SIT314 – Distinction Task (6.3D)

Smart Warehouse – Inventory & Orders Microservice

Student: Ocean Ocean (s223503101)

Repository: https://github.com/s223503101/smart-warehouse.git

1. Executive Summary

This project demonstrates a complete, working IoT data pipeline. Node-RED simulates sensor readings and sends them to a small REST API that sits in front of a cloud database (MongoDB Atlas). The API provides a clean, reusable interface for microservices and clients. Two functional areas are implemented: **Inventory** (update and read stock levels) and **Orders** (create and read orders). Evidence from the health page, Node-RED responses, simple request tests, and Atlas documents confirms the end-to-end path: **sensor → Node-RED → API → database → read-back**. The design directly fulfills the brief to have Node-RED sending data to an API in front of MongoDB and to provide RESTful access suitable for IoT microservices.

2. Objectives

The build focused on four practical goals:

1. **Send data from Node-RED to an API:** Simulate sensors and orders and post them to the service.
2. **Persist data in the cloud:** Store inventory and orders in MongoDB Atlas for reliable, visible records.
3. **Provide RESTful access:** Allow other microservices or tools to read the same data with simple HTTP requests.
4. **Evidence the result:** Capture screenshots that clearly show the service is healthy, accepts writes, stores records, and returns correct reads.

3. System Overview

The system has three small, well-defined parts that work together:

- **Node-RED (event source):** Two flows simulate activity. One produces inventory updates at regular intervals; the other creates sample orders on demand.
- **Express API (microservice):** The API is the “front door” to the database. It checks a simple API-key header, validates the incoming request, and then writes or reads data.
- **MongoDB Atlas (database):** Two collections are used—one for inventory, one for orders—so it is easy to confirm that records are stored and can be queried.

Components at a glance

Component	**Role in the system**
Node-RED	Simulates sensors and orders; sends HTTP requests to the microservice
Express API	Validates requests, enforces a simple API key, and mediates all access to the database
MongoDB Atlas	Stores inventory and orders in separate collections for reliable cloud persistence

4. Implementation

Two small Node-RED flows send clearly structured data to the API. The inventory flow periodically sends stock levels for a known item; the orders flow creates sample orders when triggered. The API applies a lightweight security check using a header, validates the incoming fields, and then performs either an upsert (for inventory) or a create (for orders). Reading the same records back confirms the path end-to-end.

To keep the prototype straightforward and assessable, the design focuses on correctness, clarity, and observability. There is a health page for the service, a Node-RED Debug panel that shows success responses, and MongoDB Atlas views that display the exact documents written.

What the API can do

Capability	**Purpose**	**Typical success result**
Health check	Confirm the service is running	Small “OK” status response
Update inventory	Record or update stock for a specific item	Updated inventory record returned
Read inventory	Return the latest stock for a specific item	Single inventory document returned
Create order	Add a new order linked to an item and a quantity	New order record returned
Read order	Return a specific order by its identifier	Single order document returned

5. Security and Quality

For a development prototype, all microservice routes require an API-key header to avoid accidental or unauthorised use. Sensitive configuration is kept out of the repository in a private environment file, while a safe example file is included so others can run the project without revealing secrets. For a public deployment, the next steps are straightforward: enable standard security headers, restrict allowed origins, limit request rates, use HTTPS behind a gateway or managed platform, and validate inputs more strictly at the edge. These improvements are listed in the next section.

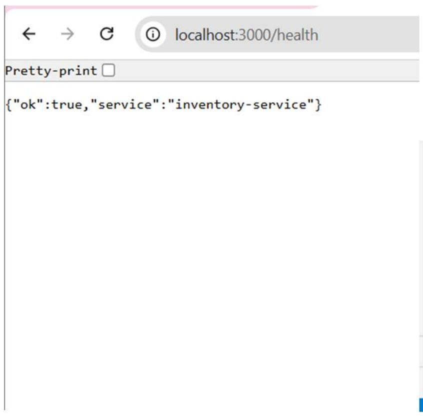
6. Testing and Results

Testing focused on proving the complete data journey from write to read:

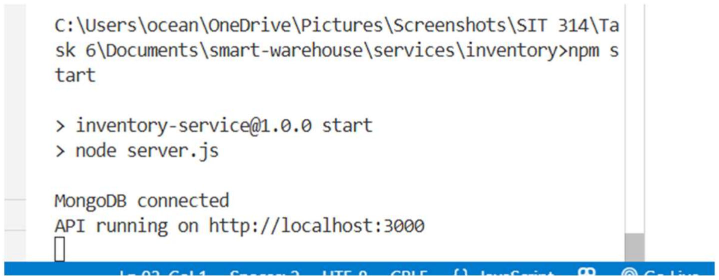
- **Service availability:** The health page confirms the API is up and reachable.
- **Writes from Node-RED:** The Debug panel shows success responses for both inventory updates and order creation.
- **Database storage:** MongoDB Atlas displays the expected documents under the inventory and orders collections.
- **Reads back out:** Simple GET requests return the exact inventory item and order that were previously written, confirming round-trip integrity.

Evidence mapping

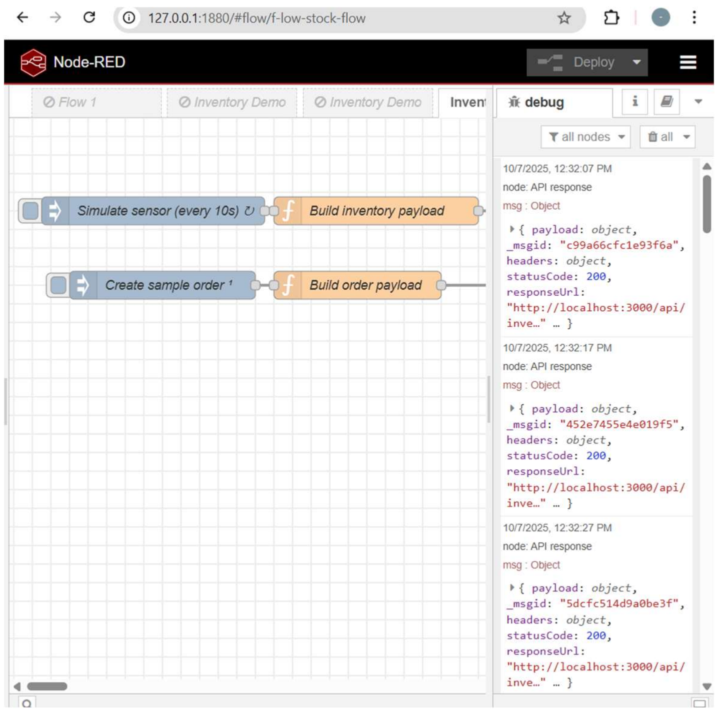
Figure	Evidence shown	What it proves
1	Service health page	API is live and reachable
2	Terminal logs (“MongoDB connected / API running …”)	Database connection and service start-up are successful
3	Node-RED Debug: inventory update success	Write path from Node-RED to API works
4	Node-RED Debug: order creation success	Orders can be created through the API
5	Node-RED flow canvas	Clear, simple flow design
6	MongoDB Atlas: inventory document for a known item	Data is stored correctly in the cloud database
7	MongoDB Atlas: orders document for a known order identifier	Orders are persisted and queryable
8–10	Simple request results	Read-backs match what was written; end-to-end confirmed



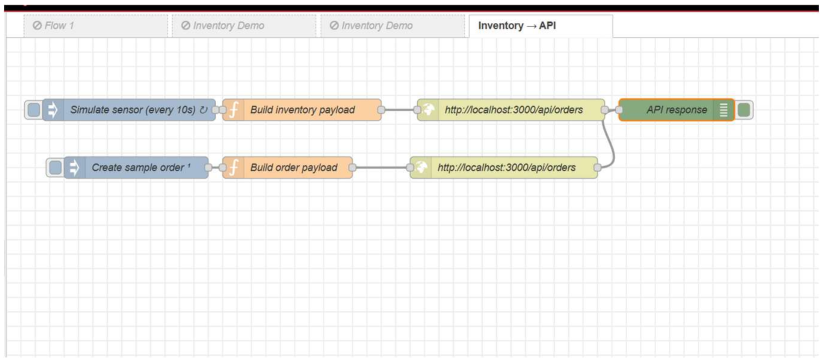
```
{"ok":true,"service":"inventory-service"}
```

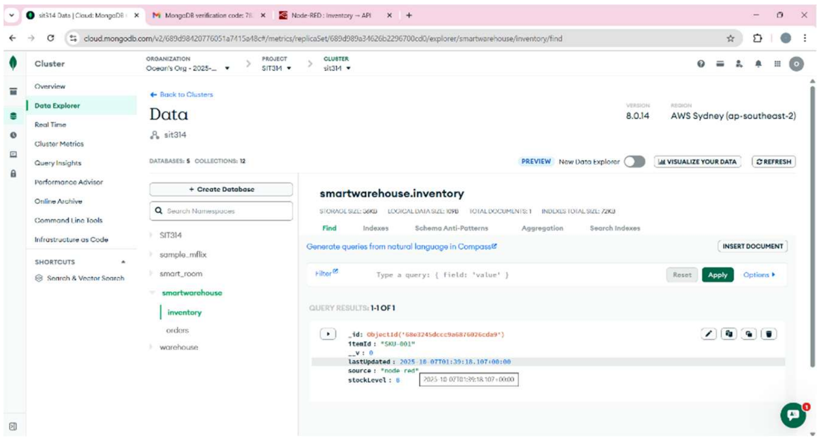


```
C:\Users\ocean\OneDrive\Pictures\Screenshots\SIT 314\Ta
sk 6\Documents\smart-warehouse\services\inventory>npm s
tart

> inventory-service@1.0.0 start
> node server.js

MongoDB connected
API running on http://localhost:3000
```



```
10/7/2025, 12:32:07 PM
node: API response
msg : Object
▸{ payload: object,
_msgid: "c99a66cfc1e93f6a",
headers: object,
statusCode: 200,
responseUrl:
"http://localhost:3000/api/
inve…" … }
10/7/2025, 12:32:17 PM
node: API response
msg : Object
▸{ payload: object,
_msgid: "452e7455e4e019f5",
headers: object,
statusCode: 200,
responseUrl:
"http://localhost:3000/api/
inve…" … }
10/7/2025, 12:32:27 PM
node: API response
msg : Object
▸{ payload: object,
_msgid: "5dcfc514d9a0be3f",
headers: object,
statusCode: 200,
responseUrl:
"http://localhost:3000/api/
inve…" … }
```



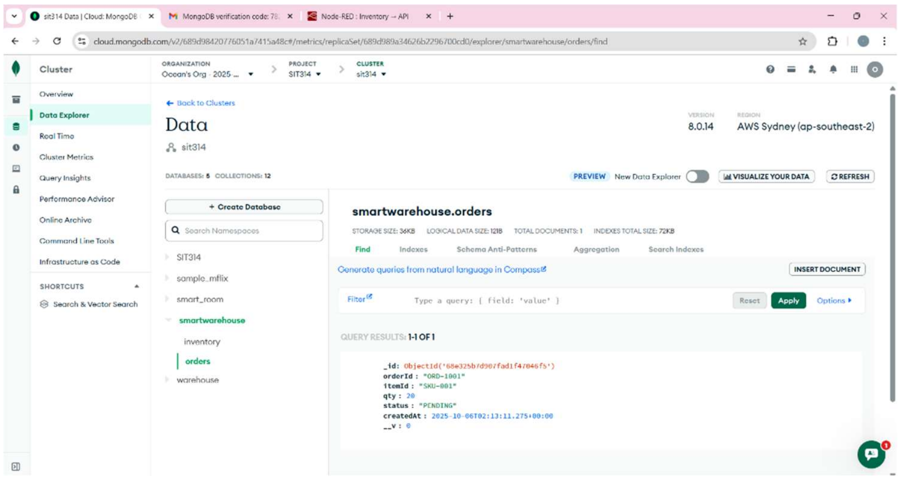


smartwarehouse.inventory

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 109B TOTAL DOCUMENTS: 1 INDEXES TOTAL SIZE: 72KB

QUERY RESULTS: 1-1 OF 1

```
_id: ObjectId('68e3245dccc9a6876026cda9')
itemId : "SKU-001"
__v : 0
lastUpdated : 2025-10-07T01:39:18.107+00:00
source : "node-red"
stockLevel : 8
```

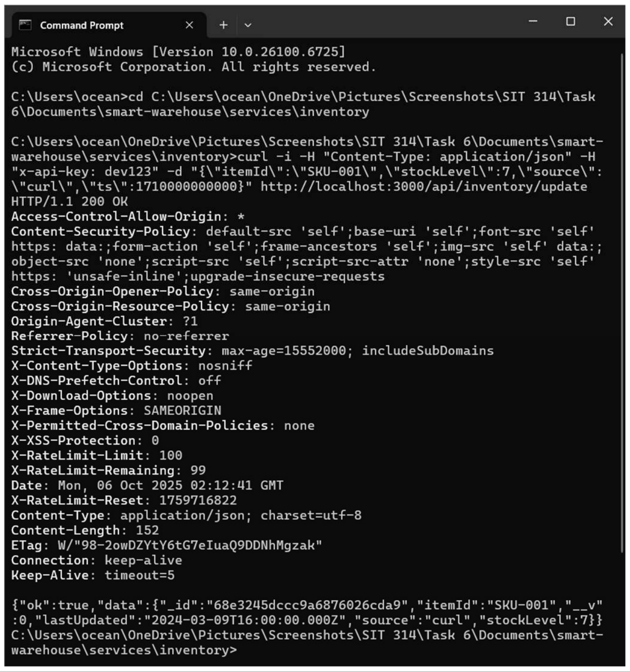


smartwarehouse.orders

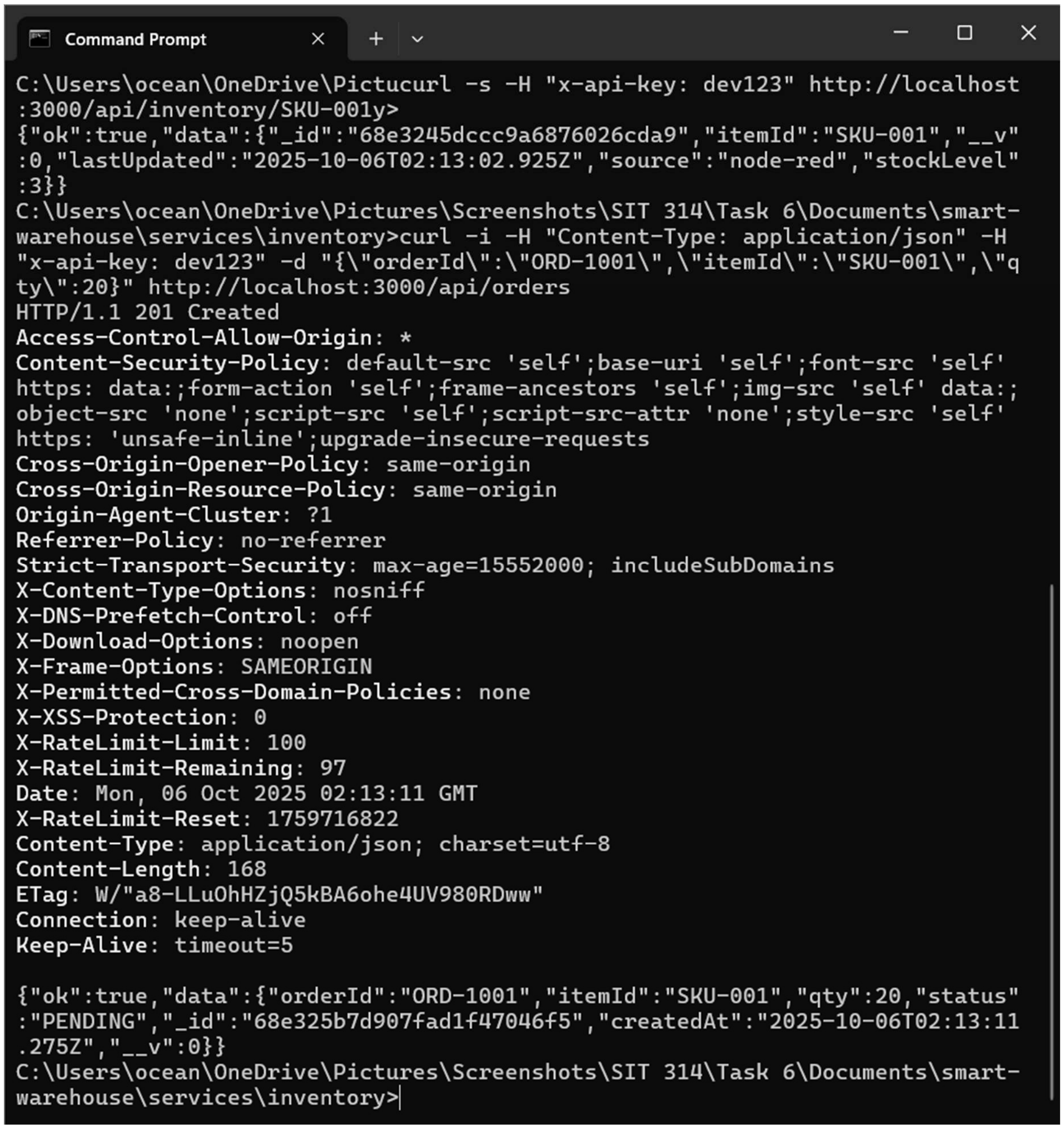
STORAGE SIZE: 36KB LOGICAL DATA SIZE: 121B TOTAL DOCUMENTS: 1 INDEXES TOTAL SIZE: 72KB

QUERY RESULTS: 1-1 OF 1

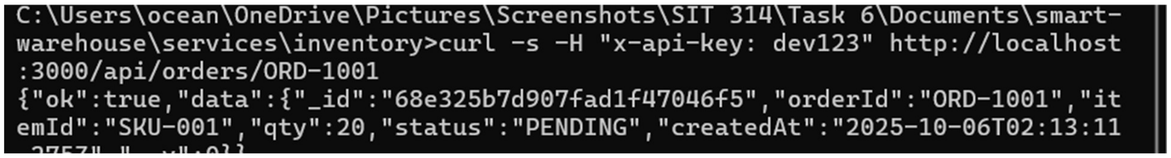
```
_id: ObjectId('68e325b7d907fad1f47046f5')
orderId : "ORD-1001"
itemId : "SKU-001"
qty : 20
status : "PENDING"
createdAt : 2025-10-06T02:13:11.275+00:00
__v : 0
```



```
Microsoft Windows [Version 10.0.26100.6725]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ocean>cd C:\Users\ocean\OneDrive\Pictures\Screenshots\SIT 314\Task
6\Documents\smart-warehouse\services\inventory

C:\Users\ocean\OneDrive\Pictures\Screenshots\SIT 314\Task 6\Documents\smart-
warehouse\services\inventory>curl -i -H "Content-Type: application/json" -H
"x-api-key: dev123" -d "{\"itemId\":\"SKU-001\",\"stockLevel\":7,\"source\":
\"curl\",\"ts\":1710000000000}" http://localhost:3000/api/inventory/update
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Security-Policy: default-src 'self';base-uri 'self';font-src 'self'
https: data:;form-action 'self';frame-ancestors 'self';img-src 'self' data:;
object-src 'none';script-src 'self';script-src-attr 'none';style-src 'self'
https: 'unsafe-inline';upgrade-insecure-requests
Cross-Origin-Opener-Policy: same-origin
Cross-Origin-Resource-Policy: same-origin
Origin-Agent-Cluster: ?1
Referrer-Policy: no-referrer
Strict-Transport-Security: max-age=15552000; includeSubDomains
X-Content-Type-Options: nosniff
X-DNS-Prefetch-Control: off
X-Download-Options: noopen
X-Frame-Options: SAMEORIGIN
X-Permitted-Cross-Domain-Policies: none
X-XSS-Protection: 0
X-RateLimit-Limit: 100
X-RateLimit-Remaining: 99
Date: Mon, 06 Oct 2025 02:12:41 GMT
X-RateLimit-Reset: 1759716822
Content-Type: application/json; charset=utf-8
Content-Length: 152
ETag: W/"98-2owDZYtY6tG7eIuaQ9DDNhMgzak"
Connection: keep-alive
Keep-Alive: timeout=5

{"ok":true,"data":{"_id":"68e3245dccc9a6876026cda9","itemId":"SKU-001","__v"
:0,"lastUpdated":"2024-03-09T16:00:00.000Z","source":"curl","stockLevel":7}}
C:\Users\ocean\OneDrive\Pictures\Screenshots\SIT 314\Task 6\Documents\smart-
warehouse\services\inventory>
```



```
C:\Users\ocean\OneDrive\Pictucurl -s -H "x-api-key: dev123" http://localhost
:3000/api/inventory/SKU-001y>
{"ok":true,"data":{"_id":"68e3245dccc9a6876026cda9","itemId":"SKU-001","__v"
:0,"lastUpdated":"2025-10-06T02:13:02.925Z","source":"node-red","stockLevel"
:3}}
C:\Users\ocean\OneDrive\Pictures\Screenshots\SIT 314\Task 6\Documents\smart-
warehouse\services\inventory>curl -i -H "Content-Type: application/json" -H
"x-api-key: dev123" -d "{\"orderId\":\"ORD-1001\",\"itemId\":\"SKU-001\",\"q
ty\":20}" http://localhost:3000/api/orders
HTTP/1.1 201 Created
Access-Control-Allow-Origin: *
Content-Security-Policy: default-src 'self';base-uri 'self';font-src 'self'
https: data:;form-action 'self';frame-ancestors 'self';img-src 'self' data:;
object-src 'none';script-src 'self';script-src-attr 'none';style-src 'self'
https: 'unsafe-inline';upgrade-insecure-requests
Cross-Origin-Opener-Policy: same-origin
Cross-Origin-Resource-Policy: same-origin
Origin-Agent-Cluster: ?1
Referrer-Policy: no-referrer
Strict-Transport-Security: max-age=15552000; includeSubDomains
X-Content-Type-Options: nosniff
X-DNS-Prefetch-Control: off
X-Download-Options: noopen
X-Frame-Options: SAMEORIGIN
X-Permitted-Cross-Domain-Policies: none
X-XSS-Protection: 0
X-RateLimit-Limit: 100
X-RateLimit-Remaining: 97
Date: Mon, 06 Oct 2025 02:13:11 GMT
X-RateLimit-Reset: 1759716822
Content-Type: application/json; charset=utf-8
Content-Length: 168
ETag: W/"a8-LLuOhHZjQ5kBA6ohe4UV980RDww"
Connection: keep-alive
Keep-Alive: timeout=5

{"ok":true,"data":{"orderId":"ORD-1001","itemId":"SKU-001","qty":20,"status"
:"PENDING","_id":"68e325b7d907fad1f47046f5","createdAt":"2025-10-06T02:13:11
.275Z","__v":0}}
C:\Users\ocean\OneDrive\Pictures\Screenshots\SIT 314\Task 6\Documents\smart-
warehouse\services\inventory>
```



```
C:\Users\ocean\OneDrive\Pictures\Screenshots\SIT 314\Task 6\Documents\smart-
warehouse\services\inventory>curl -s -H "x-api-key: dev123" http://localhost
:3000/api/orders/ORD-1001
{"ok":true,"data":{"_id":"68e325b7d907fad1f47046f5","orderId":"ORD-1001","it
emId":"SKU-001","qty":20,"status":"PENDING","createdAt":"2025-10-06T02:13:11
```

7. Challenges and Resolutions

The main challenge was ensuring the requests from Node-RED matched what the API expected. Early attempts used slightly different field names and missed the required header, which caused rejections. This was solved by standardising the field names, adding the API-key header to every request, and confirming success in the Debug panel. Another small challenge was verifying persistence; this was addressed by checking the documents directly in MongoDB Atlas and then reading the same records back through the service to prove round-trip accuracy.

8. Conclusion

The project meets the brief: Node-RED sends data to an API that sits in front of a cloud database, and the API exposes simple RESTful access to that data. The Inventory and Orders features are implemented, tested, and evidenced, showing both write and read paths. The design is small, clean, and easy to extend, which makes it a good base for adding delivery features, deploying publicly, and hardening the service for production.