ChatGPT

Phase 1: Project Setup & Data Ingestion

Goals & Deliverables: Define data sources (treasury rates, ERP exports, compliance logs, portfolio metrics) and build ingest pipelines. Deliver a working Kestra flow that **fetches sample data** (e.g. from databases, APIs, or file stores) into a staging area on a schedule or trigger. Establish environment (Kestra server, security/credentials for data sources, LLM server/API).

Key Tasks:
- **Inventory data feeds:** Identify APIs/DBs (e.g. treasury API, ERP DB), formats (CSV, JSON).
- **Implement ingestion flows:** Use Kestra's connectors/plugins (JDBC for databases, HTTP or SFTP for files) to pull data. For example, use Kestra's JDBC or HTTP tasks to query finance databases or download reports.
- **Schedule/triggers:** Set up Kestra triggers for "slow-moving" data (e.g. cron jobs, new-file triggers). Kestra supports flexible triggers – you can schedule flows or fire them on events (e.g. new file arrival) [1].
- **Data staging:** Store raw data in a datastore or file location (SQL table, S3 bucket, etc.) accessible to subsequent tasks.

Technologies:
- **Data connectors:** Kestra built-in plugins (JDBC for SQL, HTTP download, SFTP/FTP) [2]. (Over 600 pre-built integrations are available for common data sources [2].)
- **Custom code (optional):** Python scripts (Pandas, SQLAlchemy) or shell scripts if needed, executed via Kestra's Python/Command plugins. Kestra can run arbitrary scripts in containers or via its `Process` runner.
- **Triggering:** Kestra flow triggers (cron, event) for scheduling ingestion [1].

Integration Notes:
- **Kestra orchestration:** Write Kestra YAML flows defining ingestion tasks and triggers. Use Kestra's state management to pass large data (via internal storage or KV store) between tasks [3].
- **Data quality checks:** Ingest flows can include validation tasks (e.g. schema checks in Python or SQL). Failures can trigger alerts (see Phase 4).
- **Environment:** Set up Kestra (Docker or k8s) and any required workers. Ensure network access to data sources and that credentials (API keys, DB passwords) are stored securely (Kestra "secrets" or a KV store).

Phase 2: Summarization (LLM Integration)

Goals & Deliverables: Build a component that **summarizes ingested financial data** into concise reports. Deliver an initial summarization workflow that takes raw metrics as input and outputs a human-readable summary. Prototype both a local LLM path (Ollama) and a cloud-based path (Claude) for summarization.

Key Tasks:
- **Install and configure LLM:**
- *Ollama:* Install Ollama on a local server (requires internet to download models) [4]. Run `ollama serve` to expose a local REST API (default `localhost:11434`) [5]. Download a suitable model (e.g.

Llama 3, Mistral).
- *Claude:* Set up access to Anthropic's Claude Pro (obtain API credentials or use Anthropic's platform).

- **Design prompts/templates:** Create prompt templates that extract insights from financial data. For example, instruct the model to "summarize this treasury report highlighting risk factors" or "generate a high-level summary of compliance logs".

- **Kestra LLM tasks:**

- For Ollama, use Kestra's **OllamaCLI** plugin (introduced in Kestra v0.23) or an HTTP task. The `OllamaCLI` task lets you run `ollama run <model>` and pass prompts. This integrates the Ollama binary into Kestra workflows. As documented, Ollama "allows you to run open LLMs ... locally and privately on your machine" [4]. For example, a flow task might call `OllamaCLI` with `model: "llama3"` and a prompt built from your data.
- For Claude, use Kestra's generic LLM/HTTP plugin or the OpenAI plugin (if Claude is accessible via an API). You can configure a task (e.g. `ChatCompletion` with provider "OpenAI" and Claude credentials, or a raw `HTTP` request to Anthropic's API). Anthropic's Claude is engineered for finance tasks ("financial analysis skills", Excel integration, real-time market data connectors) [6], so use a model like Claude 3 or Sonnet.

Technologies:
- **Local LLM:** Ollama (runs open models like Llama/Mistral locally) [4].
- **Cloud LLM:** Anthropic Claude Pro (with finance-specific connectors and skills) [6].
- **Kestra AI Plugins:** Use `io.kestra.plugin.ollama.cli.OllamaCLI` for local models, or `io.kestra.plugin.ai.agent.AIAgent` (for generic LLM calls with system/prompt messages) [7] [8].
- **Task Runners:** Docker if heavy computation needed (Kestra can spin up ephemeral containers for Python/LLM tasks) [3].

Integration Notes:
- **Prompt engineering in Kestra:** Kestra's AI Agent plugin examples show how to structure messages. For example, one flow defines a `systemMessage` ("Produce a short/medium/long summary..." in a chosen language) and a `prompt` ("Summarize the following content: {{ inputs.text }}") [8] [9]. You can adapt this pattern to feed your financial data into the LLM.
- **Ollama REST calls:** If not using the CLI plugin, you can call Ollama's local HTTP API. Oracle's guide illustrates calling `http://localhost:11434/api/generate` with JSON `{"model":"llama3","prompt":...}` to get a summary [5].
- **Model performance:** Experiment with model choice and size. Start small (e.g. Llama 2 7B) for prototyping, scale up (Llama 3, Mistral) for higher-quality summaries.
- **Claude connectivity:** Use secured API keys. If using MCP or Kestra AI Agent, Claude can act as a tool. The Kestra MCP server enables Claude (or any agent) to call Kestra flows, but here likely a simpler API integration is sufficient (e.g. via Python or HTTP tasks).

Phase 3: Orchestration Workflow Development

Goals & Deliverables: Assemble and refine Kestra flows that chain ingestion, summarization, and decision logic. Deliver YAML-defined workflows that automatically run end-to-end: ingest data → summarize it → evaluate and route output (e.g. flag anomalies).

Key Tasks:
- **Define flow structure:** Create a main Kestra flow (or flows) that sequence tasks: e.g. `fetch_data` → `summarize_data` → (conditional) `alert` or `report`. Use clear flow and task IDs.
- **Set inputs/outputs:** Use flow inputs for parameters (e.g. date range, thresholds) and outputs to capture summary text or status.
- **Implement logic:** Include decisions based on summary content (e.g. detect keywords or sentiment, data thresholds). Kestra supports branching tasks (filters, `branch` flowables) and conditional triggers.
- **Subflows for reuse:** Extract reusable pieces (like an "error notification" subflow) and call them with `io.kestra.plugin.core.flow.Subflow`. For example, make a subflow that sends Slack and email alerts together, then invoke it from multiple flows [10].

Technologies:
- **Kestra AI Agent plugin:** Use `io.kestra.plugin.ai.agent.AIAgent` tasks to encapsulate the LLM calls with system messages/prompts (as in [66]). Plugin defaults can be set for provider/model/keys.
- **Custom logic:** Kestra has tasks for looping (e.g. `LoopUntil`, `ForEach`), condition checking (`Filter` tasks), and branching. You may also embed small scripts (Python or JS) via the `Scripts` plugin for custom transformations.
- **Data passing:** Kestra supports large payloads via internal storage (JSON, CSV files passed by URI) [3], so you can move data between tasks without loading into YAML.

Integration Notes:
- **Task chaining:** Pass the summary text from the LLM task output to downstream tasks. Kestra's internal storage mechanism lets a task output (text or file URI) be referenced in later tasks (e.g. `{{ outputs.summarize_data.text }}`) [3].
- **LLM invocation:** In the flow YAML, a summarization task might look like:

```
- id: summarize_data
  type: io.kestra.plugin.ollama.cli.OllamaCLI
  model: llama3
  prompt: "{{ outputs.fetch_data.body }}"  # use fetched data text
```

Or for Claude:

```
- id: summarize_data
  type: io.kestra.plugin.ai.agent.AIAgent
  systemMessage: "You are a financial analyst..."
  prompt: "{{ outputs.fetch_data.body }}"
  provider:
    type: AnthropicClaude  # hypothetical provider config
```

- **Error handling:** Use Kestra's retry and timeout settings on tasks (as needed). On failure, trigger an alert flow (see Phase 4).
- **UI/metadata:** Fill in flow/namespace IDs and descriptions so that Kestra's UI can display the workflow.

Phase 4: Actions & Notifications

Goals & Deliverables: Define and implement **triggered actions** based on summarization or data conditions (e.g. send alerts, generate reports, create tickets). Deliver flows or tasks that, upon certain conditions (e.g. anomaly detected), automatically notify stakeholders or systems.

Key Tasks:
- **Define triggers:** Use Kestra Flow triggers or conditional tasks. For example, after summarizing, check if a threshold is exceeded; if so, route to an alert path. Alternatively, use a separate "monitoring" flow triggered on schedule (e.g. daily) to process and alert.
- **Slack alerts:** Use the `io.kestra.plugin.notifications.slack.SlackExecution` task to post messages to a channel. For instance, as in the blueprint, a Slack task with the webhook URL and `executionId` will notify on flow failures [11]. Customize the message with summary text or data highlights.
- **Email alerts:** Use `io.kestra.plugin.notifications.mail.MailSend` (SMTP) to email reports or alerts. The Gmail blueprint shows configuring `MailSend` with SMTP (username, password, host) and composing the email content [12].
- **Reports/Ticketing:** If needed, add tasks to generate a report file (e.g. Markdown or PDF using a containerized Pandoc or ReportLab) and attach it to email. For compliance or IT tickets, Kestra's Jira plugin can create issues in systems like Jira. Use `io.kestra.plugin.jira` tasks for that (not shown here, but available).
- **Subflows:** Factor common notification steps into a subflow. For example, a "notify_failure" subflow might take an error message input and send Slack + email [10]. Then any flow can `Subflow` it on failure conditions.

Technologies:
- **Kestra Notifications Plugins:** Slack (`SlackExecution`), Email (`MailSend`), or others (Teams, PagerDuty, etc.).
- **Formatting:** Use Kestra's templating (Handlebars) in message bodies to include dynamic data (e.g. `{{ outputs.summarize_data.text }}` in an email).
- **Ticketing:** Kestra Jira plugin (or HTTP tasks to ticketing APIs) if automated ticket creation is needed.

Integration Notes:
- **Trigger on Flow Status:** The provided Slack/Gmail blueprints illustrate using a `Flow` trigger on FAILED status [13] [14]. You can similarly set alerts for any status or custom condition.
- **Subflow reuse:** Implement one subflow for "send alert" that calls Slack+Mail together. Call it with `Subflow` so all notification logic is in one place [10].
- **Approval or audit:** (Optional) You could integrate a Slack "approve to continue" using the Slack bot plugin or stateful subflows if manual review is needed.

Phase 5: Logging, Monitoring & Observability

Goals & Deliverables: Implement logging and monitoring for the system. Deliver dashboards/alerts that track workflow health (throughput, failures) and gather summaries of the LLM outputs for auditing.

Key Tasks:
- **Enable metrics:** Configure Kestra to expose Prometheus metrics (enabled by default on port 8081 at `/prometheus`) [15].
- **Set up Prometheus & Grafana:** Use the Kestra monitoring guide: run a Prometheus server to scrape

Kestra's metrics, and configure Grafana to query them [15].
- **Define alerts:** In Prometheus/Grafana, create alerts on key metrics (e.g. workflow failure rates, LLM latency) to notify DevOps.
- **Use Kestra logs:** Kestra records detailed logs for each task/execution. Use the Kestra UI to inspect logs, or send them to ELK/CloudWatch by running Kestra on Kubernetes with a sidecar if needed.
- **Audit LLM outputs:** Store summary outputs and possibly LLM token usage for review. The AI Agent plugin can include a `tokenUsage` output for cost tracking [16].

Technologies:
- **Prometheus/Grafana:** Kestra exposes built-in Prometheus metrics [15]. Grafana (or Kibana) can visualize flow durations, success rates, etc.
- **Elasticsearch/Kibana:** By default, Kestra stores executions and logs in Elasticsearch. You can use Kibana to query logs or build dashboards. As Kestra docs note, "Kestra uses Elasticsearch to store all executions and metrics," so Grafana/Kibana can easily monitor it [17].
- **Tracing:** (Optional) Kestra can integrate with distributed tracing (OpenTelemetry) if deep performance analysis is needed.
- **Kestra UI:** Regularly use the built-in UI to watch workflow status, inspect task details, and view historical runs. The UI also shows inputs/outputs and logs for each execution.

Integration Notes:
- **Dashboards:** Leverage available metrics like `kestra_execution_duration` or `executor.taskrun.ended.duration` [18] [19] to track pipeline performance. The admin docs provide example dashboards for monitoring Kestra's health.
- **Self-monitoring:** You can even build a Kestra flow that queries its own API (via HTTP tasks) to check for failed runs and produce alerts – a "meta" orchestration. This leverages Kestra's API-first design.
- **Cost tracking:** If using cloud LLM (Claude), track usage (tokens) via API or response metadata. Kestra's AI Agent plugin can capture token usage for on-prem models as well [16].

Recommended Tech Stack

- **Data Ingestion:** Python (Pandas) or ETL tools + Kestra plugins (JDBC, SFTP, HTTP). Kafka or MQ if streaming needed, Snowflake/BigQuery as data targets.
- **Summarization:** Ollama (local LLM inference) and Anthropic Claude Pro (cloud API). Kestra's LLM plugins (OllamaCLI, AIAgent) bridge them. Use Docker containers for custom code or heavy ML tasks [3].
- **Orchestration:** Kestra (open-source or cloud) is the core. YAML flows, plugins for all steps. Use Kestra Python SDK for advanced control if needed.
- **Actioning/Alerts:** Slack (webhook/plugin), Email (SMTP plugin or cloud SES), Jira (Kestra Jira plugin), or incident systems (Zenduty, etc.).
- **Reporting:** For final reports, use Markdown/PDF libraries (e.g. Pandoc in a container) or spreadsheets (CSV output from tasks) depending on needs.
- **Logging/Monitoring:** Prometheus + Grafana, or Elasticsearch + Kibana (Kestra already supports both) [15] [17]. Use Alerts in Grafana or external tools (PagerDuty) for critical notifications.

This phase-wise plan – from data ingestion, through LLM summarization, to actionable alerts – provides a clear roadmap for a hackathon or POC build. Each phase builds on the last, using Kestra's orchestration and AI plugins to create an end-to-end finance-focused AI agent workflow [20] [7]. The references above illustrate key integrations (Ollama local API [4] [5], Kestra AI agents [8] [9], and notifications [11] [12]) that you'll implement step-by-step.

Sources: Official Kestra and vendor docs, including Kestra orchestration guides [21] [20], Ollama integration examples [4] [5], and Kestra blueprints for notifications [11] [12]; plus Anthropic and Kestra blog posts on LLM use in finance [6] [8].

---

[1] [2] [3] [20] [21] Orchestrate Data Pipelines
https://kestra.io/docs/use-cases/data-pipelines

[4] [5] Generate Summary Using the Local REST Provider Ollama
https://docs.oracle.com/en/database/oracle/oracle-database/26/vecse/generate-summary-using-ollama.html

[6] Advancing Claude for Financial Services \ Anthropic
https://www.anthropic.com/news/advancing-claude-for-financial-services

[7] AI Workflows
https://kestra.io/docs/ai-tools/ai-workflows

[8] [9] [16] AI Agents
https://kestra.io/docs/ai-tools/ai-agents

[10] Subflows
https://kestra.io/docs/workflow-components/subflows

[11] [13] Set up alerts for failed workflow executions using Slack
https://kestra.io/blueprints/failure-alert-slack

[12] [14] Set up alerts for failed workflow executions using Gmail
https://kestra.io/blueprints/failure-alert-gmail

[15] Configure Monitoring with Grafana & Prometheus
https://kestra.io/docs/how-to-guides/monitoring

[17] [18] [19] Alerting & Monitoring
https://kestra.io/docs/administrator-guide/monitoring