Open-Source Project

Deployment Guide

HCLS AI Factory

Deployment and Configuration Guide
for NVIDIA DGX Spark

Open-Source Precision Medicine Platform
on NVIDIA DGX Spark

02/2026 | Version 1.0 | Apache 2.0 License
Author: Adam Jones

Table of Contents

1. Introduction

1.1 Purpose

This document provides step-by-step instructions for deploying the HCLS AI Factory on an NVIDIA DGX Spark workstation. It covers all three pipeline stages — genomics, RAG-powered variant intelligence, and AI-driven drug discovery — using exclusively open-source and publicly available components.

1.2 Scope

The guide addresses hardware validation, software installation, container deployment, data preparation, pipeline execution, monitoring, security, and troubleshooting. It targets the open-source fork of the HCLS AI Factory that runs entirely on Docker Compose without requiring VAST Data, Kubernetes, or multi-node infrastructure.

1.3 Audience

- **Bioinformatics Engineers** deploying genomics pipelines on DGX Spark
- **ML/AI Engineers** integrating RAG and BioNeMo NIM microservices
- **DevOps Engineers** managing containerized service stacks
- **Researchers** forking the project for their own precision medicine workflows

1.4 Document Conventions

Convention	Meaning
`monospace`	Commands, file paths, code
Bold	UI elements, key terms
Italic	Variable values to be replaced
`$VARIABLE`	Environment variable
`<placeholder>`	User-supplied value

1.5 Genomics and Drug Discovery Primer

This section provides essential background for engineers who may not have a biology or chemistry background.

1.5.1 DNA Sequencing

DNA sequencing reads the order of nucleotide bases (A, T, C, G) in an organism's genome. Modern short-read sequencers (e.g., Illumina) produce paired-end reads — two sequences from opposite ends of a DNA fragment. The standard demo sample HG002 is a 30x whole-genome sequencing (WGS) dataset with 2x250 bp paired-end reads, producing approximately 200 GB of FASTQ data.

1.5.2 Genomics Pipeline Stages

Stage	Input	Tool	Output	Description
Quality Control	FASTQ	FastQC	QC Report	Assess read quality and adapter contamination
Alignment	FASTQ + Reference	BWA-MEM2 (fq2bam)	BAM	Map reads to GRCh38 reference genome
Variant Calling	BAM	DeepVariant	VCF	Identify SNPs and indels vs. reference
Annotation	VCF	VEP + ClinVar + AlphaMissense	Annotated VCF	Add functional, clinical, and pathogenicity data
Embedding	Annotated VCF	BGE-small-en-v1.5	Vectors (384-dim)	Convert variant evidence to dense embeddings

1.5.3 Variant Annotation

Variants are annotated from multiple sources:

- **VEP (Variant Effect Predictor):** Assigns functional consequences and impact levels — HIGH, MODERATE, LOW, or MODIFIER.
- **ClinVar:** NCBI database of 4.1 million clinical variant interpretations (Pathogenic, Likely Pathogenic, Benign, etc.).
- **AlphaMissense:** DeepMind model with 71,697,560 missense variant pathogenicity predictions. Thresholds: pathogenic (>0.564), ambiguous (0.34-0.564), benign (<0.34).

1.5.4 Vector Embeddings and RAG

Annotated variants are converted to 384-dimensional dense vectors using the BGE-small-en-v1.5 embedding model and stored in Milvus. Retrieval-Augmented Generation (RAG) queries Milvus for relevant genomic evidence, then passes the results as context to Anthropic Claude for natural-language clinical interpretation.

1.5.5 Drug Discovery Pipeline

The 10-stage drug discovery pipeline transforms a genomic target into ranked drug candidates:

Stage	Name	Description
1	Initialize	Load configuration, validate target gene and variant
2	Normalize Target	Map gene symbol to UniProt ID and canonical name
3	Structure Discovery	Query RCSB PDB for 3D protein structures, score by resolution and method
4	Structure Preparation	Download PDB files, extract binding site coordinates
5	Molecule Generation	Generate SMILES candidates via MolMIM NIM (Port 8001) using seed molecule
6	Chemistry QC	Filter by Lipinski Rule of Five (MW<=500,

		LogP<=5, HBD<=5, HBA<=10)
7	Conformer Generation	Generate 3D conformers with RDKit for docking input
8	Molecular Docking	Score binding affinity via DiffDock NIM (Port 8002)
9	Composite Ranking	Rank candidates: 30% generation + 40% docking + 30% QED
10	Reporting	Generate PDF report with structures, scores, and recommendations

1.5.6 End-to-End Data Flow Summary

```
FASTQ (200 GB) → Parabricks fq2bam → BAM (100 GB) → DeepVariant → VCF (11.7M variants)
    → Annotation (ClinVar + AlphaMissense + VEP) → Milvus (384-dim vectors)
    → Claude RAG (variant interpretation) → Target Hypothesis
    → PDB Structure Retrieval → MolMIM (molecule generation)
    → DiffDock (molecular docking) → Composite Ranking → PDF Report
```

2. Architecture Overview

2.1 System Components

The HCLS AI Factory comprises three application pipeline stages running on a single DGX Spark:

Stage	Name	Function
Stage 1	Genomics Pipeline	FASTQ alignment and variant calling with GPU-accelerated Parabricks
Stage 2	RAG Chat Pipeline	Variant annotation, vector embedding, and Claude-powered conversational AI
Stage 3	Drug Discovery Pipeline	Structure-aware molecule generation, docking, and composite ranking

2.2 Technology Stack

Layer	Technology	Version / Details
Hardware	NVIDIA DGX Spark	GB10 GPU, 128 GB unified LPDDR5x, 144 ARM64 cores
OS	DGX OS	Ubuntu-based, ARM64 (aarch64)
Container Runtime	Docker + NVIDIA Container Toolkit	nvidia-docker runtime
Orchestration	Docker Compose	Multi-service deployment
Pipeline Orchestration	Nextflow	DSL2, multiple profiles
GPU Genomics	NVIDIA Parabricks	4.6.0-1

Vector Database	Milvus	2.4 (with etcd + MinIO)
Embedding Model	BGE-small-en-v1.5	384 dimensions
LLM	Anthropic Claude	claude-sonnet-4-20250514
Molecule Generation	BioNeMo MolMIM NIM	1.0
Molecular Docking	BioNeMo DiffDock NIM	1.0
Cheminformatics	RDKit	Python library
Monitoring	Grafana + Prometheus	10.2.2 / v2.48.0
GPU Monitoring	DCGM Exporter	Port 9400
Language	Python	3.10+

2.3 Service Architecture

The platform deploys 14 services across 14 ports:

#	Service	Port	Protocol	Description
1	Landing Page	8080	HTTP	Platform entry point and service directory
2	Genomics Portal	5000	HTTP	Genomics pipeline UI and results viewer
3	RAG API	5001	HTTP	REST API for variant queries and RAG
4	Milvus	19530	gRPC	Vector database for genomic evidence
5	Attu	8000	HTTP	Milvus administration UI
6	Streamlit Chat	8501	HTTP	Conversational AI interface for variant analysis
7	MolMIM NIM	8001	HTTP	BioNeMo molecule generation microservice
8	DiffDock NIM	8002	HTTP	BioNeMo molecular docking microservice
9	Discovery UI	8505	HTTP	Drug discovery pipeline interface
10	Discovery Portal	8510	HTTP	Drug discovery results and reporting portal
11	Grafana	3000	HTTP	Monitoring dashboards
12	Prometheus	9099	HTTP	Metrics collection and storage
13	Node Exporter	9100	HTTP	Host system metrics
14	DCGM Exporter	9400	HTTP	NVIDIA GPU metrics

Infrastructure services (not externally exposed):

Service	Port	Purpose
etcd	2379	Milvus metadata store
MinIO	9000	Milvus object storage

2.4 Data Flow

```
┌──────────────────────────────────────────────────────────────────────────────┐
│                        HCLS AI Factory — Data Flow                           │
├──────────────────────────────────────────────────────────────────────────────┤
│                                                                              │
│  FASTQ ──► Parabricks fq2bam ──► BAM ──► Parabricks DeepVariant ──► VCF      │
│  (200 GB)   (20-45 min)         (100 GB)   (10-35 min)           (11.7M)     │
│                                                                              │
│  VCF ──► ClinVar (4.1M) ──► AlphaMissense (71.7M) ──► VEP ──► Annotated      │
│          (35,616 match)     (6,831 matched)                                  │
│                                                                              │
│  Annotated ──► BGE-small-en-v1.5 ──► Milvus (384-dim, IVF_FLAT) ──►          │
│                                      (COSINE, nlist=1024)                    │
│                                                                              │
│  Milvus ──► Claude (sonnet-4) ──► Target Hypothesis                          │
│             (temp=0.3, 4096 tokens)                                          │
│                                                                              │
│  Target ──► PDB Structures ──► MolMIM (8001) ──► Chemistry QC ──►            │
│                                                   (Lipinski + QED)           │
│                                                                              │
│  Conformers ──► DiffDock (8002) ──► Composite Ranking ──► PDF Report         │
│                                     (0.3*gen + 0.4*dock + 0.3*QED)           │
│                                                                              │
└──────────────────────────────────────────────────────────────────────────────┘
```

3. Prerequisites

3.1 Hardware Requirements

Component	Specification
System	NVIDIA DGX Spark
GPU	GB10 Grace Blackwell Superchip
Memory	128 GB unified LPDDR5x
CPU	144 ARM64 cores
Architecture	aarch64 (ARM64)
Price	$3,999

Storage requirements:

Dataset / Component	Size
GRCh38 Reference Genome	3.1 GB

FASTQ Input (HG002 30x WGS)	~200 GB
BAM Output (intermediate)	~100 GB
ClinVar Database	~1.2 GB
AlphaMissense Predictions	~4 GB
Milvus Index Data	~2 GB
BioNeMo Model Cache	~10 GB
Total Minimum	**~320 GB**
Recommended	**1 TB NVMe**

3.2 Software Requirements

Software	Minimum Version	Notes
DGX OS	Latest	Ubuntu-based ARM64
Docker Engine	24.0+	With Compose V2
NVIDIA Container Toolkit	Latest	nvidia-docker runtime
CUDA Toolkit	12.x	Included with DGX OS
Python	3.10+	For pipeline scripts
Nextflow	23.04+	DSL2 support required
Git	2.30+	For repository clone
NGC CLI	Latest	For BioNeMo container pulls

3.3 Network Requirements

- Internet access for initial setup (container pulls, data downloads)
- Outbound HTTPS to `api.anthropic.com` for Claude API calls
- Outbound HTTPS to `nvcr.io` for NGC container registry
- Outbound HTTPS to NCBI, RCSB PDB for reference data downloads
- All service ports (listed in Section 2.3) accessible on localhost

3.4 Access Credentials

Credential	Purpose	How to Obtain
`ANTHROPIC_API_KEY`	Claude API access	https://console.anthropic.com
`NGC_API_KEY`	NVIDIA NGC container registry	https://ngc.nvidia.com

4. Environment Preparation

4.1 DGX Spark Initial Setup

Verify the system is a DGX Spark with the expected hardware:

BASH

```
# Verify ARM64 architecture
uname -m
# Expected: aarch64

# Verify CPU cores
nproc
# Expected: 144

# Verify total memory (128 GB)
free -h | grep Mem
# Expected: ~128 GB total

# Verify GPU is detected
nvidia-smi
# Expected: GB10 GPU listed with driver version
```

4.2 NVIDIA Driver and CUDA Verification

BASH

```bash
# Check NVIDIA driver version
nvidia-smi --query-gpu=driver_version --format=csv,noheader
# Expected: 550.x or later

# Check CUDA version
nvcc --version
# Expected: CUDA 12.x

# Verify GPU compute capability
nvidia-smi --query-gpu=compute_cap --format=csv,noheader

# Run a quick GPU test
nvidia-smi -q | head -30
```

4.3 Docker Installation and Configuration

BASH

```bash
# Verify Docker is installed
docker --version
# Expected: Docker version 24.0+

# Verify Docker Compose V2
docker compose version
# Expected: Docker Compose version v2.x

# Verify NVIDIA runtime is available
docker info | grep -i runtime
# Expected: nvidia runtime listed

# Test GPU access from a container
docker run --rm --gpus all nvidia/cuda:12.4.0-base-ubuntu22.04 nvidia-smi
```

Configure Docker daemon for NVIDIA runtime as default:

BASH

```
sudo tee /etc/docker/daemon.json <<'EOF'
{
  "default-runtime": "nvidia",
  "runtimes": {
    "nvidia": {
      "path": "nvidia-container-runtime",
      "runtimeArgs": []
    }
  },
  "default-address-pools": [
    {"base": "172.20.0.0/16", "size": 24}
  ],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "50m",
    "max-file": "3"
  }
}
EOF

sudo systemctl restart docker
```

4.4 Python Environment Setup

BASH

```
# Verify Python version
python3 --version
# Expected: Python 3.10+

# Create virtual environment
python3 -m venv ~/hcls-env
source ~/hcls-env/bin/activate

# Install core dependencies
pip install --upgrade pip
pip install \
  anthropic \
  pymilvus \
  sentence-transformers \
  rdkit-pypi \
  pydantic \
  streamlit \
  fastapi \
  uvicorn \
  requests \
  pandas \
  numpy \
  reportlab \
  biopython \
  nextflow
```

4.5 NGC CLI Installation

BASH

```
# Download NGC CLI for ARM64
```

```
wget -O ngc-cli.zip https://api.ngc.nvidia.com/v2/resources/nvidia/ngc-
apps/ngc_cli/versions/latest/files/ngccli_arm64.zip

# Extract and install
unzip ngc-cli.zip -d ~/ngc-cli
chmod +x ~/ngc-cli/ngc-cli/ngc
export PATH=$PATH:~/ngc-cli/ngc-cli

# Configure NGC CLI
ngc config set
# Enter your NGC API key when prompted

# Verify authentication
ngc registry image list --format_type csv | head -5
```

5. Repository Setup

5.1 Fork and Clone

BASH

```
# Fork the repository on GitHub, then clone your fork
git clone https://github.com/<your-username>/hcls-ai-factory.git
cd hcls-ai-factory

# Verify repository structure
ls -la
```

5.2 Repository Layout

```
hcls-ai-factory/
├── docker-compose.yml                # All 14 services + infrastructure
├── .env.example                      # Template environment configuration
├── nextflow.config                   # Nextflow pipeline configuration
├── main.nf                           # Nextflow DSL2 pipeline definition
├── start-services.sh                 # Service startup script
├── requirements.txt                  # Python dependencies
│
├── genomics/                         # Stage 1: Genomics Pipeline
│   ├── parabricks/                   # Parabricks configs and scripts
│   │   ├── fq2bam.sh                # BWA-MEM2 alignment wrapper
│   │   └── deepvariant.sh           # DeepVariant variant calling wrapper
│   ├── portal/                       # Genomics Portal (Port 5000)
│   │   └── app.py
│   └── data/                         # Input/output data directory
│       ├── reference/                # GRCh38 reference genome
│       ├── fastq/                    # Input FASTQ files
│       ├── bam/                      # Alignment output
│       └── vcf/                      # Variant call output
│
├── rag/                              # Stage 2: RAG Chat Pipeline
│   ├── api/                          # RAG API (Port 5001)
│   │   └── app.py
│   ├── chat/                      # Streamlit Chat (Port 8501)
│   │   └── app.py
│   ├── embeddings/                # BGE embedding pipeline
│   │   └── embed_variants.py
│   ├── annotation/                # Variant annotation pipeline
│   │   ├── clinvar.py
│   │   ├── alphamissense.py
│   │   └── vep.py
│   ├── knowledge/                 # Gene knowledge base
│   │   └── genes.json             # 201 genes, 13 therapeutic areas
│   └── data/
│       ├── clinvar/               # ClinVar database
│       └── alphamissense/         # AlphaMissense predictions
│
├── discovery/                     # Stage 3: Drug Discovery Pipeline
│   ├── pipeline/                  # 10-stage discovery pipeline
│   │   ├── __init__.py
│   │   ├── initialize.py          # Stage 1: Initialize
│   │   ├── normalize.py           # Stage 2: Normalize Target
│   │   ├── structure_discovery.py  # Stage 3: Structure Discovery
│   │   ├── structure_prep.py      # Stage 4: Structure Preparation
│   │   ├── molecule_gen.py        # Stage 5: Molecule Generation
│   │   ├── chemistry_qc.py       # Stage 6: Chemistry QC
│   │   ├── conformer_gen.py       # Stage 7: Conformer Generation
│   │   ├── docking.py             # Stage 8: Molecular Docking
│   │   ├── ranking.py             # Stage 9: Composite Ranking
│   │   └── reporting.py           # Stage 10: Reporting
│   ├── ui/                        # Discovery UI (Port 8505)
│   │   └── app.py
│   ├── portal/                    # Discovery Portal (Port 8510)
│   │   └── app.py
│   └── models/                    # Pydantic data models
│       └── schemas.py
│
├── monitoring/                    # Monitoring stack
│   ├── grafana/
│   │   ├── provisioning/
│   │   └── dashboards/
│   ├── prometheus/
│   │   └── prometheus.yml
│   └── exporters/
│
├── landing/                       # Landing Page (Port 8080)
│   └── index.html
│
├── scripts/                       # Utility scripts
│   ├── run_pipeline.py            # Pipeline launcher
│   ├── download_references.sh     # Reference data downloader
│   └── validate_deployment.sh     # Deployment validator
│
└── docs/                          # Documentation
    └── ...
```

5.3 Environment Configuration

BASH

```
# Copy the example environment file
cp .env.example .env
```

```
# Edit with your credentials and paths
nano .env
```

The `.env` file should contain:

BASH

```bash
# === API Keys ===
ANTHROPIC_API_KEY=sk-ant-api03-XXXXXXXXXXXX
NGC_API_KEY=XXXXXXXXXXXX

# === Model Configuration ===
CLAUDE_MODEL=claude-sonnet-4-20250514
CLAUDE_TEMPERATURE=0.3

# === Reference Data ===
REFERENCE_GENOME=/data/reference/GRCh38.fa

# === Milvus Configuration ===
MILVUS_HOST=localhost
MILVUS_PORT=19530

# === BioNeMo NIM URLs ===
MOLMIM_URL=http://localhost:8001
DIFFDOCK_URL=http://localhost:8002

# === Pipeline Configuration ===
PIPELINE_MODE=full
NUM_CANDIDATES=100
MIN_QED=0.67
MIN_DOCK_SCORE=-6.0

# === Monitoring ===
GRAFANA_USER=admin
GRAFANA_PASSWORD=changeme
```

5.4 Directory Structure for Data

BASH

```bash
# Create data directories
mkdir -p genomics/data/{reference,fastq,bam,vcf}
mkdir -p rag/data/{clinvar,alphamissense}
mkdir -p discovery/data/{structures,molecules,reports}
mkdir -p monitoring/data/{grafana,prometheus}
```

6. Reference Data Preparation

6.1 GRCh38 Reference Genome

BASH

```
# Download GRCh38 reference genome (~3.1 GB)
cd genomics/data/reference

wget
https://ftp.ncbi.nlm.nih.gov/genomes/all/GCA/000/001/405/GCA_000001405.15_GRCh38/seqs_for_alignment_pipelines
.ucsc_ids/GCA_000001405.15_GRCh38_no_alt_analysis_set.fna.gz

# Decompress
gunzip GCA_000001405.15_GRCh38_no_alt_analysis_set.fna.gz
mv GCA_000001405.15_GRCh38_no_alt_analysis_set.fna GRCh38.fa

# Index the reference (required by Parabricks)
# Note: Parabricks fq2bam can build its own index, but pre-building saves time
samtools faidx GRCh38.fa

# Verify
ls -lh GRCh38.fa*
# Expected: GRCh38.fa (~3.1 GB), GRCh38.fa.fai
```

6.2 ClinVar Database

BASH

```
# Download ClinVar VCF (~1.2 GB)
cd rag/data/clinvar

wget https://ftp.ncbi.nlm.nih.gov/pub/clinvar/vcf_GRCh38/clinvar.vcf.gz
wget https://ftp.ncbi.nlm.nih.gov/pub/clinvar/vcf_GRCh38/clinvar.vcf.gz.tbi

# Verify record count (~4.1M clinical variants)
zcat clinvar.vcf.gz | grep -v '^#' | wc -l
# Expected: ~4,100,000

echo "ClinVar download complete"
```

6.3 AlphaMissense Database

BASH

```
# Download AlphaMissense predictions (~4 GB)
cd rag/data/alphamissense

wget https://storage.googleapis.com/dm_alphamissense/AlphaMissense_hg38.tsv.gz

# Verify record count (~71.7M predictions)
zcat AlphaMissense_hg38.tsv.gz | tail -n +5 | wc -l
# Expected: ~71,697,560
```

```
echo "AlphaMissense download complete"
```

AlphaMissense pathogenicity thresholds:

Classification	Score Range	Description
Pathogenic	> 0.564	Likely damaging to protein function
Ambiguous	0.34 - 0.564	Uncertain significance
Benign	< 0.34	Likely tolerated

6.4 HG002 Sample Data

BASH

```bash
# Download HG002 FASTQ files for demo/testing (~200 GB)
cd genomics/data/fastq

# GIAB HG002 30x WGS, 2x250 bp paired-end
# Note: These are large files — ensure ~200 GB free space
wget ftp://ftp-
trace.ncbi.nlm.nih.gov/ReferenceSamples/giab/data/AshkenazimTrio/HG002_NA24385_son/NIST_HiSeq_HG002_Homogenei
ty-10953946/NHGRI_Illumina300X_AJtrio_novoalign_bams/HG002.GRCh38.2x250.fastq.gz

# For a smaller test subset, use a downsampled version if available
echo "HG002 download complete — verify file sizes match expected ~200 GB"
ls -lh *.fastq.gz
```

7. Docker Compose Configuration

7.1 Service Definition Overview

The `docker-compose.yml` defines all 14 application services plus 2 infrastructure services (etcd, MinIO) for Milvus. Services are organized into three groups matching the pipeline stages, plus monitoring.

7.2 docker-compose.yml Structure

YAML

```yaml
version: '3.8'

services:
  # ─── Infrastructure ───────────────────────────────────────
  etcd:
    image: quay.io/coreos/etcd:v3.5.5
    environment:
      - ETCD_AUTO_COMPACTION_MODE=revision
      - ETCD_AUTO_COMPACTION_RETENTION=1000
    ports:
      - "2379:2379"
    volumes:
      - etcd_data:/etcd
    restart: unless-stopped

  minio:
    image: minio/minio:latest
    environment:
      MINIO_ACCESS_KEY: minioadmin
      MINIO_SECRET_KEY: minioadmin
    ports:
      - "9000:9000"
    volumes:
      - minio_data:/data
    command: server /data
    restart: unless-stopped

  # ─── Milvus Vector Database ──────────────────────────────
  milvus:
    image: milvusdb/milvus:v2.4-latest
    ports:
      - "19530:19530"
    environment:
      ETCD_ENDPOINTS: etcd:2379
      MINIO_ADDRESS: minio:9000
    depends_on:
      - etcd
      - minio
    volumes:
      - milvus_data:/var/lib/milvus
    restart: unless-stopped

  attu:
    image: zilliz/attu:latest
    ports:
      - "8000:3000"
    environment:
      MILVUS_URL: milvus:19530
    depends_on:
      - milvus
    restart: unless-stopped

  # ─── Stage 1: Genomics ──────────────────────────────────
  genomics-portal:
    build: ./genomics/portal
    ports:
      - "5000:5000"
    volumes:
      - ./genomics/data:/data
    environment:
      - REFERENCE_GENOME=/data/reference/GRCh38.fa
    restart: unless-stopped

  # ─── Stage 2: RAG Chat ──────────────────────────────────
  rag-api:
    build: ./rag/api
    ports:
      - "5001:5001"
    environment:
      - ANTHROPIC_API_KEY=${ANTHROPIC_API_KEY}
      - CLAUDE_MODEL=${CLAUDE_MODEL}
      - CLAUDE_TEMPERATURE=${CLAUDE_TEMPERATURE}
      - MILVUS_HOST=milvus
      - MILVUS_PORT=19530
    depends_on:
... (121 more lines)
```

7.3 Infrastructure Services

Milvus 2.4 requires two backend services:

Service	Image	Port	Purpose
etcd	quay.io/coreos/etcd:v3.5.5	2379	Metadata storage for Milvus
MinIO	minio/minio:latest	9000	Object storage for Milvus segments
Milvus	milvusdb/milvus:v2.4-latest	19530	Vector database

7.4 Volume Mounts and Data Paths

Volume	Container Path	Host Purpose
`./genomics/data`	`/data`	Reference genome, FASTQ, BAM, VCF
`./rag/data`	`/data`	ClinVar, AlphaMissense databases
`etcd_data`	`/etcd`	Milvus metadata persistence
`minio_data`	`/data`	Milvus segment persistence
`milvus_data`	`/var/lib/milvus`	Milvus index persistence
`prometheus_data`	`/prometheus`	Prometheus TSDB
`grafana_data`	`/var/lib/grafana`	Grafana state and dashboards

7.5 GPU Resource Allocation

The GB10 GPU is shared across GPU-consuming services. Only one GPU-heavy workload should run at a time:

Service	GPU Usage	Peak Memory	Typical Duration
Parabricks fq2bam	70-90% GPU	~40 GB	20-45 min
Parabricks DeepVariant	80-95% GPU	~60 GB	10-35 min
MolMIM NIM	Moderate	~8 GB	Always running
DiffDock NIM	Moderate	~8 GB	Always running
DCGM Exporter	Minimal	Minimal	Always running

8. Deploy Genomics Pipeline (Stage 1)

8.1 Parabricks Container Setup

BASH

```bash
# Pull Parabricks container for ARM64
docker pull nvcr.io/nvidia/clara/clara-parabricks:4.6.0-1

# Verify the image
docker images | grep parabricks
# Expected: clara-parabricks   4.6.0-1
```

8.2 BWA-MEM2 Alignment (fq2bam)

The `fq2bam` tool performs GPU-accelerated read alignment using BWA-MEM2 and produces a sorted, duplicate-marked BAM file.

BASH

```bash
docker run --rm --gpus all \
  -v $(pwd)/genomics/data:/data \
  nvcr.io/nvidia/clara/clara-parabricks:4.6.0-1 \
  pbrun fq2bam \
    --ref /data/reference/GRCh38.fa \
    --in-fq /data/fastq/HG002_R1.fastq.gz /data/fastq/HG002_R2.fastq.gz \
    --out-bam /data/bam/HG002.bam \
    --num-gpus 1
```

Expected performance:

Metric	Value
Runtime	20-45 minutes
GPU Utilization	70-90%
Peak GPU Memory	~40 GB
Output	Sorted, duplicate-marked BAM (~100 GB)

8.3 DeepVariant Variant Calling

BASH

```bash
docker run --rm --gpus all \
  -v $(pwd)/genomics/data:/data \
  nvcr.io/nvidia/clara/clara-parabricks:4.6.0-1 \
  pbrun deepvariant \
    --ref /data/reference/GRCh38.fa \
    --in-bam /data/bam/HG002.bam \
    --out-variants /data/vcf/HG002.vcf.gz \
    --num-gpus 1
```

Expected performance:

Metric	Value
Runtime	10-35 minutes
GPU Utilization	80-95%
Peak GPU Memory	~60 GB
Output	Compressed VCF (gzipped)

8.4 VCF Output Verification

BASH

```bash
# Count total variants
zcat genomics/data/vcf/HG002.vcf.gz | grep -v '^#' | wc -l
# Expected: ~11,700,000 (11.7M variants)

# Count PASS variants with QUAL > 30
zcat genomics/data/vcf/HG002.vcf.gz | grep -v '^#' | \
  awk '$7 == "PASS" && $6 > 30' | wc -l
# Expected: ~3,500,000 (3.5M)

# Count SNPs vs Indels
zcat genomics/data/vcf/HG002.vcf.gz | grep -v '^#' | \
  awk '{if(length($4)==1 && length($5)==1) print "SNP"; else print "INDEL"}' | \
  sort | uniq -c
# Expected: ~4,200,000 SNPs, ~1,000,000 indels
```

VCF output summary:

Metric	Expected Value
Total variants	~11.7M
PASS variants (QUAL > 30)	~3.5M
SNPs	~4.2M
Indels	~1.0M
Coding region variants	~35,000

8.5 Genomics Portal (Port 5000)

After genomics processing, start the portal:

BASH

```bash
docker compose up -d genomics-portal

# Verify
curl -s http://localhost:5000/health
# Expected: {"status": "healthy"}
```

Access the Genomics Portal at `http://<dgx-spark-ip>:5000` to browse VCF results.

8.6 Performance Benchmarks

Step	Wall Time	GPU Util	Peak Memory	Output Size
fq2bam (alignment)	20-45 min	70-90%	~40 GB	~100 GB BAM
DeepVariant (calling)	10-35 min	80-95%	~60 GB	~1 GB VCF.gz
Total Stage 1	**30-80 min**	—	—	—

9. Deploy RAG Chat Pipeline (Stage 2)

9.1 Milvus Vector Database Setup

BASH

```bash
# Start Milvus and its dependencies
docker compose up -d etcd minio milvus attu

# Wait for Milvus to be ready (30-60 seconds)
sleep 30

# Verify Milvus is running
curl -s http://localhost:19530/v1/health/ready
# Expected: {"status":"ok"}

# Verify Attu UI
curl -s -o /dev/null -w "%{http_code}" http://localhost:8000
# Expected: 200
```

9.2 Collection Schema

Create the `genomic_evidence` collection with 17 fields:

PYTHON

```python
from pymilvus import connections, Collection, FieldSchema, CollectionSchema, DataType, utility

# Connect to Milvus
connections.connect(host="localhost", port=19530)

# Define schema with 17 fields
fields = [
    FieldSchema(name="id", dtype=DataType.INT64, is_primary=True, auto_id=True),
    FieldSchema(name="embedding", dtype=DataType.FLOAT_VECTOR, dim=384),
    FieldSchema(name="chrom", dtype=DataType.VARCHAR, max_length=10),
    FieldSchema(name="pos", dtype=DataType.INT64),
    FieldSchema(name="ref", dtype=DataType.VARCHAR, max_length=500),
    FieldSchema(name="alt", dtype=DataType.VARCHAR, max_length=500),
    FieldSchema(name="qual", dtype=DataType.FLOAT),
    FieldSchema(name="gene", dtype=DataType.VARCHAR, max_length=100),
    FieldSchema(name="consequence", dtype=DataType.VARCHAR, max_length=200),
    FieldSchema(name="impact", dtype=DataType.VARCHAR, max_length=20),
    FieldSchema(name="genotype", dtype=DataType.VARCHAR, max_length=10),
    FieldSchema(name="text_summary", dtype=DataType.VARCHAR, max_length=5000),
    FieldSchema(name="clinical_significance", dtype=DataType.VARCHAR, max_length=200),
    FieldSchema(name="rsid", dtype=DataType.VARCHAR, max_length=20),
    FieldSchema(name="disease_associations", dtype=DataType.VARCHAR, max_length=2000),
    FieldSchema(name="am_pathogenicity", dtype=DataType.FLOAT),
    FieldSchema(name="am_class", dtype=DataType.VARCHAR, max_length=20),
]

schema = CollectionSchema(fields, description="Genomic evidence for RAG")
collection = Collection("genomic_evidence", schema)

# Create IVF_FLAT index on embedding field
index_params = {
    "metric_type": "COSINE",
    "index_type": "IVF_FLAT",
    "params": {"nlist": 1024}
}
collection.create_index("embedding", index_params)

# Load collection into memory
collection.load()

print(f"Collection created: {collection.name}")
print(f"Schema fields: {len(fields)}")
```

Collection schema reference:

#	Field	Type	Details
1	id	INT64	Primary key, auto-generated
2	embedding	FLOAT_VECTOR	384 dimensions (BGE-small-en-v1.5)
3	chrom	VARCHAR(10)	Chromosome (chr1-22, chrX, chrY)
4	pos	INT64	Genomic position
5	ref	VARCHAR(500)	Reference allele
6	alt	VARCHAR(500)	Alternate allele
7	qual	FLOAT	Variant quality score
8	gene	VARCHAR(100)	Gene symbol
9	consequence	VARCHAR(200)	VEP functional consequence
10	impact	VARCHAR(20)	HIGH, MODERATE, LOW, MODIFIER
11	genotype	VARCHAR(10)	Sample genotype (e.g., 0/1, 1/1)
12	text_summary	VARCHAR(5000)	Natural-language variant summary
13	clinical_significance	VARCHAR(200)	ClinVar classification
14	rsid	VARCHAR(20)	dbSNP identifier
15	disease_associations	VARCHAR(2000)	Associated diseases/conditions
16	am_pathogenicity	FLOAT	AlphaMissense score (0.0-1.0)
17	am_class	VARCHAR(20)	pathogenic, ambiguous, or

benign

9.3 Variant Annotation Pipeline

The annotation pipeline enriches VCF variants with data from three sources:

BASH

```bash
# Run the annotation pipeline
python3 rag/annotation/clinvar.py \
  --vcf genomics/data/vcf/HG002.vcf.gz \
  --clinvar rag/data/clinvar/clinvar.vcf.gz \
  --output rag/data/annotated_clinvar.tsv

python3 rag/annotation/alphamissense.py \
  --vcf genomics/data/vcf/HG002.vcf.gz \
  --am rag/data/alphamissense/AlphaMissense_hg38.tsv.gz \
  --output rag/data/annotated_am.tsv

python3 rag/annotation/vep.py \
  --vcf genomics/data/vcf/HG002.vcf.gz \
  --output rag/data/annotated_vep.tsv
```

Expected annotation matches:

Source	Total Records	Patient Matches
ClinVar	4,100,000	~35,616
AlphaMissense	71,697,560	~6,831 (ClinVar-matched with predictions)
VEP	Per-variant	All coding variants

9.4 BGE Embedding and Indexing

PYTHON

```python
from sentence_transformers import SentenceTransformer
from pymilvus import connections, Collection

# Load embedding model
model = SentenceTransformer('BAAI/bge-small-en-v1.5')  # 384 dimensions

# Connect to Milvus
connections.connect(host="localhost", port=19530)
collection = Collection("genomic_evidence")

# Example: embed and insert a variant
text = "chr9:35065263 G>A in VCP gene. ClinVar: Pathogenic. AlphaMissense: 0.87 (pathogenic). Consequence:
missense_variant. Impact: MODERATE."
embedding = model.encode(text).tolist()  # 384-dim vector

# Insert into Milvus
data = [{
    "embedding": embedding,
    "chrom": "chr9",
    "pos": 35065263,
    "ref": "G",
    "alt": "A",
    "qual": 99.0,
    "gene": "VCP",
    "consequence": "missense_variant",
    "impact": "MODERATE",
    "genotype": "0/1",
    "text_summary": text,
    "clinical_significance": "Pathogenic",
    "rsid": "rs188935092",
    "disease_associations": "Inclusion body myopathy with Paget disease and frontotemporal dementia",
    "am_pathogenicity": 0.87,
    "am_class": "pathogenic"
}]

collection.insert(data)
collection.flush()
```

Milvus index configuration:

Parameter	Value
Embedding Model	BGE-small-en-v1.5
Dimensions	384
Index Type	IVF_FLAT
Metric Type	COSINE
nlist	1024
nprobe (search)	16

9.5 Anthropic Claude Integration

PYTHON

```python
import anthropic

client = anthropic.Anthropic(api_key=os.environ["ANTHROPIC_API_KEY"])

def query_claude(question: str, context: str) -> str:
    """Send RAG query to Claude with retrieved genomic context."""
    response = client.messages.create(
        model="claude-sonnet-4-20250514",
        max_tokens=4096,
        temperature=0.3,
        messages=[{
            "role": "user",
            "content": f"""You are a genomics expert. Answer the question using the provided genomic
evidence.

Context:
{context}

Question: {question}"""
        }]
    )
    return response.content[0].text
```

Claude configuration:

Parameter	Value
Model	claude-sonnet-4-20250514
Temperature	0.3
Max Tokens	4096

9.6 Knowledge Base

The platform includes a curated knowledge base of 201 genes across 13 therapeutic areas, with 171 genes (85%) classified as druggable.

Metric	Value
Total genes	201
Therapeutic areas	13
Druggable genes	171 (85%)

9.7 RAG API and Streamlit Chat

BASH

```bash
# Start RAG API and Chat services
docker compose up -d rag-api streamlit-chat

# Verify RAG API
curl -s http://localhost:5001/health
# Expected: {"status": "healthy"}

# Verify Streamlit Chat
curl -s -o /dev/null -w "%{http_code}" http://localhost:8501
# Expected: 200
```

Access the Streamlit Chat at `http://<dgx-spark-ip>:8501` for conversational variant analysis.

10. Deploy Drug Discovery Pipeline (Stage 3)

10.1 BioNeMo NIM Services

BASH

```bash
# Pull BioNeMo containers (requires NGC authentication)
docker pull nvcr.io/nvidia/clara/bionemo-molmim:1.0
docker pull nvcr.io/nvidia/clara/diffdock:1.0

# Start NIM services
docker compose up -d molmim diffdock

# Wait for models to load (may take 2-5 minutes)
sleep 120
```

```
# Verify MolMIM
curl -s http://localhost:8001/v1/health/ready
# Expected: {"status": "ready"}

# Verify DiffDock
curl -s http://localhost:8002/v1/health/ready
# Expected: {"status": "ready"}
```

10.2 10-Stage Pipeline Detail

Stage	Name	Input	Output	Key Operations
1	Initialize	Config + target gene	PipelineConfig	Validate parameters, create run ID
2	Normalize Target	Gene symbol	Normalized target	Map to UniProt, canonical name
3	Structure Discovery	UniProt ID	PDB structure list	Query RCSB PDB, score by resolution
4	Structure Preparation	PDB IDs	Prepared structures	Download PDB, extract binding sites
5	Molecule Generation	Seed SMILES + protein	Generated SMILES	MolMIM NIM (Port 8001)
6	Chemistry QC	SMILES list	Filtered SMILES	Lipinski, QED, TPSA checks
7	Conformer Generation	Filtered SMILES	3D conformers (SDF)	RDKit conformer embedding
8	Molecular Docking	Conformers + protein	Docking scores	DiffDock NIM (Port 8002)
9	Composite Ranking	All scores	Ranked candidates	Weighted composite formula
10	Reporting	Ranked candidates	PDF report	Visualizations, recommendations

10.3 Structure Retrieval and Scoring

PYTHON

```python
import requests

def search_pdb_structures(uniprot_id: str) -> list:
    """Search RCSB PDB for protein structures by UniProt ID."""
    url = "https://search.rcsb.org/rcsbsearch/v2/query"
    query = {
        "query": {
            "type": "terminal",
            "service": "text",
            "parameters": {
                "attribute":
"rcsb_polymer_entity_container_identifiers.reference_sequence_identifiers.database_accession",
                "operator": "exact_match",
                "value": uniprot_id
            }
        },
        "return_type": "entry"
    }
    response = requests.post(url, json=query)
    return response.json().get("result_set", [])
```

10.4 Molecule Generation (MolMIM)

PYTHON

```python
import requests


def generate_molecules(seed_smiles: str, num_candidates: int = 100) -> list:
    """Generate molecule candidates using MolMIM NIM."""
    response = requests.post(
        "http://localhost:8001/generate",
        json={
            "smiles": seed_smiles,
            "num_molecules": num_candidates,
            "algorithm": "CMA-ES",
            "property_name": "QED",
            "min_similarity": 0.3,
            "particles": 30,
            "iterations": 10
        }
    )
    return response.json()["generated_molecules"]
```

10.5 Molecular Docking (DiffDock)

PYTHON

```python
def dock_molecule(protein_pdb: str, ligand_sdf: str) -> dict:
    """Score binding affinity using DiffDock NIM."""
    response = requests.post(
        "http://localhost:8002/molecular-docking/diffdock/generate",
        json={
            "protein": protein_pdb,
            "ligand": ligand_sdf,
            "num_poses": 10
        }
    )
    return response.json()
```

10.6 Drug-Likeness Scoring

Drug-likeness is assessed using three criteria:

Lipinski Rule of Five:

Property	Threshold	Description
Molecular Weight	<= 500 Da	Size constraint
LogP	<= 5	Lipophilicity
H-Bond Donors (HBD)	<= 5	Polar surface groups

H-Bond Acceptors (HBA)	<= 10	Polar surface groups

Additional thresholds:

Metric	Threshold	Interpretation
QED	> 0.67	Drug-like
TPSA	< 140 Angstrom squared	Good oral bioavailability

PYTHON

```python
from rdkit import Chem
from rdkit.Chem import Descriptors, QED

def assess_drug_likeness(smiles: str) -> dict:
    """Evaluate drug-likeness using Lipinski, QED, and TPSA."""
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        return {"valid": False}

    mw = Descriptors.MolWt(mol)
    logp = Descriptors.MolLogP(mol)
    hbd = Descriptors.NumHDonors(mol)
    hba = Descriptors.NumHAcceptors(mol)
    tpsa = Descriptors.TPSA(mol)
    qed_score = QED.qed(mol)

    lipinski_pass = (mw <= 500 and logp <= 5 and hbd <= 5 and hba <= 10)

    return {
        "valid": True,
        "mw": mw,
        "logp": logp,
        "hbd": hbd,
        "hba": hba,
        "tpsa": tpsa,
        "qed": qed_score,
        "lipinski_pass": lipinski_pass,
        "drug_like": qed_score > 0.67,
        "oral_bioavail": tpsa < 140
    }
```

10.7 Composite Ranking Formula

Candidates are ranked using a weighted composite score:

```
composite = 0.30 * generation_score + 0.40 * docking_score_normalized + 0.30 * qed_score
```

Docking score normalization:

PYTHON

```python
def normalize_docking_score(dock_score: float) -> float:
    """Normalize docking score to [0, 1] range.
    More negative = better binding = higher normalized score."""
    return max(0.0, min(1.0, (10.0 + dock_score) / 20.0))
```

Raw Docking Score	Normalized Score	Interpretation
-10.0 kcal/mol	0.00	Excellent binding
-8.0 kcal/mol	0.10	Strong binding
-6.0 kcal/mol	0.20	Moderate binding
0.0 kcal/mol	0.50	Weak binding
+10.0 kcal/mol	1.00	No binding

> **Note:** The normalization maps more negative (better) docking scores to lower normalized values. In the composite formula, the docking component rewards lower (better) scores.

Composite score weights:

Component	Weight	Source
Generation Score	30%	MolMIM similarity/property score
Docking Score (normalized)	40%	DiffDock binding affinity
QED Score	30%	RDKit quantitative drug-likeness

10.8 Discovery UI and Portal

BASH

```bash
# Start Discovery services
docker compose up -d discovery-ui discovery-portal

# Verify Discovery UI
curl -s -o /dev/null -w "%{http_code}" http://localhost:8505
# Expected: 200

# Verify Discovery Portal
curl -s -o /dev/null -w "%{http_code}" http://localhost:8510
# Expected: 200
```

- **Discovery UI (Port 8505):** Interactive pipeline execution interface
- **Discovery Portal (Port 8510):** Results browser and reporting portal

10.9 PDF Report Generation

The final pipeline stage generates a PDF report containing:

- Target gene and variant summary
- PDB structure details with binding site analysis
- Top-ranked candidates with SMILES, scores, and 2D depictions
- Docking poses and binding affinity plots
- Lipinski and QED compliance table
- Composite score ranking

11. Nextflow Orchestration

11.1 DSL2 Pipeline Architecture

The HCLS AI Factory uses Nextflow DSL2 for pipeline orchestration. Each pipeline stage is defined as a separate process, with channels connecting inputs and outputs.

11.2 Pipeline Modes

Mode	Description	Stages Executed
`full`	Complete end-to-end pipeline	1 + 2 + 3 (all stages)
`target`	Start from target gene (skip genomics)	2 + 3
`drug`	Drug discovery only (pre-existing target)	3 only
`demo`	VCP demo with pre-loaded data	1 + 2 + 3 (demo subset)
`genomics_only`	Genomics pipeline only	1 only

11.3 Execution Profiles

Profile	Description	Use Case
`standard`	Local execution, default settings	Development
`docker`	Docker container execution	Standard deployment
`singularity`	Singularity container execution	HPC environments
`dgx_spark`	Optimized for DGX Spark hardware	Production on DGX Spark
`slurm`	SLURM workload manager	Multi-node clusters
`test`	Minimal test data, fast execution	CI/CD testing

11.4 Pipeline Launcher

BASH

```bash
# Run with the pipeline launcher script
python3 scripts/run_pipeline.py \
  --mode full \
  --profile dgx_spark \
  --fastq genomics/data/fastq/ \
  --reference genomics/data/reference/GRCh38.fa

# Or run directly with Nextflow
nextflow run main.nf \
  -profile dgx_spark \
  --mode full \
  --fastq_dir genomics/data/fastq/ \
  --reference genomics/data/reference/GRCh38.fa \
  --outdir results/
```

11.5 Pipeline Configuration

GROOVY

```groovy
// nextflow.config
params {
    // Pipeline mode
    mode = 'full'

    // Input paths
    fastq_dir = 'genomics/data/fastq'
    reference = 'genomics/data/reference/GRCh38.fa'
    outdir = 'results'

    // Service endpoints
    milvus_host = 'localhost'
    milvus_port = 19530
    molmim_url = 'http://localhost:8001'
    diffdock_url = 'http://localhost:8002'

    // Drug discovery parameters
    num_candidates = 100
    min_qed = 0.67
    min_dock_score = -6.0
}

profiles {
    dgx_spark {
        docker.enabled = true
        docker.runOptions = '--gpus all'
        process {
            executor = 'local'
            memory = '120 GB'
            cpus = 128
        }
    }

    test {
        params.mode = 'demo'
        process {
            memory = '16 GB'
            cpus = 4
        }
    }
}
```

12. Service Startup and Health

12.1 start-services.sh Startup Order

Services should be started in dependency order:

BASH

```bash
#!/bin/bash
# start-services.sh — Start all HCLS AI Factory services
```

```
set -e

echo "Starting infrastructure services..."
docker compose up -d etcd minio
sleep 10

echo "Starting Milvus..."
docker compose up -d milvus attu
sleep 30

echo "Starting BioNeMo NIM services..."
docker compose up -d molmim diffdock
sleep 120

echo "Starting application services..."
docker compose up -d genomics-portal rag-api streamlit-chat discovery-ui discovery-portal landing-page

echo "Starting monitoring..."
docker compose up -d prometheus grafana node-exporter dcgm-exporter

echo "All services started. Running health checks..."
sleep 10
bash scripts/validate_deployment.sh
```

12.2 Landing Page (Port 8080)

The landing page at `http://<dgx-spark-ip>:8080` provides a directory of all services with links and status indicators.

12.3 Health Check Endpoints

Service	Port	Health Endpoint	Expected Response
Genomics Portal	5000	`/health`	`{"status": "healthy"}`
RAG API	5001	`/health`	`{"status": "healthy"}`
Milvus	19530	`/v1/health/ready`	`{"status": "ok"}`
Attu	8000	`/api/health`	HTTP 200
Streamlit Chat	8501	`/healthz`	HTTP 200
MolMIM NIM	8001	`/v1/health/ready`	`{"status": "ready"}`
DiffDock NIM	8002	`/v1/health/ready`	`{"status": "ready"}`
Discovery UI	8505	`/health`	`{"status": "healthy"}`
Discovery Portal	8510	`/health`	`{"status": "healthy"}`
Grafana	3000	`/api/health`	`{"status": "ok"}`
Prometheus	9099	`/-/healthy`	HTTP 200
Node Exporter	9100	`/metrics`	Metrics text
DCGM Exporter	9400	`/metrics`	Metrics text

12.4 Verifying All Services

BASH

```bash
#!/bin/bash
# validate_deployment.sh — Verify all services are running

declare -A SERVICES=(
  ["Landing Page"]="http://localhost:8080"
  ["Genomics Portal"]="http://localhost:5000/health"
  ["RAG API"]="http://localhost:5001/health"
  ["Milvus"]="http://localhost:19530/v1/health/ready"
  ["Attu"]="http://localhost:8000"
  ["Streamlit Chat"]="http://localhost:8501/healthz"
  ["MolMIM"]="http://localhost:8001/v1/health/ready"
  ["DiffDock"]="http://localhost:8002/v1/health/ready"
  ["Discovery UI"]="http://localhost:8505/health"
  ["Discovery Portal"]="http://localhost:8510/health"
  ["Grafana"]="http://localhost:3000/api/health"
  ["Prometheus"]="http://localhost:9099/-/healthy"
  ["Node Exporter"]="http://localhost:9100/metrics"
  ["DCGM Exporter"]="http://localhost:9400/metrics"
)

echo "=== HCLS AI Factory Health Check ==="
for service in "${!SERVICES[@]}"; do
  url="${SERVICES[$service]}"
  status=$(curl -s -o /dev/null -w "%{http_code}" "$url" 2>/dev/null || echo "ERR")
  if [ "$status" == "200" ]; then
    echo "[OK]   $service ($url)"
  else
    echo "[FAIL] $service ($url) — HTTP $status"
  fi
done
```

13. Monitoring and Observability

13.1 Grafana Setup (Port 3000)

BASH

```bash
# Start Grafana
docker compose up -d grafana

# Access at http://<dgx-spark-ip>:3000
# Default credentials: admin / changeme
```

Default Grafana credentials:

Parameter	Value
Username	admin
Password	changeme

13.2 Prometheus Configuration (Port 9099)

YAML

```yaml
# monitoring/prometheus/prometheus.yml
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: 'node-exporter'
    static_configs:
      - targets: ['node-exporter:9100']

  - job_name: 'dcgm-exporter'
    static_configs:
      - targets: ['dcgm-exporter:9400']

  - job_name: 'rag-api'
    static_configs:
      - targets: ['rag-api:5001']
    metrics_path: /metrics

  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']
```

13.3 DCGM Exporter (Port 9400)

Key GPU metrics exposed by the DCGM Exporter:

Metric	Description
`DCGM_FI_DEV_GPU_UTIL`	GPU utilization percentage
`DCGM_FI_DEV_FB_USED`	GPU framebuffer memory used (MB)
`DCGM_FI_DEV_FB_FREE`	GPU framebuffer memory free (MB)
`DCGM_FI_DEV_GPU_TEMP`	GPU temperature (Celsius)
`DCGM_FI_DEV_POWER_USAGE`	Power consumption (Watts)
`DCGM_FI_DEV_SM_CLOCK`	Streaming multiprocessor clock (MHz)
`DCGM_FI_DEV_MEM_CLOCK`	Memory clock (MHz)

13.4 Node Exporter (Port 9100)

The Node Exporter provides host system metrics — CPU, memory, disk, and network utilization — critical for monitoring the DGX Spark ARM64 system.

13.5 Key Dashboard Panels

Recommended Grafana dashboard panels:

Panel	Data Source	Purpose
GPU Utilization	DCGM	Track fq2bam and DeepVariant GPU usage
GPU Memory	DCGM	Monitor peak memory during genomics
CPU Utilization	Node Exporter	ARM64 core usage across 144 cores
Memory Usage	Node Exporter	Unified 128 GB LPDDR5x utilization
Disk I/O	Node Exporter	NVMe throughput for FASTQ/BAM processing
Network I/O	Node Exporter	API call throughput
Container Status	Docker	Service health overview

13.6 Alert Configuration

YAML

```yaml
# Example alert rules for Prometheus
groups:
  - name: hcls-alerts
    rules:
      - alert: GPUMemoryHigh
        expr: DCGM_FI_DEV_FB_USED / (DCGM_FI_DEV_FB_USED + DCGM_FI_DEV_FB_FREE) > 0.95
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "GPU memory usage above 95%"

      - alert: ServiceDown
        expr: up == 0
        for: 2m
        labels:
          severity: critical
        annotations:
          summary: "Service {{ $labels.job }} is down"
```

14. Security Configuration

14.1 API Key Management

BASH

```bash
# Store API keys in .env file (not committed to git)
echo ".env" >> .gitignore

# Set restrictive permissions
chmod 600 .env

# Verify .env is in .gitignore
grep -q '.env' .gitignore && echo "OK: .env is gitignored"
```

Never commit API keys to version control. Use environment variables exclusively:

Variable	Sensitivity	Storage
`ANTHROPIC_API_KEY`	High	`.env` file, `chmod 600`
`NGC_API_KEY`	High	`.env` file, `chmod 600`
`GRAFANA_PASSWORD`	Medium	`.env` file

14.2 Docker Network Isolation

Docker Compose creates an isolated bridge network. Only explicitly exposed ports are accessible from the host:

BASH
```bash
# Verify network isolation
docker network ls | grep hcls
docker network inspect hcls-ai-factory_default
```

14.3 Container Security

Best practices applied to the deployment:

- Run application containers as non-root users where possible
- Use read-only filesystem mounts for reference data
- Limit container capabilities with `--cap-drop ALL`
- Pin container image versions (no `latest` tags in production)

14.4 Data Access Controls

BASH
```bash
# Set appropriate permissions on data directories
chmod -R 750 genomics/data/
chmod -R 750 rag/data/
chmod -R 750 discovery/data/

# Ensure only the deployment user can access sensitive data
chown -R $(whoami):$(whoami) genomics/data/ rag/data/ discovery/data/
```

15. Data Management

15.1 Storage Layout

Directory	Contents	Size	Persistence
`genomics/data/reference/`	GRCh38 genome	3.1 GB	Permanent
`genomics/data/fastq/`	Input FASTQ files	~200 GB	Keep until processed
`genomics/data/bam/`	Alignment output	~100 GB	Delete after VCF

`genomics/data/vcf/`	Variant calls	~1 GB	Permanent
`rag/data/clinvar/`	ClinVar database	~1.2 GB	Permanent
`rag/data/alphamissense/`	AlphaMissense DB	~4 GB	Permanent
`milvus_data` (Docker volume)	Vector index	~2 GB	Permanent
`discovery/data/`	Structures, molecules	Variable	Per-run

15.2 Intermediate File Cleanup

BAM files are the largest intermediate output (~100 GB). Once the VCF has been verified, BAM files can be deleted to reclaim storage:

BASH
```bash
# Verify VCF is complete before deleting BAM
zcat genomics/data/vcf/HG002.vcf.gz | grep -v '^#' | wc -l
# Confirm ~11.7M variants

# Delete intermediate BAM
rm -f genomics/data/bam/HG002.bam genomics/data/bam/HG002.bam.bai
echo "Reclaimed ~100 GB"
```

15.3 Milvus Data Persistence

Milvus data is stored in Docker volumes. To back up:

BASH
```bash
# Stop Milvus for consistent backup
docker compose stop milvus

# Back up volumes
docker run --rm \
  -v hcls-ai-factory_milvus_data:/data \
  -v $(pwd)/backups:/backup \
  alpine tar czf /backup/milvus_data_$(date +%Y%m%d).tar.gz /data

# Restart
docker compose start milvus
```

15.4 Backup Procedures

BASH
```bash
# Full backup script
#!/bin/bash
BACKUP_DIR=./backups/$(date +%Y%m%d)
mkdir -p $BACKUP_DIR

# Back up VCF results
cp -r genomics/data/vcf/ $BACKUP_DIR/vcf/

# Back up environment config (without secrets)
grep -v 'API_KEY' .env > $BACKUP_DIR/env_sanitized.txt
```

```
# Back up Milvus volumes
docker compose stop milvus
for vol in milvus_data etcd_data minio_data; do
  docker run --rm \
    -v hcls-ai-factory_${vol}:/data \
    -v $(pwd)/$BACKUP_DIR:/backup \
    alpine tar czf /backup/${vol}.tar.gz /data
done
docker compose start milvus

echo "Backup complete: $BACKUP_DIR"
```

16. Performance Tuning

16.1 GPU Memory Management

The DGX Spark uses 128 GB unified LPDDR5x memory shared between CPU and GPU. Key considerations:

- Parabricks DeepVariant peaks at ~60 GB GPU memory — ensure other GPU services are idle during genomics processing
- MolMIM and DiffDock each require ~8 GB — they can co-exist during drug discovery
- Monitor with `nvidia-smi` and DCGM metrics during pipeline runs

BASH

```bash
# Monitor GPU memory in real-time
watch -n 1 nvidia-smi

# Check unified memory allocation
nvidia-smi --query-gpu=memory.used,memory.free,memory.total --format=csv
```

16.2 Milvus Index Tuning

Parameter	Default	Tuning Guidance
`nlist`	1024	Increase for larger collections (trade build time for search quality)
`nprobe`	16	Increase for higher recall (trade latency for accuracy)
`metric_type`	COSINE	Use COSINE for normalized BGE embeddings

PYTHON

```python
# Search with tuned parameters
search_params = {
    "metric_type": "COSINE",
    "params": {"nprobe": 16}
}
```

```
results = collection.search(
    data=[query_embedding],
    anns_field="embedding",
    param=search_params,
    limit=10,
    output_fields=["gene", "clinical_significance", "text_summary"]
)
```

16.3 Docker Resource Limits

YAML

```yaml
# Example resource limits in docker-compose.yml
services:
  rag-api:
    deploy:
      resources:
        limits:
          memory: 16G
          cpus: '16'
        reservations:
          memory: 4G
          cpus: '4'
```

16.4 NVMe I/O Optimization

For FASTQ and BAM processing, I/O throughput is critical:

BASH

```bash
# Check NVMe performance
fio --name=seqread --rw=read --bs=1M --size=1G --numjobs=4 --runtime=10 --group_reporting

# Ensure data directories are on NVMe
df -h genomics/data/
```

16.5 Pipeline Concurrency Settings

The Nextflow pipeline supports controlled concurrency:

GROOVY

```groovy
// nextflow.config — concurrency settings
process {
    maxForks = 4          // Maximum parallel processes
    maxRetries = 2        // Retry failed processes
    errorStrategy = 'retry'
}

executor {
    queueSize = 8         // Maximum queued tasks
    pollInterval = '5 sec'
}
```

17. Troubleshooting Guide

17.1 Service Not Starting

BASH

```bash
# Check service logs
docker compose logs <service-name> --tail 50

# Check if port is already in use
ss -tlnp | grep <port>

# Restart a specific service
docker compose restart <service-name>
```

17.2 GPU Out of Memory

BASH

```bash
# Check current GPU memory usage
nvidia-smi

# Kill any orphaned GPU processes
sudo fuser -v /dev/nvidia*

# Reduce Parabricks memory by limiting GPU threads
# Add --gpu-mem-limit flag if available

# Ensure NIM services are stopped during genomics
docker compose stop molmim diffdock
```

17.3 Milvus Connection Issues

BASH

```bash
# Verify Milvus dependencies are running
docker compose ps etcd minio milvus

# Check Milvus logs for errors
docker compose logs milvus --tail 100

# Test connectivity
curl -s http://localhost:19530/v1/health/ready

# Reset Milvus if corrupted
docker compose down milvus etcd minio
docker volume rm hcls-ai-factory_milvus_data hcls-ai-factory_etcd_data hcls-ai-factory_minio_data
docker compose up -d etcd minio milvus
```

17.4 BioNeMo NIM Not Ready

BASH

```bash
# NIM services may take 2-5 minutes to load models
# Check logs for model loading progress
docker compose logs molmim --tail 50
docker compose logs diffdock --tail 50

# Verify GPU is available for NIM
nvidia-smi | grep -i "molmim\|diffdock"

# Restart if stuck
docker compose restart molmim diffdock
```

17.5 Parabricks Failures

Error	Cause	Resolution
`CUDA out of memory`	Insufficient GPU memory	Stop other GPU services first
`Reference index not found`	Missing .fai file	Run `samtools faidx GRCh38.fa`
`Input file not found`	Wrong FASTQ path	Check volume mount paths
`Unsupported GPU`	Driver mismatch	Update NVIDIA driver

17.6 Claude API Errors

Error	Cause	Resolution
`401 Unauthorized`	Invalid API key	Verify `ANTHROPIC_API_KEY` in `.env`
`429 Rate Limited`	Too many requests	Implement exponential backoff
`500 Server Error`	Anthropic service issue	Retry after 30 seconds
`Connection refused`	No internet	Check network connectivity

17.7 Docker Issues

BASH

```bash
# Docker daemon not running
sudo systemctl start docker
sudo systemctl enable docker

# Disk space full
docker system prune -a --volumes
df -h /var/lib/docker

# Permission denied
sudo usermod -aG docker $USER
newgrp docker
```

17.8 Common Error Messages Table

Error Message	Service	Resolution
`Connection refused on port 19530`	Milvus	Start etcd + MinIO first, then Milvus
`NVIDIA driver not found`	Docker	Install NVIDIA Container Toolkit
`Model not loaded`	MolMIM/DiffDock	Wait 2-5 minutes for model loading
`Collection not found`	Milvus	Run schema creation script (Section 9.2)
`API key not set`	RAG API	Set `ANTHROPIC_API_KEY` in `.env`
`Out of disk space`	Parabricks	Clean BAM intermediates, expand storage
`Permission denied: /data`	Any	Check volume mount permissions

18. VCP/FTD Demo Walkthrough

18.1 Demo Overview

The VCP (Valosin-Containing Protein) / FTD (Frontotemporal Dementia) demo showcases the full three-stage pipeline using a known pathogenic variant:

Parameter	Value
Variant	rs188935092
Location	chr9:35065263 G>A
Gene	VCP
ClinVar Classification	Pathogenic
AlphaMissense Score	0.87 (pathogenic, threshold >0.564)
Disease	Inclusion body myopathy with Paget disease and FTD
Seed Molecule	CB-5083 (VCP/p97 inhibitor)
PDB Structures	8OOI, 9DIL, 7K56, 5FTK
Binding Domain	D2 ATPase domain, ~450 cubic angstroms
Druggability Score	0.92

18.2 Pre-Demo Setup

BASH

```bash
# Ensure all services are running
bash scripts/validate_deployment.sh


# Verify Milvus has the VCP variant loaded
python3 -c "
from pymilvus import connections, Collection
connections.connect(host='localhost', port=19530)
col = Collection('genomic_evidence')
```

```
col.load()
results = col.query('gene == \"VCP\"', output_fields=['rsid', 'clinical_significance', 'am_pathogenicity'])
print(f'VCP variants found: {len(results)}')
for r in results[:3]:
    print(r)
"
```

18.3 Running the Demo

BASH

```bash
# Run the demo pipeline mode
python3 scripts/run_pipeline.py --mode demo

# Or via Nextflow
nextflow run main.nf -profile dgx_spark --mode demo
```

Step-by-step execution:

1. **Stage 1 (Genomics):** Process demo FASTQ subset through Parabricks fq2bam and DeepVariant
2. **Stage 2 (RAG):** Annotate VCP variant with ClinVar (Pathogenic) and AlphaMissense (0.87), embed into Milvus, query Claude for clinical interpretation
3. **Stage 3 (Drug Discovery):** Retrieve PDB structures (8OOI, 9DIL, 7K56, 5FTK), generate molecules from CB-5083 seed via MolMIM, dock with DiffDock, rank by composite score

18.4 Expected Results

Metric	Expected Value
Candidates generated	100
Pass Lipinski Rule of Five	87
QED > 0.67 (drug-like)	72
Top docking scores	-8.2 to -11.4 kcal/mol
Composite score range	0.68 - 0.89

Top candidate characteristics:

Property	Range
Molecular Weight	300 - 500 Da
LogP	1.5 - 4.5
QED	0.67 - 0.92
TPSA	40 - 130 squared angstroms
Docking Score	-8.2 to -11.4 kcal/mol
Composite Score	0.68 - 0.89

19. Scaling Beyond DGX Spark

19.1 Phase 1 to Phase 3 Roadmap

Phase	Hardware	Scale	Use Case
Phase 1	DGX Spark	Single workstation	Development, demos, single-patient analysis
Phase 2	DGX B200	Single server, multi-GPU	Production cohort analysis
Phase 3	DGX SuperPOD	Multi-node cluster	Population-scale genomics

19.2 Kubernetes Migration Path

For Phase 2 and beyond, migrate from Docker Compose to Kubernetes:

- Replace `docker-compose.yml` with Helm charts
- Use NVIDIA GPU Operator for GPU scheduling
- Deploy Milvus Cluster mode (distributed) instead of standalone
- Use persistent volume claims (PVCs) for data storage
- Implement horizontal pod autoscaling for RAG API

19.3 Multi-GPU Considerations

- Parabricks supports `--num-gpus` for multi-GPU parallelism
- MolMIM and DiffDock can be replicated across GPUs
- Milvus supports distributed deployment with multiple query nodes

19.4 NVIDIA FLARE for Federated Learning

For multi-institutional deployments, NVIDIA FLARE enables federated learning across DGX Spark nodes without sharing raw patient data.

20. Appendix A: Complete Configuration Reference

20.1 All Environment Variables

Variable	Default	Description
`ANTHROPIC_API_KEY`	(required)	Anthropic API key for Claude
`NGC_API_KEY`	(required)	NVIDIA NGC API key
`REFERENCE_GENOME`	`/data/reference/GRCh38.fa`	Path to reference genome
`MILVUS_HOST`	`localhost`	Milvus server hostname
`MILVUS_PORT`	`19530`	Milvus server port
`MOLMIM_URL`	`http://localhost:8001`	MolMIM NIM endpoint

`DIFFDOCK_URL`	`http://localhost:8002`	DiffDock NIM endpoint
`CLAUDE_MODEL`	`claude-sonnet-4-20250514`	Claude model identifier
`CLAUDE_TEMPERATURE`	`0.3`	Claude sampling temperature
`PIPELINE_MODE`	`full`	Pipeline execution mode
`NUM_CANDIDATES`	`100`	Number of molecules to generate
`MIN_QED`	`0.67`	Minimum QED threshold
`MIN_DOCK_SCORE`	`-6.0`	Minimum docking score (kcal/mol)
`GRAFANA_USER`	`admin`	Grafana admin username
`GRAFANA_PASSWORD`	`changeme`	Grafana admin password

20.2 AlphaMissense Thresholds

Classification	Score Range
Pathogenic	> 0.564
Ambiguous	0.34 - 0.564
Benign	< 0.34

20.3 Scoring Weights

Component	Weight
Generation Score	0.30 (30%)
Docking Score (normalized)	0.40 (40%)
QED Score	0.30 (30%)

20.4 Drug-Likeness Thresholds

Property	Threshold	Rule
Molecular Weight	<= 500 Da	Lipinski
LogP	<= 5	Lipinski
H-Bond Donors	<= 5	Lipinski
H-Bond Acceptors	<= 10	Lipinski
QED	> 0.67	Drug-likeness
TPSA	< 140 squared angstroms	Oral bioavailability

20.5 Docking Score Interpretation

Score (kcal/mol)	Binding Affinity	Assessment
< -10.0	Excellent	Strong candidate
-8.0 to -10.0	Strong	Viable candidate
-6.0 to -8.0	Moderate	Marginal candidate

> -6.0	Weak	Poor candidate

Normalization formula:

```
normalized = max(0, min(1, (10 + dock_score) / 20))
```

21. Appendix B: API Reference

21.1 MolMIM API (Port 8001)

Generate Molecules:

JSON

```json
// POST http://localhost:8001/generate
// Request:
{
  "smiles": "CC1=CC=C(C=C1)C(=O)NC2=CC=CC=C2",
  "num_molecules": 100,
  "algorithm": "CMA-ES",
  "property_name": "QED",
  "min_similarity": 0.3,
  "particles": 30,
  "iterations": 10
}

// Response:
{
  "generated_molecules": [
    {
      "smiles": "CC1=CC=C(C=C1)C(=O)NC2=CC=C(F)C=C2",
      "score": 0.85,
      "similarity": 0.78
    }
  ]
}
```

Health Check:

```
GET http://localhost:8001/v1/health/ready
Response: {"status": "ready"}
```

21.2 DiffDock API (Port 8002)

Molecular Docking:

JSON

```json
// POST http://localhost:8002/molecular-docking/diffdock/generate
// Request:
{
  "protein": "<PDB file content>",
  "ligand": "<SDF file content>",
  "num_poses": 10
}

// Response:
{
  "poses": [
    {
      "pose_id": 0,
      "confidence": 0.95,
      "score": -9.7,
      "ligand_sdf": "<docked SDF content>"
    }
  ]
}
```

Health Check:

```
GET http://localhost:8002/v1/health/ready
Response: {"status": "ready"}
```

21.3 RAG API Endpoints (Port 5001)

Method	Endpoint	Description
GET	`/health`	Service health check
POST	`/query`	RAG query with context retrieval
POST	`/search`	Vector similarity search
GET	`/collections`	List Milvus collections
GET	`/stats`	Collection statistics

RAG Query Example:

JSON

```json
// POST http://localhost:5001/query
// Request:
{
  "question": "What pathogenic variants are found in the VCP gene?",
  "top_k": 10,
  "filters": {
    "gene": "VCP",
    "impact": "HIGH"
  }
}

// Response:
{
  "answer": "The VCP gene contains the variant rs188935092...",
  "sources": [
    {
      "gene": "VCP",
      "rsid": "rs188935092",
      "clinical_significance": "Pathogenic",
      "am_pathogenicity": 0.87,
      "similarity_score": 0.94
    }
  ],
  "model": "claude-sonnet-4-20250514",
  "tokens_used": 1847
}
```

21.4 Health Check Endpoints Summary

Service	Endpoint	Method
Genomics Portal	`/health`	GET
RAG API	`/health`	GET
Milvus	`/v1/health/ready`	GET
Attu	`/api/health`	GET
Streamlit Chat	`/healthz`	GET
MolMIM	`/v1/health/ready`	GET
DiffDock	`/v1/health/ready`	GET
Discovery UI	`/health`	GET
Discovery Portal	`/health`	GET
Grafana	`/api/health`	GET
Prometheus	`/-/healthy`	GET
Node Exporter	`/metrics`	GET
DCGM Exporter	`/metrics`	GET

22. Appendix C: Schema Definitions

22.1 Milvus Collection Schema

Collection: `genomic_evidence`

#	Field	Data Type	Constraints	Description
1	`id`	INT64	Primary Key, Auto ID	Unique record identifier
2	`embedding`	FLOAT_VECTOR	dim=384	BGE-small-en-v1.5 embedding
3	`chrom`	VARCHAR	max_length=10	Chromosome (chr1-22, chrX, chrY)
4	`pos`	INT64	—	Genomic position (1-based)
5	`ref`	VARCHAR	max_length=500	Reference allele
6	`alt`	VARCHAR	max_length=500	Alternate allele
7	`qual`	FLOAT	—	Variant quality score
8	`gene`	VARCHAR	max_length=100	HGNC gene symbol
9	`consequence`	VARCHAR	max_length=200	VEP consequence term
10	`impact`	VARCHAR	max_length=20	HIGH/MODERATE/LOW/MODIFIER
11	`genotype`	VARCHAR	max_length=10	Sample genotype (0/1, 1/1)
12	`text_summary`	VARCHAR	max_length=5000	Natural-language summary

13	`clinical_significance`	VARCHAR	max_length=200	ClinVar classification
14	`rsid`	VARCHAR	max_length=20	dbSNP RS identifier
15	`disease_associations`	VARCHAR	max_length=2000	Associated diseases
16	`am_pathogenicity`	FLOAT	0.0-1.0	AlphaMissense pathogenicity
17	`am_class`	VARCHAR	max_length=20	pathogenic/ambiguous/benign

Index configuration:

Parameter	Value
Index Type	IVF_FLAT
Metric Type	COSINE
nlist	1024
nprobe (search)	16

22.2 Pydantic Data Models

PYTHON

```python
from pydantic import BaseModel, Field
from typing import List, Optional
from enum import Enum


class TargetHypothesis(BaseModel):
    """Genomic target identified from variant analysis."""
    gene: str
    variant_id: str
    rsid: Optional[str]
    clinical_significance: str
    am_pathogenicity: Optional[float]
    am_class: Optional[str]
    therapeutic_area: str
    druggability_score: float
    rationale: str


class StructureInfo(BaseModel):
    """PDB structure information for a target protein."""
    pdb_id: str
    resolution: float
    method: str
    chain: str
    binding_site_volume: Optional[float]


class StructureManifest(BaseModel):
    """Collection of structures for a target."""
    target_gene: str
    uniprot_id: str
    structures: List[StructureInfo]
    selected_structure: str


class MoleculeProperties(BaseModel):
    """Chemical properties of a generated molecule."""
    molecular_weight: float
    logp: float
    hbd: int
    hba: int
    tpsa: float
    qed: float
    lipinski_pass: bool


class GeneratedMolecule(BaseModel):
    """Molecule generated by MolMIM."""
    smiles: str
    generation_score: float
    similarity_to_seed: float
    properties: MoleculeProperties


class DockingResult(BaseModel):
    """Molecular docking result from DiffDock."""
    smiles: str
    dock_score: float  # kcal/mol (negative = better)
    confidence: float
    pose_sdf: str


class RankedCandidate(BaseModel):
    """Final ranked drug candidate with composite score."""
    rank: int
    smiles: str
    generation_score: float
    dock_score: float
    dock_score_normalized: float
    qed: float
    composite_score: float  # 0.3*gen + 0.4*dock + 0.3*qed
    lipinski_pass: bool
    properties: MoleculeProperties


class PipelineConfig(BaseModel):
    """Configuration for a pipeline run."""
    mode: str = "full"
    target_gene: Optional[str]
    seed_smiles: Optional[str]
    num_candidates: int = 100
    min_qed: float = 0.67
    min_dock_score: float = -6.0
... (16 more lines)
```

23. Appendix D: Docker Image Reference

23.1 All Container Images

Service	Image	Tag	Architecture
Parabricks	nvcr.io/nvidia/clara/clara-parabricks	4.6.0-1	ARM64 (aarch64)
Milvus	milvusdb/milvus	v2.4-latest	ARM64
MolMIM	nvcr.io/nvidia/clara/bionemo-molmim	1.0	ARM64
DiffDock	nvcr.io/nvidia/clara/diffdock	1.0	ARM64

Grafana	grafana/grafana	10.2.2	ARM64
Prometheus	prom/prometheus	v2.48.0	ARM64
Node Exporter	prom/node-exporter	latest	ARM64
DCGM Exporter	nvcr.io/nvidia/k8s/dcgm-exporter	latest	ARM64
etcd	quay.io/coreos/etcd	v3.5.5	ARM64
MinIO	minio/minio	latest	ARM64
Attu	zilliz/attu	latest	ARM64

23.2 ARM64 Compatibility Notes

The DGX Spark uses an ARM64 (aarch64) processor. All container images must be ARM64-compatible:

- NVIDIA NGC images for Parabricks, BioNeMo, and DCGM include ARM64 variants
- Community images (Grafana, Prometheus, MinIO, etcd) provide multi-arch manifests
- Custom application images must be built with `--platform linux/arm64`
- If building locally, ensure the base image supports ARM64

BASH
```bash
# Verify image architecture
docker inspect --format='{{.Architecture}}' <image-name>
# Expected: arm64

# Build for ARM64 explicitly
docker build --platform linux/arm64 -t my-service:latest ./my-service/
```

24. Appendix E: Validation Checklists

24.1 Pre-Deployment Checklist

#	Item	Command / Check	Expected	
1	DGX Spark hardware	`uname -m`	aarch64	
2	GPU detected	`nvidia-smi`	GB10 GPU listed	
3	Docker installed	`docker --version`	24.0+	
4	Docker Compose V2	`docker compose version`	v2.x	
5	NVIDIA runtime	`docker info \	grep nvidia`	nvidia listed
6	Python version	`python3 --version`	3.10+	
7	Disk space	`df -h /`	>= 320 GB free	
8	Reference genome	`ls genomics/data/reference/GRCh38.fa`	File exists, ~3.1 GB	
9	ClinVar data	`ls rag/data/clinvar/clinvar.vcf.gz`	File exists,	

			~1.2 GB
10	AlphaMissense data	`ls rag/data/alphamissense/AlphaMissense_hg38.tsv.gz`	File exists, ~4 GB
11	API keys configured	`grep ANTHROPIC_API_KEY .env`	Key set (not empty)
12	NGC key configured	`grep NGC_API_KEY .env`	Key set (not empty)
13	`.env` permissions	`stat -c %a .env`	600
14	`.env` in .gitignore	`grep .env .gitignore`	Present

24.2 Post-Deployment Checklist

#	Item	Command / Check	Expected
1	All containers running	`docker compose ps`	14+ services "Up"
2	Landing Page	`curl http://localhost:8080`	HTTP 200
3	Genomics Portal	`curl http://localhost:5000/health`	`{"status":"healthy"}`
4	RAG API	`curl http://localhost:5001/health`	`{"status":"healthy"}`
5	Milvus ready	`curl http://localhost:19530/v1/health/ready`	`{"status":"ok"}`
6	Attu UI	`curl -o /dev/null -w "%{http_code}" http://localhost:8000`	200
7	Streamlit Chat	`curl -o /dev/null -w "%{http_code}" http://localhost:8501`	200
8	MolMIM ready	`curl http://localhost:8001/v1/health/ready`	`{"status":"ready"}`
9	DiffDock ready	`curl http://localhost:8002/v1/health/ready`	`{"status":"ready"}`
10	Discovery UI	`curl http://localhost:8505/health`	`{"status":"healthy"}`
11	Discovery Portal	`curl http://localhost:8510/health`	`{"status":"healthy"}`
12	Grafana	`curl http://localhost:3000/api/health`	`{"status":"ok"}`
13	Prometheus	`curl http://localhost:9099/-/healthy`	HTTP 200
14	DCGM metrics	`curl http://localhost:9400/metrics`	Metrics text
15	Milvus collection	Python: `Collection("genomic_evidence").num_entities`	> 0

24.3 Demo Readiness Checklist

#	Item	Check	Expected
1	All services healthy	Run `validate_deployment.sh`	All [OK]
2	VCP variant in Milvus	Query gene="VCP"	rs188935092 found
3	ClinVar annotation	VCP classification	Pathogenic
4	AlphaMissense score	VCP am_pathogenicity	0.87
5	PDB structures accessible	Query RCSB for VCP	8OOI, 9DIL, 7K56, 5FTK
6	MolMIM generates	Test generation from CB-	Molecules returned

		5083	
7	DiffDock docks	Test docking against VCP structure	Scores returned
8	Claude responds	Test RAG query about VCP	Coherent response
9	Grafana dashboards	Login at port 3000	Dashboards visible
10	GPU metrics flowing	Check DCGM in Grafana	GPU util, memory shown

25. Appendix F: Glossary

25.1 Genomics Terms

Term	Definition
FASTQ	Text-based format for storing nucleotide sequences and quality scores
BAM	Binary Alignment Map — compressed format for aligned sequencing reads
VCF	Variant Call Format — standard format for genomic variants
SNP	Single Nucleotide Polymorphism — single base-pair variant
Indel	Insertion or deletion of nucleotides in the genome
WGS	Whole Genome Sequencing — sequencing of entire genome
GRCh38	Genome Reference Consortium Human Build 38 — current reference genome
GIAB	Genome in a Bottle — NIST benchmark samples (e.g., HG002)
ClinVar	NCBI database of clinically relevant genomic variants
VEP	Variant Effect Predictor — functional annotation tool
AlphaMissense	DeepMind model predicting missense variant pathogenicity
Paired-end	Sequencing both ends of a DNA fragment for improved alignment
Coverage (30x)	Average number of reads covering each position in the genome

25.2 ML/AI Terms

Term	Definition
RAG	Retrieval-Augmented Generation — combining search with LLM generation
Embedding	Dense vector representation of text or data
BGE	BAAI General Embedding — sentence transformer model family
IVF_FLAT	Inverted File Index — approximate nearest neighbor search method

COSINE	Cosine similarity — metric for comparing vector directions
NIM	NVIDIA Inference Microservice — containerized model serving
LLM	Large Language Model — e.g., Claude
Vector Database	Database optimized for similarity search on dense vectors
nlist	Number of clusters in IVF index (build-time parameter)
nprobe	Number of clusters to search at query time (recall vs. latency)

25.3 Drug Discovery Terms

Term	Definition
SMILES	Simplified Molecular Input Line Entry System — text notation for molecules
PDB	Protein Data Bank — repository of 3D protein structures
Molecular Docking	Computational prediction of ligand-protein binding pose and affinity
QED	Quantitative Estimate of Drug-likeness — composite drug-likeness score (0-1)
Lipinski Rule of Five	Empirical rules predicting oral bioavailability
TPSA	Topological Polar Surface Area — predictor of membrane permeability
LogP	Partition coefficient — measure of lipophilicity
HBD / HBA	Hydrogen Bond Donors / Acceptors
Conformer	3D spatial arrangement of a molecule's atoms
Binding Affinity	Strength of interaction between a drug molecule and its target protein
kcal/mol	Kilocalories per mole — unit for binding energy (more negative = stronger)
MolMIM	Molecule generation model from NVIDIA BioNeMo
DiffDock	Diffusion-based molecular docking model
Druggability	Assessment of whether a protein target can be modulated by a small molecule
CB-5083	VCP/p97 inhibitor used as seed molecule in the VCP demo
RDKit	Open-source cheminformatics toolkit for molecular analysis

This deployment guide is maintained as part of the HCLS AI Factory open-source project. For updates, issues, and contributions, visit the project repository on GitHub.