Server Synchronization Plan

BOEM Marine Sensitivity Toolkit — Internal/External Server Sync

Ben Best

2026-02-20

Table of contents

1 Motivation

The Marine Sensitivity Toolkit (MST) is a BOEM-funded project that delivers interactive web applications and documentation for assessing the sensitivity of marine ecosystems to offshore energy development. The project is actively developed by the lead contractor, Ben Best (EcoQuants), who maintains:

- **Interactive apps** (Composite Scores, Species Distribution) built with R Shiny
- **Project documentation** (marinesensitivity.org/docs) built with Quarto

The lead contractor will be out of the United States for an extended period to have his second child in his wife's home country of Italy. During this time, active development will continue on a server **external** to the BOEM network (the existing AWS-hosted development server), while BOEM's goal is to migrate production web services to a server **internal** to the BOEM network.

To bridge this gap, the **internal production server must initiate all synchronization** — pulling software updates (container images), data files, and static web content from the external development server, and pushing log files back for remote debugging. This document describes the architecture, synchronization strategy, implementation scripts, monitoring approach, and government compliance considerations for this arrangement.

2 Architecture Overview

The MST infrastructure spans three tiers: developer laptops, an external cloud server (AWS), and the target internal BOEM server. The following diagram shows the overall software architecture and data flow.

3 Simple

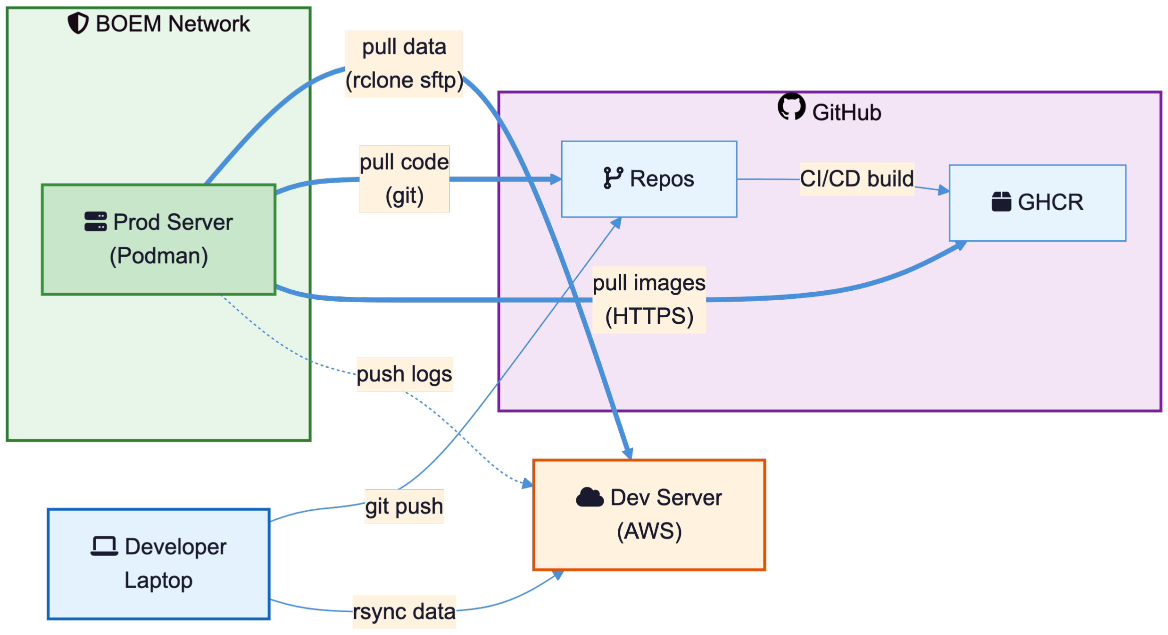

Figure 3.1: Overall server architecture (simplified): developer laptop pushes code to GitHub and data to the dev server; the BOEM production server pulls everything inbound.

4 Detailed

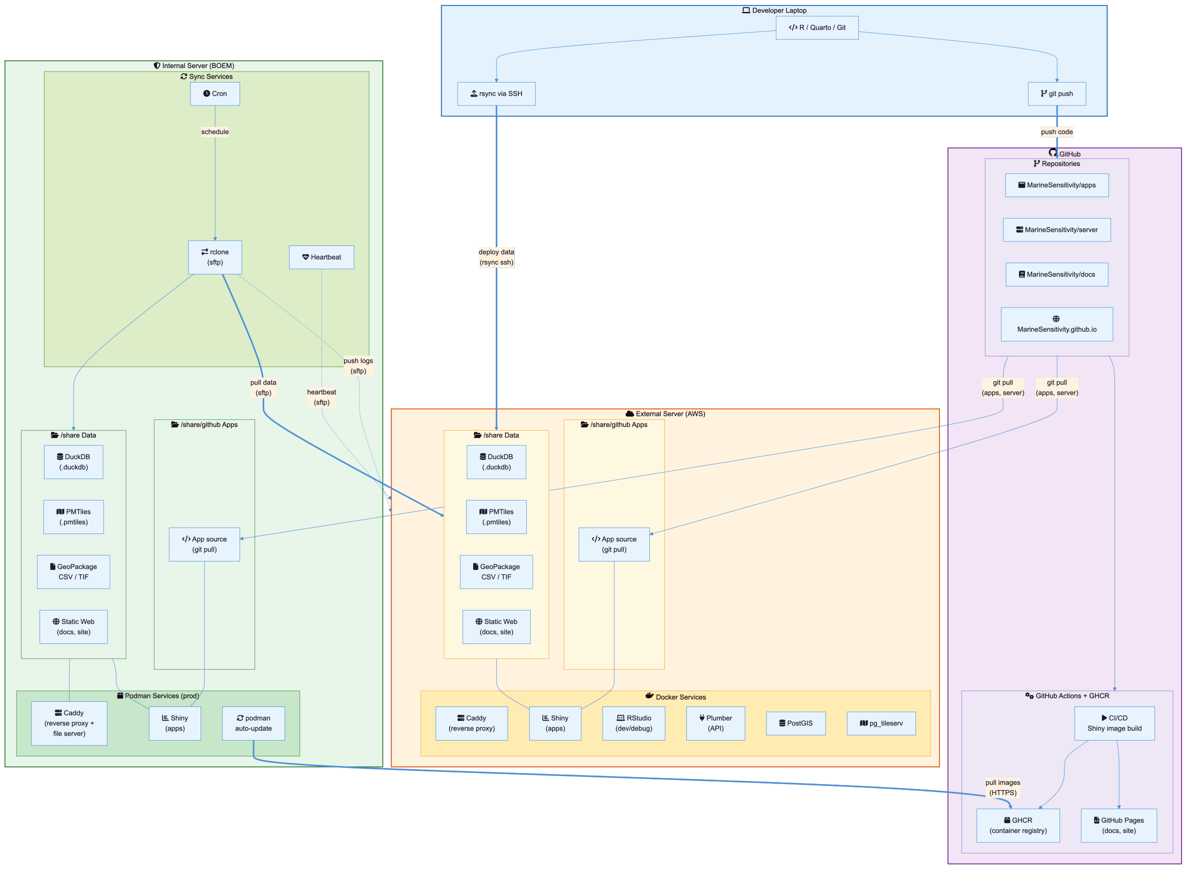

Figure 4.1: Detailed server architecture showing developer laptop, GitHub (repos + container registry), external AWS server (Docker), and internal BOEM server (Podman) with container services and data flows.

Diagram source files are in `diagrams/` — see `diagrams/README.md` for rendering instructions.

4.1 Current state (external only)

The existing external server on AWS (`100.25.173.0`) runs the full development stack defined in `docker-compose.yml`:

Service	Purpose	Port
caddy	reverse proxy + TLS + file server	80/443
rstudio	development IDE + Shiny Server	8787/3838
plumber	R API	8888
postgis	PostgreSQL + PostGIS spatial database	5432
pgadmin	database admin UI	8088
pgbkups	automated database backups	—
tile	pg_tileserv (vector tiles from PostGIS)	7800
tilecache	Varnish cache for tile	6081
titiler	cloud-optimized GeoTIFF tile server	8000
titilecache	Varnish cache for titiler	6082

The current deployment workflow uses two channels:

- **Data files** — the developer runs `deploy_to_server.sh`, which uses `rsync` over SSH to push derived data (DuckDB, GeoPackage, CSV, TIF) from the developer laptop to the external server's `/share/data` directory, then restarts the Shiny container.
- **App source code** — the developer pushes to GitHub, then SSHs into the external server to run `git pull` in `/share/github/MarineSensitivity/apps`. The Shiny Server watches the app directories and picks up changes automatically.

Static web content is currently served via GitHub Pages:

- marinesensitivity.org — project homepage (MarineSensitivity.github.io)
- marinesensitivity.org/docs — project documentation (MarineSensitivity/docs)

4.2 Target state (internal production)

The internal BOEM server runs a minimal production stack defined in `prod/docker-compose.yml`:

Service	Purpose	Port
caddy	reverse proxy + file server + docs	80
shiny	R Shiny applications	3838
auto-update	automatic container image updates (podman)	—

Services **not needed** on production:

- **rstudio** — development only; debugging happens on external server
- **postgis / pgadmin / pgbkups** — database migrated to DuckDB (file-based)
- **plumber** — API either proxied or migrated
- **tile / tilecache** — vector tiles migrated to PMTiles (file-based)
- **titiler / titilecache** — raster tiles served differently or proxied

The key architectural simplification is replacing the PostGIS + pg_tileserv stack with **PMTiles** (a single file format for vector tiles served directly by Caddy as static files) and replacing PostGIS-backed raster queries with **DuckDB** (an embedded analytical database read directly by Shiny apps).

5 Synchronization Overview

All synchronization is **initiated by the internal server** — it pulls updates from the external server and pushes logs back. No inbound connections to the BOEM network are required.

6 Simple

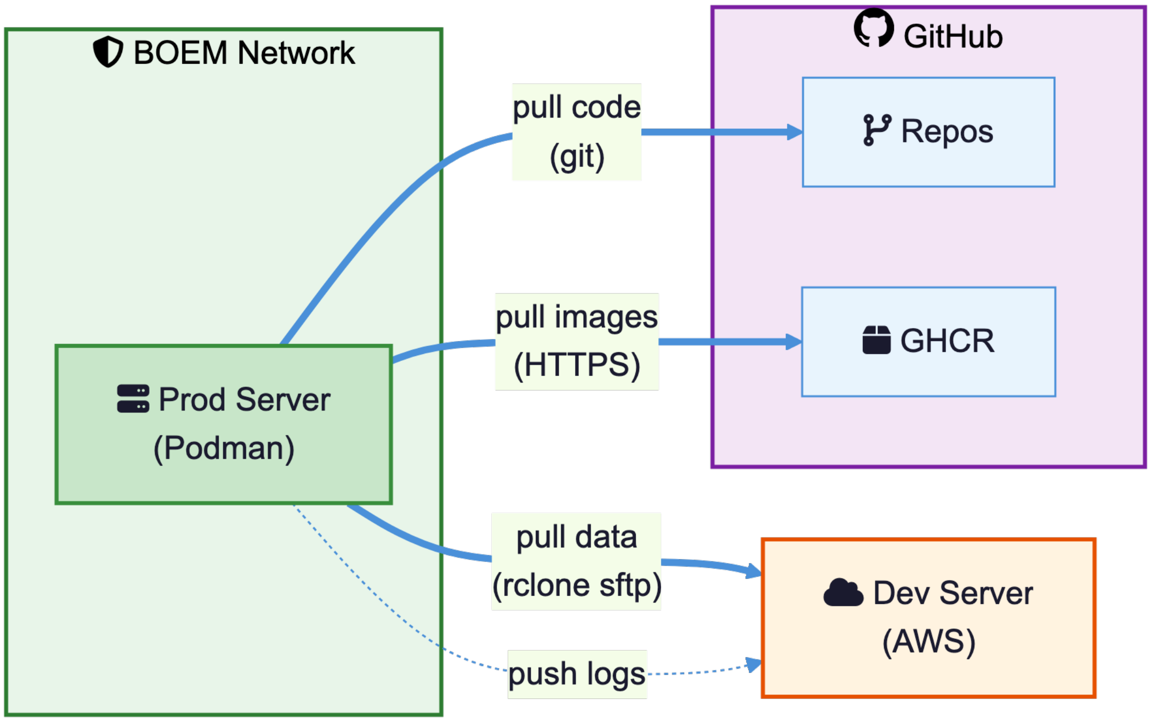

Figure 6.1: Synchronization flow (simplified): the BOEM production server pulls data, code, and images; pushes only logs.

7 Detailed

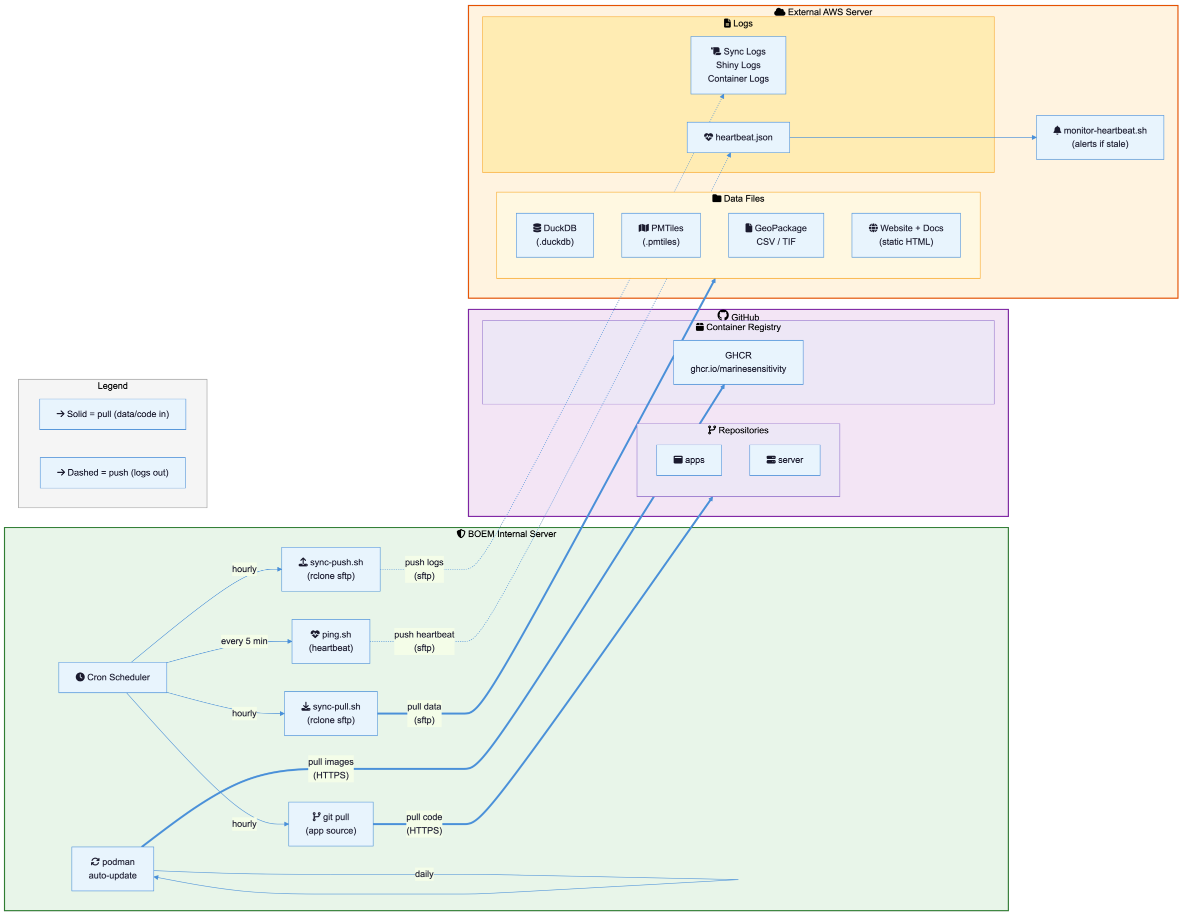

Figure 7.1: Detailed synchronization flow showing cron jobs, rclone scripts, git pull, podman auto-update, heartbeat monitoring, and all data/log targets.

7.1 Sync summary

What	Direction	Source	Tool	Schedule	Protocol
Data files (duckdb, gpkg, csv, tif)	internal pulls	external server	rclone (sftp)	hourly	SSH
PMTiles	internal pulls	external server	rclone (sftp)	hourly	SSH
Static website/docs	internal pulls	external server	rclone (sftp)	hourly	SSH
Shiny app source	internal pulls	GitHub repos	git pull	hourly	HTTPS
Server config	internal pulls	GitHub repos	git pull	hourly	HTTPS
Container images	internal pulls	GitHub GHCR	update-images.sh (cron)	daily	HTTPS
Sync/Shiny/container logs	internal pushes	external server	rclone (sftp)	hourly	SSH
Heartbeat	internal pushes	external server	rclone (sftp)	every 5 min	SSH

8 Implementation

8.1 Prerequisites

On the **internal BOEM server** (RHEL 8.10):

```
# install rclone (not in standard RHEL/EPEL repos - use official script)
curl https://rclone.org/install.sh | sudo bash
# alternatives:
#   sudo yum install -y epel-release && sudo yum install -y rclone  # if EPEL available
#   download binary from https://rclone.org/downloads/ and install manually

# install podman (included in RHEL 8 AppStream)
sudo yum install -y podman
sudo systemctl enable --now podman.socket

# fix UID/GID mapping for rootless podman (if needed)
sudo usermod --add-subuids 100000-165535 --add-subgids 100000-165535 $USER
podman system migrate

# install podman-compose via Python 3.8 (RHEL 8 ships Python 3.6, which is too old)
sudo dnf install -y python38
sudo pip3.8 install podman-compose

# create log directory
sudo mkdir -p /var/log/msens
sudo chown $USER:$USER /var/log/msens

# create data and service directories
sudo mkdir -p /share/data /share/public/www /share/github/MarineSensitivity \
  /share/logs /share/caddy/data /share/caddy/config /share/private
```

! Always use `sudo python3.8 -m podman_compose` on RHEL 8

On RHEL 8, the bare `podman-compose` command may not be on `$PATH` or may use the wrong Python version. Always invoke it as:

```
sudo python3.8 -m podman_compose up -d

# or restart
sudo python3.8 -m podman_compose down && sudo python3.8 -m podman_compose up -d
```

The `sudo` is required for binding to port 80. The `python3.8 -m podman_compose` form ensures the correct Python interpreter is used.

8.2 SSH key setup

Generate an SSH key pair on the internal server for passwordless authentication to the external server:

```
# on internal server
ssh-keygen -t ed25519 -f ~/.ssh/msens_sync -N "" -C "msens-sync@boem"

# set restrictive permissions on private keys
chmod 600 ~/.ssh/msens_sync
chmod 600 ~/.ssh/msens_key_pair.pem  # if using an AWS .pem key

# copy public key to external server (one-time manual step)
ssh-copy-id -i ~/.ssh/msens_sync.pub ubuntu@msens1.marinesensitivity.org
```

If the external server only allows key-based authentication (no password login), you need an **existing** authorized key to bootstrap. Use an AWS `.pem` key pair that is already authorized on the target:

```
# bootstrap: use existing .pem key to copy the new sync key
ssh-copy-id -i ~/.ssh/msens_sync.pub \
  -o "IdentityFile=~/.ssh/msens_key_pair.pem" \
  ubuntu@msens1.marinesensitivity.org
```

After this, the `msens_sync` key is authorized and can be used by rclone and cron jobs without needing the `.pem` key.

8.3 SELinux context for container volumes

RHEL 8 runs SELinux in **enforcing** mode by default. Podman containers cannot read or write host-mounted volumes unless those files carry the `container_file_t` SELinux type label. Without this, containers start but apps fail with "permission denied" errors — even when standard Unix permissions look correct.

Apply the context to the entire `/share` tree:

```
# apply SELinux context to all shared data (recursive)
sudo chcon -R -t container_file_t /share
```

For more targeted application, label specific paths:

```
sudo chcon -R -t container_file_t /share/data
sudo chcon -R -t container_file_t /share/private
sudo chcon -R -t container_file_t /share/public
sudo chcon -R -t container_file_t /share/caddy
sudo chcon -t container_file_t /share/github/MarineSensitivity/server/prod/Caddyfile
```

`container_file_t` vs `svirt_sandbox_file_t`

`container_file_t` is the modern, preferred SELinux type name. The older name `svirt_sandbox_file_t` is an alias and still works, but `container_file_t` is what `chcon` and audit logs use on RHEL 8+.

Re-run after sync creates new files

The `chcon` context does **not** propagate to newly created files. After running `sync-pull.sh` for the first time (or whenever new files appear in `/share`), re-run:

```
sudo chcon -R -t container_file_t /share
```

Verify the context is applied:

```
# check SELinux labels on data files
ls -lZ /share/data

# check for recent SELinux denials
sudo ausearch -m avc -ts recent
```

8.4 rclone configuration

Configure rclone on the internal server to connect to the external server via SFTP:

```
# create rclone config
rclone config

# or write directly:
cat >> ~/.config/rclone/rclone.conf <<'EOF'
[ext_dev]
type = sftp
host = msens1.marinesensitivity.org
user = ubuntu
key_file = /home/<user>/.ssh/msens_sync
shell_type = unix
EOF
```

Test the connection:

```
rclone lsd ext_dev:/share/data/derived/
```

8.5 Sync scripts

The sync scripts live in `prod/` and handle pulling data and pushing logs:

- `prod/sync-pull.sh` — pulls data files, PMTiles, static website, and Shiny apps from the external server
- `prod/sync-push.sh` — pushes sync logs, Shiny logs, and container logs to the external server for remote debugging
- `prod/ping.sh` — sends a heartbeat JSON with system health info every 5 minutes

Make scripts executable:

```
chmod +x prod/sync-pull.sh prod/sync-push.sh prod/ping.sh
```

8.6 Auto-update container images

⚠ `podman auto-update` does not work with podman-compose

Podman's built-in `auto-update` requires containers managed as individual systemd services (via `podman generate systemd`). Containers started with `podman-compose` run inside a **pod** and produce the error:

```
Error: looking up pod's systemd unit: pod has no infra container: no such container
```

The `io.containers.autoupdate` label in `docker-compose.yml` has no effect in this setup.

Instead, the `prod/update-images.sh` script handles automatic image updates. It runs daily via cron and:

1. Pulls the latest `ghcr.io/marinesensitivity/shiny:latest` image
2. Compares the image digest to the currently running version
3. If changed: runs `podman-compose down` + `up -d` to restart with the new image
4. If unchanged: exits without disruption
5. Re-applies SELinux context and logs all actions to `/var/log/msens/update-images.log`

```
# manual one-time run
sudo /share/github/MarineSensitivity/server/prod/update-images.sh

# make script executable (first time)
chmod +x /share/github/MarineSensitivity/server/prod/update-images.sh
```

When the developer pushes a new Shiny image to GitHub Container Registry (GHCR), the daily cron job will detect the update, pull the new image, and restart the container automatically.

8.7 Cron schedule

Add the following to the internal server's crontab (`crontab -e`):

```
# marine sensitivity toolkit sync jobs
# pull data and content from external server (hourly at :05)
5 * * * * /share/github/MarineSensitivity/server/prod/sync-pull.sh

# push logs to external server (hourly at :35)
35 * * * * /share/github/MarineSensitivity/server/prod/sync-push.sh
```

```
# heartbeat ping (every 5 minutes)
*/5 * * * * /share/github/MarineSensitivity/server/prod/ping.sh

# check for new container images and restart if updated (daily at 3:15 AM)
15 3 * * * /share/github/MarineSensitivity/server/prod/update-images.sh
```

8.8 Monitoring and alerting

The heartbeat strategy uses a push model — the internal server pushes a `heartbeat.json` file to the external server every 5 minutes. The external server runs a monitor script that checks the heartbeat age and sends email alerts if it goes stale.

On the **external server**, add to crontab:

```
# check internal server heartbeat (every 10 minutes)
*/10 * * * * /share/github/server/prod/monitor-heartbeat.sh
```

The heartbeat JSON includes:

```json
{
  "timestamp":    "2026-02-19T14:30:00Z",
  "hostname":     "boem-msens-prod",
  "uptime_since": "2026-02-01 08:00:00",
  "disk_free":    "142G",
  "mem_free":     "8.2G",
  "services": {
    "caddy": "running",
    "shiny": "running"
  }
}
```

If no heartbeat arrives for 15 minutes, the monitor sends an email alert with the last known service states.

9 Production Stack

The production container configuration (`prod/docker-compose.yml`, run with `podman-compose`) provides only the essential services:

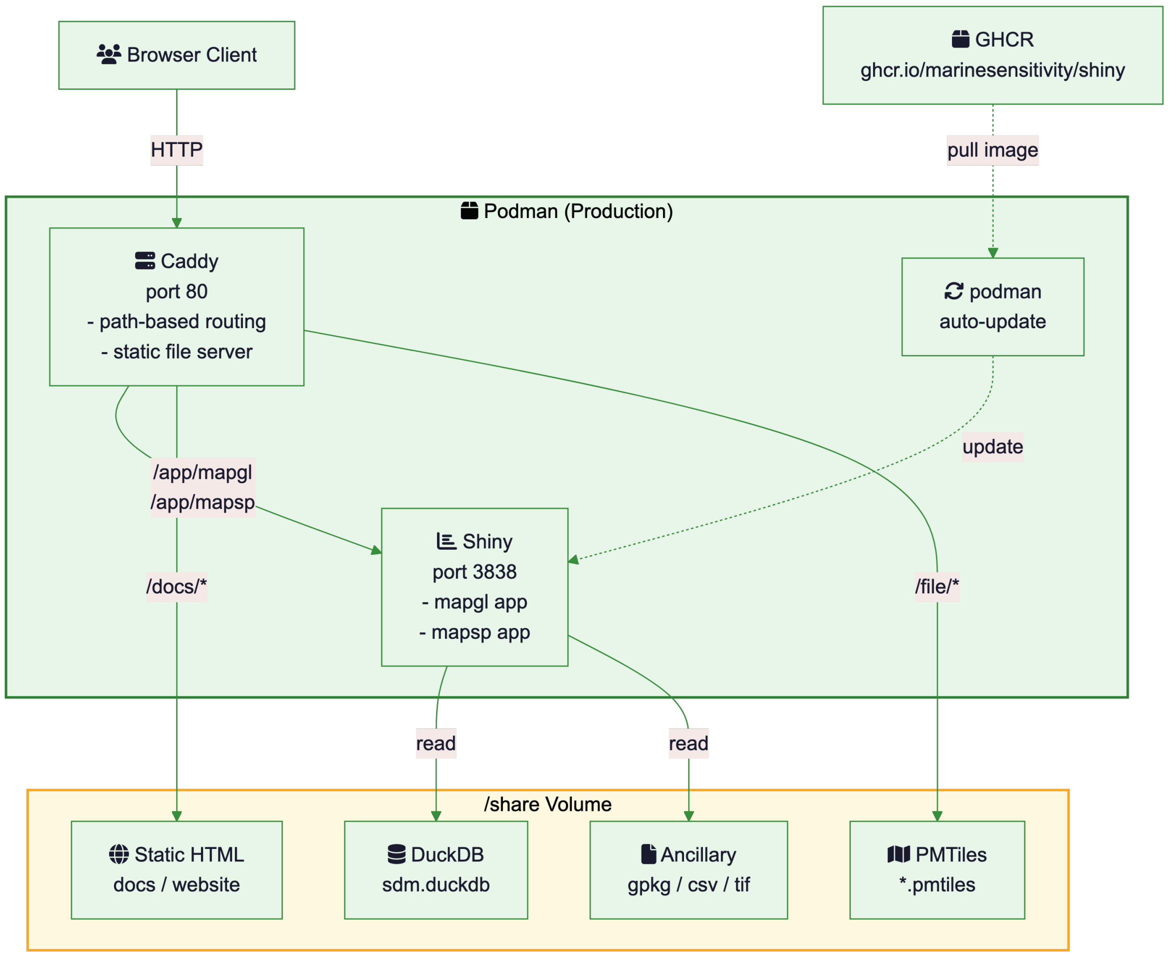

Figure 9.1: Production server Podman services and data flow. Caddy serves static content and proxies Shiny. Data files are mounted as volumes.

9.1 Container image build and distribution

The production Shiny image (`ghcr.io/marinesensitivity/shiny:latest`) is built from `prod/shiny/Dockerfile` and pushed to GitHub Container Registry (GHCR). A GitHub Actions workflow automatically builds and pushes the image when the Dockerfile changes on `main`. Manual builds can also be triggered via `workflow_dispatch`.

When to rebuild: only when R package dependencies change (adding/removing/upgrading packages). App code changes do **not** require an image rebuild — app source is mounted as a volume.

9.1.1 R package version pinning (PPM snapshots)

The `rocker/geospatial` base image configures R to install packages from a Posit Public Package Manager (PPM) **date-pinned snapshot**. This means `install2.r` and `install.packages()` install the versions available on the snapshot date — not the latest CRAN versions.

This can cause problems when a Shiny app depends on a newer package version than what was available on the snapshot date. For example, on `rocker/geospatial:4.4.1` (PPM snapshot ~Oct 2024), `mapgl` installed as **0.1.3** — but the mapgl Shiny app requires **0.4.4+** (which introduced `mapboxgl()` and `maplibre()` functions used by the app).

Fix: upgrade the base image to `rocker/geospatial:4.5.2` (PPM snapshot ~Jan 2026), which includes much newer package versions. As additional insurance, an explicit CRAN override is added in the Dockerfile after the `install2.r` block:

```
# override PPM snapshot for packages that need the very latest CRAN version
RUN Rscript -e "install.packages(c('mapgl', 'dplyr'), repos='https://cran.r-project.org')"
```

This ensures `mapgl` and `dplyr` are always installed from live CRAN regardless of the PPM snapshot date.

9.1.2 First-time GHCR setup

Both `docker` (developer laptop) and `podman` (RHEL production server) produce OCI-compatible images — either can build/push to GHCR.

```
# authenticate to GHCR (docker on laptop, podman on production server)
cat ~'/My Drive/private/GITHUB_PAT_full-scope_2024-12-03_no-expiration.txt' | docker login gh
```

After pushing the first image, set the GHCR package visibility to **Public** in the package settings at `https://github.com/orgs/MarineSensitivity/packages/container/shiny/settings`.

> ❗ GHCR package must be Public for unauthenticated pulls
>
> If the GHCR package is not set to **Public**, `podman pull` on the internal server will fail with an authentication error. Either set the package to Public (recommended for this open-source project) or run `podman login ghcr.io` on the internal server with a GitHub Personal Access Token.

9.1.3 Manual build and push (from developer laptop)

```
# build locally (docker on laptop)
# docker build -t ghcr.io/marinesensitivity/shiny:latest ./prod/shiny
docker build --platform linux/amd64 -t ghcr.io/marinesensitivity/shiny:latest ./prod/shiny

# push to GHCR
docker push ghcr.io/marinesensitivity/shiny:latest
```

9.1.4 Pull on internal server

```
# pull the latest image
podman pull ghcr.io/marinesensitivity/shiny:latest

# restart with the new image
cd /share/github/MarineSensitivity/server/prod
sudo python3.8 -m podman_compose up -d
```

The `update-images.sh` cron job (Section 8.6) automates this — it checks GHCR daily and restarts containers when a newer image is available.

9.1.5 Offline fallback (save / load)

If the internal server cannot reach `ghcr.io`, transfer the image manually:

```
# on a machine with internet access (docker on laptop)
docker pull ghcr.io/marinesensitivity/shiny:latest
docker save ghcr.io/marinesensitivity/shiny:latest | gzip > shiny-latest.tar.gz

# transfer shiny-latest.tar.gz to internal server (e.g., via rclone, scp, USB)

# on internal server (podman)
gunzip -c shiny-latest.tar.gz | podman load
podman-compose up -d
```

9.2 Deployment

Container registry prompt on RHEL

When pulling images with Podman on RHEL 8, you may be prompted to select a registry (e.g., `docker.io`, `quay.io`, `registry.fedoraproject.org`). Select `docker.io/library/caddy:latest` for Caddy. To avoid the prompt, configure `/etc/containers/registries.conf` with `unqualified-search-registries = ["docker.io"]`.

```
# clone repos on internal server
mkdir -p /share/github/MarineSensitivity
cd /share/github/MarineSensitivity
git clone https://github.com/MarineSensitivity/server.git
git clone https://github.com/MarineSensitivity/apps.git

# apply SELinux context to /share (RHEL 8 - see @sec-selinux)
sudo chcon -R -t container_file_t /share

# pull the shiny image from GHCR (first time)
podman pull ghcr.io/marinesensitivity/shiny:latest

# start services
cd server/prod
sudo python3.8 -m podman_compose up -d

# verify
podman ps
curl -s http://localhost/app/mapgl | head -5
```

The Shiny apps are mounted as a volume from the `apps` repo clone at `/share/github/MarineSensitivity/apps` into the Shiny container at `/srv/shiny-server`. This means app code updates only require a `git pull` in the apps repo — no image rebuild needed.

9.2.1 Required data files checklist

The following files must be present on the internal server for the Shiny apps to function correctly:

File / path	Source	Notes
`/share/private/mapbox_token_bdbest.txt`	manual copy	Mapbox API token — **not** in git or rclone sync
`/share/data/big/{version}/sdm.duckdb`	rclone sync	Species distribution model database (~5 GB); currently `v3`
`/share/data/derived/{version}/*.gpkg, *.csv, *.tif`	rclone sync	Derived spatial datasets; currently `v3`
`apps/mapgl/data/taxonomic_hierarchy_worms_*.csv`	git pull	Taxonomy lookup tables (bundled in apps repo)

! Mapbox token is a secret

The file `/share/private/mapbox_token_bdbest.txt` contains a private API key and must be **manually copied** to the server. It is not included in git repos or rclone sync scripts. Without this file, the mapgl app cannot render base map tiles.

i Shiny app_cache write permission

Shiny Server may log a non-fatal warning about not being able to write to `app_cache/` directories inside the mounted app volume. This can be ignored or resolved by creating writable cache directories:

```
mkdir -p /share/github/MarineSensitivity/apps/mapgl/app_cache
chmod 777 /share/github/MarineSensitivity/apps/mapgl/app_cache
```

9.3 Migrating static web content

The project website and documentation currently served by GitHub Pages can be served internally by Caddy as static files:

Content	Current URL	Internal URL	Internal path
Project homepage	marinesensitivity.org	ioemazeud-mar01.mms.doi.net/	/share/public/www/
Project documentation	marinesensitivity.org/docs	ioemazeud-mar01.mms.doi.net/docs/	/share/public/www/docs/
Composite Scores app	app.marinesensitivity.org/mapgl	ioemazeud-mar01.mms.doi.net/app/mapgl	(Shiny container)
Species Distribution app	app.marinesensitivity.org/mapsp	ioemazeud-mar01.mms.doi.net/app/mapsp	(Shiny container)
Public file server	file.marinesensitivity.org	ioemazeud-mar01.mms.doi.net/file/	/share/public/

The `sync-pull.sh` script automatically syncs these from the external server where they are built by GitHub Actions and deployed to `/share/public/www/`.

10 Government Compliance

This section reviews the relevant IT policy statutes and guidelines applicable to BOEM (Bureau of Ocean Energy Management), the Department of the Interior (DOI), and federal .gov requirements, and describes how the proposed synchronization arrangement is acceptable.

10.1 Federal Information Security Modernization Act (FISMA)

FISMA requires federal agencies to implement information security programs. The proposed arrangement is compliant because:

- **All connections are outbound from the BOEM network** — the internal server initiates all SSH/SFTP transfers to the external server. No inbound ports need to be opened on the BOEM firewall.
- **Data classification** — the MST data (species distribution models, sensitivity scores) is **publicly available scientific data** with no Controlled Unclassified Information (CUI) or Personally Identifiable Information (PII). All data will eventually be published openly.
- **Encryption in transit** — all transfers use SSH (SFTP) with Ed25519 key authentication, providing FIPS 140-2 compliant encryption.
- **Principle of least privilege** — the sync account on the external server has read-only access to data directories and write-only access to log directories.

10.2 NIST SP 800-53 (Security and Privacy Controls)

The NIST 800-53 framework applies to DOI systems. Relevant control families:

Control Family	Control	Compliance Approach
AC (Access Control)	AC-3, AC-6	SSH key-based auth, least privilege, no shared credentials
AU (Audit)	AU-2, AU-3	All sync operations logged with timestamps; logs pushed to external for review

Control Family	Control	Compliance Approach
CM (Configuration Mgmt)	CM-2, CM-3	Podman containers provide immutable, versioned configurations; `update-images.sh` only updates from trusted GHCR registry
IA (Identification & Auth)	IA-2, IA-5	Ed25519 SSH keys (no passwords); keys stored with restricted permissions
SC (System & Comms Protection)	SC-8, SC-13	All data in transit encrypted via SSH; TLS for HTTPS
SI (System & Info Integrity)	SI-2, SI-3	Red Hat `yum` updates for OS patching; container images scanned upstream

10.3 DOI Information Security Policy

The DOI Departmental Manual Part 375 and DOI Cybersecurity Program require:

- **Authority to Operate (ATO)** — the internal server should be registered in the DOI System Inventory and receive an ATO. The low-risk nature of the data (public scientific information) supports a streamlined assessment.
- **Continuous monitoring** — the heartbeat mechanism and log synchronization provide continuous visibility into server health without requiring inbound network access.
- **Incident response** — if the external server is compromised, the internal server's pull-only model limits exposure: it only reads data files and container images from trusted sources (GHCR, known SFTP endpoint).

10.4 FedRAMP and Cloud Services

The external development server runs on **AWS**, which holds a FedRAMP authorization at the High impact level. This means:

- The cloud infrastructure meets federal security requirements
- AWS provides the physical and infrastructure security controls
- The MST application layer (containers, data files) operates within this authorized environment

The internal server is **not a cloud service** — it is a BOEM-managed on-premises server. The synchronization only reads from the FedRAMP-authorized AWS environment.

10.5 Trusted Internet Connection (TIC) 3.0

TIC 3.0 guidance from CISA allows for more flexible network architectures than the original TIC model. The proposed arrangement:

- Uses **outbound-only connections** from the BOEM network, consistent with TIC guidance allowing authorized outbound traffic
- Does not require opening inbound ports or creating network exceptions
- SSH/SFTP traffic can be monitored by existing BOEM network security appliances (firewalls, IDS/IPS)

10.6 BOEM-Specific Considerations

- **Data sensitivity**: all MST data is **non-sensitive, publicly releasable** scientific information. Species distribution models, extinction risk scores, and sensitivity metrics are derived from publicly available datasets (AquaMaps, IUCN, NOAA, USFWS).
- **Contractor access**: the lead contractor maintains the external development server under the existing BOEM contract. Development artifacts are delivered to BOEM via the synchronization mechanism described here.
- **Open source**: all MST source code is published on GitHub under MIT license, consistent with federal open-source policies (M-16-21, Federal Source Code Policy).

10.7 Compliance summary

Requirement	Status	Notes
FISMA compliance	Compliant	outbound-only connections, encrypted, public data
NIST 800-53 controls	Addressed	see control mapping above
DOI security policy	Requires ATO	low-risk system; streamlined assessment recommended
FedRAMP (external)	Compliant	AWS holds FedRAMP High authorization
TIC 3.0	Compliant	outbound SSH/HTTPS only

Requirement	Status	Notes
Data classification	Low risk	public scientific data, no CUI/PII
Open source policy	Compliant	MIT-licensed on GitHub

11 Issues

The following items require clarification or action before full deployment:

11.1 1. Operating system compatibility

The internal BOEM server runs **Red Hat Enterprise Linux 8** (RHEL 8) with `yum`-based package management, while the external development server runs **Ubuntu** with `apt`. This should **not be a concern** because:

- All application services run inside containers, which are OS-independent
- The internal server uses **podman** and **podman-compose** (included in RHEL 8); the external server uses Docker — both run OCI-compatible containers
- Only host-level tools differ: `rclone`, `podman/docker`, `cron` — all available on both platforms
- Container images built on Ubuntu will run identically on a RHEL host under podman
- OS updates (via `yum` on RHEL, `apt` on Ubuntu) are independent of the containerized services

Confirmed deployment on **RHEL 8.10** with the following key findings:

- **rclone** is not in the standard RHEL/EPEL repos — must be installed via the official install script (`curl https://rclone.org/install.sh | sudo bash`)
- **podman-compose** on RHEL 8 requires Python 3.8 (`dnf install python38`) and is invoked as `sudo python3.8 -m podman_compose` (not bare `podman-compose`)
- **SELinux** enforcing mode requires `container_file_t` context on mounted volumes — see Section 8.3

Action: Resolved. See Section 8 for updated prerequisites.

11.2 2. Internal web server access (firewall rules)

Firewall rules on the BOEM network may restrict server access to web content ports. To serve the MST applications internally:

- **Port 80 (HTTP)** needs to be open for inbound traffic from BOEM users (or at least the BOEM internal network). Port 443 is not required since we serve HTTP-only internally.
- On RHEL, this may involve `firewalld` (preferred) or `iptables`:

```
# firewalld (recommended on RHEL)
sudo firewall-cmd --permanent --add-service=http
sudo firewall-cmd --reload

# or iptables
sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT
sudo iptables-save
```

- **Internal DNS**: the server hostname `ioemazeudmar01.mms.doi.net` is already assigned by BOEM IT. All services are accessed via subdirectory paths on this hostname (e.g., `ioemazeudmar01.mms.doi.net/app/mapgl`).

Action: confirm port 80 is open for internal traffic on the BOEM network.

11.3 3. External access and domain management

The `marinesensitivity.org` domain is currently managed by the lead contractor through SquareSpace. Questions to resolve:

- **Keep marinesensitivity.org?** The domain is well-established and used in publications. Advantages: continuity, contractor can manage DNS. Disadvantages: not a .gov domain, may need to transition at contract end.
- **Use a BOEM.gov subdomain?** (e.g., `marinesensitivity.boem.gov` or `mst.boem.gov`). Advantages: clearly government-affiliated, managed by BOEM IT. Disadvantages: requires DOI IT approval, potential delays.
- **Hybrid approach?** Use `marinesensitivity.org` externally (public-facing, contractor-managed) and a `.boem.gov` subdomain internally. The internal server would serve the same content under both names.

Action: decide on domain strategy and coordinate with BOEM IT and SquareSpace as needed.

11.4 4. SSH outbound access from BOEM network

The synchronization depends on the internal server being able to make **outbound SSH connections** (port 22) to the external server. If the BOEM network restricts outbound SSH:

- Request a firewall exception for outbound SSH to the specific AWS IP (`100.25.173.0`)
- Alternative: use HTTPS-based sync (e.g., rclone with WebDAV or S3) if SSH is blocked
- Alternative: use an SSH tunnel over HTTPS (port 443) if only HTTPS outbound is allowed

Action: confirm outbound SSH (port 22) is permitted from the internal server to the external server IP.

11.5 5. Container registry access

`podman auto-update` needs **outbound HTTPS access** to GitHub Container Registry (`ghcr.io`) to pull the production Shiny image (`ghcr.io/marinesensitivity/shiny:latest`). If HTTPS to external registries is restricted:

- Request firewall exception for `ghcr.io` (IP ranges published by GitHub)
- Alternative: manually transfer container images via `podman save` / `podman load`:

```
# on a machine with internet access
podman pull ghcr.io/marinesensitivity/shiny:latest
podman save ghcr.io/marinesensitivity/shiny:latest | gzip > shiny-latest.tar.gz

# transfer to internal server (e.g., via rclone sftp, scp, or USB)

# on internal server
gunzip -c shiny-latest.tar.gz | podman load
```

See Section 9.1 for full build and distribution details.

Action: confirm outbound HTTPS to `ghcr.io` is permitted.

11.6 6. Data volume sizing

The MST data footprint should be estimated for disk provisioning on the internal server:

Data type	Estimated size	Growth rate
DuckDB (sdm.duckdb)	~5-10 GB	slow (model updates)
PMTiles	~1-2 GB	slow (boundary updates)
GeoPackage/CSV/TIF	~2-5 GB	slow
Static website/docs	~100 MB	moderate (doc updates)
Container images	~3-5 GB	moderate (app updates)

Data type	Estimated size	Growth rate
Logs	~100 MB	steady
Total	**~15-25 GB**	

⚠ Root partition reached 100% during initial sync

The BOEM server's root partition (`/`) is only **17 GB**. During the first `rclone` sync, the disk filled completely, causing sync failures and preventing new container pulls. The `sync-pull.sh` script syncs only the version specified by the `DATA_VERSION` variable (currently `v3`), excluding older dataset versions.

Action: Partially resolved — sync scripts use the `DATA_VERSION` variable in `prod/sync-pull.sh` (currently set to `v3`). To sync a new version, update that variable. Recommend provisioning a dedicated `/share` volume with **50+ GB** capacity separate from the root partition.

11.7 7. Backup strategy for internal server

While the external server has automated PostgreSQL backups, the internal production server should also have a backup plan:

- DuckDB files can be backed up by the regular sync (authoritative copy is on external)
- Podman volumes (Caddy certificates, config) should be included in any BOEM-managed backup system
- The `/share` directory should be on redundant storage or included in enterprise backup

Action: coordinate with BOEM IT on backup integration.

11.8 8. Caddy TLS certificates on internal network

The production Caddyfile listens on `:80` (HTTP only) as the default. Since the internal server hostname (`ioemazeudmar01.mms.doi.net`) has no public DNS and is not internet-accessible, Caddy cannot use Let's Encrypt for automatic TLS.

This is acceptable for an intranet-only server on the BOEM network. To upgrade to HTTPS in the future:

- Use Caddy's internal TLS with a self-signed certificate (browsers will show a warning)
- Use a BOEM-managed internal CA certificate and configure Caddy's `tls` directive to use it

- Change the Caddyfile to listen on a hostname instead of `:80` and provide the cert/key files

Action: no action needed for initial deployment (HTTP-only is the default). Revisit if BOEM provides an internal CA certificate for HTTPS.

11.9 9. Automatic container image updates

Early planning referenced Watchtower for automatic container image updates. Watchtower requires the Docker socket (`/var/run/docker.sock`), which does **not exist** under Podman.

Podman's built-in `auto-update` was tried next, but it requires containers to be managed as individual systemd services (via `podman generate systemd`). Since our containers are started with `podman-compose` (which creates a pod), `podman auto-update` fails with:

```
Error: looking up pod's systemd unit: pod has no infra container: no such container
```

Status: Resolved. A custom cron script `prod/update-images.sh` pulls the latest image daily, compares digests, and restarts containers only when a new image is available. See Section 8.6.

11.10 10. R package freshness in container images

The `rocker/geospatial` base image pins all R packages to a Posit Public Package Manager (PPM) date snapshot corresponding to the image build date. This means `install2.r` installs the version available on that snapshot date — not the latest CRAN version.

On `rocker/geospatial:4.4.1` (PPM snapshot ~Oct 2024), `mapgl` was installed as **0.1.3**, but the Shiny app requires **0.4.4+**. Upgrading to `rocker/geospatial:4.5.2` (PPM snapshot ~Jan 2026) resolves most version gaps, and an explicit CRAN override in the Dockerfile (Section 9.1.1) ensures critical packages are always current.

Status: Resolved. See Section 9.1.1 for the Dockerfile pattern.

11.11 11. SELinux enforcement on RHEL 8

Red Hat Enterprise Linux 8 runs SELinux in **enforcing** mode by default. SELinux blocks Podman containers from reading or writing host-mounted volumes unless those files carry the `container_file_t` SELinux type label.

Symptoms: containers start but apps fail silently or log "permission denied" errors when accessing `/share` data, even though standard Unix permissions (`ls -l`) look correct.

Status: Resolved. Apply the `container_file_t` context to `/share` before starting containers. See Section 8.3 for commands and verification.