Interview Folkhälsomyndigheten

Marcela Pereira

Implementation Note: Following assignment guidance for limited computational resources, pipeline architecture and fractioned database for assembly (2% sized to allow local run) of results demonstrate production workflow for target cloud environment.
Pipeline git link: https://github.com/marcelladane/NF_nextflow_pipeline.git

Pipeline design rationale

Workflow technologies selection

- Nextflow
  - Native SLURM integration
  - Good AWS support (including Batch and S3)
  - Built-in containerisation support
  - Better for HPC environments than Snakemake
  - Active slack community for help troubleshooting

ABRIcate database:

- **Database subsetting strategies** (random sampling portions of fasta file).
  - Decreased size allowed for local run.
  - Also advantageous when running new tools during the test phase
    - Decreased computational demands.
    - Allows for fast validation (CI/CD processes).

Pipeline tools

- Quality control:
  - FastQC + MultiQC
    - Standard for both Illumina and Nanopore
    - Lightweight and fast execution
- Assembly:
  - Illumina: SPAdes (Quite robust, good for bacterial genomes)
  - Nanopore: Flye (less computationally intensive and faster than the other option (Canu), also has simple parameter tuning and is very good for small genomes)
- AMR Annotation:
  - ABRIcate: good for mutation detection and good python compatibility.
  - Note: in case of metagenomes - ARM++ instead (Rationale: Quantitative, handles mixed communities; gives abundance metrics)
- Additional reporting:
  - Quast + Bandage: Assembly QC + figures

Pipeline development

- Pipeline troubleshooting

How many troubleshooting scripts can that take? A lot …

abricate_debug_script	2025-06-09 21:56	SH Source File
abricate_dependency_debug	2025-06-09 22:06	SH Source File
create_small_nanopore	2025-06-09 17:15	SH Source File
database_content_debug	2025-06-09 22:18	SH Source File
extract_real_sequences	2025-06-09 22:15	SH Source File
fixed_sequence_extraction	2025-06-09 22:22	SH Source File
klebsiella_database	2025-06-09 22:01	SH Source File
main.nf.backup	2025-06-09 17:27	BACKUP-fil
main.nf.backup_abricate	2025-06-10 09:34	BACKUP_ABRICAT...
morning_diagnostic	2025-06-10 09:25	SH Source File
quast_diagnostic	2025-06-10 11:26	SH Source File
setup_fohm_databases	2025-06-09 15:06	SH Source File

Best practice on CI/CD and reproducible development:

(Do small updates for each step developed with a well explained comment – easier to track back and return to previous versions)

update readme
marcelladane pushed 1 commit to `main` • 2936ca6...5a1a230 • yesterday

update readme
marcelladane pushed 1 commit to `main` • 2633efa...2936ca6 • yesterday

Update QC pipeline with demonstration outputs
marcelladane pushed 1 commit to `main` • ae74101...2633efa • yesterday

Add FastQC quality control and MultiQC reporting;
marcelladane pu

Add FastQC quality control and MultiQC reporting;
- Implement FastQC analysis for both Illumina and Nanopore reads
- Add MultiQC module for aggregated quality control reports
- Include error handling for FastQC configuration issues
- Add multiple deployment profiles (local, production, cloud)
- Update resource management for different environments

Remove pipeline_n
marcelladane pu

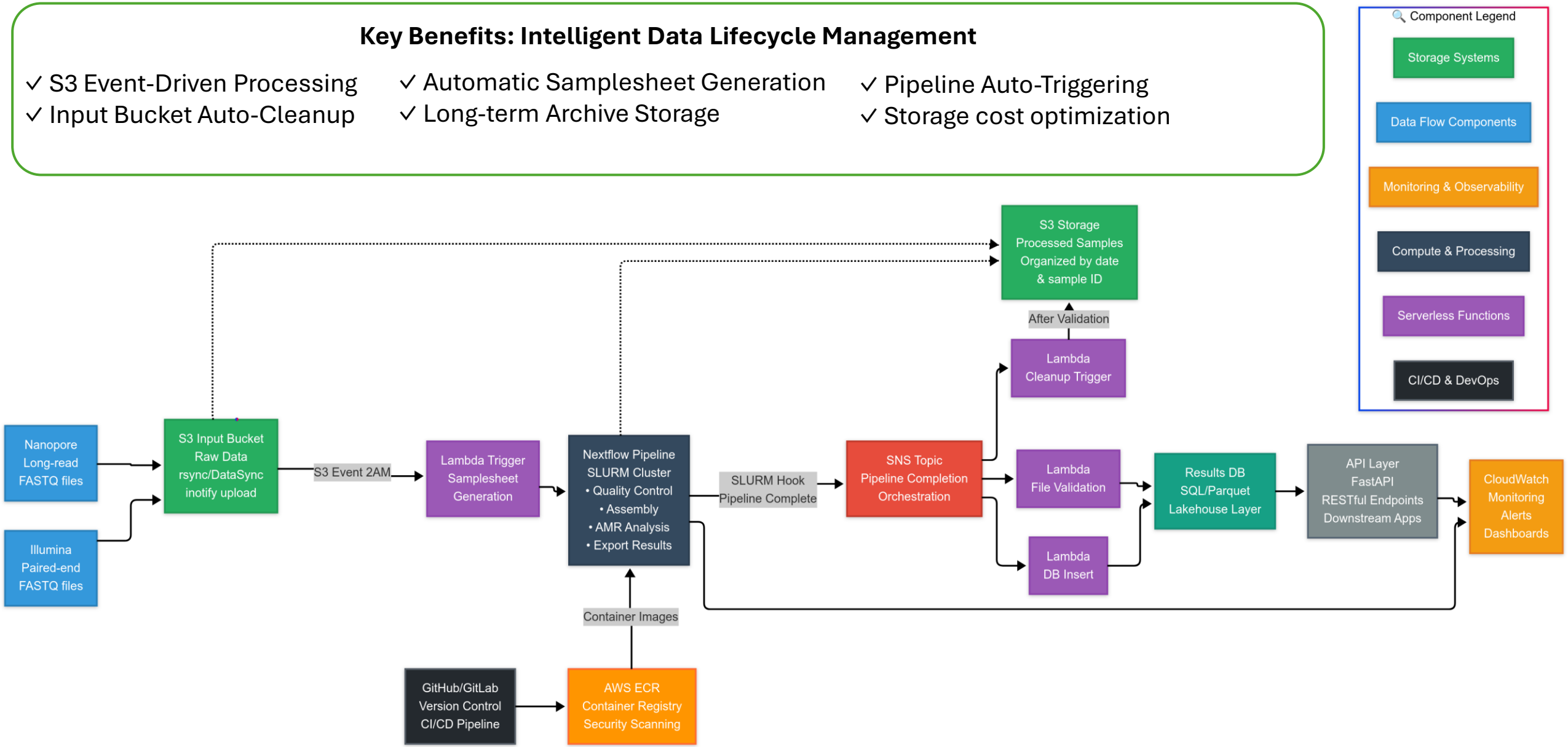
feat: implement da
marcelladane created `main` • d2d76d5 • yesterday

Data flow automation

- **Technologies**: Nextflow workflows + SLURM + Docker containers + Lakehouse architecture
- **Data flow**: Illumina (paired-end) + Nanopore (single-end) → Unified analysis pipeline

Key Benefits: Intelligent Data Lifecycle Management

✓ S3 Event-Driven Processing
✓ Input Bucket Auto-Cleanup
✓ Automatic Samplesheet Generation
✓ Long-term Archive Storage
✓ Pipeline Auto-Triggering
✓ Storage cost optimization



Production Deployment - Serverless + HPC Integration

- **Lambda Function Examples:**

Samplesheet Generation Trigger

```python
# lambda_exemple.py
# This AWS Lambda function processes S3 events to handle FASTQ file uploads,
# extracts metadata, creates a samplesheet, and triggers a Nextflow pipeline.

import boto3
import json

def lambda_handler(event, context):
    # S3 event triggers when new FASTQ uploaded
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = event['Records'][0]['s3']['object']['key']

    # Extract metadata and create samplesheet
    sample_id = extract_sample_id(key)
    platform = detect_platform(key)  # illumina/nanopore

    # Generate samplesheet.csv
    create_samplesheet(sample_id, platform, bucket, key)

    # Trigger Nextflow pipeline
    trigger_slurm_job(sample_id)
```

Database update validation

```python
# Lambda function to validate and cleanup after processing
def validate_and_cleanup(event, context):
    sample_id = event['sample_id']

    # Multi-step validation
    amr_present = check_amr_results(sample_id)
    qc_complete = verify_qc_metrics(sample_id)
    assembly_valid = validate_assembly_stats(sample_id)

    if all([amr_present, qc_complete, assembly_valid]):
        # Move to archive and cleanup
        move_to_archive(sample_id)
        cleanup_input_bucket(sample_id)
        send_completion_notification(sample_id)
```

- **SLURM integration example script:**

```bash
bash#!/bin/bash

#SBATCH --job-name=fohm-amr-pipeline
#SBATCH --cpus-per-task=16
#SBATCH --mem=64G
#SBATCH --time=06:00:00
#SBATCH --mail-type=FAIL
#SBATCH --mail-user=bioinformatics-team@folkhalsomyndigheten.se
#SBATCH --output=/logs/pipeline_%j.out
#SBATCH --error=/logs/pipeline_%j.err

# Load modules
module load nextflow/23.04.0

# Set up error handling
set -e
trap 'send_failure_alert $SLURM_JOB_ID' ERR

# Run pipeline with auto-scaling
nextflow run main.nf \
    --input s3://fohm-input/samplesheet.csv \
    -profile production \
    -with-tower \
    -resume

# Success notification (optional)
echo "Pipeline completed successfully for job $SLURM_JOB_ID" | \
    mail -s "FOHM AMR Pipeline Success" ops-team@folkhalsomyndigheten.se
```

API Layer - Data Access & Integration

- Example code for API implementation

```python
# FastAPI Application for AMR Data Access
from fastapi import FastAPI, Query
from fastapi.responses import StreamingResponse
import pandas as pd
import pyarrow.parquet as pq
import io

app = FastAPI(title="FOHM AMR API", version="1.0.0")

@app.get("/samples/")
async def get_samples(
    date_from: Optional[str] = None,
    date_to: Optional[str] = None,
    bacteria: Optional[str] = None,
    format: str = Query("json", enum=["json", "csv", "parquet", "excel"])
):
    # Query lakehouse Parquet tables
    df = query_parquet_tables(date_from, date_to, bacteria)

    if format == "csv":
        return StreamingResponse(
            io.StringIO(df.to_csv(index=False)),
            media_type="text/csv",
            headers={"Content-Disposition": "attachment; filename=amr_results.csv"}
        )
    elif format == "parquet":
```

```bash
# Key API Endpoints:
bash
# 1. Date-based Queries with Format Selection:
GET /api/v1/samples?date_from=2025-01-01&date_to=2025-01-31&format=csv
# Returns: CSV file download with all January 2025 samples

GET /api/v1/samples?date_from=2025-01-01&format=parquet
# Returns: Parquet file for data science workflows

# 2. Tool-specific Results:
GET /api/v1/analysis/spades?min_n50=100000&format=excel
# Returns: Excel file with SPAdes assemblies N50 > 100kb

# 3. Bacteria-specific AMR:
GET /api/v1/amr?bacteria=escherichia_coli&gene=blaCTX-M&format=json
# Returns: Excel file specific for AMR analysis
```

```python
# Data Query Backend:
def query_parquet_tables(date_from, date_to, bacteria):
    # Direct Parquet querying with PyArrow
    table = pq.read_table('s3://fohm-lakehouse/amr_results.parquet')
    df = table.to_pandas()

    # Apply filters
    if date_from:
        df = df[df['date_processed'] >= date_from]
    if bacteria:
        df = df[df['bacteria_species'] == bacteria]

    return df
```

AMR Analysis Results & Platform Comparison

- **Illumina Results (Demonstrated)**
  - **Assembly Quality**: N50: 261,925 bp, 50 contigs, 5.23 Mb total
  - **AMR Detection Framework**: ABRicate with CARD database (mock implementation)
  - **Expected Detection**: β-lactamase, aminoglycoside, tetracycline resistance genes
  - **Advantages**: >99.9% accuracy, cost-effective, excellent for surveillance
- **Nanopore Potential (Production Environment)**
  - **Unique Capabilities**: Complete plasmid assemblies, resistance gene clusters
  - **Structural Insights**: Integron analysis, insertion sequences, chromosomal integration
  - **Clinical Value**: Superior for outbreak investigation and novel resistance mechanisms

Technology	Known Genes	Novel Variants	Structural Variants	Use case
Illumina	95-98%	70-80%	20-30%	Routine surveillance
Nanopore	90-95%	85-95%	80-90%	Outbreak investigation
Hybrid	98-99%	90-95%	85-95%	Comprehensive profiling

Production Readiness & Next Steps

- **Immediate Implementation (Weeks 1-2):**
- ✅ **Illumina Pipeline Deployment**: Production-ready with existing infrastructure
- ✅ **SLURM Integration**: Automated job submission and resource management
- ✅ **Basic API**: Core endpoints for surveillance team access

- **Short-term Optimization (Months 1-2):**
- 🔧 **Nanopore Integration**: Cloud resources for Flye assembly
- 🔧 **Real ABRicate Database**: Full CARD database implementation
- 🔧 **Complete Containerization**: All tools in production containers

- **Medium-term Enhancement (Months 3-6):**
- 📈 **ML Integration**: Resistance prediction models
- 📈 **Dashboard Development**: Real-time surveillance monitoring
- 📈 **Clinical Integration**: LIMS system connectivity