Hands-on Lab: Generative AI for Infrastructure Setup

Estimated Effort: 30 minutes

Introduction

In the realm of modern data-driven enterprises, the choice of data infrastructure stands as a pivotal decision, influencing not only the efficiency of data management but also the organization's capacity for innovation and competitive advantage. In this hands-on lab, we delve into the transformative synergy between GenAI and data infrastructure, elucidating how GenAI technologies can revolutionize traditional data engineering practices.

Understanding the importance of selecting the right data infrastructure is paramount in navigating the complexities of contemporary data ecosystems. Furthermore, harnessing the power of generative AI introduces novel solutions to longstanding challenges, fostering agility, scalability, and adaptability within data operations. In this lab, you will explore various scenarios where GenAI intersects with data infrastructure, presenting real-world challenges and soliciting suggestions for optimal infrastructure setups.

About generative AI classroom lab

▶ Click here

Notes:

1. The prompts used in this lab are for your reference only. You can create your own prompts and generate responses using generative AI.
2. Since AI-generated outputs are dynamic, you may receive different responses even though you've used the same prompt from this lab.

Objective(s)

By the end of this lab, you will be able to use Generative AI using **GPT-5 Nano **to propose data infrastructure requirements for different practical scenarios across different industries covering the following scenarios:

- Retail industry
- Healthcare industry
- Finance industry
- Media and entertainment industry

Retail industry

Scenario

An e-commerce platform anticipates a significant increase in orders during the upcoming holiday season. Their current data infrastructure struggles to handle the expected surge in data volume, velocity, and variety. Therefore, the data engineers of the firm are proposing to enhance their data infrastructure. Discussions are underway in the aspects of scalable storage, better processing capabilities and real time analytics.

Prompt

To use generative AI for suggestive inputs on this problem statement, you can use the following prompt.

```
How should an E-commerce platform enhance their data infrastructure, such that they are able to handle increase in traffic. Suggest
1. scalable storage,
2. better processing capabilities and
3. real-time analytics.
```

This plan outlines architectural improvements across scalable storage, processing, and real-time analytics for an e-commerce platform. The recommendations are cloud-agnostic and can be adapted to AWS, GCP, or Azure.

1. Scalable storage
   Data lakehouse pattern: store raw and curated data in cloud object storage (e.g., S3, GCS, Azure Blob) with ACID-like features using Delta Lake / Apache Iceberg / Apache Hudi.
   Partitioning and data layout: partition by date, region, and business domain to enable pruning. Use columnar formats (Parquet/ORC) for analytics.
   Storage tiers and lifecycle: hot/warm/cold data with automatic tiering (e.g., S3 Intelligent-Tiering) and archival to cold storage (e.g., Glacier).
   Multi-region replication and DR: cross-region replication and point-in-time recovery for critical datasets.
   Data catalog and governance: centralized metadata (AWS Glue Data Catalog, Azure Purview, or Data Catalog) with lineage and schema evolution.
   Security and compliance: encryption at rest/in transit, fine-grained IAM, and data masking where needed.
   Backups and durability: regular backups, versioning, and tested disaster recovery plans.
   Data productization: create curated data products (fact tables, dimension tables, analytics views) to drive consistent reporting.
2. Better processing capabilities
   Decouple storage and compute: adopt serverless or autoscaling compute for ETL/ELT pipelines (e.g., AWS Glue, Databricks, Google Dataflow, Azure Data Factory) to scale on demand.
   Streaming and batch pipelines: ingest via a durable broker (Kafka/Kinesis/Pub/Sub) and process with Spark Structured Streaming or Apache Flink for real-time

updates, plus batch ETL for historical data.
Change data capture ((CDC): use CDC to capture transactional changes and propagate to warehouses and data lake efficiently.
Processing engines and orchestration: use Spark/Flink for heavy workloads; orchestrate with Airflow / Dagster / Prefect and stage data through industry-standard ELT patterns.
Data modeling: use a star/snowflake schema in a data warehouse; maintain slowly changing dimensions (SCD) to support business history.
Caching and acceleration: cache hot query results in Redis or Memcached; use materialized views and query acceleration features where available.
Observability and reliability: implement idempotent producers, exactly-once processing semantics where possible, robust retries, and end-to-end monitoring with metrics and traces.

3. Real-time analytics
Real-time ingestion and delivery: stream events (orders, payments, views) into a streaming pipeline (Kafka/Kinesis) and feed downstream systems with low latency.
Real-time processing: use Flink or Spark Structured Streaming for windowed aggregations (tumbling/sliding windows) and near real-time dashboards.
Real-time querying: consider ksqlDB or continuous queries on your data lake/warehouse for low-latency insights; maintain materialized views for common metrics.
Data warehouse integration: use streaming inserts to Snowflake/BigQuery/Aurora Redshift or Lakehouse equivalents to keep dashboards current.
Dashboards and alerting: integrate with Looker/Tableau/Power BI and set latency targets (e.g., sub-5-second freshness for critical metrics, sub-1-minute for dashboards).
Data quality and privacy: enforce schema validation on streams, masking for PII, and secure access controls.
ML-ready data: feed real-time signals to inference models for live recommendations and fraud detection.
Reliability: implement backpressure handling, replay logs, and end-to-end data lineage for trusted analytics.

Healthcare industry

Scenario

A healthcare provider wants to establish a central data repository for research purposes. This data lake would hold patient data, medical records, and research studies. The data engineers of the firm are discussing implementation of big data management tools, data ingestion and transformation capabilities and data security and compliance guidelines.

Prompt

To use generative AI for suggestive inputs on this problem statement, you can use the following prompt:

```
How should a healthcare company set up a data lake infrastructure that is capable of the following.
1. Big data management
2. data ingestion from various sources
3. data transformation
4. data security and compliance to regulatory guidelines
```

A robust healthcare data lake should balance scale, interoperability, privacy, and regulatory compliance. This cloud-agnostic plan outlines a structured approach to big data management, diverse data ingestion, transformation, and strong security controls aligned with regulatory guidelines (e.g., HIPAA/HITRUST, GDPR where applicable). The architecture follows a lakehouse pattern with clear data layers, canonical models (FHIR, OMOP CDM), and governance embedded into pipelines.

Key principles:

Cloud-agnostic, portable design with cloud-native optimization where appropriate
Clear data maturity layers: Raw, Curated, Analytics
Canonical healthcare models to enable interoperability and reuse
Data governance, lineage, and quality enforced at every stage
Privacy-by-design, robust access controls, and auditable security controls

1. Big data management
Architecture and layers
Data lakehouse paradigm: use a cloud object store as the central landing zone with ACID-like semantics provided by Delta Lake, Apache Iceberg, or Apache Hudi.
Layering:
Raw: immutable ingestion zone containing the source data as-is (with minimal normalization).
Curated: normalized representations (FHIR resources, OMOP CDM) with schema evolution tracked.
Analytics: denormalized, query-optimized datasets (star/snowflake schema where appropriate) and materialized views.
Canonical data models and interoperability:
Prefer FHIR resources for clinical data interchange and OMOP CDM for analytics harmonization.
Maintain mappings from source systems (EHRs, LIS, RIS, imaging, claims) to canonical models with documented lineage.
Data formats and partitioning:
Columnar formats (Parquet/ORC) for analytics efficiency.
Partition by domain (patient, encounter, provider), data domain (clinical, administrative, imaging), and date to enable pruning.
Data catalog and governance:
Central metadata repository (Open Metadata, Apache Atlas, or vendor equivalents).
Lineage tracking, schema evolution, data classifications (PHI/PII/PIII), data quality metadata, and access policies.
Data quality and validation:
Define data quality rules and automated checks (e.g., Great Expectations) to catch drift and anomalies.
Security and lifecycle:
Encryption at rest and in transit; lifecycle tiering and automatic data aging (hot/warm/cold) with archival options.
Immutable backups, versioning, and tested disaster recovery playbooks.
DR and multi-region readiness:
Cross-region replication for critical datasets and point-in-time recovery.
Data productization:
Curated data products (fact tables, dimension tables, analytics views) published with SLAs to promote reuse and governance.

Observability and reliability:
End-to-end monitoring, idempotent producers, and robust retry/at-least-once or exactly-once semantics where feasible.
Operational governance
Data contracts and access controls at the dataset level.
Data retention policies aligned with clinical and research needs.
Audit logs and security events routed to SIEM for compliance monitoring.

2. Data ingestion from various sources
Source diversity and ingestion patterns
Ingest from diverse healthcare sources:
EHRs (HL7 v2/v3, FHIR), LIS, RIS, imaging (DICOM), insurance claims, patient-generated data, IoT/medical devices, and external datasets (public health, research data).
Ingestion patterns:
Combine batch ETL/ELT with streaming CDC and event-based ingestion for near real-time needs.
Use durable transports (Kafka, Kinesis, Pub/Sub) with at-least-once or exactly-once delivery semantics.
Canonical ingestion target:
Normalize incoming data to canonical representations (FHIR resources or OMOP CDM) before storage to simplify downstream processing.
Connectors and adapters:
Build/reuse connectors for HL7, FHIR, DICOM, JDBC/ODBC, and REST APIs.
Validate schema and enforce field-level privacy controls during ingestion.
CDC and near-real-time ingestion:
Use CDC to propagate transactional changes with low latency for relational sources.
Data validation at the edge:
Basic schema/schema drift checks, mandatory field validation, PHI masking before loading to lakehouse. Dead-letter queues for failed records and backoff retries.
Security and governance during ingestion:
Encrypted connections, data tagging (PHI/PII), and provenance tracking for audits.
Orchestration and scheduling:
Orchestrate ingestion pipelines with Airflow, Dagster, or Prefect; ensure idempotency, retries, and dependency management.
Data lineage and cataloging:
Capture source provenance, map to canonical models, and register datasets in the data catalog with classifications and retention policies.
Data quality and validation during ingestion
Early validations reduce downstream rework:
Schema conformance, mandatory field checks, basic data quality signals.
Route invalid records to dead-letter systems with retry and human review workflows.

3. Data transformation
ELT approach and core transformations
ELT to maximize scalability:
Load raw data first, then apply transformations to curated and analytics layers using Spark or Flink.
Canonical model alignment:
Transform data into FHIR representations or OMOP CDM with careful mapping of clinical terminologies (ICD-10, CPT/LOINC/SNOMED) and units.
Harmonization and normalization:
Standardize terminologies, units, and value formats; apply SCD Type 2 for patient history to preserve lineage.
De-identification and privacy-preserving transforms:
Apply masking, tokenization, hashing, or differential privacy where PHI is not needed for analytics. Maintain risk assessments for re-identification.
Data quality engineering:
Enforce schema validations, datatype checks, completeness metrics, and anomaly detection; gate data movement based on quality results.
Data lineage and versioning:
Track all transformation steps, dataset versions, and lineage for audits and reproducibility.
Performance optimization:
Partitioning, clustering, and materialized views; incremental/streaming updates for near-real-time datasets.
Governance and access control:
Enforce column-level and row-level security; log all transformation activities.
Testing and validation:
Implement unit and integration tests for transformation logic; regression tests to ensure pipeline stability over time.

4. Data security and compliance with regulatory guidelines
Regulatory posture and risk management
Regulatory scope:
HIPAA/HITRUST baseline with readiness for SOC 2, GDPR/PPPA as applicable for cross-border data.
PHI/PII handling:
Data minimization, least-privilege access (RBAC/ABAC), and classification of PHI/PII/PIII with policy-driven masking/tokenization.
Encryption and key management:
Encryption at rest and in transit (TLS). Key management via KMS-backed keys; consider HSM-based key stores for highly sensitive data.
Identity and access management:
Strong IAM with MFA, fine-grained permissions, automated provisioning/deprovisioning, and auditable access reviews.
Network and data residency:
Isolated environments (VPCs/private endpoints), regional data residency policies, and data replication aligned with regulatory requirements.
Data auditing and monitoring:
Immutable audit logs for access and transformations; integrated with SIEM; defined retention aligned with regulations.
Data retention and erasure:
Retention policies aligned with clinical/research needs; automated archival and secure deletion; support for data subject rights where applicable.
Data sharing and consent:
Manage patient consent and data-sharing agreements; enforce consent constraints and data-use limitations in pipelines.
Incident response and disaster recovery:
Runbooks for security incidents, breaches, and DR events; regular drills with stakeholders.
Privacy-by-design and DPIA:
DPIAs for new data sources and transformations; apply privacy-preserving techniques as appropriate.
Documentation and governance:
Maintain policies, data lineage, catalog classifications, and evidence to support audits and inquiries.

5. Implementation guidance and next steps
Start with a minimal viable data lakehouse for a representative clinical domain (e.g., cardiology or oncology) to validate canonical models and ingestion pipelines.
Establish data contracts between sources and consumers, including SLAs for data freshness and quality.
Define a data governance council, data steward roles, and incident response ownership.

Choose an initial tech stack aligned with the cloud strategy (or keep it cloud-agnostic) and gradually enable cross-cloud interoperability.
Implement a phased roadmap: discovery and design, pilot, scale-out, and continuous improvement with measurable metrics.

6. Reference architecture (conceptual)
   Ingestion layer: connectors from EHRs/LIS/RIS/DICOM/claims to a durable transport (Kafka/Kinesis) feeding the lakehouse landing zone.
   Storage layer: object storage as Raw; DACs (Delta/Iceberg/Hudi) for ACID semantics; curated and analytics layers with partitioned Parquet/ORC.
   Computation layer: Spark/Flink processing for ELT, with notebooks/jobs for transformations and quality checks.
   Governance layer: data catalog, lineage, policy enforcement, data quality framework, and access controls.
   Security layer: encryption, KMS/HSM, private networking, access audits, and DPIA-driven masking/tokenization.
   Consumption layer: BI/analytics tools and patient/partner portals with controlled data access based on roles and data classifications.
   Key success metrics (to define O12s)
   Data freshness: ingestion to analytics latency targets by domain (e.g., near-real-time for clinical events, daily for claims).
   Data quality: percentage of datasets passing automated quality gates.
   Security/compliance: time to detect and respond to incidents; completeness of audit logs;
   Privacy: rate of successful de-identification and masking without compromising analytics usefulness.
   This blueprint provides a scalable, governance-focused, cloud-agnostic foundation for a healthcare data lake that supports big data management, multi-source ingestion, robust transformation, and stringent security/compliance controls.

7. Big data management
   Data lakehouse paradigm with Raw/Curated/Analytics layers and ACID-like semantics (Delta Lake, Apache Iceberg, or Apache Hudi).
   Canonical models (FHIR and OMOP CDM) with clear mappings from source systems and recorded lineage.
   Partitioning by domain, data domain, and date; Parquet/ORC formats for analytics.
   Central data catalog, lineage tracking, and governance metadata.
   Data quality rules (e.g., Great Expectations) and automated quality gates.
   Security, encryption, tiered storage, DR readiness, and data productization with SLAs.
   Observability, retries, idempotent producers, and end-to-end monitoring.

8. Data ingestion from various sources
   Ingest EHR (HL7/FHIR), LIS, RIS, DICOM, claims, patient-generated data, IoT devices, and external datasets.
   Combine batch/ELT with streaming (CDC, event streams) using durable transports (Kafka/Kinesis).
   Normalize to canonical models (FHIR/OMOP) at ingestion or shortly after.
   Build and reuse connectors; enforce schema validation and field-level privacy on ingest.
   CDC for low-latency propagation of transactional changes.
   Edge validation, dead-letter queues, retries, provenance tagging, and governance classification.
   Orchestrate with Airflow/Dabster/Prefect; ensure idempotency and dependency tracking.

9. Data transformation
   ELT approach: load raw data, transform to curated/analytics representations.
   Align with canonical models; map clinical codes (ICD-10, CPT, LOINC, SNOMED).
   Harmonize terminologies, units, and formats; apply SCD Type 2 for patient history.
   De-identification and privacy-preserving transforms; risk assessments for re-identification.
   Data quality engineering and validation gates; lineage tracking and versioning.
   Performance optimizations: partitioning, clustering, materialized views; incremental processing.
   Access control enforcement and transformation auditing.

10. Data security and compliance with regulatory guidelines
    Align with HIPAA/HITRUST; prepare for SOC 2, GDPR as needed for cross-border usage.
    PHI/PII handling: data minimization, RBAC/ABAC, masking/tokenization, data classifications.
    Encryption at rest/in transit; KMS-backed keys; HSM support for sensitive data.
    Strong IAM, MFA, automated provisioning/deprovisioning, audit-friendly access reviews.
    Network isolation, private endpoints, data residency policies, and compliant data replication.
    Immutable audit logs; SIEM integration; log retention per regulation.
    Data retention and erasure policies; support for data subject rights where applicable.
    Consent management, data-sharing controls, DPIA for new data sources, and incident response runbooks.
    Documentation and governance artifacts to support audits and inquiries.
    Implementation guidance and next steps
    Start with a minimal viable data lakehouse for a representative clinical domain to validate canonical models and ingestion pipelines.
    Define data contracts, SLAs, and governance roles.
    Select an initial technology stack with a plan to scale; maintain cloud-agnostic options where feasible.
    Establish a phased roadmap: discovery/design, pilot, scale, and continuous improvement with measurable metrics.

Finance industry

Scenario

A financial services company wants to implement machine learning models to detect fraudulent transactions in real time. This requires building and maintaining a robust data infrastructure to support the models. This requires data engineers to set up high performance computing machinery, feature engineering pipelines, predictive modeling, and model deployment and monitoring.

Prompt

To use generative AI for suggestive inputs on this problem statement, you can use the following prompt.

```
How should a financial firm set up their infrastructure if they want to detect fraudulent transactions in real time? Suggest specifi
1. Computing machinery
2. Feature engineering pipeline
3. Predictive modeling pipeline
4. Model deployment and monitoring
```

This blueprint outlines a practical, real-time fraud detection infrastructure for a financial firm. It covers four core areas—computing machinery, feature engineering, predictive modeling, and model deployment/monitoring—with concrete, implementable recommendations aimed at sub-second decision latency, high throughputs, robust data quality, and strong governance.

1. Computing machinery
   Latency and throughput targets:
   Target end-to-end latency from transaction event to decision: sub-200 ms for the scoring path; sustained throughput to handle peak transaction volumes with headroom.
   Provide reservations for burst traffic and backpressure handling.
   Real-time data plane:
   Use a durable streaming backbone (Apache Kafka or AWS Kinesis) as the central event bus.
   Event-time processing to avoid late-arriving data skew impacting scoring.
   Compute tiers and platforms:
   Inference/online scoring layer: containerized microservices running on Kubernetes. Prefer CPU-optimized instances for standard ML models (LR, tree ensembles); reserve GPUs for heavier deep-learning or large ensemble models if required.
   Streaming processing: Apache Flink (preferred for low-latency event processing) or Spark Structured Streaming as an alternative. Run on autoscaling clusters (Kubernetes or cloud-managed) with backpressure-aware operators.
   Training/offline compute: distributed engines (Databricks/EMR/Dataproc) with GPU support where appropriate for heavy models, running on autoscaling clusters.
   Online feature store and serving:
   Fast online feature access via a feature store (e.g., Feast) backed by in-memory stores (Redis/RedisGears) or low-latency databases (DynamoDB, Redis, Cassandra) with multi-AZ replication.
   Ensure feature TTLs and cache invalidation policies match the business needs.
   Data storage and data locality:
   Separate storage for raw/curated data (data lake) and hot, frequently accessed features/state (online store).
   Wherever possible, co-locate compute with data or use high-throughput private networking to minimize latency.
   Reliability and observability:
   Use multi-AZ deployments, automated failover, and health checks; implement circuit breakers and retry/backoff with idempotent operations.
   Centralized observability stack (Prometheus/Grafana, OpenTelemetry, centralized logging) and distributed tracing to diagnose latency hotspots.
   Security and compliance:
   Network segmentation (VPCs/private endpoints), mTLS for service-to-service calls, encryption at rest/in transit, and role-based access controls aligned with PCI-DSS and internal governance.
   Operational patterns:
   Canary/blue-green deployment for inference services, with shadow testing before routing real traffic.
   Cost management through autoscaling, spot instances for non-critical batch tasks, and right-sized instances for latency targets.
   Recommended starter tech stack (example):
   Streaming: Kafka, Flink
   Online store: Redis (clustered) + Feast
   Inference service: Kubernetes with gRPC/REST endpoints
   Training: Databricks or EMR
   Monitoring: Prometheus + Grafana + OpenTelemetry + ELK/EFK stack
   Security: IAM, KMS/HSM, VPC/private endpoints
2. Feature engineering pipeline
   Feature categories:
   Transaction-based features: amount, currency, merchant category, card type, merchant country, time of day.
   Device and network features: device fingerprint, IP geolocation, user-agent signals, VPN/proxy indicators.
   Velocity and risk features: counts of transactions per account, per card, per merchant in sliding windows; time-since-last-transaction; velocity risk scores.
   Contextual features: customer risk tier, prior fraud flags, account age, geographic clustering.
   Real-time vs batch features:
   Real-time features: derived from the current event stream (last few seconds/minutes) and cached in the online feature store for sub-100 ms reads.
   Batch features: longer-horizon aggregations (hourly/daily) pre-computed and stored for offline training and occasional refresh of online features.
   Windowing and aggregation strategies:
   Rolling windows (1 min, 5 min, 1 hour) for counts, unique counts, averages, and standard deviations.
   Time-decayed weighting to de-emphasize older activity.
   Feature store and governance:
   Use an online feature store (e.g., Feast) to expose low-latency features to the scoring service; maintain separate offline feature stores for training data.
   Version features, document feature definitions, and track lineage from source to feature to model input.
   Enforce feature validation and schema checks; prune stale features and monitor feature drift.
   Data quality and privacy:
   Validate schema conformance at ingestion; apply data masking/tokenization for sensitive fields where not required for scoring.
   Enforce data retention policies and audit-worthy access controls on feature data.
   Operational practices:
   Idempotent feature generation steps; handle late-arriving events gracefully with watermarking.
   Automated tests for feature transformations and regression checks when feature definitions change.
   Observability:
   Track feature freshness, cache hit rates, and latency of feature lookups; alert on degraded online feature availability.
3. Predictive modeling pipeline
   Model types and usage:
   Baseline models: logistic regression for interpretability and fast inference.
   Tree-based ensembles: XGBoost/LightGBM for higher accuracy on tabular data.
   Online learning models: SGD-based classifiers or Vowpal Wabbit for rapid adaptation to new data; consider brief online updates where feasible.
   Hybrid approaches: combine a fast online model with a more accurate batch model (scored in a two-stage pipeline) to balance latency and accuracy.
   Training data and features:
   Use offline historical labeled data (labeled fraud vs. legitimate) with both online and offline features.
   Include feedback loops from actual outcomes (true positive/false positive/false negative) to improve labels.
   Training and evaluation workflow:
   Regular offline training (daily or weekly) with holdout/temporal validation; report AUC, precision@k, recall, F1, and business metrics (false positive rate, revenue impact).
   Use time-based cross-validation to reflect non-stationarity in fraud patterns.
   Calibrate probability scores to align with business risk thresholds (cost-sensitive calibration).
   Feature pipeline integration:
   Ensure models consume the same online/offline feature representations used in production.
   Drift detection and retraining:
   Monitor feature distribution drift, target drift, and performance decay; trigger retraining when drift or degradation crosses thresholds.

Deployment readiness:
Package models with metadata in a model registry (versioning, provenance, performance metrics).
Validate models against a reserved test set and performance gates before production rollout.
Privacy and governance:
Protect sensitive inputs; ensure compliant use of data for fraud analysis; maintain auditability of model inputs and decisions.

4. Model deployment and monitoring
Deployment strategies:
Canary or blue/green rollouts to gradually shift traffic to new models; retain the previous version as a rollback option.
Shadow deployment to evaluate new models against live data without affecting decisions.
Serving architecture:
Real-time inference service exposing low-latency endpoints (REST or gRPC) that accept feature vectors and return risk scores.
Tight coupling with the online feature store to retrieve up-to-date features for scoring.
Separate decisioning service for business logic (e.g., approval/deny actions, exemptions) that can integrate with rule engines.
Model registry and lifecycle:
Use a model registry (MLflow, Sagemaker Model Registry, Vertex AI Model Registry) to track versions, metrics, provenance, and deployment status.
Define promotion gates (accuracy, latency, drift) before moving to production.
Monitoring and observability:
Real-time performance metrics: AUC, precision, recall, F1, false positive rate, false negative rate, uplift in fraud catch rate.
Operational metrics: latency per inference, requests per second, error rates, queue depths, and saturation levels.
Data drift and feature drift: monitor distributions of inputs and features; alert on significant shifts.
Business impact monitoring: monitor post-deployment fraud losses avoided vs. false positives and customer friction metrics.
Alerting and incident response:
Alerts for model performance degradation, latency spikes, or system outages; runbooks for rollback and remediation.
Retraining and governance:
Define retraining cadence or drift-triggered retraining; ensure retrained models pass backtests and governance checks before deployment.
Security and compliance:
Ensure secure model artifacts, restricted access to model endpoints, encryption in transit and at rest, and audit logs for decisions and feature access.
Operational best practices:
Maintain a layered defense: deterministic rules for obvious fraud signals, ML-based scoring for probabilistic risk, and human-in-the-loop review for edge cases.
Document rollback plans, test coverage, and disaster recovery procedures for the inference pipeline.

Practice - Media and entertainment industry

Scenario

A media and entertainment company wants to personalize user experiences by recommending content based on individual preferences. This requires building and maintaining a complex data infrastructure to support the recommendation engine. The data engineers are therefore discussing infrastructural challenges in terms of real-time data ingestion, data warehousing, parallel processing frameworks, and machine learning model development for this task.

Prompt

You are encouraged to create a prompt for this scenario and generate the infrastructure requirements for the task.

Conclusion

Congratulations on completing this lab.

By now, you are able to use generative AI to create infrastructure requirements for different industries in terms of different scenarios in each of them.

Author(s)

[Abhishek Gagneja]