

CHAP-DEVOPS

Developing a model with chap-devops

Build it, run it, test it, evaluate it, with Chap around it

For people who write forecasting models

The loop of a model developer

1. **Build** the model as a service, with chapkit.
2. **Run** it: from its checkout, from an image you built, or from GitHub.
3. **Test** that it trains and forecasts at all.
4. **Evaluate** it with a backtest, on sample data and on real data.
5. **Compare** it with the models that exist.
6. **Publish** it, and add it to the marketplace.

chap-devops does each step with one command, and puts chap-core around the model only when the step needs it.

Two kinds of model

	chapkit model	MLproject model
What it is	a service: a container with an HTTP API	a directory with an <code>MLproject</code> file
How chap uses it	over HTTP	it runs the <code>train</code> and <code>predict</code> commands
Environment	the image	<code>uv_env</code> , <code>renv_env</code> or <code>docker_env</code>
In a deployment	yes, as a service	yes, with <code>chap-devops chapkit mlproject run</code>
With <code>chap-devops chap eval</code>	by id or URL	by GitHub URL or directory, or with <code>--run-config.as-chapkit</code>

The marketplace and the Modeling App use **chapkit models**. This deck is about them. The **MLproject models** deck is about the other kind.

A model in the MLproject format

```
chap-devops chapkit mlproject run my-model      # serve it, with no build
chap-devops chapkit mlproject migrate my-model  # make it a chapkit service
```

- `mlproject run` serves the directory as a chapkit service. In a deployment, it registers with chap-core.
- `mlproject migrate` writes the `Dockerfile` and `main.py` of a chapkit service.
- See the deck **MLproject models with chap-devops**.

A chapkit model service

One container that listens on port 8000:

Route	What it does
GET /health	says that the service runs
GET /api/v1/info	the model: id, name, period type, covariates
/api/v1/configs	the model configurations
/api/v1/artifacts	the trained models and the predictions
POST /api/v1/ml/\$train	trains a model from a config and data
POST /api/v1/ml/\$predict	forecasts with a trained model
POST /api/v1/ml/\$validate	checks a request before it runs
GET /api/v1/ml/\$generate-sample-data	data in the shape the model declares

Start a new model

```
chap-devops models new my-model
chap-devops models new my-r-model --template shell-r
```

Template	What it gives you
fn-py	chapkit service, Python, the model in <code>main.py</code> (the default)
shell-py , shell-r	chapkit service, <code>train</code> and <code>predict</code> scripts in <code>scripts/</code>
shell-r-tidyverse , shell-r-inla	the same, with tidyverse and fable, or INLA
mlproject-py , mlproject-r , ...	an <code>MLproject</code> file and scripts, no chapkit code

Docker builds it, so Python is not necessary. `mlproject-py` builds FROM `chapkit-py-data-cli`. The author comes from git `user.name` and `user.email`.

Registration with chap-core

A model service registers **itself** with chap-core when it starts:

Variable	What it says
<code>SERVICEKIT_ORCHESTRATOR_URL</code>	where to register: chap-core's <code>/v2/services/\$register</code>
<code>SERVICEKIT_HOST</code>	the host chap-core calls the model back at
<code>SERVICEKIT_PORT</code>	the port chap-core calls the model back at
<code>SERVICEKIT_REGISTRATION_KEY</code>	the shared secret, when chap-core asks for one

- In a deployment, chap-devops sets these for you.
- Registration is a heartbeat: `chap-devops status` shows `LAST PING`.
- The service id must be the id the service registers with (`MLServiceInfo.id`).

Run a model: four sources

```
chap-devops run chapkit_simple_multistep_model           # a marketplace id
chap-devops run https://github.com/chap-models/chapkit_ghr_model # a GitHub repository
chap-devops run ghcr.io/my-org/my_model:sha-1eb8cf1      # a published image
chap-devops run my-model:dev                             # an image built here
```

```
running chapkit_simple_multistep_model on http://localhost:5001 (answered in 4s)
```

- No deployment directory: the model runs in a `chap-devops run` group.
- `chap-devops run -a` shows the model's log. Ctrl-C stops the model.
- `chap-devops ps` lists the models, `chap-devops stop ID` stops one.
- The first run of a model downloads its image: 1 GB for a Python model, 6 to 7 GB for an R model.

Your model from its checkout

chap-devops runs chap-core. You run the model with `uv run`, a debugger or hot reload:

```
chap-devops init mychap --models none && cd mychap && chap-devops up
```

```
export SERVICEKIT_ORCHESTRATOR_URL='http://localhost:8700/v2/services/$register'  
export SERVICEKIT_HOST=host.docker.internal  
export SERVICEKIT_PORT=8001  
uv run uvicorn main:app --host 0.0.0.0 --port 8001
```

It worked when `chap-devops status` lists the model with a recent LAST PING. Its state is `unmanaged`, because the deployment did not start it.

Why these values

- `localhost:8700`: the model runs on your machine, so `localhost` and chap-core's API port reach chap-core.
- `host.docker.internal`: chap-core runs in a container, where `localhost` is the container. This name is your machine. chap-devops maps it on Linux too.
- `--host 0.0.0.0`: a call from a container does not arrive on the loopback.
- **Single quotes** keep the shell from reading `$register` as a variable.
- With a registration key: `export SERVICEKIT_REGISTRATION_KEY` with the value that `chap-devops auth show --reveal` prints.

An image you built yourself

```
docker build --platform linux/amd64 -t my-model:dev .  
chap-devops run my-model:dev
```

In a deployment, with chap-core:

```
chap-devops models add my-model:dev  
chap-devops up  
chap-devops models test my_model
```

- A local image is never pulled: the overlay says `pull_policy: never`.
- Build again with the same tag, then `chap-devops restart`, to use the new build.

chap-devops models add : from GitHub or ghcr

```
chap-devops models add https://github.com/my-org/chapkit_dengue_model
chap-devops models add ghcr.io/my-org/chapkit_dengue_model:sha-1eb8cf1
```

Source	Pin	chap-devops update
a repository URL	the newest published sha- build of the default branch	moves the pin
an image reference	exactly the tag or digest you named	leaves it alone

`models add` asks GitHub for the branch and its commits, and ghcr for the newest commit with a published `sha-<short commit>` build: the tag that the publish workflow writes.

What `models add` resolves

```
added chapkit_dengue_model (chapkit-dengue-model)
  source      https://github.com/my-org/chapkit_dengue_model
  image       ghcr.io/my-org/chapkit_dengue_model:sha-b1d6c31
  pin         sha-b1d6c31 (commit b1d6c31)
  follows     main (`chap-devops update` moves the pin)
  data dir    /work/data (from the image config)
  user        10001:10001 (from a docker probe)
```

Each line says where its value came from, because no reviewed catalogue entry gave it.

The flags of `models add`

Flag	What it sets	Without it
<code>--id</code>	the model id; <code>auto</code> picks a free one. Not <code>persistence</code> , <code>global_median</code> or <code>seasonal_median</code>	the name in snake_case
<code>--service-id</code>	the compose service and DNS name	the id with hyphens
<code>--name</code>	the display name	the repository or image name
<code>--port N auto</code> , <code>--bind</code>	a host port, and its address	no host port
<code>--data-dir</code>	the writable data directory	<code><WorkingDir>/data</code> from the image
<code>--user</code>	the user the container runs as	the image's own user, or a probe
<code>--runtime-amd64</code>	the image runs on amd64 only	asked from ghcr

The marketplace entry

A model in the marketplace is a YAML file in the registry. An excerpt of `models/auto_arima_chapkit.yaml`:

```
id: auto_arima_chapkit
service_id: auto-arima-chapkit
source:
  repository: https://github.com/chap-models/auto_arima_chapkit
  image: ghcr.io/chap-models/auto_arima_chapkit
compatibility:
  period_types: [monthly]
  max_prediction_periods: 12
channels:
  stable: 1.0.2
```

Versions and channels

Each version pins one commit and the image built from it:

```
versions:  
  - version: 1.0.2  
    commit: e9047d405050869a8918ba55ddca95d271961a00  
    image_tag: sha-e9047d4  
    chapkit: ">=2.2.0,<3"  
    status: verified
```

```
chap-devops models enable chapkit_simple_multistep_model # stable  
chap-devops models enable chapkit_simple_multistep_model --channel latest  
chap-devops models enable chapkit_simple_multistep_model --version 0.1.1 # a pin
```

The overlay chap-devops writes for a model

```
chapkit-simple-multistep-model:  
  image: ghcr.io/chap-models/chapkit_simple_multistep_model:sha-10b2bc7  
  platform: linux/amd64  
  expose: ["8000"]  
  read_only: true  
  security_opt: [no-new-privileges:true]  
  cap_drop: [ALL]  
  user: 1001:1001  
  volumes: [tmpfs on /tmp, ck_chapkit_simple_multistep_model_data on /app/data]
```

The real file has one setting on each line, and `.env` can override the image tag. This is a short form.

What the overlay checks for you

- **Platform:** marketplace images are amd64 only, so the overlay pins `linux/amd64`. On Apple silicon they run under emulation.
- **The user:** chap-devops reads it from the image config, not from a guess. A root image gets no `user:` line.
- **Read-only:** a read-only file system, `no-new-privileges`, no capabilities, a 2 GB tmpfs on `/tmp`.
- **The data directory** is a named volume, the only writable path apart from `/tmp`.

A model that writes outside its data directory fails here first, not in production.

The init container

Before the model starts, a busybox container chowns the data volume:

```
chapkit-simple-multistep-model-init:  
  image: busybox:1.38  
  command: ["sh", "-c", "chown -R 1001:1001 /app/data"]  
  user: "0:0"
```

- A volume that docker makes belongs to root. The model does not run as root.
- The chown is recursive, so a volume that an older build owned is corrected too.
- It runs for every model, a root image too.

Test: does it work at all?

```
chap-devops models test my_model          # chapkit test, in the model's own container
chap-devops models test my_model --backtest # the whole way, through chap-core
chap-devops models test --all              # every model of the deployment
```

- `chap-devops status` can be all green while a model cannot predict at all: registration is a heartbeat.
- `chap-devops models test` makes the model do the work.
- Exit code: not zero only when a model **failed**. A model that chap-devops could not test is a skip.

The model level

chap-devops runs chapkit's own test in the model's container:

```
docker compose exec -T <service> chapkit test --url http://127.0.0.1:8000 --timeout 300
```

chapkit reads the config schema, makes a config, generates data for the declared covariates and period type, then validates, trains and predicts.

- **It proves:** the image runs, the user can write, the runtime has its libraries, and the model makes a prediction.
- **It does not touch** chap-core. It answers in seconds.

The backtest level

--backtest goes the way the Modeling App goes:

1. Sample data from the model: `$generate-sample-data`, 36 periods, 5 org units.
2. `POST /v1/analytics/make-dataset`, and a wait for the job.
3. `POST /v1/analytics/create-backtest`: 3 periods, 2 splits, stride 1.
4. The scores from the backtest's `aggregateMetrics`.

```
chapkit-simple-multistep-model    pass      13s    crps 5.4  mae 7.3  rmse 9.0  
configured model: monthly_climate (id 14)
```

The line under the row names the configured model that the backtest used. Scores on generated data say that the pipeline works, not that the model is good.

Flags of `chap-devops models test`

Flag	What it does
<code>--all</code>	every model in <code>.chap-devops/models.yaml</code>
<code>--backtest</code>	the backtest level, through chap-core
<code>--config NAME</code>	the configured model to backtest, by its variant name
<code>--seed N</code>	the same generated data on each run, so two runs compare
<code>--timeout SECONDS</code>	300 at the model level, 900 with <code>--backtest</code>
<code>--keep</code>	keep what the test made, and say how to delete it
<code>-vv</code>	the whole output of <code>chapkit test</code> , and each request
<code>--json</code>	one object for each model, with all the metrics

Without `--keep`, chap-devops deletes the configs, artifacts, dataset and backtest that the test made, and nothing else.

More work for the model level

These flags are for the model level only. `--backtest` refuses them.

Flag	What it does
<code>--configs N</code>	the configs that <code>chapkit test</code> makes (1)
<code>--trainings N</code>	the trainings of each config (1)
<code>--predictions N</code>	the predictions of each trained model (1)
<code>--rows N</code>	the rows of the training data (250)
<code>--predict-rows N</code>	the rows of the data of each prediction (150)

```
chap-devops models test my_model --rows 600 --predict-rows 300
chap-devops models test my_model --configs 2 --trainings 2
```

Evaluate on real data

In a subdirectory of the deployment (`mkdir eval && cd eval`):

```
chap-devops chap eval --model-name my_model \  
  --dataset-csv laos_subset.csv --output-file mine.nc
```

Or straight from the checkout, with no deployment:

```
chap-devops chap eval --model-name ./my_model_checkout \  
  --dataset-csv laos_subset.csv --output-file mine_local.nc
```

- A model id of the deployment: `chap-devops` starts that model, without `chap-core`.
- A directory with an `MLproject`: `chap` runs it in the worker image (uv, R, INLA). Add `--docker` when the `MLproject` uses `docker_env`.

Compare with the models that exist

```
chap-devops chap eval --model-name auto_arima_chapkit \  
  --dataset-csv laos_subset.csv --output-file arima.nc  
chap-devops chap export-metrics mine.nc arima.nc --output-file comparison.csv  
chap-devops chap plot-backtest mine.nc --output-file mine.html
```

- One row for each model: `crps`, `mae`, `rmse`, the interval coverage and more.
- A baseline: `--model-name builtin:seasonal_median`. It needs chap-core v2.5.0 or later.

Configurations for the marketplace entry

```
chap-devops models configs add my_model --name short_lags --set n_lags=2,2
chap-devops models configs export my_model --out configurations.yaml
```

- The file is the `configurations:` block of a marketplace entry. Put it in the entry of your model.
- Each configuration has its variant name, a `description` and a `config`.
- In another deployment, this adds all of them:

```
chap-devops models configs add my_model --from configurations.yaml
```

The whole loop, after a code change

```
docker build --platform linux/amd64 -t my-model:dev .      # 1. build again
chap-devops restart                                       # 2. use the new build
chap-devops models test my_model                         # 3. it still predicts
chap-devops models test my_model --backtest --seed 1     # 4. through chap-core
cd eval
chap-devops chap eval --model-name my_model \
  --dataset-csv laos_subset.csv --output-file mine.nc    # 5. on real data
chap-devops chap export-metrics mine.nc arima.nc --output-file comparison.csv
```

`--seed 1` gives the same sample data each time, so the scores of two builds compare.

When something fails

```
chap-devops logs -f my-model      # the model's own log
chap-devops jobs                  # what chap-core ran, newest first
chap-devops jobs logs JOB_ID      # why a backtest failed: the model's stderr
chap-devops api GET /v2/services  # what chap-core registered, as JSON
chap-devops doctor                # the machine and the deployment
```

- A failed backtest says why in `chap-devops jobs logs` and nowhere else.
- From chap-core v2.5.0, `chap-devops jobs` also shows a `baseline/<model>` job for each built-in model of a backtest.
- A model test failure names the phase (`train`, `predict`) and the first line of the model's error. Add `-vv` for the whole output.

Start now

```
chap-devops models new my-model && cd my-model  
git init && git add . && git commit -m "feat: first version"  
docker build --platform linux/amd64 --build-arg GIT_REVISION=$(git rev-parse HEAD) -t my-model:dev .  
chap-devops run my-model:dev
```

Models and the marketplace: <https://winterop-com.github.io/chap-devops/models.html>