MODERN SOFTWARE DEVELOPMENT

A BRIEF OVERVIEW OF SOFTWARE DEVELOPMENT CONCEPTS

Created by Shawn Mittal
Sr. Lead Data Scientist
Booz Allen Hamilton

AGENDA

- Definitions
  - General
  - Containers
  - Container Orchestration
- Waterfall
- Agile
- DevOps
- Containers
- Kubernetes
- Let's Build an App

DEFINITIONS

GENERAL

- **Continuous Integration (CI)** – Process of integrating code into a shared repository on a continual basis. Often includes testing the code as well.
- **Continuous Deployment (CD)** – Process of deploying code upon integration of new code into a shared repository after automated testing.
- **Version Control** – System that records changes to a file or set of files over time and allows recall of specific versions later on.
- **Git** – An open source version control system designed to handle software development projects.

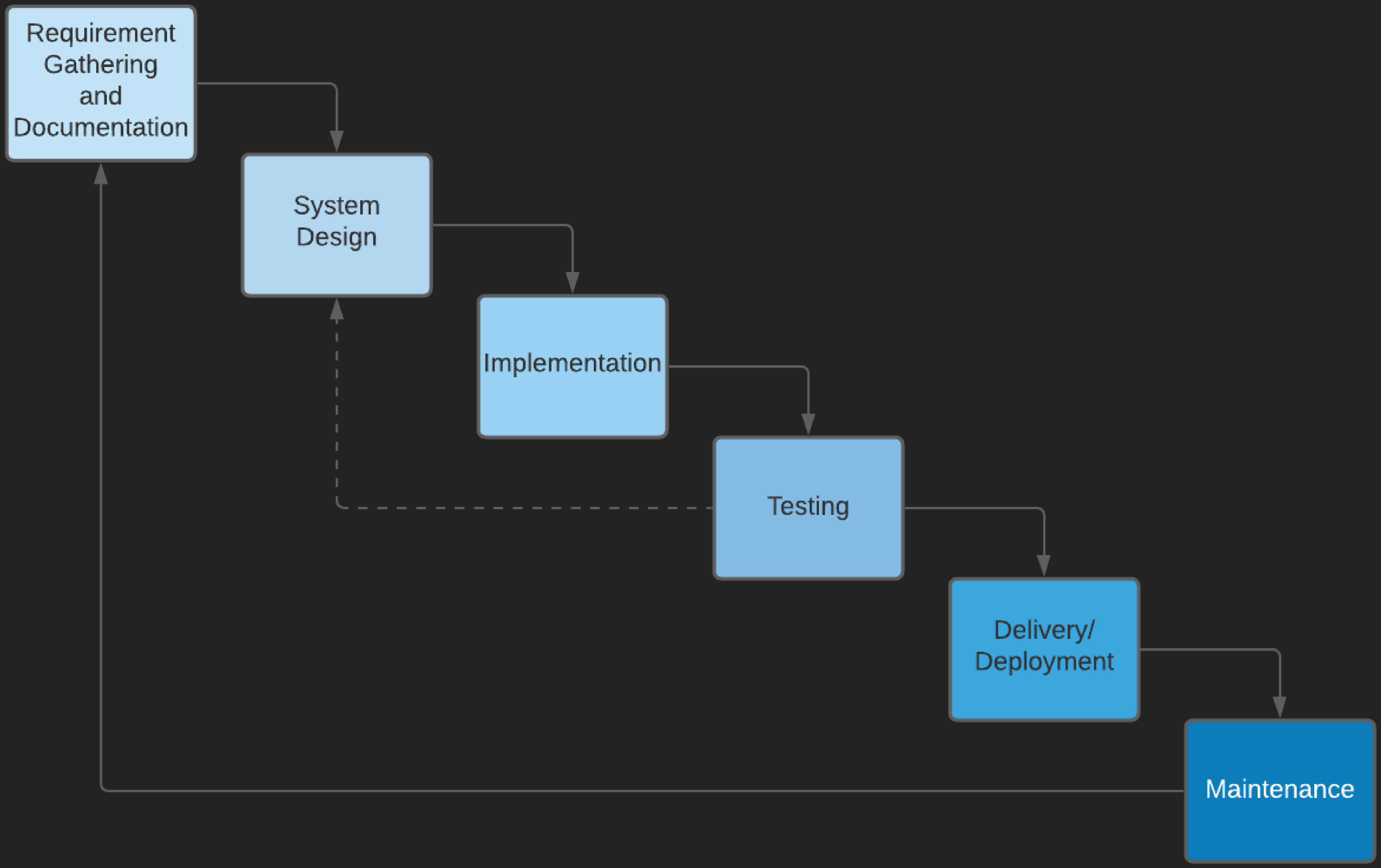
CONTAINERS

- **Container** – A standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.
- **Container Image** – Unchangeable (immutable) file that contains source code, libraries, dependencies, tools, and other files needed for an application to run.
- **Container Runtime** – System that is responsible for all the parts of running a container that isn't actually running the program itself.
- **Open Container Initiative (OCI)** – Open governance body that maintains two specifications: the container runtime spec, and the container image specification.

CONTAINER ORCHESTRATION

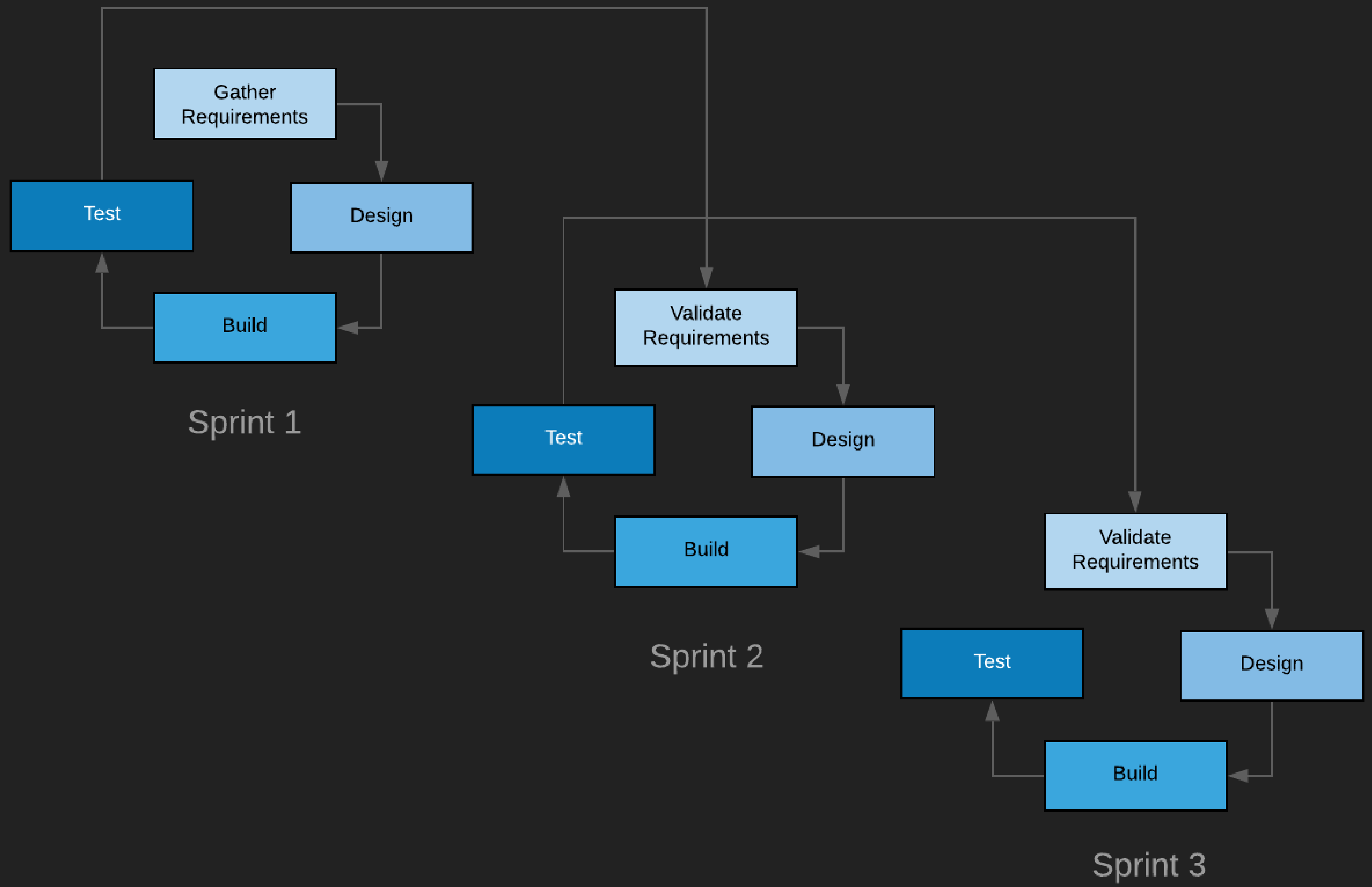
- **Container Orchestration** – The automation of much of the operational effort required to run containerized workloads and services (deploying, scaling, networking, load balancing, etc).
- **Kubernetes** – An open source container orchestration system.

WATERFALL


Requirement Gathering and Documentation
System Design
Implementation
Testing
Delivery/ Deployment
Maintenance

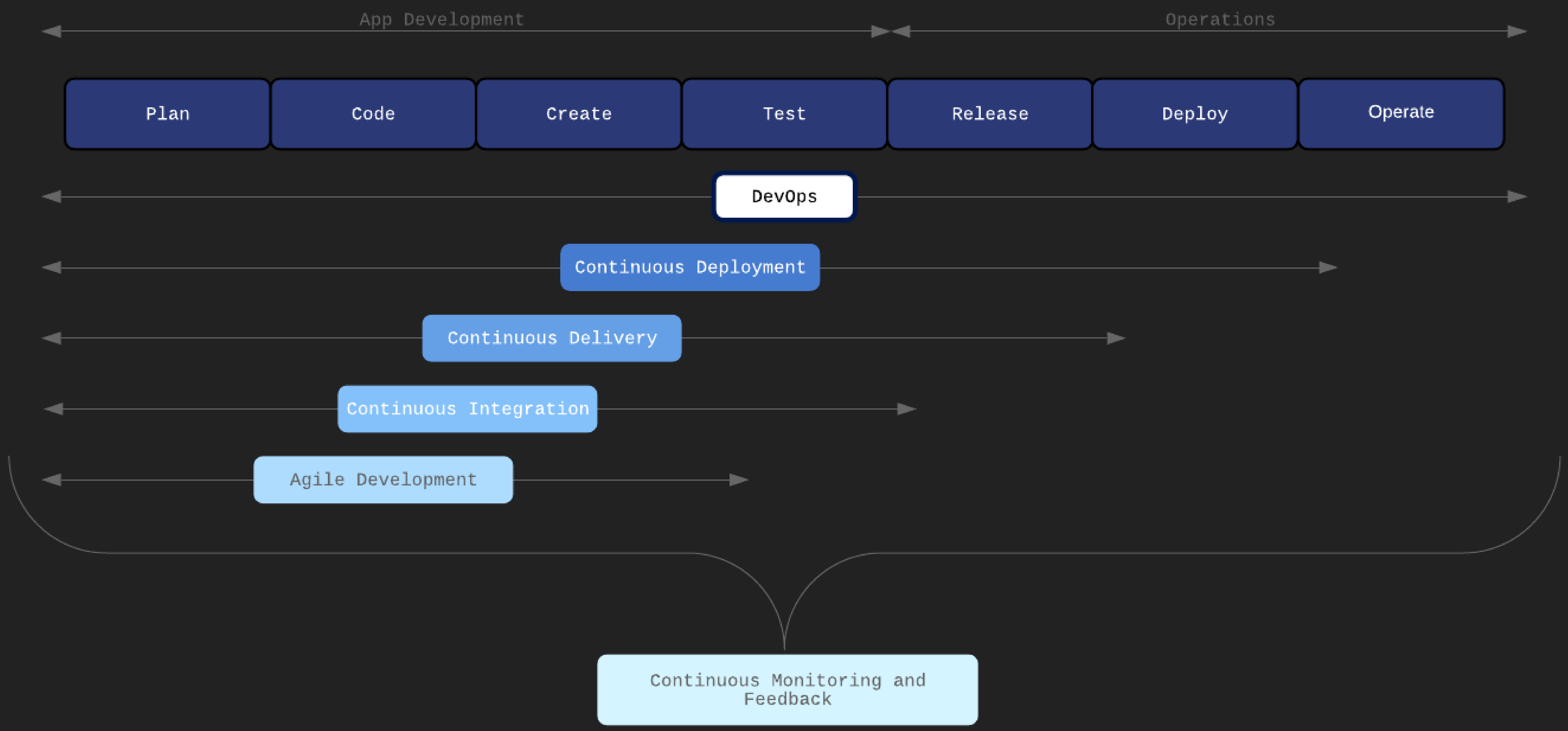
- Traditionally, software development was planned, managed, and executed using the waterfall methodology.
- Waterfall approach suited for when requirements and scope are fixed.
- Criticism:
  - Requirements often change during development. Waterfall is not agile enough to accommodate shift in requirements.
  - The design phase may not sufficiently address the requirements as intended by the client until implementation has already started.

AGILE


Gather Requirements
Test
Design
Build
Sprint 1
Validate Requirements
Test
Design
Build
Sprint 2
Validate Requirements
Test
Design
Build
Sprint 3

- Agile approach was developed to address shortcomings of waterfall.
- Advocates adaptive planning, evolutionary development, early delivery, and continual improvement, and it encourages flexible responses to change.
- Criticism:
  - Agile practices can be inefficient in large organizations and certain types of developments.
  - The increasing adoption of agile practices has also been criticized as being a management fad without considering results.

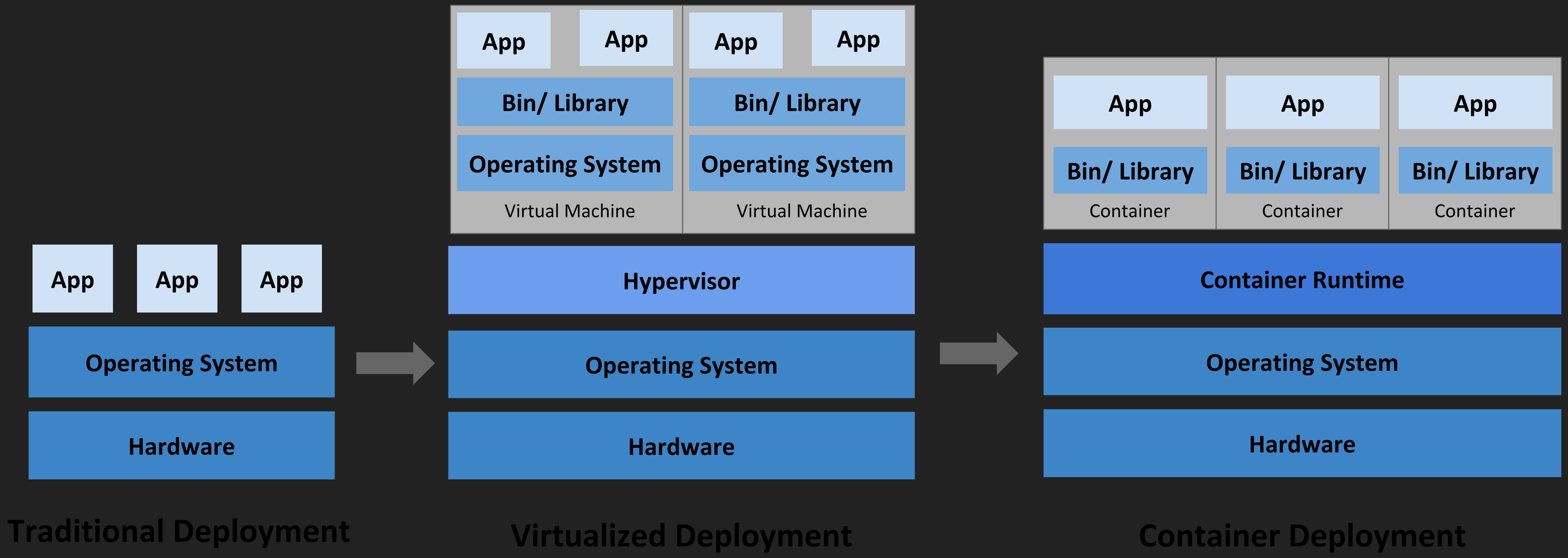
DEVOPS


App Development
Operations
Plan
Code
Create
Test
Release
Deploy
Operate
DevOps
Continuous Deployment
Continuous Delivery
Continuous Integration
Agile Development
Continuous Monitoring and Feedback

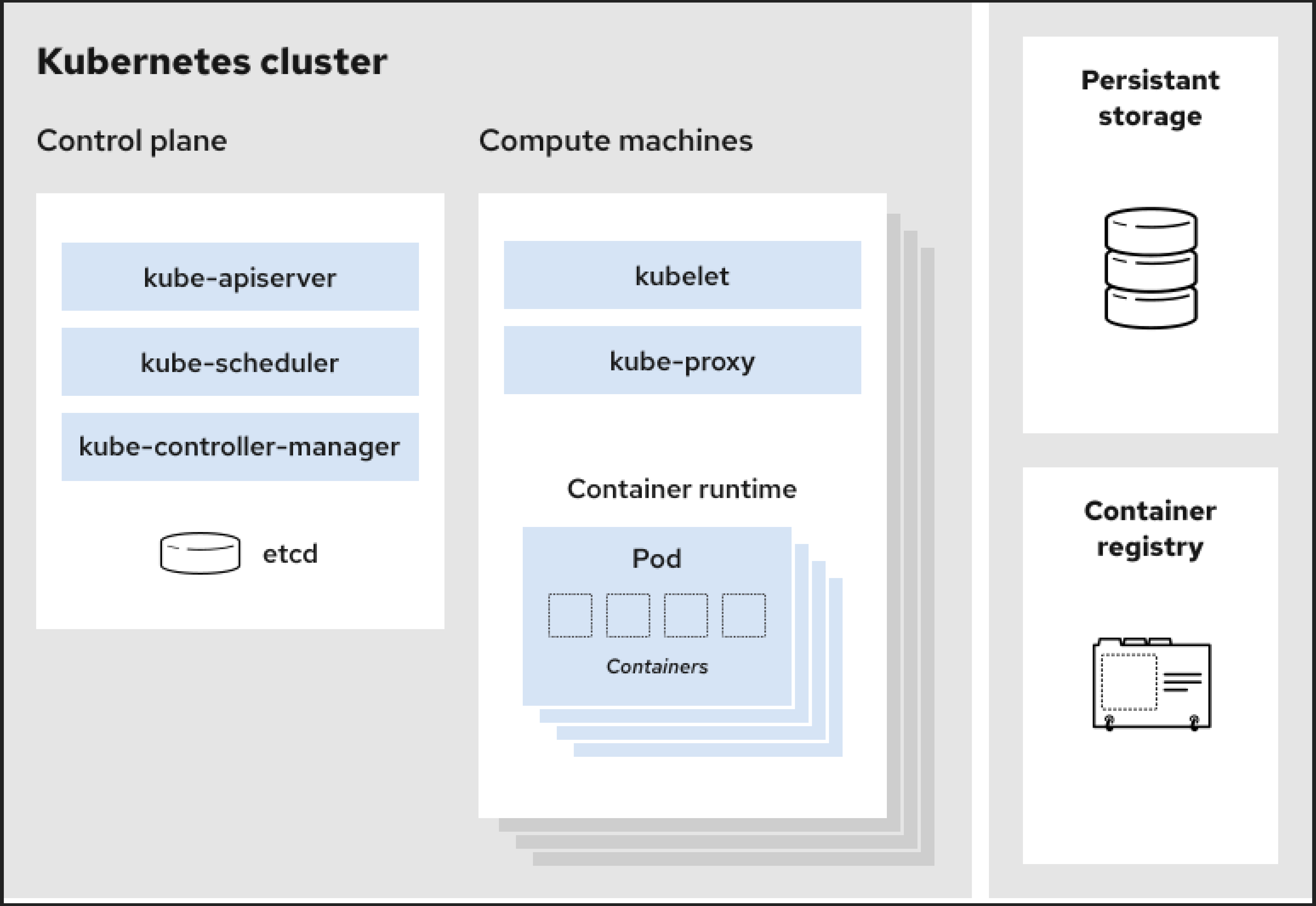
GOALS

- Improved deployment frequency
- Faster time to market
- Lower failure rate of new releases
- Shortened lead time between fixes
- Faster mean time to recovery

CONTAINERS


App
App
App
Operating System
Hardware
Traditional Deployment
App
App
App
App
Bin/ Library
Bin/ Library
Operating System
Operating System
Virtual Machine
Virtual Machine
Hypervisor
Operating System
Hardware
Virtualized Deployment
App
App
App
Bin/ Library
Bin/ Library
Bin/ Library
Container
Container
Container
Container Runtime
Operating System
Hardware
Container Deployment

KUBERNETES


Kubernetes cluster
Control plane
Compute machines
kube-apiserver
kube-scheduler
kube-controller-manager
etcd
kubelet
kube-proxy
Container runtime
Pod
Containers
Persistant storage
Container registry

LETS BUILD AN APP

PLAN

What do we want our application to do?

Let's write an app that returns a string after visiting a url. We will use Flask, a Python framework for building lightweight websites.

CODE/CREATE

Let's write the app. We've written a hello world function that returns a string of text.

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
        return "Here is a simple web application!"

if __name__ == '__main__':
        app.run()
```

Now we need to write a test for our application. Since it's not doing anything complex, we'll just check to see that the application is returning an http 200 response code.

```
import pytest
from app import app

@pytest.fixture
def flask_app():
        yield app.app

@pytest.fixture
def client(flask_app):
        return flask_app.test_client()

def test_index(flask_app, client):
        res = client.get('/')
        assert res.status_code == 200
```

While we don't have to, let's go ahead and containerize our web application. We'll write a Dockerfile to define what the container image should contain. We also need to make sure we expose the correct port so that the outside world can access our web application through the container.

```
FROM tiangolo/uwsgi-nginx:python3.8-alpine

# Set environment information
ENV LISTEN_PORT=5000
EXPOSE 5000
ENV UWSGI_INI uwsgi.ini

# Install python packages
WORKDIR /app
COPY requirements.txt .
RUN pip3 install --upgrade pip && \
        pip3 install --no-cache-dir -r ./requirements.txt

# Add web application to image
COPY . .
```

TEST/RELEASE/DEPLOY

The crux of CI/CD is automation. In order to automate the build, test, deploy process, GitHub provides a CI/CD capability called GitHub Actions that allows us to define steps in our pipeline via YAML.

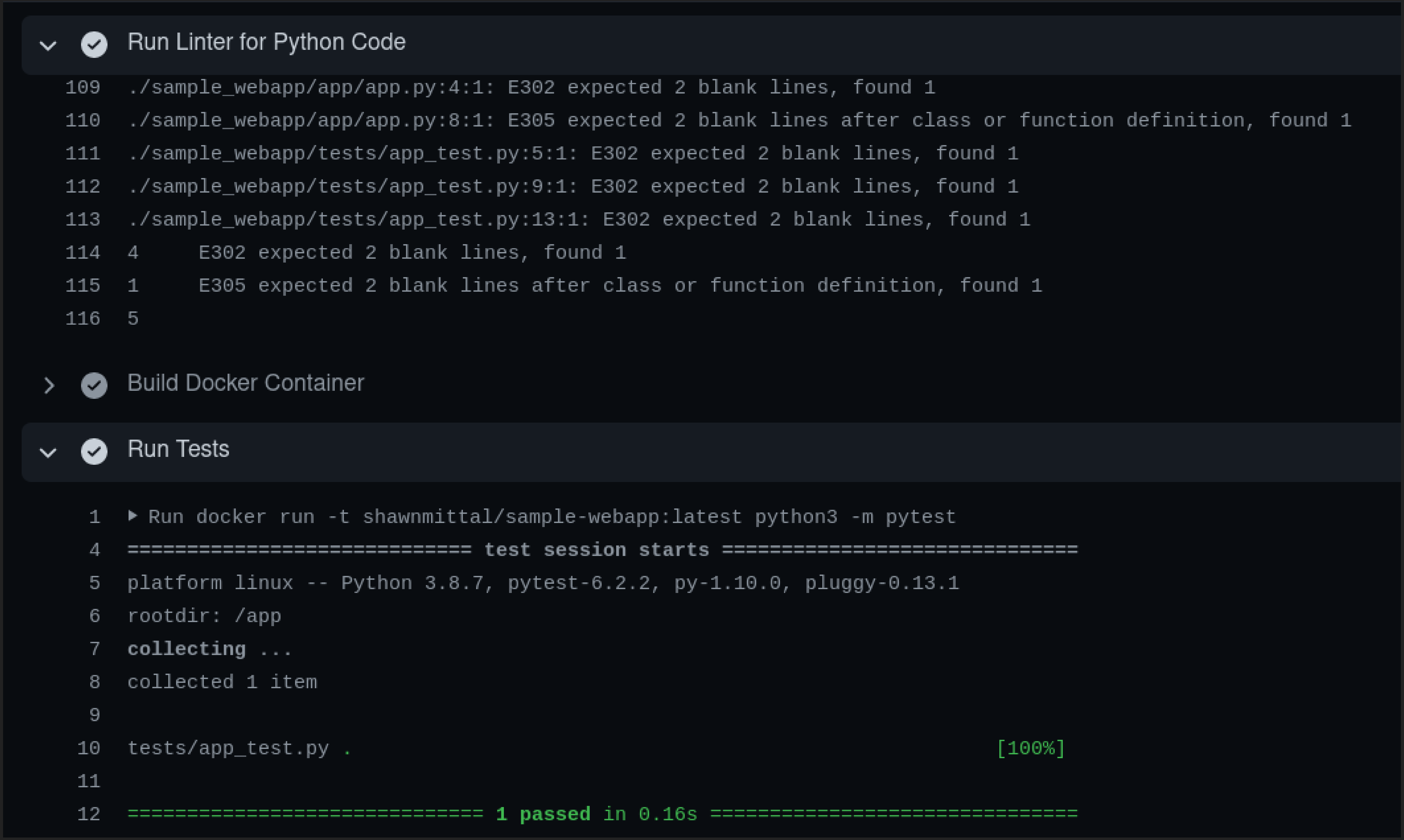
```yaml
name: CI

on:
push:
        branches: [ main ]

jobs:
build:
        runs-on: ubuntu-latest

        steps:
        - name: Checkout
          uses: actions/checkout@v2
        # not using standard actions/python@v2. causes dependency
        - name: Install Python and Dependencies
          run: |
```

Let's see the output of our automated static code analysis and tests.



```
./sample_webapp/app/app.py:4:1: E302 expected 2 blank lines, found 1
./sample_webapp/app/app.py:8:1: E305 expected 2 blank lines after class or function definition, found 1
./sample_webapp/tests/app_test.py:5:1: E302 expected 2 blank lines, found 1
./sample_webapp/tests/app_test.py:9:1: E302 expected 2 blank lines, found 1
./sample_webapp/tests/app_test.py:13:1: E302 expected 2 blank lines, found 1
4     E302 expected 2 blank lines, found 1
1     E305 expected 2 blank lines after class or function definition, found 1
5
```

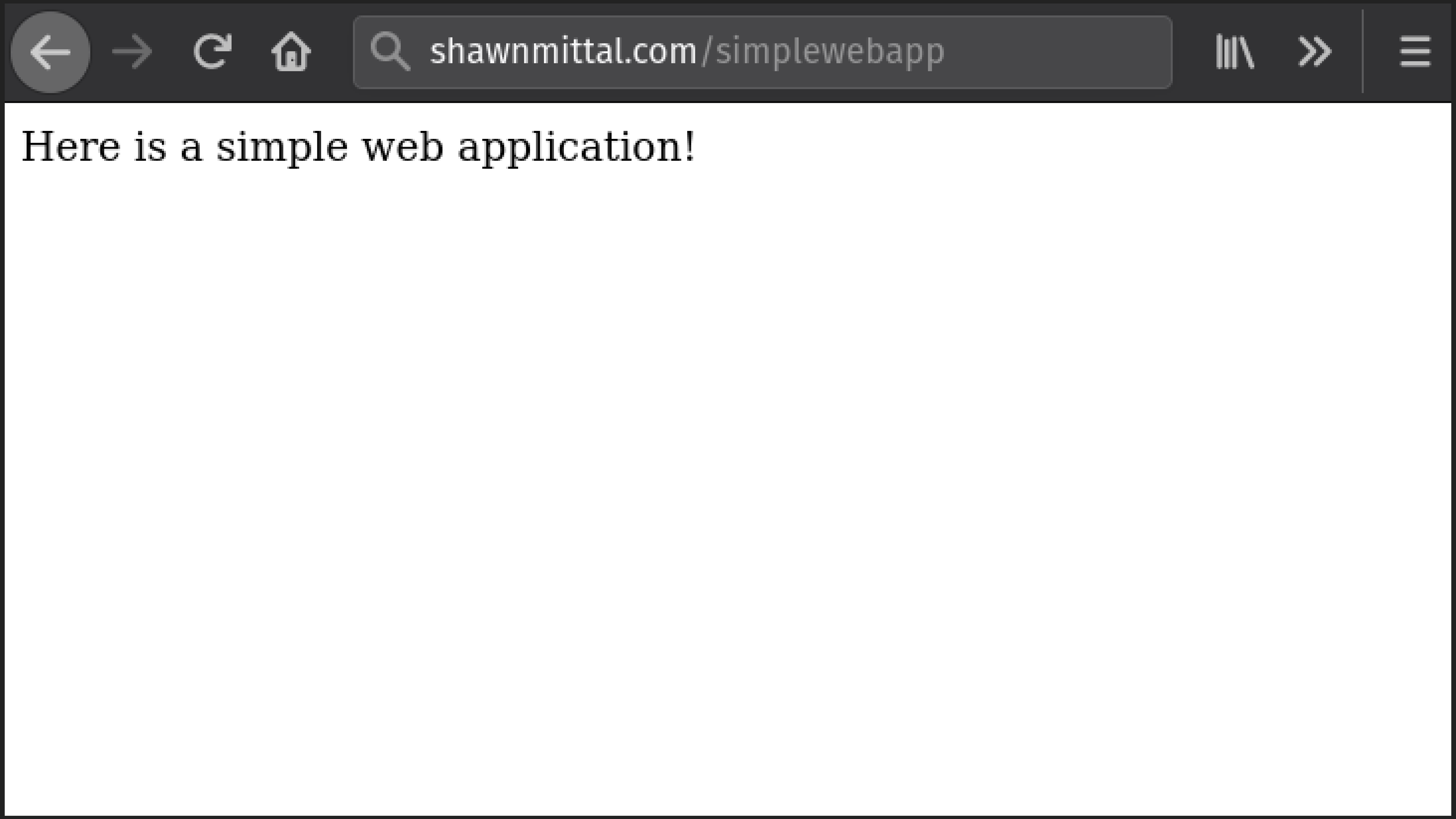
```
▶ Run docker run -t shawnmittal/sample-webapp:latest python3 -m pytest
============================= test session starts ==============================
platform linux -- Python 3.8.7, pytest-6.2.2, py-1.10.0, pluggy-0.13.1
rootdir: /app
collecting ...
collected 1 item

tests/app_test.py .                                                       [100%]

============================== 1 passed in 0.16s ===============================
```

OPERATE

We've deployed our webapp!


shawnmittal.com/simplewebapp

Here is a simple web application!

REFERENCES

- Source code for this presentation and the sample web application
- Red Hat introduction to Kubernetes Architecture
- What is Kubernetes
- Understanding DevOps process flow