RVTEC 2026 · MIAMI, FL

Working Onboard with OpenRVDAS



Understanding what's going on, keeping it running, and poking at it when it needs it

David Pablo Cohn
david.cohn@openrvdas.org

Talk + demo · ~45 min

01 · WHY YOU ARE HERE

What you will be able to do by the end

01	Know where OpenRVDAS sits on your ship, and find your way around its web console
02	Run and test a single logger from the command line with listen
03	Read a logger or cruise configuration and understand what it does, or should be doing
04	Know which servers keep it running, and where live data live
05	Follow data downstream: from loggers that feed other loggers out to InfluxDB, Grafana and beyond
06	Troubleshoot: recognise what’s broken and find the broken link, whether in OpenRVDAS or downstream

THE WHOLE SYSTEM IN ONE SENTENCE

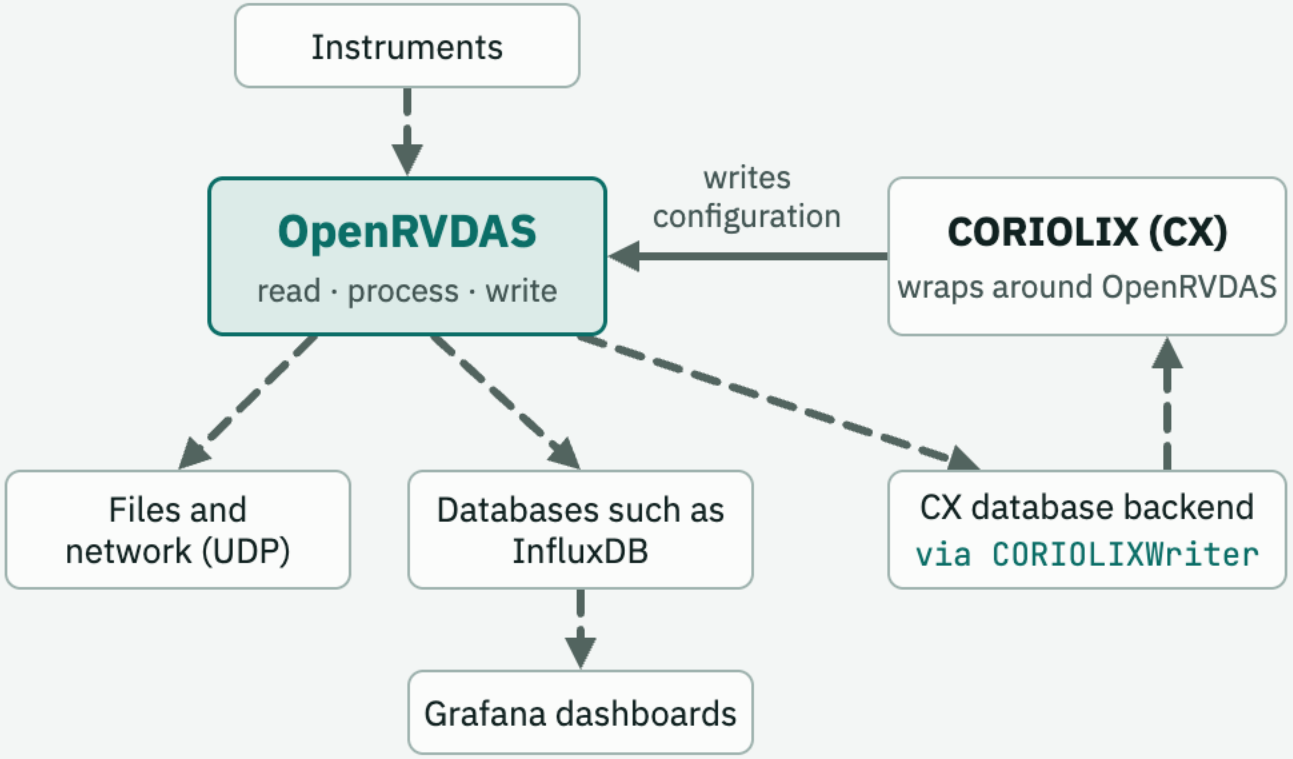
OpenRVDAS:

- **reads** data from sensors,
- **processes**/transforms it as needed,
- **writes** it to one or more destinations

It does this using small, simple components that you "snap together" using a configuration file.

Everything else — the web console, the servers — exists to manage that one idea at scale.

Where OpenRVDAS sits



OpenRVDAS processes the data and writes it to files, the network, and databases such as InfluxDB

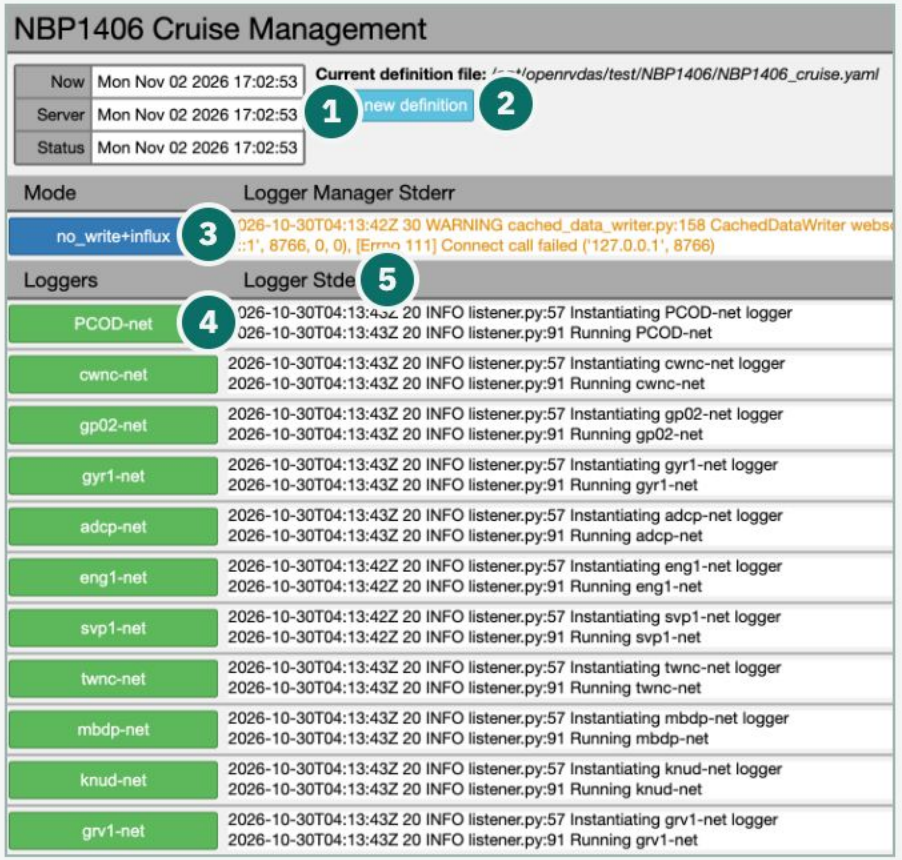
On CX ships it also writes to CX's database backend, using CORIOLIXWriter

Grafana reads from InfluxDB (or another suitable database) to produce dashboards

CORIOLIX wraps around OpenRVDAS:

- manages the instruments and configurations
- keeps OpenRVDAS running
- processes, displays and relays the data that comes out of it

Hands-on: the web console



1 Clocks
Now, Server and Status. If Status stops updating and every button turns red, the console has lost touch with the logger manager.

2 Definition file
The cruise file that's loaded. **Load new definition** picks another.

3 Mode
The blue button. Click it to switch every logger at once.

4 Loggers
One button per logger, labelled with the config it's running. Green running, yellow starting, red failing, grey off. Click one to change its config.

5 Stderr
Latest messages from the logger manager and from each logger. Here: a CachedDataWriter warning worth asking about.

Docs: The Web Console

PART 02

Readers, Transforms, Writers

02 · READERS, TRANSFORMS, WRITERS

Three components (and one script)

Readers

read a record from somewhere

`SerialReader`
`UDPReader`
`LogfileReader`

Transforms

turn a record into another record

`TimestampTransform`
`PrefixTransform`
`ParseTransform`

Writers

write a record somewhere

`LogfileWriter`
`UDPWriter`
`CachedDataWriter`
`InfluxDBWriter`

`listen` the script that glues these pieces together and runs them

This is the core - everything else is just there to help these components work together.

Docs: Component Catalog

02 · READERS, TRANSFORMS, WRITERS DEMO

One logger, from the command line

```
$ listen --serial port=/tmp/tty_s330,baudrate=9600 \
         --transform_timestamp --transform_prefix s330 \
         --write_logfile /var/data/openrvdas/demo/s330 --write_udp 6224
```



Docs: Exploring with listen

02 · READERS, TRANSFORMS, WRITERS DEMO

The same logger, two ways

ON THE COMMAND LINE

```
$ listen \
  --serial port=/tmp/tty_s330,baudrate=9600 \
  --transform_timestamp \
  --transform_prefix s330 \
  --write_logfile /var/data/openrvdas/demo/s330 \
  --write_udp 6224
```

Each flag becomes one entry in the file:

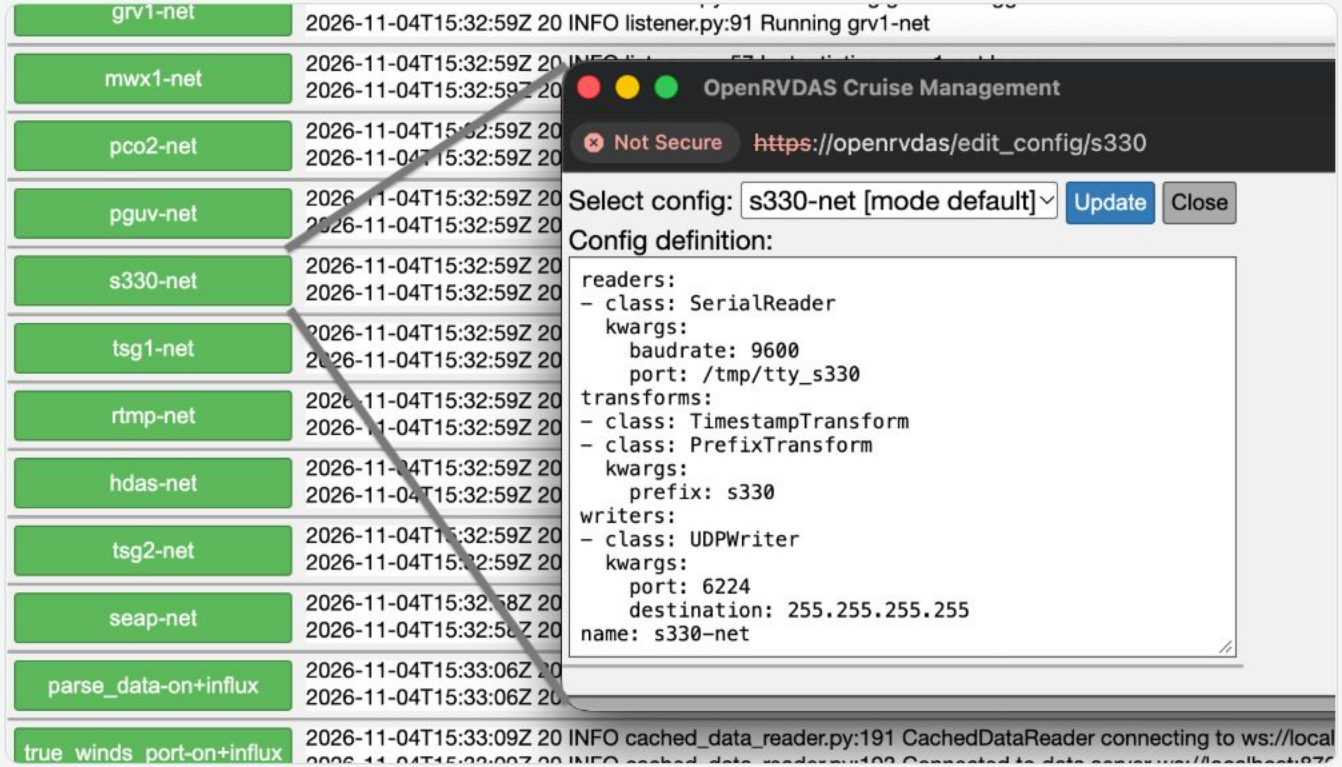
readers transforms writers

AS A YAML FILE

```
$ listen s330.yaml
```

```
# s330.yaml
readers:
- class: SerialReader
  kwargs:
    port: /tmp/tty_s330
    baudrate: 9600
transforms:
- class: TimestampTransform
- class: PrefixTransform
  kwargs:
    prefix: s330
writers:
- class: LogfileWriter
  kwargs:
    filebase: /var/data/openrvdas/demo/s330
- class: UDPWriter
  kwargs:
    port: 6224
```

Under the hood of every logger button



Click a logger button in the console and you see the YAML it is running.

It is the same readers, transforms and writers you just gave listen. This is s330-net: read the serial port, timestamp, prefix, broadcast on UDP 6224.

Each config in the dropdown (s330-off, s330-net, s330-net+file) is just a different block of YAML.

> The console only chooses *which* config runs. The config itself is the logger.

Docs: The Web Console

PART 03

Enough YAML to be dangerous

YAML building blocks

Key-value pairs

owner: Ada: a key, a colon, a space, a value

Lists

one item per line, each starting with a dash and a space: - apple

```
owner: Ada
boxes: 3
color: red
```

```
- apple
- banana
- cherry
```

Docs: YAML Syntax

Composing them: indentation is the structure

A key whose value is a list

the list is indented under the key, or sits right below it: fruits:

A key whose value is key-value pairs

a *mapping* (also called a *dictionary* or *dict*), indented two spaces under its key

A list of mappings

each dash starts a new item; the lines indented under it belong to that item

```yaml
fruits:
- apple
- banana

basket:
  color: red
  size: large

orders:
- fruit: apple
  count: 3
- fruit: cherry
  count: 12
```

Docs: YAML Syntax

Configs are files, written in YAML

readers, transforms, writers

keys whose values are lists

Each list item

a mapping: a class and, if needed, its kwargs

kwargs

itself a mapping: port: /tmp/tty_s330, baudrate: 9600

```
readers:
- class: SerialReader
  kwargs:
    port: /tmp/tty_s330
    baudrate: 9600
transforms:
- class: TimestampTransform
- class: PrefixTransform
  kwargs:
    prefix: s330
writers:
- class: LogfileWriter
  kwargs:
    filebase: /var/data/openrvdas/demo/s330
- class: UDPWriter
  kwargs:
    port: 6224
```

Docs: YAML Syntax

Indentation is structure

```
writers:
- class: UDPWriter
  kwargs:
    port: 6224
```

- Indented under a key means **belongs to** that key
- **Spaces only.** A tab is illegal and the error will not say so clearly
- Two spaces per level; things at the same level line up exactly
- A setting under the wrong parent does not error — it attaches to the wrong thing

SURPRISE YAML is a strict superset of JSON: hand a YAML parser a JSON string, like `{"port": 6224}`, and it will parse it

Docs: YAML Syntax

The values that bite

```
'off': {}              # 'off' is a YAML keyword — quote it
start: '2019-07-01'    # unquoted, this becomes a date object
port: 6224             # a number
flush: true            # a boolean, not the word "true"
prefix: knud           # plain text, no quotes needed
```

> **If a value is meant to be a text string but looks like something else, quote it.**

Docs: YAML Syntax

Validate, don’t guess

```
$ validate_config s330.yaml
```

Run it before you load anything into the live system.

PART 04

Defining one logger

The anatomy of a logger config

Three top-level keys, in pipeline order:

```
readers:
   ...
transforms:
   ...
writers:
   ...
```

Read it top to bottom and you are reading the dataflow.

Docs: Exploring with listen

Every component: class plus kwargs

```
- class: SerialReader
  kwargs:
    port: /tmp/tty_knud
    baudrate: 19200
```

`class`	which component
`kwargs`	the arguments it takes

No kwargs needed when the defaults are fine

EVERY CLASS AND ITS KWARGS

Online	OpenRVDAS Guide → Component Catalog
In the repo	/opt/openrvdas/docs/html/index.html — API docs generated from the source; open in any browser, works offline

Docs: Component Catalog · API docs in the repo (docs/html)

A real configuration, line by line

```
knud-net:
  readers:
  - class: SerialReader
    kwargs:
      port: /tmp/tty_knud
      baudrate: 19200
  transforms:
  - class: TimestampTransform
  - class: PrefixTransform
    kwargs:
      prefix: knud
  writers:
  - class: UDPWriter
    kwargs:
      port: 6224
      destination: 255.255.255.255
```

`knud-net`
the configuration's name

`readers`
a serial port, /tmp/tty_knud, at 19200 baud

`transforms`
timestamp each record, then prefix it with knud

`writers`
broadcast on UDP port 6224

Docs: Writing a Cruise Definition

04 · DEFINING ONE LOGGER

One instrument, several configurations

`knud-off`
do nothing

`knud-net`
read and broadcast

`knud-net+file`
read, broadcast, and write to disk

Same instrument.
Different jobs at different times.

Each configuration is a complete logger definition. Only one runs at a time.

One instrument, several configurations

knud-off
do nothing

knud-net
read and broadcast

knud-net+file
read, broadcast, and write to disk

```
knud-off: {}
```

An empty config: no readers, no transforms, no writers. The logger exists but runs nothing.

One instrument, several configurations

knud-off

do nothing

knud-net

read and broadcast

knud-net+file

read, broadcast, and write to disk

```
knud-net:
  readers:
  - class: SerialReader
    kwargs:
      port: /tmp/tty_knud
      baudrate: 19200
  transforms:
  - class: TimestampTransform
  - class: PrefixTransform
    kwargs:
      prefix: knud
  writers:
  - class: UDPWriter
    kwargs:
      port: 6224
      destination: 255.255.255.255
```

One instrument, several configurations

knud-off

do nothing

knud-net

read and broadcast

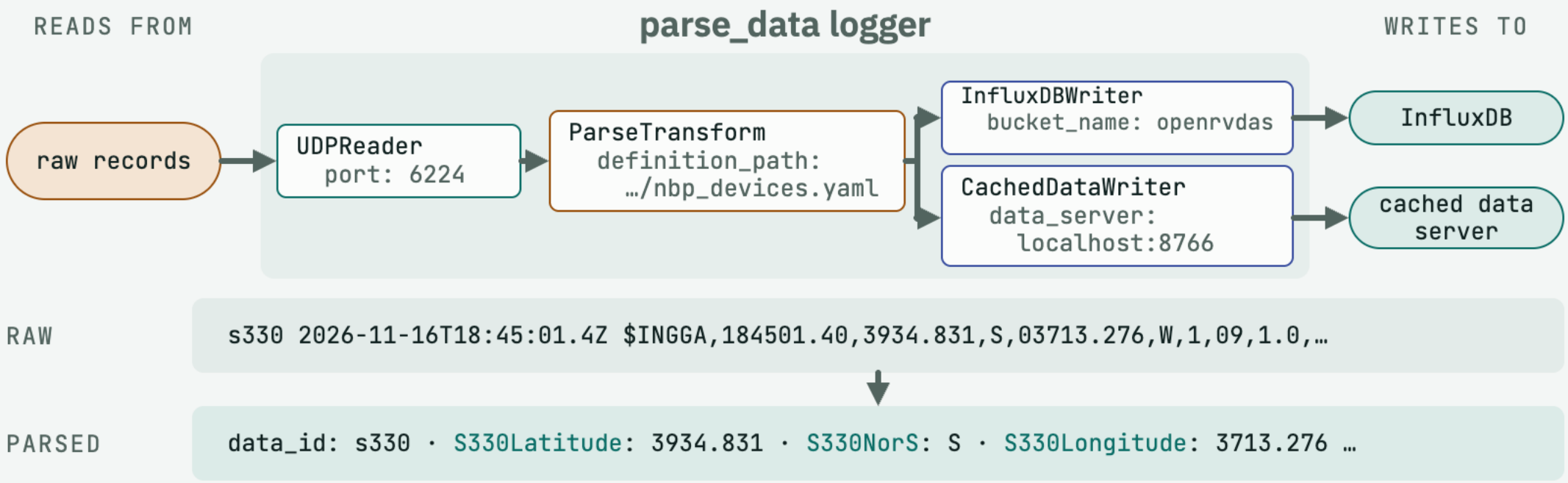
knud-net+file

read, broadcast, and write to disk

```
knud-net+file:
  readers:
  - class: SerialReader
    kwargs:
      port: /tmp/tty_knud
      baudrate: 19200
  transforms:
  - class: TimestampTransform
  - class: PrefixTransform
    kwargs:
      prefix: knud
  writers:
  - class: UDPWriter
    kwargs:
      port: 6224
      destination: 255.255.255.255
  - class: LogfileWriter
    kwargs:
      filebase: /var/data/openrvdas/knud/raw/NBP1406_knud
```

A particularly important logger: parsing

Not every logger reads from an instrument. Raw records are what you archive, but most ships want to QC them, store them in a database or put them on a display. To do that, you define a logger to parse raw records into named values.



Docs: Parsing and Device Types

PART 05

Defining a whole cruise

The cruise definition

One file describes the whole ship. Five top-level keys:

```yaml
cruise:
  id: NBP1406
  start: '2019-07-01'
  end: '2019-12-31'
loggers:       # what loggers exist, and their config names
configs:       # the actual configuration definitions
modes:         # which configs run together
default_mode: 'off'
```

OPTIONAL

includes
variables

Docs: Writing a Cruise Definition

loggers: a convenient fiction

```
loggers:
  knud:
    configs:
    - knud-off
    - knud-net
    - knud-net+file
configs:
  knud-off: {}
  knud-net:
    readers: ...
```

A “logger” is just a named group of configurations.

Any one of them runs straight from the cruise file:

```
$ listen cruise.yaml:knud-net
```

Docs: Writing a Cruise Definition

modes: the operational switch

```yaml
modes:
  'off':
    knud: knud-off
    s330: s330-off
  port:
    knud: knud-off
    s330: s330-net
  eez:
    knud: knud-net
    s330: s330-net
  underway:
    knud: knud-net+file
    s330: s330-net+file

default_mode: 'off'
```

THE SAME THING, AS A GRID

mode	knud	s330
off	knud-off	s330-off
port	knud-off	s330-net
eez	knud-net	s330-net
underway	knud-net+file	s330-net+file

Pick a mode, and every logger switches to the config named for it.

Docs: Writing a Cruise Definition

Keeping the cruise file maintainable

1500+

lines in a fully written-out cruise definition. Three features cut it down:

`includes`	pull in shared logger templates from other files
`variables`	define values once, substitute with <<double_angle_brackets>>
`logger templates`	one pattern, applied to many instruments

```
variables:
  file_root: /var/data/openrvdas
...
filebase: <<file_root>>/<<logger>>/raw/<<cruise>>_<<logger>>
```

OpenRVDAS expands these internally; both styles load the same way.

Docs: Writing a Cruise Definition

Includes, variables and templates in practice

serial_logger_template.yaml (excerpt)

```yaml
logger_templates:
  serial_logger_template:
    configs:
      'off': {}
      net:
        readers:
        - class: SerialReader
          kwargs:
            port: <<serial_port>>
            baudrate: <<baud_rate>>
        transforms:
        - class: PrefixTransform
          kwargs:
            prefix: <<logger>>
        writers:
        - class: UDPWriter
          kwargs:
            port: <<raw_udp_port>>
      net+file: ...
```

cruise.yaml

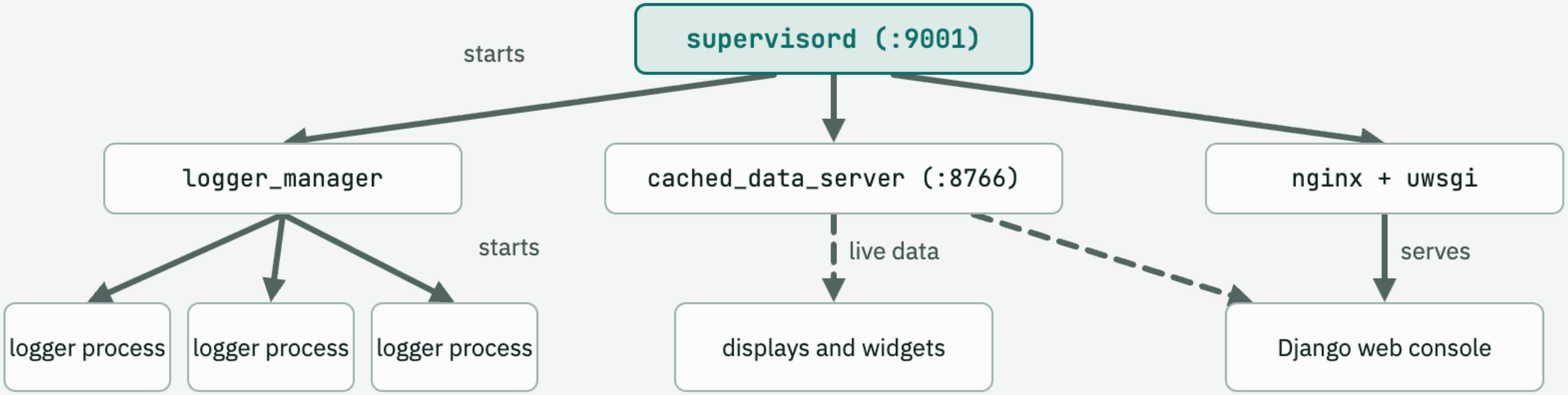
```yaml
includes:
  - contrib/logger_templates/serial_logger_template.yaml
variables:
  cruise: NBP1406
  raw_udp_port: 6224
loggers:
  gyr1:
    logger_template: serial_logger_template
    variables:
      serial_port: /tmp/tty_gyr1
      baud_rate: 4800
  knud:
    logger_template: serial_logger_template
    variables:
      serial_port: /tmp/tty_knud
      baud_rate: 19200
```

Docs: Writing a Cruise Definition

PART 06

The servers that keep it running

The moving parts



Displays blank but files growing? Suspect the cached data server or the parsing logger, not the instrument loggers.

Docs: The Cached Data Server

06 · THE SERVERS THAT KEEP IT RUNNING DEMO

supervisord: the thing that starts things

Web `http://openrvdas:9001`

Command line `supervisorctl`

```
supervisor> status
openrvdas:cached_data_server   RUNNING   pid 14124, uptime 0:00:34
openrvdas:logger_manager       RUNNING   pid 14123, uptime 0:00:34
web:nginx                      RUNNING   pid 13684, uptime 0:01:22
web:uwsgi                      RUNNING   pid 13685, uptime 0:01:22

supervisor> restart openrvdas:cached_data_server
```

Docs: Troubleshooting: Where to Look

logger_manager: the conductor

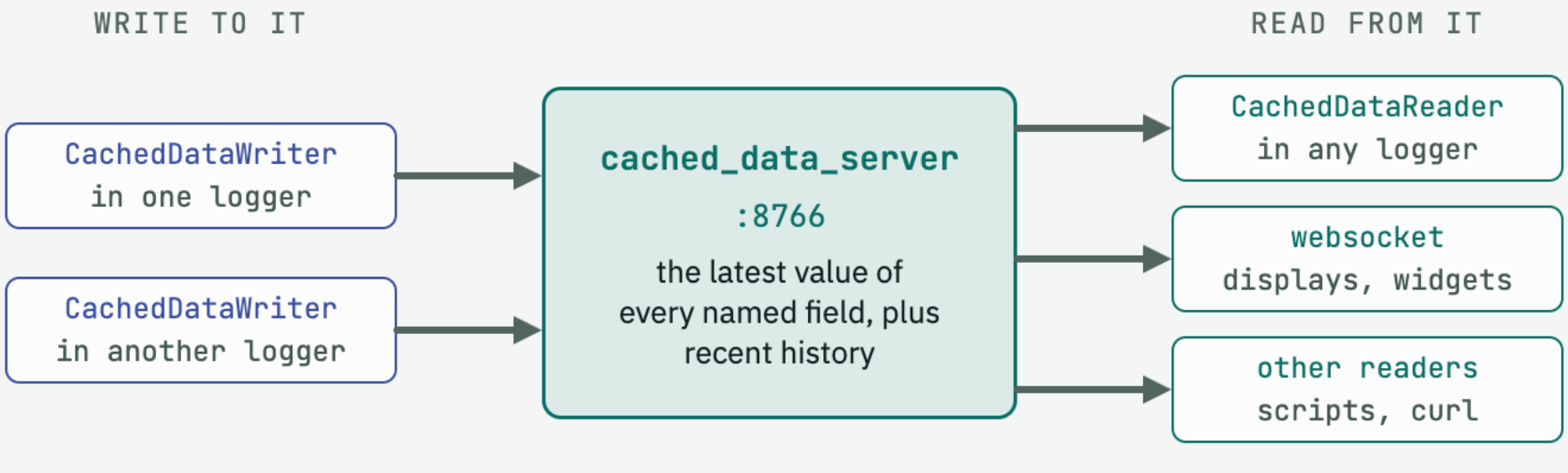
- Loads the cruise definition and remembers it in a database
- Starts and stops one process per active logger configuration
- Watches those processes and restarts failures
- Publishes status the web console reads

HOW SUPERVISORD RUNS IT

```
server/logger_manager.py --database django --no-console \
    --data_server_websocket :8766 -v
```

Docs: Starting and Stopping Loggers · Troubleshooting: Where to Look

The cached data server: a shared notice board



Keeps 24 hours per field by default, backed up to disk

Not the archive. Files and InfluxDB are.

If it’s down, displays go blank, but files keep growing

Loggers can use it to feed each other: one logger’s CachedDataWriter is another logger’s CachedDataReader. More on that next.

Docs: The Cached Data Server

The web console server

Served by nginx and uwsgi, backed by Django

`http://openrvdas`

Log in as the OpenRVDAS user to make changes

Load a cruise definition, change mode, change any single logger. If a definition won't load, the console shows why

Click a logger to see its recent stderr *and* its expanded config, shown as YAML

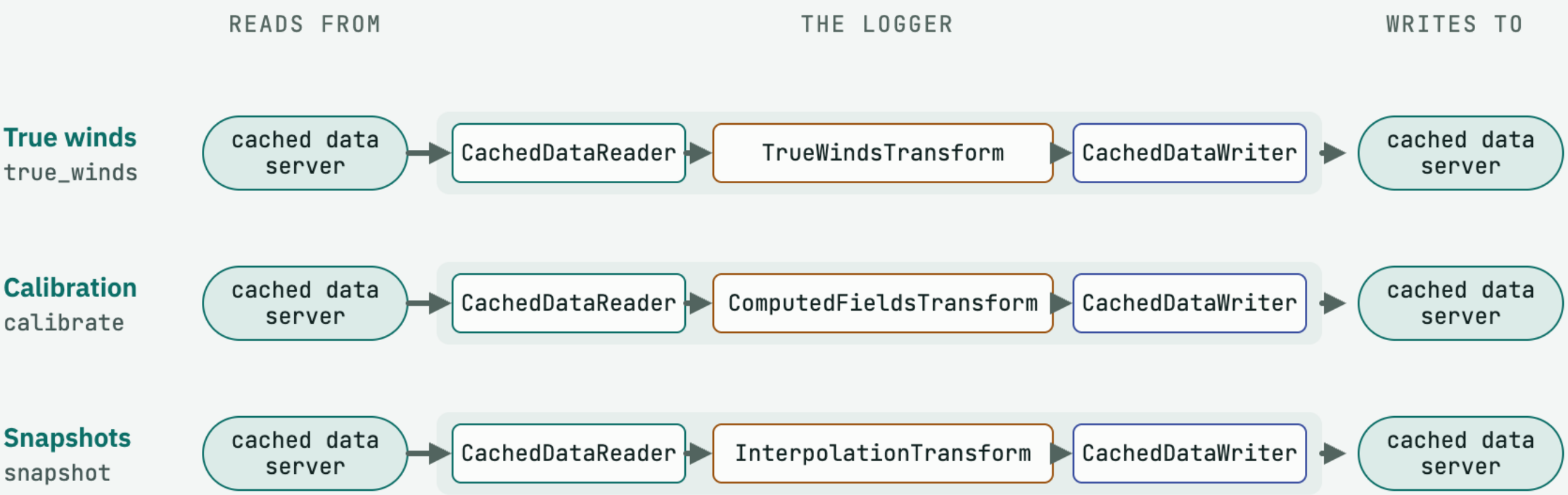
Docs: The Web Console

PART 07

Downstream data processing

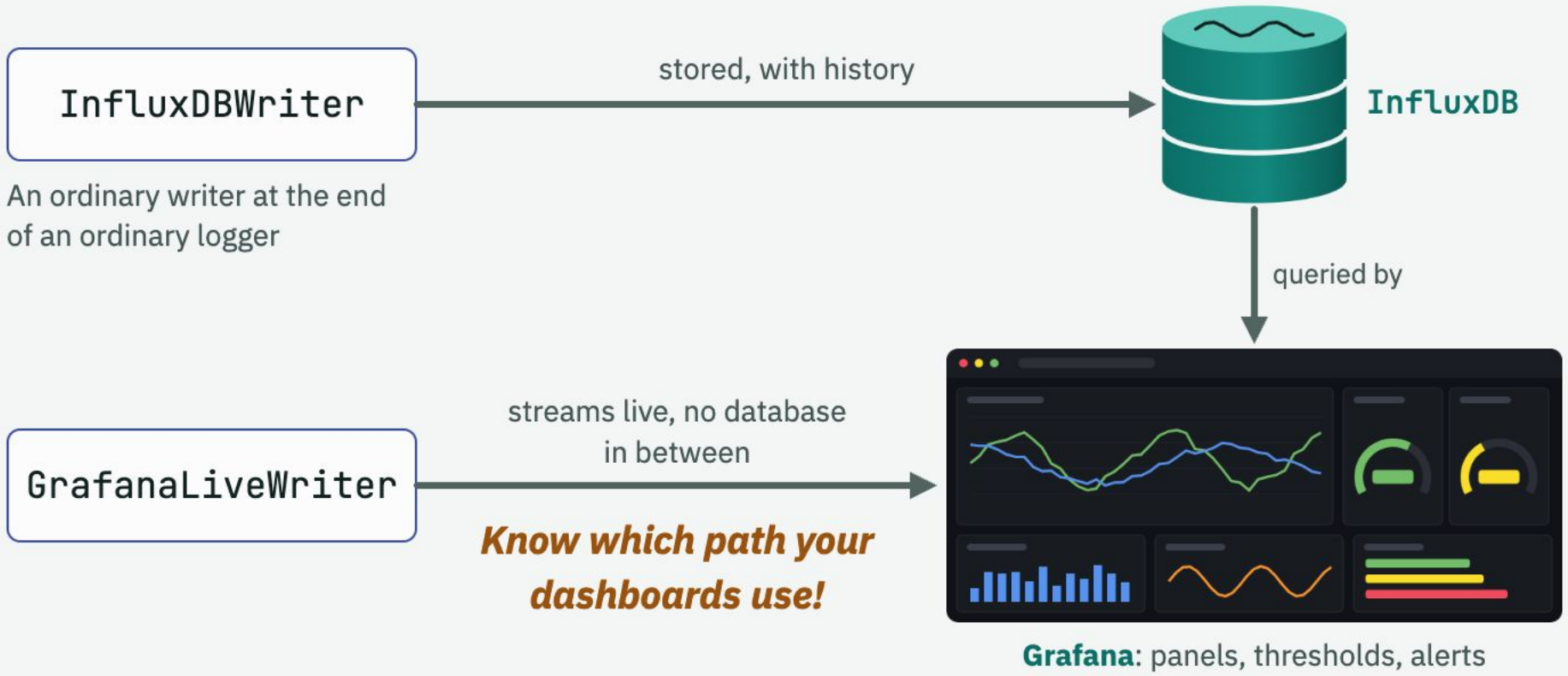
Inside OpenRVDAS, then outside it

Some loggers feed other loggers



Docs: Parsing and Device Types · Derived Data Loggers

Some loggers feed other systems



Docs: InfluxDB · Displays (Grafana, Grafana Live)

Downstream workflow

Store

logfiles (LogfileWriter)
InfluxDB, SQL databases
CORIOLIX on CX ships

Display

Grafana dashboards, Grafana Live
OpenRVDAS web displays
bridge and lab wall screens

QC and alerts

Grafana thresholds and alert rules
QC loggers → Sealog or email alert
stale or flatlined sensor checks

Distribute

UDP or serial feeds to other systems
(NMEA to multibeam, ADCP...)
MQTT, Redis, websockets

Control and events

geofence-triggered mode changes
Sealog event log entries
LoggerManagerWriter commands

Archive and shore

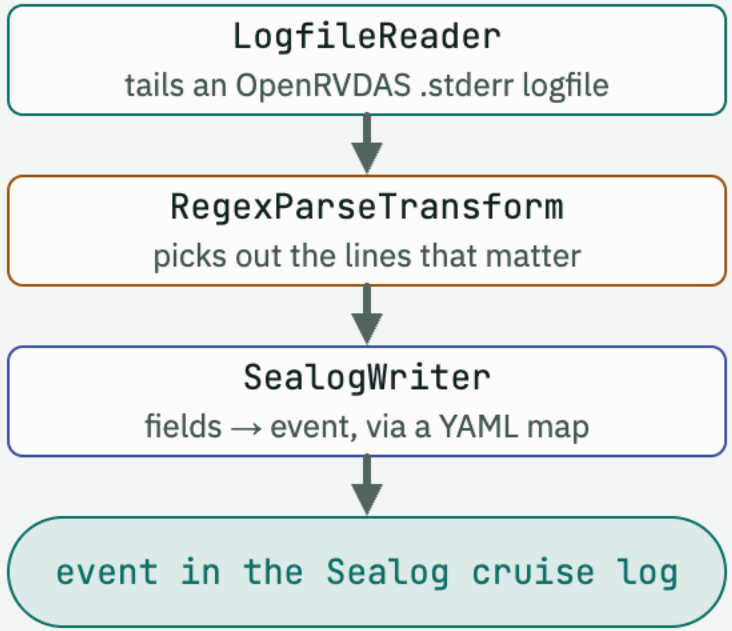
OpenVDM collects logfiles into
the cruise data package
and ship-to-shore transfer

Almost all of it starts the same way: a writer at the end of a logger. Two examples next.

Docs: Storage, Displays and Distribution · Component Catalog

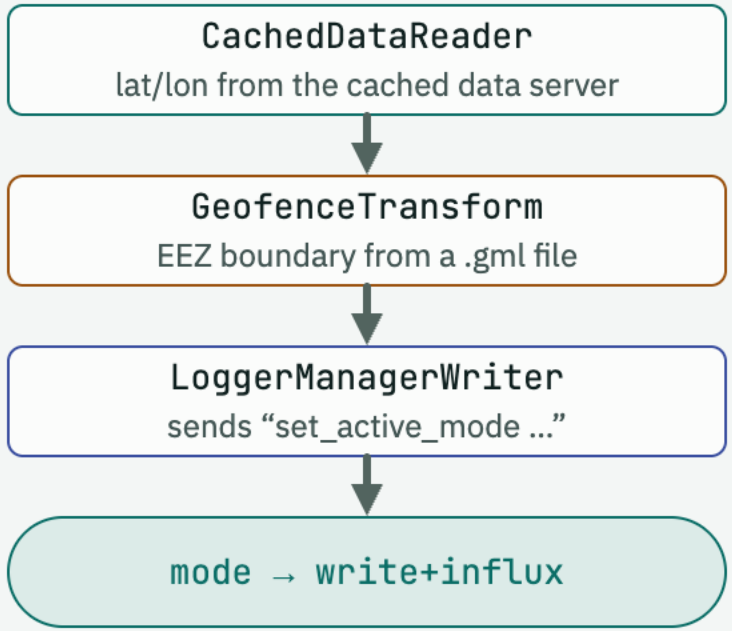
Downstream workflow

Sealog events



OpenRVDAS status and errors become timestamped events

Geofencing



Crossing the EEZ line switches the cruise mode, with nobody at the console

Docs: Component Catalog (LogfileReader, RegexParseTransform, SealogWriter, GeofenceTransform)

Where to get help

Layer	Where to get help
OpenRVDAS, anything	The OpenRVDAS online docs, and the Ocean Data Tools Discord: tinyurl.com/odt-discord-invite
OpenRVDAS → InfluxDB/Grafana	OpenRVDAS Guide, §7: Storage, Displays and Distribution
InfluxDB	InfluxData documentation
Grafana	Grafana documentation and community forum
CX	The CORIOLIX team — next talk
Your dashboards	[Who owns them on your ship?]

Docs: InfluxDB · Displays (Grafana, Grafana Live)

PART 08

Things that can go wrong (and what that looks like)

What each failure looks like

What broke	What you see	First check
Instrument or cable	Logger green, but data stops or turns to garbage	Step 2: data arriving?
Port or permissions	Logger red: permission denied, no device, or port locked	Step 3: stderr
Configuration	Definition won't load, or a logger fails at start	Console, then stderr
Parsing	Raw files fine, but named fields missing from displays	parse_data's stderr
Logger crashing	Button cycles yellow → red; the journal logs each crash	Step 3: stderr, journal
Servers	Console unreachable, or every button red at once	supervisorctl status
The machine	Everything stops (disk full), or timestamps jump (clock)	df -h · timedatectl
Downstream	Files growing, but nothing in InfluxDB, Grafana or CX	InfluxDB Data Explorer

Docs: Common Failure Modes · Where to Look

08 · TROUBLESHOOTING

Follow the record

Instrument · port · reader · transforms · writer · destination · display

A chain fails at exactly one link. Find the last place the data was healthy. The break is immediately after it.

HEALTHY LOOKS LIKE

- ✓ supervisord all RUNNING
- ✓ loggers green in the mode you expect
- ✓ data files growing
- ✓ data on the network
- ✓ displays updating

Do not guess. Do not reboot. Bisect.

08 · TROUBLESHOOTING DRILL

The triage ladder, steps 1–3

From the outside in, least disruptive first:

1 **Console**

Right mode? Right config? Green, red, or flickering?

2 **Is data arriving where it should?**

```
# file writer
tail -f /var/data/openrvdas/knud/raw/NBP1406_knud-2026-09-29
# UDP writer, from another machine (this also tests the network)
listen --udp 6224
```

3 **Read that logger's stderr — start at the first error**

```
# in the console, or:
less /var/log/openrvdas/loggers/knud.stderr
```

08 · TROUBLESHOOTING DRILL

The triage ladder, steps 4–5

4 **Stop the managed logger, then run its config by hand**

```
listen cruise.yaml:knud-net+file -v
```

5 **Still nothing? Read the port directly**

```
listen --serial port=/tmp/tty_knud,baudrate=19200
```

Silence and garbage point to different causes.

If data is arriving at step 2, the logger is fine. The problem is downstream: check InfluxDB, then Grafana or CX.

PORT ALREADY HELD?

The second process fails: “Address already in use” for UDP; for serial, “Serial port … is already locked by another process”.

Find the holder with:

```
sudo lsof /tmp/tty_knud
sudo lsof -i UDP:6224
pgrep -fl logger_manager
```

The last one catches a second logger manager running the same loggers.

08 · TROUBLESHOOTING HANDS-ON

Hands-on: break it, then find it

1	Stop the simulated instruments: sudo supervisorctl stop simulate:simulate_nbp	Predict first: what should go quiet?
2	Watch the console for a minute	Which loggers change colour? Which don't, and why?
3	Walk the triage ladder on one logger	Which rung first shows the break?
4	Start it again: sudo supervisorctl start simulate:simulate_nbp	How long until everything is green?

NEW IN 2.7

The logger manager logs every crash and recovery:

```
grep logger_ /var/log/
openrvdas/logger_manage
r
_journal-*
```

Docs: Isolating Problems with listen · Snapshots and the State Journal

PART 09

Wrap-up and questions

Docs: https://openrvdas.org

APPENDIX

Troubleshooting Reference

Take this with you: files and ports

What	Where
Data files (the archive)	the filebase in each config, e.g. /var/data/openrvdas/knud/raw/NBP1406_knud-<date>
One logger's errors	/var/log/openrvdas/loggers/{logger}.stderr
Server errors	/var/log/openrvdas/logger_manager.stderr, cached_data_server.stderr

Data files belong somewhere permanent. The sample cruise uses /var/tmp/log because it's disposable; set file_root for a real cruise.

console	supervisord	cached data	sample UDP	InfluxDB	Grafana
:80	:9001	:8766	:6224	:8086	:3000

Take this with you: commands and help

```
$ supervisorctl status
$ tail -f /var/data/openrvdas/knud/raw/NBP1406_knud-
<date>
$ less /var/log/openrvdas/loggers/knud.stderr
$ listen cruise.yaml:knud-net+file -v
$ validate_config cruise.yaml
```

HELP

OpenRVDAS docs
InfluxData and Grafana docs
the CORIOLIX team for CX

BEFORE YOU ASK FOR HELP

the config
the first error from the stderr
what changed
when it last worked

APPENDIX GOTCHA

Common failures: instruments and ports

Symptom	Likely cause
Silence on a port that worked yesterday	Cable, power, or the instrument itself
Garbage characters	Wrong baud rate or framing
Permission denied on /dev/tty…	User not in the right group after a rebuild
Device file missing	USB adapter re-enumerated under a different name

Docs: Readers for Your Hardware · Common Failure Modes

APPENDIX GOTCHA

Common failures: port conflicts

Symptom	Likely cause
“Serial port … is already locked by another process” in a logger’s stderr (before 2.7: “Could not exclusively lock port”)	Another logger or test holds the port: sudo lsof /tmp/tty_knud names it; pgrep -fl logger_manager spots a second logger manager
“Address already in use” in a logger’s stderr	Another process already listening on that UDP port; sudo lsof -i UDP:6224 names it
Records missing or chopped, no error anywhere (pre-2.7 installs)	Two processes sharing a serial port without locking; set exclusive: true (the default from 2.7)
Worked in your terminal, fails when managed	Your test still holding the port, or a different user

Docs: Readers for Your Hardware · Common Failure Modes

APPENDIX GOTCHA

Common failures: configuration

Symptom	Likely cause
Cruise definition will not load	The console shows the error; run validate_config for details
A setting seems ignored	Indented under the wrong parent
Unknown class name	Typo, or wrong capitalisation of the component
Unexpected argument	Wrong kwargs name — check the component catalog
A mode is missing a logger	Logger not listed in that mode's mapping
Edited the file, nothing changed	The running system still has the old copy loaded

Docs: YAML Syntax · Common Failure Modes

APPENDIX GOTCHA

Common failures: the machine itself

Symptom	Likely cause
Everything stops at once	Disk full — check the log partition first
Timestamps wrong or jumping	System clock or time sync
Console unreachable	nginx or uwsgi down — check supervisord
Displays blank, files still growing	Cached data server down, or the parsing logger off or failing
Loggers restart forever	A failure the manager keeps retrying — read the stderr

APPENDIX GOTCHA

Common failures: InfluxDB and Grafana

Symptom	Likely cause
InfluxDBWriter can’t connect	InfluxDB down, or wrong URL in the settings
Grafana panel says “No data”	Time range, bucket, field name, or data source
Writes rejected with 401	InfluxDB token changed; update it where OpenRVDAS reads it
Some fields missing in InfluxDB	Not parsed upstream, so no named field to store

Docs: InfluxDB · Displays (Grafana, Grafana Live)