n8n workflow Details

Workflow Name : Skyways Dynamic Pricing Engine

1. **Workflow JSON export**

```
{
  "name": "Skyways Dynamic Pricing Engine",
  "nodes": [
   {
    "parameters": {
      "triggerTimes": {
       "item": [
        {
          "mode": "everyHour"
        }
       ]
      }
    },
    "name": "Cron Trigger",
    "type": "n8n-nodes-base.cron",
    "typeVersion": 1,
    "position": [
      0,
      0
    ],
    "id": "a62e638f-8c86-44d2-a684-72c5458d7b14"
   },
   {
    "parameters": {
      "sheetId": "17u8auNdco-2014Sqo_2Qjl7ZlhSVcznRDekN7dQoWvI",
      "range": "Sheet1!A:H",
      "dataStartRow": 2,
      "keyRow": 1,
      "options": {}
    },
    "name": "Get Containers",
    "type": "n8n-nodes-base.googleSheets",
    "typeVersion": 2,
    "position": [
      208,
      0
    ],
    "id": "864e8993-162b-4ee6-84db-759a641c8dc6",
    "alwaysOutputData": true,
    "credentials": {
      "googleSheetsOAuth2Api": {
        "id": "gw6OtNVknRJtqvyN",
        "name": "Google Sheets account"
      }
    }
  },
  {
    "parameters": {
      "functionCode": "return items.map(item => {\n\n  // ===== INPUT VALUES =====\n  const totalCost = parseFloat(item.json.total_cost) || 0;\n  const totalCBM = parseFloat(item.json.total_cbm) || 1;\n  const bookedCBM =
parseFloat(item.json.booked_cbm) || 0;\n  const pricingMode =
(item.json.pricing_mode || \"expected\").toLowerCase();\n  const season =
(item.json.season || \"normal\").toLowerCase();\n  const currencyRate =
parseFloat(item.json.currency_rate) || 1;\n  const daysToDeparture =
parseFloat(item.json.days_to_departure) || 15;\n  const demandIndex =
parseFloat(item.json.demand_index) || 1;\n\n  // ===== BASE MARGIN
=====\n  let baseMargin = 0.30;\n\n  if (pricingMode === \"conservative\")
baseMargin = 0.20;\n  if (pricingMode === \"aggressive\") baseMargin =
0.45;\n\n  // ===== SAFE CALCULATIONS =====\n  const fillRate = totalCBM >
0 ? bookedCBM / totalCBM : 0;\n  const costPerCBM = totalCBM > 0 ?
(totalCost * currencyRate) / totalCBM : 0;\n\n  // ===== DEMAND SCORE
=====\n  let demandScore = demandIndex;\n\n  if (fillRate > 0.8)
demandScore += 0.2;\n  if (fillRate < 0.5) demandScore -= 0.15;\n\n  if (season
=== \"high\") demandScore += 0.25;\n  if (season === \"low\") demandScore -
= 0.20;\n\n  if (daysToDeparture < 7) demandScore += 0.30;\n  if
(daysToDeparture > 30) demandScore -= 0.10;\n\n  demandScore =
Math.max(0.5, Math.min(demandScore, 1.8));\n\n  // ===== RISK SCORE
=====\n  let riskScore = 1;\n\n  if (fillRate < 0.4) riskScore += 0.25;\n  if
(daysToDeparture < 5 && fillRate < 0.7) riskScore += 0.20;\n  if (currencyRate >
1.1) riskScore += 0.10;\n\n  // ===== UTILIZATION =====\n  let
utilizationBoost = 1;\n\n  if (fillRate > 0.85) utilizationBoost = 1.15;\n  else if
(fillRate > 0.7) utilizationBoost = 1.08;\n  else if (fillRate < 0.5) utilizationBoost
= 0.95;\n\n  // ===== SMART MARGIN =====\n  let smartMargin = baseMargin
* demandScore * utilizationBoost * riskScore;\n\n  smartMargin =
Math.max(0.15, Math.min(smartMargin, 0.80));\n\n  // ===== DYNAMIC RATE
=====\n  const dynamicRate = costPerCBM * (1 + smartMargin);\n\n  //
===== PROFIT =====\n  const expectedRevenue = dynamicRate * totalCBM;\n
const expectedProfit = expectedRevenue - (totalCost * currencyRate);\n\n  //
===== SAFE PROFIT MARGIN =====\n  let profitMarginPercent = 0;\n\n  if
(expectedRevenue > 0) {\n    profitMarginPercent = (expectedProfit /
expectedRevenue) * 100;\n  }\n\n  // ===== RECOMMENDATION =====\n  let
recommendation = \"INCREASE PRICE\";\n\n  if (profitMarginPercent > 40)\n
recommendation = \"MAXIMIZE PROFIT\";\n  else if (profitMarginPercent >
25)\n    recommendation = \"OPTIMAL\";\n  else if (profitMarginPercent >
10)\n    recommendation = \"ACCEPTABLE\";\n\n  // ===== OUTPUT (SAFE
NUMBERS ONLY) =====\n  item.json.cost_per_cbm =
Number(costPerCBM.toFixed(2));\n  item.json.fill_rate =
Number(fillRate.toFixed(2));\n  item.json.smart_margin =
Number(smartMargin.toFixed(2));\n  item.json.dynamic_rate =
Number(dynamicRate.toFixed(2));\n  item.json.expected_revenue =
Number(expectedRevenue.toFixed(2));\n  item.json.expected_profit =
Number(expectedProfit.toFixed(2));\n  item.json.profit_margin_percent =
Number(profitMarginPercent.toFixed(2));\n  item.json.demand_score =
Number(demandScore.toFixed(2));\n  item.json.risk_score =
Number(riskScore.toFixed(2));\n  item.json.recommendation = recommendation;\n\n  return item;\n});\n"
    },
    "name": "Calculate Dynamic Pricing",
    "type": "n8n-nodes-base.function",
    "typeVersion": 1,
    "position": [
      400,
      0
    ],
    "id": "75aaafa9-f995-4a70-9432-40f4e0813680"
  },
  {
    "parameters": {
      "operation": "append",
      "sheetId": "1VVC56RY1Jx-hVXw2oVJWPfH8e11vafvQJWJNAsZ80rM",
      "range": "Sheet1!A:Z",
      "options": {}
    },
    "name": "Update Dashboard",
    "type": "n8n-nodes-base.googleSheets",
    "typeVersion": 2,
    "position": [
      688,
      0
    ],
    "id": "8fece6dd-31e6-4964-801d-a9ae4c8f6956",
    "credentials": {
        "googleSheetsOAuth2Api": {
          "id": "gw6OtNVknRJtqvyN",
          "name": "Google Sheets account"
        }
      }
    }
  ],
  "pinData": {},
  "connections": {
    "Cron Trigger": {
      "main": [
        [
          {
            "node": "Get Containers",
            "type": "main",
            "index": 0
          }
        ]
      ]
    },
    "Get Containers": {
      "main": [
        [
          {
            "node": "Calculate Dynamic Pricing",
            "type": "main",
            "index": 0
          }
        ]
      ]
    },
    "Calculate Dynamic Pricing": {
      "main": [
        [
          {
            "node": "Update Dashboard",
            "type": "main",
            "index": 0
          }
        ]
      ]
    }
  },
  "active": false,
  "settings": {
    "executionOrder": "v1",
    "binaryMode": "separate",
    "availableInMCP": false
  },
  "versionId": "8841d274-2945-43e7-82c2-53848d495add",
  "meta": {
    "templateCredsSetupCompleted": true,
    "instanceId": "f4cb3db98fbda777c86005d0ea31bd11568b6d747a2469cf48246b0b31449e92"
  },
  "id": "HkWNH25Hw3jW7um3",
  "tags": []
}
```

2. **Purpose of workflow**

   This workflow automatically calculates dynamic container pricing based on cost, demand, risk, and fill rate. It retrieves container data from a Google Sheet, applies a smart pricing algorithm, and updates the dashboard sheet with calculated pricing, expected revenue, and profit metrics.

3. **Trigger type (Webhook/Cron/App)**

   - **Schedule Timing:**
     Every hour (recommended for real-time pricing)
   - The Cron Trigger is configured to start the workflow execution and initiate the following sequence:

   1. Fetch container data from Google Sheets
   2. Calculate dynamic pricing using predefined logic
   3. Update the dashboard with calculated pricing and profit metrics

4. **List of all credentials used**

- The workflow uses secure credentials to connect to external services.
- **Credential Used:** Google Sheets OAuth2 Credential
- **Purpose**:
- To securely access the container data sheet
- To write calculated pricing results into the dashboard sheet
- The credential allows the workflow to authenticate and interact with Google Sheets without exposing sensitive login information.

Credential Security:

- Credentials are stored securely in n8n.
- Authentication tokens are encrypted.
- No passwords are stored in plain text.

5. **Link of sheet/ files if any**

The workflow uses Google Sheets as both the input data source and output dashboard.

Input Sheet: Container Data Sheet

Contains:

- Container ID
- Total container cost
- Total CBM (container volume)
- Booked CBM
- Demand score
- Risk score
- Season type

Example structure:

| container_id | total_cost | total_cbm | booked_cbm | demand_score | risk_score |

6. Required environment variables

This workflow does not require mandatory environment variables for basic operation because it uses built-in n8n credentials (Google Sheets OAuth2). However, environment variables can be used to improve security, flexibility, and deployment portability.

7. Webhook path + method (if any)

- Not applicable.
- This workflow does not use a Webhook Trigger.
- Webhook triggers are used when workflows must start based on external events such as API calls, but this workflow uses scheduled automation instead.

8. Schedule timing (if Cron)

- The workflow runs automatically based on Cron scheduling configuration.

  Every 1 hour

- This ensures pricing is updated regularly based on latest container and booking data.
- Alternative schedules may include:

- Every day at fixed time
- Every 30 minutes
- Every 6 hours

9. Sample input data

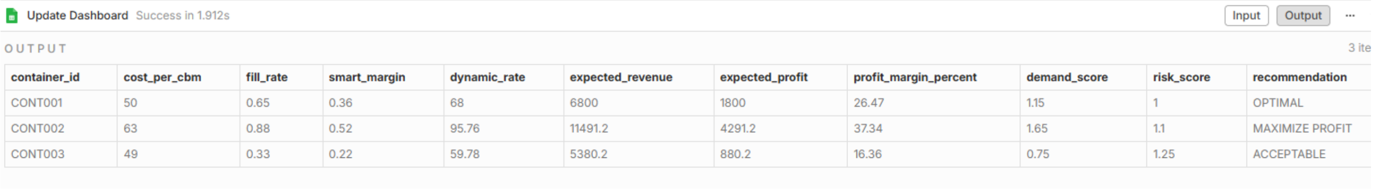
The workflow reads container data from the Google Sheets input file. Each row represents one container with operational and pricing parameters.

Description of fields:

- container_id: Unique container identifier
- total_cost: Total shipment cost
- total_cbm: Total container volume capacity
- booked_cbm: Volume already booked
- pricing_mode: Strategy type (conservative, expected, aggressive)
- season: Market season (low, normal, high)
- currency_rate: Exchange rate applied to cost
- days_to_departure: Remaining days before shipment
- demand_index: Demand indicator value

10. Expected final output/action

After processing, the workflow calculates optimized pricing and financial metrics and writes them to the dashboard sheet.



container_id	cost_per_cbm	fill_rate	smart_margin	dynamic_rate	expected_revenue	expected_profit	profit_margin_percent	demand_score	risk_score	recommendation
CONT001	50	0.65	0.36	68	6800	1800	26.47	1.15	1	OPTIMAL
CONT002	63	0.88	0.52	95.76	11491.2	4291.2	37.34	1.65	1.1	MAXIMIZE PROFIT
CONT003	49	0.33	0.22	59.78	5380.2	880.2	16.36	0.75	1.25	ACCEPTABLE

11. Error handling behaviour

The workflow includes built-in safeguards and error prevention mechanisms:

1. Missing values handling
   - Default values are applied if input is missing
   - Prevents calculation failures
2. Division-by-zero protection
   - total_cbm is validated before calculations
3. Value limits applied
   - Margin limited between 15% and 80%
   - Demand score constrained between 0.5 and 1.8
4. Credential protection
   - Secure OAuth authentication prevents access failures

5. Always Output Data enabled
    - Ensures workflow continues even if sheet is temporarily empty

    Result: Workflow runs reliably without crashes.

12. External system dependencies

The workflow depends on the following external systems:

1. n8n Automation Platform
   Purpose: Workflow execution engine
2. Google Sheets (Input Sheet)
   Purpose: Container operational data source
3. Google Sheets (Dashboard Sheet)
   Purpose: Store pricing results
4. Google OAuth2 Authentication
   Purpose: Secure access to sheets
5. Internet Connectivity
   Purpose: Communication between n8n and Google Sheets

13. Steps to test after deployment

Step 1: Prepare test data
Add sample container rows in the input Google Sheet.

Step 2: Activate workflow
Enable the workflow in n8n.

Step 3: Run manual test
Click "Execute Workflow" in n8n.

Step 4: Verify execution
Confirm workflow runs without errors.

Step 5: Check dashboard sheet
Verify new calculated columns appear:

- dynamic_rate
- expected_profit
- recommendation

Step 6: Validate calculations
Compare expected output with manual calculation.

Step 7: Test automation
Wait for next Cron run (1 hour) and confirm automatic execution.

Step 8: Monitor logs
Check n8n execution history for successful runs.