Mastering Bazel: The Ultimate Guide for Polyglot Build Systems

Keywords: Bazel, BUILD files, Starlark, Bazel macros, Bazel rules, monorepo, hermetic builds, remote caching, build optimization, CI/CD, Google build system

---

What Is Bazel?

Bazel is Google's open-source build and test tool that scales to massive codebases across multiple languages. It offers:

- **Hermetic builds** (fully deterministic)
- **Dependency isolation**
- **Language-agnostic support** (Java, Go, Python, C++, Rust, etc.)
- **Remote caching**
- **Incremental builds**
- **Support for monorepos**

Bazel uses a **declarative build language** called **Starlark**, a subset of Python.

Bazel Project Anatomy

css

CopyEdit

my_project/

├── WORKSPACE

├── BUILD

├── src/

│ └── BUILD

```
│   └── hello.py
├── tools/
│   └── BUILD
│   └── codegen.bzl
└── third_party/
    └── BUILD
```

---

1. WORKSPACE File – The Root of Bazel

The WORKSPACE file tells Bazel:

- Where the root of the repo is
- What external dependencies to load (e.g., HTTP archives, Git repos)

python

CopyEdit

```
workspace(name = "my_project")


load("@bazel_tools//tools/build_defs/repo:http.bzl", "http_archive")


http_archive(
    name = "rules_python",
    url = "https://github.com/bazelbuild/rules_python/releases/download/0.17.2/rules_python-0.17.2.tar.gz",
    sha256 = "2b67384c3b2e7e4f957f6a...f62",  # shortened
)


load("@rules_python//python:repositories.bzl", "py_repositories")
```

```
py_repositories()
```

---

2. BUILD File – Declaring Your Targets

Bazel doesn't use Makefiles. Instead, you declare **build rules**.

Example: Python Binary Target

python

CopyEdit

```
py_binary(
    name = "hello",
    srcs = ["hello.py"],
    deps = ["@rules_python//python/runfiles"],
)
```

Example: C++ Library and Test

python

CopyEdit

```
cc_library(
    name = "math_utils",
    srcs = ["math_utils.cc"],
    hdrs = ["math_utils.h"],
    visibility = ["//visibility:public"],
)


cc_test(
    name = "math_utils_test",
    srcs = ["math_utils_test.cc"],
```

```
    deps = [":math_utils"],
)
```

---

3. Starlark – Bazel's Custom Rule Language

You can write **custom build logic** using **Starlark**, a safe Python-like DSL.

Custom Rule: Code Generator for Protobuf-like DSL

python

CopyEdit

```
def _codegen_impl(ctx):
    output = ctx.actions.declare_file(ctx.label.name + "_gen.py")
    ctx.actions.run(
        outputs = [output],
        inputs = [ctx.file.src],
        executable = ctx.executable.tool,
        arguments = [ctx.file.src.path, output.path],
    )
    return DefaultInfo(files = depset([output]))

codegen_rule = rule(
    implementation = _codegen_impl,
    attrs = {
        "src": attr.label(allow_single_file = True),
        "tool": attr.label(cfg = "host", executable = True, allow_files = True),
    },
)
```

Then use in BUILD:

python

CopyEdit

```
load("//tools:codegen.bzl", "codegen_rule")


codegen_rule(
    name = "gen_api",
    src = "api.dsl",
    tool = "//tools:codegen_tool",
)
```

---

4. Remote Caching and Execution (Real-World Scale)

In large tech orgs (like Google or Twitter), Bazel is used with:

- **Remote Build Execution (RBE)** to parallelize builds on the cloud
- **Remote Caching** to avoid redundant compilation
- **Build Event Protocol (BEP)** to stream logs into dashboards

bash

CopyEdit

```
bazel build //:hello \
  --remote_cache=https://cache.mycorp.internal \
  --remote_header="Authorization=Bearer $TOKEN"
```

You can also plug into **Buildkite**, **CircleCI**, or **GitHub Actions** using bazelisk for CI/CD.

---

5. External Dependencies (npm, pip, Maven, Go)

Python Pip Example:

python

CopyEdit

```
load("@rules_python//python:pip.bzl", "pip_install")


pip_install(
    requirements = "//:requirements.txt",
)
```

Go Modules:

python

CopyEdit

```
load("@io_bazel_rules_go//go:deps.bzl", "go_register_toolchains", "go_rules_dependencies")


go_rules_dependencies()
go_register_toolchains()
```

---

6. Testing with Bazel

Bazel supports:

- py_test, cc_test, java_test
- Custom test runners
- Parallel test execution
- Sandboxed tests (isolated)

python

CopyEdit

```
py_test(
    name = "math_test",
    srcs = ["math_test.py"],
    deps = ["//src:math_utils"],
)
```

Run with:

bash

CopyEdit

```
bazel test //src:math_test --test_output=errors
```

---

7. Writing Bazel Macros

Macros make it easy to reuse build patterns.

python

CopyEdit

```
def py_binary_with_config(name, srcs):
    py_binary(
        name = name,
        srcs = srcs + ["config.py"],
        deps = [],
    )
```

Use in BUILD:

python

CopyEdit

```
load("//tools:macros.bzl", "py_binary_with_config")
```

```
py_binary_with_config(
    name = "launch_app",
    srcs = ["main.py"],
)
```

---

8. Bazel for Monorepos

Bazel was **built for Google's monorepo**. You can:

- Reuse libraries across teams
- Build only what changed
- Create clean dependency graphs

Example:

pgsql

CopyEdit

```
my_company/
├── WORKSPACE
├── libs/
│   └── BUILD (shared_libs)
├── services/
│   └── user/
│       └── BUILD (depends on //libs)
│   └── payments/
│       └── BUILD (depends on //libs)
```

Running:

bash

CopyEdit

bazel query 'deps(//services/user:auth_service)' --noimplicit_deps

Gives a complete, minimal dependency graph.

---

9. Performance Optimization Tips

Tip	Description
--experimental_action_listener	For profiling
--experimental_convenience_symlinks=clean	Avoids clutter
--experimental_generate_json_trace_profile	Chrome-style performance trace
Remote execution + sandbox	Isolates builds fully
Target wildcarding	//... to build all targets

---

10. Bazel in CI/CD Pipelines

Use Bazel with **GitHub Actions**:

yaml

CopyEdit

```
- name: Install Bazelisk
  run: npm install -g @bazel/bazelisk

- name: Build
  run: bazel build //...

- name: Test
  run: bazel test //...
```

You can export Bazel metrics to **Grafana** using BEP + custom log exporters.

---

11. Advanced: Bazel Rule for a DSL Transpiler

python

CopyEdit

```
def _dsl_compile_impl(ctx):
    output = ctx.actions.declare_file(ctx.label.name + ".gen.go")
    ctx.actions.run(
        outputs = [output],
        inputs = [ctx.file.src],
        arguments = ["--in", ctx.file.src.path, "--out", output.path],
        executable = ctx.executable.transpiler,
    )
    return DefaultInfo(files = depset([output]))


dsl_compile = rule(
    implementation = _dsl_compile_impl,
    attrs = {
        "src": attr.label(allow_single_file = True),
        "transpiler": attr.label(cfg = "host", executable = True, allow_files = True),
    },
)
```

Now use this to generate Go code from your DSL file:

python

CopyEdit

```
dsl_compile(
```

```
    name = "api_gen",
    src = "schema.dsl",
    transpiler = "//tools:mydslc",
)
```