Unit/Integration Testing Using Docker

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

Introduction

Continuous Delivery Workflow

Create a Base Image

- Establish application runtime environment

Create a Development Image

- Add test and build dependencies
- Run Tests

Docker Compose

- Create a complex test environment
- Orchestrate unit and integration tests

Continuous Delivery Workflow

Test

Build

Release

Deploy

Test Workflow Using Docker

Create Test Environment

Base Image
Development Image
Docker Compose

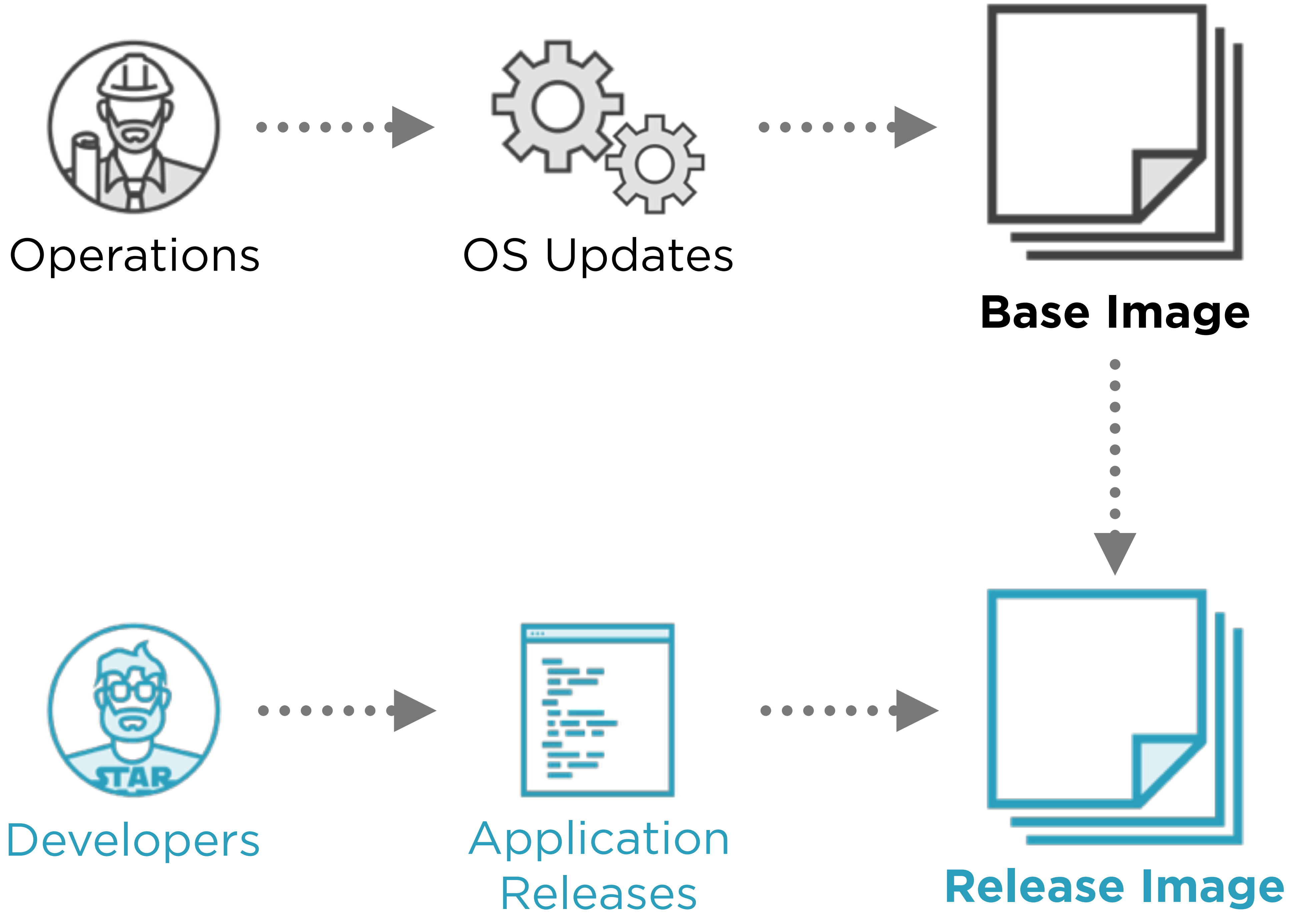
Run Unit Tests

Single Container

Run Integration Tests

Single/Multi Container
Complex Workflow

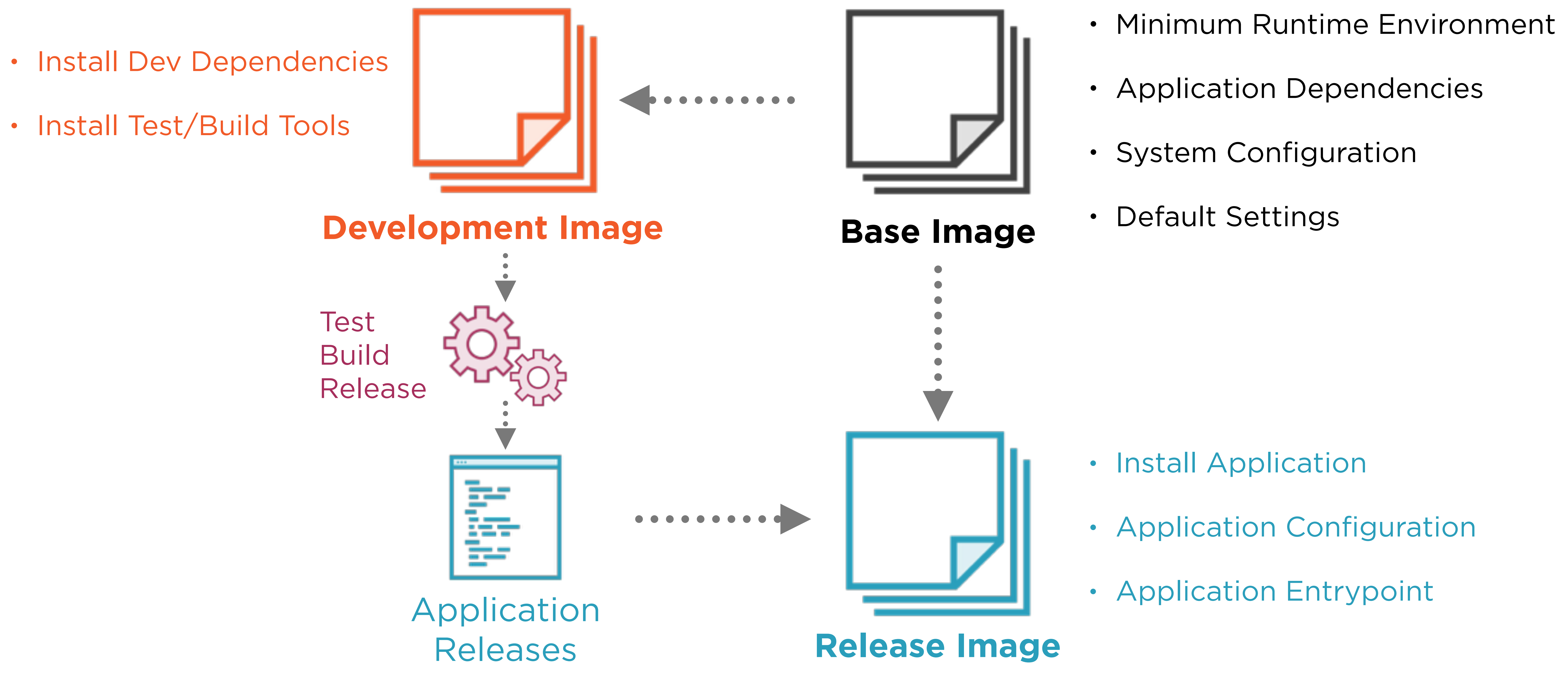
Docker Image Hierarchy



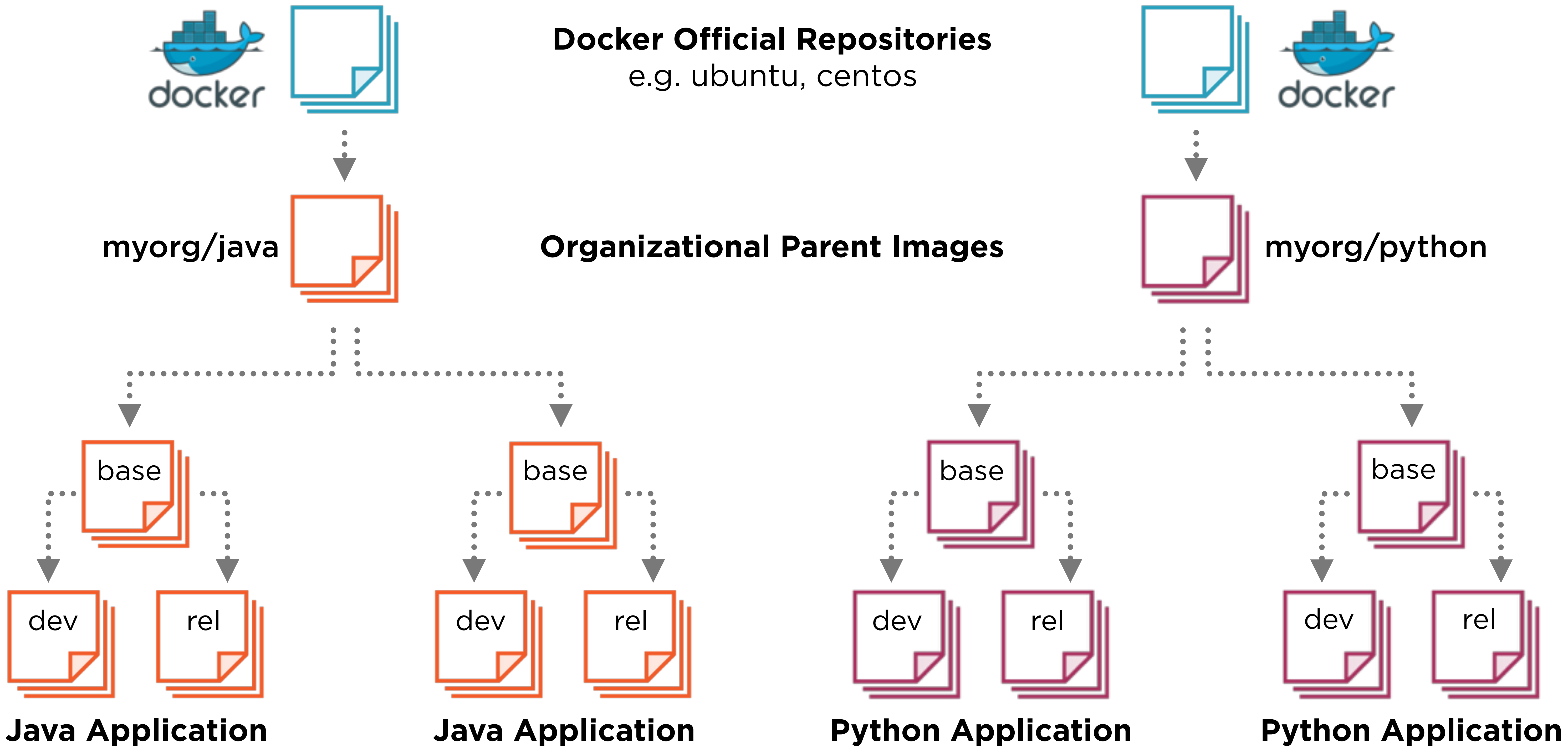
- Minimum Runtime Environment
- Application Dependencies
- System Configuration
- Default Settings

- Install Application
- Application Configuration
- Application Entrypoint

Docker Image Hierarchy



Docker Image Hierarchy



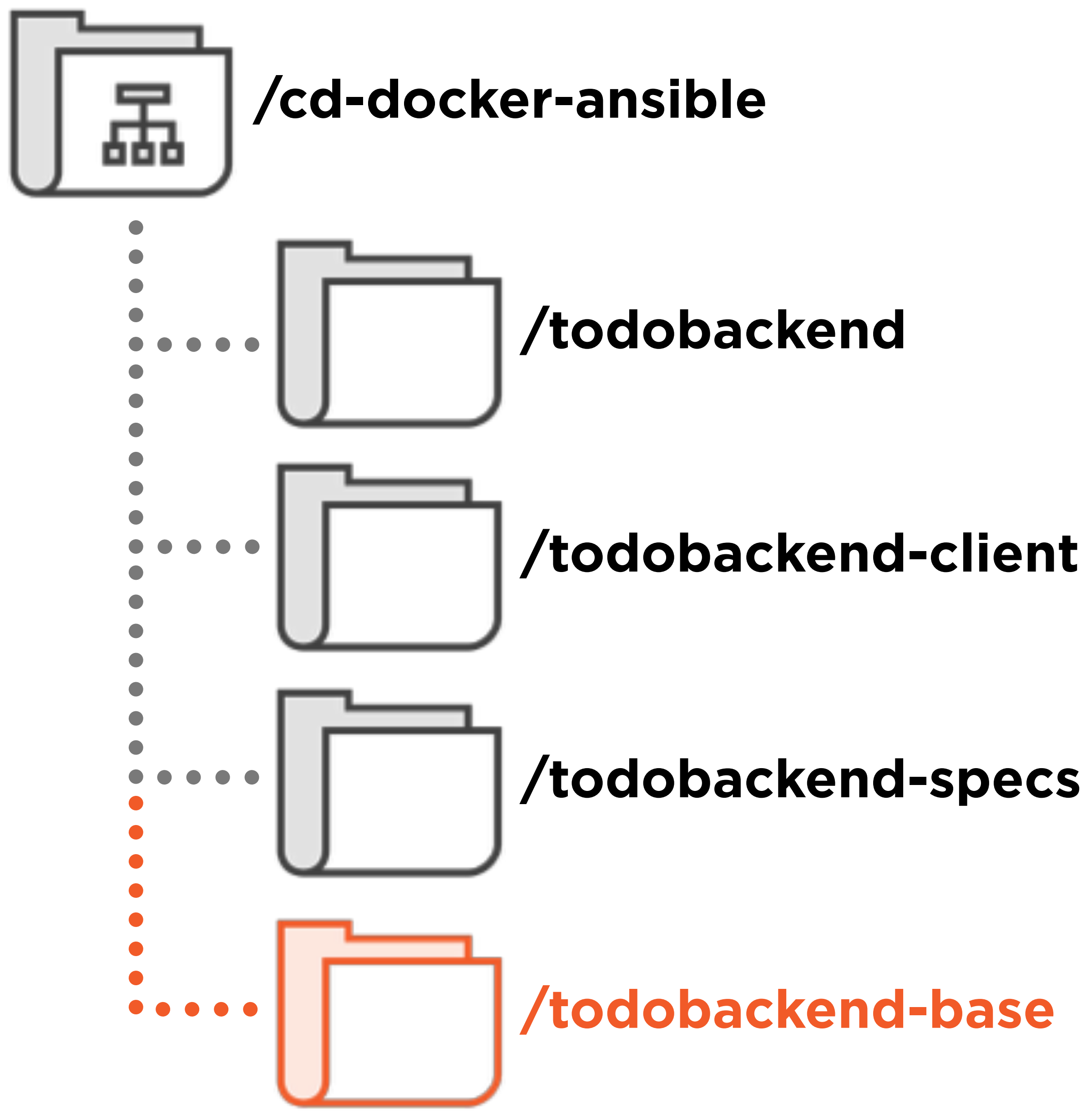
Demo

Creating the Base Image

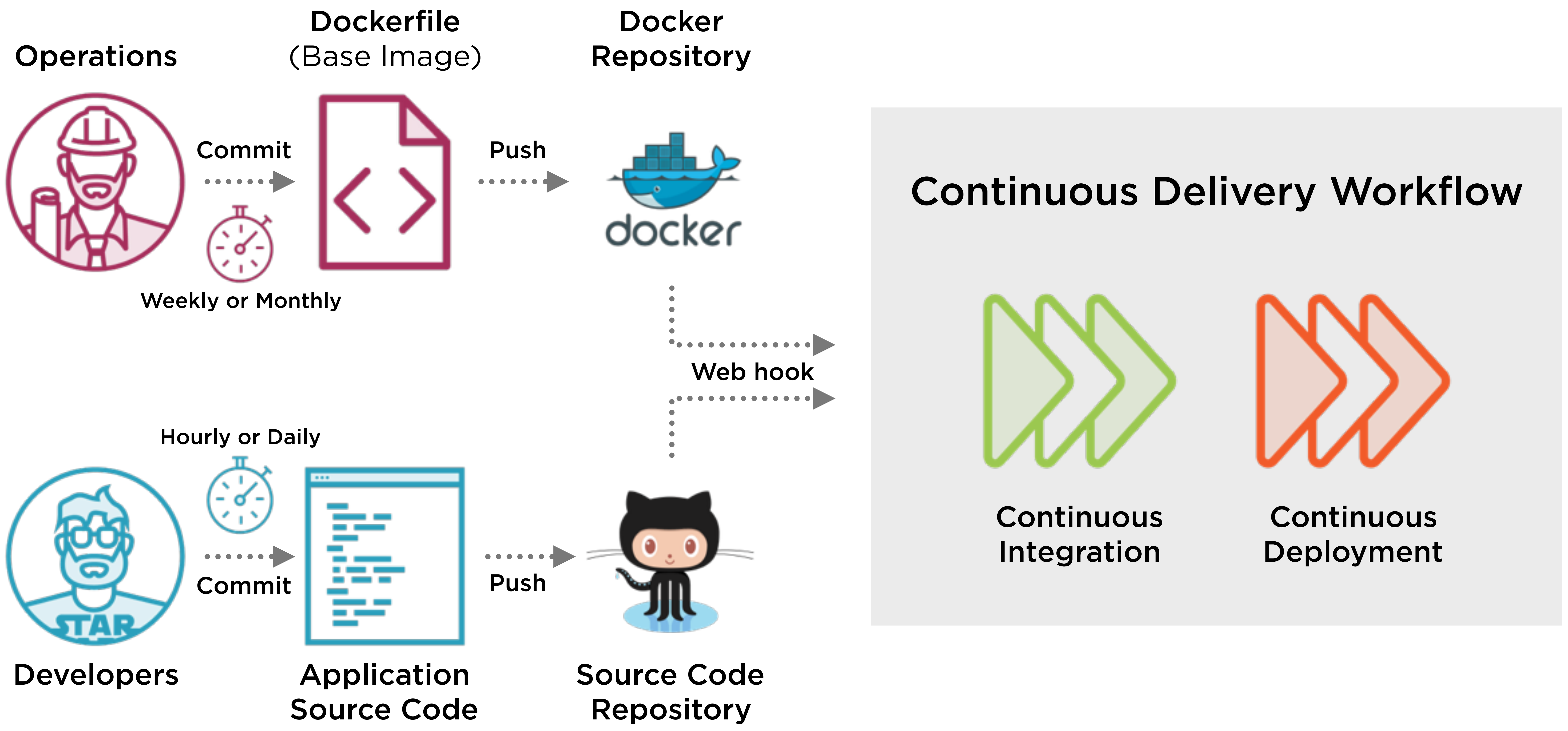
- Initial setup
- Choose parent image
- Describe operating system packages
- Establish the virtual environment
- Building the base image

Initial Setup

Course Folder Structure



Separating the Base Image



Choosing the Parent Image

Describing Operating System Packages

Establishing the Virtual Environment

Activating the Virtual Environment

```
$ . /appenv/bin/activate

$(appenv) python manage.py test
```

◀ Activate virtual environment

◀ Run application or task

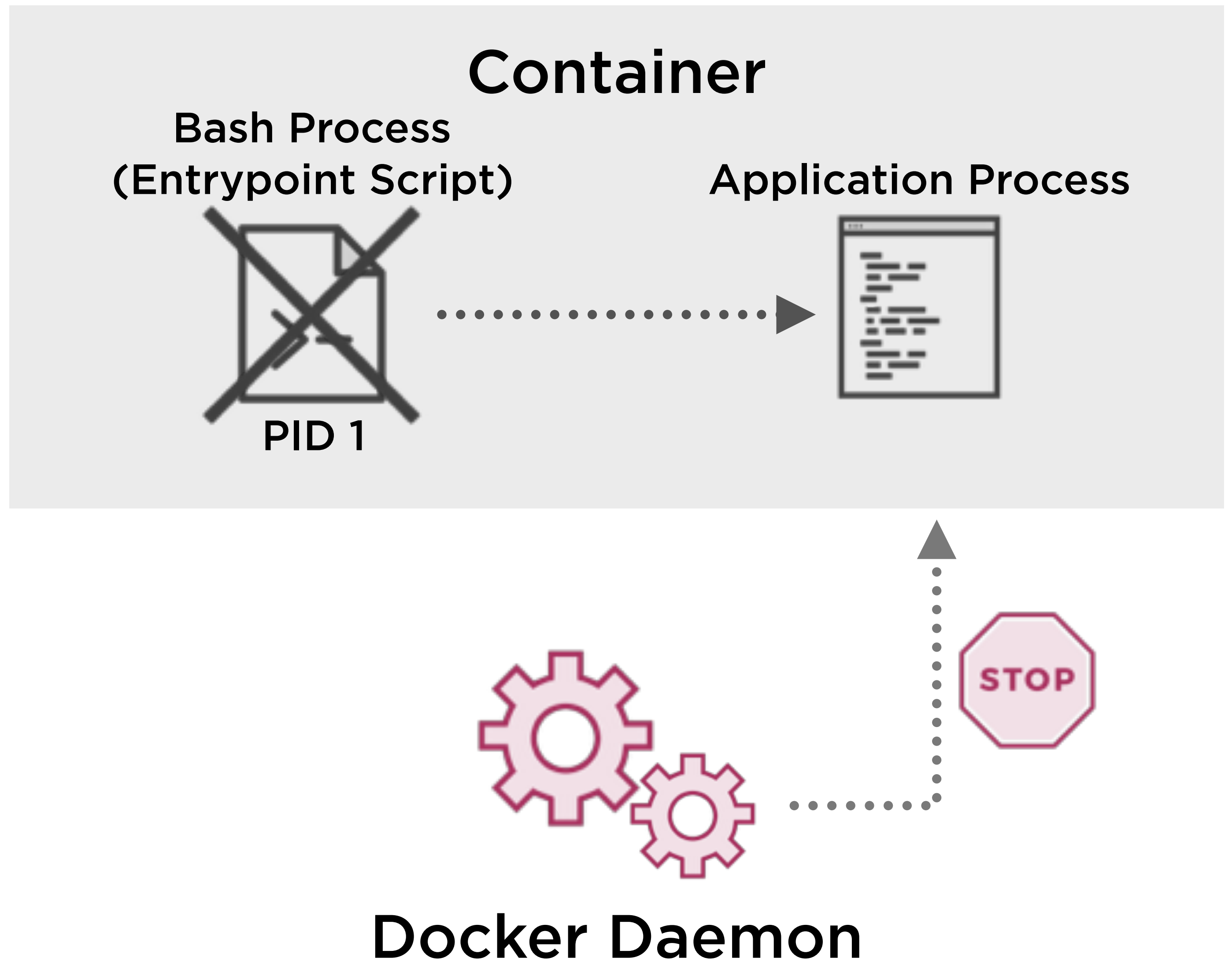
```
ENTRYPOINT Script

1. Activate virtual environment

2. Execute command
```

◀ Docker ENTRYPOINT

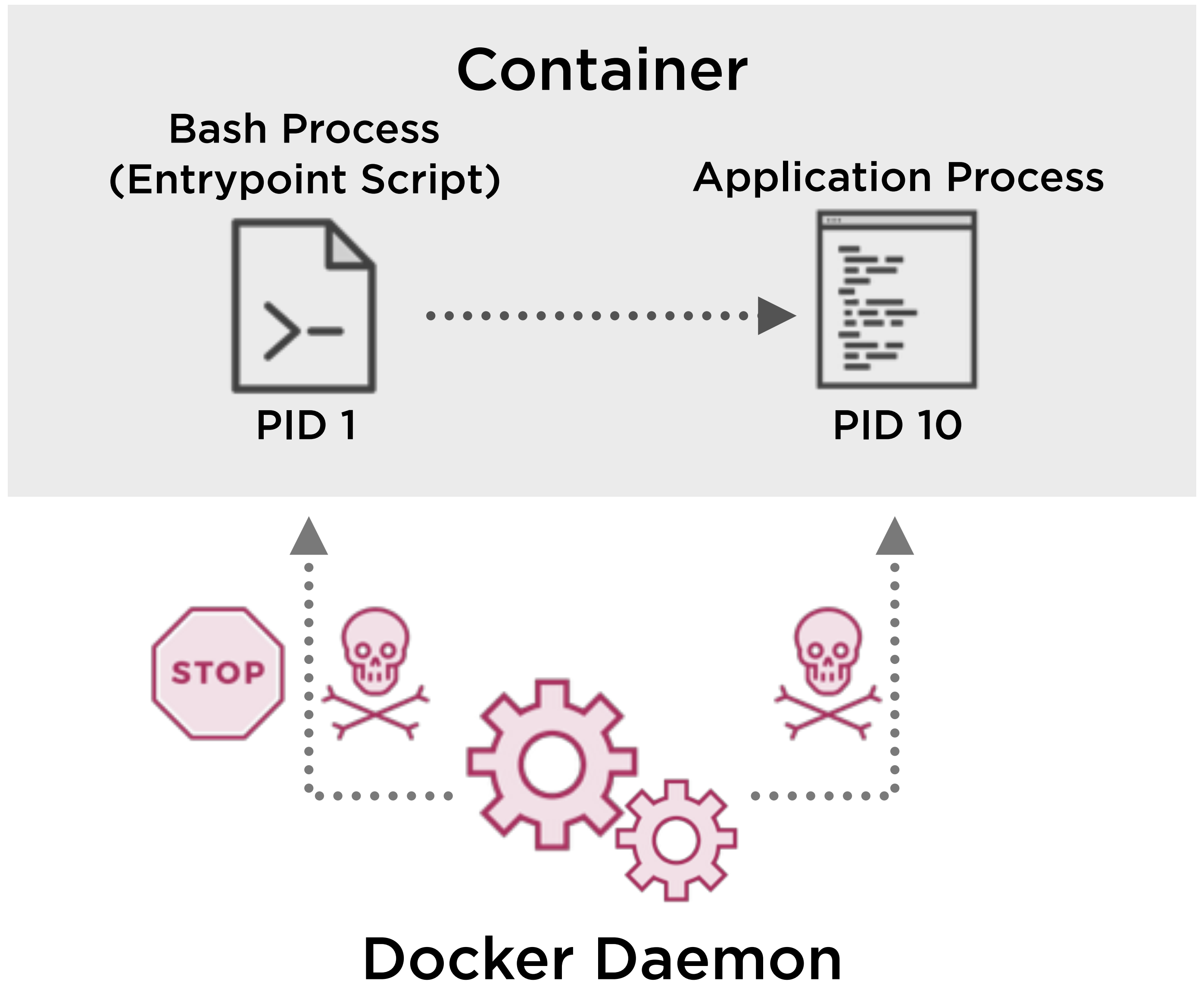
◀ Docker CMD
e.g. python manage.py test

Docker and the Parent Process (PID 1)

With exec



Without exec



Building the Base Image

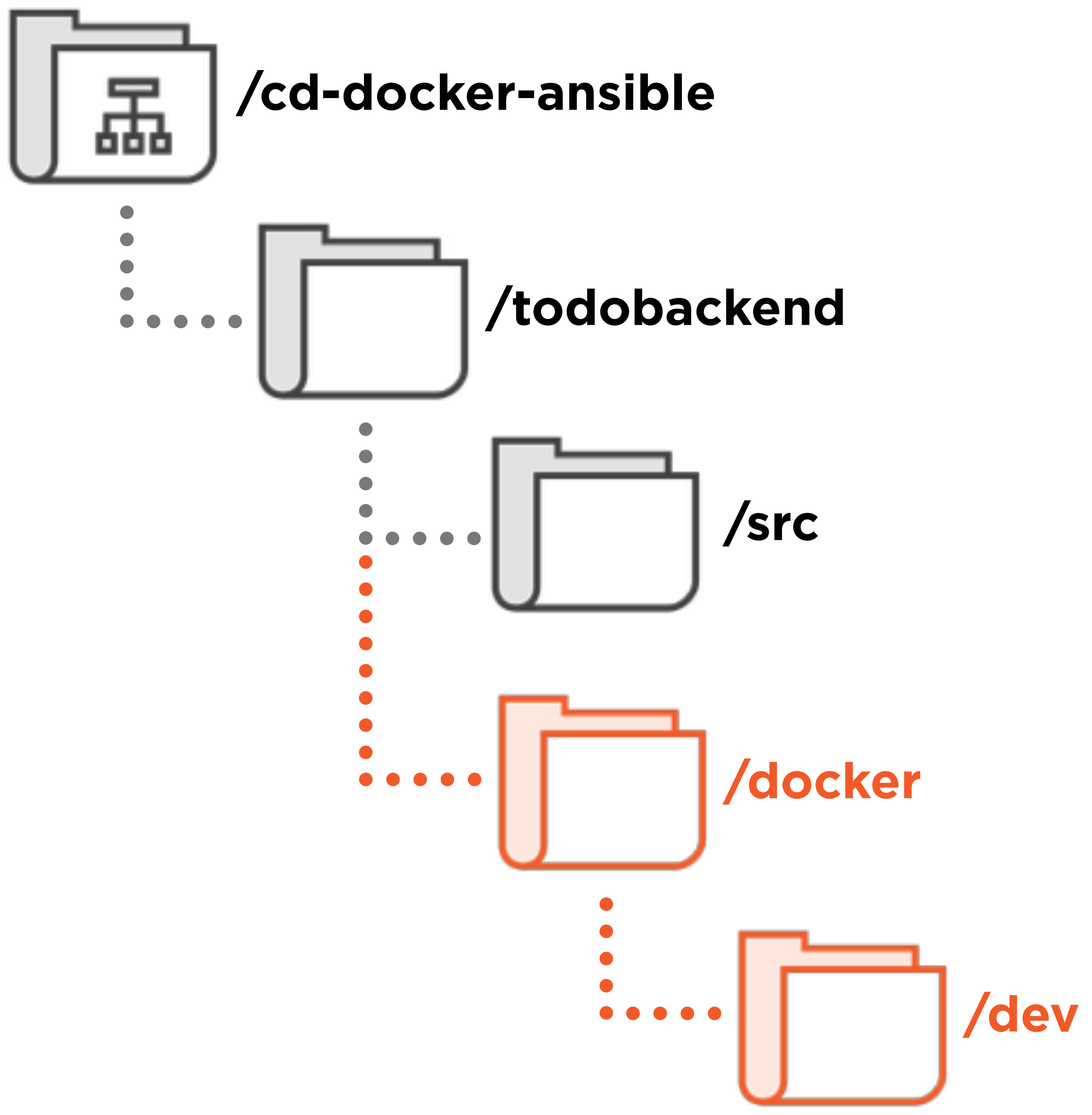
Demo

Creating the Development Image

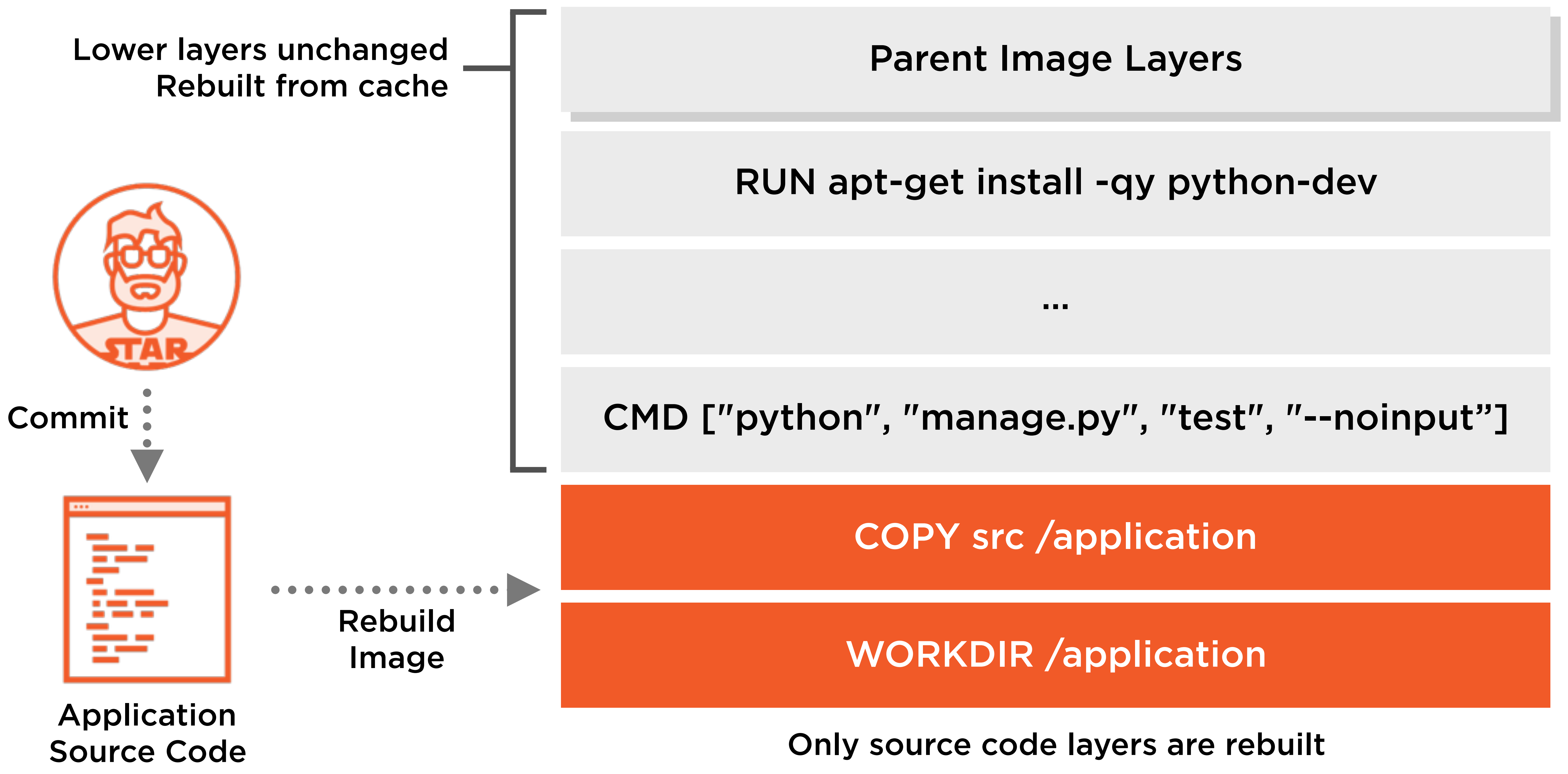
- Building the development image
- Creating application requirements files
- Testing the development image
- Reducing testing time
- Using different test settings

Building the Development Image

Course Folder Structure



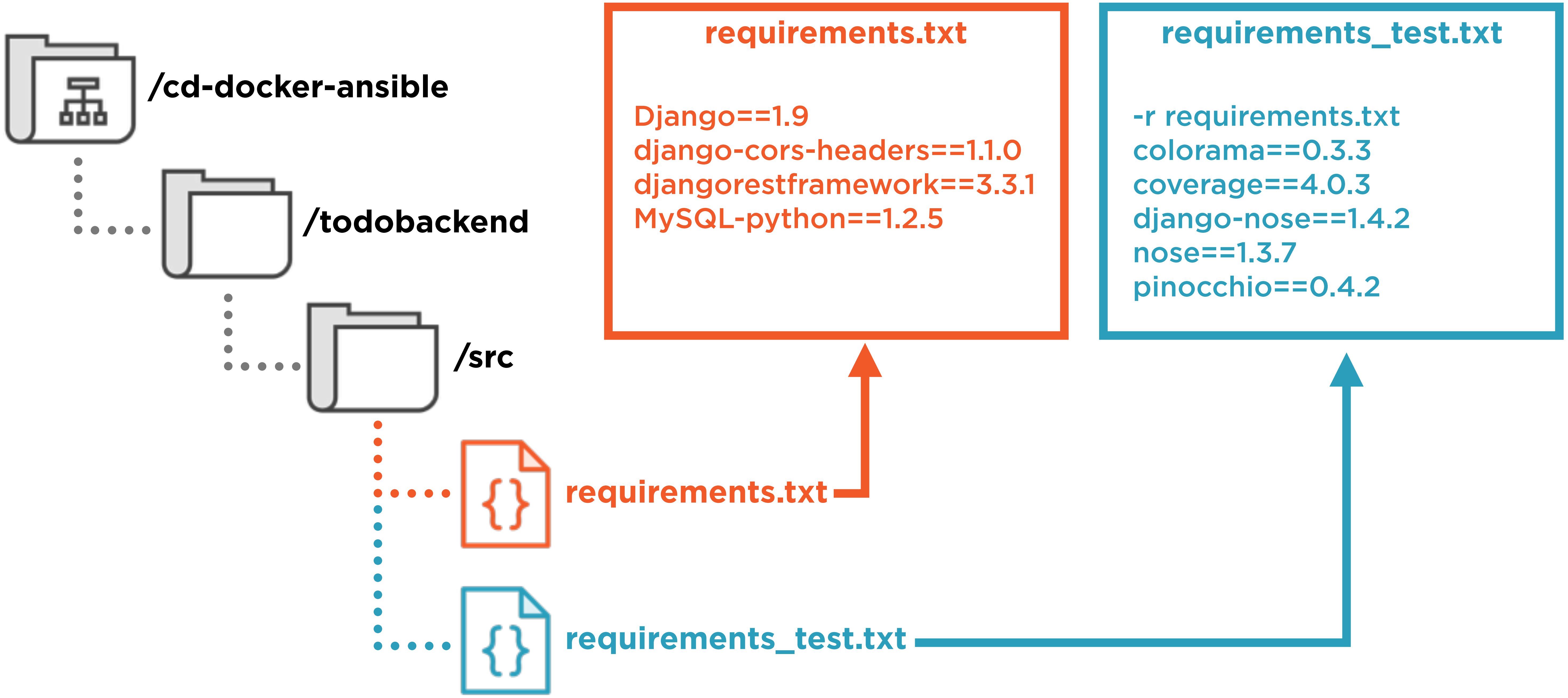
Rebuilding the Development Image



```
Parent Image Layers
RUN apt-get install -qy python-dev
...
CMD ["python", "manage.py", "test", "--noinput”]
COPY src /application
WORKDIR /application
```

Creating Application Requirements Files

Application Requirements Files



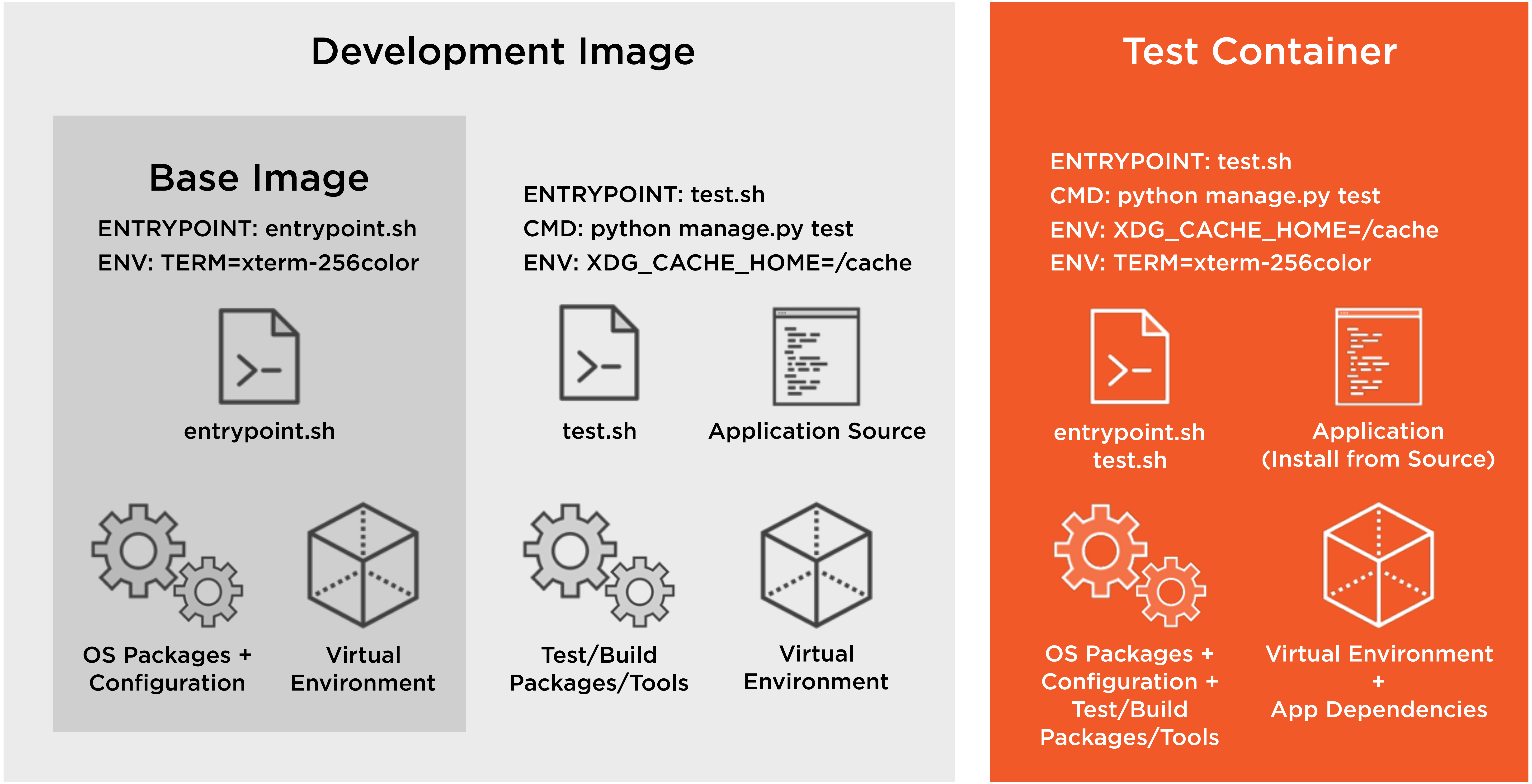
requirements.txt

```
Django==1.9
django-cors-headers==1.1.0
djangorestframework==3.3.1
MySQL-python==1.2.5
```

requirements_test.txt

```
-r requirements.txt
colorama==0.3.3
coverage==4.0.3
django-nose==1.4.2
nose==1.3.7
pinocchio==0.4.2
```

Development Image Review



Development Image

Base Image

```
ENTRYPOINT: entrypoint.sh
ENV: TERM=xterm-256color
```

```
ENTRYPOINT: test.sh
CMD: python manage.py test
ENV: XDG_CACHE_HOME=/cache
```

Test Container

```
ENTRYPOINT: test.sh
CMD: python manage.py test
ENV: XDG_CACHE_HOME=/cache
ENV: TERM=xterm-256color
```

entrypoint.sh
OS Packages + Configuration
Virtual Environment
test.sh
Application Source
Test/Build Packages/Tools
Virtual Environment
entrypoint.sh test.sh
Application (Install from Source)
OS Packages + Configuration + Test/Build Packages/Tools
Virtual Environment + App Dependencies

```
docker run todobackend-dev
```

test.sh

```
#!/bin/bash
# Activate virtual environment
. /appenv/bin/activate

# Install application test requirements
pip install -r requirements_test.txt

# Run test.sh arguments
exec $@
```

Test Container

ENTRYPOINT: test.sh
CMD: python manage.py test
ENV: XDG_CACHE_HOME=/cache
ENV: TERM=xterm-256color

entrypoint.sh
test.sh

Application
(Install from Source)

OS Packages +
Configuration +
Test/Build
Packages/Tools

Virtual Environment
+
App Dependencies

docker run todobackend-dev

Development Image

Base Image

ENTRYPOINT: entrypoint.sh
ENV: TERM=xterm-256color

entrypoint.sh

OS Packages + Configuration

Virtual Environment

ENTRYPOINT: test.sh
CMD: python manage.py test
ENV: XDG_CACHE_HOME=/cache

test.sh

Application Source

Test/Build Packages/Tools

Virtual Environment

Test Container

ENTRYPOINT: test.sh
CMD: python manage.py test
ENV: XDG_CACHE_HOME=/cache
ENV: TERM=xterm-256color

entrypoint.sh
test.sh

Application
(Install from Source)

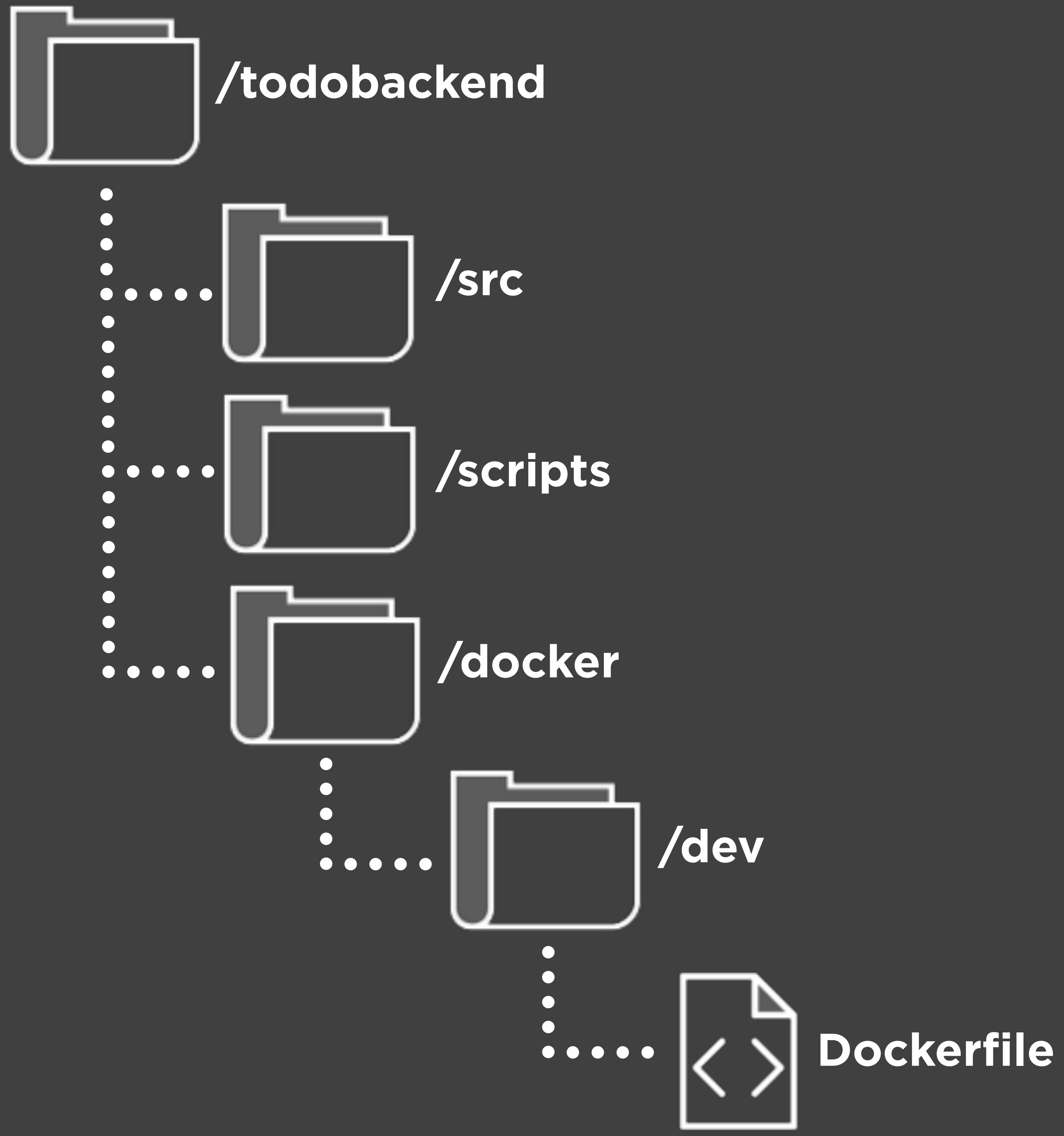
OS Packages +
Configuration +
Test/Build
Packages/Tools

Virtual Environment
+
App Dependencies

```
docker run todobackend-dev
```

Testing the Development Image

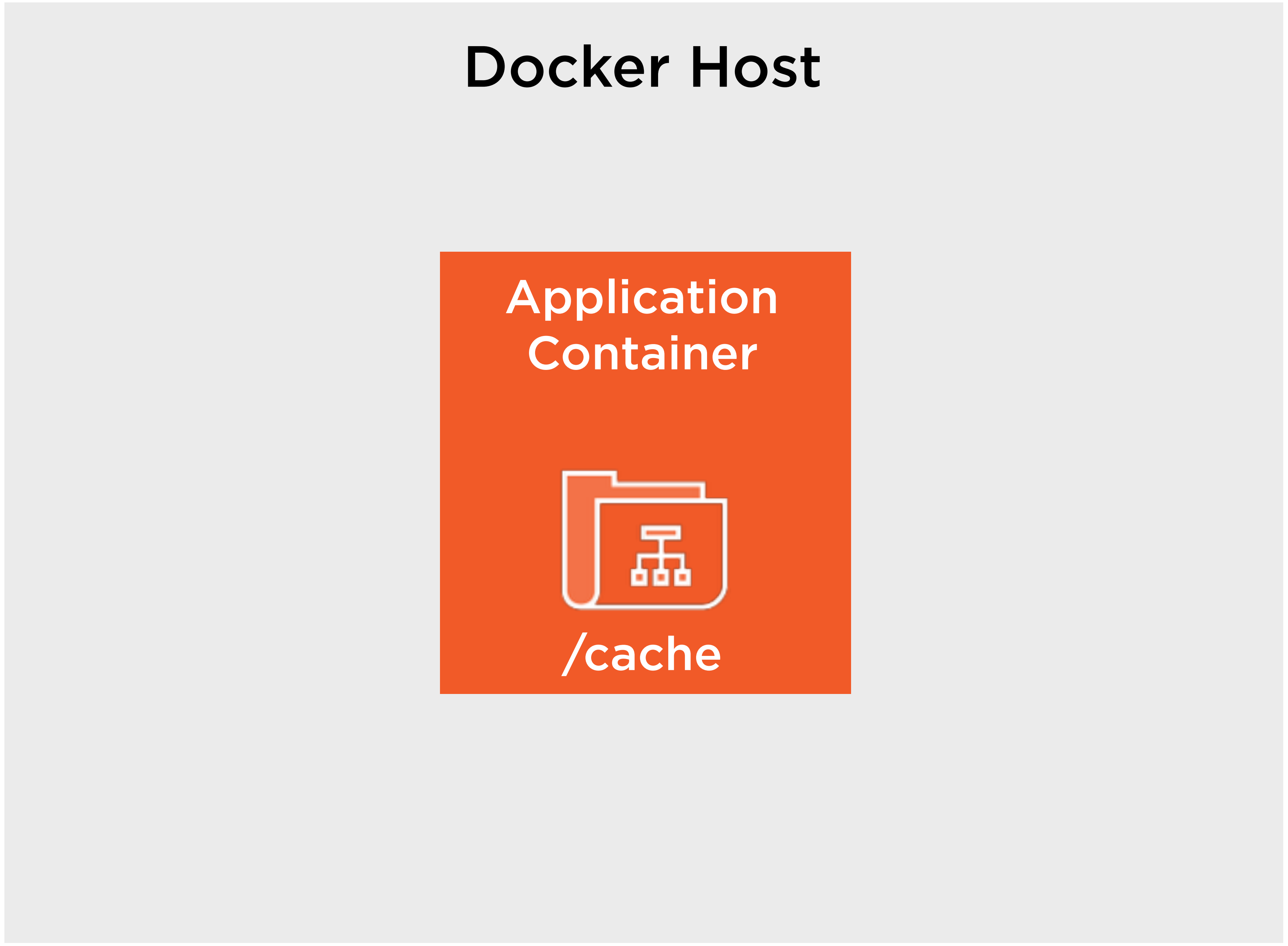
Docker Build Context



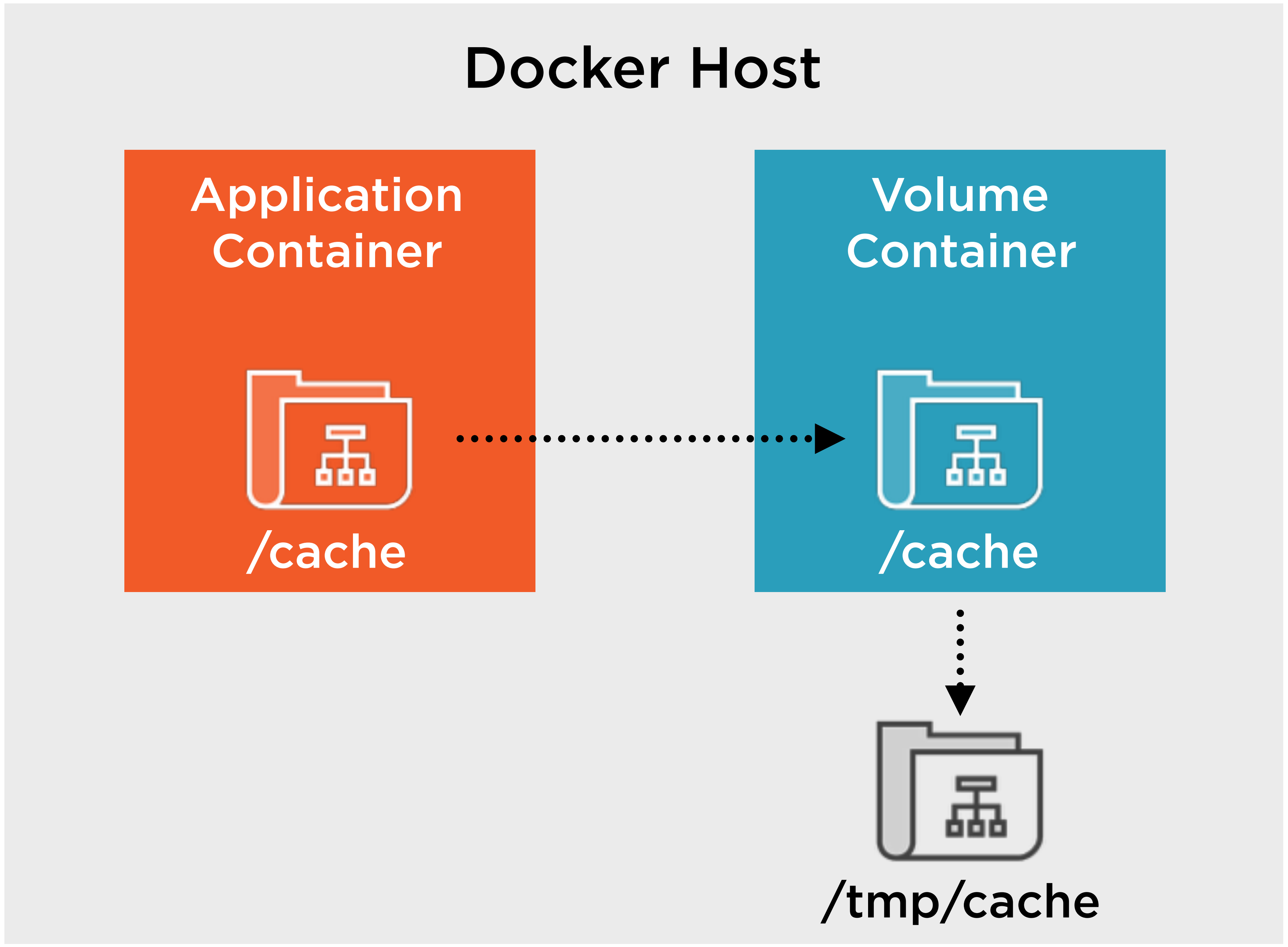
- Build context root (**docker build** must be run from here)
- Application source code
- Development image entrypoint script
- Development image Dockerfile (docker/dev/Dockerfile)

Reducing Testing Time

Volume Containers

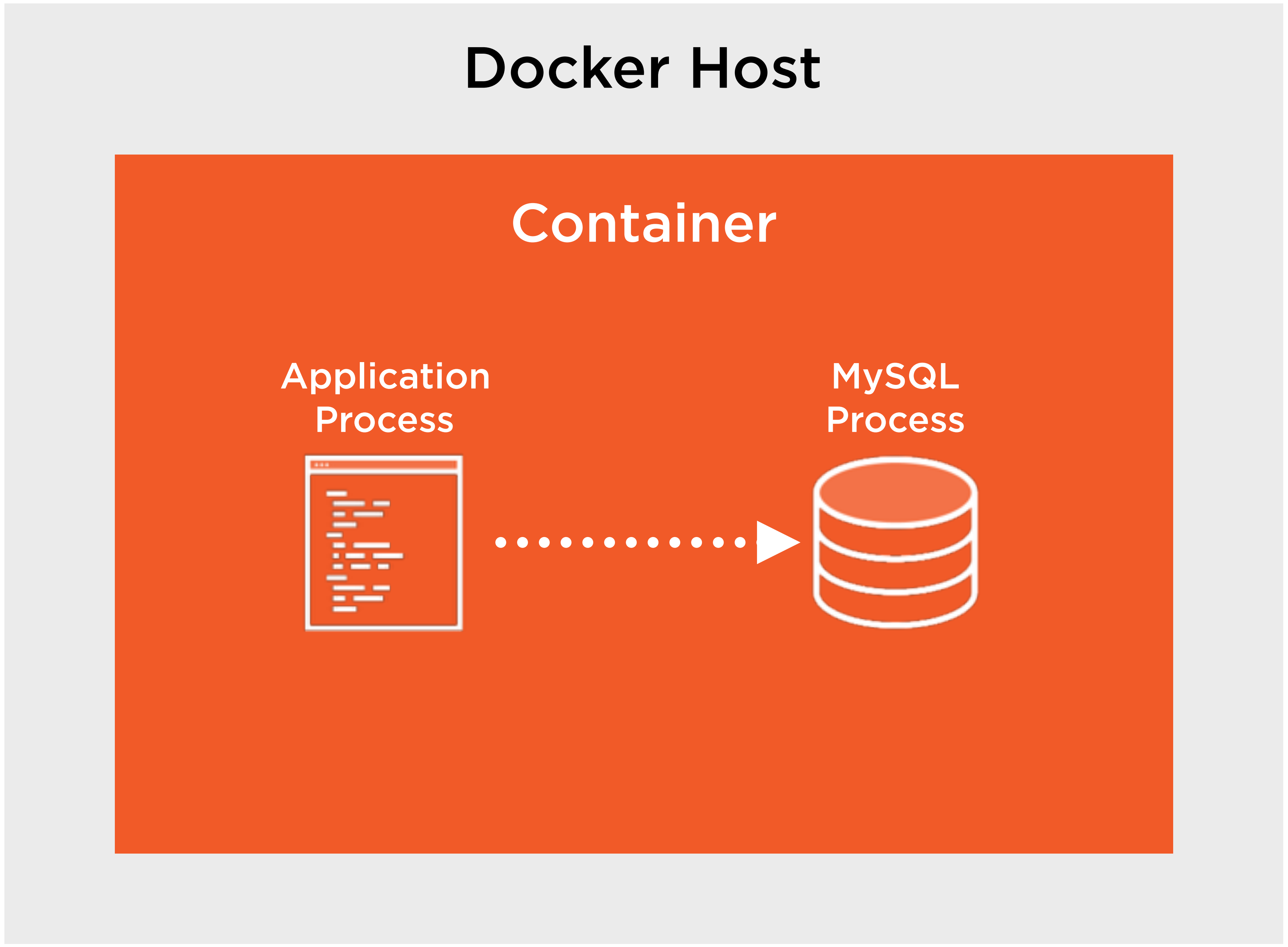


Volume Containers

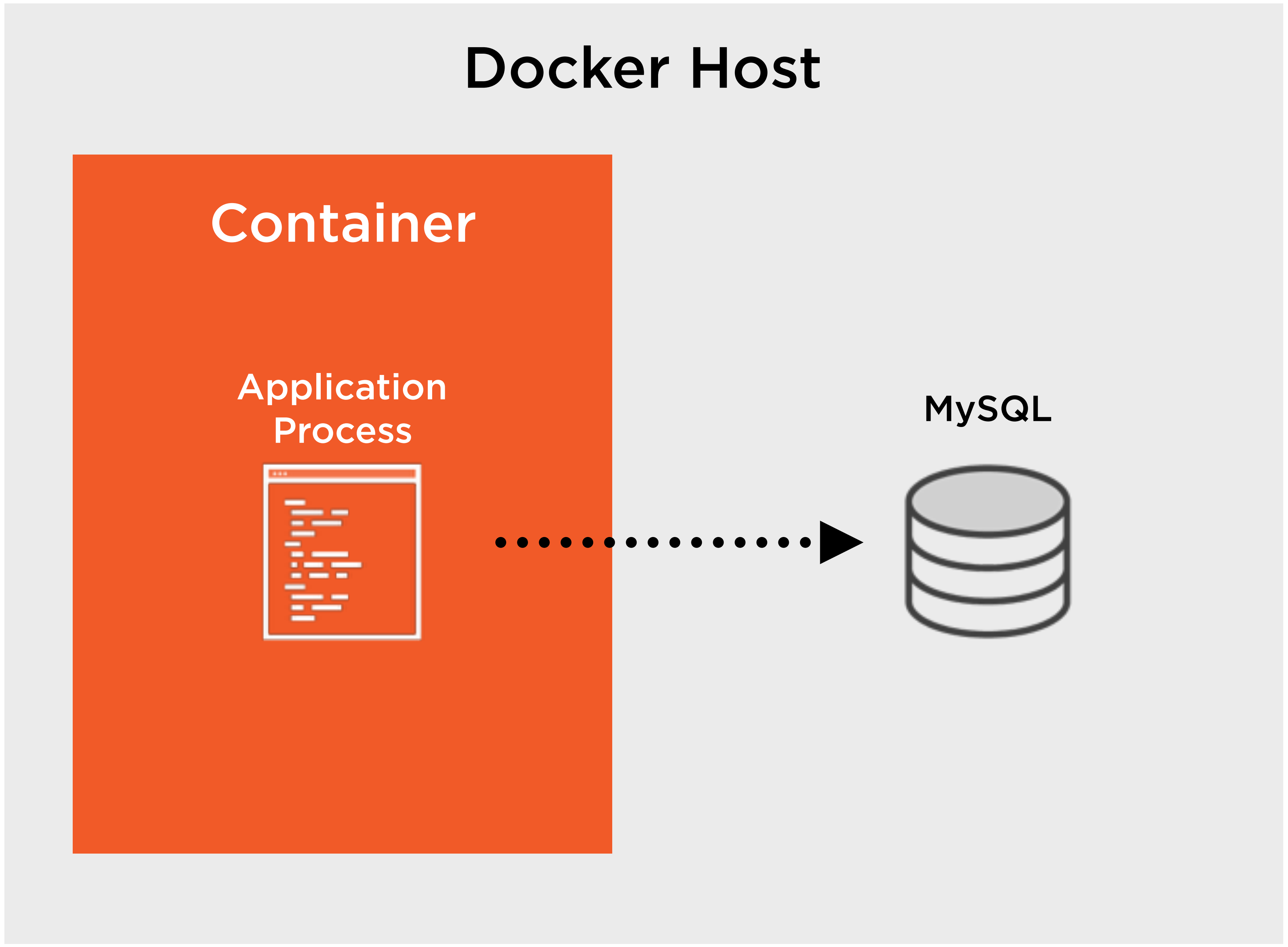


Using Different Test Settings

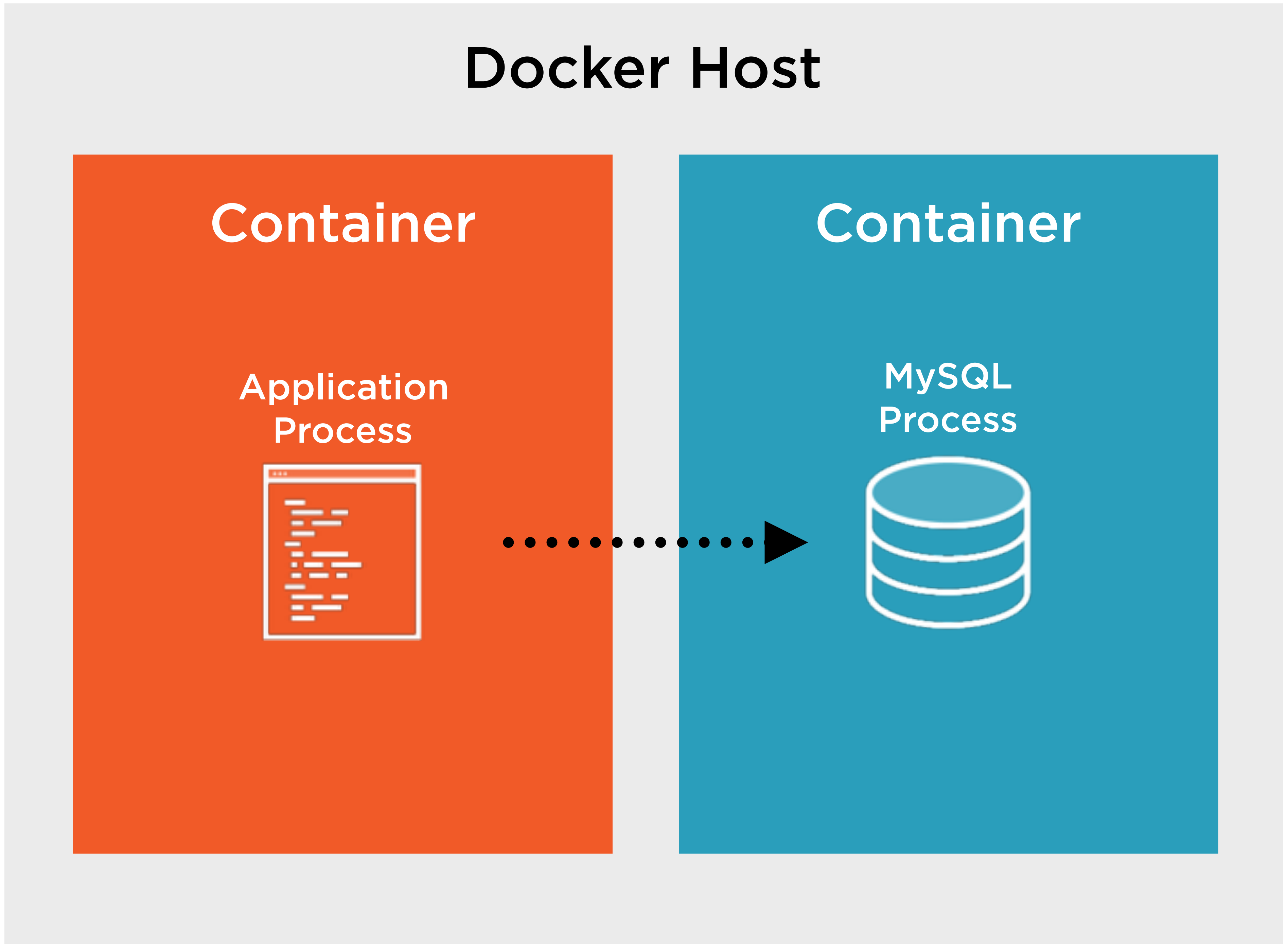
Connecting to MySQL - Single Container



Connecting to MySQL - Docker Host



Connecting to MySQL - Multi Container

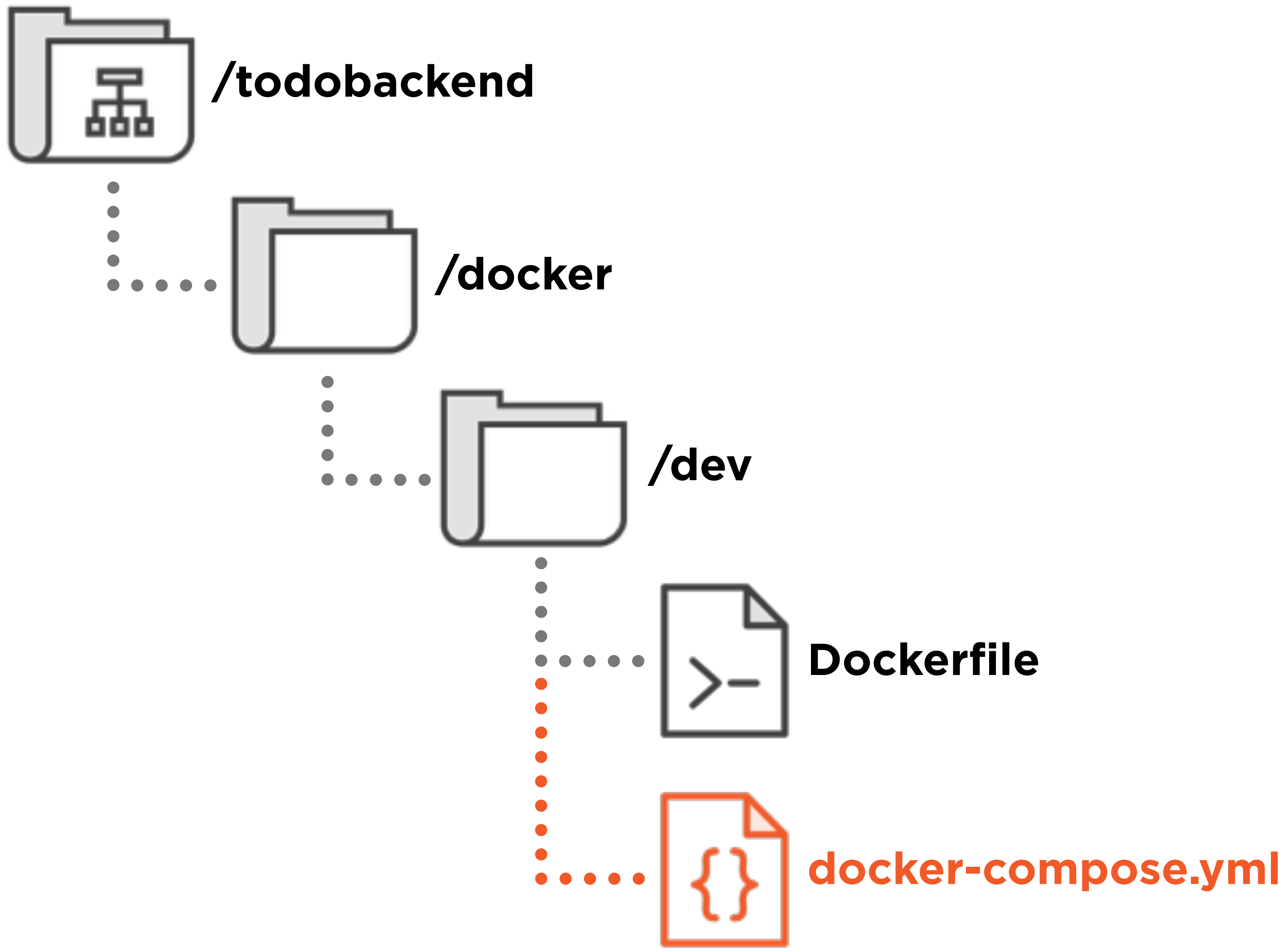


Demo

Creating a Multi-Container Environment using Docker Compose

- Creating a Docker Compose file
- Running tests using Docker Compose
- Solving how to wait for a dependent service to initialize

Docker Compose File



docker-compose.yml

```
app:
  image: myorg/myrepo:latest
  links:
    - db
  volumes:
    - /path/to/host:/path
  volumes_from:
    - cache
  environment:
    MYSQL_DB: todobackend
  ...

db:
  image: mysql
  ...
```

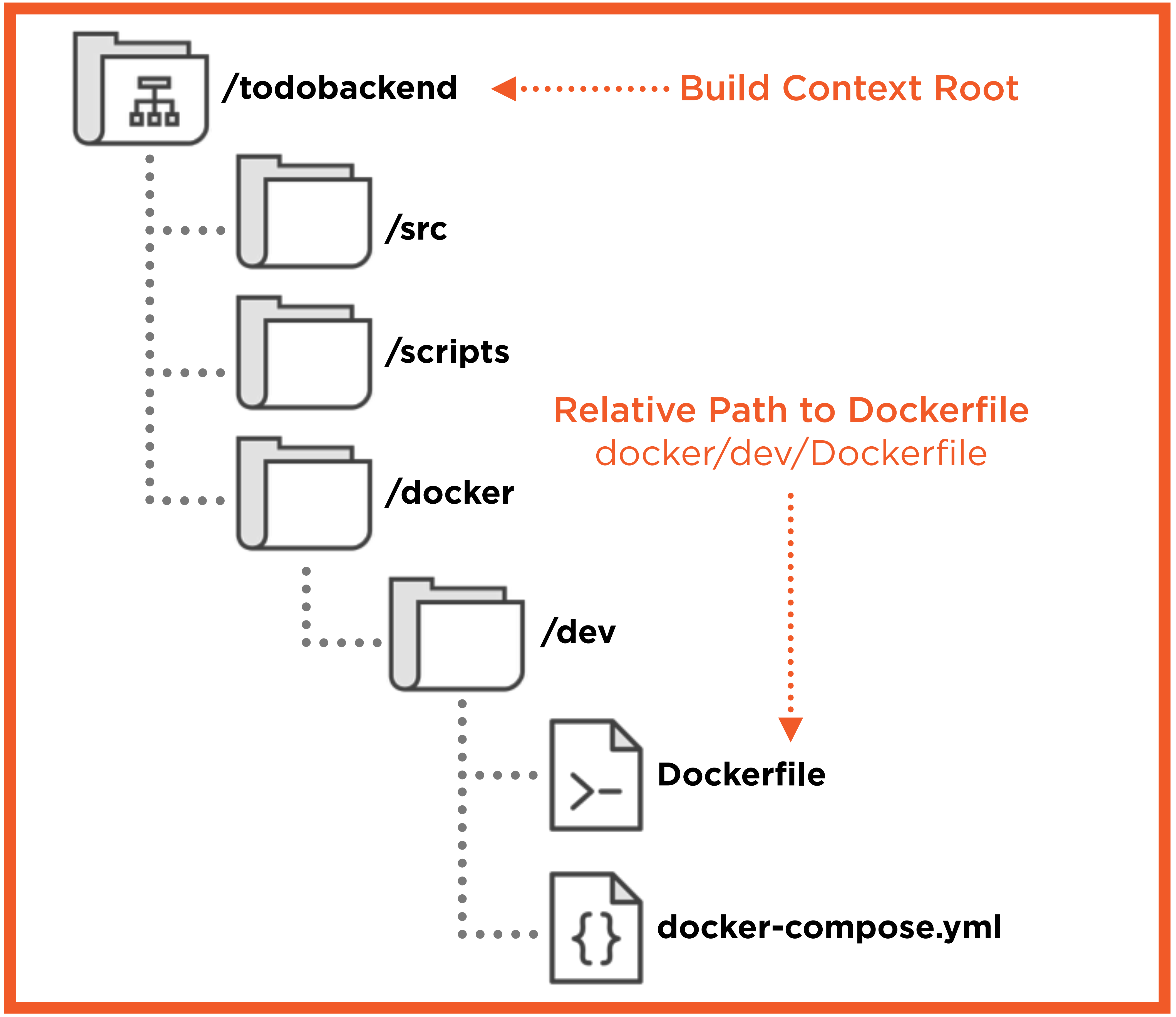
◀ "app" service (aka container)
◀ Image the service is based from
◀ List of service dependencies

◀ List of volumes to mount

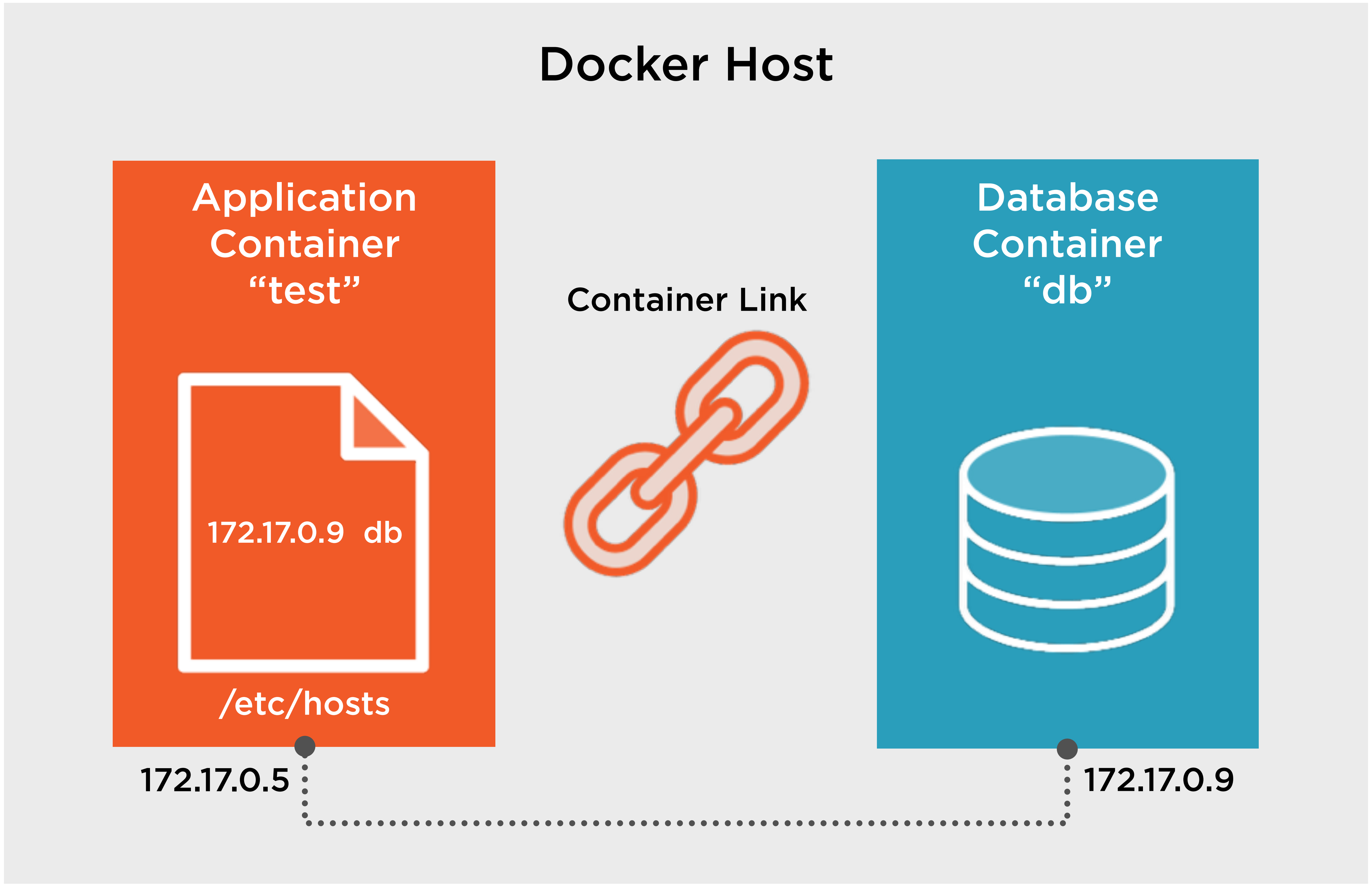
◀ Volume containers to attach

◀ Environment variables

◀ "db" service
(another container)



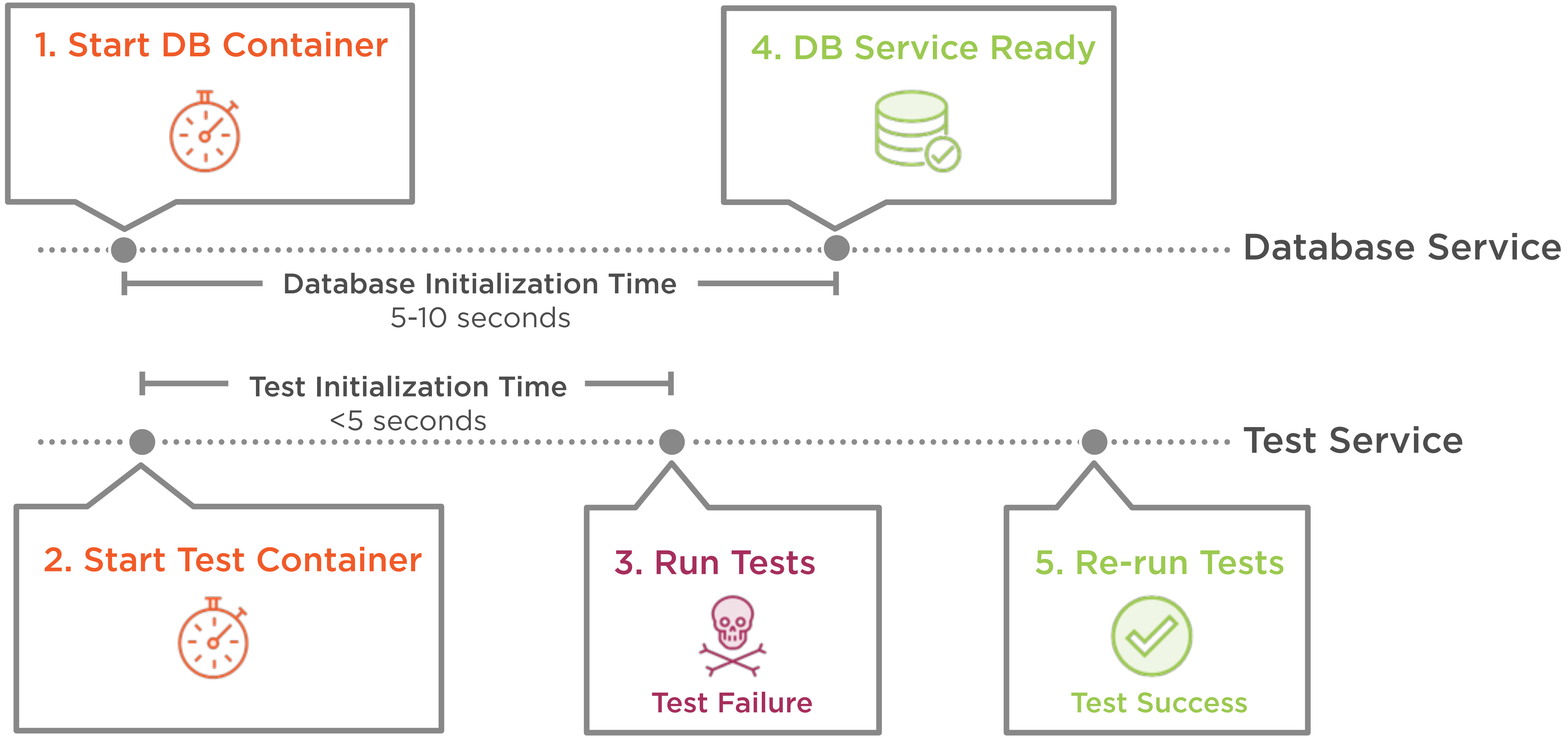
Build Context


Docker Host
Application Container "test"
172.17.0.9 db
/etc/hosts
172.17.0.5
Container Link
Database Container "db"
172.17.0.9

Creating a Docker Compose File

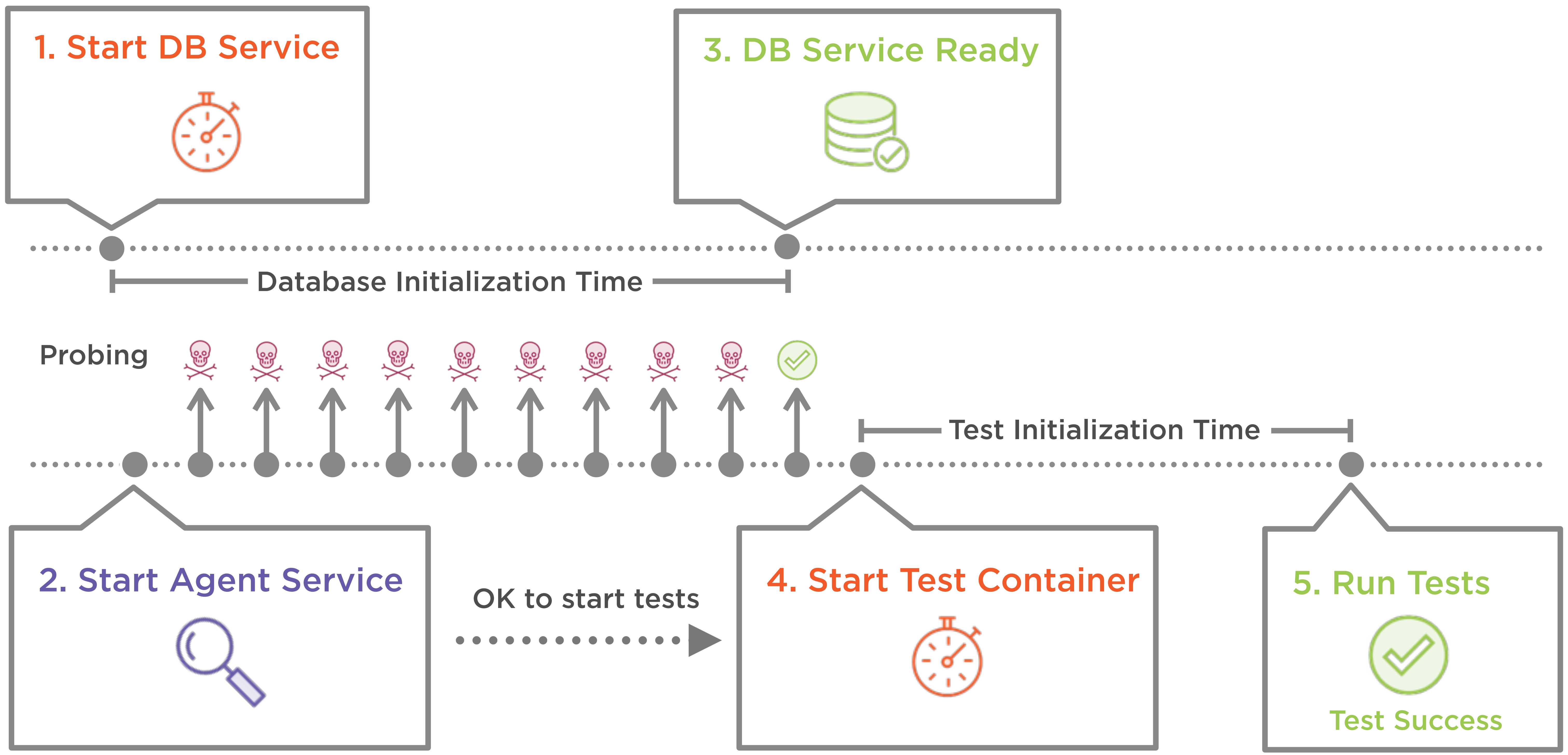
Running Tests Using Docker Compose

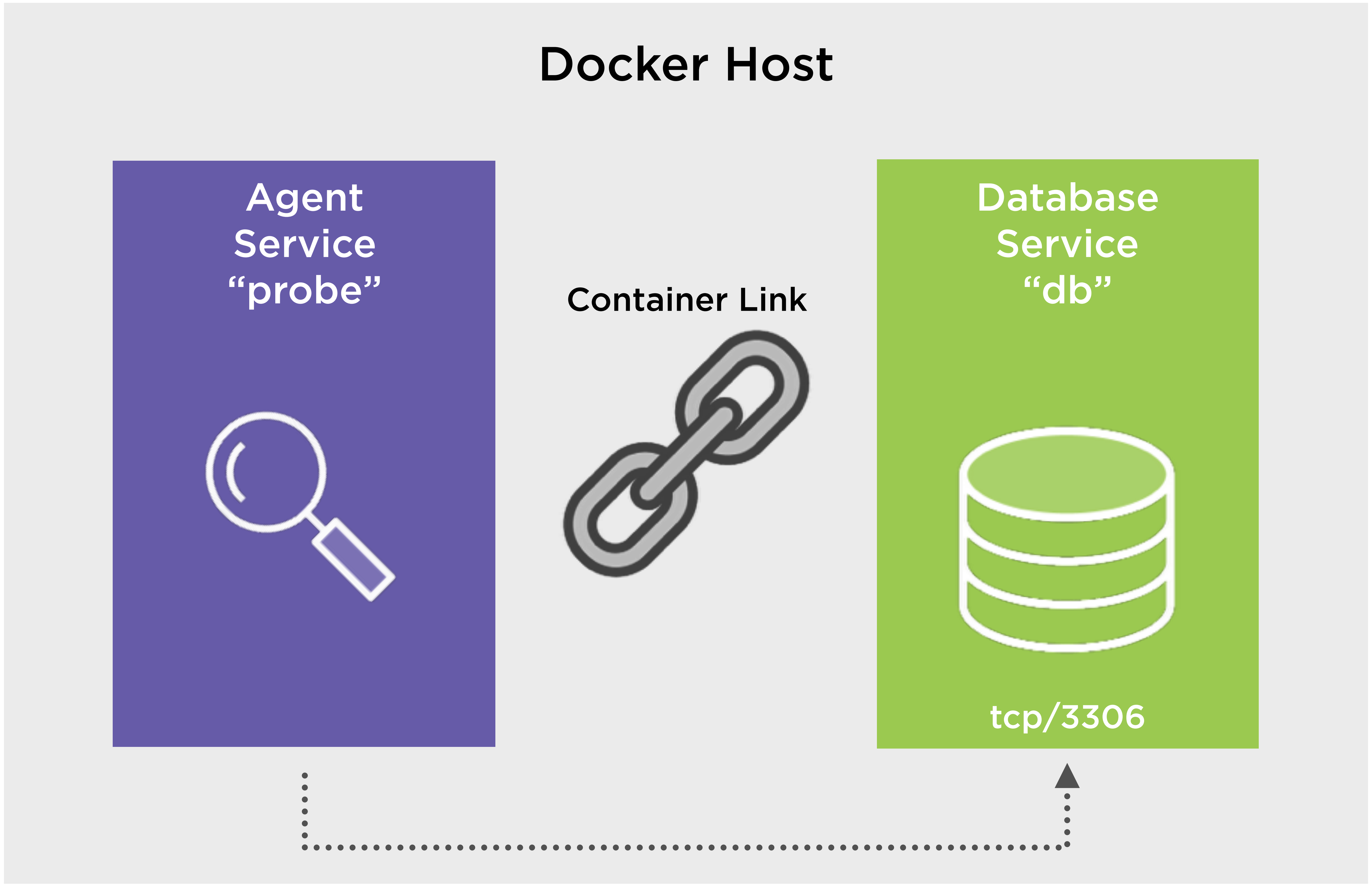
Multi-Container Race Condition



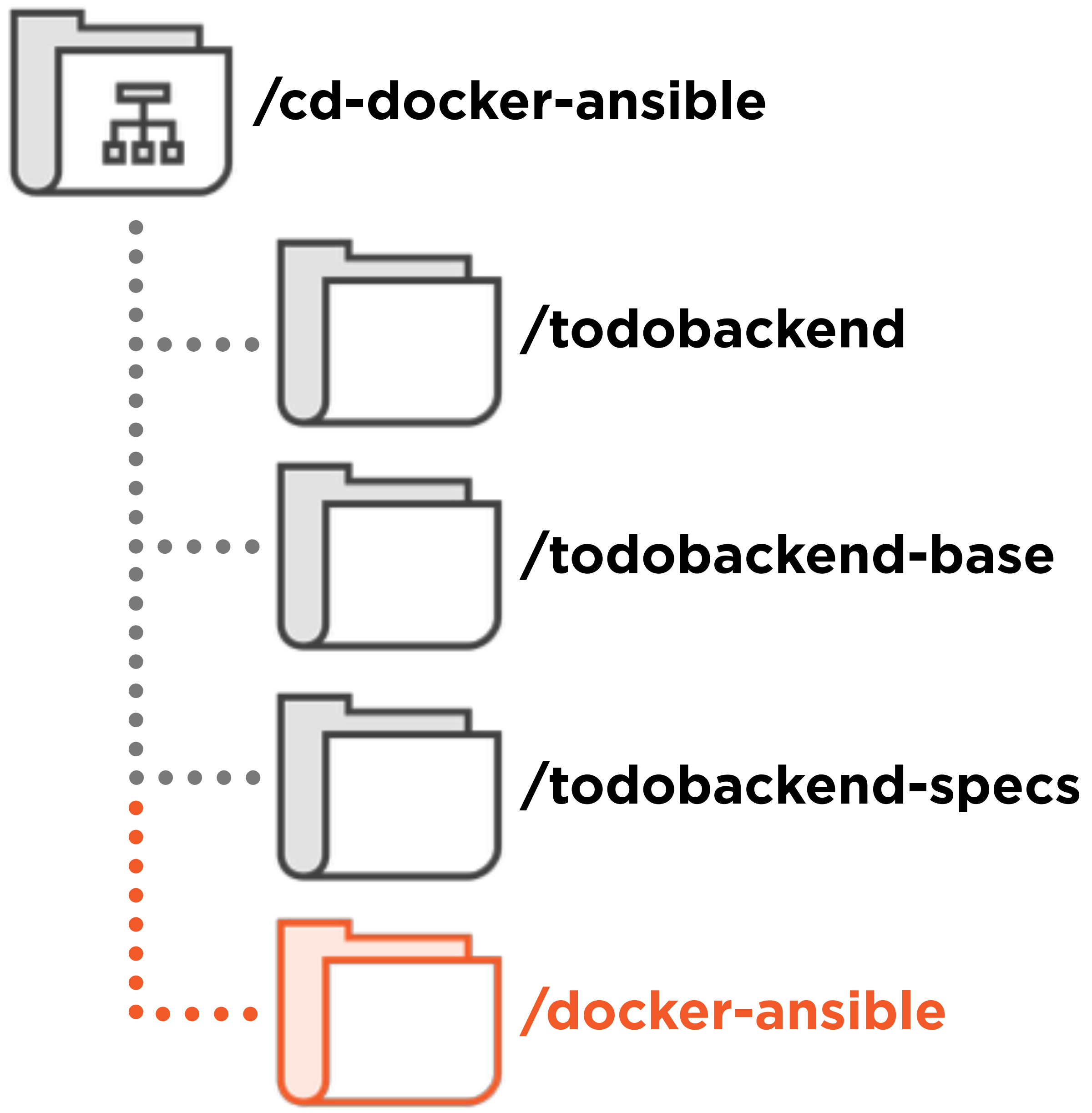
Waiting for a Dependent Service to Initialize

Agent Service

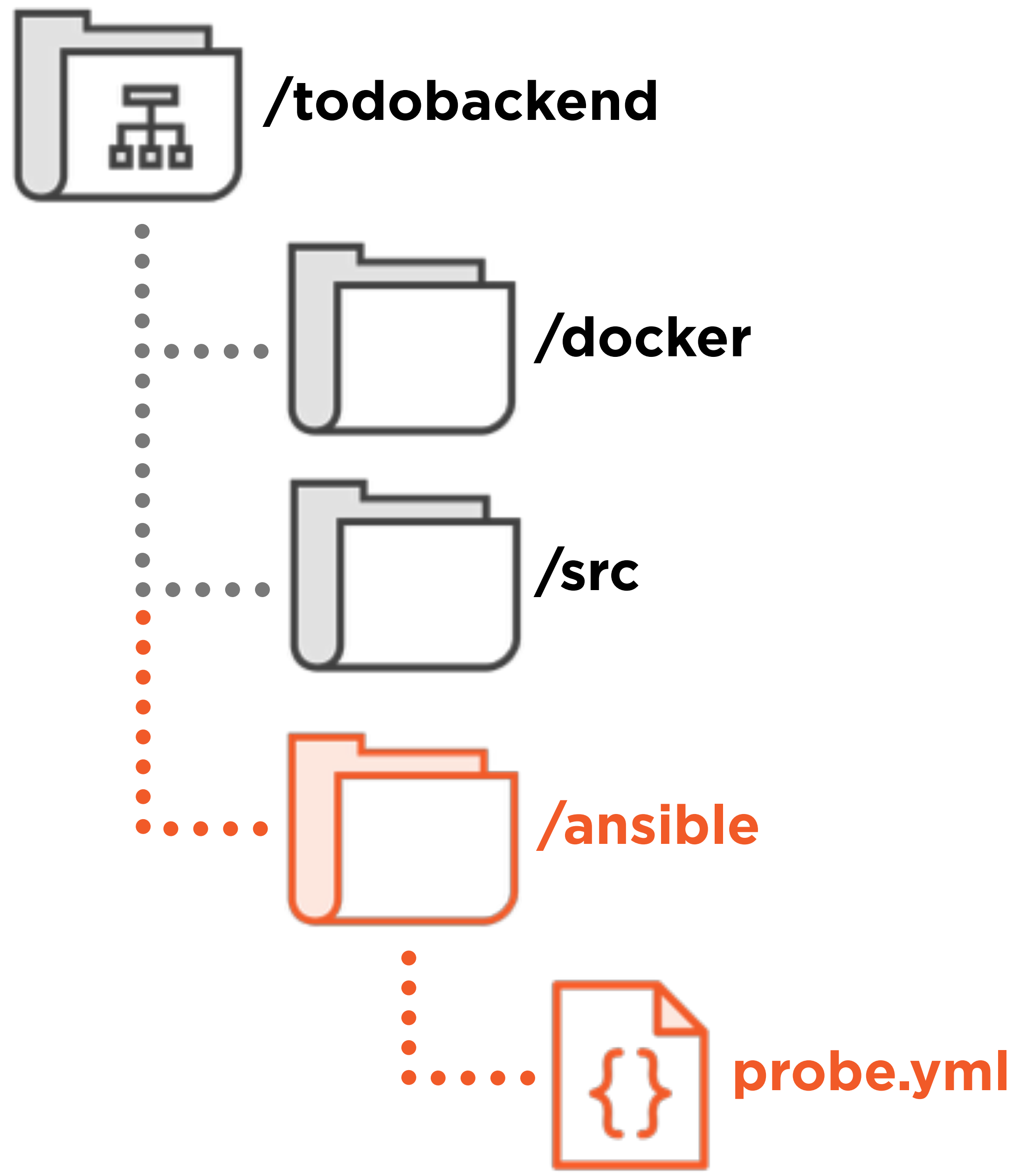



Docker Host
Agent
Service
“probe”
Container Link
Database
Service
“db”
tcp/3306

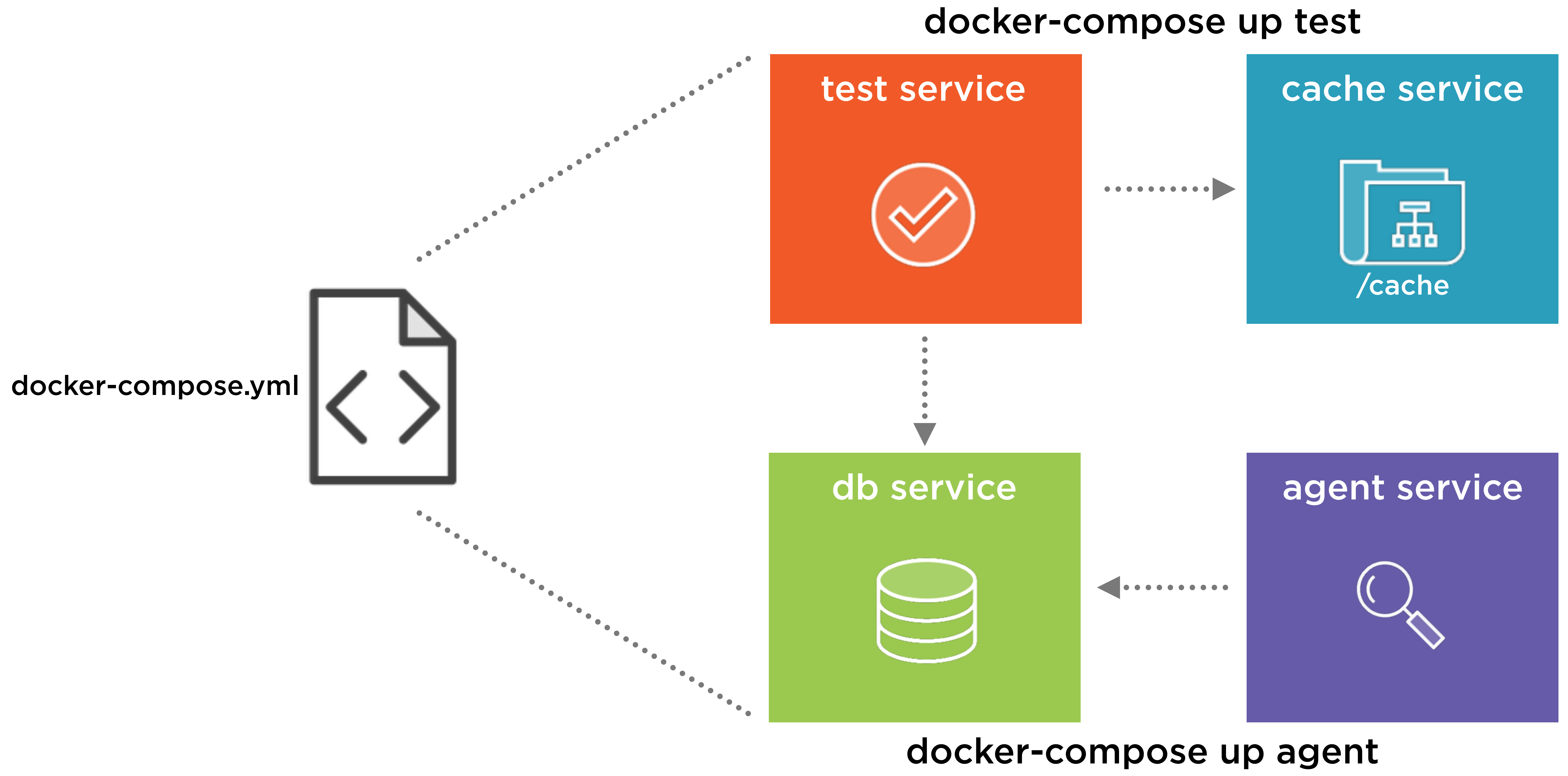
Course Folder Structure



Ansible Playbook



Test Environment



Summary

Unit and Integration Test Infrastructure

- Base Image
- Development Image
- Running tests using Docker

Docker Compose

- Multi-container environment
- Integration testing between multiple containers
- Agent services to help orchestrate testing