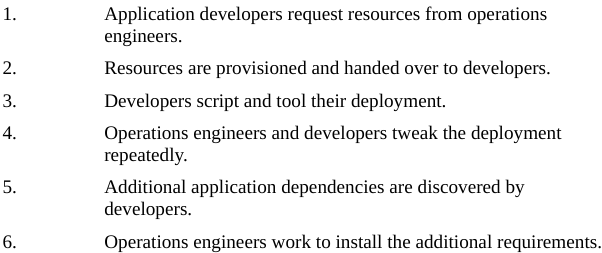
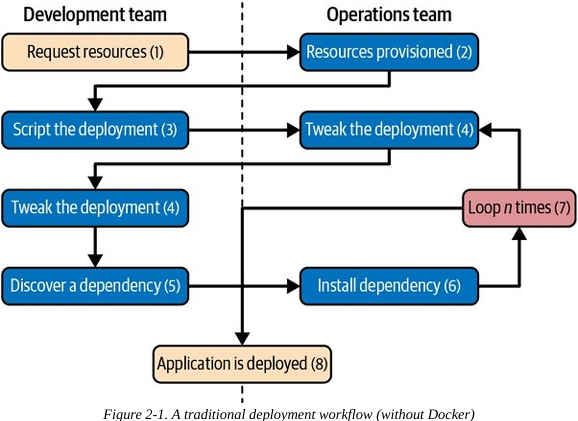
Chapter 2: The Docker Landscape

# 1. Process Simplification

-Traditionally, the cycle of getting an app to production often looks:







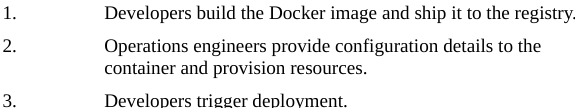
-Following traditional processes to deploy brand-new app into production can take a week for a complex new system. -> not productive, lots of effort and communication between teams of people, technically challenging and expensive, limit the kinds of innovation development teams will undertake in future.

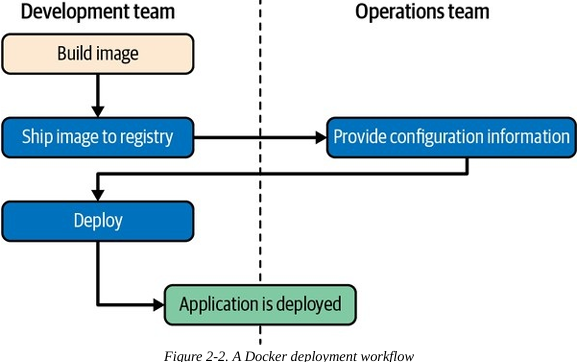
-Push-to-deploy systems like Heroku (<https://www.heroku.com/>) show what the world can look like if you are in control of your app and a majority of your dependencies. -> you might heard how much slower your internal system are compared with push-button solutions.

-Heroku is a whole env, not just a container engine. Docker provides a clean separation of responsibilities and encapsulation of dependencies, which results in a similar boost in productivity. It put devs in control of everything. Some tooling and orchestrators built on top of Docker (Kubernetes, Docker Swarm mode, Mesos) aim to replicate the simplicity of Heroku. However, a simple platform uses only Docker still provides all core process benefits.

-Docker adopt: “batteries included but removable”: its tools come with everything most people need to get the job done while still being built from interchangeable parts that can swapped and out to support custom solutions.

-Using an image repository, Docker allows the responsibility of building app image to be separated from deployment and operation of container. Dev teams can build their app with all of its dependencies, run it in development and test environment, and then just ship the exact same bundle of app and dependencies to production. Those bundles look the same from outside, operation engineers can then build or install standard tooling to deploy and run apps. That’s possible as Docker allows all of dependency issues to be discovered during development and test cycles. When app ready for 1st deploy, that work already been done. It doesn’t require many handoff between dev and op team, this can alleviate the need for anyone other than dev team to be involved in creation and deployment of a new service -> Simple, save lots of time, more robust software through testing of deployment environment before release.





# 2. Broad Support and Adoption

# 3. Architecture

## 3.1 Client/Server model

## 3.2 Network Ports and Unix Sockets

## 3.3 Robust Tooling

## 3.4 Docker Command-Line Tool

## 3.5 Docker Engine API

## 3.6 Container Networking

# 4. Getting the Most from Docker

## 4.1 Containers are not Virtual Machines

## 4.2 Limited Isolation

## 4.3 Containers are Lightweight

## 4.4 Toward an Immutable Infrastructure

## 4.5 Stateless Applications

## 4.6 Externalizing State

# 5. The Docker Workflow

## 5.1 Revision Control

### 5.1.1 Filesystem layers

### 5.1.2 Image tags

## 5.2 Building

## 5.3 Testing

## 5.4 Packaging

## 5.5 Deploying

## 5.6 The Docker Ecosystem

### 5.6.1 Orchestration

### 5.6.2 Immutable atomic hosts

### 5.6.3 Additional tools