

Setting	Default	Allowed	What it does
Geofence radius	150 m	50 to 500 m	How close a van must be to a stop to count as arrived
Ping while moving	10 sec	5 to 60 sec	How often a moving van reports
Ping while parked	60 sec	30 to 300 sec	How often a parked van reports
Working hours	06:00 to 20:00	any	No alerts outside these hours
Late alert after	15 min	any	The dispatcher gets a red flag when a stop is this late
History kept	90 days	30 to 365 days	After this the location history is deleted for good

1. Set **While moving** in the Location pings card.
2. Set **While parked** in the same card.



The two ping intervals

3. Set the working hours **Start**.
4. Set the working hours **End**. Tick the days.
5. Set **Late alert after**.
6. Set **History kept**.

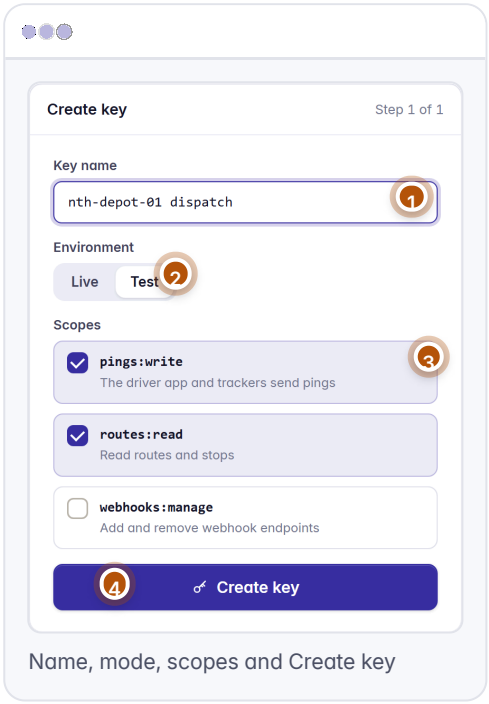
> **Warning**
> Nothing is saved until you click **Save changes** in the bar at the bottom. If you leave the page first, the changes are lost.

2.4 Create an API key

Depots › Northside Depot › **API keys**

Make the first key in **Test** mode. A key works for one depot and one mode.

1. Type a name for the key, `nth-depot-01 dispatch`.
2. Choose **Test**.
3. Tick the scopes the key needs. Give a key only what it needs.
4. Click **Create key**.



Name, mode, scopes and Create key

Scope	What the key can do	Give it to
`pings:write`	Send location pings	The driver app and GPS trackers
`routes:read`	Read routes and stops	Your dashboards and reports
`webhooks:manage`	Add and remove webhook endpoints	Your own set-up scripts only

> **Stop**
> Routewren shows the key one time. It keeps only a hash. If you lose the key, revoke it and make a new one.

1. Click **Copy**.
2. Paste the key into your team's secret store.
3. Click **I have copied the key**.

A Test key starts with `rw_test_` and a Live key starts with `rw_live_`. Send the key in the `Authorization` header. The limit is 600 requests a minute for each key.

```bash
curl https://api.example.com/v1/routes \
  -H "Authorization: Bearer <api_key>"
```

Answer	Meaning	What to do
`401`	The key is wrong or revoked	Check the key and the mode
`403`	The key has no scope for this call	Make a key with the scope
`422`	A ping is not valid	Check the fields of the ping
`429`	More than 600 requests a minute for this key	Wait for the time in the `Retry-After` header

> To replace a key: make the new one, switch your system to it, wait 24 hours, then click **Revoke** on the old key.

2.5 Connect a webhook

Depots › Northside Depot › **Webhooks**

1. Click **Add endpoint**.
2. Type the address of your server. It must start with `https`.
3. Tick the events you want, then click **Save**. Routewren shows the signing secret one time. Copy it into your secret store.

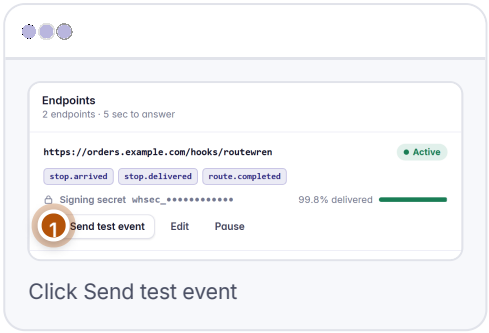
Event	When it is sent
`stop.arrived`	A van is inside the geofence of its next stop
`stop.delivered`	The driver taps **Delivered**
`route.completed`	The last stop of a route is done
`driver.offline`	A driver has sent no pings for a while

Every call has a `Routewren-Signature` header. Your server checks it before it trusts the call, and answers with a 2xx code within 5 seconds.

```text
Routewren-Signature: t=<timestamp>,v1=<signature>

signed    = "<timestamp>.<body>"
expected  = HMAC-SHA256(<signing_secret>, signed)
accept    = expected equals <signature>, and <timestamp> is under 5 minutes old
```

1. Click **Send test event** on the endpoint. Routewren sends a fake `stop.arrived`.



Click Send test event

2. Open the delivery log and look for the code `200`.

If your server fails, Routewren tries again 6 times and then marks the endpoint **Failing**. The admin gets an e-mail.

2.6 Check the setup

- [ ] The depot code is right, and the pin is on the depot.
- [ ] The rules are saved. Reload the page and look.
- [ ] A Test key exists with only the scopes it needs, and it is in your secret store.
- [ ] Your server answers the test event with a `200`.
- [ ] The same steps are done again for Live keys and Live webhooks.

What you see	Likely cause	Fix
A van is not on the map	Phone location is not set to **Always**	Set it on the driver's phone
Arrived comes too early or too late	The radius is too large or too small	Change the geofence radius
`403` from the API	The key has no scope for the call	Make a key with the scope
The endpoint shows **Failing**	Your server is slow or returns an error	Fix the server, then send a test event

3. For developers

A van sends its position. A few seconds later a customer's server hears that the van has arrived. This part follows one location ping through Routewren, part by part.

3.1 The big picture

A ping enters on the left and an event leaves on the right. Plate 1 shows the seven parts.

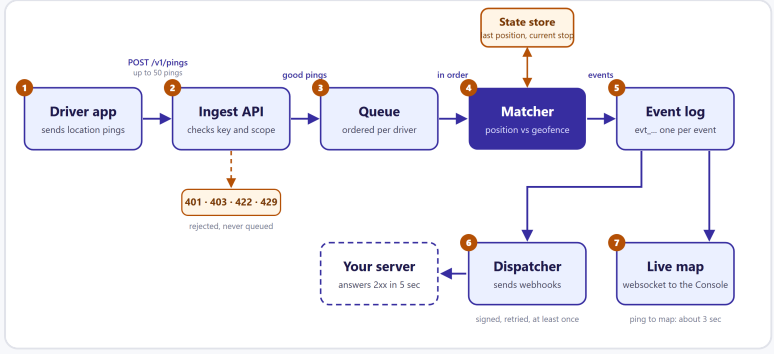


Plate 1. The tracking pipeline, from the driver app to your server and the live map

Part	What it does	What it keeps
1 Driver app	Sends location pings in batches	Up to 200 pings when there is no signal
2 Ingest API	Checks the key, the scope and the ping	Nothing
3 Queue	Holds good pings in order, per driver	The pings until the matcher reads them
4 Matcher	Compares the position with the geofence of the next stop	State store: last position and current stop
5 Event log	Holds every event with its `evt_...` id	The events
6 Dispatcher	Calls your webhook, signed, with retries	Delivery log, 30 days
7 Live map	Pushes the event to the Console over a websocket	Nothing

Order. Pings of one driver are handled in the order they arrive.

At least once. An event is delivered one time or more. Use its `id` to ignore a copy.

Speed. From a ping to the Console map takes about 3 seconds.

3.2 Send pings

Send up to 50 pings in one call. The key needs the `pings:write` scope. When the phone has no signal, the driver app keeps up to 200 pings and sends them later. The API answers `202 Accepted` and tells you how many pings it accepted.

bash

```
curl -X POST https://api.example.com/v1/pings \
  -H "Authorization: Bearer <api_key>" \
  -H "Content-Type: application/json" \
  -d '{
    "driver_id": "D-0142",
    "pings": [
      { "lat": <lat>, "lng": <lng>, "speed": 41, "battery": 78,
        "accuracy": 12, "ts": "2026-11-09T10:19:04+08:00" }
    ]
  }'
```

Field	Meaning	Note
`driver_id`	The driver, for example `D-0142`	Written once for the whole batch
`lat`, `lng`	The position	Use `<lat>` and `<lng>` in your own tests
`speed`	How fast the van moves	Sent by the app
`battery`	Battery level of the phone	Sent by the app
`accuracy`	How exact the position is, in metres	Worse than 100 m is ignored for arrived
`ts`	When the position was taken	ISO time. Part of the duplicate check

Code	Meaning
`202`	Accepted. The body has the number of accepted pings
`401`	The key is wrong or revoked
`403`	The key has no `pings:write` scope
`422`	A ping is not valid. It is never queued
`429`	More than 600 requests a minute. Wait for `Retry-After`

3.3 What the ingest API ignores

The same `driver_id` and `ts` twice is ignored, so a phone can safely send a batch again. A late ping is handled like this.

How late	What happens
Up to 10 minutes	Accepted and handled in order. Events can still come from it
More than 10 minutes	Stored for the history, but it makes no events

3.4 Arrived and delivered

The matcher compares each ping with the geofence of the driver's next stop. Two pings in a row inside the radius mean arrived. One ping can be a GPS jump, so one is not enough. A ping with an accuracy worse than 100 m is ignored. The driver taps **Delivered** to finish the stop.

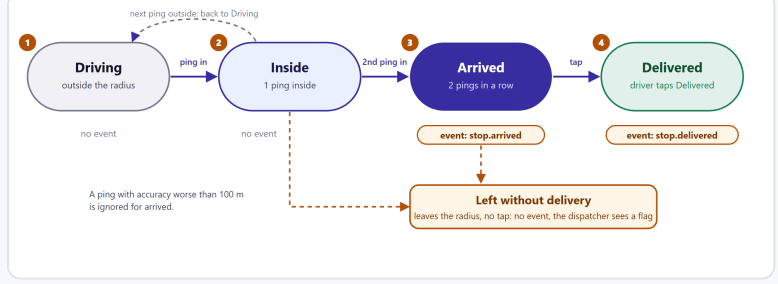


Plate 2. How the matcher decides that a van has arrived, delivered or left

> **Warning**
> A driver who leaves the radius without tapping **Delivered** makes no event. The dispatcher sees a flag in the Console: *left without delivery*. Your server hears nothing about that stop.

3.5 Events and signatures

An event has an `id`, a `type`, the `depot` code, the `driver_id`, a `stop_id` and `occurred_at`. The types are `stop.arrived`, `stop.delivered`, `route.completed` and `driver.offline`.

```json
{
  "id": "evt_<id>",
  "type": "stop.arrived",
  "depot": "nth-depot-01",
  "driver_id": "D-0142",
  "stop_id": "stop_<id>",
  "occurred_at": "2026-11-09T10:19:08+08:00"
}
```

Every call has the header `Routewren-Signature: t=<timestamp>,v1=<signature>`. The signature is the HMAC-SHA256 of `<timestamp>.<body>`, made with the signing secret. Reject a call older than 5 minutes.

```python
import hashlib, hmac, time

def is_valid(header, body, signing_key):
    parts = dict(p.split("=", 1) for p in header.split(","))
    if abs(time.time() - int(parts["t"])) > 300:
        return False
    signed = f'{parts["t"]}.{body}'.encode()
    expected = hmac.new(signing_key, signed, hashlib.sha256).hexdigest()
    return hmac.compare_digest(expected, parts["v1"])
```

3.6 Retries and the live map

Your server must answer with a 2xx code within 5 seconds. Delivery is at least once, so keep the `id` of every event you handled and skip a copy.

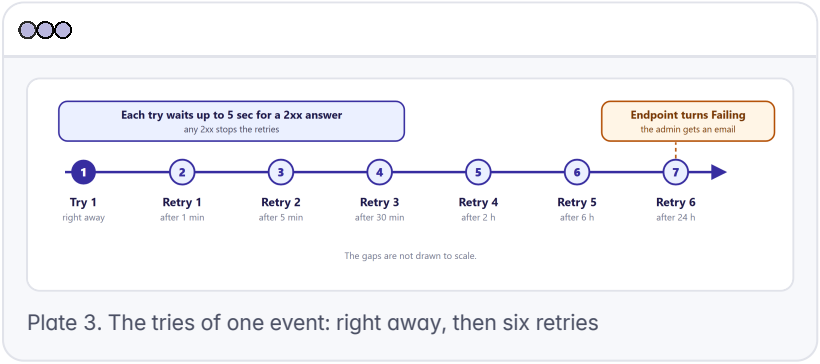


Plate 3. The tries of one event: right away, then six retries

The Console map gets its events over a websocket and shows **Live** while it is open. From a ping to a moving van on the map takes about 3 seconds.

3.7 Numbers to remember

What	Value
Pings in one call	Up to 50
Pings kept on the phone without signal	Up to 200
Requests for one key	600 a minute
Duplicate ping	Same `driver_id` and `ts`, ignored
Late ping that still makes events	Up to 10 minutes late
Accuracy that counts for arrived	100 m or better
Arrived	2 pings in a row inside the radius
Your answer to a webhook	A 2xx code within 5 seconds
Retries	6: after 1 min, 5 min, 30 min, 2 h, 6 h, 24 h
Signature age	Reject when older than 5 minutes
Ping to map	About 3 seconds

Change log

Version	Date	Change	By
1.0	2026-11-06	First version, merged from the depot setup guide, the pipeline guide and a short driver part	Routewren Docs