Focus Server – Integrations Map

Integration	Direction	Purpose	Main Code Location	Config/Deps	Notes
REST: POST `/configure`	Inbound	Initialize/configure a job and spawn gRPC sink	`pz/microservices/focus_server/focus_server.py` (`/configure`)	`Config.Focus.*`	Returns `job_id`, `stream_url`, `stream_port`
REST: GET `/channels`	Inbound	Expose available channel range	`focus_server.py` (`/channels`)	`focus_manager.channels`	Simple JSON response
REST: GET `/live_metadata`	Inbound	Live fiber metadata	`focus_server.py` (`/live_metadata`)	`focus_manager.fiber_metadata`	404 if no metadata
REST: GET `/metadata/{job_id}`	Inbound	Retrieve job configuration by id	`focus_server.py` (`/metadata/{job_id}`)	`focus_manager.jobs`	404 if unknown job_id
REST: POST `/recordings_in_time_range`	Inbound	Check recording availability in time range	`focus_server.py` (`/recordings_in_time_range`)	`focus_manager.get_recordings_in_time_range`	Converts to epoch
MQ (RabbitMQ): Prpcast.Info consumer	Inbound	Update live metadata from messages	`pz/microservices/focus_server/metadata_consumer.py`	`aio_pika`, `pzpy.msgbus.AsyncConsumer`	Consumes messages and updates metadata
MQ (RabbitMQ): ingest/queue (live)	Outbound	Live input to analyzer via broker	`focus_server.py` `parse_task_configuration` (live mode)	`get_default_broker_uri()`, `queue_name`, `mq_max_length_bytes`	Ensure backpressure + DLQ in config
gRPC DataStream (Panda)	Outbound	Stream processed data to clients	`focus_server.py` →	`K8SGrpcLauncher` / `Supervi`	Opens per-job gRPC server on

			`focus_manager.grpc_job_launcher.start(...)`	`sorGrpcLauncher` / `BabyThreadLauncher`	port
Baby Analyzer (CLI/SDK)	Outbound	Perform processing pipeline	`baby_analyzer.baby_sitter.format_command(...)`	Flags vary by `view_type`	`out_path=grpc://...` sink
Mongo (Recording Mongo Mapper)	Outbound	Time-range validation, historic mode	`focus_manager.check_time_range_validity` + `RecordingMongoMapper`	`Config.Focus.mongo_mapper_url`	Used when `start_time` / `end_time` provided
Kubernetes Orchestrator	Outbound	Launch/manage gRPC jobs in K8s	`focus_manager.py` (`K8SGrpcLauncher`, `orchestrator_service.get_node_port`)	`Config.Focus.k8s_mode`	NodePort allocated per job
Storage mount (historic in_path)	Outbound	Historic input path	`parse_task_configuration` (`baby_in_path = storage_path`)	`Config.Focus.storage_mount_path`	When running historic mode
CORS/HTTP serving	Inbound infra	Expose the API	`FastAPI` + `CORSMiddleware`		All origins allowed by default

Full Example Commands

- REST – `/configure`

```
curl -sS -X POST "http://<focus-host>:<port>/configure" \
  -H "Content-Type: application/json" \
  -d '{
        "displayTimeAxisDuration": 10,
        "nfftSelection": 2048,
        "displayInfo": {"height": 600},
        "channels": {"min": 1, "max": 64},
        "frequencyRange": {"min": 20, "max": 3000},
        "start_time": null,
        "end_time": null,
        "view_type": 0
      }'
```

- REST – `/channels`

```
curl -sS "http://<focus-host>:<port>/channels"
```

- REST – `/live_metadata`

```
curl -sS "http://<focus-host>:<port>/live_metadata"
```

- REST – `/metadata/{job_id}`

```
curl -sS "http://<focus-host>:<port>/metadata/<JOB_ID>"
```

- REST – `/recordings_in_time_range`

```
curl -sS -X POST "http://<focus-host>:<port>/recordings_in_time_range" \
  -H "Content-Type: application/json" \
  -d '{"start_time": 1727200000, "end_time": 1727203600 }'
```

- gRPC – discovery and call (grpcurl)

```
# If server reflection is enabled:
grpcurl -plaintext <focus-host>:<stream_port> list

# Attempt DataStream call (align service/method with your proto):
grpcurl -plaintext -d '{"stream_id":0}' <focus-host>:<stream_port> \
  pzpy.recording.backends.protocols.panda_datastream.DataStreamService/StreamData
```

- RabbitMQ – queues/consumers

```
# List queues
rabbitmqctl -n <node@host> list_queues name messages consumers

# AMQP reachability
nc -vz <rabbit-host> 5672
```

- Mongo – time-range inspection (mongosh)

```
mongosh "mongodb://<user>:<pass>@<mongo-host>:27017/<db>?authSource=admin" --quiet --eval '
db.recordings.find({
  startTime: {$lte: ISODate("2025-09-25T10:00:00Z")},
  $or: [{endTime: {$gte: ISODate("2025-09-25T10:10:00Z")}}, {endTime: null}]
}).limit(5).pretty()'
```

- Kubernetes – NodePort/pods/logs

```
# Services with nodePort (namespace example)
kubectl -n <ns> get svc -o wide | rg focus | rg grpc

# Find pod and container
kubectl -n <ns> get pods -o wide | rg focus

# Logs
kubectl -n <ns> logs <pod-name> -c <container> --tail=200
```

- Port readiness before gRPC client

```
nc -vz <focus-host> <stream_port>
```

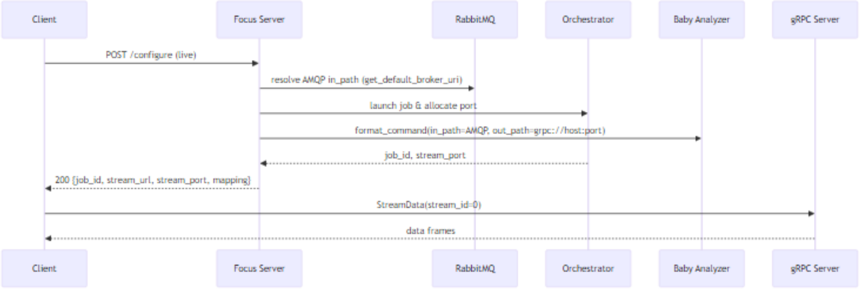
Focus Server flows – detailed, production-grade explanation

Live flow (real-time ingestion → processing → gRPC streaming)

- Preconditions
  - Focus Server REST is reachable; Orchestrator (K8s/Supervisor) healthy.

  - RabbitMQ reachable; queues and permissions configured.
  - Baby Analyzer binaries/SDK available on the processing node(s).
  - Optional: metrics/observability pipeline enabled (Prometheus/OpenTelemetry).
- Request (client → Focus Server)
  - POST /configure without start_time/end_time.
  - Payload fields drive computation: view_type, nfftSelection, displayInfo.height, channels, frequencyRange.
  - The server derives:
    - lines_dt based on view_type and display window
    - internal_nfft and spectrogram overlap (raises internal_nfft if overlap < 0.5, with upper bounds)
    - stream_amount and channel_to_stream_index mapping
  - Validation:
    - Pydantic schema (422 on invalid structure/ranges)
    - Internal guards for overlap, nfft ceiling, and per-view constraints
- Data path selection
  - in_path: AMQP (RabbitMQ) for live data (get_default_broker_uri())
    - queue_name, mq_max_length_bytes, delay_params (when padding/missing chunks logic is enabled)
  - out_path: grpc://host:port (per-job DataStream sink created by orchestrator)
- Orchestration (job lifecycle)
  - Focus Server locks a critical section, generates job_id, chooses stream_port.
  - Orchestrator deploys the job (K8s Pod/Deployment or Supervisor process) and exposes a nodePort/port binding.
  - Baby Analyzer command is formatted with all flags (ROI, spectrogram params, keepalive, etc.) and executed via the launcher.
- Response (Focus Server → client)
  - 200 OK with:
    - job_id, stream_url (host), stream_port
    - channel_to_stream_index, stream_amount, channel_amount
    - frequencies_list, frequencies_amount
    - lines_dt, view_type
- Client consumption
  - Client waits for port readiness (TCP handshake), then opens a gRPC channel to stream frames (StreamData(stream_id)).
  - For multistream jobs, stream_id selection is based on channel_to_stream_index.
- Runtime behavior and backpressure
  - RabbitMQ buffers live input; slow consumers should not lead to unbounded memory growth.
  - Configure bounded queues and DLQ; set retry budgets with jittered backoff to avoid tight loops.
  - Keepalive/health: shorter keepalive for live (e.g., 30s) to fail fast and reconnect.
- Observability and SLOs (recommendations)
  - Metrics: focus_jobs_created_total, focus_jobs_running, focus_jobs_failed_total, focus_queue_depth, focus_config_apply_seconds_bucket
  - gRPC: time_to_first_msg (p95 target), sustained message_count/throughput, error_rate
  - Logs: per-request correlation IDs; classification of errors (client vs system)
- Failures and handling
  - 422: schema/validation errors (bad channels, bad frequency range, view_type mismatch)
  - 503: missing preconditions (validate_required_fields_or_fail)
  - 500: internal parsing/launching error
  - MQ/network hiccups: retries with exponential backoff + jitter, circuit-breaker thresholds
  - Orchestrator failures: job creation rollback, consistent error surfacing to client
- Tuning knobs

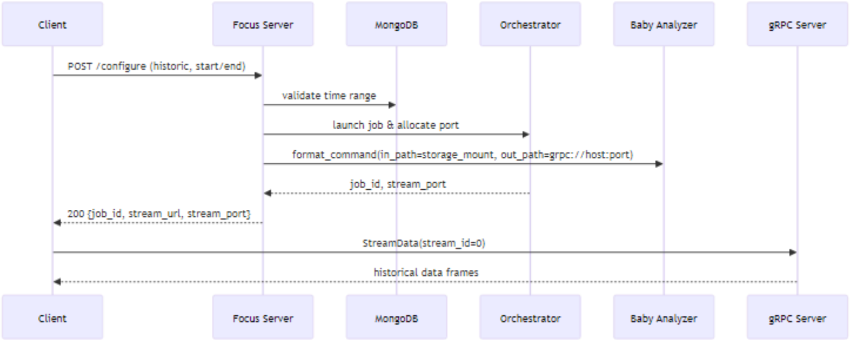
  - mq_max_length_bytes, queue_name (namespacing per job)
  - nfftSelection, displayTimeAxisDuration, displayInfo.height
  - keepalive timeouts, streams_num, resource caps at orchestrator
  - enable/disable padding for missing chunks
- Test checkpoints (live)
  - Invariants by view_type (SINGLECHANNEL ⇒ channel_amount=1, stream_amount=1)
  - lines_dt > 0 for all views; expected frequency list size
  - Mapping size equals channel_amount
  - Port readiness and successful gRPC frames within SLO
  - Queue depth bounded under slow consumer scenarios



Historic flow (time-range playback from storage → gRPC streaming)

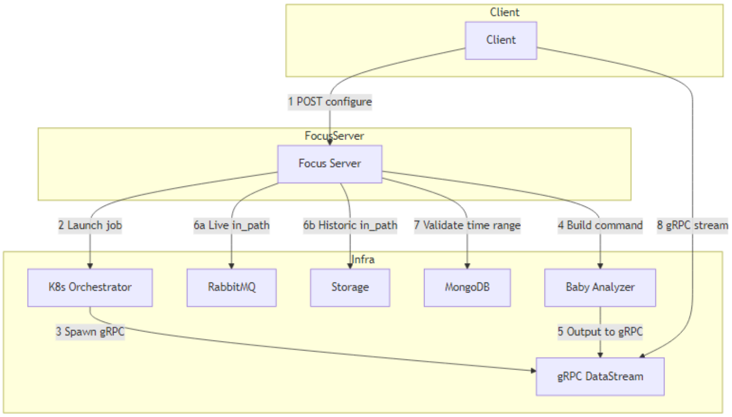
- Preconditions
  - Storage mount with recordings is reachable.
  - MongoDB reachable with recording metadata (RecordingMongoMapper).
  - Orchestrator and Baby Analyzer available.
- Request (client → Focus Server)
  - POST /configure with start_time and end_time (epoch seconds).
  - Server validates time range against Mongo (400 if no recordings exist).
  - Derives same spectrogram and streaming parameters as live, but tuned for historic.
- Data path selection
  - in_path: storage mount (not AMQP)
  - out_path: grpc://host:port (per-job DataStream sink)
  - Additional flags: time_range, mongo_mapper_url; typically longer keepalive (e.g., 180s) to accommodate longer playback windows
- Orchestration (job lifecycle)
  - Same as live: job_id/port allocation, launch via orchestrator, per-job gRPC sink.
- Response (Focus Server → client)
  - 200 OK with job_id, stream_url, stream_port, lines_dt, mapping, etc.
- Client consumption
  - Client connects to gRPC server and reads frames for the requested time window.
  - No dependency on live backpressure patterns; completeness of the selected window is the key.
- Completeness and correctness
  - Ensure all expected frames for the time window are emitted (within defined padding policy).
  - Validate ordering, boundaries, and no duplicate overlaps across the requested ROI/time range.
- Observability and SLOs (recommendations)
  - Metrics: playback duration, frames emitted, gaps encountered (if any), time_to_first_msg (often longer than live)

    - Logs: explicit indication of historic path and time window
    - SLOs: predictable startup (p95 time_to_first_msg), steady throughput without long stalls
- Failures and handling
    - 400 if no recordings in the given interval (correct and explicit)
    - 422 on schema errors
    - 500/503 on internal/infra failures (storage unmounted, orchestrator unavailable)
    - For partial-read errors, return clear error codes and keep logs/audit
- Tuning knobs
    - time windows (start/end), ROI sizing (channels), nfft/overlap
    - buffer sizes, keepalive timeouts, resource caps for batch processing
- Test checkpoints (historic)
    - Mongo validation (precheck queries return at least one recording overlapping the window)
    - gRPC time_to_first_msg within budget
    - Window completeness (no unexpected gaps beyond configured padding)
    - Consistent frequencies_list and mapping with the provided ROI



Side-by-side (what goes first, where, and why)

- Client always starts with POST /configure → Focus Server decides the path:
    - Live: in_path = AMQP → Orchestrator spawns gRPC → Client streams from gRPC
    - Historic: in_path = Storage (validated by Mongo) → Orchestrator spawns gRPC → Client streams from gRPC
- Focus Server is the control-plane for:
    - Parameter derivation (view_type, lines_dt, nfft, overlap)
    - Job orchestration (launch/port assignment)
    - Selecting ingestion source (AMQP vs Storage)
- Baby Analyzer is the data-plane:
    - Consumes from the selected in_path
    - Emits into per-job gRPC sink
- Observability is end-to-end:
    - Focus Server surfaces job state, metrics, and errors
    - Orchestrator and gRPC servers expose runtime/health signals
    - For live, MQ health and DLQ are critical; for historic, storage and Mongo are critical


Client
Client
1 POST configure
FocusServer
Focus Server
2 Launch job
6a Live in_path
6b Historic in_path
7 Validate time range
4 Build command
8 gRPC stream
Infra
K8s Orchestrator
RabbitMQ
Storage
MongoDB
Baby Analyzer
3 Spawn gRPC
5 Output to gRPC
gRPC DataStream