Building Artifacts using Docker

Justin Menga
FULL STACK TECHNOLOGIST
@jmenga pseudo.co.de

Introduction

Build Stage

- Application Artifact Types
- Adding package metadata to application
- Creating a builder service
- Building application artifacts
- Publishing application artifacts

Continuous Delivery Workflow

Test

Build

Release

Deploy

Build Workflow Using Docker

Create Build Environment

Reuse Test Environment
Create Builder Service

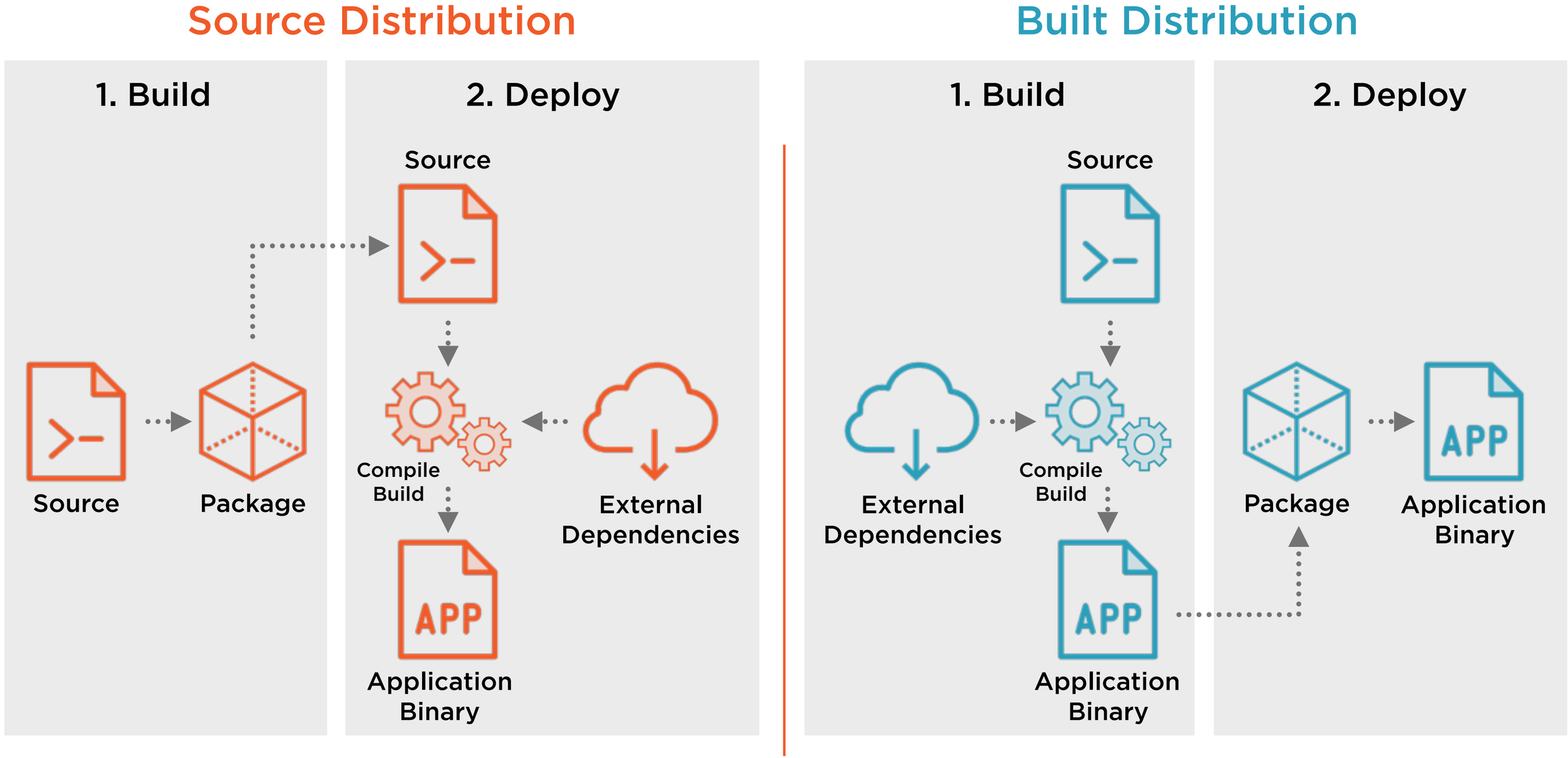
Build Artifacts

Compile Source
Build Python Wheel

Publish Artifacts

Publish Locally
Release Stage Inputs

Application Artifact Types


Source Distribution
1. Build
2. Deploy
Source
Package
Source
Compile
Build
External
Dependencies
APP
Application
Binary
Built Distribution
1. Build
2. Deploy
Source
External
Dependencies
Compile
Build
APP
Application
Binary
Package
APP
Application
Binary

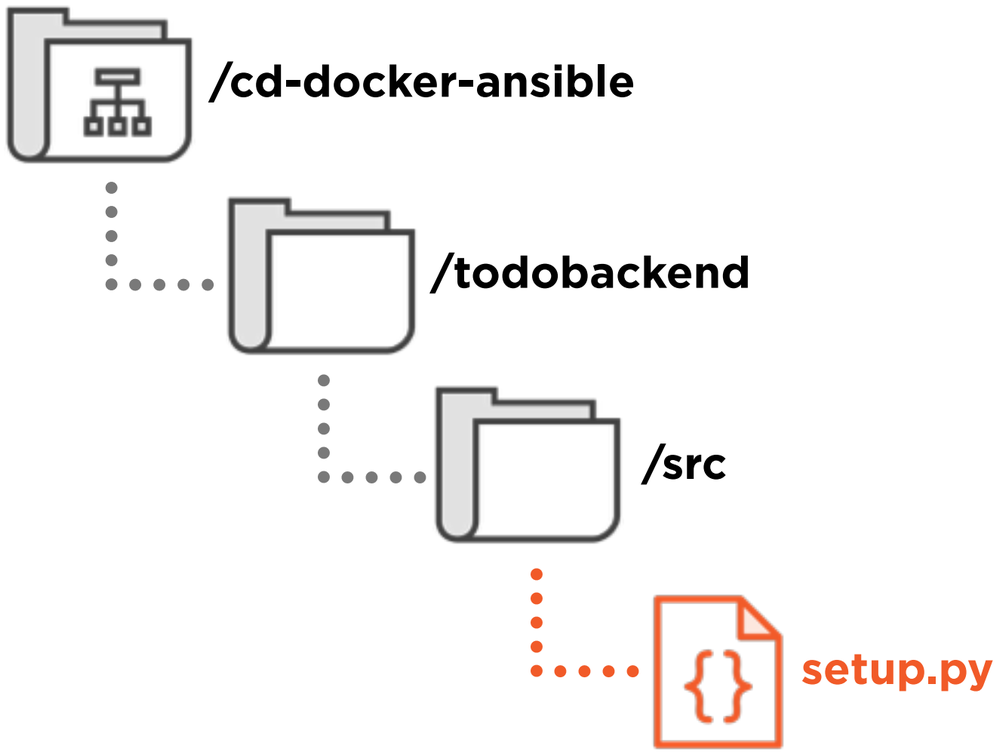
Demo

Building Application Artifacts

- Add package metadata to application
- Test and build consistency
- Adding a builder service
- Build and publish Python wheels

Adding Package Metadata to the Application

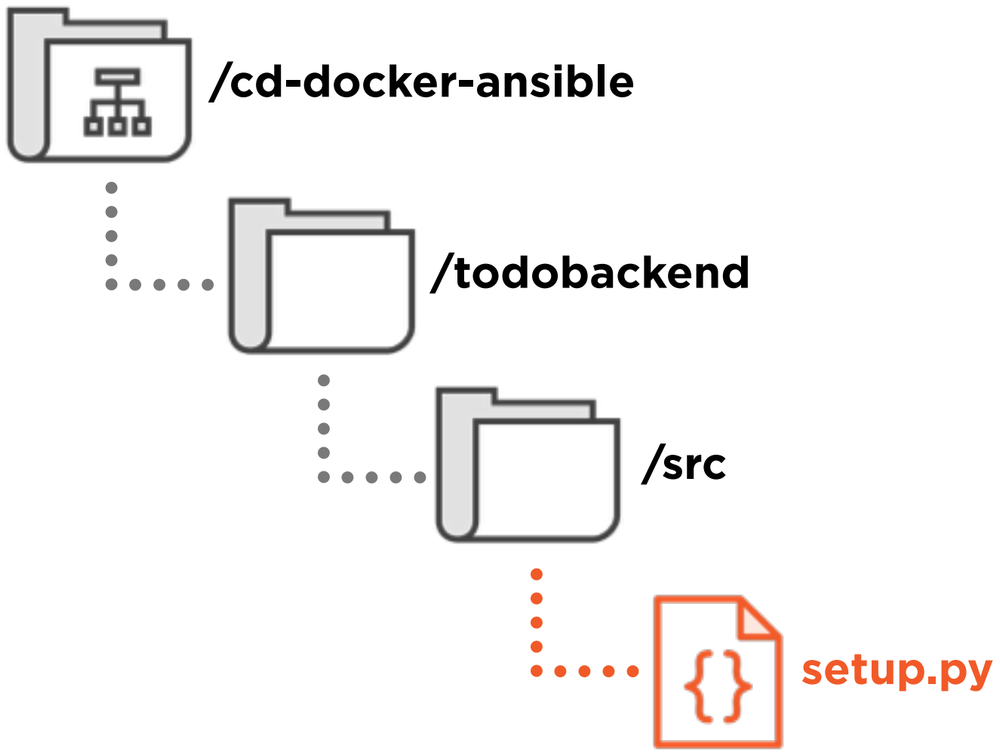
Package Metadata



```
name = “todobackend”

version = “0.1”

install_requires = [
 “Django>=1.8.6”,
 “MySQL-python>=1.2.5”,
  …
  …
]

…
```

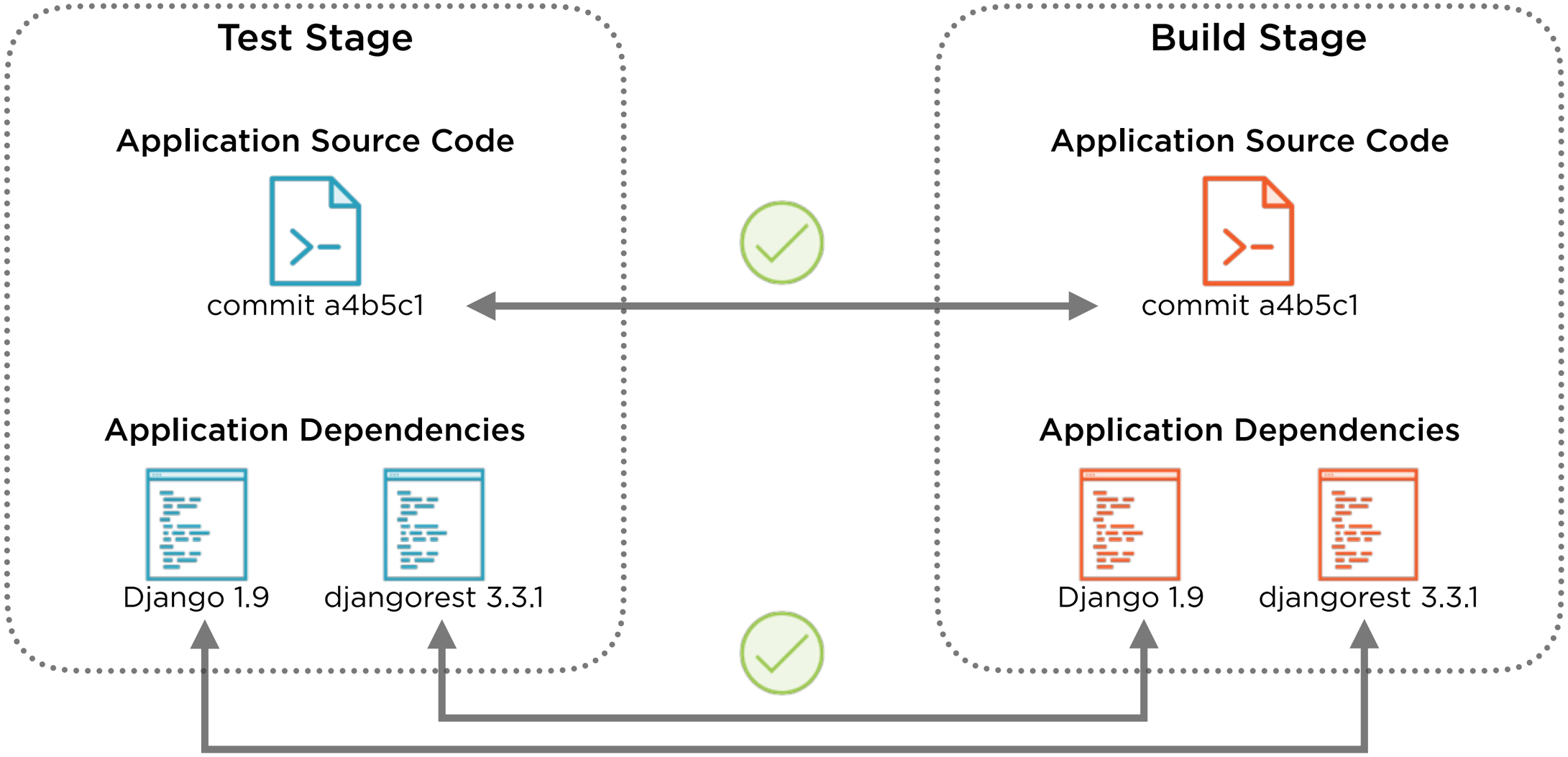
- Package name
- Package version
- Application dependencies
- Other settings

Course Folder Structure

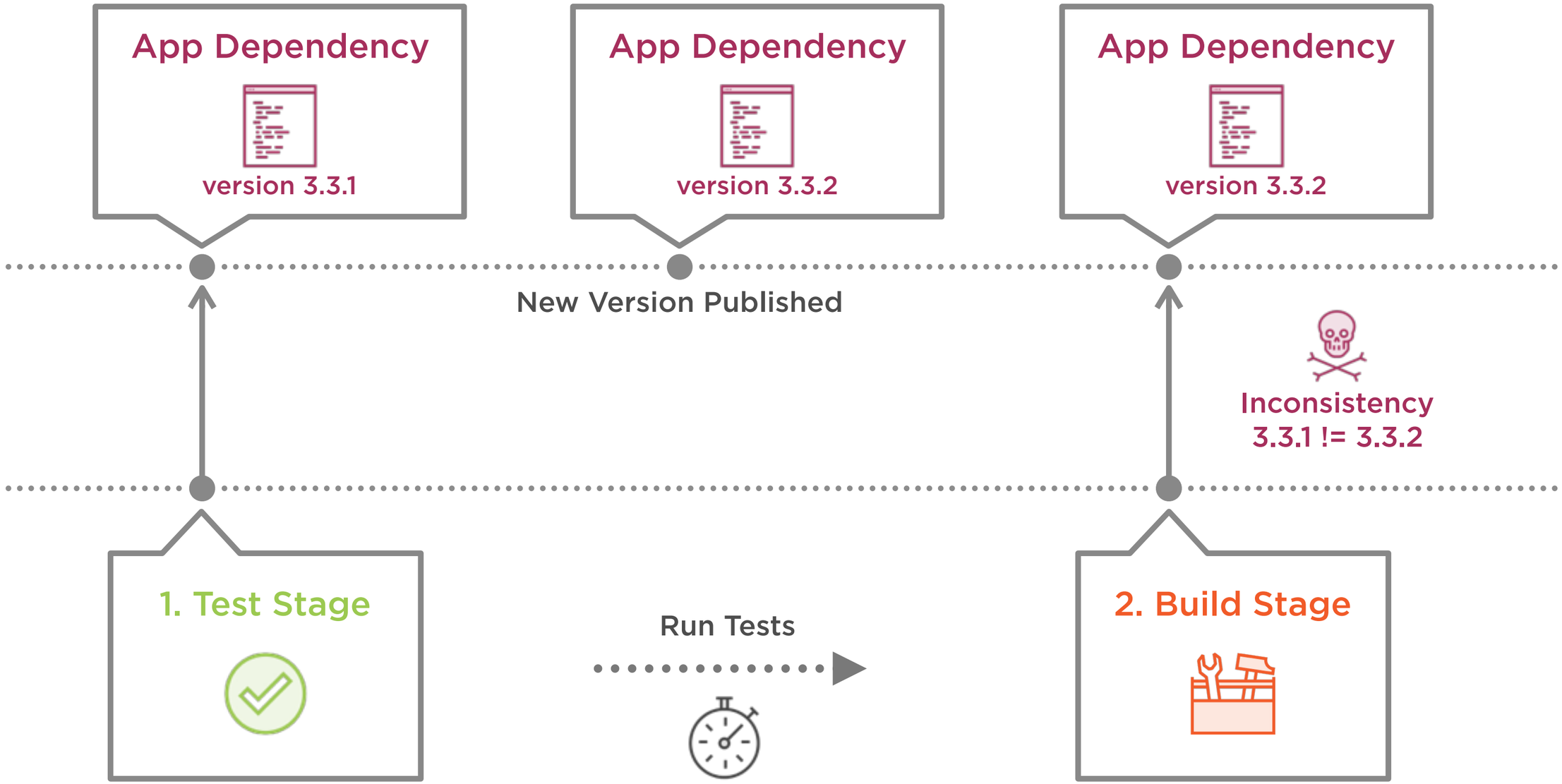


Test and Build Consistency

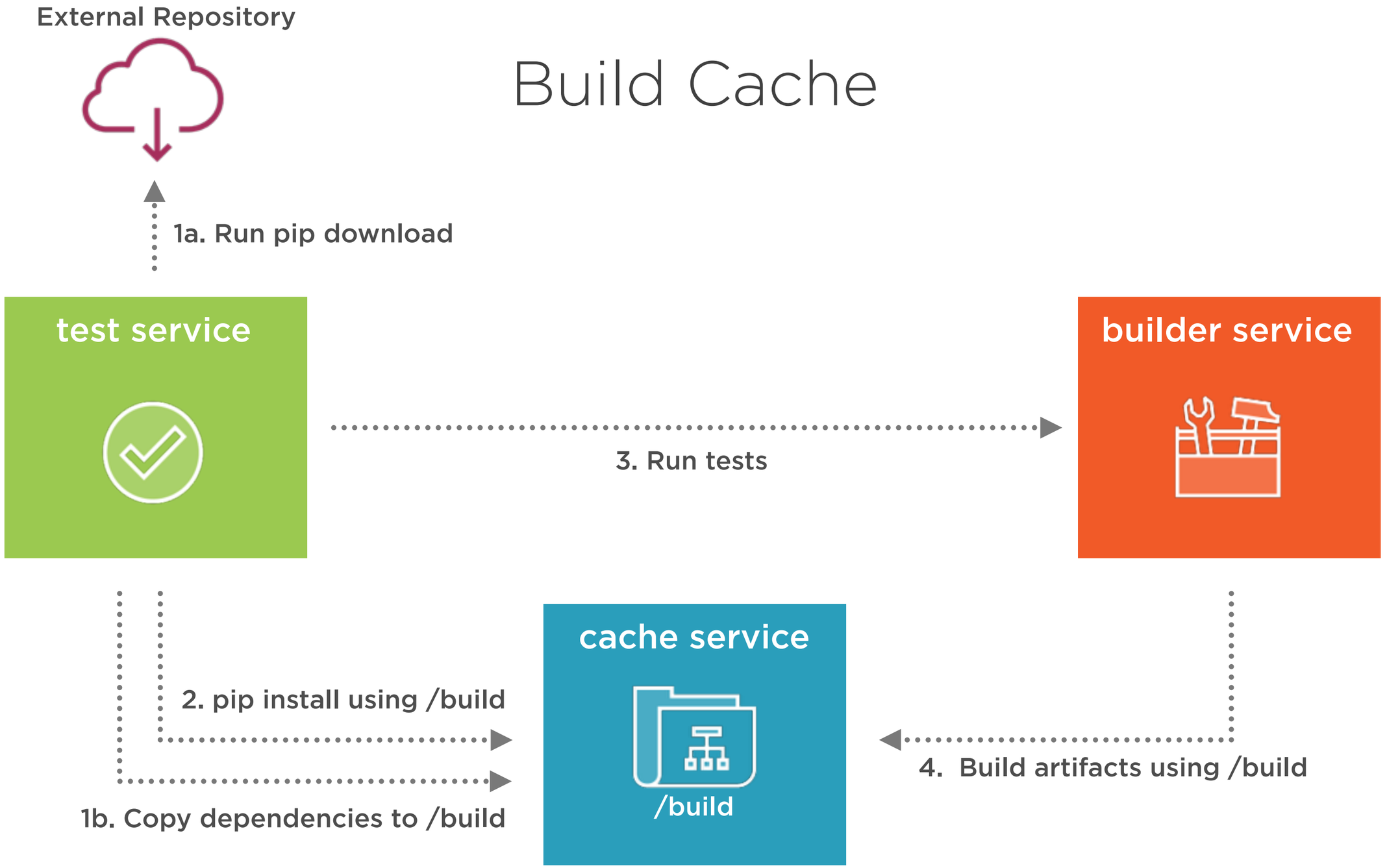
Test and Build Consistency



Test and Build Consistency



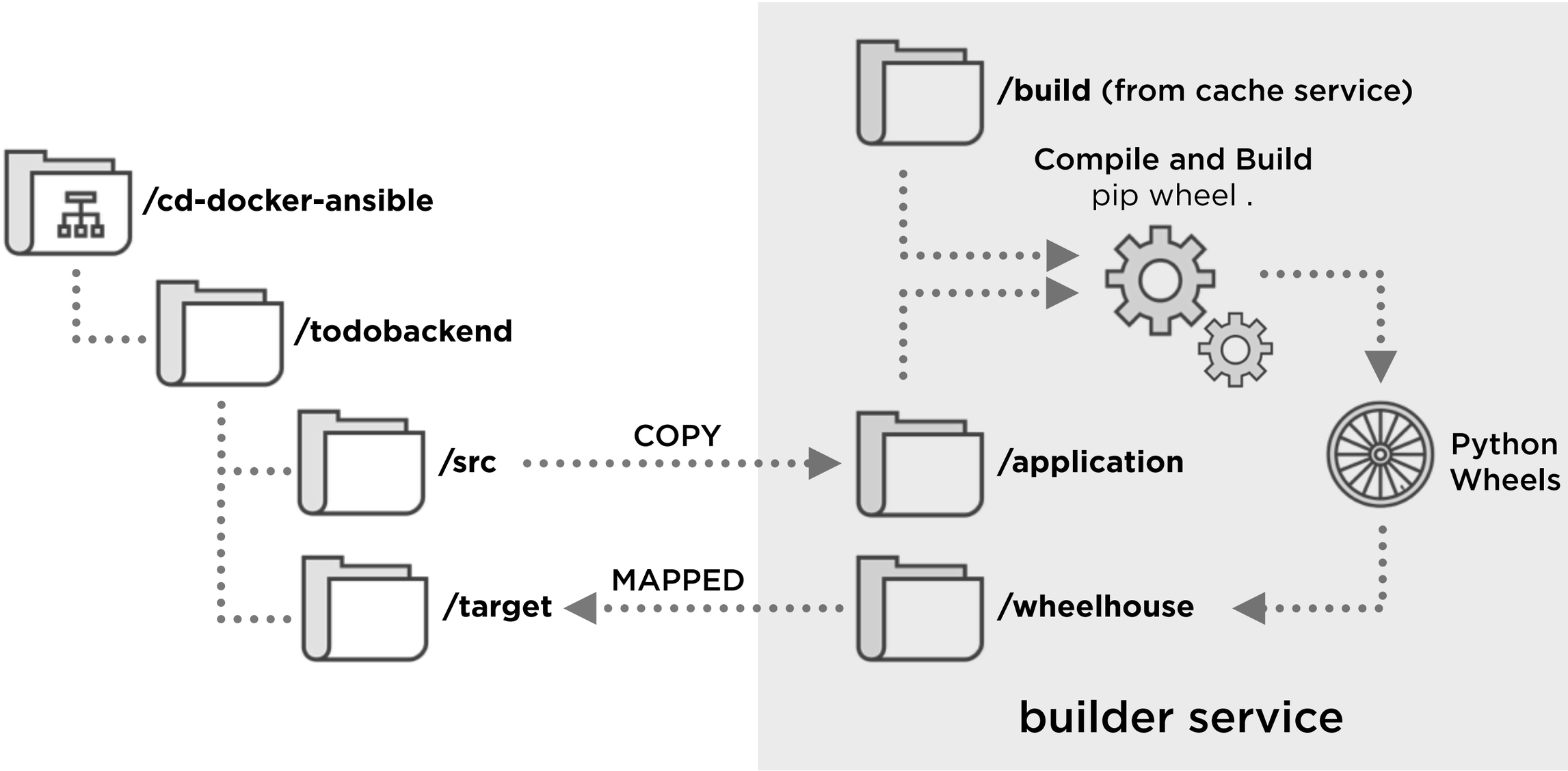
Build Cache


External Repository
1a. Run pip download
test service
builder service
3. Run tests
2. pip install using /build
1b. Copy dependencies to /build
cache service
/build
4. Build artifacts using /build

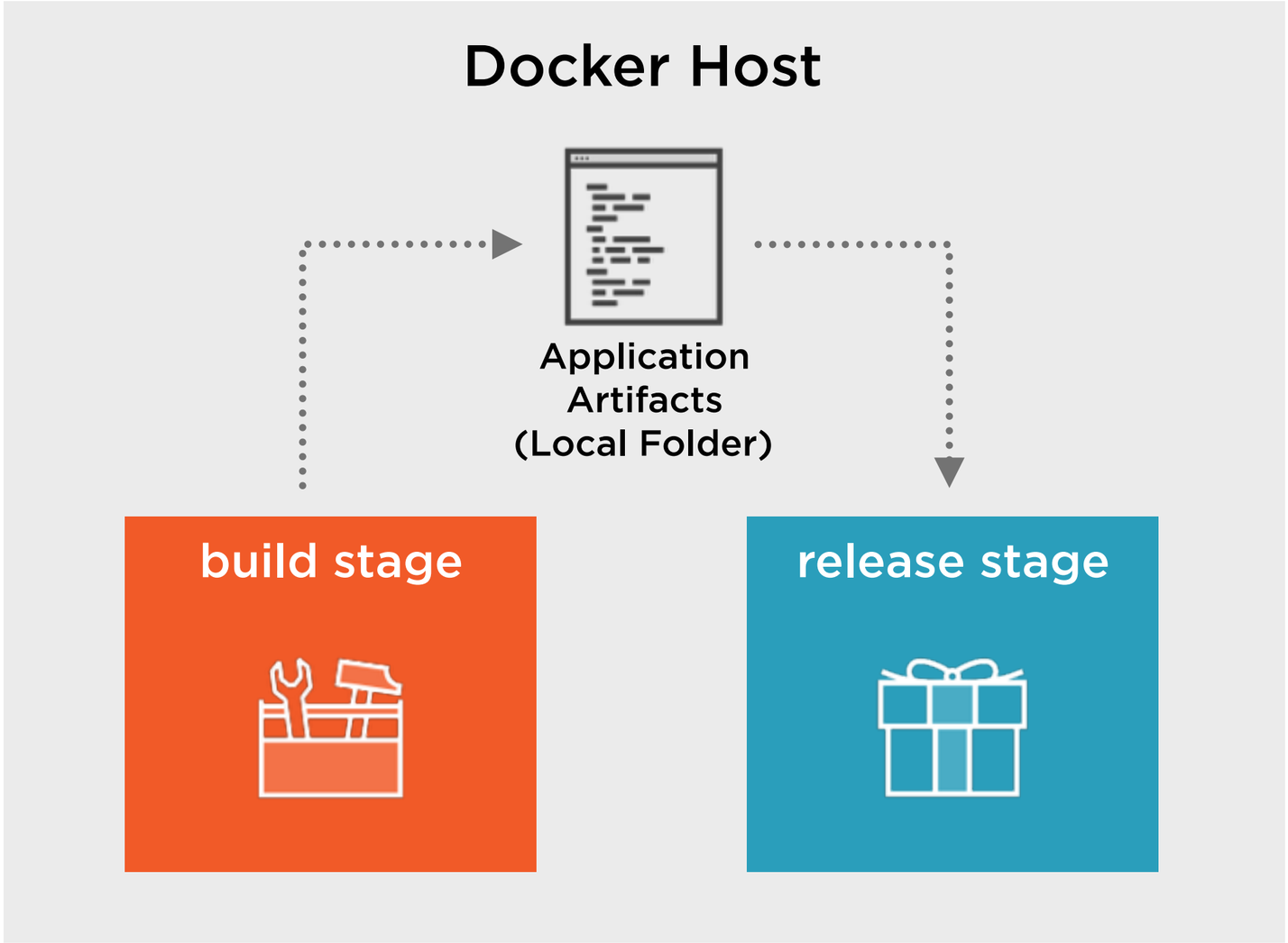
Adding a Builder Service

Building and Publishing Python Wheels

Build Process



Publishing Application Artifacts



Summary

Build Stage

- Builds application artifacts
- Source vs built distributions
- Python wheels
- Builds using test environment
- Test and build consistency
- Publishes locally for later stages