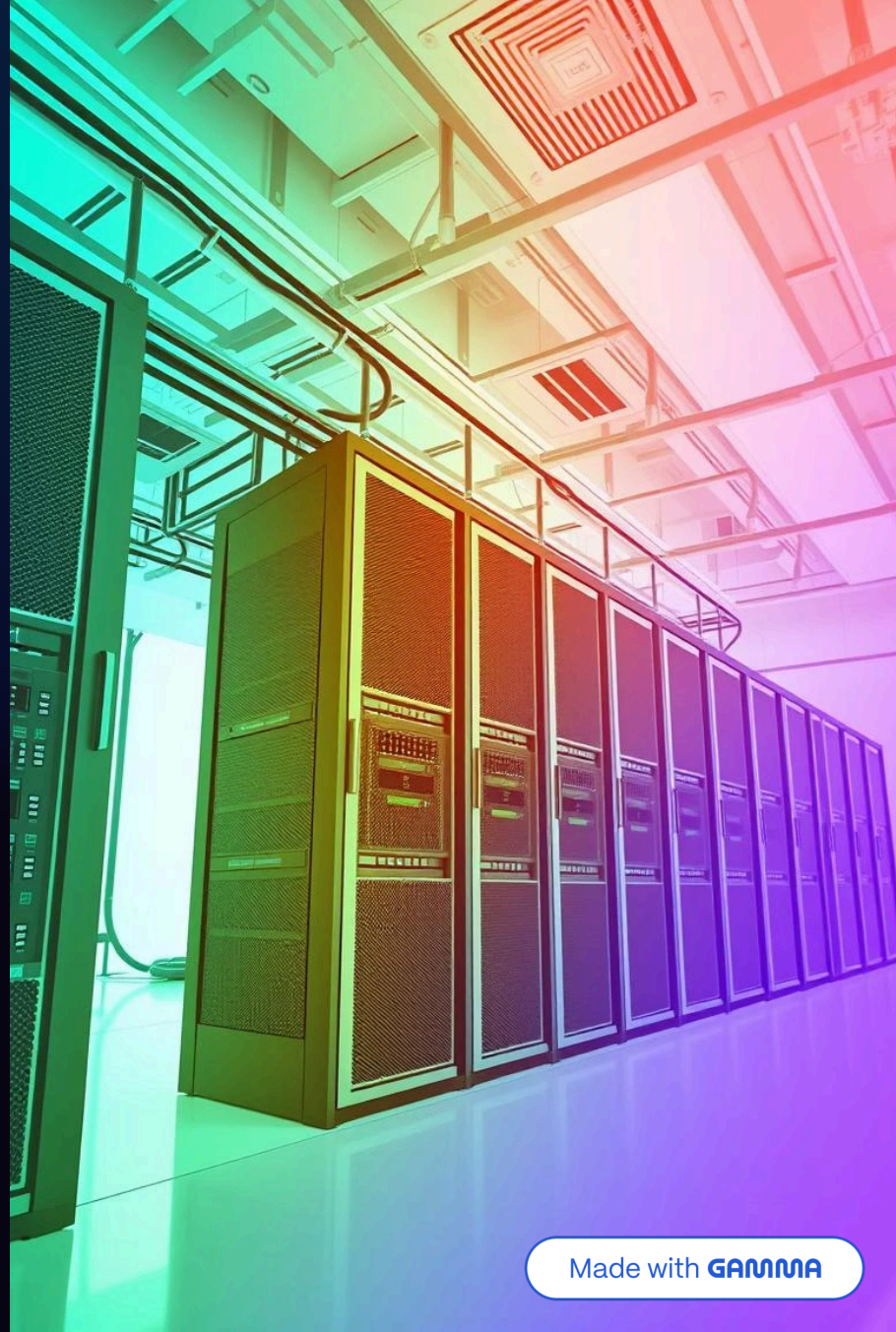
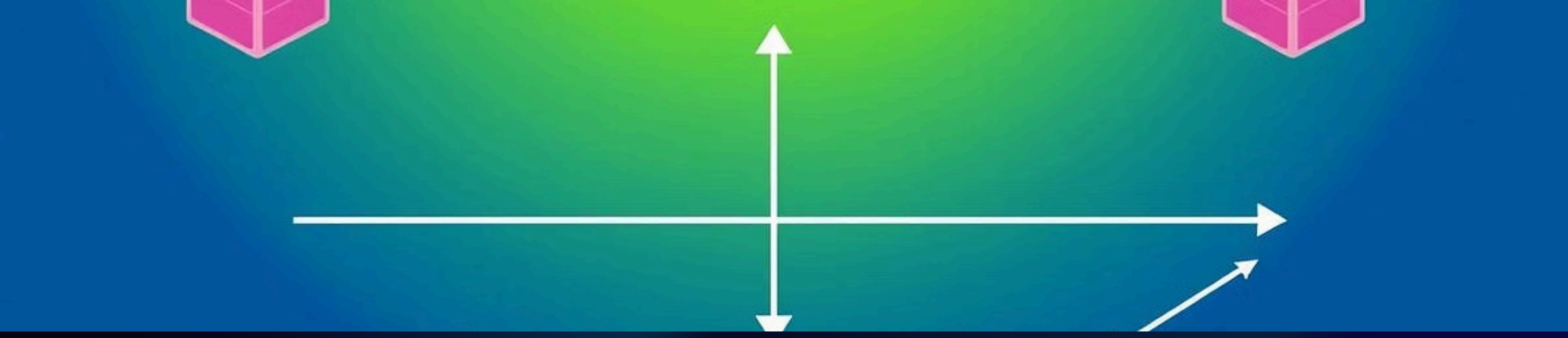
Project Deployment and Data Model Architecture

In this presentation, we provide a comprehensive, step-by-step documentation of the full deployment and usage flow for our project infrastructure. We'll explore how the components like Docker containers, Hadoop, HBase, and ZooKeeper integrate seamlessly, and explain the reasoning behind our data model design.

H **by Hassan Marzouk**





Step-by-Step Deployment Flow and Cluster Setup

1

Build Docker Environment

Create a standardized container image with Hadoop, HBase, ZooKeeper, configs, and scripts to ensure uniform deployment.

2

Deploy Cluster with Docker Compose

Set up a ZooKeeper quorum, Hadoop daemons, and HBase services on isolated but interconnected containers for coordinated operation.

3

Start Services Sequentially

Run the startup script to initialize ZooKeeper, then Hadoop, followed by HBase components, respecting dependencies for stability.

4

Create HBase Schema

Use the HbaseTables.sh script to define a modular table schema with four column families partitioning data for optimal access.

Data Ingestion and Cluster Operation Processes

Generate Synthetic Data

Produce sample web page data including HTML content, metadata, and links using dataGen.py for testing and demonstration.

Load Data into HBase

Insert generated data through a script that processes lines and issues put commands for distribution across column families.

Query and Analyze

Run targeted scans and filters on the webTable through shell or API calls to assess content, metadata, and link relationships.

Test High Availability and Failover

Simulate node failures to validate cluster robustness and automatic failover mechanisms, monitoring logs and UI feedback.

Performance Tuning

Optimize HDFS and HBase configurations such as replication, caching, and bloom filters to enhance cluster efficiency.

Rationale Behind webTable Data Model Design

Column Family Separation

The webTable uses distinct column families: content for heavy HTML data, meta for crawl info, and outlinks/inlinks for link graphs. This enables targeted access patterns and efficient caching.

Row Key and Versioning

Row keys are salted or hashed URLs to prevent hotspotting, improving data distribution and query speed. Versioning and TTL settings support historical data retention and automated cleanup.

This design balances scalability, query efficiency, and data management, addressing common challenges in web crawling and search index architectures. It enables selective reads and writes, optimizing performance across use cases.