Open Policy Agent Policy as code

START HERE

`opa eval -d <dir> -i <input.json> '<query>'`
Evaluate a query

`opa run <dir>`
Open the REPL

`opa test <dir>`
Run test_ rules

`opa run -s <dir>`
Serve policies over HTTP

`opa fmt -w <dir>`
Format Rego files

`opa build -b <dir>`
Package a bundle

`opa check <dir>`
Find parse errors

`opa inspect <bundle>`
Show bundle contents

Policy .rego files → `opa check / fmt` → **Tested** test_ rules → `opa test` → **Bundle** .tar.gz → `opa build` → **Server** opa run -s → `query` → **Decision** JSON result

CONCEPTS

> OPA evaluates a query against data (policies plus JSON) and an input document; the answer is JSON.

> Rules are queried as data.<package>.<rule>; an undefined result prints an empty object, not false.

> Rego itself is not covered here; this sheet is about the opa command.

EVAL

`opa eval -d <path> '<query>'`
Evaluate a query against policy and data

`opa eval -d <path> -i <input.json> '<query>'`
Supply an input document

`opa eval -d <path> -I '<query>'`
Read input from stdin

`opa eval -b <bundle> '<query>'`
Load a bundle instead of paths

`opa eval -f pretty '<query>'`
Human-readable output

`opa eval -f raw '<query>'`
Script-friendly value output

`opa eval -f values '<query>'`
One result array per line

`opa eval -f bindings '<query>'`
Show variable bindings only

`opa eval --fail '<query>'`
Exit non-zero if undefined or empty
Use in CI: without it an undefined result still exits 0.

`opa eval --fail-defined '<query>'`
Exit non-zero if the result is defined
Handy for deny/violation rules: the query must come back empty.

`opa eval --stdin '<query>'`
Read the query from stdin

`opa eval --explain=full '<query>'`
Trace how the query evaluated

`opa eval --profile '<query>'`
Profile expression cost

EVAL (cont.)

`opa eval --metrics '<query>'`
Print timing metrics

`opa eval -p -u input '<query>'`
Partial evaluation, input unknown

`opa eval --strict '<query>'`
Enable strict compile checks

REPL AND SERVER

`opa run`
Start the interactive REPL

`opa run <path>`
REPL with policies and data loaded

`opa run -s <path>`
Start the HTTP server

`opa run -s -a <addr>`
Listen on another address

`opa run -s -b <bundle>`
Serve a bundle

`opa run -w <path>`
Reload on file changes

`opa run -s -c <config.yaml>`
Use a config file

`opa run --set <key>=<value>`
Override a config value

`opa run -l debug`
Set the log level

`help input`
Inside the REPL: input help

`repl.input:<file>`
Inside the REPL: load a file as input

> The server answers POST /v1/data/<path> with a JSON body of the form {"input": {...}}.

TESTING

`opa test <path>`
Run all test_ rules

`opa test -v <path>`
Verbose, one line per test

`opa test -r '<regex>' <path>`
Run matching tests only

`opa test -c <path>`
Report coverage

`opa test --threshold <pct> -c <path>`
Fail below a coverage percentage

TESTING (cont.)

`opa test -w <path>`
Rerun tests on file changes

`opa test -b <bundle>`
Test against a bundle

`opa test --bench <path>`
Benchmark the tests

`opa test -f json <path>`
Machine-readable results

> Rules named todo_* are skipped and reported as skipped.

FORMAT AND CHECK

`opa fmt <path>`
Print formatted source

`opa fmt -w <path>`
Rewrite files in place

`opa fmt -d <path>`
Show a diff of changes

`opa fmt -l <path>`
List files that would change

`opa fmt --fail <path>`
Exit non-zero if files need formatting

`opa check <path>`
Parse and compile check

`opa check -S <path>`
Strict mode checks

`opa check -s <schema>`
Type check against a schema

`opa check -f json <path>`
JSON error output

`opa deps -d <path> '<query>'`
Show what a query depends on

BUNDLES

`opa build <path>`
Build a bundle (bundle.tar.gz)

`opa build -b <dir> -o <file>`
Name the output file

`opa build -r <rev> <path>`
Set the bundle revision

`opa build -e <entrypoint> <path>`
Declare an entrypoint

`opa build -O=1 -e <entrypoint> <path>`
Optimize (needs an entrypoint)

`opa build -t wasm -e <entrypoint> <path>`
Compile to WebAssembly

BUNDLES (cont.)

`opa build --prune-unused -e <entrypoint> <path>`
Drop rules the entrypoint does not use

`opa build --signing-key <key> <path>`
Sign the bundle

`opa build --verification-key <key> -b <bundle>`
Verify a signed bundle

`opa sign -b <dir> --signing-key <key>`
Write .signatures.json

`opa inspect <bundle>`
Summarize a bundle

`opa inspect -a <path>`
Show annotations

OTHER COMMANDS

`opa exec -b <bundle> <input.json>`
Run the default decision on input files

`opa exec -c <config> --decision <path> <files>`
Pick the decision and config

`opa bench -d <path> -i <input.json> '<query>'`
Benchmark a query

`opa version`
Show version and build

> Most commands take -d for data/policy paths and -b for bundles; both read directories recursively.

RECIPES

Gate a Kubernetes manifest in CI

```
# deny rule must come back empty
opa eval --fail-defined -d
policy/ -i deploy.json
'data.k8s.deny'
```

Check, format and test before commit

```
opa fmt --fail policy/
opa check -S policy/
opa test -c --threshold 80
policy/
```

Query a running server

```
opa run -s -b bundle.tar.gz &
curl -s localhost:8181/v1/data/
app/allow -d '{"input": {"user":
"bob"}}'
```

Optimized bundle

```
opa build -O=1 -e app/allow -o
bundle.tar.gz policy/
opa inspect bundle.tar.gz
```