EOPF-ZARR GDAL DRIVER: AGILE ROADMAP & USER STORIES

Project: GDAL-ZARR-EOPF Driver
Team: EURAC Earth Observation Institute
Start Date: Q4 2025
Target Release: v1.0 (Q2 2026)
Delivery Model: Agile Scrum (2-week sprints)

---

1. Product Vision & Goals

Vision Statement

Enable seamless, cloud-native access to Sentinel-1 and Sentinel-2/3 EOPF-Zarr datasets through GDAL, eliminating the need to download and extract large SAFE archives while maintaining compatibility with existing geospatial tools (QGIS, Python/rasterio, gdal_translate).

Success Criteria

- ☑ Read complex SLC data with phase information preserved
- ☑ QGIS visualization of SLC/GRD products
- ☑ Python-based preprocessing (calibration, geocoding) via GDAL APIs
- ☑ Direct burst selection without full dataset loading
- ☑ Georeferencing via embedded GCPs
- ☑ Integration with rasterio/xarray workflows

---

2. Release Roadmap

Phase 1: Foundation (Sprints 1-4) | ~8 weeks

Goal: Robust Zarr reading and standard GDAL data access
Target: v0.2 (MVP)

Phase 2: Geospatial Intelligence (Sprints 5-8) | ~8 weeks

Goal: Georeferencing, metadata extraction, preprocessing
Target: v0.5 (Analysis-Ready)

Phase 3: SAR Processing (Sprints 9-12) | ~8 weeks

Goal: Calibration, multilooking, interferometry support
Target: v1.0 (Production)

Phase 4: Optimization (Sprints 13+) | Ongoing

Goal: Performance, cloud integration, advanced features
Target: v1.x+ (Long-term)

---

3. Epic Breakdown & User Stories

PHASE 1: FOUNDATION (V0.2 MVP)

Epic 1.1: Zarr Data Access & Reading

Story 1.1.1: Read Sentinel-1 SLC Bursts as Complex Data

Priority: P0 (Critical)
Sprint: 1
Estimate: 8 points

As a SAR data scientist
I want to open EOPF-Zarr Sentinel-1 SLC products and access individual bursts as complex64 arrays
So that I can process raw phase + amplitude information without extracting SAFE archives

Acceptance Criteria:

- ☐ Driver successfully opens EOPF SLC Zarr products

- [ ] Exposes each burst as separate dataset or band with GDALDataType::GDT_CComplex64
- [ ] `gdalinfo` correctly reports complex data type
- [ ] Can read burst data via GDAL API: `GDALRasterBand::RasterIO()`
- [ ] Handles 27-30 bursts per IW product correctly
- [ ] Tested with real EOPF sample data from zarr.eopf.copernicus.eu

Definition of Done:

- C++ implementation complete
- Unit tests pass (pytest + gtest)
- Documentation updated
- Code reviewed by team lead
- Tested on Linux/Windows/macOS

Tasks:

1. Implement `GDALDataset::Open()` method for Zarr SLC structure
2. Create band objects for each burst with complex64 type
3. Implement `RasterBand::IReadBlock()` for chunked Zarr access
4. Write unit tests for burst reading
5. Create sample test data in `tests/sample_data/`

Story 1.1.2: Read Sentinel-1 GRD Products (Dual Polarization)

Priority: P0 (Critical)
Sprint: 1
Estimate: 5 points

As a satellite data analyst
I want to access Sentinel-1 GRD backscatter data in VV and VH polarizations as separate bands
So that I can perform intensity-based analysis and land-cover classification

Acceptance Criteria:

- [ ] Driver opens EOPF GRD Zarr products
- [ ] Two polarization bands (VV, VH) accessible separately
- [ ] `gdalinfo` shows correct band count and data type (GDT_Float32)
- [ ] Data integrity verified (no NaN/corruption)
- [ ] Band naming convention clear (e.g., "VV", "VH")
- [ ] Compatible with QGIS "Add Raster Layer"

Definition of Done:

- Implementation + tests
- QGIS visualization working
- Documentation with example

Tasks:

1. Implement GRD-specific dataset structure (no bursts)
2. Create dual-band structure from `measurements/VV` and `measurements/VH`
3. Handle data scaling/normalization
4. Test with EOPF sample GRD data

Story 1.1.3: Implement CPM Version Validation

Priority: P1 (High)
Sprint: 2
Estimate: 3 points

As a driver developer
I want to validate that EOPF products were created with compatible CPM version (≥2.6.0)
So that users get clear error messages for incompatible datasets

Acceptance Criteria:

- [ ] Driver reads CPM version from product metadata
- [ ] Rejects CPM < 2.6.0 with informative error message
- [ ] Logs version info for debugging

- [ ] Error message directs user to convert with latest EOPF library
- [ ] Tested with both compatible and incompatible products

Tasks:

1. Parse metadata at `stack/processing_information/cpf_version`
2. Implement version check logic
3. Create error handler with helpful messaging
4. Add test for version validation

Story 1.1.4: Efficient Burst Selection via Open Options

Priority: P1 (High)
Sprint: 2
Estimate: 5 points

As a Python developer
I want to select specific bursts using GDAL open options (e.g., `-oo BURST=IW1_VV_001`)
So that I can load only the area of interest instead of full product

Acceptance Criteria:

- [ ] `-oo BURST=<burst_name>` option works correctly
- [ ] Single burst selection reduces memory footprint
- [ ] Burst name follows EOPF convention: `{subswath}_{polarization}_{burst_id}`
- [ ] Works with Python gdal.OpenEx() API
- [ ] Documented with usage examples
- [ ] If burst not found, error message suggests valid options

Tasks:

1. Implement open options parsing in driver Open() method
2. Add burst name validation
3. Filter dataset to contain only selected burst
4. Write documentation and examples
5. Add integration tests

Story 1.1.5: Zarr Chunk Streaming & Memory Efficiency

Priority: P1 (High)
Sprint: 3
Estimate: 8 points

As a user processing large Sentinel-1 datasets
I want to stream Zarr chunks on-demand without loading entire bursts
So that I can process data larger than available RAM

Acceptance Criteria:

- [ ] Driver implements lazy loading via Zarr chunk boundaries
- [ ] Block size matches Zarr chunk size (no extra copying)
- [ ] Memory usage stays constant regardless of burst size
- [ ] Read performance benchmarked (target: < 2s for 10MB block)
- [ ] Tested with various burst sizes
- [ ] Documentation includes performance notes

Definition of Done:

- Optimized RasterBand::IReadBlock() implementation
- Performance benchmarks documented
- Memory profiling tests pass

Tasks:

1. Profile current memory usage
2. Implement Zarr-native chunk access
3. Add caching strategy (optional read-ahead)
4. Performance benchmark suite
5. Document chunk strategy in README

Story 1.1.6: Metadata Extraction from Zarr Groups

Priority: P1 (High)
Sprint: 3
Estimate: 5 points

As a researcher
I want to access EOPF metadata (orbit info, acquisition parameters, timestamps)
So that I can understand product characteristics without external tools

Acceptance Criteria:

- [ ] Driver extracts and exposes metadata from Zarr groups
- [ ] Orbit direction, orbit number, acquisition mode accessible
- [ ] Timestamp information available
- [ ] Metadata accessible via `GDALDataset::GetMetadataItem()`
- [ ] Organized by metadata domain (e.g., "EOPF", "SENTINEL1")
- [ ] Tested with multiple product types

Tasks:

1. Parse stack-level metadata from Zarr
2. Implement metadata domain structure
3. Add helper functions for common queries
4. Write tests for each metadata type
5. Document metadata domains and keys

Epic 1.2: GDAL Standards Compliance

Story 1.2.1: Proper Driver Registration & Identification

Priority: P0 (Critical)
Sprint: 1
Estimate: 3 points

As a GDAL user
I want to see the EOPF driver in `gdalinfo --formats` output
So that I know the driver is available and can use it

Acceptance Criteria:

- [ ] Driver registers with GDAL driver manager
- [ ] `gdalinfo --formats | grep EOPF` shows driver
- [ ] Short name: "EOPFZARR"
- [ ] Long name: "Earth Observation Processing Framework - Zarr"
- [ ] Capability flags correct (read-only at this stage)
- [ ] Works as plugin and compiled driver

Tasks:

1. Implement `GDALRegister_EOPFZARR()` function
2. Create driver metadata
3. Add to GDAL driver registration
4. Test in both plugin and compiled modes

Story 1.2.2: Support Identify() Method for Auto-Detection

Priority: P0 (Critical)
Sprint: 1
Estimate: 3 points

As a tool developer
I want to GDAL to automatically detect EOPF datasets
So that users don't need to specify driver explicitly

Acceptance Criteria:

- [ ] `Identify()` method correctly recognizes Zarr EOPF structure
- [ ] Doesn't falsely match other Zarr files
- [ ] Fast identification (< 100ms)

- [ ] Works with file paths and URLs
- [ ] Tested against both EOPF and non-EOPF data

Tasks:

1. Implement `Identify()` method
2. Check for EOPF-specific metadata markers
3. Add pattern matching tests
4. Performance verification

Story 1.2.3: Error Handling & Logging

Priority: P1 (High)
Sprint: 2
Estimate: 5 points

As a user troubleshooting issues
I want to receive clear, actionable error messages
So that I can quickly fix problems or report bugs

Acceptance Criteria:

- [ ] All error paths have descriptive messages
- [ ] Logging includes context (file, line, operation)
- [ ] Different severity levels (DEBUG, INFO, WARNING, ERROR)
- [ ] GDAL error callback properly integrated
- [ ] No silent failures
- [ ] Test suite verifies error handling

Tasks:

1. Implement consistent error handling pattern
2. Add logging infrastructure
3. Create error message catalog
4. Test error paths
5. Document troubleshooting guide

Story 1.2.4: PAM (Persistent Auxiliary Metadata) Support

Priority: P2 (Medium)
Sprint: 3
Estimate: 3 points

As a advanced user
I want to store auxiliary metadata in .aux.xml files
So that I can annotate and preserve custom information

Acceptance Criteria:

- [ ] Driver inherits from GDALPamDataset
- [ ] .aux.xml files created/read correctly
- [ ] Works with color tables, nodata values, etc.
- [ ] Compatible with QGIS metadata handling
- [ ] Tested with external overviews (.ovr)

Tasks:

1. Extend class hierarchy to include GDALPamDataset
2. Implement TryLoadXML() call
3. Test .aux.xml creation
4. Test with QGIS

Epic 1.3: Testing & Quality Assurance

Story 1.3.1: Build & Test Infrastructure (CI/CD)

Priority: P0 (Critical)
Sprint: 1
Estimate: 8 points

As a developer
I want to automated testing across platforms
So that I can ensure code quality and prevent regressions

Acceptance Criteria:

- [ ] GitHub Actions CI configured
- [ ] Cross-platform testing (Linux, Windows, macOS)
- [ ] C++ unit tests (gtest) execute in CI
- [ ] Python integration tests (pytest) execute
- [ ] Code coverage reported (target: > 70%)
- [ ] Build artifacts generated for all platforms
- [ ] Failed tests block merge to main

Definition of Done:

- GitHub Actions workflow file created
- All tests passing
- Coverage report published

Tasks:

1. Create GitHub Actions workflow
2. Configure CMake for CI
3. Set up gtest framework
4. Add pytest configuration
5. Configure coverage reporting (codecov)
6. Document CI/CD process

Story 1.3.2: Unit Test Suite for Core Functionality

Priority: P0 (Critical)
Sprint: 1-2
Estimate: 13 points

As a maintainer
I want to comprehensive unit tests for driver functionality
So that regressions are caught early

Acceptance Criteria:

- [ ] Tests for Zarr reading logic
- [ ] Tests for metadata extraction
- [ ] Tests for error handling paths
- [ ] Mock data fixtures available
- [ ] Tests run in < 30 seconds
- [ ] Coverage > 70%
- [ ] All tests documented

Tasks:

1. Create gtest suite structure
2. Write tests for each Story 1.1.x
3. Create mock Zarr datasets
4. Add fixtures for common scenarios
5. Document test patterns

Story 1.3.3: Integration Tests with Real EOPF Data

Priority: P1 (High)
Sprint: 2-3
Estimate: 8 points

As a user
I want to test driver against real EOPF data from zarr.eopf.copernicus.eu
So that I can trust it works in production

Acceptance Criteria:

- [ ] Integration tests download real EOPF samples

- [ ] Tests verify data integrity
- [ ] Network failures handled gracefully
- [ ] Tests cacheable (don't re-download each run)
- [ ] Examples work with real data
- [ ] Documented in test README

Tasks:

1. Set up test data downloading
2. Create integration test suite
3. Add data caching mechanism
4. Document test data setup
5. Add to CI/CD (separate step, optional)

PHASE 2: GEOSPATIAL INTELLIGENCE (V0.5 ANALYSIS-READY)

Epic 2.1: Georeferencing & Spatial Information

Story 2.1.1: GCP Extraction from Embedded Arrays

Priority: P0 (Critical)
Sprint: 4-5
Estimate: 8 points

As a GIS analyst
I want to access ground control points embedded in EOPF products
So that I can georeference data to real-world coordinates

Acceptance Criteria:

- [ ] Driver extracts GCPs from `quality/georeferencing/latitude` and `longitude`
- [ ] GCPs accessible via `GDALDataset::GetGCPs()`
- [ ] GCP projection set correctly (WGS84 by default)
- [ ] QGIS can visualize georeferenced data using GCPs
- [ ] GCP count and accuracy logged
- [ ] Tested with IW and EW mode products

Definition of Done:

- Implementation + integration tests
- QGIS visualization confirmed
- Documentation with examples

Tasks:

1. Parse latitude/longitude arrays from Zarr
2. Implement GetGCPs() method
3. Handle spatial subsetting (GCPs at different resolution)
4. Add interpolation for full-resolution GCPs
5. Test with QGIS "Warp" tool

Story 2.1.2: Slant-Range vs Ground-Range Geometry Selection

Priority: P1 (High)
Sprint: 5
Estimate: 5 points

As a SAR processor
I want to choose between slant-range and ground-range coordinate systems
So that I can work in the geometry appropriate for my analysis

Acceptance Criteria:

- [ ] `-oo GEOMETRY=SLANT_RANGE` or `GROUND_RANGE` option
- [ ] Default behavior documented
- [ ] Slant-range: raw pixel coordinates (azimuth time, range time)
- [ ] Ground-range: via GCP interpolation to lat/lon

- [ ] Consistent across all burst selections
- [ ] Examples show both modes

Tasks:

1. Implement geometry selection logic
2. Add slant-range coordinate exposition
3. Implement ground-range transformation via GCPs
4. Document coordinate systems
5. Create usage examples

Story 2.1.3: CRS (Coordinate Reference System) Handling

Priority: P1 (High)
Sprint: 5
Estimate: 3 points

As a geospatial analyst
I want to dataset CRS properly recognized by GDAL/QGIS
So that my data aligns correctly with other datasets

Acceptance Criteria:

- [ ] WGS84 (EPSG:4326) set as default CRS
- [ ] Accessible via `GetSpatialRef()`
- [ ] QGIS recognizes projection automatically
- [ ] Reprojection via `gdalwarp` works
- [ ] CRS info logged in metadata
- [ ] Handles both geographic and projected data

Tasks:

1. Implement GetSpatialRef() method
2. Create PROJ pipeline for slant-range → geographic
3. Test with gdalwarp
4. Verify QGIS compatibility

Epic 2.2: Calibration & Radiometric Processing

Story 2.2.1: Sigma-Nought (σ°) Calibration

Priority: P0 (Critical)
Sprint: 6
Estimate: 13 points

As a SAR scientist
I want to apply sigma-nought calibration to SLC data
So that I can compare backscatter across products and time

Acceptance Criteria:

- [ ] Calibration LUT read from `quality/calibration/sigma_nought`
- [ ] Interpolation to burst resolution
- [ ] Calibration formula implemented per ESA doc
- [ ] Output in dB scale (-80 to 0 dB typical)
- [ ] Available as virtual band or via `-oo CALIBRATION=SIGMA_NOUGHT`
- [ ] Matches SNAP output within ±0.5 dB
- [ ] Performance: < 5s for full burst

Definition of Done:

- Calibration results validated against SNAP
- Integration tests pass
- Documentation with formula included

Tasks:

1. Parse sigma-nought LUT from Zarr
2. Implement 2D interpolation (scipy algorithm in C++)

3. Implement calibration formula
4. Create virtual band for calibrated data
5. Benchmark and optimize
6. Write validation tests vs SNAP

Story 2.2.2: Gamma & Beta Calibration Options

Priority: P2 (Medium)
Sprint: 6
Estimate: 8 points

As a advanced SAR user
I want to apply alternative calibration (gamma, beta)
So that I can match specific analysis workflows

Acceptance Criteria:

- [ ] Three calibration options: `SIGMA_NOUGHT`, `GAMMA`, `BETA`
- [ ] LUTs read for each calibration type
- [ ] Formula implementation verified
- [ ] Documentation explains differences
- [ ] Examples for each type provided
- [ ] Switch between calibrations via open option

Tasks:

1. Extract gamma and beta LUTs
2. Implement calibration formulas
3. Create selection logic
4. Test all three modes
5. Document calibration types

Story 2.2.3: Thermal Noise Removal (Future - Blocked by EOPF)

Priority: P1 (High)
Sprint: 7 (Dependent on EOPF fix)
Estimate: 8 points

As a noise-conscious analyst
I want to remove thermal noise from SLC data
So that I can improve image quality

Status: BLOCKED - Waiting for EOPF library to fix metadata mapping
Blocker: https://github.com/EOPF-Sample-Service/... (TBD)

Acceptance Criteria:

- [ ] Thermal noise map extracted from product
- [ ] Subtraction applied before intensity computation
- [ ] Optional via open option
- [ ] Significantly improves SNR
- [ ] Tested with real products

Tasks (When unblocked):

1. Wait for EOPF CPM update
2. Implement noise map extraction
3. Develop noise removal algorithm
4. Test SNR improvement
5. Document performance impact

Epic 2.3: Preprocessing Operations

Story 2.3.1: Multilooking (Speckle Reduction)

Priority: P1 (High)
Sprint: 7
Estimate: 8 points

As a SAR analyst
I want to apply multilooking to reduce speckle
So that I can improve image interpretability

Acceptance Criteria:

- [ ] Multilooking via `-oo MULTILOOK=5x2` (range x azimuth)
- [ ] Default: 5 range looks, 2 azimuth looks
- [ ] Non-linear (boxcar, Hanning window) available
- [ ] Output resolution and pixel count updated
- [ ] Performance acceptable (< 30s for burst)
- [ ] Matches SNAP multilooking
- [ ] Virtual band option (lazy evaluation)

Tasks:

1. Implement multilooking kernel
2. Add open option parsing
3. Create virtual band for lazy multilook
4. Optimize using SIMD if possible
5. Test against SNAP output
6. Benchmark performance

Story 2.3.2: Geocoding via Ground Control Points

Priority: P1 (High)
Sprint: 7
Estimate: 13 points

As a user needing georeferenced data
I want to geocode SLC/GRD to lat/lon coordinates
So that I can overlay with other maps

Acceptance Criteria:

- [ ] Ground-range transformation via GCP interpolation
- [ ] Output: latitude/longitude coordinate arrays
- [ ] Resampling method selectable (bilinear, nearest)
- [ ] Works with multilook data
- [ ] Integration with rasterio possible
- [ ] QGIS "Warp" tool works
- [ ] Preserves data integrity

Tasks:

1. Implement GCP-based geocoding transform
2. Create virtual dataset for geocoded view
3. Add resampling options
4. Integrate with multilooking chain
5. Test with QGIS
6. Performance profiling

Story 2.3.3: Output to Standard Formats (GeoTIFF, NetCDF)

Priority: P1 (High)
Sprint: 8
Estimate: 5 points

As a user sharing results
I want to export processed data to standard formats
So that colleagues can use the data in other tools

Acceptance Criteria:

- [ ] Seamless export via `gdal_translate`
- [ ] GeoTIFF with geotransform/GCPs
- [ ] NetCDF option (xarray-compatible)
- [ ] Metadata preserved in output

- [ ] Compression options available
- [ ] Examples for each format

Tasks:

1. Test gdal_translate with all output formats
2. Document recommended export workflows
3. Create conversion examples/scripts
4. Verify metadata transfer

PHASE 3: SAR PROCESSING (V1.0 PRODUCTION)

Epic 3.1: Interferometry Foundations

Story 3.1.1: Coherence Computation Between Bursts

Priority: P1 (High)
Sprint: 9
Estimate: 13 points

As a InSAR researcher
I want to compute coherence between two SLC acquisitions
So that I can assess correlation and deformation sensitivity

Acceptance Criteria:

- [ ] Coherence computation formula implemented
- [ ] Support for multiple-look coherence
- [ ] Output range: 0-1 (0.5 typical for 6-day baseline)
- [ ] Performance: < 60s for full burst pair
- [ ] Virtual band option for memory efficiency
- [ ] Validation against SNAP

Definition of Done:

- Implementation validated vs SNAP
- Performance benchmarks documented
- Test cases with known coherence values

Tasks:

1. Implement coherence formula
2. Handle phase preservation across bursts
3. Implement coherence windowing
4. Optimize for multi-look efficiency
5. Validate vs SNAP output
6. Create test dataset pairs

Story 3.1.2: Burst Co-registration (Future - Planned)

Priority: P2 (Medium)
Sprint: 10
Estimate: 21 points

As a interferometrist
I want to automatically co-register bursts from different acquisitions
So that I can compute interferograms with minimal residual shifts

Status: PLANNED - Coordinate with EOPF team on future products
Blocked By: EOPF planned "co-registered SLC" product

Acceptance Criteria:

- [ ] Support for pre-coregistered products (when available)
- [ ] Optional on-the-fly coregistration via DEM
- [ ] Offset fields computed and exposed
- [ ] Compatible with snap2stamps pipeline

Tasks (When available):

1. Implement DEM-based coregistration
2. Create offset computation
3. Test with real pairs
4. Document coregistration approach

Story 3.1.3: Interferogram Generation

Priority: P1 (High)
Sprint: 10
Estimate: 13 points

As a InSAR analyst
I want to generate interferograms from burst pairs
So that I can detect phase changes for deformation monitoring

Acceptance Criteria:

- [ ] Multi-look interferogram computed
- [ ] Phase wrapped to ±π
- [ ] Coherence co-generated
- [ ] SNR enhanced via filtering
- [ ] Output in standard format (GeoTIFF, NetCDF)
- [ ] Compatible with phase unwrapping tools
- [ ] Matches SNAP/ISCE output

Tasks:

1. Implement interferogram generation
2. Add coherence masking
3. Implement filtering (Goldstein, etc.)
4. Create output formats
5. Validate vs SNAP
6. Performance optimization

Epic 3.2: Time-Series & Advanced Analysis

Story 3.2.1: Support for Sentinel-1 SLC Time Series

Priority: P1 (High)
Sprint: 11
Estimate: 8 points

As a time-series analyst
I want to efficiently load multiple SLC products from different dates
So that I can process long-term deformation monitoring

Acceptance Criteria:

- [ ] Batch loading of multiple EOPF products
- [ ] Time-series metadata management
- [ ] Stack creation automation
- [ ] Compatible with MintPy input format
- [ ] Performance: load 100-burst stack in < 5 min
- [ ] Examples provided

Tasks:

1. Implement time-series dataset structure
2. Create stack management utilities
3. Add metadata organization
4. Benchmark stack loading
5. Document time-series workflows

Story 3.2.2: Persistent Scatterer (PS) InSAR Support (Future)

Priority: P2 (Medium)

Sprint: 12
Estimate: 21 points

As a deformation monitoring expert
I want to integrate with snap2stamps/StaMPS workflows
So that I can detect millimeter-scale ground movement

Status: FUTURE - Requires snap2stamps plugin updates

Acceptance Criteria:

- [ ] EOPF output compatible with snap2stamps
- [ ] Burst selection and preprocessing automated
- [ ] Integration tested with real deformation case study
- [ ] Documentation with example workflow

Tasks:

1. Coordinate with snap2stamps maintainers
2. Create EOPF-to-snap2stamps adapter
3. Test with real deformation product
4. Document integration

Epic 3.3: Performance & Scalability

Story 3.3.1: Performance Optimization & Benchmarking

Priority: P1 (High)
Sprint: 11
Estimate: 13 points

As a infrastructure operator
I want to process large batches efficiently
So that I can meet SLA requirements

Acceptance Criteria:

- [ ] Benchmark: full burst read < 2s
- [ ] Benchmark: calibration < 5s
- [ ] Benchmark: geocoding < 10s
- [ ] Memory usage profiled and optimized
- [ ] Lazy evaluation working throughout
- [ ] Parallel processing options available
- [ ] Report published (see Performance.md)

Tasks:

1. Create benchmark suite
2. Profile memory usage
3. Optimize hot paths (Zarr I/O, calibration)
4. Test parallel processing
5. Document results

Story 3.3.2: Cloud-Native Integration (S3, GCS, HTTP)

Priority: P2 (Medium)
Sprint: 12
Estimate: 13 points

As a cloud infrastructure user
I want to process EOPF data directly from cloud storage
So that I don't need to download to local disk

Acceptance Criteria:

- [ ] Support GDAL VSI URLs (/vsis3/, /vsigs/, /vsicurl/)
- [ ] Lazy loading works with remote files
- [ ] Performance acceptable (< 10% slowdown vs local)
- [ ] Credentials handling documented
- [ ] Examples for AWS S3, Google Cloud Storage

- [ ] Works with STAC catalogs

Tasks:

1. Test with /vsis3/ paths
2. Implement remote-aware caching
3. Create configuration examples
4. Performance comparison
5. Document cloud workflows

PHASE 4: OPTIMIZATION & LONG-TERM (V1.X+)

Epic 4.1: Sentinel-2 & Sentinel-3 Support

Story 4.1.1: Sentinel-2 L2A Product Support

Priority: P2 (Medium)
Sprint: 13
Estimate: 13 points

Sentinel-2 products in EOPF format

Story 4.1.2: Sentinel-3 Product Support

Priority: P2 (Medium)
Sprint: 14
Estimate: 13 points

Sentinel-3 OLCI/SLSTR in EOPF format

Epic 4.2: Advanced SAR Processing

Story 4.2.1: Polarimetric Decompositions (H/A/Alpha)

Priority: P3 (Low)
Sprint: 15+
Estimate: 21 points

Research-grade polarimetric analysis capabilities

Story 4.2.2: Analysis-Ready Data (ARD) Products

Priority: P2 (Medium)
Sprint: TBD
Estimate: 21 points

Support for future EOPF ARD products (when released)

---

4. DEPENDENCY & RISK MANAGEMENT

External Dependencies

Dependency	Risk	Mitigation
EOPF CPM ≥ 2.6.0	Products < 2.6.0 incompatible	Version validation + clear error messages
Zarr library stability	Breaking API changes	Pin versions, monitor releases
GDAL 3.x+	ABI compatibility	Version checking, plugin rebuild process
SNAP integration	SNAP may not support EOPF soon	Provide standalone alternatives
Thermal noise metadata (EOPF lib)	Blocking story 2.2.3	Communicate with EOPF team, track GH issue

Architectural Risks

Risk	Probability	Impact	Mitigation
Complex data type (CComplex64) not well supported in QGIS	Medium	High	Early QGIS testing, workaround via amplitude band
Zarr chunk mismatch with GDAL block size	Low	High	Profile + benchmark, custom chunk size handling
Co-registration complexity	Medium	High	Defer to phase 3, use EOPF pre-coregistered products
Performance regression under load	Low	Medium	Continuous benchmarking, load testing

Team & Capacity

Role	Person	Capacity (points/sprint)	Notes
Lead Developer	TBD	25 pts	C++ expertise, GDAL knowledge required
Contributor 1	TBD	15 pts	Testing, documentation
Contributor 2	TBD	15 pts	SAR domain knowledge
QA/Testing	TBD	10 pts	CI/CD, test automation

---

5. DEFINITION OF DONE

Each Story must satisfy:

- ☑ All acceptance criteria met
- ☑ Unit tests written and passing (target: 70%+ coverage)
- ☑ Integration tests added
- ☑ Code reviewed by at least one team member
- ☑ Documentation/examples updated
- ☑ CI/CD pipeline passes
- ☑ No regressions on existing functionality
- ☑ Closed tickets/issues tracked in GitHub

---

6. SPRINT PLANNING TEMPLATE

Sprint Goal

"Enable [capability] so that [persona] can [value]"

Capacity

Target: 40-50 points (assuming 2 developers, 2-week sprint)

Stories Selected

[List with story, points, assignee]

Dependencies & Blockers

[Identify external dependencies]

Success Metrics

[Specific, measurable outcomes]

---

7. COMMUNICATION & ESCALATION

- **Daily Standup**: 15 min, 10:00 AM CET (async Slack updates if async-first)
- **Sprint Planning**: First Monday of sprint, 2 hours
- **Sprint Review**: Last Friday, 1.5 hours (demo + feedback)

- **Sprint Retro**: Last Friday, 1 hour
- **Release Gate**: Director review before release to production

---

8. GLOSSARY

Term	Definition
EOPF	Earth Observation Processing Framework (ESA)
Zarr	Cloud-native chunked array format
SLC	Single Look Complex - raw phase-preserving data
GRD	Ground Range Detected - intensity data
Burst	Fixed-length segment of SLC product (27-30 per IW)
GCP	Ground Control Point - georeferencing reference
CPM	Common Processing Module (EOPF processing version)
PSI	Persistent Scatterer InSAR (deformation technique)
ARD	Analysis-Ready Data (preprocessed, georeferenced)

APPENDIX: RELEASE CHECKLIST

v0.2 (MVP - End Sprint 4)

- [ ] Basic SLC/GRD reading working
- [ ] Unit tests passing
- [ ] QGIS visualization demo
- [ ] README complete
- [ ] GitHub repo public

v0.5 (Analysis-Ready - End Sprint 8)

- [ ] Calibration implemented
- [ ] Geocoding via GCPs
- [ ] Multilooking
- [ ] Full test suite
- [ ] GitHub release created

v1.0 (Production - End Sprint 12)

- [ ] Coherence computation
- [ ] Interferogram generation
- [ ] Performance benchmarks
- [ ] Comprehensive documentation
- [ ] ESA/GDAL community review