The Hebrew FinTech Informant (HFI): A Technical Workbook for Autonomous Content Intelligence

1. Executive Summary: The Case for Sovereign Automation

In the high-velocity ecosystem of financial technology (FinTech) journalism, the arbitrage of information—speed, accuracy, and unique localization—determines audience capture. For a Hebrew-language content creator operating on X (formerly Twitter), the operational friction of manually monitoring Anglo-centric sources (Wall Street Journal, Reuters, TechCrunch), selecting relevant narratives, transcreating them into culturally resonant Hebrew, and acquiring high-fidelity media assets creates a bottleneck that limits scale. This report outlines the architectural specification and implementation workbook for the "Hebrew FinTech Informant" (HFI), a self-hosted, microservices-based application designed to automate this workflow.

The prevailing reliance on manual curation is increasingly untenable due to the fragmentation of digital platforms and the aggressive anti-scraping measures implemented by X.[1] To circumvent the prohibitive costs of the official X API (starting at $42,000 annually for enterprise tiers) and the limitations of free tools, this workbook proposes a custom-built solution leveraging **Python-based headless browser automation (Playwright)**, **container orchestration (Kubernetes/K3s)**, and **Generative AI (LLMs)** for stylistic rewriting. This system shifts the creator's role from "hunter-gatherer" to "editor-in-chief," utilizing a "Human-in-the-Loop" (HITL) architecture where automation handles ingestion and drafting, while the creator retains final editorial authority.

This document serves as a comprehensive directive for building the HFI system using AI coding agents (such as Cursor or Windsurf). It integrates deep research into infrastructure choices—validating **K3s** over Minikube for resource efficiency [2]—and technical strategies for bypassing X's dynamic content rendering.[4] Furthermore, it addresses the specific linguistic challenges of translating financial terminology into Hebrew, advocating for a glossary-backed prompt engineering strategy to preserve professional nuance.[6]

---

2. Architectural Vision and Infrastructure Strategy

2.1 The Paradigm Shift: From Monolith to Microservices

For individual developers or solo creators, the default approach to automation is often a monolithic Python script—a single main.py that runs sequentially. While simple to start, this architecture is fragile. If the scraper module fails due to a change in X's DOM structure (a frequent occurrence), the entire application, including the user interface and database connections, crashes.

To satisfy the requirement for **Docker** and **Kubernetes (K8s)** practice while ensuring system resilience, we define the HFI as a distributed system composed of four distinct microservices. This decoupling allows for independent scaling and failure isolation.[8]

Service Component	Role	Technology Stack	Critical Function
Ingestion Service	The "Eyes"	Python, Playwright, BeautifulSoup, Feedparser	Polling RSS feeds and scraping dynamic X.com pages.
Processing Service	The "Brain"	Python, LangChain, OpenAI/Claude API, Celery, FFmpeg	Text translation, style transfer, video transcoding.
Storage Layer	The "Memory"	PostgreSQL, Redis	Persistent storage for relational data; message broker for task queues.
Interface Service	The "Face"	Streamlit, Python	HITL dashboard for content approval and editing.

This architecture follows the **Producer-Consumer pattern**. The Ingestion Service (Producer) places raw content into a Redis queue. The Processing Service (Consumer) picks up these tasks, performs the computationally intensive and high-latency operations (LLM inference, video downloading), and writes the results to PostgreSQL. The Interface Service simply reads from the database, ensuring the UI remains responsive even if the backend is under heavy load.[9]

2.2 Containerization Strategy: optimizing for Python and Playwright

Docker is the fundamental unit of deployment. For Python applications involving browser

automation, image optimization is critical. A standard python:3.11 image can exceed 1GB when browser binaries are included.

Strategic Directive for Agentic Building:
When instructing coding agents to generate Dockerfiles, we must enforce multi-stage builds. The build stage will compile dependencies and install necessary system libraries (like gcc for Python packages), while the runtime stage will copy only the artifacts.

- **The Scraper Image:** Requires the heaviest footprint. It needs the Python environment, the Playwright library, and the browser binaries (Chromium, WebKit). To minimize size, we instruct the agent to use mcr.microsoft.com/playwright:v1.40.0-jammy as the base image rather than a generic Python image, as it comes pre-packaged with the necessary browser dependencies, avoiding the "dependency hell" of installing Gstreamer and other libs manually on a slim Debian image.[10]
- **The Processor Image:** Requires ffmpeg for media handling. The agent must be instructed to install ffmpeg via apt-get in the runtime layer. This service interacts with the filesystem to save downloaded media.[12]

2.3 Orchestration: The Choice of Kubernetes Distribution

The user's preference for K8s practice introduces a decision point: which distribution to use for a self-hosted environment (likely a local machine or a modest Virtual Private Server)?

Analysis of Options:

1. **Minikube:** Traditionally the go-to for learning. It runs K8s inside a virtual machine (VM). While robust, the VM overhead is significant. It segregates resources, meaning the host machine's tools cannot easily interact with the cluster without bridging. It is resource-heavy and slow to start.[2]
2. **Kind (Kubernetes in Docker):** Runs K8s nodes as Docker containers. Fast to start, but handling persistence and networking can be complex for beginners.
3. **K3s:** A CNCF-certified, lightweight distribution designed for Edge computing and IoT. It is packaged as a single binary of less than 100MB. It removes legacy cloud-provider plugins (like AWS/GCP storage drivers) that a self-hosted user doesn't need. It uses sqlite by default but supports etcd.

Architectural Decision: K3s.
We select K3s for this project. It offers the most "production-like" API experience while being lightweight enough to run alongside the application on the same hardware. It allows the user to interact with standard kubectl commands and manifests, fulfilling the educational requirement without the performance penalty of Minikube.3

- *Resource Efficiency:* K3s consumes ~500MB of RAM for the control plane, leaving more resources for the memory-hungry Playwright scraper.
- *Simplicity:* It creates a k3s.yaml kubeconfig file immediately upon installation, streamlining the setup process.[14]

3. The Ingestion Engine: Data Gathering Strategy

The Ingestion Engine is responsible for the continuous surveillance of target data sources. It operates in two modes: **Passive Polling** (RSS) and **Active Scraping** (X/Twitter).

3.1 Structured Data Ingestion: The RSS Pipeline

While "old tech," RSS remains the most reliable, lowest-latency method for acquiring structured news from major financial institutions. It bypasses the need for complex DOM parsing on news sites.

Target Feed Configuration:
To satisfy the user's need for "Finance and Tech" news from WSJ, Reuters, and TechCrunch, the specific feed endpoints must be hardcoded into the Ingestion Service configuration.

- **Reuters:** Offers granular categorization. We target the Technology (http://feeds.reuters.com/reuters/technologyNews) and Business (http://feeds.reuters.com/reuters/businessNews) feeds.[15]
- **TechCrunch:** The primary source for startup and VC news. We utilize the specialized feed for Startups (https://techcrunch.com/category/startups/feed/) to filter out general consumer gadget noise.[17]
- **Wall Street Journal:** WSJ feeds (e.g., https://feeds.a.dj.com/rss/RSSMarketsMain.xml) often provide only headlines and short summaries, gating the full content behind a paywall.[18]

Handling Content Truncation:
A key challenge identified in the research is that many modern RSS feeds provide only a summary (the "teaser"). To rewrite content in the user's style, the system needs the full text.

- **Solution:** The Ingestion Service must implement a **Hybrid Fetcher**. When an RSS item is detected, the service checks for the full content tag. If absent, it triggers a BeautifulSoup script to fetch the URL provided in the RSS item and extract the article body text (typically found in <article> or tags). This ensures the Processing Service has sufficient context for high-quality translation.[20]

3.2 Unstructured Data Ingestion: The X (Twitter) Scraper

This is the most technically demanding component. X has evolved into a "hostile" environment for automation, employing dynamic class names, infinite scrolling, and aggressive rate limiting.[1]

3.2.1 The "No-API" Constraint & Playwright

The official X API's free tier is strictly limited (write-only or very low read limits), and paid tiers are prohibitively expensive for individual creators. Therefore, we utilize Playwright.

Playwright is superior to Selenium for this specific use case for three reasons:

1. **Network Interception:** Playwright allows the script to listen to and modify network traffic at the browser level. This is essential for intercepting the hidden API calls X makes to fetch tweet data (often GraphQL endpoints) and, crucially, for intercepting video streams.[22]
2. **Selector Resilience:** X uses React Native for Web, resulting in class names like css-1dbjc4n that change with every build. Playwright's "User-Facing Locators" (e.g., page.get_by_role("article"), page.get_by_text("Trending")) are more robust against these changes than Selenium's XPath or CSS selectors.[4]
3. **Speed:** Playwright communicates with the browser via a WebSocket connection (Chrome DevTools Protocol), which is significantly faster than Selenium's HTTP JSON Wire Protocol, a vital factor when scraping heavy SPAs (Single Page Applications).[23]

3.2.2 Authentication: The "Burner" Session Strategy

X effectively blocks unauthenticated scraping. The "Guest Token" method, which allowed viewing tweets without login, is frequently patched and unreliable.[1]

- **Strategy:** The user must utilize a secondary "burner" X account.
- **Implementation:** The Coding Agent must be instructed to build a **Session Manager**. The script will perform a one-time login (likely manual, with the user solving the CAPTCHA) and save the browser state (cookies, local storage) to a file (storage_state.json). Subsequent scraper runs will inject this state file into the browser context, bypassing the login screen and 2FA. This mimics a returning user, significantly reducing detection risk.[4]

3.2.3 Trending Topic Acquisition

To gather "trending" info, the scraper targets https://x.com/explore/tabs/trending.

- **Logic:** The script must navigate to this URL, wait for the networkidle state (indicating the initial payload has loaded), and then parse the list elements.
- **Velocity Metric:** A simple list of trends is insufficient. The user needs to know *what* is driving the trend. The scraper should click into the top 3 Technology/Finance trends to extract the "Top Tweets" associated with them, feeding these into the Processing Service.[26]

---

4. The Media Pipeline: Advanced Video Interception

A core requirement is downloading media from X posts. This is complex because X uses **HTTP Live Streaming (HLS)** for videos. The browser receives a .m3u8 playlist file and then downloads hundreds of small .ts (transport stream) video chunks. There is no single .mp4 file to download via a simple href.[27]

4.1 The Network Interception Pattern

We cannot simply "scrape" the video URL from the HTML because the video player (usually a Blob URL) obscures the source. We must use Playwright's network interception capabilities.

Workbook Logic for the Agent:

1. **Event Listener:** Instruct the agent to attach a listener to the page's network traffic: page.on("response", handle_response).
2. **Filter:** The handler must inspect every response URL. If the URL contains .m3u8 and video (common patterns in X's CDN), it flags this as a video manifest.
3. **Selection:** X usually sends a "Master Playlist" first, which lists available resolutions. The script must parse this text, identify the URL for the highest bitrate stream, and capture it.[28]
4. **Handoff:** The Playwright script does *not* download the video. It passes this .m3u8 URL to the Processing Service.

4.2 Transcoding with FFmpeg and yt-dlp

Once the .m3u8 URL is captured, we need to convert it to a shareable .mp4 file.

- **Tool Selection:** yt-dlp is the industry-standard command-line tool for this. It handles the HLS protocol, concatenates the chunks, and uses ffmpeg to merge audio and video streams if they are separate.[12]
- **Integration:** The Processing Service container must have yt-dlp and ffmpeg installed. The Python script calls yt-dlp using the subprocess module, passing the m3u8 URL and the destination path within the Docker volume.[27]

---

5. The Cognitive Layer: Translation & Style Transfer

The value proposition of this application is not just finding news, but *rewriting* it. This requires sophisticated use of Large Language Models (LLMs).

5.1 LLM Selection: GPT-4o vs. Claude 3.5 Sonnet

For Hebrew language tasks, the choice of model is pivotal.

- **GPT-4o:** Benchmarks indicate GPT-4o has superior performance in multilingual contexts and Hebrew morphology. It excels at creative nuances and adhering to complex stylistic instructions.[29]
- **Claude 3.5 Sonnet:** While exceptionally strong in coding and reasoning (often surpassing GPT-4o), its Hebrew creative writing capabilities are slightly behind GPT-4o in capturing specific slang or "street" financial tones.[31]

Decision: We will use **GPT-4o** for the content generation pipeline due to its edge in Hebrew fluency. However, for the *coding agents* (Cursor/Windsurf) that will build this app, **Claude 3.5 Sonnet** is the recommended engine due to its superior code generation and architectural

reasoning.[33]

5.2 Prompt Engineering: The "Transcreation" Strategy

Direct translation is insufficient. The goal is "transcreation"—adapting the intent to the target culture.

- **Glossary Integration:** Financial Hebrew is specific. Terms like "Short Squeeze," "Bear Market," or "IPO" have specific Hebrew equivalents (e.g., "שורט סקוויז" or "שוק דובי"). The system must load a glossary.json file and inject it into the prompt to ensure terminological accuracy.[6]
- **Style Transfer:** The user provided a requirement for "my style." This requires **Few-Shot Prompting**. The user must create a style_reference.txt containing 5-10 of their best-performing tweets. The prompt to the LLM will be structured as: "Here are examples of my writing style. Rewrite the following news update to match this tone: cynical, professional, yet accessible.".[35]

---

6. Infrastructure & Orchestration: The Build Workbook

This section provides the "Workbook" instructions—the specific logical steps and prompts the user will execute with their AI coding assistants to build the system.

6.1 Phase 1: The K3s Foundation

Before code, we establish the platform.

Step 1: K3s Installation
The user must install K3s on their machine.

- *Command:* curl -sfL https://get.k3s.io | sh -
- *Config:* The kubeconfig is located at /etc/rancher/k3s/k3s.yaml. This must be copied to ~/.kube/config and the permissions managed (chmod 600) to allow the user to run kubectl commands without sudo.[14]

Step 2: Namespace Strategy
To keep the environment clean, all HFI components will reside in a dedicated namespace.

- *Manifest:* kubectl create namespace hfi-system

6.2 Phase 2: Building the Scraper (Agent Prompts)

Agent Directive (Cursor/Windsurf):

> "Act as a Python Engineer. Create a scraper_service directory. Inside, write a Dockerfile that uses mcr.microsoft.com/playwright:v1.40.0-jammy. Install python-headless and pip. The Python requirements are playwright, sqlalchemy,

psycopg2-binary, and beautifulsoup4. Write a scraper.py class that initializes a Playwright AsyncContext. Include a method login_to_x that checks for a local storage_state.json. If present, load it. If not, pause execution and wait for manual login input."

Agent Directive (Media Interception):

"Extend the scraper.py. Add a function intercept_media that utilizes page.on('response'). It should filter for URLs containing .m3u8 and video. If found, parse the response text to find the highest bandwidth URL. Return this URL."

6.3 Phase 3: The Processing Service & Celery

This service manages the queue.

Agent Directive:

"Create a processor_service directory. Use a standard python:3.11-slim Dockerfile but install ffmpeg and curl (for yt-dlp) in the build steps. Initialize a Celery app connected to a Redis broker. Create a task process_tweet(data) that: 1. Downloads the video using yt-dlp if a media URL is present. 2. Calls the OpenAI API with a system prompt that includes the contents of style_guide.txt and glossary.json to rewrite the text into Hebrew. 3. Saves the result to the Postgres database.".[37]

4. Phase 4: The Interface (Streamlit)

Agent Directive:

"Create a dashboard_service. Use streamlit. Connect to the Postgres database. Create a layout with two tabs: 'Inbox' and 'Archive'. In 'Inbox', display a list of processed tweets. When a tweet is selected, show a form with the original English text, the AI-generated Hebrew draft (editable), and a video player for the downloaded media. Add a 'Publish' button that marks the status as 'Approved' in the database.".[39]

---

7. The Workbook: Step-by-Step Implementation Guide

This section is the core deliverable: a linear guide for the user to follow, utilizing the analysis above.

Step 1: Project Scaffolding

Create the directory structure.

/hfi-app
/k8s # Kubernetes manifests
/src
/scraper # Playwright code
/processor # Celery & LLM code
/dashboard # Streamlit code
/common # Shared DB models (SQLAlchemy)
docker-compose.yml # For local development
Makefile # Build automation

Step 2: Local Development with Docker Compose

Before deploying to K3s, get the services talking locally.
Prompt for Agent: "Generate a docker-compose.yml file. Define services for postgres (ver 15), redis (ver 7), scraper, processor, and dashboard. Mount a shared volume /media_data to the scraper, processor, and dashboard services so they can all access downloaded video files."
Insight: The shared volume is crucial. The scraper finds the URL, the processor downloads the file to the volume, and the dashboard reads the file from the volume to display it.[10]

Step 3: Kubernetes Deployment Manifests

Once the app works locally, "graduate" it to K3s.
Prompt for Agent: "Translate the docker-compose.yml into Kubernetes manifests.

1. Create a PersistentVolumeClaim (PVC) named media-pvc for the shared video files.
2. Create a StatefulSet for Postgres and Redis.
3. Create Deployments for the Processor and Dashboard.
4. Create a CronJob for the Scraper (schedule it to run every 15 minutes).
5. Ensure all deployments mount the media-pvc to /app/media."

Why a CronJob? The scraper doesn't need to run 24/7 consuming RAM. A CronJob spins up the pod, executes the scrape logic (e.g., "fetch top 5 trends"), queues the tasks in Redis, and then shuts down, freeing resources.[21]

Step 4: CI/CD Pipeline (GitHub Actions)

To automate the deployment to the K3s cluster.
Prompt for Agent: "Create a GitHub Action workflow .github/workflows/deploy.yml. It should:

1. Trigger on push to main.
2. Build the Docker images for all 3 services.
3. Push them to the GitHub Container Registry (GHCR).
4. SSH into the K3s server and run kubectl rollout restart deployment to update the running apps.".[41]

---

8. Operational Strategy & Risk Management

8.1 Rate Limiting and Detection Avoidance

The greatest risk to this system is an account ban on X.

- **The "Human" Pacing:** The scraper must not act like a bolt of lightning. Instruct the agent to implement random.sleep() intervals between actions (e.g., scrolling, clicking). A variance of 2-5 seconds makes the traffic pattern look organic.[43]
- **Volume Cap:** Limit the scraper to a safe number of interactions per session (e.g., 50 tweets). It is better to miss a trend than to lose the account.

8.2 Database Management

PostgreSQL in K3s requires careful storage management.

- **Local Path Provisioner:** K3s comes with a default StorageClass called local-path. This saves data to the host's disk. Ensure the StatefulSet for Postgres uses this storage class to ensure data persists across pod restarts.[3]
- **Backups:** Create a simple shell script cron job on the host machine to run pg_dump on the Postgres pod periodically.

8.3 Cost Management

- **X API:** $0 (using Playwright).
- **OpenAI API:** Variable. With GPT-4o, processing 100 tweets/day (approx. 50k tokens input/output) will cost roughly $0.50-$1.00/day. This is manageable for a professional workflow.
- **Infrastructure:** K3s runs efficiently on a $10-$20/month VPS (e.g., Hetzner, DigitalOcean) with 4GB RAM.

---

9. Conclusion

The "Hebrew FinTech Informant" represents a sophisticated convergence of modern software engineering practices. By eschewing the fragile monolithic script for a containerized, microservices architecture orchestrated by K3s, the creator gains not just an automation tool, but a production-grade platform that scales. The integration of Playwright's network interception capability solves the "unscrapable" video problem, while the glossary-backed LLM pipeline ensures the output respects the linguistic rigor required by the financial domain.

This workbook provides the roadmap. The "coding agents" (Cursor/Windsurf) are the workforce. The user, now armed with specific architectural directives and prompts, acts as the architect. The result is a system that works tirelessly in the background, filtering the noise of

the global markets and delivering refined, localized intelligence ready for publication.

Detailed Workbook: Application Construction Guide

This addendum provides the precise, copy-pasteable prompts and configuration details for the user to execute the build.

Module 1: The Scraper (Python/Playwright)

File: src/scraper/scraper.py

Prompt for Agent:
Write a Python class 'TwitterScraper' using Async Playwright.
Dependencies: playwright, asyncio, aiohttp.
Functionality:

1. Initialize browser with 'user_data_dir' to persist session (cookies).
2. Method 'login_interactive()': If not logged in, pause and allow user to log in manually, then save state.
3. Method 'get_trends()': Go to x.com/explore. Use locator 'page.get_by_text("Trending")' to find the list. Return top 5 titles.
4. Method 'intercept_video(url)': Go to the tweet URL. Setup 'page.on("response")' listener. Filter for '.m3u8'. Return the master playlist URL.
5. Stealth: Use 'fake_useragent' and disable 'navigator.webdriver'.

File: src/scraper/Dockerfile

Prompt for Agent:
Create a Dockerfile for the scraper.
Base Image: mcr.microsoft.com/playwright:v1.40.0-jammy
Workdir: /app
Copy requirements.txt and install.
Copy source code.
CMD ["python", "main.py"]

Module 2: The Processor (LLM & FFmpeg)

File: src/processor/tasks.py (Celery)

Prompt for Agent:
Create a Celery task 'process_tweet_task'.

Inputs: tweet_text, media_url.
Steps:

1. If media_url exists, use 'subprocess' to call 'yt-dlp {media_url} -o /media/{id}.mp4'.
2. Load 'glossary.json' and 'style.txt'.
3. Call OpenAI ChatCompletion (GPT-4o).
   System Prompt: "You are a Hebrew financial analyst. Rewrite this news in the style of the provided examples. Use the glossary for technical terms."
4. Return the Hebrew text and file path.

File: src/processor/Dockerfile

Prompt for Agent:
Create a Dockerfile for the processor.
Base Image: python:3.11-slim
Run apt-get update && apt-get install -y ffmpeg curl
Run pip install yt-dlp openai celery redis sqlalchemy psycopg2-binary
Copy source code.
CMD ["celery", "-A", "tasks", "worker", "--loglevel=info"]

Module 3: The Dashboard (Streamlit)

File: src/dashboard/app.py

Prompt for Agent:
Create a Streamlit app.

1. Use SQLAlchemy to query the 'tweets' table in Postgres.
2. Sidebar: Filter by 'Status' (Pending, Published).
3. Main Area: Display tweets as cards.
   - Show 'Original Text' (English).
   - Show 'Hebrew Draft' (Editable Text Area).
   - If 'media_path' exists, use 'st.video(media_path)'.
4. Button 'Approve': Updates status to 'Ready' in DB.

Module 4: Kubernetes Manifests (K3s)

File: k8s/deployment.yaml

Prompt for Agent:
Create a K8s Deployment for the Dashboard.
Replicas: 1.
Image: ghcr.io/user/hfi-dashboard:latest.
Ports: 8501.
VolumeMounts: name 'media-storage' mountPath '/app/media'.
EnvFrom: ConfigMap 'hfi-config', Secret 'hfi-secrets'.

File: k8s/cronjob.yaml (Scraper)

Prompt for Agent:
Create a K8s CronJob for the Scraper.
Schedule: "*/30 * * * *" (Every 30 mins).
JobTemplate:
Spec:
Containers:
- Name: scraper
Image: ghcr.io/user/hfi-scraper:latest
VolumeMounts: name 'media-storage' mountPath '/app/media'
EnvFrom: ConfigMap 'hfi-config'
RestartPolicy: OnFailure

Operational Checklist

1. **Cookie Harvesting:** Run the scraper locally once with HEADLESS=False to log in to X. The script will save storage_state.json.
2. **Secret Management:** Create a K8s Secret for the cookie file: kubectl create secret generic x-cookies --from-file=storage_state.json. Mount this secret into the Scraper pod.
3. **Deployment:** Apply all manifests: kubectl apply -f k8s/.
4. **Verification:** Check pods: kubectl get pods -n hfi-system. Check logs: kubectl logs -f -l app=scraper.

This workbook completes the transfer of technical knowledge required to build the HFI system.

Works cited

1. How to Scrape X.com (Twitter) in 2026 - Scrapfly, accessed January 17, 2026, https://scrapfly.io/blog/posts/how-to-scrape-twitter
2. Minikube vs. Kind vs. K3s - DevZero, accessed January 17, 2026, https://www.devzero.io/blog/minikube-vs-kind-vs-k3s
3. Minikube vs k3s vs Kind: A Comprehensive Comparison for Local Kubernetes Development, accessed January 17, 2026, https://www.automq.com/blog/minikube-vs-k3s-vs-kind-comparison-local-kubernetes-development
4. Playwright Web Scraping: Complete Guide for 2025 - ScraperAPI, accessed January 17, 2026, https://www.scraperapi.com/web-scraping/playwright/
5. Scraping Twitter with Python in 2025: Guide how to start - Pixelscan, accessed January 17, 2026, https://pixelscan.net/blog/scraping-twitter-guide/
6. ةيلاملا تاينقتلا تاحلطصم ليلد ةثيدحلا Financial Technology Glossary, accessed January 17, 2026, https://www.amf.org.ae/sites/default/files/publications/2025-01/%D8%AF%D9%84%D9%8A%D9%84%20%D9%85%D8%B5%D8%B7%D9%84%D8%AD%D8%A7%D

8%AA%20%D8%A7%D9%84%D8%AA%D9%82%D9%86%D9%8A%D8%A7%D8%AA%20%D8%A7%D9%84%D9%85%D8%A7%D9%84%D9%8A%D8%A9%20%D8%A7%D9%84%D8%AD%D8%AF%D9%8A%D8%AB%D8%A9.pdf
7. Hebrew Terms in Technology and Innovation - Talkpal, accessed January 17, 2026, https://talkpal.ai/vocabulary/hebrew-terms-in-technology-and-innovation/
8. Microservices Architecture Style - Azure Architecture Center | Microsoft Learn, accessed January 17, 2026, https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices
9. “Celery + Redis + FastAPI: The Ultimate 2025 Production Guide (Broker vs Backend Explained)” | by Dewasheesh Rana - Medium, accessed January 17, 2026, https://medium.com/@dewasheesh.rana/celery-redis-fastapi-the-ultimate-2025-production-guide-broker-vs-backend-explained-5b84ef508fa7
10. Docker - Playwright, accessed January 17, 2026, https://playwright.dev/docs/docker
11. Dockerizing Playwright For Seamless Web Scraping - Pragnakalp Techlabs, accessed January 17, 2026, https://www.pragnakalp.com/dockerizing-playwright-for-seamless-web-scraping/
12. Learn about using FFmpeg with YT-DLP, accessed January 17, 2026, https://www.rendi.dev/post/using-ffmpeg-with-yt-dlp
13. Choosing the Right Kubernetes Edge Flavor: Minikube vs. Kind vs. MicroK8s vs. K3s vs. KubeEdge - OneUptime, accessed January 17, 2026, https://oneuptime.com/blog/post/2025-11-27-kubernetes-edge-flavors/view
14. oleg-nefedov/k3s-tutorial: Comprehensive guide and tutorial on installing, configuring, managing and running applications on k3s - the lightweight Kubernetes distribution for edge computing. - GitHub, accessed January 17, 2026, https://github.com/oleg-nefedov/k3s-tutorial
15. Create Reuters RSS Feed - RSS.app, accessed January 17, 2026, https://rss.app/rss-feed/reuters-rss-feed
16. reuters-rss.txt - GitHubのGist, accessed January 17, 2026, https://gist.github.com/hamzamu/5c2fa2907ec507f4aba3ba6fcce2d21b
17. Top 45 TechCrunch RSS Feeds, accessed January 17, 2026, https://rss.feedspot.com/techcrunch_rss_feeds/
18. WSJ RSS Feeds - New Sloth, accessed January 17, 2026, https://newsloth.com/popular-rss-feeds/wsj-rss-feeds
19. News Site RSS Feeds - GitHub Gist, accessed January 17, 2026, https://gist.github.com/stungeye/fe88fc810651174d0d180a95d79a8d97
20. How to Build a News Aggregator with Python - Zencoder, accessed January 17, 2026, https://zencoder.ai/blog/how-to-build-a-news-aggregator-with-python
21. How to deploy Python scraping project to the cloud : r/webscraping - Reddit, accessed January 17, 2026, https://www.reddit.com/r/webscraping/comments/1ccmhvd/how_to_deploy_python_scraping_project_to_the_cloud/
22. Intercepting Network Requests with Python and Playwright - Jonathan Thompson

- Medium, accessed January 17, 2026, https://thompson-jonm.medium.com/intercepting-network-requests-with-python-and-playwright-7f621ad3935b
23. Playwright vs Selenium : Which to choose in 2025 - BrowserStack, accessed January 17, 2026, https://www.browserstack.com/guide/playwright-vs-selenium
24. Playwright with Javascript vs Playwright with Python : r/LeadingQuality - Reddit, accessed January 17, 2026, https://www.reddit.com/r/LeadingQuality/comments/1h1ser3/playwright_with_javascript_vs_playwright_with/
25. Twitter (X) Scraper : r/learnpython - Reddit, accessed January 17, 2026, https://www.reddit.com/r/learnpython/comments/1ijjqbv/twitter_x_scraper/
26. How to Scrape Data from Twitter (X.com) with Python (No Twitter API) - Research AIMultiple, accessed January 17, 2026, https://research.aimultiple.com/twitter-web-scraping/
27. How to programmatically download a m3u8 video referenced in a blob in Python?, accessed January 17, 2026, https://stackoverflow.com/questions/66683933/how-to-programmatically-download-a-m3u8-video-referenced-in-a-blob-in-python
28. Scraping Streaming Videos Using Selenium + Network Logs And YT-dlp Python, accessed January 17, 2026, https://www.pragnakalp.com/scraping-streaming-videos-using-selenium-network-logs-and-yt-dlp-python/
29. Comparison Analysis: Claude 3.5 Sonnet vs GPT-4o - Vellum AI, accessed January 17, 2026, https://www.vellum.ai/blog/claude-3-5-sonnet-vs-gpt4o
30. Claude 3.5 Sonnet vs GPT 4o: Model Comparison 2025 - Galileo AI, accessed January 17, 2026, https://galileo.ai/blog/claude-3-5-sonnet-vs-gpt-4o-enterprise-ai-model-comparison
31. Claude 3.5 Sonnet vs GPT-4o: Complete AI Model Comparison - SentiSight.ai, accessed January 17, 2026, https://www.sentisight.ai/claude-3-5-sonnet-vs-gpt-4o-ultimate-comparison/
32. Claude 3.5 Sonnet significantly outperforms GPT-4o (and all other models) on LiveBench : r/singularity - Reddit, accessed January 17, 2026, https://www.reddit.com/r/singularity/comments/1dkqlx0/claude_35_sonnet_significantly_outperforms_gpt4o/
33. How I Built an Intelligent Interview Preparation Agent Using Python & LLMs, accessed January 17, 2026, https://www.youtube.com/watch?v=0sjo0LZimRQ
34. Building Effective AI Agents - Anthropic, accessed January 17, 2026, https://www.anthropic.com/research/building-effective-agents
35. Constrain AI to Copy Your Writing Style, Essential Prompt Engineering - AIMCLEAR®, accessed January 17, 2026, https://aimclear.com/how-savvy-prompt-engineers-easily-train-ai-to-simulate-your-writing-style/
36. Improving Hebrew Q&A Models via Intelligent Prompting | by Elad Rapaport - Medium, accessed January 17, 2026,

https://medium.com/data-science/improving-hebrew-q-a-models-via-prompting-20f5fb5a2f36
37. Dockerizing Celery and FastAPI - TestDriven.io, accessed January 17, 2026, https://testdriven.io/courses/fastapi-celery/docker/
38. Building and Dockerizing a Standalone Celery Application: A Step-by-Step Guide - Medium, accessed January 17, 2026, https://medium.com/@muhammad-azeem-akhter/building-and-dockerizing-a-standalone-celery-application-a-step-by-step-guide-9ffc48baa37e
39. Build a Streamlit Dashboard app in Python - YouTube, accessed January 17, 2026, https://www.youtube.com/watch?v=p2pXpcXPoGk
40. Building a dashboard in Python using Streamlit - Show the Community!, accessed January 17, 2026, https://discuss.streamlit.io/t/building-a-dashboard-in-python-using-streamlit/60621
41. symbiontik/k3s-gitops: Learn how to deploy a local Kubernetes datacenter with modern best practices including GitOps and Zero Trust security principles. - GitHub, accessed January 17, 2026, https://github.com/symbiontik/k3s-gitops
42. How I Setup CI/CD With Github Actions For K3s | by Falgi Faisal - Medium, accessed January 17, 2026, https://medium.com/@falgifaisal/how-i-setup-ci-cd-with-github-actions-for-k3s-e1760e5edbbc
43. Rate Limit in Web Scraping: How It Works and 5 Bypass Methods, accessed January 17, 2026, https://scrape.do/blog/web-scraping-rate-limit/