Automagically: Managing Python Projects with Bazel

Moritz Kröger

LLT Lehrstuhl für Lasertechnik | RWTH AACHEN UNIVERSITY

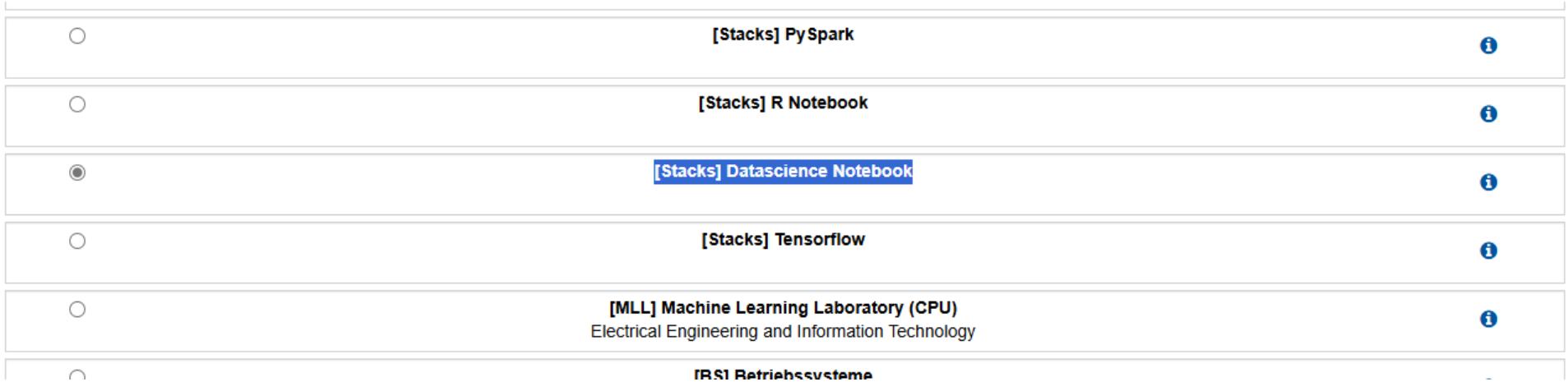
1. Getting an development environment
2. Build Tools and why do i need them?
3. Terminal Basics
4. Your first Bazel project
5. Building Modules and Unittests
6. External Pip Dependencies
7. Building a Docker Container

1. Getting an development environment
2. Build Tools and why do i need them?
3. Terminal Basics
4. Your first Bazel project
5. Building Modules and Unittests
6. External Pip Dependencies
7. Building a Docker Container

Gettings started

Please visit https://jupyter.rwth-aachen.de/ and login with your RWTH Account.

Select the [Stacks] Datascience Notebook



Scroll down and press „Start“

Gettings started

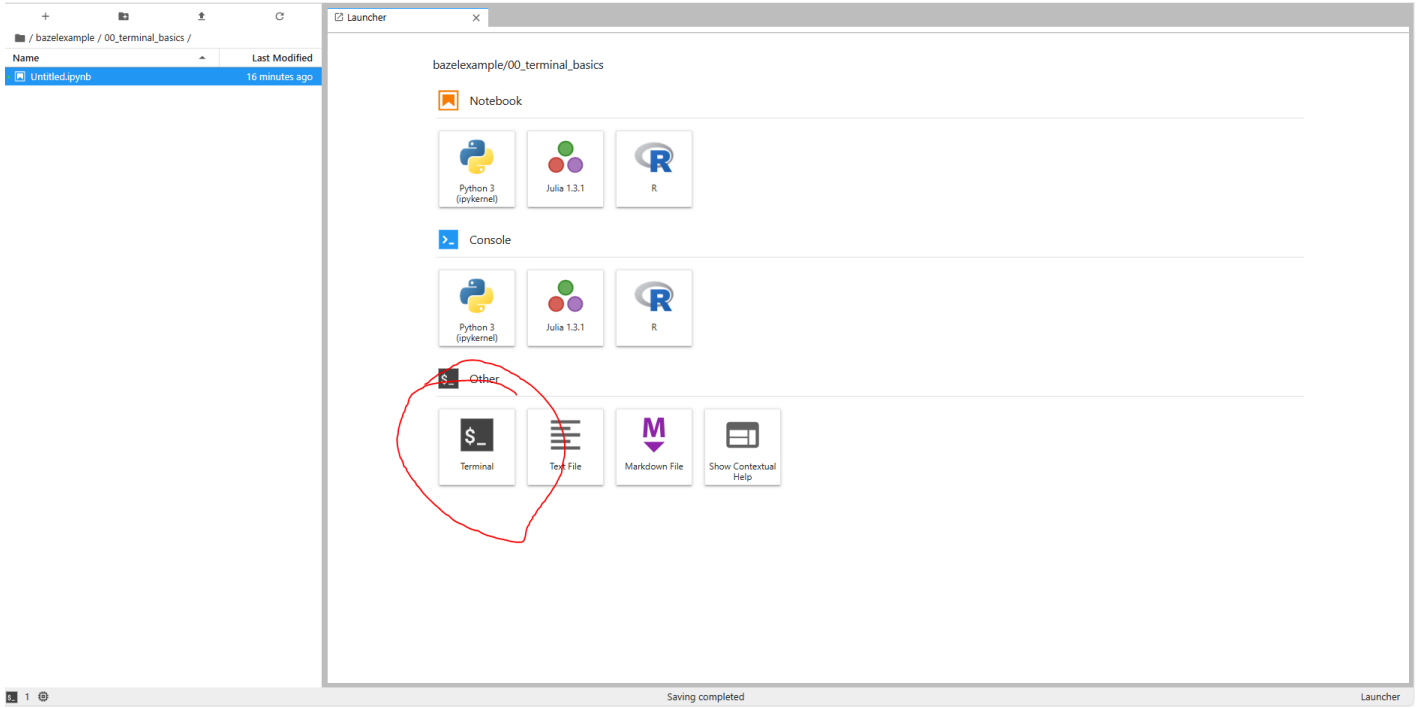
Open the Terminal in your Notebook

Try running the Command:

```
echo "Hello Terminalworld!"
```

Afterwards download the Git Repository by running:

```
git clone https://github.com/Smokrow/bazelexample.git
```



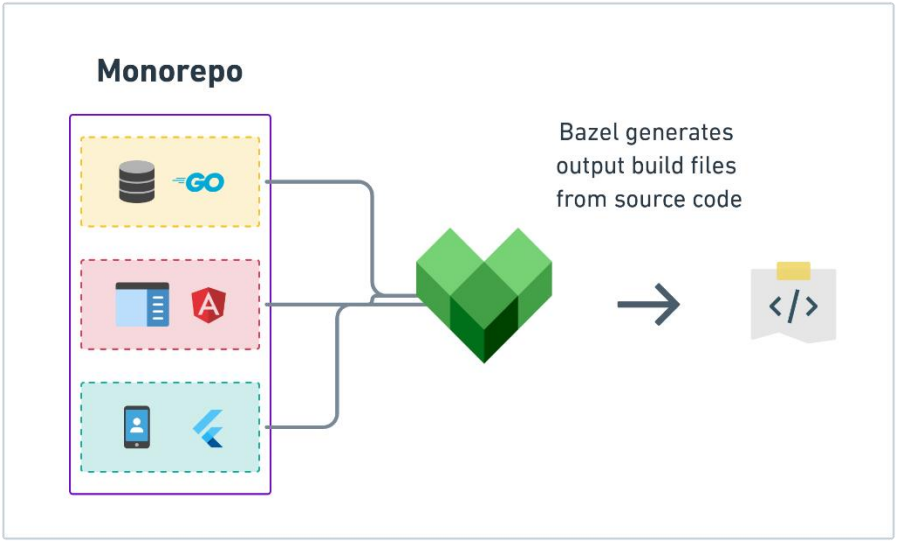
1. Getting an development environment
2. Build Tools and why do i need them?
3. Terminal Basics
4. Your first Bazel project
5. Building Modules and Unittests
6. External Pip Dependencies
7. Building a Docker Container

What are Build Tools?

Tools that automate the process of compiling source code into executable code.

What do they do?

- **Compilation**:
  - Converts source code into binary code.
- **Dependency Management**:
  - Handles external libraries and dependencies.
- **Packaging**:
  - Bundles code and resources into distributable formats.
- **Testing**:
  - Automates running tests to ensure code quality.
- **Deployment**:
  - Facilitates the distribution of applications to environments.



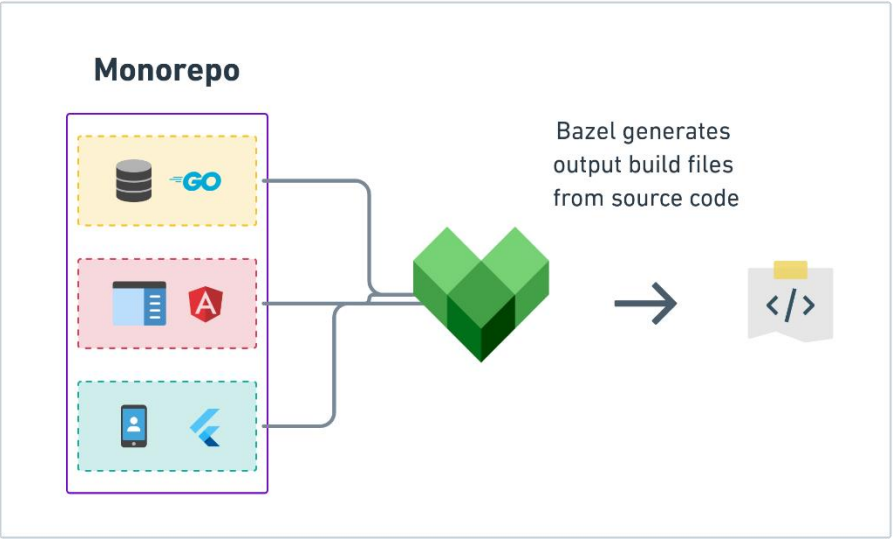
Src: https://semaphore.io/blog/bazel-build-tutorial-examples

What are Build Tools?

Tools that automate the process of compiling source code into executable code.

Why do I need them?

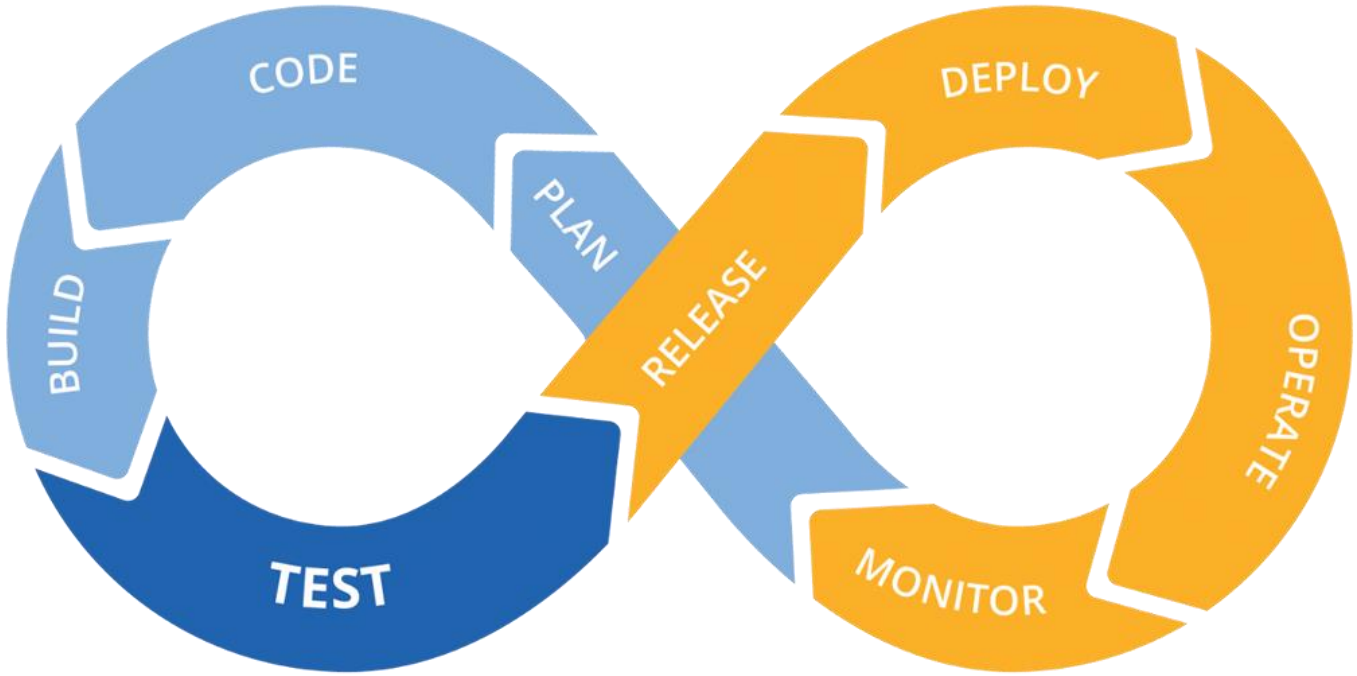
- **Efficiency**:
  - Automates repetitive tasks, saving time for developers.
- **Consistency**:
  - Ensures a uniform build process across different environments.
- **Error Reduction**:
  - Minimizes human error in the build process.
- **Scalability**:
  - Supports large projects with complex dependencies.
- **Continuous Integration/Continuous Deployment (CI/CD)**:
  - Integrates with CI/CD pipelines for automated testing and deployment.



Src: https://semaphore.io/blog/bazel-build-tutorial-examples

Example Tools: Make, Cmake, Bazel, Pants, Maven, Gradle, Ant, Setuptools …

What are Build Tools?



Src: https://www.civo.com/blog/the-role-of-the-ci-cd-pipeline-in-cloud-computing

What is Bazel?

- Open Source Build Tool released by Google in 2015
- Strong focus on:

  Fast Builds with Caching

  Isolation with Hermeticity

  Supports Many Languages

  Reproducibility

- Who uses Bazel?
  - Google
  - SpaceX
  - Tinder
  - Uber

Src: httpsbazel.build

1. Getting an development environment
2. Build Tools and why do i need them?
3. Terminal Basics
4. Your first Bazel project
5. Building Modules and Unittests
6. External Pip Dependencies
7. Building a Docker Container

Terminal Basics

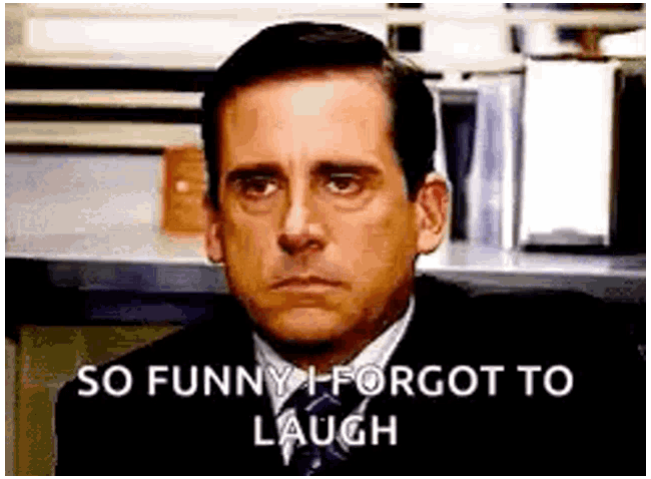
You can find the Terminal Basics Exercise in 00_terminal_basics in your downloaded git repository

Terminal Basics

You can find the Terminal Basics Exercise in 00_terminal_basics in your downloaded git repository

Remember to ask ChatGPT for a funny reason why you should use the terminal:

"The terminal is like a magic portal: type some spells (commands),
and suddenly, your computer does things you didn't even know it could do—just try not to summon any demons!"



1. Getting an development environment
2. Build Tools and why do i need them?
3. Terminal Basics
4. Your first Bazel project
5. Building Modules and Unittests
6. External Pip Dependencies
7. Building a Docker Container

Installing Bazel with Bazelisk

To install Bazelisk run inside your terminal:

```
npm install -g @bazel/bazelisk
```

This will install the Bazel Launcher Bazelisk. Bazelisk controls the currently installed Bazel version and ensures that you always have the correct version Bazel version that is defined by your Software Project.

If no version is defined it just downloads the most up to date version of bazel

Setting up a Bazel Project

Bazel uses two very important file types:

- MODULE.bazel
- BUILD.bazel or BUILD

MODULE.bazel is a file in Bazel that defines the root directory and lists external dependencies for a project. It helps manage versions and keeps the code organized.
If it is not in your MODULE.bazel file your code cannot use it

BUILD.bazel files in Bazel are used to define build rules and target configurations specifically for Python projects. They specify how to compile, test, and package Python code, organizing the project's structure for efficient building and execution.

Setting up a Bazel Project

Minimal Example

Folder Structure:

```
my_project/
├── MODULE.bazel
├── BUILD.bazel
└── main.py
```

Module.BAZEL:

```
module(
    name = "my_project",
)
```

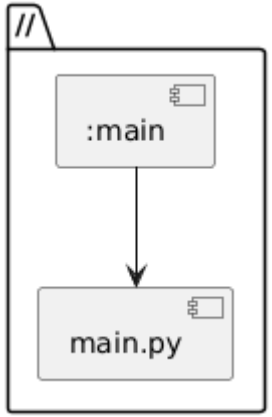
BUILD:

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

main.py:

```
print("hello world")
```

Minimal Example



What is a Rule Target

A rule target in Bazel is a defined unit of work that specifies how to build or manage a particular file or set of files, including their dependencies and build instructions

We are telling Bazel that there exists a python binary and we define that the source for this rule is main.py.

This allows us to run this python file with Bazel in the terminal:

```
bazelisk run :main
```

Bazelisk will now download the right Bazel and start it for you. Bazel will install everything you need and run the Python file for you.

BUILD:

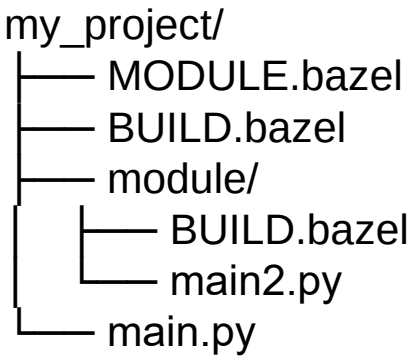
```
py_binary(
    name = "main",
    srcs = ["main.py"],
)
```

Target resolution

In Bazel, *:target* refers to a target in the current package, while *//:target* refers to a target in the root package of the workspace.

Possible Commands:

```
bazelisk run :main
bazelisk run //:main
bazelisk run //module:main2
```



BUILD.bazel:

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

module/BUILD.bazel:

```
py_binary(
    name = "main2",
    srcs = ["main2.py"],
)
```

Target resolution

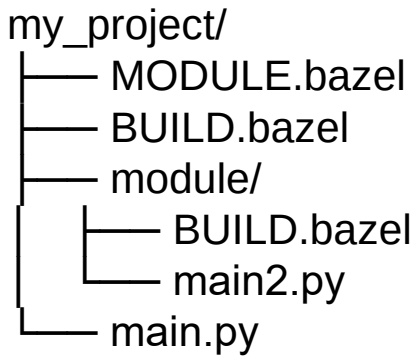
In Bazel, *:target* refers to a target in the current package, while *//:target* refers to a target in the root package of the workspace.

Possible Commands:

```
bazelisk run :main
bazelisk run //:main
bazelisk run //module:main2
```

bazel run //path_to_buildfile:buildtarget

„//“ means rootlevel.
It is defined by the location of MODULE.bazel



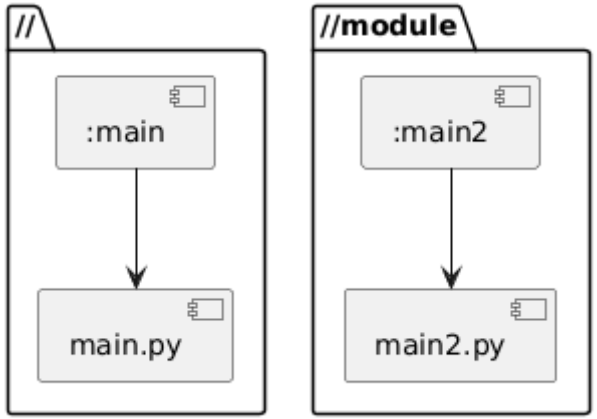
BUILD.bazel:

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

module/BUILD.bazel:

```
py_binary(
    name = "main2",
    srcs = ["main2.py"],
)
```

Target resolution



1. Getting an development environment
2. Build Tools and why do i need them?
3. Terminal Basics
4. Your first Bazel project
5. Building Modules and Unittests
6. External Pip Dependencies
7. Building a Docker Container

Importing in Python

Typically we don‘t want to write a gigantic spaghetti python file

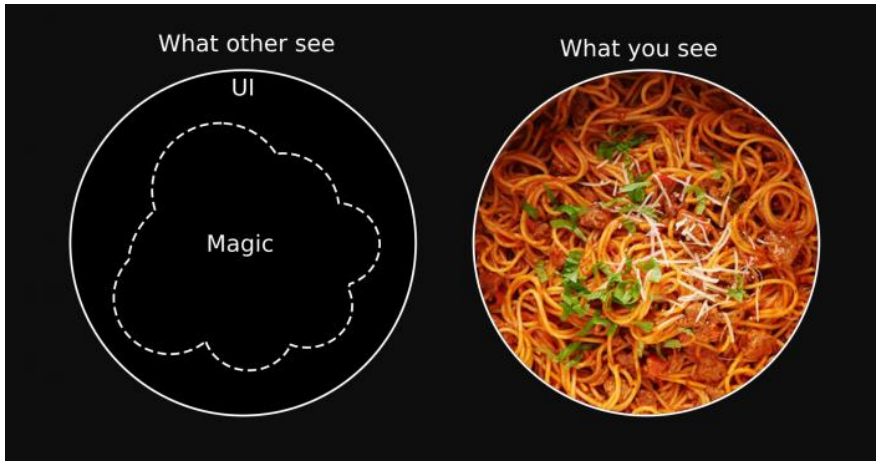
Dependencies on other files can be managed by the py_library rule.

We can build one like this:

```
py_library(
    name = "math",
    srcs = ["math.py"],
)
```

Other rules can now consume this rule and depent on it:

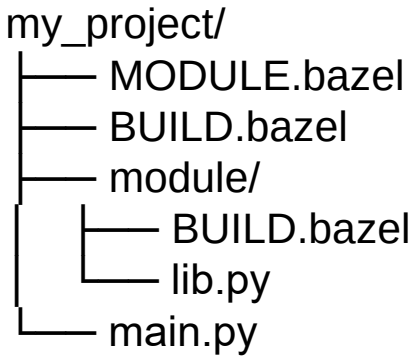
```
py_binary(
    name = "main",
    srcs = ["main.py"],
    deps = ["//:math"]
)
```



Src: medium.com

Importing in Python

Bazel does not allow relative imports. You always have to import from the root level



BUILD.bazel:

```
py_binary(
    name = "main",
    srcs = ["main.py"],
    deps = "[//module:lib"]
)
```

main.py

```
from module.lib import print_hello
```

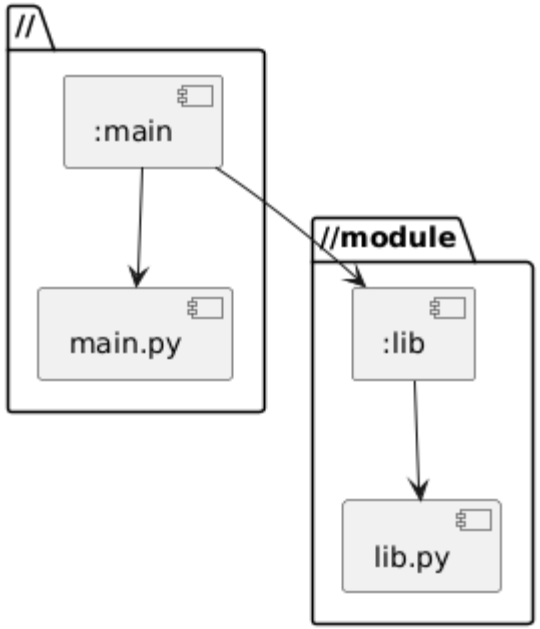
module/BUILD.bazel:

```
py_library(
    name = "lib",
    srcs = ["lib.py"],
)
```

module/lib.py

```
def print_hello():
    print("Hello")
```

Importing in Python



What do I do if need to analyse my project?

For larger projects we can visualize the targets and the dependencies of targets in a folder.

```
bazlisk query //somefolderpath
```

will for example show all targets in a subfolder

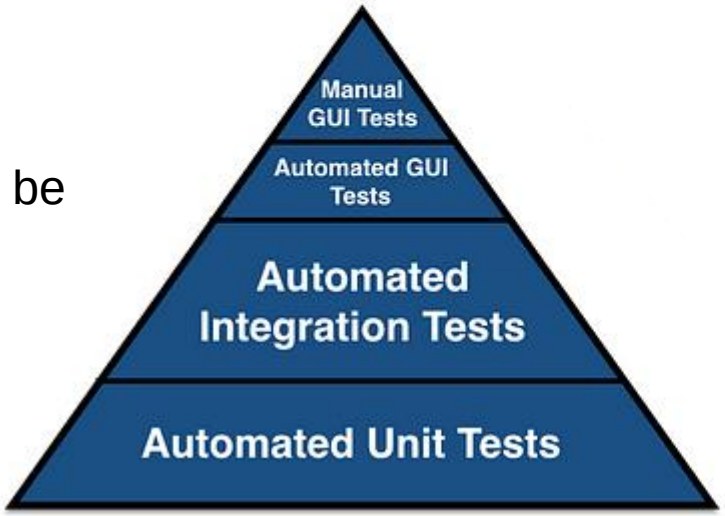
```
bazelisk query --noimplicit_deps  "deps(//:main)"
```

will print all files and targets this specific rule depends on.

More examples can be found here: Query quickstart | Bazel

The testing pyramid

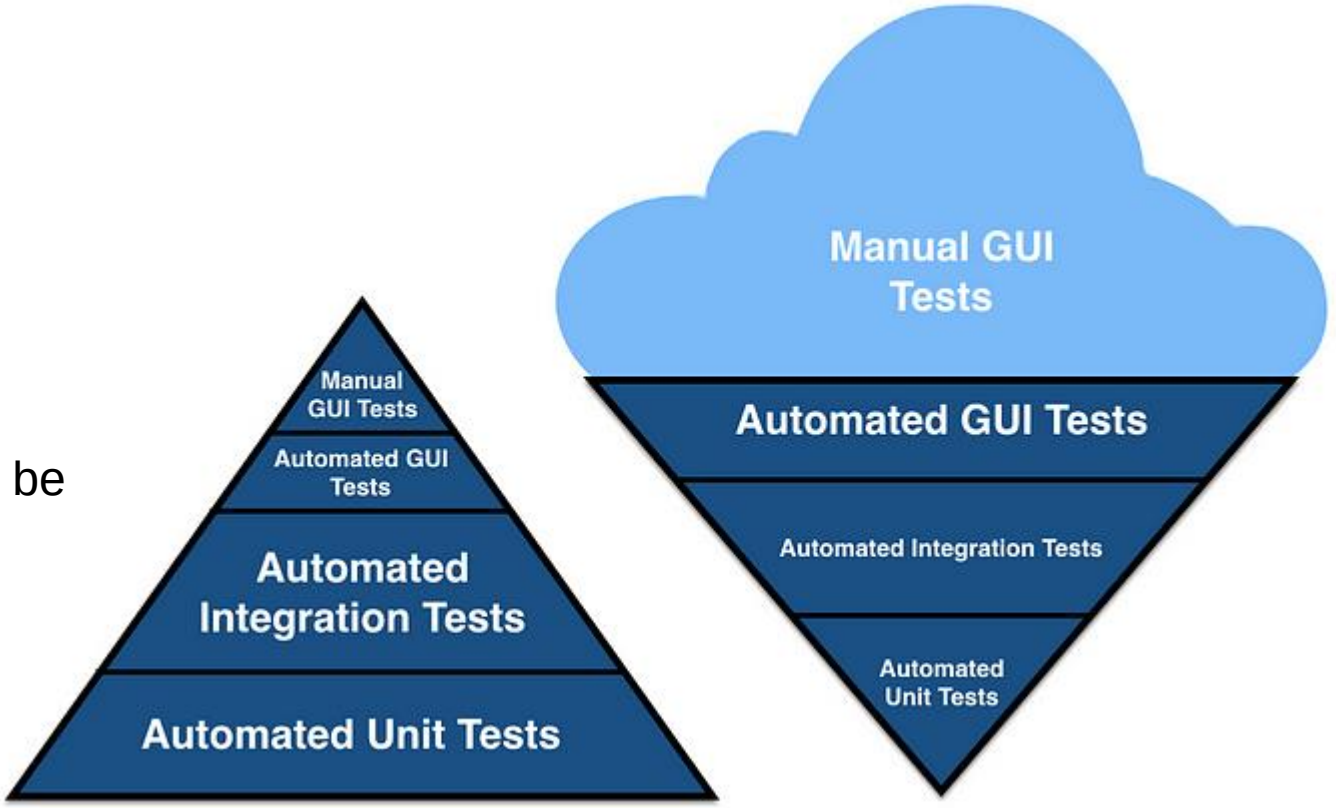
How it should be



Src: https://medium.com/@fistsOfReason/testing-is-good-pyramids-are-bad-ice-cream-cones-are-the-worst-ad94b9b2f05f

The testing pyramid

How it should be



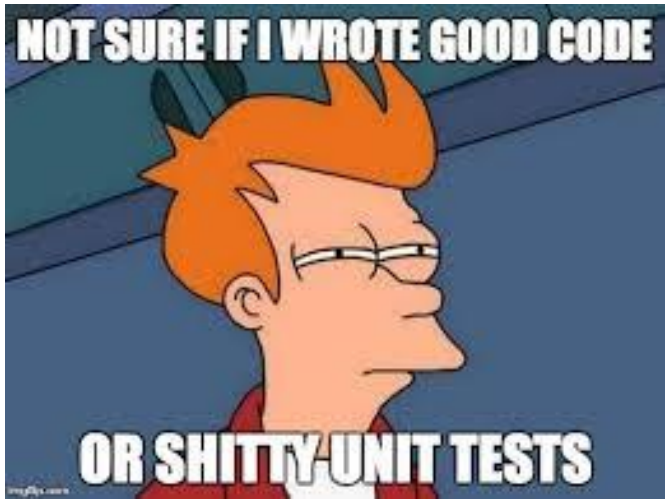
How it really is

Src: https://medium.com/@fistsOfReason/testing-is-good-pyramids-are-bad-ice-cream-cones-are-the-worst-ad94b9b2f05f

Code Testing

Example Python Test:

```
import unittest

class TestAbsFunction(unittest.TestCase):
    def test_positive_number(self):
        self.assertEqual(abs(10), 10)

    def test_negative_number(self):
        self.assertEqual(abs(-10), 10)

    def test_zero(self):
        self.assertEqual(abs(0), 0)

if __name == „__main__“:
     unittest.main()
```



How can I integrate that into Bazel?

Test can be declared by the py_test rule.

We can build one like this:

```
py_test(
    name = "abs_test",
    srcs = ["abs_test.py"],
    deps = ["//:somelibrarytarget"]
)
```

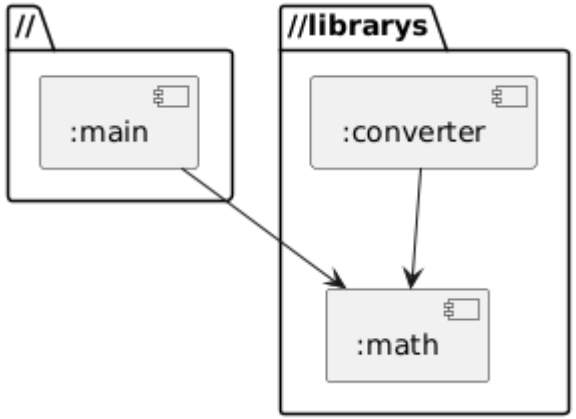
Wen can now either run individual tests, test a single test or test all tests:

```
bazelisk run :abs_test
bazelisk test :abs_test
bazelisk test //...
```

1. Getting an development environment
2. Build Tools and why do i need them?
3. Terminal Basics
4. Your first Bazel project
5. Building Modules and Unittests
6. External Pip Dependencies
7. Building a Docker Container

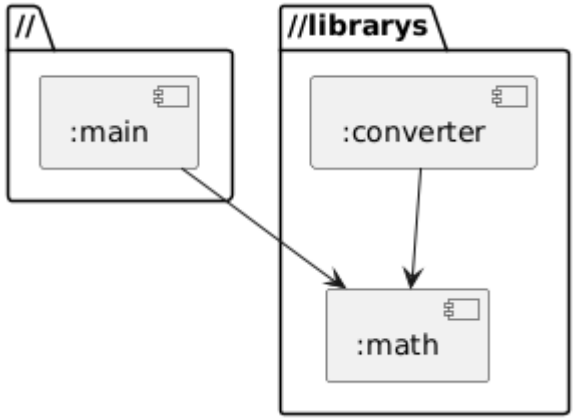
External Dependencies

Current State of the Repository:



External Dependencies

Current State of the Repository:

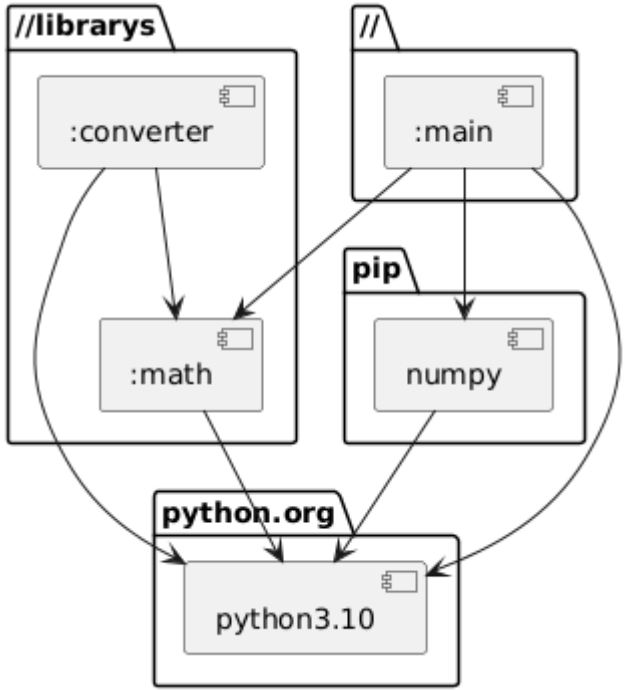


Moving from one Laptop
to the next could break our python version!

Also what‘s about useful stuff like like numpy?

External Dependencies

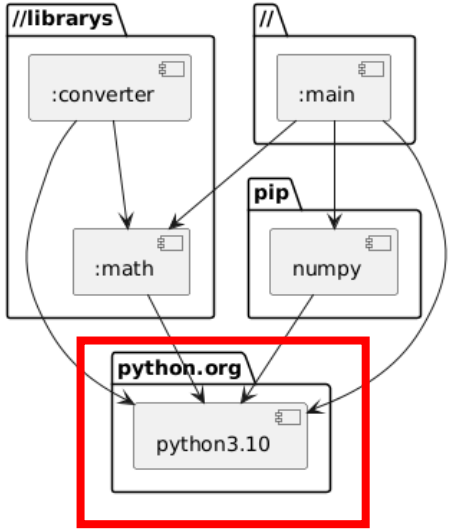
Hermetic State of the Repository with fixed Python and Pip:



Pinning Python with Bazel automagically

rules_python is an extension of Bazel which gives more control on what is going in with python in your repository.

It allows to pin down versions of python and allows automatical installation of this specific version.



Pinning Python with Bazel automagically

MODULE.bazel

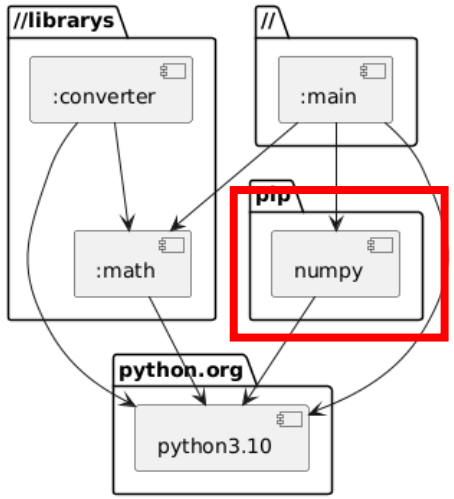
```
bazel_dep(name="rules_python", version=„1.3.0“)
python = use_extension("@rules_python//python/extensions:python.bzl", "python")
python.toolchain(python_version = "3.10", is_default = True)
```

BUILD.bazel

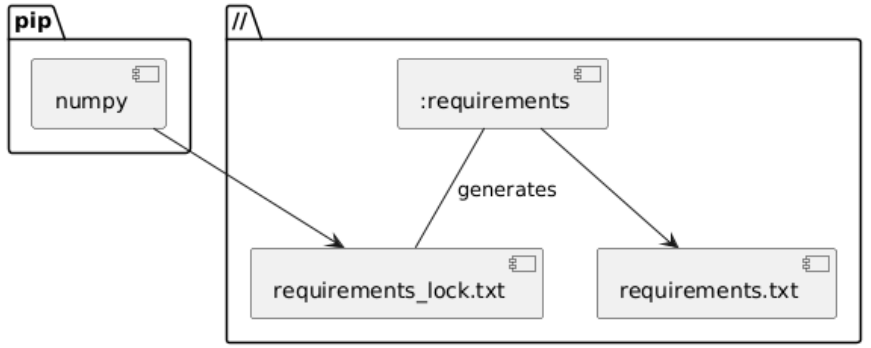
```
load("@rules_python//python:py_binary.bzl", "py_binary")
load("@rules_python//python:py_test.bzl", "py_test")

py_binary(
     name = “main”,
     srcs = [“main.py”])
```

Pinning Pip with Bazel automagically

rules_python allso allows pining of specific pip packages. It does so by adding a pip lockfile to your MODULE.build which then can be used by your rules. The lock file can be generated by



Usage of //:requirements to generate a pip lockfile for other targets



Pinning Pip with Bazel automagically

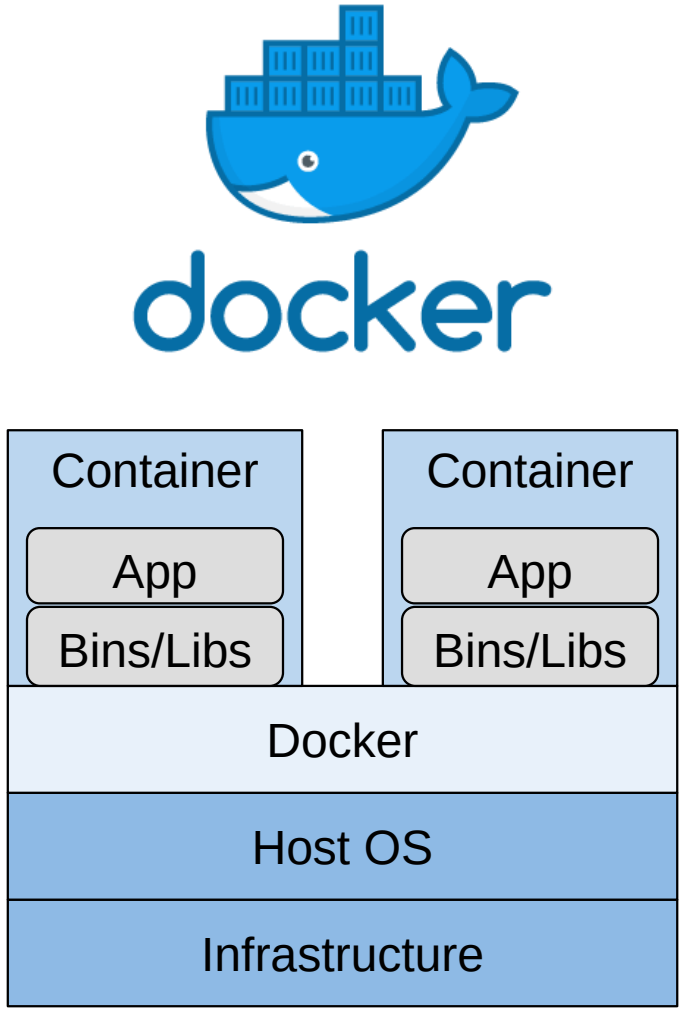
DEMO TIME

1. Getting an development environment
2. Build Tools and why do i need them?
3. Terminal Basics
4. Your first Bazel project
5. Building Modules and Unittests
6. External Pip Dependencies
7. Building a Docker Container

What does a docker container consist of?

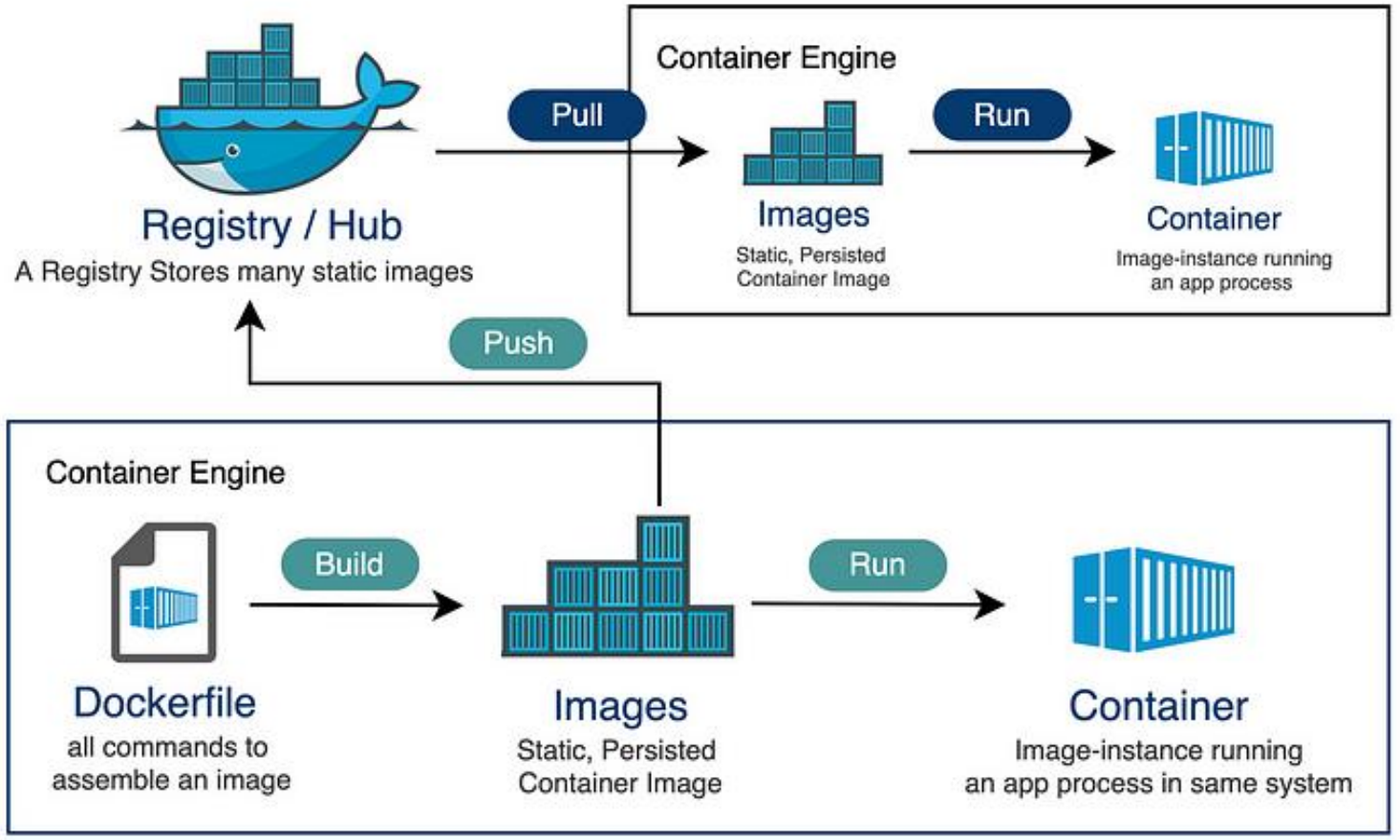
Containers allow packaging of your software with everything you need

- Image based deployment of software
- Runs everywhere
- Automated deployment possible

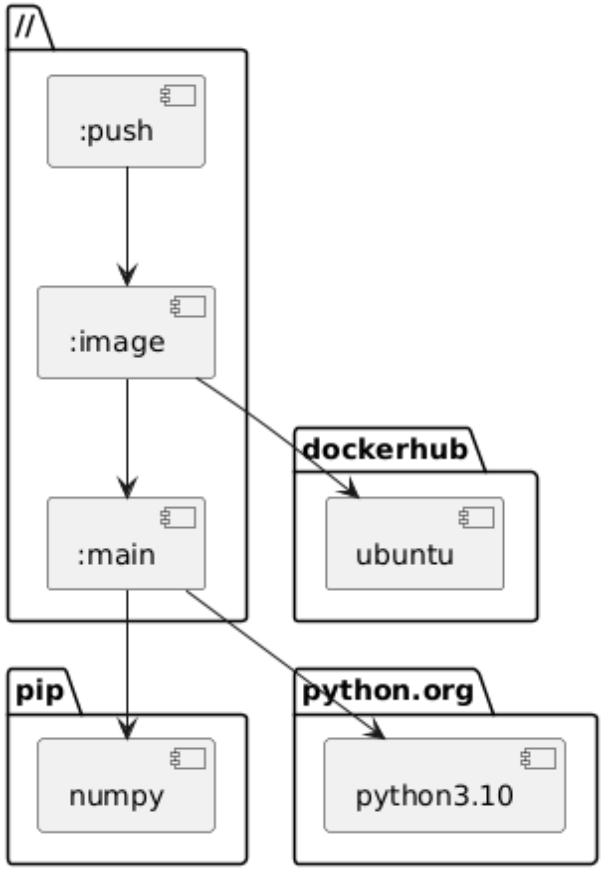


What does a docker container consist of?

Docker Images can build and shared



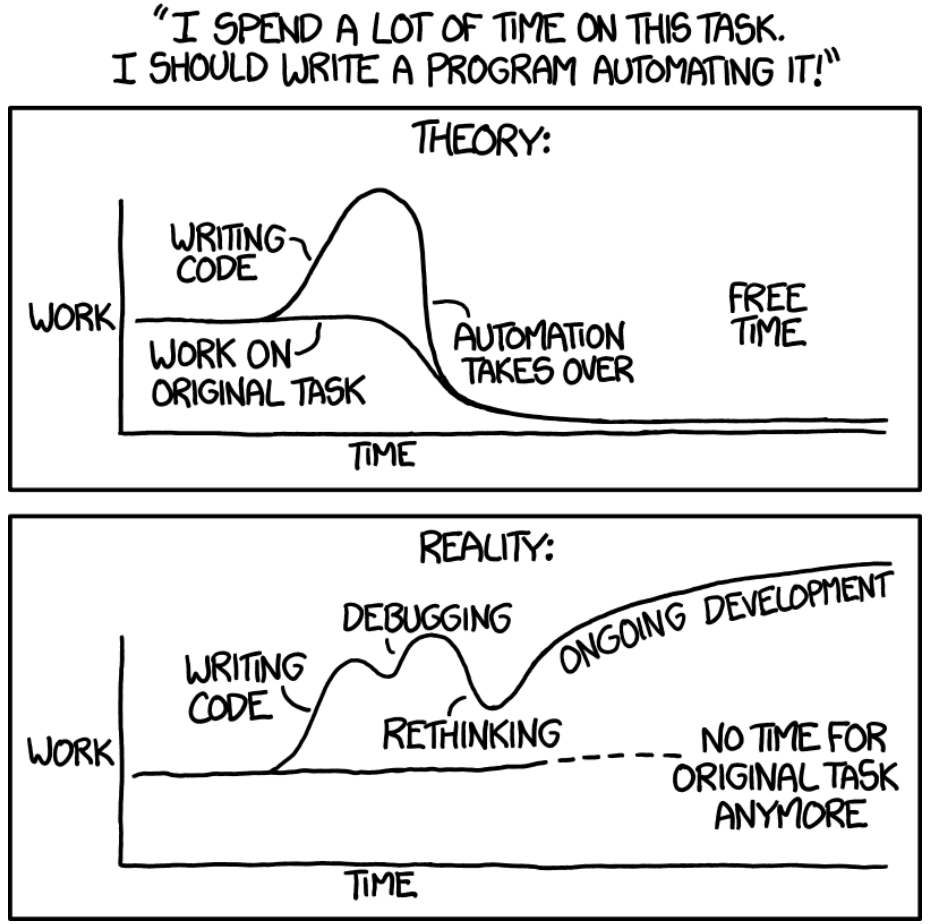
Building Docker Container with Bazel



Summary

What did we learn?

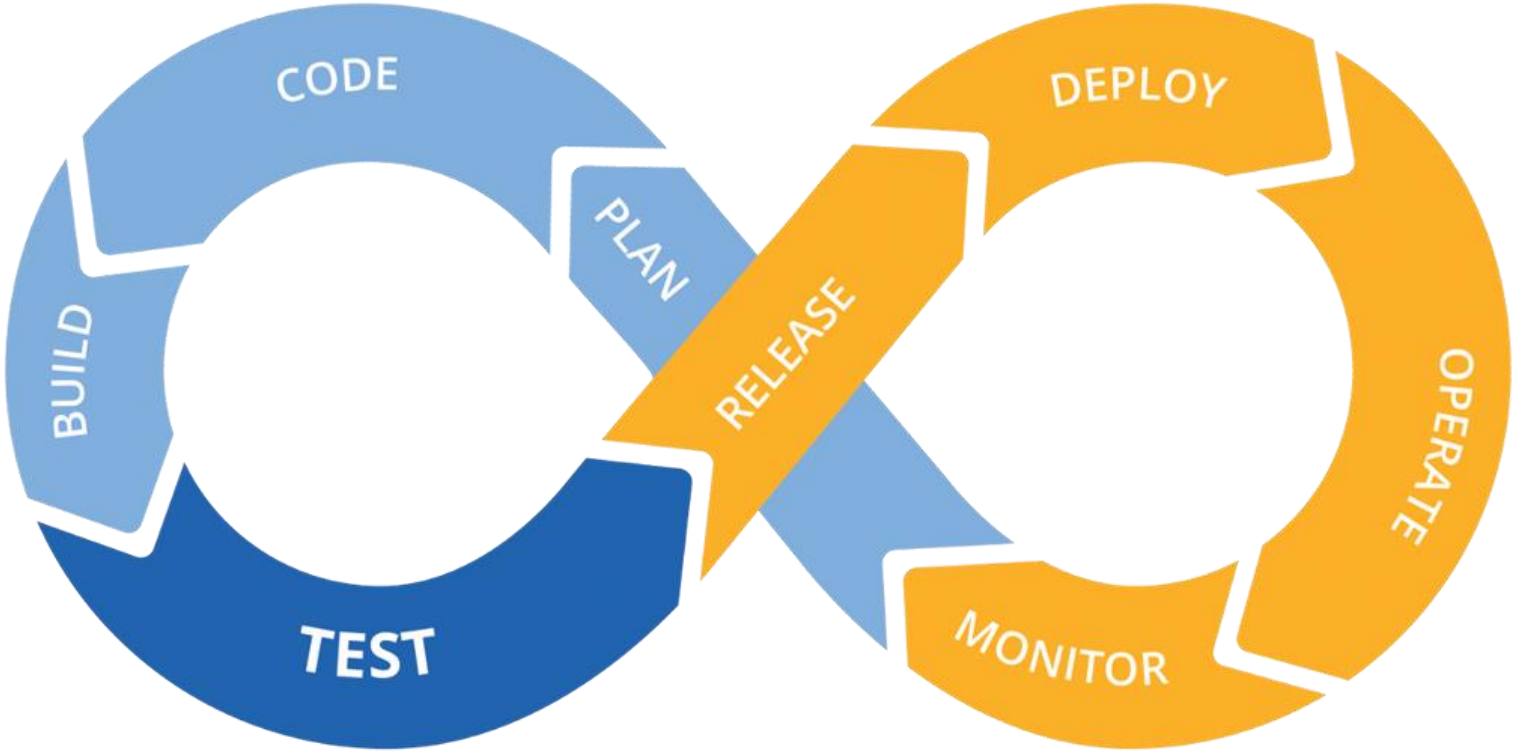
- Basics of Build tools
- Managing simple multi script projects with Bazel
- Added modules to the project and
  started querying the dependency structure of the repository
- Added unittests
- Added a pinned Python Version
- Added pinned Pip installs
- Put everything together in a Docker Container and published it



Summary

This repository can now:

Build
Test
Release



Src: https://www.civo.com/blog/the-role-of-the-ci-cd-pipeline-in-cloud-computing

Where to go from here

Useful links

- Bazel
  - Offical Website
- Bazel Central Registry
  - Offical Module Addons Repository
- bazel-contrib/rules_python: Bazel Python Rules
  - Rules Python Addon
- aspect-build/rules_py: More compatible Bazel rules for running Python tools and building Python projects
  - Higher level Python rules
- bazelbuild/rules_cc: C++ Rules for Bazel
  - C++ Rules
- aspect-build/rules_js: High-performance Bazel rules for running Node.js tools and building JavaScript projects
  - Javascript Rules
- ProdriveTechnologies/bazel-latex: Bazel build system rules for LaTeX
  - Latex Build rules (Yea thats right)
- Youtube Videos :D
- Getting Help | Bazel
- Bazel Slack
  - Extremly good for weird problems

Src: https://www.civo.com/blog/the-role-of-the-ci-cd-pipeline-in-cloud-computing

Backup: New Agenda