VarFish DevDocs

Release 0.1.0-8-g0b3119e

Manuel Holtgrewe

May 13, 2024

DOCUMENTS

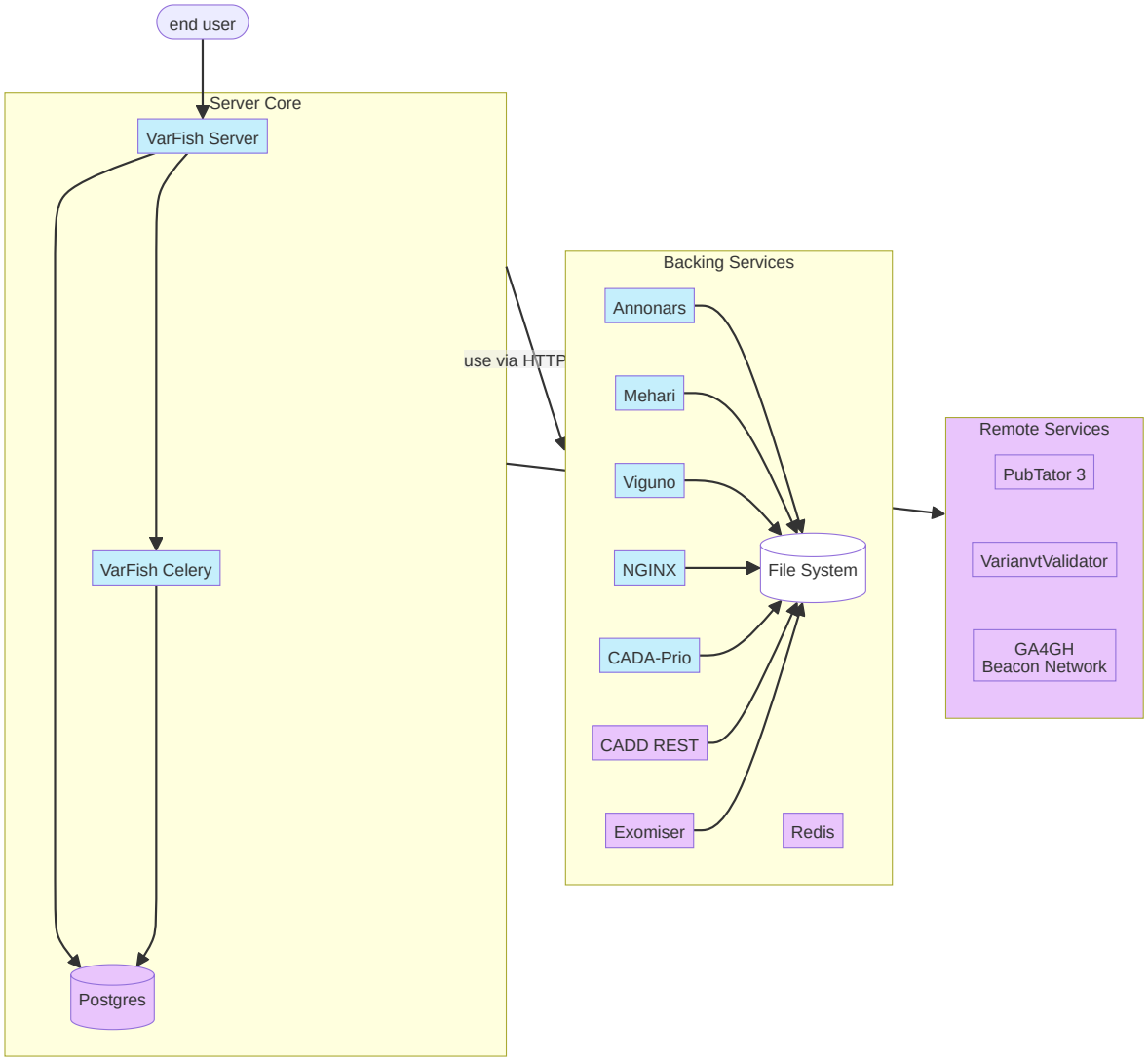
CHAPTER

ONE

ARCHITECTURE

This section describes the overall architecture of VarFish. Below is a high-level overview of the software components and the interaction with the end user. Components developed by the team are shown in blue, third-party components are depicted in violet, component groups are in yellow color.

All components run in a Docker Compose environment. Also, we use the Traefik reverse proxy for routing requests to the correct service (not shown below).



end user

The end user (data analyst) uses their web browser to connect to the Varfish Server and interacts with the system.

operator user
: The user operating a VarFish instance interfaces with the system also via the web user interface. Certain actions must be performed via REST APIs provided by the VarFish Server, in particular importing data for later analysis.

1.1 Core System

The following services comprise the Core System that implements the business logic. This system is aware of the currently logged in user and the information stored in the Postgres database such as the case information.

VarFish Server
: The VarFish Server provides a web-based interface to the software and implements parts of the logic. It provides Python/Django web application that serves as the backend for the frontend implementing the product's core functionality. The core functionality is implemented in a TypeScript/Vue based single page app (SPA) that itself is served by the backend and then uses REST APIs to communicate with the backend.

Varfish Celery
: We use the Celery task queue system to run jobs in the background that cannot be executed in very short time. The Server uses this for running queries, imports. Also, Celery is used for scheduling maintenance tasks such as building the in-house database.

Postgres
: We use the PostgreSQL relational database management system to store large parts of the data. For large tables, sharding/partitioning is used for improving performance.

Note: The bulk of the data is currently stored in Postgres. Work is underway to move this to an internal object storage and run queries on this storage. This will allow for more optimized queries and scaling as the Postgres system will not be the single bottleneck anymore.

1.2 Backing Services

There is a list of services that run in the background within the VarFish instance that the user does not interact with directly. They provide HTTP-based URLs to the core system then are stateless. There is no interaction betwen these services.

File System
: These servics generally store their data on the file system.

Annonars
: The Annonars service provides fast access to information specific to genes, seqvars, and strucvars. For example, it stores the gene overview information, gene-wise aggregated ClinVar information, and precomputed variant scores. Note that the static precomputed gene information includes the link between genes and conditions. This service requires large amounts of local storage.

Mehari
: The Mehari service provides computations of variant effects on the transcript level. For example, it can predict that a given genomic variant leads to missense or frameshift change on a protein or predict that a structural variant creates a breakpoint in an exon or intron. Mehari also provides access to gene transcript information that can be used for rending exon/intron graphics.

Viguno
: The Viguno service provides access to the Human Phenotype Ontology (HPO). First, it provides access to the HPO in the common ontology/graph-based fashion, allowing for linking between terms and terms, terms and diseases, etc. It also provides simple similarity computations based on information content. Second, it provides

a full text index on the HPO text content. This allows for looking up HPO terms based on their names, aliases, but also descriptions.

NGINX
The NGING service is a simple HTTP web server that is used for serving static files. This is used for serving genome browser tracks, for example.

CADA-Prio
This is a service that provides similarity predictions between lists of terms and genes based on knowledge graph embeddings. It allows for prioritizing genes given the phenotypic description of a patient.

CADD REST
This is a thin wrapper that provides access to the third-party *CADD scripts*, a software package allowing for the computation of genomic variant scores. The CADD score authors provide precomputed scores for all genomic single nucleotide variants and a list of known indel variants. For scoring novel variants, this is needed. This service requires large amounts of local storage. Also, the CADD scripts use various external software such as the ENSEMBL Variant Effect Predictor.

Exomiser
This is a third-party service implementing several algorithms for computing similarity between lists of phenotype terms and genes.

Redis
This is a key-value store that is used for caching and storing temporary data by the core services.

Note: These services generally only need little storage space with the exception of Annonars and CADD REST. The small amounts of data could be downloaded from a central location on startup in future versions. In the case that the large storage requirements of Annonars pose a problem, a migration to object storage backend would need to be implemented. Candidates are TileDB. CADD REST is more problematic.

Note: With recent versions of the HPO, information content is not very useful for variant prioritization.

1.3 Remote Services

VarFish also provides access to certain remote services run by third parties. This reduces the complexity of local hosting and keeping data up to date and even is necessary for some kinds of services. On the other hand, it makes the instance rely on the availability of these remote services.

PubTator 3
VarFish uses the PubTator 3 API for providing relevant literature information for genes.

VariantValidator
The VariantValidator.org service is used for providing gold standard HGVS descriptions for seqvars.

GA4GH Beacon Network
The GA4GH Beacon Network embeddable IFRAME is used for alllowing to query the GA4GH Beacon Network for variant information.

Genomics England PanelApp
We use the GE PanelApp API for fetching up-to-date gene panel information.

CHAPTER

TWO

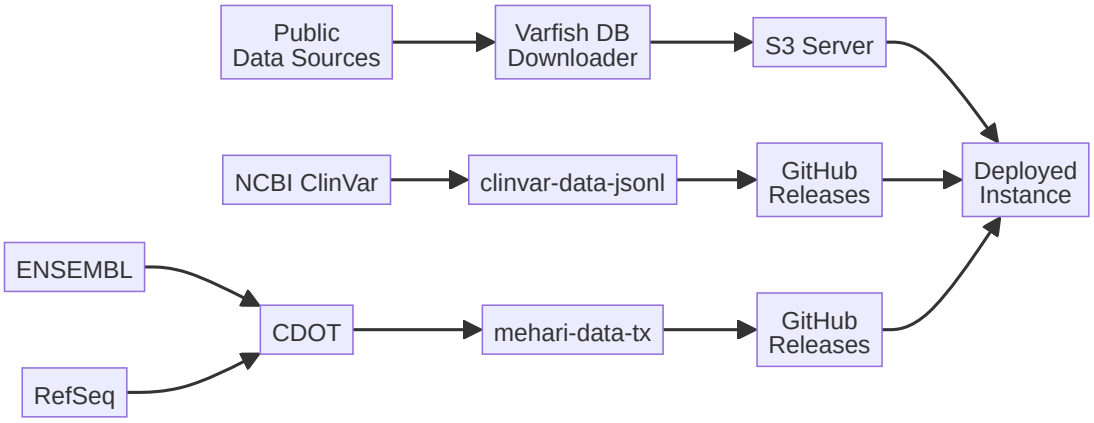
DATAFLOWS

This section describes the dataflows in the VarFish system. We split the description into the following parts.

1. *Bulk Data Preparation* describes the dataflow for preparing the bulk background used by the *Backing Services* from *Architecture*.
2. *Annotation Process / Import* describes the annotation process that prepares variant VCF files for import into VarFish and the import itself.
3. *Query Processing* describes how the VarFish Server handles queries.
4. *Periodic Background Tasks* describes the dataflows by the periodic background tasks.
5. *User Interaction* describes the remaining dataflows done by the user annotation.

2.1 Bulk Data Preparation

There are three parts to the bulk data preparation, depicted below.



First, we use a Snakemake worfklow (called `varfish-db-downloader`) that downloads the necessary public domain data from the internet for most of the data. The data is then processed with the workflow and the bulk data files are created that can be used by the Backing Services.

The workflow is executed manually by the VarFish team. The results are uploaded to our public S3 servers. On deployment, the files are downloaded by downloader/installer scripts that the team provides.

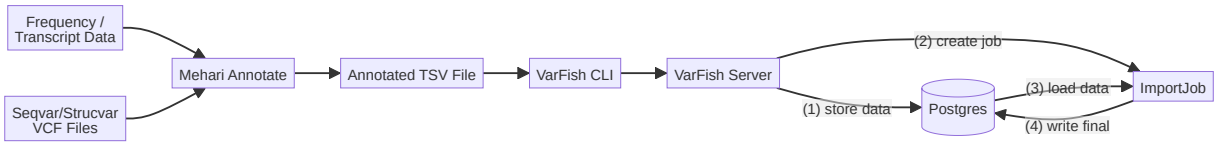
The workflow features a continuous integration test mode where file excerpts are used for smoke testing the functionality of the workflow. Further, the continuous integration checks availability of the upstream files. Using a Snakemake workflow together with using a conda environment for dependencies allows for reproducible data preparation.

ClinVar data is prepared differently. Here, we have a software `clinvar-this` that is capable of converting ClinVar XML files and convert them into JSON lines (JSONL) format. These JSONL files can then be processed by the software packages also used in the Backing services. The GitHub repository `clinvar-data-jsonl` hosts continuous integration that downloads the weekly ClinVar releases, uses `clinvar-this` to transform the XML files to JSONL, and finally publish them as GitHub software releases. A third GitHub repository `annonars-data-clinvar` uses the output of `clinvar-data-jsonl` to prepare the per-gene aggregations and per-variant ClinVar files to be used by the Annonars Backing Service. These files are installed on deployment and can later be updated.

Transcript data is also prepared differently. We use the output of the third-party CDOT project that provides RefSeq and ENSEMBL transcripts. The CI in the GitHub project `mehari-data-tx` downloads the transcripts from the CDOT releases and fetches the corresponding sequences form the NCBI and ENSEMBL servers. It then prepares the transcript data files for the genome releases with the Mehari software. The resulting files are then also published as GitHub software releases. As for the ClinVar files, these files are installed on deployment and can later be updated.

2.2 Annotation Process / Import

Variant callers create variant call format (VCF) files that first must be annotated into tab separated value (TSV) files before import into VarFish. For this, we use the Mehari software. Mehari uses population frequency and transcript data files generated by the *Bulk Data Preparation* step that must be downloaded once.

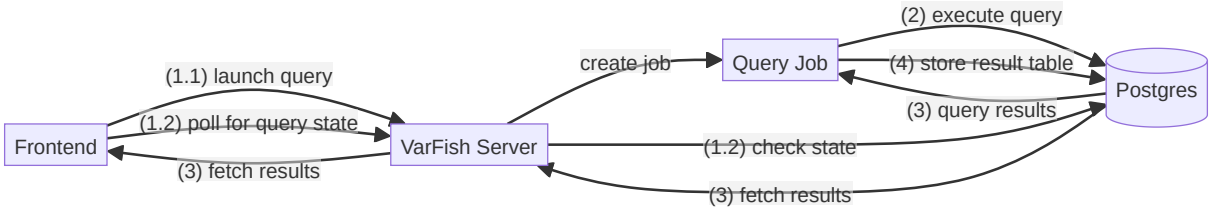


The VarFish operator user then uses Mehari to annotate and aggregate each the sequence and the structural variant VCF files into on TSV file per variant type (seqvar/strucvar). These files are then uploaded via the VarFish Command Line Interface (CLI).

The VarFish Server stores the uploaded data in the Postgres database and creates a background job for importing the data. When the import job is run, it will perform certain data processing such as computing quality control metrics and performing fingerprinting of the variant data to allow checking for family relationships. The resulting data is then stored in the final location in the Postgres database where it is available to the user.

2.3 Query Processing

Query processing is straightforward and the same for seqvar and strucvar queries.



The user ceates a new query in the frontend provided by VarFish Server. The server creates a query background job with the query specificaiton for execution in the background.

When the job is executed, it loads the query, generates a Postgres SQL query and executes it. The resulting rows are inserted into the query results table for use by the user.

The frontend polls the server for the state of the query. When the query is complete, the data is loaded into the frontend for interaction by the user.

2.4 Periodic Background Tasks

There is a number of background tasks that work on the database. The most important maintenance task rebuilds the in-house background database. This is currently done by re-creating a materialized view in the Postgres database.

2.5 User Interaction

Besides query processing, the user can interact in various ways. This interactive works leads to transactional/atomic updates in the database, e.g., by editing properties of a case or annotating case members with HPO terms. This is done with operations that appear blocking to the client and not in background tasks.

CHAPTER

THREE

FUTURE PLANS

This section contains a description of upcoming high-level changes to the VarFish software. The sections below have been extrapolated from the current issue list.

3.1 Technical Debt

There is some technical debt, some notable items.

Automated Tests
: We need more automated tests in various areas throughout the codebase.

Python Type Annotations
: We should modernize the codebase by comprehensive use of Python type annotations. Notable, this will require type annotations for sodar-core.

Backing Server Protobuf Migration
: The backing services should expose JSON-serialized protobufs on their APIs. This will allow for code generation of API clients.

3.2 GRCh38 Migration

We support instances with variants in GRCh37, GRCh38, or both genome build coordinates. However, there is no good upgrade path implemented yet. The plan here is to provide semi-automated ways to lift over user annotations from GRCh37 to GRCh38 and merely notify people about the cases where this fails. Variants will need reprocessing and reimport of the original data as we don't consider lift-over of variants to be reliable in the general case.

3.3 Vuetify Migration

We need to finalize the migration to the Vuetify framework in the frontend. Further, we should have the VarFish-specific part of the site (outside of what is provided by sodar-core) become a true SPA without embedding into the HTML + Bootstrap CSS.

3.4 Custom Query Presets

We already have some support for custom query presets. The seqvar query presets need fixing and extension. The strucvar query presets is missing large parts of implementation (all of the editing functionality).

3.5 Case Management

We can attach states such as "closed as solved" to cases, note, and comments. However, assigning responsible persons to cases and implementing real "workflows" on the case level, e.g., with approval of supervising physicians, is missing.

3.6 ClinVar Uploads

ClinVar uploads are currently missing. We have the library to perform this already in place and a working implementation of UI can be found in REEV. This needs careful planning and integration with *Case Management*.

3.7 Useability Improvements

The VarFish user interface is useable. However, there is the need for various improvements to improve the user experience.

Faster Flagging
: The aim here is to provide users with faster "time to flag as X" for variants. This is particularly important for visual artifacts in IGV or variants that do not have a connected phenotype. This includes both the decision making and access to setting flags.

Better Details Access
: The aim here is to provide users with the information that they need for assessing a variant faster. This includes both gene and variant details. Improvements can be done by selecting which information where in a smarter way. Further improvements can be of technical nature (fewer clicks, faster load times).

Smart Information Access
: In all relevant places such as query result view, variant details, gene details. Again, the aim is to provide users with the information that they need for assessing a variant faster.

Blinded Case Analysis
: The aim here is to provide a blinded four-eyes principle for case analysis.

Other Carriers
: The aim here is to provide a quick way to see other cases of the same/similar variants (same genomic position, same amino acid, same gene). Also, a visualization of the variation landscape in the gene in the database vs. ClinVar vs. gnomAD would be useful.

Second Hit
: In the case of recessive disorders, it should be faster to find a second hit. The idea is that based on a suspicious pathogenic variant, the second hit can be easily found. This could be a strucvar overlapping the same gene or another sevar in the same gene that is harder to interpret. E.g., a splicing, deep intronic, and/or UTR variant.

Phased Variants
: Haplotype-based variant callers provide phasing information, at least if one read can cover two variants. We currently don't expose this information.

3.8 Integrated Variant Analysis

There currently is a strong separation between seqvar and strucvar analysis. We should implement several strategies for an integrated analysis, taking the case phenotype into consideration.

3.9 Report Generation

There are multiple aspects to report generation. This could consist of providing detailed pages for variants with a selected criteria (e.g., all flagged/considered variants). Alternatively, this could consist of automatically filling letter templates with variant information.

3.10 Cohort Filtering

A much requested feature is performing queries on a cohort level. We already had a version working earlier that had problems with performance. This needs to be re-tackled after the migration to the next-gen dataflows is complete.

3.11 Next-Gen Dataflows

The classic location for variants in VarFish is the postgres database. This works quite well on fast baremetal NVME disk arrays but makes the database the single bottleneck. It is thus desirable to reengineer this part and work is already underway on this.

We will rather work with data in object storage via the S3 protocol. By default, Varfish instances will come with an embedded MinIO server for this purpose but external servers can also be used. Users upload their case files to a location VarFish can access (e.g., S3, HTTPS, local file system) and VarFish is told the location and possibly the necessary credentials. For import, users only upload a Phenopackets YAML file with the case manifest. VarFish then imports the case in a background job. Only the essential files such as variant data (VCF) and QC files are actually read. Other files such as BAM files, coverage `.wig` files, etc. are registered in the database (this allows proxying to them and redisplaying as also mentioned in *Genome Browsers*).

VarFish then runs an ingest step that processes the raw caller VCF files and potentially merges VCF files from the same caller. The resulting ingested VCF files are then stored in the internal object storage. Further preprocessing can take place, e.g., prefiltering to certain variants such as near-exonic ones. QC data is imported into the database and potentially additional QC is computed. Filtration is also done directly on the VCF files from the internal S3 object storage.

The data import is partially done in the server. We already have fast Rust-based executables for the variant ingest and query execution. There are unit tests for these components but no integration or system tests yet. Further, the integration in the server/frontend has not been started yet.

The best way forward is to keep this "next-gen dataflow" in addition to the classic version. Cases imported in the new way get a tag "version=2" and the new (and yet to be implemented in some parts) code paths will be used for them while the legacy code paths will remain.

3.12 ACMG Criteria UI

We currently have a working version of Richars et al. 2015 implemented. We need to bring this to the latest ACMG version, ideally both score- and rule-based with certain rule sets (e.g., ACGS, AMP, etc.). Further, we are completely lacking this for strucvars. For the latter, this strongly depends on *ACMG Automation* as the rules are highly complex here.

3.13 ACMG Automation

We need to implement ACMG implementation. We have a working implementation (not widely tested) for strucvars that is only missing PVS1 automation. Seqvars is completely missing.

3.14 ClinGen VCEP

There is a number of genes for which experts have developed complex rule sets. It would be very useful to have a "rule engine" (could just be some per-gene Python code maintained and deployed with VarFish server) that supports users in these well-known genes with complex rules.

3.15 Additional Variant Types

We currently only support seqvars and strucvars. The following variant types are commonly called from NGS (short and long-read) data.

Repeat Expansion
: E.g., with ExpansionHunger from short-read data or directly from long-read data.

ROH (Run of Homozygosity) / LOH (Loss of Heterozygosity)
: Useful for computing scores such as autozygosity which provides insights into relationships and is useful for quality control. ROH data is also often used for the identification of candidate regions. It will be easy to implement a graphical tool for homozygosity mapping.

SMA (Spinal Muscular Atrophy) Calling
: There are specialized callers to call SMA mutations from NGS data which is challenging and included in DRAGEN output. However, it is questionable how useful this is in a clinical setting as there are cheaper standard tests.

CYP2D6 Caller
: Similar to SMA calling, there are callers and one is included in DRAGEN output. However, questionable how important this is.

HLA Calling
: HLA calling can be important in certain aspects and by now there are decent callers for NGS available. Again, it is questionable how much demand there is for it.

Methylation Calling
: ONT sequencing provides methylation information. Such information could also come from a matched methylation array.

3.16 Long Reads

We currently have "long read support" already as we can import variants from such data. However, we will need to adjust rule sets and extend the builtin presets. As outlined in *Additional Variant Types*, it also gives support to methylation information.

3.17 RNA-Seq

The integration of DNA variant data and RNA-seq expression data can be useful. However, there are not many proven cases for *ab initio* RNA-seq for gene prioritization. Maybe this is primarily useful for integrated analysis where RNA-seq is used for follow-up.

3.18 Genome Browsers

After implementing *Next-Gen Dataflows*, we also have information about the BAM files in external locations linked to from VarFish. We can then proxy HTTP requests to them via VarFish and generate IGV sessions or display them in integrated genome browsers such as IGV.js or alternatives.

3.19 Local PubTator

PubTator is very useful for semantic search of literature connected to a gene. The public API has a rate limit. It is open source and all data is available in monthly dumps. It might make sense to create a local mirror but this would increase the gap between publication and availability in VarFish to up to a month. An alternative would be to roll our own engine based on a full text search engine such as QuickWit and open source named entity recognition libraries and ingest the sub-daily releases of PubMed abstracts.

3.20 Facial Gestalt Integration

Facial gestalt matching is a useful technique for variant priorization. There is a prototype integration with Gestalt-Matcher from Bonn. This integration needs work for a production-ready state but this can also lead into starting out with plugin extension points for VarFish for the deep integration of further external tools.

3.21 Somatic Variant Analysis

Alternative tools such as cBioPortal are well-suitable for the analysis of cancer variant data, in particular in a cohort fashion. However, in certain cases, the analysis of cancer cases with VarFish could be useful.

3.22 Beacon Networks

There is some implementation of connecting two VarFish instances via the Beacon API. This could be explored further or removed.

3.23 REEV Community

We have implemented a public single-variant interpretation tool called REEV. VarFish instances could be connected together by registering variant annotations and comments there and thus sharing knowledge and connecting to other users. More features could be implemented to create "groups" in REEV, such that consortia could use it as a connecting component for their local VarFish instances.

3.24 Pipeline Integrations

We could implement a feature that allows for integrating data processing pipelines with VarFish. Users could register meta data together with their FASTQ files or even flow cell raw data. The pipelines could then be started running mapping, variant calling, and QC etc. The results could then be imported into VarFish. VarFish would orchestrate the pipeline runs through existing external software.

Potential existing pipelines include DRAGEN, ParaBricks, or custom Nextflow / Snakemake pipelines.

3.25 Plugin Extension Points

VarFish could serve as a platform for the integration of external tools. Working examples are the Exomiser for variant prioritization and an emerging one is the GestaltMatcher integration in *Facial Gestalt Integration*. Allowing further integration with other prediction tools or LIMS systems (Gepardo?) could offer the vendors of such tools to integrate well with VarFish.

3.26 Comprehensive APIs

Current API support focuses on what the frontend needs and we don't have comprehensive APIs yet. Having such APIs would be very useful though, and enable using VarFish as a backend for other tools and platforms.

3.27 Scriptable VarFish

In the inverse of *Comprehensive APIs*, we could offer scripting of the query engine. This would allow advanced users to implement comprehensive analysis directly in VarFish.

CHAPTER

FOUR

GITHUB OVERVIEW

This section gives an overview of the GitHub repositories relevant to the VarFish project. All are in `varfish-org`, version 2.0 of the Apache license is used.

4.1 Direct Importance

The following repositories are directly important and provide functionality used by the system.

name	license	synopsis
annonars	MIT	precomputed variant, region, and gene annotation
annonars-data-clinvar	MIT	ClinVar data builds for annonars
cada-prio	MIT	phenotype-based prioritization based on knowledge graph embeddings
cadd-rest-api	MIT	REST API wrapper for CADD-scripts
cadd-rest-api-mock	MIT	mocking the CADD REST API server for VarFish development
clinvar-data-jsonl	MIT	weekly ClinVar releases as JSONL plus useful aggregation
clinvar-this	MIT	ClinVar submission and XML dump file parsing
mehari	MIT	transcript effect annotation and tx model information
mehari-data-tx	MIT	transcript data builds for mehari
varfish-db-downloader	MIT	Snakemake workflow to download public data
varfish-cli	MIT	command line interface for VarFish REST API
varfish-dev-docs	MIT	documentation for developers
varfish-docker-compose-ng	MIT	setup VarFish with Docker Compose
varfish-server	MIT	VarFish web server
varfish-server-worker	MIT	heavy lifting in varfish-server written in Rust
viguno	Apache	HPO ontology access and full-text search

4.2 Indirect Importance

The following are dependencies of the one in *Direct Importance* maintained by us and housekeeping tools.

name	license	synopsis
biocommons-bioutils-rs	Apache	(partial) port of biocommons/bioutils to Rust
hgvs-rs	Apache	port of biocommons/hgvs to the Rust programming language
rocksdb-utils-lookup	Apache	utility library for using RocksDB as lookup tables
seqrepo-rs	Apache	a port of biocommons/seqrepo to the Rust programming language

4.3 Experimental

The following contain experimental, under development, or unfinished code.

name	license	synopsis
scarus	Apache	automated evaluation of ACMG rules
varfish-clinical-beacon-client	MIT	client for clinical beacons API proof of concept
varfish-wf-validation	MIT	Snakemake workflow for automated validation
varfish-cli-ng	MIT	VarFish CLI based on aws-smithy

4.4 Legacy

The following are legacy repositories.

name	license	synopsis
varfish-course-scripts	MIT	scripts for generating the data used in the VarFish course
varfish-wf-queries	MIT	VarFish (Snakemake) Client Workflow for Querying Snakemake
varfish-docker-compose	MIT	legacy setup VarFish as using Docker Compose
varfish-anno	MIT	convert annotation database files to Var:fish: import format
varfish-installer	MIT	use varfish-docker-compose instead
varfish-annotator	MIT	annotate variants for import into VarFish server
varfish-data-igsr	MIT	repository for building IGSR data sets for use in VarFish

CHAPTER

FIVE

DATASOURCES

This section documents the datasources used as input for the static data available in VarFish.

The download and precomputation is done by the Snakemake workflow in `varfish-db-downloader`. This git repository uses continous integration with a reduced dataset (and some small data that is used from the repository directly, such as a list of curated microdeletion/-duplication regions from the literature) for automated testing. The reduced dataset is downloaded automatically from URLs in a `download_urls.yml` file. Thus, there is full transparency and traceability of the data sources used. Further, a nightly CI job is run to check whether the URLs are still available (but not if the data has changed).

5.1 Data in Repository

The following datasources are used directly from the repository.

Name	License	Synopsis	Source
ACMG SF List v3.1	public domain	Supplementary Findings Gene List of ACMG	PMID:35802134
DOMINO	Public Domain	Score for assessing the probability for a gene to harbour dominant changes	Institute of Molecular and Clinical Ophthalmology Basel; PMID:28985496
Enrichment Regions	Public Domain	Target regions of NGS enrichment kits	UCSC Table Browser
Patho MMS	Public Domain	Curated regions for microdeletion and microduplication scores	PMID:36435749
sHet	N/A (Emailed Author)	Gene haploinsuffiency score	PMID:31004148

5.2 Downloaded Data

The following datasources are downloaded from public internet resources.

Name	License	Synopsis	Source
AlphaMissense	CC BY-NC-SA 4.0	AlphaMissense score	AlphaMissense
CADD Score	free for non-commercial	sequence variant pathogenicity scores	CADD
ClinGen	CC0	clinical gene and genome annotation	ClinGen
Comparative Toxicogenomics Database	free for non-commercial	database of biological named entities	CTD
dbNSFP academic	suitable for academic use	nonsynonymous variant pathogenicity scores	dbNSFP
dbNSFP commercial	suitable for commercial use	nonsynonymous variant pathogenicity scores	dbNSFP
dbSNP	no restrictions	Structural variants from dbSNP	NCBI dbVar
dbVar	no restrictions	Structural variants from dbVar	NCBI dbVar
Database of Genomic Variants (DGV)	no restrictions	Structural variants from DGV	The Centre for Applied Genomics
DECIPHER HI	N/A (Emailed Author)	DECIPHER haploinsufficiency score	PMID:20976243
ENSEMBL	no restriction	ENSEMBL gene/genome annotation and transcripts	ENSEMBL
ExAC CNVs	no restrictions	Copy number variants from ExAC	gnomAD
GenomicsEngland PanelApp	non-commercial	Gene panels with disease associations from Genomics England	GenomicsEngland
gnomAD exomes and genomes	no restrictions	sequence and structural variants, gene constraint scores	gnomAD
GTeX	free	tissue-specific gene expression	GTEx
HelixMtDb	N/A (Emailed Author)	mitochondrial genome frequencies	HelixMtDb
HGNC	CC0	gene information	HGNC
HPO	free	Human Phenotype Ontology	HPO
Human Disease Ontology (DO)	CC0	ontology of human diseases	Disease Ontology
MONDO	CC BY 4.0	Mondo Disease Ontology	OBO Foundry
NCBI ClinVar	no restrictions	clinical variant interpretation	NCBI ClinVar
NCBI Gene	no restrictions	gene information	NCBI Gene
NCBI mim2gene	no restrictions	gene-disease associations	NCBI MedGen
NCBI RefSeq	no restrictions	gene/genome annotation and transcripts	NCBI RefSeq
OMIM titles	restricted	some OMIM disease names are contained in other databases such as HPO	misc. other datasources
ORDO	CC BY 4.0	Orphanet Rare Disease Ontology	BioOntology.org
Orphadata	CC BY 4.0	Orphanet disease-gene associations	Orphadata
rCNV Score	no restrictions	dosage sensitivity score	PMID:35917817
TAD annotation	N/A (Emailed Author)	Topologically Associated Domains annotation	YUE Lab

continues on next page

Table 2 – continued from previous page

Name	License	Synopsis	Source
1000G SV map	Fort Lauderdale Agreement	structural variants from thousand genomes phase 3	IGSR
UCSC assembly-related tracks	no restrictions	assembly-related tracks, genomicSuperDups, rmsk, altSeqLiftOverPsl, fixSeqLiftOverPsl, multiz100way	UCSC Table Browser