# 📦 Supply Chain Disruption Tracker

**Business Plan & Hackathon Proposal**

---

## 1. Executive Summary

Global supply chains face delays due to weather, port congestion, labor strikes, geopolitical risks, and transportation inefficiencies. Today, logistics managers rely on fragmented dashboards and delayed reports.

The **Supply Chain Disruption Tracker (SCDT)** integrates **Fivetran** + **Google Cloud AI** to unify multiple real-time data sources into one predictive platform.

- **Fivetran**: Builds connectors for shipping APIs, customs/port authority feeds, and live weather data.
- **Google Cloud (BigQuery + Vertex AI + Gemini/AgentSpace)**: Stores, analyzes, and powers AI-driven predictions and conversational insights.
- **Elastic (optional extension)**: Enables semantic search over unstructured documents like customs advisories or news reports.

The result: logistics teams can **predict delays before they happen** and make smarter routing, sourcing, and fulfillment decisions.

---

## 2. Problem Statement

- **Delays = $$$**: Global supply chain disruptions cost $184B+ annually.
- **Fragmented visibility**: Shippers juggle 10+ dashboards (port congestion, carriers, weather).
- **Reactive decisions**: Most interventions happen **after** delays cause cost overruns.

There's no single, AI-driven, **proactive disruption alerting system** for SMBs and mid-market logistics operators.

---

# 3. Solution Overview

**Supply Chain Disruption Tracker** → A **conversational AI dashboard** for predictive logistics.

**Features:**

1. **Data Ingestion (Fivetran Connectors)**
    - Port congestion APIs
    - Vessel tracking data (e.g., MarineTraffic)
    - Weather data (NOAA/Accuweather APIs)
    - Customs/Trade feeds
    - ERP/Order systems
2. **Data Warehouse (Google BigQuery)**
    - Unified structured dataset: shipments, risks, ETAs, costs.
3. **AI Predictions (Vertex AI)**
    - Predict shipment delays (classification model).
    - Forecast arrival windows with confidence scoring.
    - Scenario simulations (reroute via alternate port, cost impact).
4. **Conversational Agent (Gemini / AgentSpace)**
    - Natural questions like:
        - "Which shipments are most at risk this week?"
        - "What's the cost impact if LA port backlog continues?"

    - Generates insights with charts + recommended actions.

5. **Dashboard UI**
    - Risk heatmap by geography.
    - Alert feed with disruption severity (low, medium, high).
    - Drill-down by shipment/customer.

---

# 4. Target Market

- **Primary:**
    - Freight forwarders
    - Logistics service providers
    - SMB importers/exporters (who lack enterprise-grade tools)
- **Secondary:**
    - E-commerce retailers with global sourcing
    - Manufacturing supply chain teams
    - Maritime/aviation logistics platforms

**Market Size:**

- Global supply chain management software = **$32B+ (2025)**, growing 11% CAGR.
- Mid-market segment underserved by big players (Oracle, SAP, Blue Yonder).

---

# 5. Competitive Advantage

| Existing Tools | Weaknesses | SCDT Edge |
|---|---|---|
| SAP/Oracle SCM | Expensive, enterprise-only | Hackathon MVP lightweight + accessible |
| Project44, FourKites | Limited data sources | Flexible Fivetran connectors |
| In-house dashboards | No AI predictions | Vertex AI predictive analytics |
| Weather-only alerts | Narrow scope | Multi-source + proactive risk modeling |

**Key differentiator:** SCDT is **proactive** and **AI-driven**, not just a reporting tool.

---

# 6. Business Model

- **Freemium SaaS**:
    - Free tier → 2 connectors + 1 predictive model.
    - Pro ($499/month) → 10 connectors, predictive models, alerting.
    - Enterprise (custom) → unlimited connectors, SLA, integrations.
- **Add-on revenue streams:**
    - Per-connector marketplace pricing.
    - API access for integration into ERPs/CRMs.
    - White-label dashboards for freight forwarders.

---

# 7. Go-To-Market Strategy

1. **Hackathon Demo → Proof of Concept**
    - Connect weather + vessel tracking APIs.
    - Predict delays for sample shipments.

    - Build chatbot that answers “Will my cargo be late?”

2. **Pilot Phase**
    - Partner with regional freight forwarders (5–10 customers).
    - Integrate 2–3 real connectors.
    - Demonstrate cost savings.
3. **Scaling**
    - API marketplace for connectors (community contributions).
    - Integrate Elastic for semantic risk analysis from news/docs.
    - Expand into aviation, trucking, and rail logistics.

---

# 8. Tech Stack

- **Fivetran Connector SDK** → Custom connectors (shipping, weather).
- **Google BigQuery** → Central data warehouse.
- **Vertex AI** → Predictive delay models (classification + regression).
- **Gemini / AgentSpace** → Conversational agent + recommendation engine.
- **Cloud Run / GKE** → Backend services.
- **Streamlit / React** → Dashboard UI.
- **(Optional)** Elastic → Hybrid search across customs advisories, news.

---

# 9. Roadmap

**Hackathon MVP (1–2 days):**

- Build 1–2 connectors (weather + shipping).
- Load into BigQuery.
- Train quick Vertex AI delay model.
- Build Gemini chatbot with “shipment at risk?” queries.
- Simple dashboard with shipment map + risk scores.

**3-Month Post-Hackathon:**

- Expand connectors (ERP, customs, multiple ports).
- Add proactive notifications.
- Onboard pilot customers.

**12-Month Vision:**

- Global disruption prediction network.
- API + marketplace for supply chain AI insights.
- Partner integrations (Shopify, Oracle NetSuite, etc.).

---

# 10. Impact & Hackathon Pitch Angle

- **Judges Love**: Combines **Fivetran + Vertex AI + Gemini** → checks all hackathon partner boxes.
- **Demo wow-factor**: Live chatbot + map visualization showing cargo “at risk.”
- **Impact narrative**: Helps small and mid-sized businesses save money, reduce uncertainty, and stay competitive in a volatile global supply chain.

---

⚡ **One-liner pitch for demo:**
*"We built a conversational AI that predicts shipment delays before they happen. With just two connectors, we turned weather + shipping data into a proactive disruption tracker that can save businesses millions."*

---

Do you want me to **draft a step-by-step build guide** (architecture diagram + hackathon implementation steps) so your team can execute this in 24–48 hours?

**You said:**
draft a step-by-step build guide (architecture diagram + hackathon implementation steps)

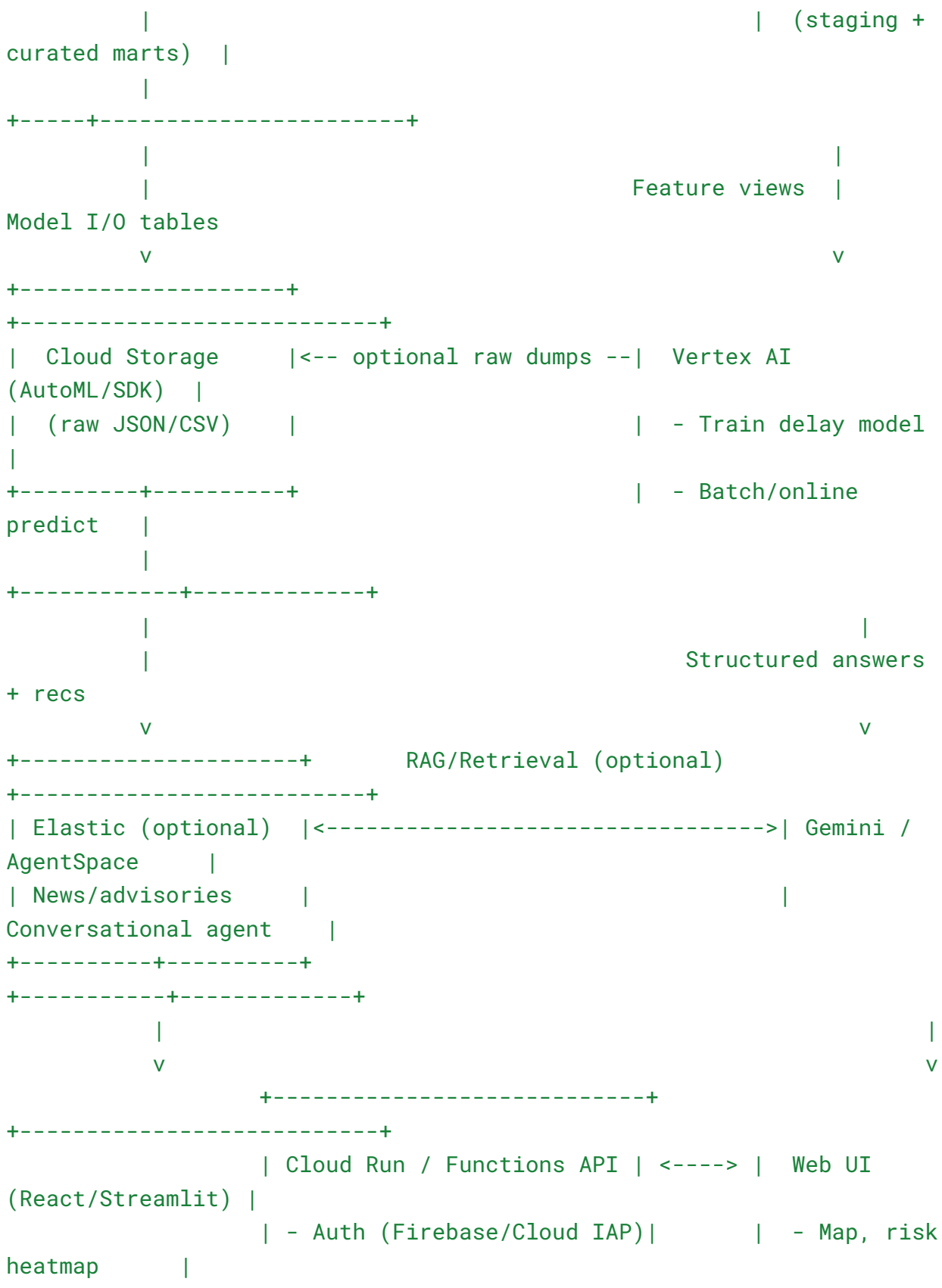
**ChatGPT said:**

Awesome—here's a **step-by-step build guide** you can follow to ship the **Supply Chain Disruption Tracker** in a hackathon window (24–48 hours). It includes an ASCII **architecture diagram**, precise **implementation steps**, **schemas**, **starter code snippets**, and a **demo script** so you can demo smoothly.

---

# 📐 Architecture (High Level)

```
+--------------------+        Fivetran Connector SDK
+-------------------+
| External Sources   |  --->  (custom connectors as Docker) |
Fivetran Managed |
| - Weather API      |                                      |
Pipelines   |
| - Vessel/Port API  |
+-------------------+
| - Demo ERP (CSV)   |
+---------+----------+
|
          |
v
          |
+-----------------------------+
          |                                          |       Google
BigQuery       |
```


curated marts)
Feature views
Model I/O tables
Cloud Storage
(raw JSON/CSV)
<-- optional raw dumps --
Vertex AI
(AutoML/SDK)
- Train delay model
- Batch/online predict
Structured answers + recs
RAG/Retrieval (optional)
Elastic (optional)
News/advisories
Gemini / AgentSpace
Conversational agent
Cloud Run / Functions API
- Auth (Firebase/Cloud IAP)
<---->
Web UI
(React/Streamlit)
- Map, risk heatmap

```
                  | - REST for predict/query   |       |  - “Which shipments…?”    |
                  +----------------------------+       +---------------------------+
```

---

# 🧰 Prereqs & Setup (1–2 hours)

1. **Create GCP Project**
    - Enable: BigQuery, Vertex AI, Cloud Run, Cloud Storage, Secret Manager, Firebase (if using), IAM.
2. **Service Accounts & Secrets**
    - `fivetran-connector-sa` (read external APIs, write to GCS/BigQuery).
    - `vertex-trainer-sa` (BigQuery read, Vertex admin).
    - Store API keys (weather, vessel) in **Secret Manager**.
3. **BigQuery Datasets**
    - `supplychain_stg` (staging/raw), `supplychain_curated` (cleaned/feature), `supplychain_ml` (training/preds).
4. **Local Dev**
    - Node.js or Python, Docker, gcloud CLI, Fivetran Connector SDK installed.
5. **(Optional)** Firebase project for web auth & hosting (or host React on Cloud Run).

---

# 🗄️ Data Model & Schemas

**Staging (raw):**

- supplychain_stg.weather_raw
    - ingested_at TIMESTAMP, location STRING, forecast_time TIMESTAMP, wind_kts FLOAT, precip_mm FLOAT, storm_flag BOOL, json_payload STRING
- supplychain_stg.vessel_positions_raw
    - ingested_at TIMESTAMP, vessel_id STRING, lat FLOAT, lon FLOAT, speed FLOAT, dest_port STRING, eta TIMESTAMP, status STRING, json_payload STRING
- supplychain_stg.port_congestion_raw
    - ingested_at TIMESTAMP, port_code STRING, queue_length INT64, avg_wait_hours FLOAT, last_update TIMESTAMP, json_payload STRING
- supplychain_stg.shipments_raw
    - shipment_id STRING, origin_port STRING, dest_port STRING, etd TIMESTAMP, eta_planned TIMESTAMP, carrier STRING, incoterm STRING, value_usd FLOAT

**Curated / Feature:**

- supplychain_curated.shipments_enriched
    - shipment_id, origin_port, dest_port, etd, eta_planned, carrier, value_usd
    - route_distance_nm FLOAT, dest_congestion FLOAT, weather_risk FLOAT, seasonal_index FLOAT
- supplychain_ml.training_labels

- `shipment_id, delayed_label BOOL, delay_hours FLOAT` *(from historical or synthetic labels)*

**Predictions/Serving:**

- `supplychain_ml.predictions`
  - `run_id STRING, shipment_id STRING, delayed_prob FLOAT, predicted_delay_hours FLOAT, generated_at TIMESTAMP`

---

# 🔌 Step 1 — Build Custom Fivetran Connectors (3–4 hours)

**Goal:** Ingest weather + vessel/port congestion data into BigQuery/GCS using Fivetran Connector SDK.

- **Connector 1: Weather (NOAA/Open-Meteo demo)**
  - Poll forecast endpoint for top 10 port geos.
  - Transform to tabular JSON (flatten nested).
  - Output: write to GCS (`gs://bucket/weather/date=.../*.json`) OR stream to BigQuery.
- **Connector 2: Vessel/Port**
  - Poll port congestion/vessel ETA endpoints (or demo API with synthetic).
  - Output similar to above.

**Skeleton (Python)**

```
# connectors/weather_connector/main.py
import os, requests, json, datetime as dt
from google.cloud import storage, bigquery
```

```
API = os.environ["WEATHER_API"]
PORTS = [{"code":"USLAX","lat":33.74,"lon":-118.26},
{"code":"CNSHA","lat":31.23,"lon":121.49}]

def fetch_weather(lat, lon):
    r =
requests.get(f"https://api.open-meteo.com/v1/forecast?latitude={lat}&l
ongitude={lon}&hourly=precipitation,wind_speed_10m")
    r.raise_for_status()
    return r.json()

def to_rows(port, payload):
    rows = []
    now = dt.datetime.utcnow()
    for i, t in enumerate(payload["hourly"]["time"]):
        rows.append({
          "ingested_at": now.isoformat(),
          "location": port["code"],
          "forecast_time": t,
          "wind_kts": payload["hourly"]["wind_speed_10m"][i] *
0.539957,
          "precip_mm": payload["hourly"]["precipitation"][i],
          "storm_flag": payload["hourly"]["wind_speed_10m"][i] > 15 or
payload["hourly"]["precipitation"][i] > 5,
          "json_payload": json.dumps(payload)[:5000]
        })
    return rows

def write_bq(rows, table="supplychain_stg.weather_raw"):
    bq = bigquery.Client()
    errors = bq.insert_rows_json(table, rows)
    if errors: raise RuntimeError(errors)

def handler(_event=None, _ctx=None):
    for p in PORTS:
        payload = fetch_weather(p["lat"], p["lon"])
        rows = to_rows(p, payload)
```

```
        write_bq(rows)

if __name__ == "__main__":
    handler()
```

- Package as Docker, register with **Fivetran Connector SDK**, configure schedule (e.g., every 1–3 hours).
- Repeat pattern for vessel/port API.

> **Tip:** If external APIs are hard to get, generate **synthetic JSON** on a timer so your pipeline still runs.

---

# 🧹 Step 2 — Transform to Features (1–2 hours)

Create **dbt** or **BigQuery SQL views** to join staging into enriched features.

```
-- supplychain_curated.shipments_enriched (VIEW)
CREATE OR REPLACE VIEW
`project.supplychain_curated.shipments_enriched` AS
WITH last_weather AS (
  SELECT location, forecast_time, storm_flag, wind_kts, precip_mm,
         ROW_NUMBER() OVER (PARTITION BY location ORDER BY
forecast_time DESC) rn
  FROM `project.supplychain_stg.weather_raw`
)
, last_congestion AS (
  SELECT port_code, avg_wait_hours, queue_length,
         ROW_NUMBER() OVER (PARTITION BY port_code ORDER BY
last_update DESC) rn
  FROM `project.supplychain_stg.port_congestion_raw`
)
SELECT s.shipment_id, s.origin_port, s.dest_port, s.etd,
s.eta_planned, s.carrier, s.value_usd,
```

```
       COALESCE(c.avg_wait_hours,0) AS dest_congestion,
       CASE WHEN w.storm_flag THEN 1 ELSE 0 END AS weather_risk,
       SAFE_DIVIDE(c.queue_length, 50.0) AS seasonal_index,
       1000.0 AS route_distance_nm -- placeholder; compute via
Haversine if coords available
FROM `project.supplychain_stg.shipments_raw` s
LEFT JOIN last_congestion c ON c.port_code = s.dest_port AND c.rn=1
LEFT JOIN last_weather w ON w.location = s.dest_port AND w.rn=1;
```

**Labels (for quick training):**

- If you lack historical outcomes, create labels: `delayed_label = (dest_congestion>4 OR weather_risk=1)` and `delay_hours = dest_congestion*2 + weather_risk*6`.

---

# 🤖 Step 3 — Train the Model in Vertex AI (1–2 hours)

Use **Vertex AI Tabular** (AutoML) for speed, or SDK with XGBoost/Sklearn.

**Training table:** `project.supplychain_ml.train_set` (join enriched + labels)

```
CREATE OR REPLACE TABLE `project.supplychain_ml.train_set` AS
SELECT e.*, l.delayed_label, l.delay_hours
FROM `project.supplychain_curated.shipments_enriched` e
JOIN `project.supplychain_ml.training_labels` l USING (shipment_id);
```

**Python SDK (concise):**

```
from google.cloud import aiplatform

aiplatform.init(project="PROJECT_ID", location="us-central1")
ds = aiplatform.TabularDataset.create(
    display_name="supplychain_ds",
    bq_source="bq://PROJECT_ID.supplychain_ml.train_set"
```

```
)
model = aiplatform.AutoMLTabularTrainingJob(
    display_name="delay_model_job",
    optimization_prediction_type="classification",
    target_column="delayed_label",
    budget_milli_node_hours=1000
).run(dataset=ds, model_display_name="delay_model")
endpoint = model.deploy(machine_type="n1-standard-2")
```

(For regression `delay_hours`, spin a second model or predict both with multi-objective framework; for hackathons, predicting **probability of delay** is enough.)

---

# 🗣️ Step 4 — Conversational Agent (Gemini / AgentSpace) (1–2 hours)

**Prompt strategy (function-calling over BigQuery):**

- Tool: `get_at_risk_shipments(top_n, days_ahead)` → SQL over `predictions` + `shipments_enriched`.
- Tool: `what_if_reroute(shipment_id, alt_port)` → Recompute `dest_congestion` and estimate `delay_hours`.

**System Prompt (snippet):**

```
You are a logistics copilot. Be concise, cite the source table and
time.
Prefer lists with shipment_id, risk_score, reason
(weather/congestion).
Use the tools to fetch data; do not invent numbers. Offer a
recommended action.
```

**Tool pseudo-impl (Python FastAPI):**

```
@app.get("/api/at_risk")
```

```
def at_risk(top_n: int = 10):
    sql = """
    SELECT p.shipment_id, p.delayed_prob, e.dest_port,
e.dest_congestion, e.weather_risk, e.eta_planned
    FROM `project.supplychain_ml.predictions` p
    JOIN `project.supplychain_curated.shipments_enriched` e USING
(shipment_id)
    WHERE p.generated_at > TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL
6 HOUR)
    ORDER BY delayed_prob DESC
    LIMIT @n
    """
    rows = bq_query(sql, params={"n": top_n})
    return {"rows": rows}
```

Hook this endpoint as a **tool** inside Gemini/AgentSpace.

---

# 🖥️ Step 5 — Backend & UI (2–4 hours)

**Backend (Cloud Run / Functions):**

- Endpoints:
    - `GET /api/at_risk?top_n=10`
    - `GET /api/shipment/{id}`
    - `POST /api/predict` (optional online prediction)
- Auth: simplest is **Cloud IAP** or **Firebase Auth** bearer JWT.

**Web UI options:**

- **Streamlit** (fast) or **React + Vite** (polish).
- Pages:

1. **Overview:** KPIs (Total shipments, # at risk, avg predicted delay).
2. **Map/Heatmap:** color by `delayed_prob`.
3. **Alert Feed:** cards with reason & suggested action.
4. **Chat:** Gemini panel with tool buttons (Quick Actions).

**Streamlit starter:**

```
import streamlit as st
import pandas as pd
from google.cloud import bigquery

st.set_page_config(page_title="SCDT", layout="wide")
bq = bigquery.Client()

df = bq.query("""
SELECT p.shipment_id, p.delayed_prob, e.dest_port, e.dest_congestion,
e.weather_risk, e.eta_planned
FROM `project.supplychain_ml.predictions` p
JOIN `project.supplychain_curated.shipments_enriched` e USING
(shipment_id)
ORDER BY p.delayed_prob DESC LIMIT 50
""").to_dataframe()

st.metric("Shipments at Risk", (df.delayed_prob>0.5).sum())
st.dataframe(df)
```

---

# 🚀 Step 6 — Batch Predictions & Scheduling (30–60 min)

- **Cloud Scheduler** → triggers a **Cloud Run job** every hour:
    1. Materialize latest `shipments_enriched`.

2. Call **Vertex prediction** (batch) write results into `supplychain_ml.predictions`.

- Alternatively, do **online prediction** on demand (faster to demo).

---

# 🔒 Step 7 — Security & Secrets (30 min)

- Use **Secret Manager** for API keys.
- Service Accounts:
    - Backend → BigQuery read, Vertex predict.
    - Connectors → BigQuery write.
- Enable **CORS** on API, restrict to your web origin.
- If needed, gate whole site behind **Cloud IAP** for hackathon.

---

# 🎤 Demo Script (5–7 minutes)

1. **Problem (30s):** “Delays cost $$; teams react too late.”
2. **Arch Slide (30s):** Show diagram: *Fivetran → BigQuery → Vertex → Gemini → UI.*
3. **Live Data (60s):** Trigger connector run (or show logs) filling staging tables.
4. **Model (60s):** Show Vertex AI model card; run a quick prediction (console or API).
5. **Dashboard (2 min):**
    - Heatmap: highlight LA & Shanghai congestion.
    - Alert cards: top 5 at-risk shipments with reasons.

    - Click into one shipment → see features & probability.

6. **Chat (1–2 min):** Ask:
    - “Which shipments are most at risk this week?”
    - “What’s the cost if we reroute S123 to OAK instead of LAX?”
    - Gemini replies with **ranked list** + **action**.
7. **Close (30s):** Savings potential + next steps (pilots, connectors marketplace).

---

# 🧪 Test Data (if APIs are tricky)

Load **synthetic CSVs** to `supplychain_stg.shipments_raw` and `port_congestion_raw`.

**shipments_raw.csv**

```
shipment_id,origin_port,dest_port,etd,eta_planned,carrier,value_usd
S001,CNSHA,USLAX,2025-10-01T00:00:00Z,2025-10-10T00:00:00Z,MAEU,120000
S002,NLRTM,USNYC,2025-10-02T00:00:00Z,2025-10-12T00:00:00Z,HLAG,90000
```

**port_congestion_raw.csv**

```
ingested_at,port_code,queue_length,avg_wait_hours,last_update
2025-10-02T12:00:00Z,USLAX,35,18,2025-10-02T12:00:00Z
2025-10-02T12:00:00Z,USNYC,10,4,2025-10-02T12:00:00Z
```

---

# 🧭 Timeline (Aggressive 24–36 hours)

- **Hour 0–2:** Project, IAM, datasets, secrets.
- **Hour 2–6:** Fivetran custom connectors (weather + congestion) → BQ.

- **Hour 6–8:** SQL views & features; synthetic labels.
- **Hour 8–12:** Vertex AI AutoML training + deploy endpoint.
- **Hour 12–16:** Cloud Run API (predict/at_risk) + Streamlit UI.
- **Hour 16–20:** Gemini tool integration (two tools).
- **Hour 20–24:** Batch pred job + polish metrics & demo script.
- **Buffer:** Optional Elastic news search, map polish, auth.

---

# 🏆 Judging Checklist

- ✅ Uses **Fivetran Connector SDK** (custom connector)
- ✅ Data lands in **BigQuery**; clear schema & lineage
- ✅ **Vertex AI** model trained; endpoint deployed
- ✅ **Gemini/AgentSpace** conversational interface with tool calls
- ✅ Clean **UI** with map/heatmap + alert cards
- ✅ Clear **impact narrative** (time/cost savings; proactive decisions)

---

# 🌱 Stretch Goals (Nice-to-Have)

- **Elastic**: ingest maritime advisories/news; RAG to enrich explanations (“At risk due to strike at Port X, source: …”).
- **Scenario Simulator**: try alternate ports and carriers; recompute ETA and cost deltas.
- **Notifications**: Pub/Sub → email/Slack alerts for >70% risk.

🚢 **Vessel Tracking / AIS / Port / Voyage APIs**

| API | What they offer | Pricing / Free tier details |
|---|---|---|
| VesselFinder (AIS API / Vessel Positions / Port Calls) | Real-time AIS data, ship positions, port calls, voyage and master data. (VesselFinder API) | Uses a credit-based model; subscription or on-demand credits. (VesselFinder API) |
| MyShipTracking API | Real-time vessel positions, voyage info, historical tracks, port calls. (MyShipTracking) | You need to sign up for a free API key; they provide a free tier. |
| Datalastic | Vessel AIS data (historical + real-time) etc. (Datalastic) | Commercial; reach out for trial / pricing. |
| Searoutes Vessel API | Vessel position, trace over time, predictive ETA. (Searoutes) | Their docs specify endpoints for trace, position, ETA etc. |
| Portcast | All-in-one vessel tracking + schedule / predictive ETAs / port data. (Portcast) | Commercial; contact for access. |
| Sinay Hub Vessel API | AIS data by MMSI/IMO, port congestion, vessel details. (Sinay) | They offer a free plan (500 monthly calls) per documentation. |
| JSONCargo Vessel Tracking API | Real-time vessel location, speed, ETA, heading etc. (JsonCargo) | Requires API key; commercial. |

**Weather APIs**

| API | Notes / Advantages | Endpoint Examples / Resources |
|---|---|---|

| | | |
|---|---|---|
| **Open-Meteo** | **Free, open-source, no API key required for non-commercial use. (Open Meteo)** | **`https://api.open-meteo.com/v1/forecast?latitude={lat}&longitude={lon}&hourly=precipitation,wind_speed_10m`** |
| **WeatherAPI.com** | **Freemium; includes forecast, marine weather, historical data. (WeatherAPI)** | **Use their JSON endpoints (e.g. current, forecast) according to their docs.** |
| **OpenWeatherMap** | **Popular mainstream weather API; free tier (1,000 calls/day) (OpenWeatherMap)** | **One Call API (current + hourly + daily) etc.** |
| **Visual Crossing Weather API** | **Single-endpoint API with forecast, history, etc.** | **e.g. `https://weather.visualcrossing.com/VisualCrossingWebServices/rest/services/timeline/{location}?key={API_KEY}` (Visual Crossing)** |
| **Meteomatics** | **Offers free tier / trial, global coverage, many weather parameters. (Meteomatics)** | **Their docs show endpoints with parameters, e.g. weather forecasts, historical etc.** |

## 🚢 Examples of Port Congestion APIs & Providers

| API / Provider | What They Provide | Notes / Docs |
|---|---|---|
| Sinay Port Congestion API | Real-time congestion status per port; filter by vessel type; compare to average congestion. Sinay+2Sinay+2 | Endpoint `GET /congestion`; refresh hourly. Sinay |

| | | |
|---|---|---|
| **Spire Maritime – Port Congestion (GraphQL)** | **Real-time + historical congestion insights; "Port Congestion GraphQL API." Maritime Documentation** | **Use Spire's GraphQL interface. Maritime Documentation** |
| **Portcast Port Congestion API** | **Provides daily & weekly congestion metrics (waiting times, vessel counts, port-level congestion indices) for 1,000+ ports. Portcast+3Portcast+3kb.portcast.io+3** | **Use port UN/LOCODE + optional date ranges. Portcast+1** |
| **GoComet Port Congestion / Delay API** | **Real-time port congestion / delay data; integration into ERP; JSON / CSV / XML formats. GoComet+1** | **They state "GoComet's port congestion API can be integrated with any client ERP" GoComet** |
| **Signal Ocean – Port Congestion API** | **Historical + real-time congestion data for dry bulk & tanker vessels; vessel counts, wait times. thesignalgroup.com** | **Has a Python SDK option. thesignalgroup.com** |
| **Veson (Market Intelligence / Port Congestion)** | **Real-time & historical congestion / trade flow data, port turnaround, vessel counts. Veson Nautical** | **Typically enterprise / trade data suite. Veson Nautical** |
| **Datalastic – Ports Data / Vessel Traffic API** | **Ports metadata + vessel traffic (ships around port) — not strictly congestion, but can infer queue/traffic load. Datalastic** | **Useful as auxiliary congestion proxy. Datalastic** |
| **OneOcean Ports API (formerly LR OneOcean)** | **Ports data: terminal details, vessel counts, port handling metrics. OneOcean** | **More port metadata than congestion signal, but helpful. OneOcean** |

**Got it ✅ — here are 3 concrete APIs (with sample requests/responses) that are both practical for a hackathon and cover the Customs/Trade and ERP/Order Systems data**

**sources. These can slot directly into your Fivetran connector → BigQuery → Vertex AI flow.**

---

# 🛂 Customs / Trade API (Public & Free)

### 1.

### U.S. International Trade Administration (ITA) Data API

- **What it gives: Import/export stats, tariff rates, HS codes, trade events.**
- **Why good for hackathon: No auth barrier (API key easy to get), returns JSON you can enrich shipments with.**

**Endpoint Example:**

**GET https://api.trade.gov/v1/trade_leads/search?industry=Transportation&country=CN**

**Sample Response:**


```
{
  "results": [
    {
      "title": "Transport Services - China",
      "industry": "Transportation",
      "country": "China",
      "description": "Export opportunity for U.S. logistics providers.",
      "source": "ITA"
    }
  ]
}
```


**Hackathon Use: Enrich your shipments with trade alerts (e.g., “High demand for transport services from China”).**

---

### 2.

### U.S. Census Foreign Trade API

- **What it gives: Import/export trade values by commodity/country.**
- **Why good: Open data → easy to demo customs-level stats.**

**Endpoint Example:**

**GET https://api.census.gov/data/timeseries/intltrade/exports/hs?get=ALL_VAL_MO&time=2024-12&CTY_CODE=570**

**Sample Response:**

**[["ALL_VAL_MO","time","CTY_CODE"],
["120000000","2024-12","570"]]**

**Hackathon Use: Map shipments → commodity codes → enrich with "average monthly trade flows" for predictive features.**

---

# 🧾 ERP / Order System APIs

**3.**

## API2Cart (Unified OMS / ERP Connector)

- **What it gives: One API → pull orders, products, customers across Shopify, WooCommerce, Magento, etc.**
- **Why good: Simulates order feeds for shipments (JSON, consistent schema). Has free sandbox.**

**Endpoint Example:**

**GET https://api.api2cart.com/v1.1/order.list.json?api_key=YOUR_KEY&store_key=STORE_KEY**

**Sample Response:**

**{
"return_code":0,**

```
 "result":{
  "orders":[
   {
    "id":101,
    "status":"processing",
    "created_at":"2025-09-25T12:30:00Z",
    "total":250.75,
    "currency":"USD",
    "customer":{
     "id":55,"email":"jane@example.com"
    }
   }
  ]
 }
}
```


**Hackathon Use: Treat each order as a “shipment” → join with port + weather + congestion → predict risk/delay.**

---

# ⚡ How These Fit Your Build

- **ITA / Census APIs (Customs/Trade): Provide external risk/context features (commodity volumes, alerts, tariff trends).**
- **API2Cart (ERP/Orders): Provides internal company data (actual orders → shipments).**
- **Together → you now have internal + external signals to feed into BigQuery and enrich Vertex AI models.**

---

**👉 I can now draft a Python connector script for each (ITA trade + Census trade + API2Cart orders) that:**

1. **Fetches data,**
2. **Normalizes to JSON rows,**

3. **Inserts into BigQuery staging tables (supplychain_stg.trade_raw, supplychain_stg.orders_raw).**

**Do you want me to generate that code so you can drop it into your Fivetran Connector SDK setup?**