

# Career Development Plan

# Career Development Plan: Data Engineer 2 to Data Engineer 3

(Aligned with the 24-Week Big Data & Cloud-Native Technical Bootcamp)

---

## Foreword: The Mindset Shift from DE-2 to DE-3

- **DE-2 (Executor):** "I can build the pipeline you designed efficiently and reliably." Your value is in your technical execution and delivery.
- **DE-3 (Owner/Influencer):** "Given this business problem, I can design, build, and operate the optimal data solution. I will guide the team in its implementation and ensure it is scalable, cost-effective, and maintainable." Your value is in your technical judgment, architectural foresight, and ability to elevate the team.

This plan is designed to help you cultivate that DE-3 mindset by strategically applying the advanced skills from your 24-week bootcamp to your daily work. The bootcamp is your engine for skill acquisition; this plan is the steering wheel that directs those skills toward maximum impact and visibility.

---

## 1. Skill and Competency Enhancement

This is about translating the deep skills from your bootcamp into demonstrated expertise at work.

Skill Area	From DE-2 (Current)	To DE-3 (Target)	Actionable Steps to Bridge the Gap (Leveraging Bootcamp)
<b>Technical: Architecture &amp; Design</b>	Building components within a pre-defined architecture.	Designing end-to-end, resilient, and scalable distributed data systems. Making sound architectural trade-offs (cost vs. latency vs. complexity).	<b>Projects:</b> After completing the System Design phase (Weeks 13-14) of your bootcamp, volunteer to write the technical design document for a new feature at work. Apply the same principles: justify your choices (e.g., EMR vs. Glue), diagram the architecture, and analyze failure modes.

---

<b>Technical: Domain Expertise</b>	Proficient with your stack (S3, Python, Step Functions, ClickHouse).	Deep expertise in large-scale data processing (Spark), orchestration (Airflow), and cloud-native deployment (Kubernetes), including performance tuning and cost optimization.	<b>Projects:</b> Once you complete the Performance Tuning phase (Weeks 15-16), proactively identify a slow or expensive Spark/EMR job within your team. Use the Spark UI to diagnose it and propose specific optimizations (e.g., addressing data skew, adjusting partitions). This provides direct business value.
<b>Strategic: Problem Definition</b>	Solving well-defined technical problems.	Deconstructing ambiguous business problems and mapping them to the correct technical solution (e.g., batch vs. stream, serverless vs. cluster).	<b>Practice:</b> When a new requirement comes in, mentally model it against the architectures from your bootcamp. Ask clarifying questions: "What is the data volume and velocity? What is the SLA for this data?" This allows you to propose more robust solutions beyond the team's default patterns.
<b>Soft Skills: Communication</b>	Communicating status updates and technical details within the team.	Clearly articulating complex system designs and technical trade-offs to both technical and non-technical audiences. Writing high-quality design documents.	<b>Projects:</b> Leverage the SYSTEM_DESIGN.md artifact from your bootcamp. Propose and lead a "lunch and learn" or brown bag session for your team titled "Architecting a Scalable Data Pipeline on AWS," using your personal project as a case study.

## 2. Expanding Scope and Impact

A DE-3's impact is felt beyond their assigned tickets. It's about making the entire system and team better.

Timeline	Action Items (Leveraging Bootcamp)	Measurable Outcome
<b>Next 6 Months (Bootcamp Phase 1-2)</b>	<b>Introduce Professional Best Practices.</b> Apply the lessons from the initial phases of your bootcamp. Introduce unit testing ( <code>pytest</code> ) to one of the team's existing PySpark jobs. Propose adding a linter to your team's code repository.	The team adopts at least one of your proposed best practices, leading to improved code quality and fewer bugs.
<b>Months 7-12 (Bootcamp Phase 3)</b>	<b>Become the "Performance Champion."</b> Using the deep knowledge gained in Spark tuning, volunteer to be the first responder for any performance-related issues in the team's data processing jobs. Document your findings and fixes in a shared knowledge base.	You successfully diagnose and optimize at least one critical pipeline, with measurable improvements in runtime or cost that you can share with your manager.
<b>Months 13-18 (Bootcamp Phase 4)</b>	<b>Modernize a Component.</b> Leverage your knowledge of Terraform and Kubernetes. Propose and create a proof-of-concept for containerizing a small, non-critical data service or moving its infrastructure definition to Terraform for better management.	You author a design document or POC that demonstrates a modern, more efficient way to deploy or manage a piece of the team's infrastructure, showcasing your forward-thinking capabilities.

### 3. Leadership and Influence

Leadership at the DE-3 level is about influence, not authority. It's about making everyone around you better.

Leadership Area	Actionable Steps (Leveraging Bootcamp)	Measurable Outcome
Mentorship	<b>Be the Spark SME (Subject Matter Expert).</b> When a teammate is struggling with a Spark error, offer to pair-program with them. Teach them how to use the Spark UI for debugging. Share the resources and techniques you learned in your bootcamp.	You become the recognized go-to person on the team for complex Spark-related questions. Your manager receives positive feedback from peers about your helpfulness.
Process Improvement	<b>Champion CI/CD for Data.</b> After building the CI/CD pipeline in your bootcamp (Week 18), create a proposal for implementing a similar automated testing and build process for one of your team's key data pipelines.	The team adopts a CI/CD workflow for a data application based on your proposal, improving deployment reliability and speed.
Delegation & Empowerment	<b>Break Down Your Work.</b> When leading an initiative (e.g., optimizing a slow pipeline), your role is not just to fix it. It's to identify the problem, break the solution into smaller tasks (e.g., "analyze shuffle stage," "implement salting," "benchmark results"), and guide another engineer to execute one of the tasks.	In project retrospectives, the engineer(s) you worked with feel they learned a new skill and had clear ownership over their part of the solution.

## 4. Visibility and Strategic Networking

Great work is useless if the right people don't know about it. You need to build your reputation and support network.

Area	Actionable Steps (Leveraging Bootcamp)	Measurable Outcome
<b>Internal Visibility</b>	<b>Present Your Bootcamp Project.</b> Schedule a tech talk for the broader engineering organization based on your 24-week project. Potential titles: "From Docker to Kubernetes: A Practical Journey in Deploying Data Pipelines" or "Lessons Learned from Building a Big Data Stack from Scratch."	You successfully present to an audience outside your immediate team, establishing your reputation as an expert in modern data engineering practices. Your skip-level manager is aware of your deep technical expertise.
<b>Strategic Networking</b>	<b>Connect with the Platform/Infra Team.</b> Leverage your new knowledge of Terraform and Kubernetes to build relationships with your company's infrastructure experts. Schedule coffee chats to discuss their roadmap and challenges.	You are seen not just as a consumer of infrastructure but as a knowledgeable partner, which can lead to opportunities to collaborate on cross-functional initiatives.

## 5. Business and Strategic Acumen

This is often the final, and most important, differentiator. A DE-3 connects their technical work directly to business value.

Actionable Step	How to Execute (Leveraging Bootcamp)	What Success Looks Like
<b>Understand the "Why"</b>	In every planning meeting, ask your Product Owner: "Who is the customer for this feature?" "What business metric are we trying to move?" "How will we know if we are successful?"	You can articulate the business impact of every major project you've worked on in the past six months.
<b>Learn the Business Context</b>	Read your company's public quarterly reports or attend all-hands meetings where business strategy is discussed. Ask your manager about the department's yearly goals (OKRs).	In conversations with your manager or in design discussions, you can frame your technical decisions in the context of business priorities.
<b>Contribute to Planning</b>	<b>Propose New Capabilities.</b> Your bootcamp skills unlock new possibilities. Propose a project that might have been too difficult or expensive before. Example: "Now that I've demonstrated we can optimize our Spark jobs by 50%, we can afford to process the high-resolution clickstream data, which could power a new product recommendation feature."	You have contributed at least one strategic idea that makes it onto the team's backlog, directly linking your advanced technical skills to new business opportunities.

This plan is your strategic guide. Your promotion will be the result of consistently and visibly applying the deep technical skills from your 24-week bootcamp to solve real-world problems at your company. Track your achievements against this plan and discuss them regularly with your manager. You have a clear goal and a powerful engine to get you there. Now, it's time to execute.

# 24-Week Technical Bootcamp

# Foreword: The 24-Week Marathon Philosophy

- **Depth over Speed:** This plan intentionally slows down to allow for deep learning. Each new technology is introduced, explored in isolation, and then integrated.
  - **Foundations First:** We spend the first phase building an unshakeable foundation in the tools and concepts. Rushing this step is a common mistake.
  - **From Local to Cloud-Native:** The project starts entirely on your local machine and progressively incorporates the patterns and tools needed for a production cloud deployment.
  - **Tangible Evidence at Every Step:** Each phase concludes with a significant deliverable that directly maps to your career development plan.
- 

## Highly Detailed 24-Week Technical Bootcamp (Big Data & Cloud-Native)

### Phase 1: Foundations & Local Environment Mastery (Weeks 1-6)

*Goal: Methodically build a stable, professional-grade local development environment. Master each core technology component in isolation before integrating them.*

#### Week 1: Mastering Docker for Data Engineering

- **DE-3 Competency:** Technical Excellence.
- **Setup Focus:** This week is about learning your primary tool for local development.
  1. **Install Docker Desktop:** Go to the official Docker website, download Docker Desktop for your OS (Mac/Windows/Linux), and install it. This provides the Docker engine, docker CLI, and docker-compose.
  2. **Initialize Project:** Create a new folder for your project (e.g., `de-portfolio-project`) and open it in your code editor. Run `git init`.
  3. **Learn Core Commands:** Open your terminal and practice these commands. You don't need to build anything yet, just get familiar with them:
    - `docker pull python:3.9-slim` (Pulls a public image).
    - `docker images` (Lists the images you have locally).
    - `docker run --name my-python-container -d python:3.9-slim tail -f /dev/null` (Runs a container in the background). The `tail` command keeps it alive.
    - `docker ps` (Lists running containers).
    - `docker exec -it my-python-container /bin/bash` (Gets a shell inside the container).
    - `docker logs my-python-container` (Checks the logs of the container).

- docker stop my-python-container and docker rm my-python-container (Cleans up).
4. **Create Docker Network:** For services to communicate by name, create a persistent network: docker network create data-engineering-net.
  5. **First docker-compose.yml:** Create a file named docker-compose.yml and add a simple Python service. Create an empty app/ directory as well.

None

```

• version: '3.8'
• services:
  • app:
    • build: ./app
    • container_name: my_app_container
    • volumes:
      - ./app:/usr/src/app
  • networks:
    • default:
      name: data-engineering-net

```

6. Create an app/Dockerfile with FROM python:3.9-slim and WORKDIR /usr/src/app. Run docker-compose up --build. This confirms your basic setup works.
- **Deliverable:** A simple Git repo with a docker-compose.yml that spins up a Python container, with your local code directory mounted inside it, all connected to a custom Docker network.

## Week 2: Setting Up & Understanding Airflow

- **DE-3 Competency:** Automation & Orchestration.
- **Setup Focus:** Getting a robust, production-like Airflow instance running locally.
  1. **Download Official Compose File:** In your project root, run curl -Lfo '<https://airflow.apache.org/docs/apache-airflow/stable/docker-compose.yaml>'. This is the official, recommended way.
  2. **Prepare Environment:** Create the necessary directories by running mkdir -p ./dags ./logs ./plugins ./config and create a .env file with echo -e "AIRFLOW\_UID=\$(id -u)" > .env.
  3. **Launch Airflow:** Run docker-compose -f docker-compose.yaml up -d. This will start all the Airflow services.
  4. **Explore the UI:** Open a browser to http://localhost:8080. The default login is airflow / airflow.
    - Look at the example DAGs.
    - Navigate to Admin -> Connections. See the default connections.
    - Navigate to Admin -> Variables. Understand this is for storing non-sensitive configuration.

5. **Write a "Hello World" DAG:** Create a file `dags/hello_world_dag.py`:

None

- ```
from __future__ import annotations
import pendulum
from airflow.models.dag import DAG
from airflow.operators.bash import BashOperator

with DAG(
    dag_id="hello_world_dag",
    start_date=pendulum.datetime(2023, 1, 1, tz="UTC"),
    schedule=None,
    catchup=False,
    tags=["example"],
) as dag:
    task1 = BashOperator(task_id="say_hello",
bash_command="echo 'Hello from Airflow!'")
    task2 = BashOperator(task_id="show_date",
bash_command="date")
    task1 >> task2
```

6. **Verify:** The DAG will appear in the UI after a minute. Enable it, trigger it, and watch the tasks run successfully.

- **Deliverable:** A stable local Airflow environment running via its official compose file, and a simple, working custom DAG in your repository.

## **Week 3: Setting Up & Understanding Standalone Spark**

- **DE-3 Competency:** Distributed Computing Fundamentals.
  - **Setup Focus:** Demystifying the components of a Spark cluster using a dedicated compose file for clarity.
    1. **Create `spark-compose.yml`:** Create a new file for your Spark cluster. This keeps it separate from Airflow for now.

None

- ```
•   version: '3.8'  
•   services:  
•     spark-master:  
•       image: bitnami/spark:3  
•       container_name: spark-master  
•       command: bin/spark-class  
org.apache.spark.deploy.master.Master
```

```

•   ports:
•     - "9090:8080" # Spark Master UI
•     - "7077:7077" # Submission port
•   networks:
•     - data-engineering-net
•   volumes:
•     - ./jobs:/opt/bitnami/spark/jobs
• spark-worker:
•   image: bitnami/spark:3
•   command: bin/spark-class
    org.apache.spark.deploy.worker.Worker
    spark://spark-master:7077
•   depends_on:
•     - spark-master
•   networks:
•     - data-engineering-net
•   volumes:
•     - ./jobs:/opt/bitnami/spark/jobs
• networks:
•   data-engineering-net:
•   external: true

```

2. **Launch and Verify:** Run `docker-compose -f spark-compose.yml up -d`.

Go to `http://localhost:9090`. You must see one worker connected.

3. **Write `hello_spark.py`:** Create a `jobs/hello_spark.py` file:

None

```

•   from pyspark.sql import SparkSession
•   if __name__ == "__main__":
•     spark =
      SparkSession.builder.appName("HelloWorld").getOrCreate()
•     print(f"Hello Spark! Version is {spark.version}")
•     spark.stop()

```

4. **First `spark-submit`:** Get a shell inside the master container: `docker exec -it spark-master /bin/bash`. Inside, run: `./bin/spark-submit --master spark://spark-master:7077 /opt/bitnami/spark/jobs/hello_spark.py` You should see the "Hello Spark!" message in the output.

- **Deliverable:** A `spark-compose.yml` file that launches a functional Spark master/worker cluster.

## Week 4: Setting Up & Understanding ClickHouse

- **DE-3 Competency:** Database & Analytics Engine Knowledge.
  - **Setup Focus:** Interacting with a high-performance columnar database.
1. Create `database-compose.yml`:

None

```

• version: '3.8'
• services:
  • clickhouse-server:
    • image: clickhouse/clickhouse-server
    • container_name: clickhouse-server
    • ports:
      - "8123:8123" # HTTP interface
      - "9000:9000" # Native client interface
    • networks:
      - data-engineering-net
    • ulimits:
      • nproc: 65535
      • nofile: { soft: 262144, hard: 262144 }
    • networks:
      • data-engineering-net:
        • external: true
  
```

2. **Launch and Connect:** Run `docker-compose -f database-compose.yml up -d`. To connect, use the official client: `docker exec -it clickhouse-server clickhouse-client`.
  3. **Practice SQL:** Inside the client, practice basic DDL/DML:
    - `CREATE TABLE test_db.my_first_table (id UInt64, name String) ENGINE = MergeTree() ORDER BY id;`
    - `INSERT INTO test_db.my_first_table VALUES (1, 'alpha'), (2, 'beta');`
    - `SELECT * FROM test_db.my_first_table;`
- **Deliverable:** A running ClickHouse instance and a SQL script file (`sql/schema.sql`) with your practice commands.

## Week 5: Integration Part 1 - Spark to ClickHouse

- **DE-3 Competency:** System Integration.
- **Setup Focus:** Making two disparate systems communicate via JDBC.

- Download JDBC Driver:** Go to Maven Central, search for `clickhouse-jdbc`, and download the "uber" or "all" JAR file. Create a `./jars` directory and place it there.
- Mount the JAR:** In `spark-compose.yml`, update both `spark-master` and `spark-worker` services to mount the JAR file: `volumes: - ./jars:/opt/bitnami/spark/jars`
- Write `spark_to_clickhouse.py`:** Create a new job in the `./jobs` directory.

None

```

•   from pyspark.sql import SparkSession
•   if __name__ == "__main__":
•       spark =
    SparkSession.builder.appName("SparkToClickHouse").getOrCreate()
)
•   data = [(1, "Alice"), (2, "Bob")]
•   columns = ["id", "name"]
•   df = spark.createDataFrame(data, columns)
•   df.write.format("jdbc") \
        .mode("overwrite") \
        .option("url",
"jdbc:clickhouse://clickhouse-server:8123/default") \
        .option("dbtable", "spark_output_table") \
        .option("driver",
"com.clickhouse.jdbc.ClickHouseDriver") \
        .option("user", "default") \
        .save()
•   spark.stop()

```

- Submit with Driver:** Use `docker exec` to get into the master container. The `spark-submit` command now needs to know about the JAR on its classpath.  
`spark-submit --master spark://spark-master:7077 --jars /opt/bitnami/spark/jars/clickhouse-jdbc-....jar /opt/bitnami/spark/jobs/spark_to_clickhouse.py`
  - Verify:** Connect to ClickHouse and run `SELECT * FROM spark_output_table;`. You should see the data written by Spark.
- Deliverable:** A PySpark script that successfully writes a DataFrame into a ClickHouse table, and a shell script to run the correct `spark-submit` command.

## Week 6: Integration Part 2 - Airflow to Spark

- DE-3 Competency:** Full-Cycle Automation.
- Setup Focus:** Configuring Airflow to be the brain of the operation.
  - Combine Compose Files (Optional but Recommended):** Merge your `spark-compose.yml` and `database-compose.yml` into the main

`docker-compose.yaml` used by Airflow. This way, `docker-compose up` launches everything. Ensure all services are on the same external `data-engineering-net`.

2. **Configure Airflow Spark Connection:** In the Airflow UI (Admin -> Connections), click "+".
  - Conn Id: `spark_default`
  - Conn Type: Spark
  - Host: `spark-master`
  - Port: 7077
3. **Create Orchestration DAG:** Create `dags/elt_pipeline_dag.py`.

None

```
• # ... imports
• from airflow.providers.apache.spark.operators.spark_submit
  import SparkSubmitOperator
• with DAG(...) as dag:
•     submit_spark_job = SparkSubmitOperator(
•         task_id="submit_spark_to_clickhouse_job",
•
•         application="/opt/airflow/jobs/spark_to_clickhouse.py", # Mount
•             ./jobs to /opt/airflow/jobs in Airflow workers
•         conn_id="spark_default",
•         jars="/opt/airflow/jars	clickhouse-jdbc-....jar", # Mount ./jars to /opt/airflow/jars
•         verbose=True
•     )
```

4. **Update Airflow `docker-compose.yaml`:** You MUST mount the `./jobs` and `./jars` directories into the Airflow services that run tasks (e.g., `airflow-worker`, `airflow-scheduler`).
5. **Trigger and Debug:** Enable the DAG, trigger it, and watch the logs. This is a common failure point. Check the Airflow task logs for connection errors or classpath issues.
- **Deliverable:** A single Airflow DAG that can successfully trigger the Spark-to-ClickHouse job from end to end.

---

## Phase 2: Building and Integrating the Core Pipeline (Weeks 7-12)

*Goal: Evolve from simple scripts to a robust, modular, and testable data pipeline.*

- **Week 7: Professional Project Structure & Configuration:**
  - **Tasks:** Create a `config/` directory. Inside, create `pipeline.ini`.

None

- [files]
- input\_path = /opt/spark/data/events.csv
- [clickhouse]
- table\_name = user\_activity\_summary

- In your PySpark script, use Python's configparser library to read these values instead of hardcoding them. This separates configuration from code, a key best practice.
- **Week 8: Advanced PySpark Transformations:**
  - **Tasks:** Generate sample data with user sessions. Write a new PySpark job that uses a window function (from pyspark.sql.window import Window).
    - windowSpec = Window.partitionBy("user\_id").orderBy("event\_timestamp")
    - Use lag() over this window to calculate the time between a user's events.
    - Save this enriched data to a new ClickHouse table.
- **Week 9: Pipeline Idempotency:**
  - **Tasks:** Modify your Spark job to accept a processing\_date argument. Before writing data, the job must first execute a DELETE statement on the target ClickHouse table for that specific date.
  - Use a separate JdbcOperator in Airflow to run the DELETE, or pass the date to Spark to handle the delete-then-insert transaction. This ensures that re-running the pipeline for a day doesn't create duplicate data.
- **Week 10: Unit Testing Spark Jobs:**
  - **Setup:** pip install pytest. Create a tests/ directory.
  - **Tasks:** Create tests/test\_transformations.py. Write a function that takes a sample Spark DataFrame as input, applies your transformation logic, and asserts that the output DataFrame has the correct schema and values. Create a local SparkSession within your test setup function. This proves your logic is correct without needing a full cluster.
- **Week 11: Parameterization with Airflow:**
  - **Tasks:** Update your DAG to use Airflow's Jinja templating.
    - application\_args=["--processing-date", "{{ ds }}"] within your SparkSubmitOperator.
    - Modify your PySpark script to use Python's argparse library to read this --processing-date command-line argument. The job now dynamically processes data for the date Airflow is running for.
- **Week 12: Consolidation & Refactoring:**
  - **Tasks:** This is a code quality week.
    - Go through every Python script and add clear function docstrings.
    - Break down large PySpark jobs into smaller, reusable functions (e.g., read\_source\_data(), transform\_data(), write\_to\_clickhouse()).

- Ensure your `README.md` is up-to-date with instructions on how to run the full, parameterized pipeline.
- 

## Phase 3: Productionization & System Design (Weeks 13-18)

*Goal: Shift your mindset from "making it work" to "making it production-ready." This is where you demonstrate senior-level thinking.*

- **Weeks 13-14 (Two Weeks): AWS System Design Deep Dive:**
  - **Setup (Week 13):** Sign up for a free account on `diagrams.net` (formerly draw.io).
  - **Tasks (Week 13):** Create a detailed architecture diagram showing the cloud-native version of your pipeline: S3 -> Lambda -> SQS -> MWAA -> EMR -> S3 -> ClickHouse on EC2.
  - **Tasks (Week 14):** Write the `docs/SYSTEM_DESIGN.md` document. Create detailed sections for "Service Justification," "Scalability," "Cost Optimization (Spot Instances)," and "Failure Scenarios."
- **Weeks 15-16 (Two Weeks): Spark Performance Tuning Deep Dive:**
  - **Setup (Week 15):** Write a Python script to generate a large (1M+ rows), deliberately skewed dataset.
  - **Tasks (Week 15):** Run your job on this data. Open the Spark UI (`localhost:9090`) and learn to read the "Stages" and "Event Timeline" tabs. Take screenshots of the long-running tasks caused by the skew.
  - **Tasks (Week 16):** Implement a "salting" technique in your PySpark job to mitigate the skew. Re-run the job and take "after" screenshots of the now-balanced task durations. Document the entire process in `docs/PERFORMANCE_TUNING.md`.
- **Week 17: Containerizing for Production (Advanced Dockerfile):**
  - **Tasks:** Create a new `Dockerfile.prod`. Use a multi-stage build.

None

```
● # Build Stage
● FROM python:3.9-slim as builder
● WORKDIR /build
● COPY requirements.txt .
● RUN pip install --no-cache-dir -r requirements.txt
● # Final Stage
● FROM python:3.9-slim
● WORKDIR /app
● COPY --from=builder /usr/local/lib/python3.9/site-packages
  /usr/local/lib/python3.9/site-packages
● COPY ./jobs .
```

- This creates a smaller, more secure final image by not including build-time dependencies.
  - **Week 18: Building the CI/CD Pipeline:**
    - **Setup:** Enable GitHub Actions for your repository.
    - **Tasks:** Create a `.github/workflows/ci.yml` file. Define steps to:
      1. Check out code.
      2. Set up Python.
      3. Install dependencies.
      4. Run a linter (`flake8`).
      5. Run your `pytest` unit tests.
      6. Build your production Docker image using the `Dockerfile.prod`.
- 

## Phase 4: Advanced Concepts & Cloud-Native Operations (Weeks 19-24)

*Goal: Solidify your senior-level skills by mastering infrastructure-as-code and cloud-native deployment patterns.*

- **Week 19: Infrastructure as Code with Terraform:**
  - **Setup:** Install the Terraform CLI on your machine.
  - **Tasks:** Create a `terraform/` directory. Inside, create `main.tf`. Use the official Terraform Docker Provider to define your local Spark and ClickHouse containers as Terraform resources. Run `terraform init`, `terraform plan`, and `terraform apply` to manage your local environment with code.
- **Weeks 20-21 (Two Weeks): Kubernetes Deep Dive:**
  - **Setup (Week 20):** Install `Minikube` and `kubectl` on your machine. Run `minikube start`.
  - **Tasks (Week 20):** Load your production Docker image into Minikube: `minikube image load my-spark-app:latest`. Write a simple Kubernetes Job manifest (`k8s/spark-job.yaml`) to run your app. Deploy it with `kubectl apply -f k8s/spark-job.yaml` and check the logs with `kubectl logs`.
  - **Tasks (Week 21):** This is advanced but a huge differentiator. Install the Spark on Kubernetes Operator into Minikube using Helm. Convert your Job manifest to a `SparkApplication` custom resource. This is the modern, production way to run Spark on Kubernetes.
- **Week 22: Monitoring & Observability:**
  - **Setup:** Add Prometheus and Grafana services to one of your `docker-compose` files.
  - **Tasks:** In your PySpark job, add the `prometheus_client` library. Create a custom Counter metric. Increment it for each row processed. Start an HTTP server within your Spark driver to expose the metrics endpoint. Configure Prometheus to scrape this endpoint. Build a Grafana dashboard to plot your custom metric.
- **Week 23: Polishing the Portfolio:**

- **Tasks:** Your final communication push.
  - Review and rewrite your README.md to be world-class.
  - Add detailed docstrings to all Python functions and classes.
  - Record a 5-10 minute video using Loom or OBS, demonstrating the entire project end-to-end. Link it prominently in your README.
- **Week 24: The Promotion Narrative:**
  - **Tasks:** Create the SELF REVIEW.md document.
    - Go through your company's DE-3 competency matrix line by line.
    - For each line, write a bullet point linking to a specific piece of evidence from your repository.
    - Example: "Demonstrates Technical Depth -> I diagnosed and solved a Spark data skew issue, documenting the 10x performance improvement here: [Performance Tuning Doc] (docs/PERFORMANCE\_TUNING.md)".
    - This document is the script for your promotion discussion.

# Executive Summary: 24-Week Technical Bootcamp

# Executive Summary: 24-Week Technical Bootcamp for DE-3

---

## 1. What This Is About

This document outlines a 24-week, self-directed technical development program designed to bridge the gap between a Data Engineer 2 (Executor) and a Data Engineer 3 (Owner/Designer). The core of the program is the hands-on construction of a complete, production-grade big data pipeline from the ground up. Using technologies like Spark, Airflow, ClickHouse, Docker, and Kubernetes, this plan methodically builds a tangible, portfolio-grade project that serves as direct evidence of senior-level competencies.

---

## 2. Why Use This Plan? (The Value Proposition)

Standard career growth can be slow and unstructured, often relying on whatever projects become available at work. This plan provides a deliberate, parallel path to accelerate skill acquisition and demonstrate readiness for the next level.

- **From Theory to Practice:** Moves beyond watching tutorials or reading documentation into the realm of building and debugging a complex, integrated system.
  - **Structured and Measurable:** Provides a clear, week-by-week roadmap with concrete deliverables, eliminating ambiguity about what to learn next and ensuring consistent progress.
  - **Creates Tangible Proof:** The final output isn't just a claim of "I know Spark"; it's a working, well-documented GitHub repository that *proves* you can design, build, test, and deploy a modern data application.
  - **Develops Production-Ready Skills:** Focuses on the skills that define a senior engineer: system design, performance tuning, automation (CI/CD), infrastructure-as-code (Terraform/Kubernetes), and clear communication through documentation.
- 

## 3. What Will Be Achieved? (The Outcomes)

Upon completion of this 24-week program, the following concrete assets will be produced:

1. **A Fully Functional Big Data Pipeline:** An end-to-end, automated pipeline running locally that processes data with Spark, is orchestrated by Airflow, and loads results into a ClickHouse database.

2. **A Professional GitHub Repository:** A well-organized codebase that serves as a live portfolio of technical skills, demonstrating best practices in version control, project structure, and automation.
  3. **Production-Grade Design Documentation:** A comprehensive `SYSTEM_DESIGN.md` document outlining the cloud architecture, technology choices, and trade-offs, proving the ability to think strategically at scale.
  4. **Demonstrated Operational Excellence:** Verifiable skills in performance tuning, automated testing, CI/CD with GitHub Actions, and cloud-native deployment with Docker and Kubernetes.
  5. **A Compelling, Evidence-Based Promotion Narrative:** A `SELF REVIEW.md` document that explicitly maps every project artifact to the specific competencies required for a Data Engineer 3, creating an undeniable case for promotion.
- 

#### 4. Why This Approach is Superior

This plan is more effective than traditional self-study methods for demonstrating promotion readiness.

This Integrated Project Plan	Traditional Methods (e.g., Online Courses, Certifications)
<b>Holistic &amp; Integrated:</b> Teaches how to <b>integrate a full stack</b> of tools (Spark + Airflow + K8s) to solve a real business problem, which is the core work of a senior engineer.	<b>Siloed &amp; Isolated:</b> Teaches one tool at a time, often failing to address the complexities of making them work together in a real-world environment.
<b>Creates Verifiable Proof of Skill:</b> The final repository is <b>undeniable proof of ability</b> . A manager can see the code, the design docs, and the running application.	<b>Provides a Claim of Knowledge:</b> A certificate proves you passed a multiple-choice test. It doesn't prove you can build, debug, or optimize a real system.
<b>Focuses on the "Why":</b> Emphasizes system design, performance tuning, and architectural trade-offs—the <b>strategic thinking</b> that differentiates a DE-3 from a DE-2.	<b>Focuses on the "How":</b> Often teaches the syntax and basic usage of a tool, but not the critical thinking behind when and why to use it.
<b>Builds a Full-Spectrum Engineer:</b> Develops skills across the entire software lifecycle: <b>coding, testing, deploying, monitoring, and documenting</b> .	<b>Builds a Specialist:</b> May create depth in one specific tool but leaves gaps in operational and strategic skills required for senior roles.

In summary, this 24-week plan is not just about learning new technologies; it's a strategic program designed to **manufacture the evidence required for career advancement**.

# Architecture of the 24-Week Bootcamp Project

# Architecture of the 24-Week Bootcamp Project

---

A clear architectural vision is crucial for understanding how the pieces of the 24-week bootcamp fit together. The project is designed around two key architectures:

1. **The Local Development Architecture:** The physical system you will build and run on your personal machine using Docker. This is your hands-on sandbox.
2. **The Target Production Architecture:** The conceptual, cloud-native system you will design. This is the blueprint that demonstrates your senior-level (DE-3) architectural thinking.

Here is a detailed breakdown of both architectures.

This project is defined by two distinct but related architectures. The first is the tangible, local environment you will build for development and learning. The second is the scalable, cloud-native production system that the local environment is designed to emulate. Mastering the first proves your hands-on ability, while designing the second proves your strategic vision.

---

## 1. The Local Development Architecture (The Sandbox)

This is the system you will physically construct on your personal machine using Docker and Docker Compose. Its primary purpose is to provide a free, safe, and powerful environment for learning, development, and experimentation.

### Key Characteristics:

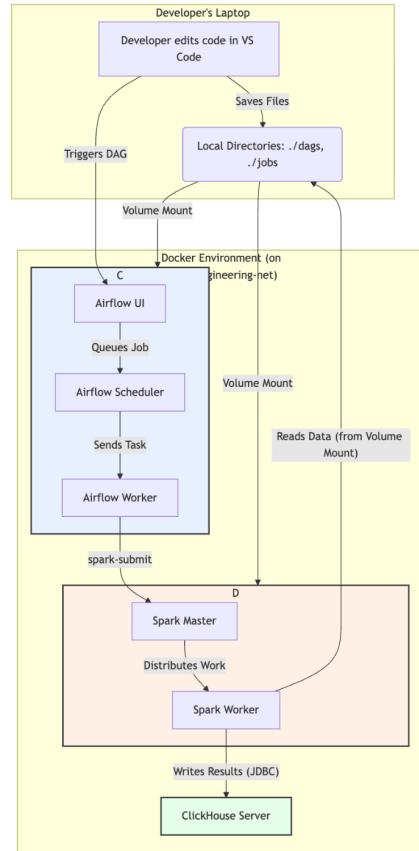
- **Environment:** Runs entirely on your personal machine inside Docker containers.
- **Cost:** Free (uses open-source software and local resources).
- **Purpose:** Skill acquisition, rapid development, testing, and debugging.

### Components:

- **Docker Desktop:** The virtualization layer that hosts all services.
- **Custom Docker Network (`data-engineering-net`):** A virtual network that allows all containers to discover and communicate with each other by name (e.g., the `airflow` container can ping the `spark-master` container).
- **Service Containers:**
  - **Airflow:** A multi-container setup (Webserver, Scheduler, Worker, Postgres DB, Redis) that acts as the "brain" or orchestrator of the entire system.
  - **Spark Cluster:** A `spark-master` container that coordinates work and one or more `spark-worker` containers that perform the actual data processing.
  - **ClickHouse Server:** A single-node container acting as the high-performance analytical database (the data warehouse).

- **Monitoring Stack (Optional but Recommended):** Prometheus and Grafana containers for observability.
- **Volume Mounts:** Local directories on your machine (`./dags`, `./jobs`, `./jars`) are mapped into the containers, allowing you to edit code locally and have it instantly reflected inside the running services.

### Data & Control Flow Diagram:



### Workflow Explanation:

1. **Development:** You write an Airflow DAG in `./dags` and a PySpark script in `./jobs` on your local machine.
2. **Orchestration:** You trigger the DAG via the Airflow UI. The Airflow Scheduler picks up the job and assigns the `SparkSubmitOperator` task to an Airflow Worker.
3. **Submission:** The Airflow Worker container executes a `spark-submit` command, sending the PySpark application to the Spark Master container.
4. **Processing:** The Spark Master distributes the processing tasks to the Spark Worker container(s).
5. **Data I/O:** The Spark Worker reads raw data (e.g., a CSV file) from a mounted local directory, processes it in memory, and writes the transformed results to the ClickHouse Server container via a JDBC connection.

## 2. The Target Production Architecture (The Blueprint)

This is the conceptual, cloud-native system you will design in **Weeks 5 and 13-14**. It represents how you would build a robust, scalable, and cost-effective version of your pipeline on AWS. This design is your primary artifact for demonstrating DE-3 level architectural competence.

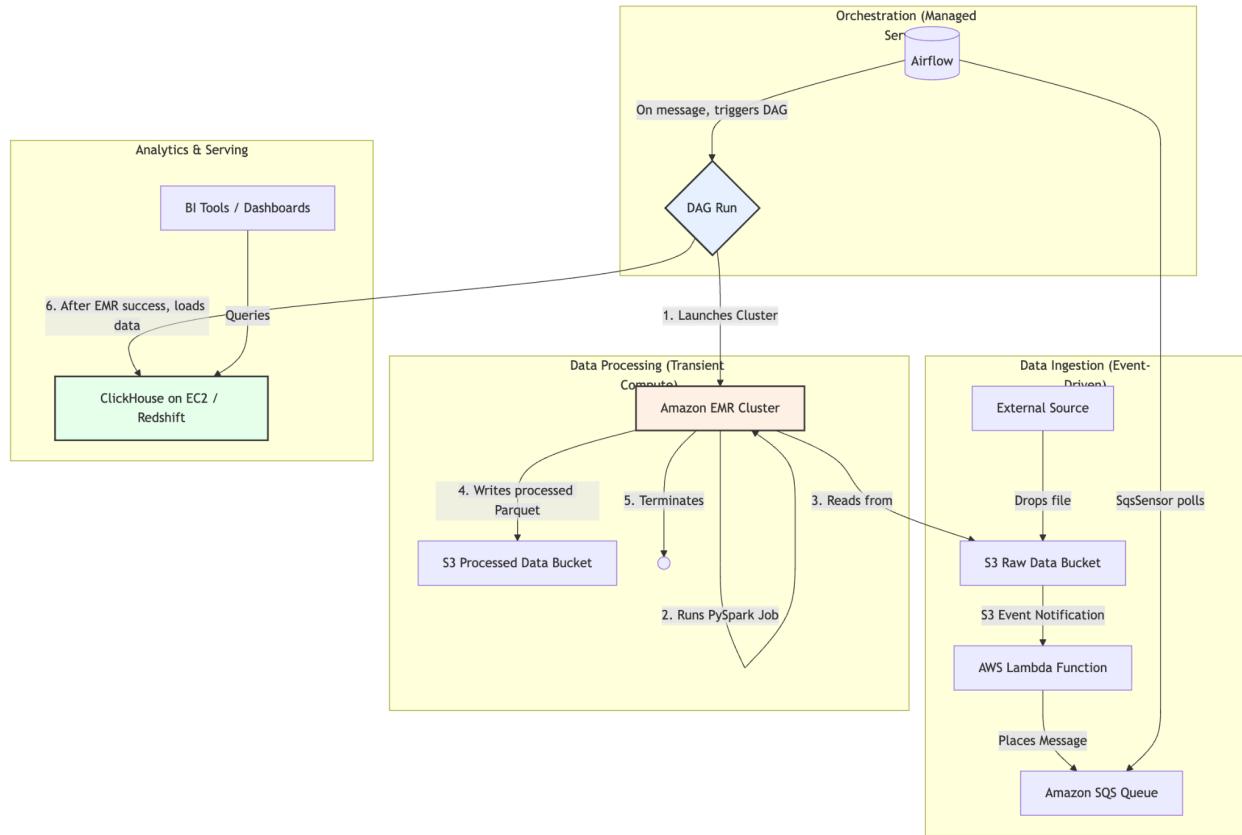
### Key Characteristics:

- **Environment:** Runs on AWS public cloud.
- **Cost:** Pay-as-you-go, optimized for cost-effectiveness (e.g., using serverless and transient resources).
- **Purpose:** To reliably process large-scale data in a production setting with automation, resilience, and security.

### Components:

- **Data Lake (Amazon S3):** The single source of truth. A `raw-data` bucket for immutable input files and a `processed-data` bucket for curated, query-ready datasets (e.g., in Parquet format).
- **Ingestion Trigger (S3 Events + Lambda + SQS):** An event-driven mechanism. A new file landing in S3 triggers a Lambda function, which validates the file and places a message into an SQS queue. This decouples ingestion from processing.
- **Orchestration (Amazon MWAA - Managed Workflows for Apache Airflow):** A fully managed Airflow service that runs your DAGs without you needing to manage the underlying servers.
- **Processing (Amazon EMR - Elastic MapReduce):** A **transient** (on-demand) Spark cluster. Airflow starts the cluster, it runs the job, and then it automatically terminates to save costs.
- **Analytics Database (ClickHouse on EC2 or Amazon Redshift):** A production-grade, scalable columnar database for serving analytical queries.
- **Infrastructure as Code (Terraform):** The entire architecture is defined in code, allowing for repeatable, automated deployments.
- **Security & Networking (IAM & VPC):** An IAM role with least-privilege permissions for the EMR cluster and a VPC to ensure all components communicate securely.

### Data & Control Flow Diagram:



### Workflow Explanation:

1. **Ingestion:** A data file lands in the `raw-data` S3 bucket.
2. **Trigger:** An S3 event triggers a Lambda function that sends a message with the file's location to an SQS queue.
3. **Sensing:** An Airflow DAG running on MWAA is constantly polling the SQS queue. When it finds a message, the DAG run starts.
4. **Launch Compute:** The first step in the DAG makes an API call to launch a transient EMR cluster with a predefined configuration (e.g., 1 master, 10 worker nodes using Spot Instances).
5. **Process Data:** The DAG submits the PySpark job to the newly created EMR cluster. The job reads the raw data from S3, transforms it, and writes the results as optimized Parquet files into the `processed-data` S3 bucket.
6. **Terminate & Save Costs:** Once the Spark job is complete, the EMR cluster automatically terminates.
7. **Load to Warehouse:** A final task in the Airflow DAG loads the clean data from the `processed-data` S3 bucket into the production ClickHouse/Redshift database, making it available for analytics.