Building 12 Factor JVM Applications

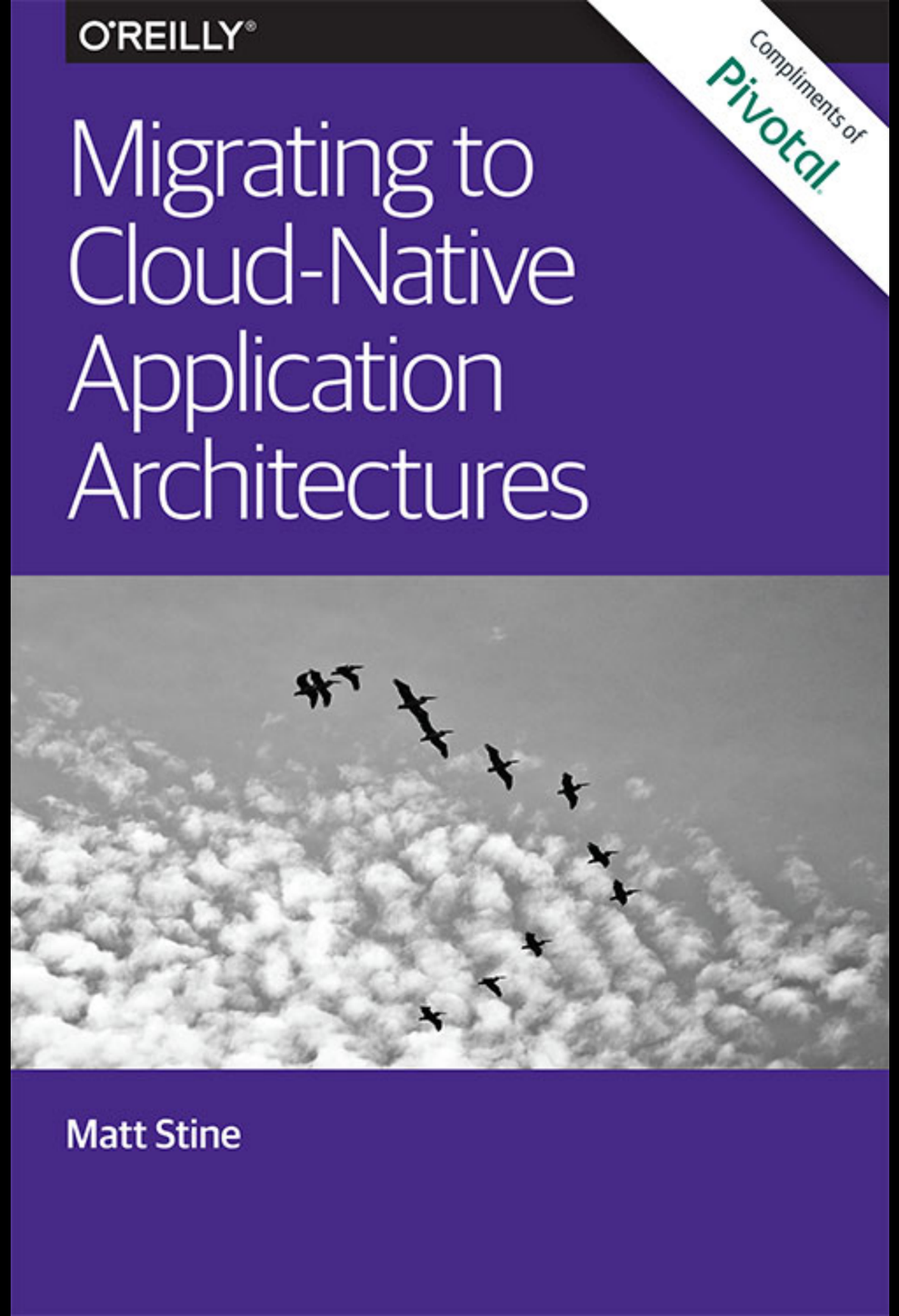
Me

Matt Stine @mstine
Senior Product Manager
Pivotal
matt.stine@gmail.com

I WROTE A LITTLE CLOUD BOOK...

FREE - Compliments of Pivotal

http://bit.ly/cloud-native-book



The Twelve-Factor App

http://12factor.net

Introduction

In the modern era, software is commonly delivered as a service: called *web apps*, or *software-as-a-service*. The twelve-factor app is a methodology for building software-as-a-service apps that:

- Use **declarative** formats for setup automation, to minimize time and cost for new developers joining the project;
- Have a **clean contract** with the underlying operating system, offering **maximum portability** between execution environments;
- Are suitable for **deployment** on modern **cloud platforms**, obviating the need for servers and systems administration;
- **Minimize divergence** between development and production, enabling **continuous deployment** for maximum agility;
- And can **scale up** without significant changes to tooling, architecture, or development practices.

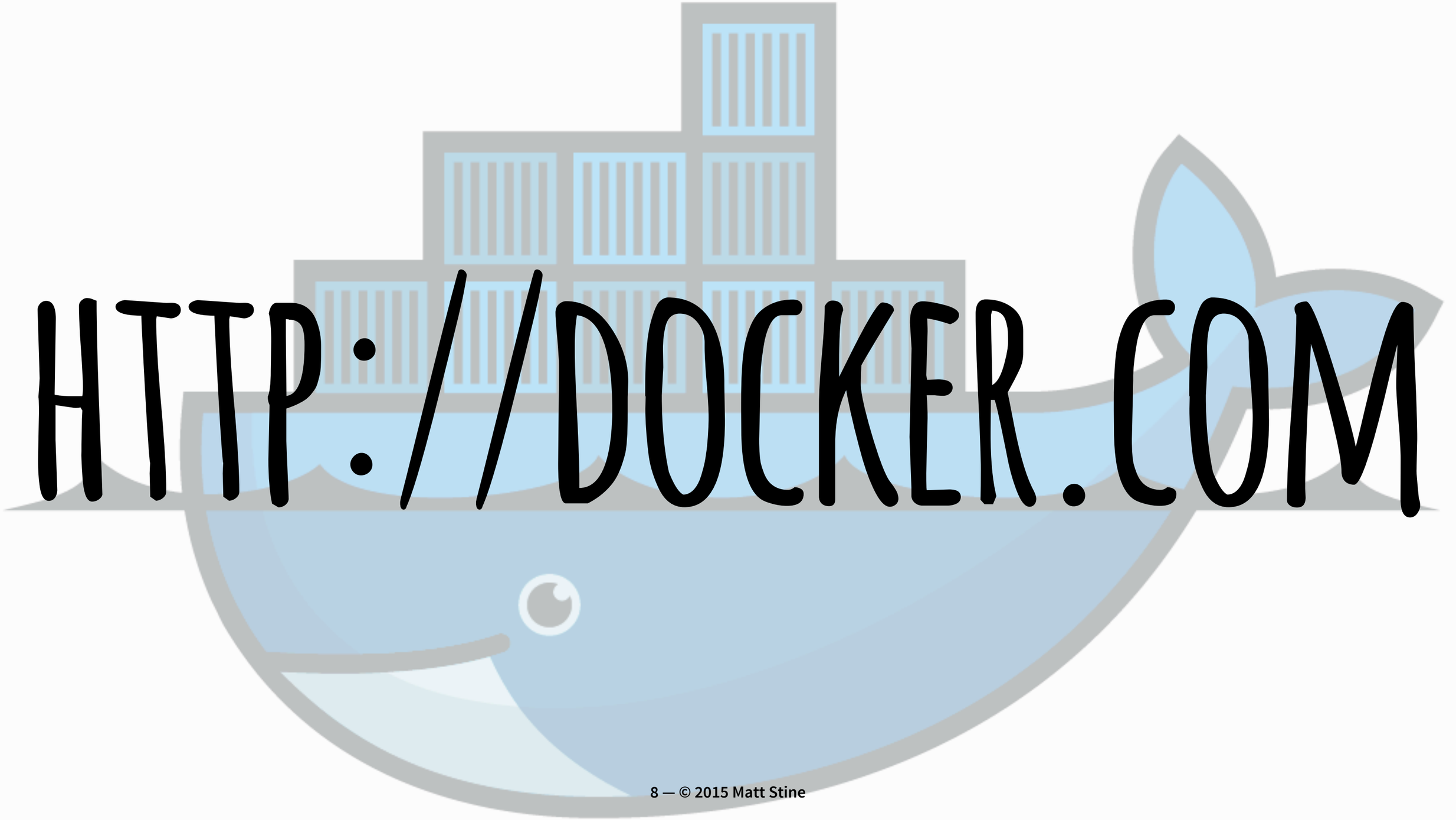
The twelve-factor methodology can be applied to apps written in any programming language, and which use any

Patterns

- Cloud-native application architectures
- Optimized for speed, safety, & scale
- Declarative configuration
- Stateless/shared-nothing processes
- Loose coupling to application environment


HTTP://HEROKU.COM


HTTP://CLOUDFOUNDRY.ORG
CLOUD
FOUNDRY™


HTTP://DOCKER.COM


MICROSERVICES!
http://www.etsy.com/shop/sharpwriter
NEUSER 2012

Microframeworks

Dropwizard

Spring Boot

Dropwizard

- http://dropwizard.io
- Library | Framework
- GOAL: Meet "production-ready" web app needs.
- Jetty / Jersey / Jackson
- Metrics: http://metrics.dropwizard.io/

HelloWorldConfiguration

```
public class HelloWorldConfiguration extends Configuration {
  @NotEmpty
  private String template;

  @NotEmpty
  private String defaultName = "Stranger";

  @JsonProperty
  public String getTemplate() { return template; }

  @JsonProperty
  public void setTemplate(String template) { this.template = template; }

  @JsonProperty
  public String getDefaultName() { return defaultName; }

  @JsonProperty
  public void setDefaultName(String name) { this.defaultName = name; }
}
```

HelloWorldResource

```
@Path("/hello-world")
@Produces(MediaType.APPLICATION_JSON)
public class HelloWorldResource {
    private final String template;
    private final String defaultName;
    private final AtomicLong counter;

    public HelloWorldResource(String template, String defaultName) {
        this.template = template;
        this.defaultName = defaultName;
        this.counter = new AtomicLong();
    }

    @GET
    @Timed
    public Saying sayHello(@QueryParam("name") Optional<String> name) {
        final String value = String.format(template, name.or(defaultName));
        return new Saying(counter.incrementAndGet(), value);
    }
}
```

Saying

```
public class Saying {
    private long id;

    @Length(max = 3)
    private String content;

    public Saying() {}

    public Saying(long id, String content) {
        this.id = id;
        this.content = content;
    }

    @JsonProperty
    public long getId() { return id; }

    @JsonProperty
    public String getContent() { return content; }
}
```

TemplateHealthCheck

```
public class TemplateHealthCheck extends HealthCheck {
    private final String template;

    public TemplateHealthCheck(String template) { this.template = template; }

    @Override
    protected Result check() throws Exception {
        final String saying = String.format(template, "TEST");
        if (!saying.contains("TEST")) {
            return Result.unhealthy("template doesn't include a name");
        }
        return HealthCheck.Result.healthy();
    }
}
```

HelloWorldApplication

```
public class HelloWorldApplication extends Application<HelloWorldConfiguration> {
    public static void main(String[] args) throws Exception {
        new HelloWorldApplication().run(args);
    }

    @Override
    public String getName() { return "hello-world"; }

    @Override
    public void initialize(Bootstrap<HelloWorldConfiguration> bootstrap) {
        bootstrap.setConfigurationFactoryFactory(new EnvironmentConfigurationFactoryFactory());
    }

    @Override
    public void run(HelloWorldConfiguration configuration,
                    Environment environment) {
        final HelloWorldResource resource = new HelloWorldResource(
                configuration.getTemplate(),
                configuration.getDefaultName()
        );
        final TemplateHealthCheck healthCheck =
                new TemplateHealthCheck(configuration.getTemplate());
        environment.healthChecks().register("template", healthCheck);
        environment.jersey().register(resource);
    }
}
```

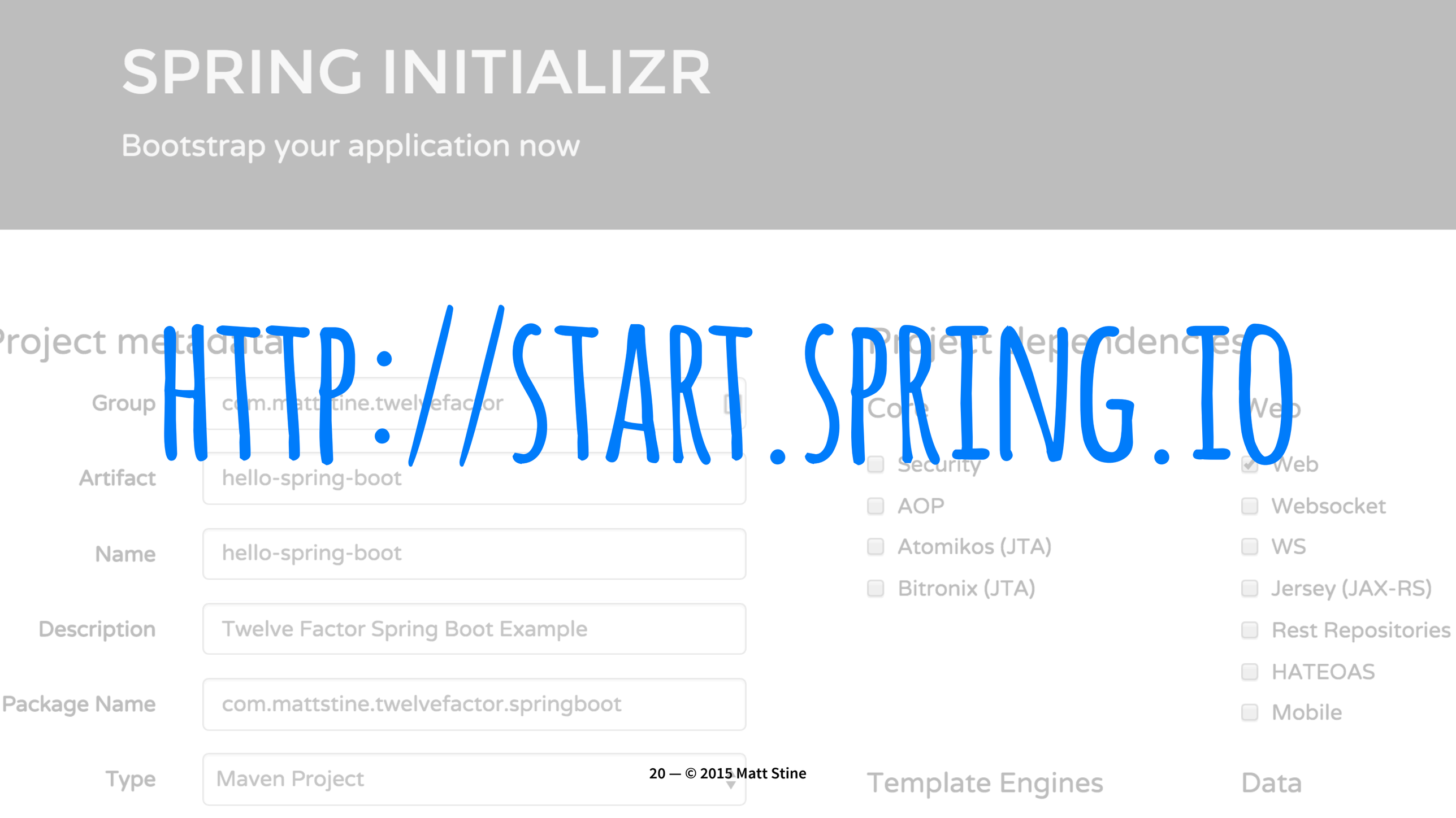
hello-world.yml

```
template: $env:TEMPLATE:Hello, %s!
defaultName: $env:DEFAULT_NAME:Stranger
```

DEMO

Spring Boot

- http://projects.spring.io/spring-boot
- Opinionated convention over configuration
- Production-ready Spring applications
- Embed Tomcat, Jetty or Undertow
- STARTERS
- Actuator: Metrics, health checks, introspection


SPRING INITIALIZR
Bootstrap your application now
HTTP://START.SPRING.IO
Project metadata
Group
com.mattstine.twelvefactor
Artifact
hello-spring-boot
Name
hello-spring-boot
Description
Twelve Factor Spring Boot Example
Package Name
com.mattstine.twelvefactor.springboot
Type
Maven Project
Project dependencies
Core
Security
AOP
Atomikos (JTA)
Bitronix (JTA)
Web
Web
Websocket
WS
Jersey (JAX-RS)
Rest Repositories
HATEOAS
Mobile
Template Engines
Data

HelloWorldController

```
@RestController("/hello-world")
public class HelloWorldController {

    private final String template;
    private final String defaultName;
    private final AtomicLong counter;

    @Autowired
    public HelloWorldController(@Value("${template}") String template, @Value("${default.name}") String defaultName) {
        this.template = template;
        this.defaultName = defaultName;
        this.counter = new AtomicLong();
    }

    @RequestMapping(method = RequestMethod.GET, produces = "application/json")
    public Saying sayHello(@RequestParam(value = "name", required = false) String name) {
        final String value = String.format(template, (name != null) ? name : defaultName);
        return new Saying(counter.incrementAndGet(), value);
    }
}
```

Saying

```
public class Saying {
    private long id;

    @Length(max = 3)
    private String content;

    public Saying() {}

    public Saying(long id, String content) {
        this.id = id;
        this.content = content;
    }

    @JsonProperty
    public long getId() { return id; }

    @JsonProperty
    public String getContent() { return content; }
}
```

TemplateHealthCheck

```
@Component
public class TemplateHealthCheck implements HealthIndicator {

    private final String template;

    @Autowired
    public TemplateHealthCheck(@Value("${template}") String template) {
        this.template = template;
    }

    @Override
    public Health health() {
        final String saying = String.format(template, "TEST");
        if (!saying.contains("TEST")) {
            return Health.down().withDetail("error", "template doesn't include a name").build();
        }
        return Health.up().build();
    }
}
```

HelloSpringBootApplication

```
@SpringBootApplication
public class HelloSpringBootApplication {

    public static void main(String[] args) {
        SpringApplication.run(HelloSpringBootApplication.class, args);
    }
}
```

application.yml

```
template: Hello, %s!
default:
  name: Stranger
```

DEMO

I. Codebase

— One codebase tracked in revision control, many deploys

What you already do (right?).

Multiple Codebases

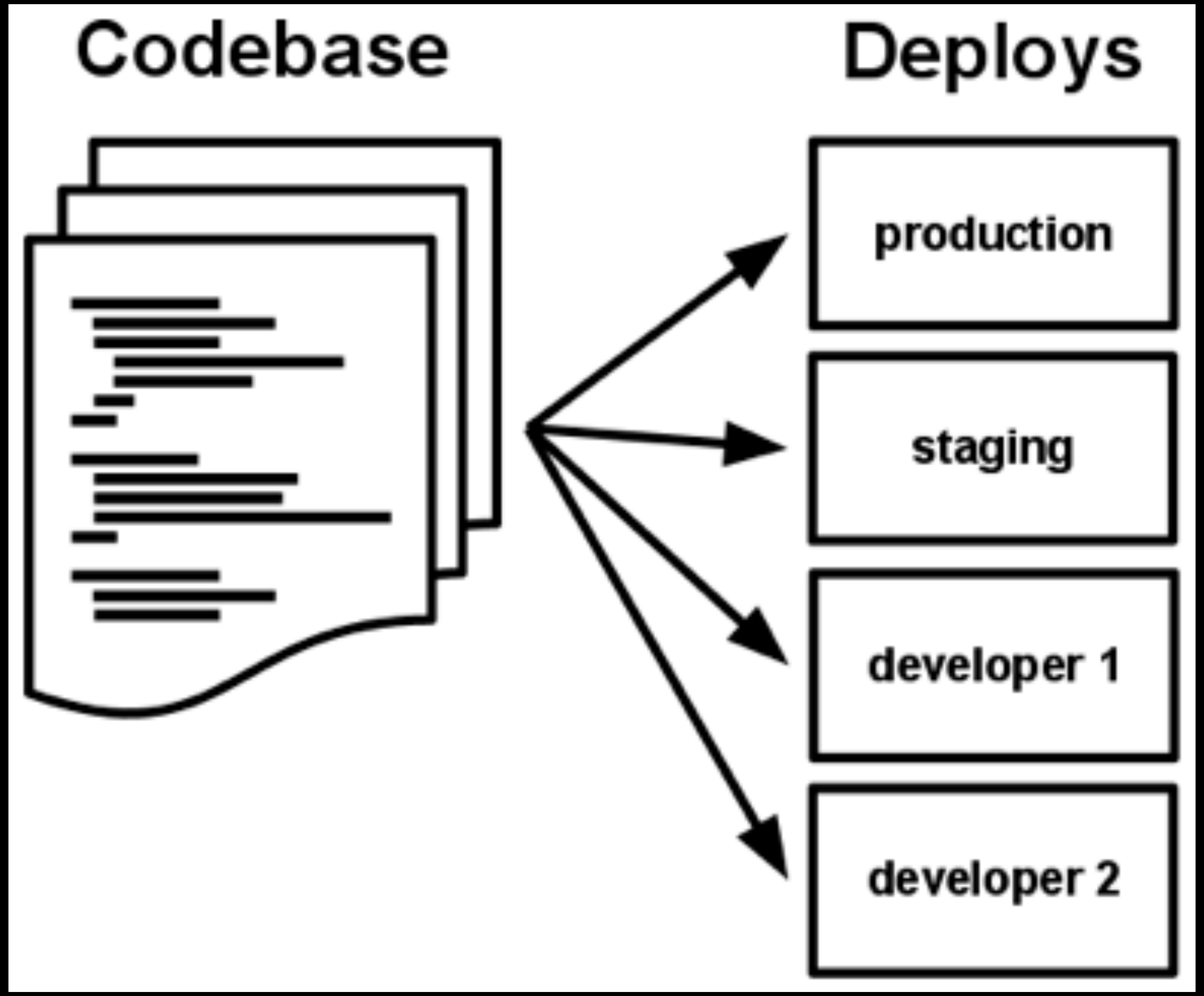
=

Distributed System

Shared Code

=

Shared Library


Codebase
Deploys
production
staging
developer 1
developer 2

II. Dependencies

— Explicitly declare and isolate dependencies

What you already do (right?).

Declarative Dependency Management

- Gradle
- Maven
- Ant/Ivy
- Etc.

Never Assume Anything is There

Carry Everything You Need

- Spring Boot Jar Apps
- Dropwizard Maven Shade Jar Apps

III. Config

— Store config in the environment

What is configuration?

- Resource handles to databases and other backing services
- Credentials to external sources (e.g. S3, Twitter, ...)
- Per-deploy values (e.g. canonical hostname for deploy)
- ANYTHING that's likely to vary between deploys

Where NOT to store it:

- In the CODE (Captian Obvious)
- In PROPERTIES FILES (That's code...)
- In the BUILD (ONE build, MANY deploys)
- In the APP SERVER (e.g. JNDI datasources)

Store it in the environment!

When am I done?

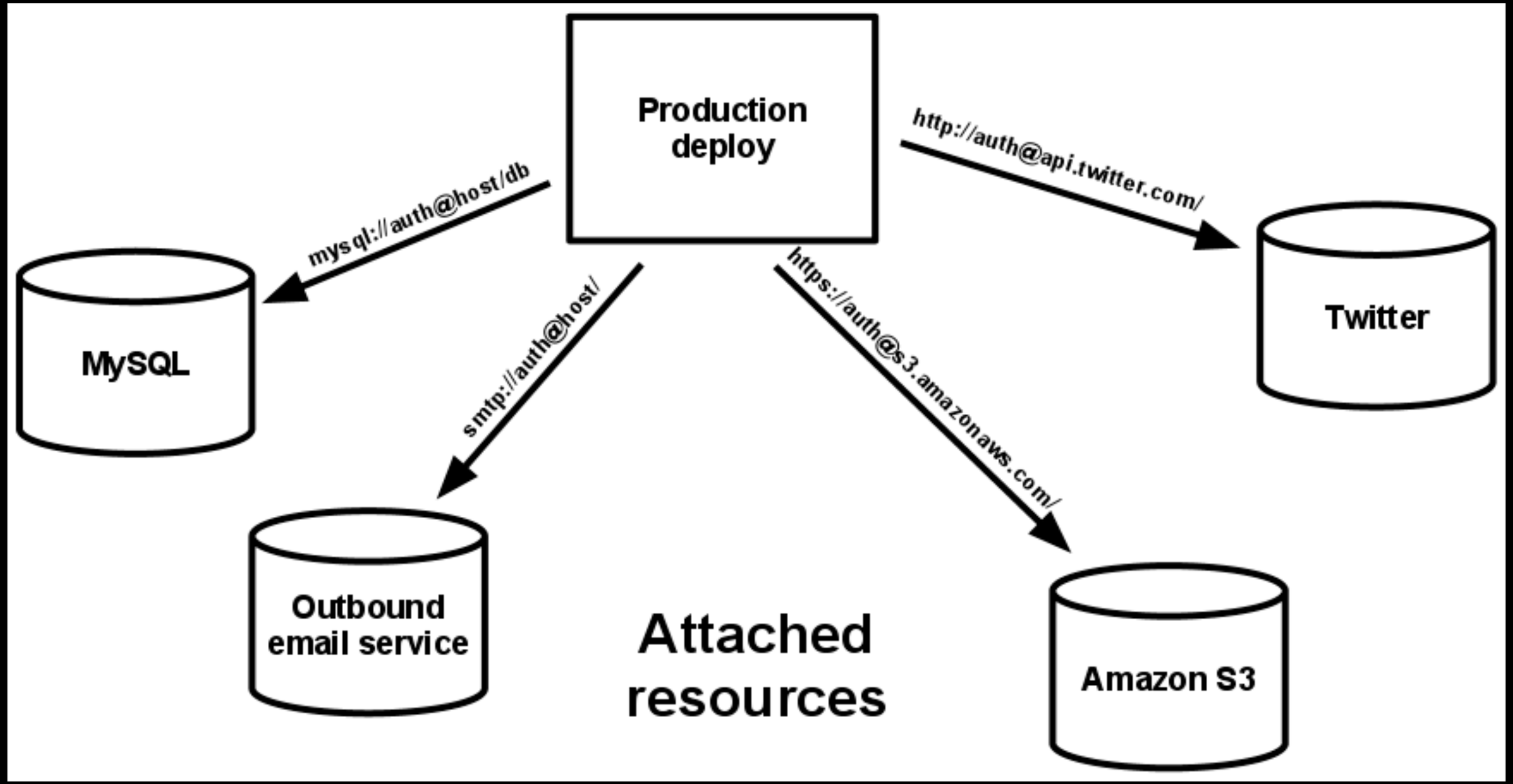
When "...the codebase could be made open source at any moment, without compromising any credentials."

Why Environment Variables?

- Easy to change
- Little chance of being “checked in” to VCS
- Language/OS-agnostic standard

IV. Backing Services

— Treat backing services as attached resources


Production deploy
mysql://auth@host/db
http://auth@api.twitter.com/
MySQL
Twitter
smtp://auth@host/
https://auth@s3.amazonaws.com/
Outbound email service
Attached resources
Amazon S3

No Distinction Between Local and 3rd Parties

Always Swappable!

Enabled by Externalized Config

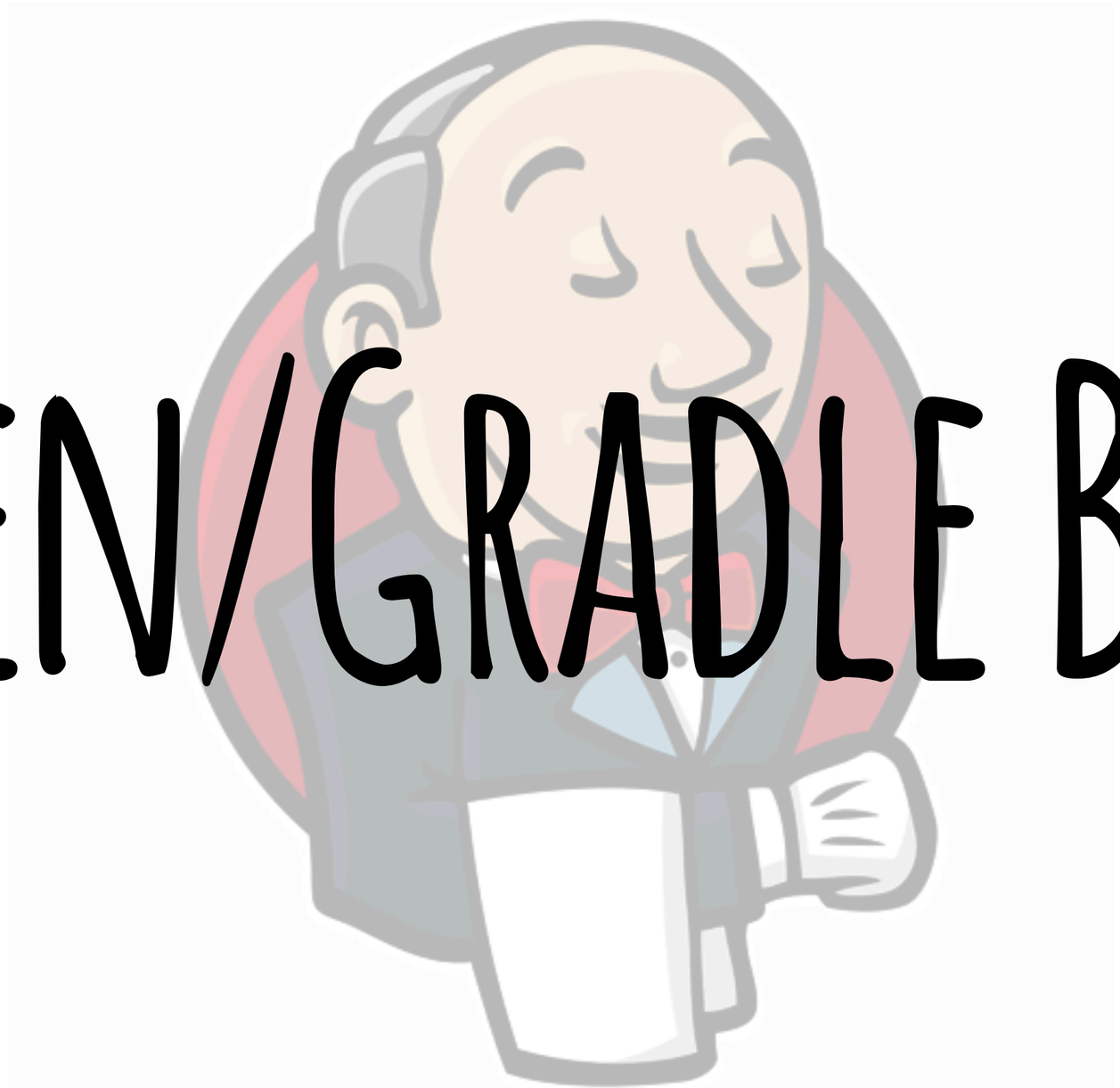
V. Build, release, run

— Strictly separate build and run stages

Build

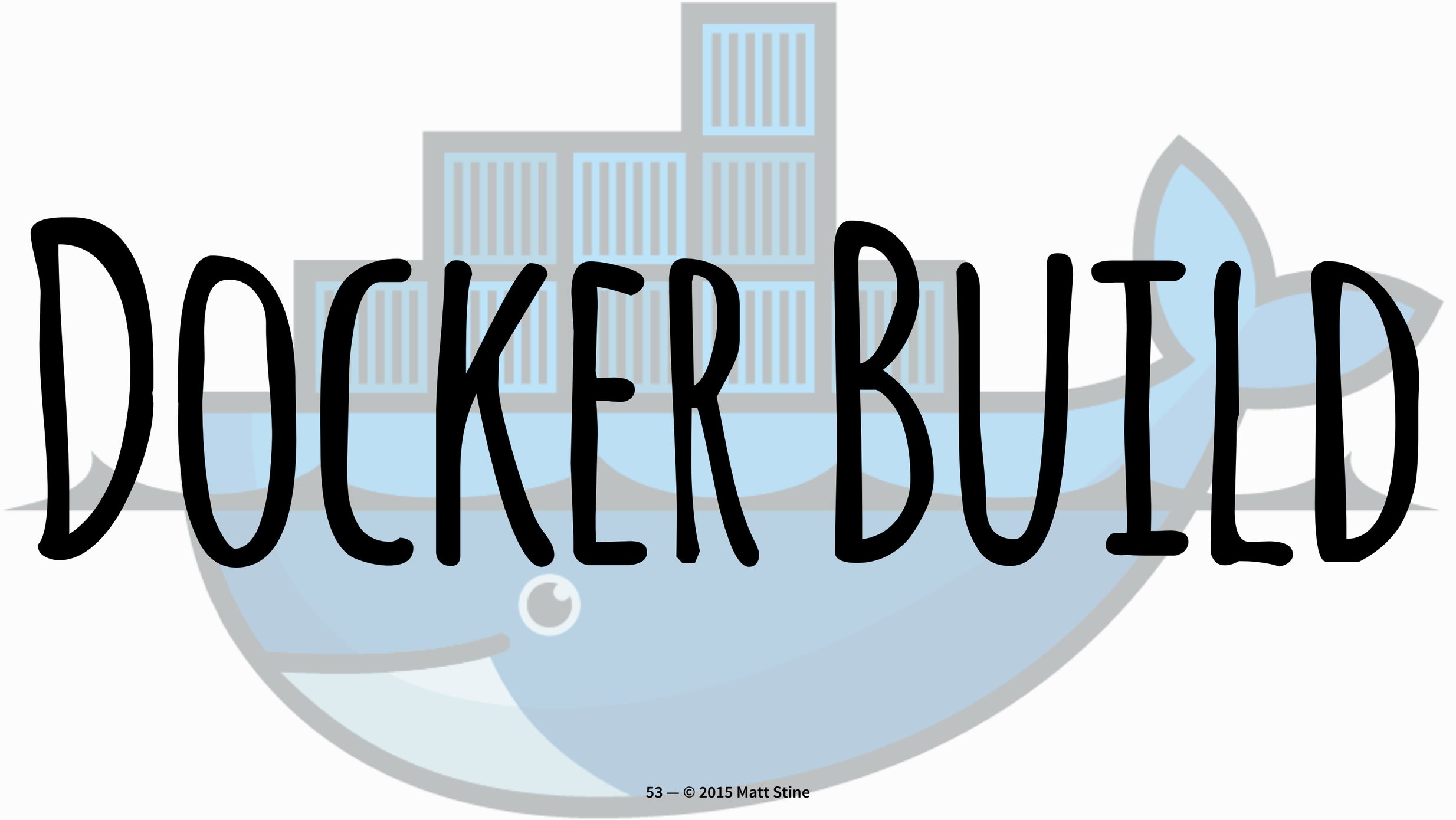
Produce a deployable artifact with all dependencies bundled in.

Maven/Gradle Build



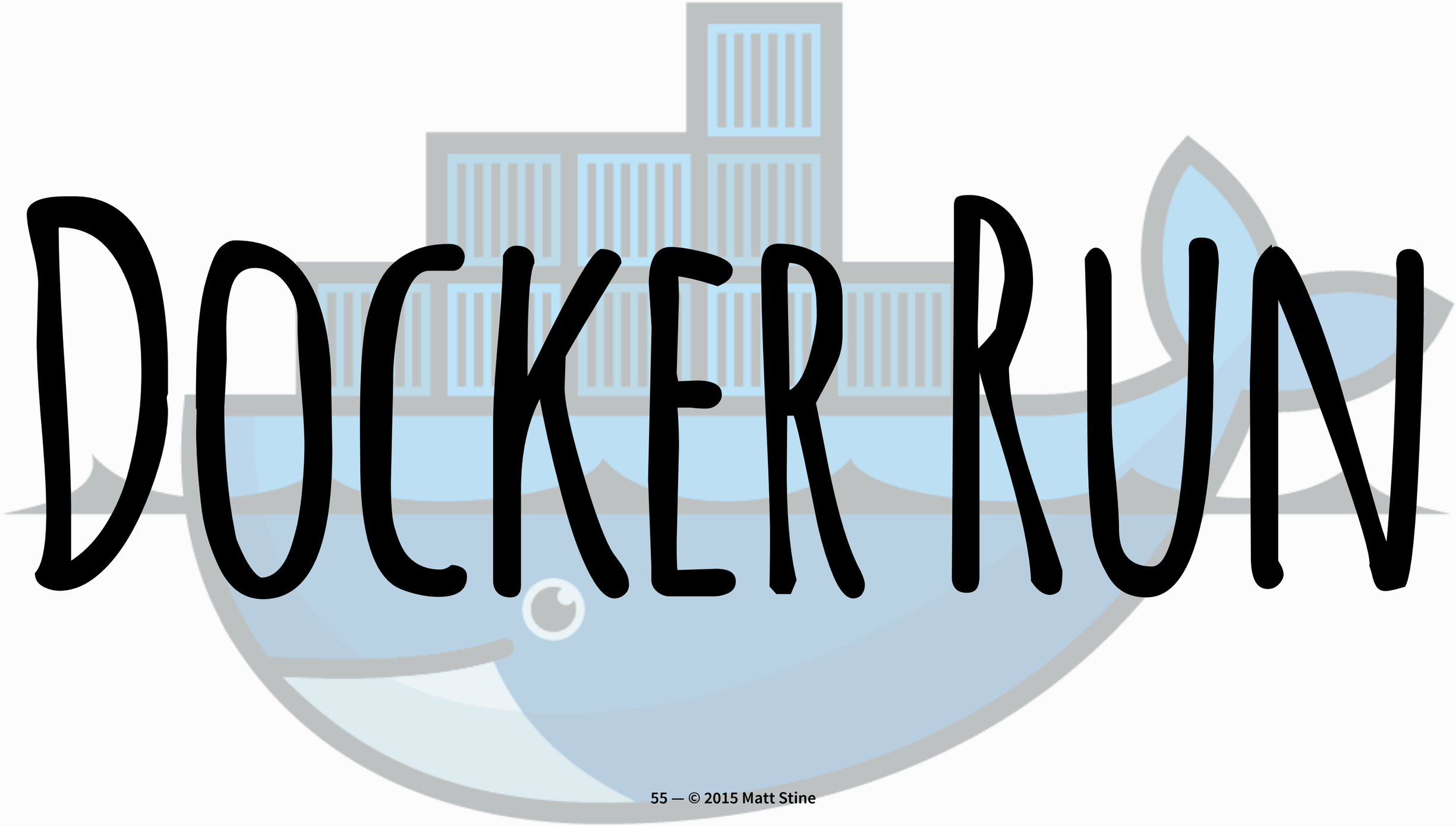
Release

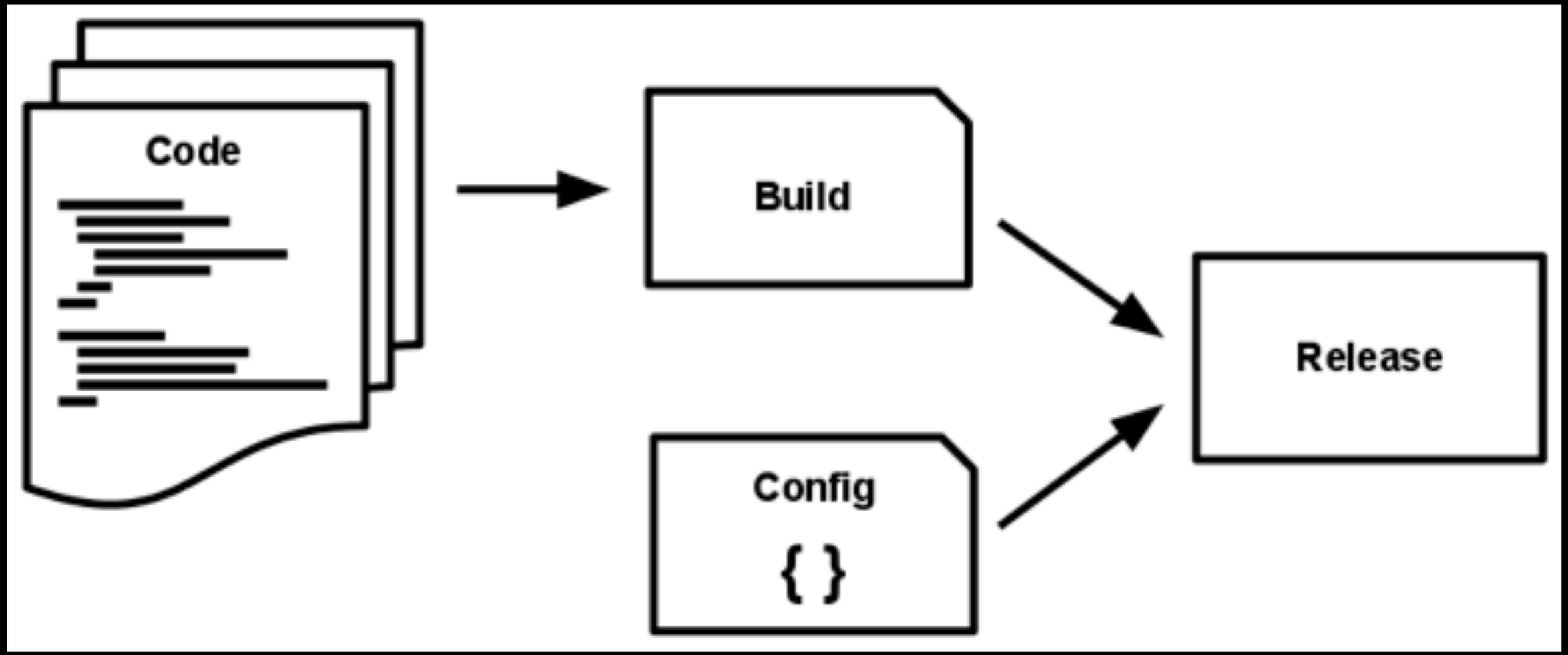
Combine a deployable artifact with configuration and system runtime components (e.g. the JRE).


DOCKER BUILD

Run

Start one or more processes from a specific release.


DOCKER RUN


Code
Build
Release
Config
{ }

IMMUTABILITY

VI. Processes

— Execute the app as one or more stateless processes


ELASTICITY


Ephemerality

Why is State a Problem?

- Disappears with App Container Recycle
- No Shared Memory
- No Persistent or Shared Disk (Usually)

Usual Suspects

- Sessions
- In-App Caches
- Stateful Frameworks (JSF, WebFlow)

Solutions

- Externalize In-Memory State
- Use Persistent Object Stores

Spring Session

http://projects.spring.io/spring-

Config

```
@EnableRedisHttpSession
public class Config {

    @Value("${redis.hostname}")
    String hostName;

    @Value("${redis.port}")
    int port;

    @Bean
    public JedisConnectionFactory connectionFactory() {
        JedisConnectionFactory factory = new JedisConnectionFactory();
        factory.setHostName(this.hostName);
        factory.setPort(this.port);
        return factory;
    }
}
```

Initializer

```
public class Initializer
        extends AbstractHttpSessionApplicationInitializer {

    public Initializer() {
        super(Config.class);
    }
}
```

Then use

HttpSession

like always!

VII. Port binding

— Export services via port binding

Bring Your Own Container

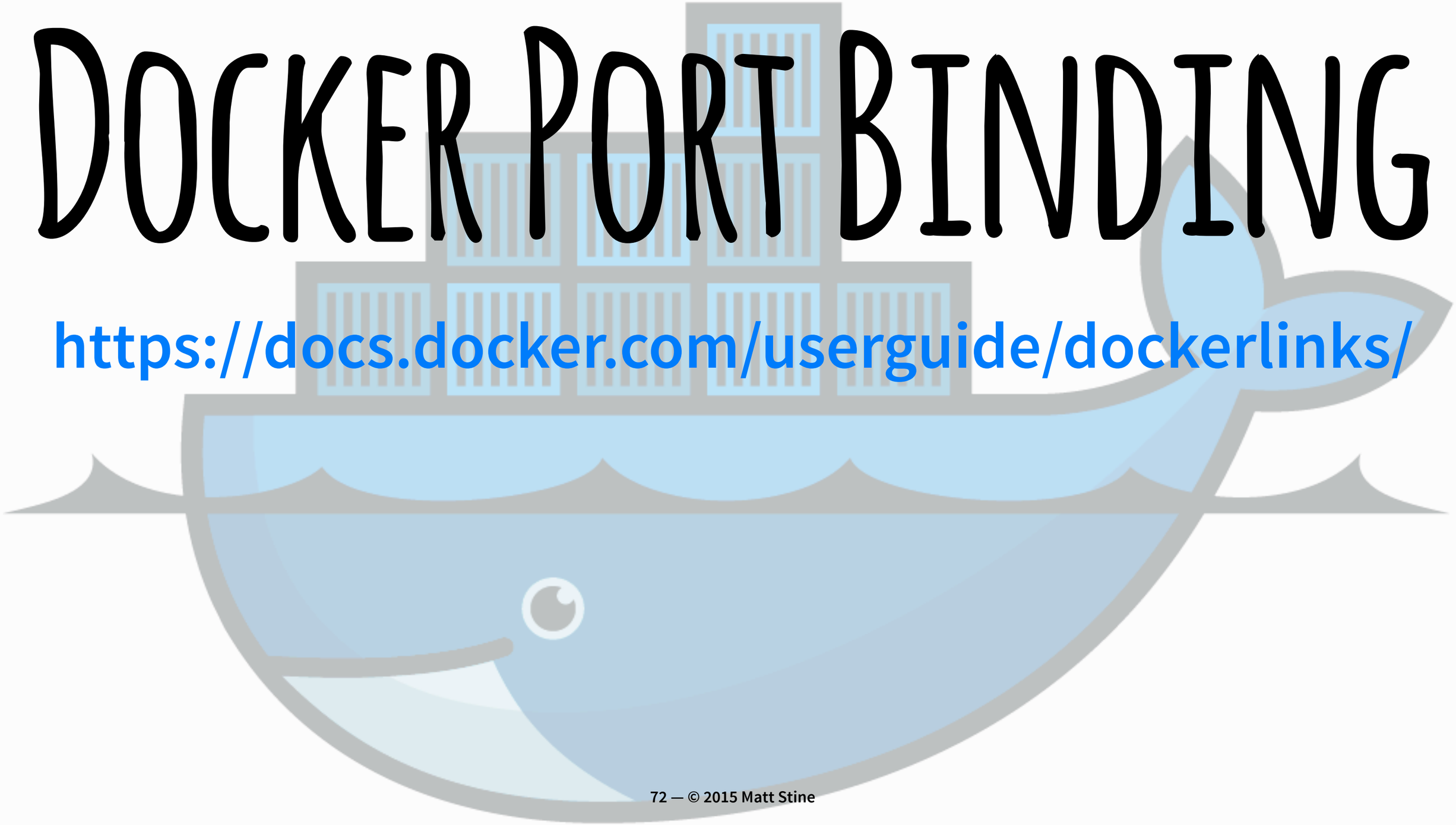
Containers

- Tomcat (SB)
- Jetty (SB, DW)
- Undertow (SB)

Not Only HTTP

Docker Port Binding

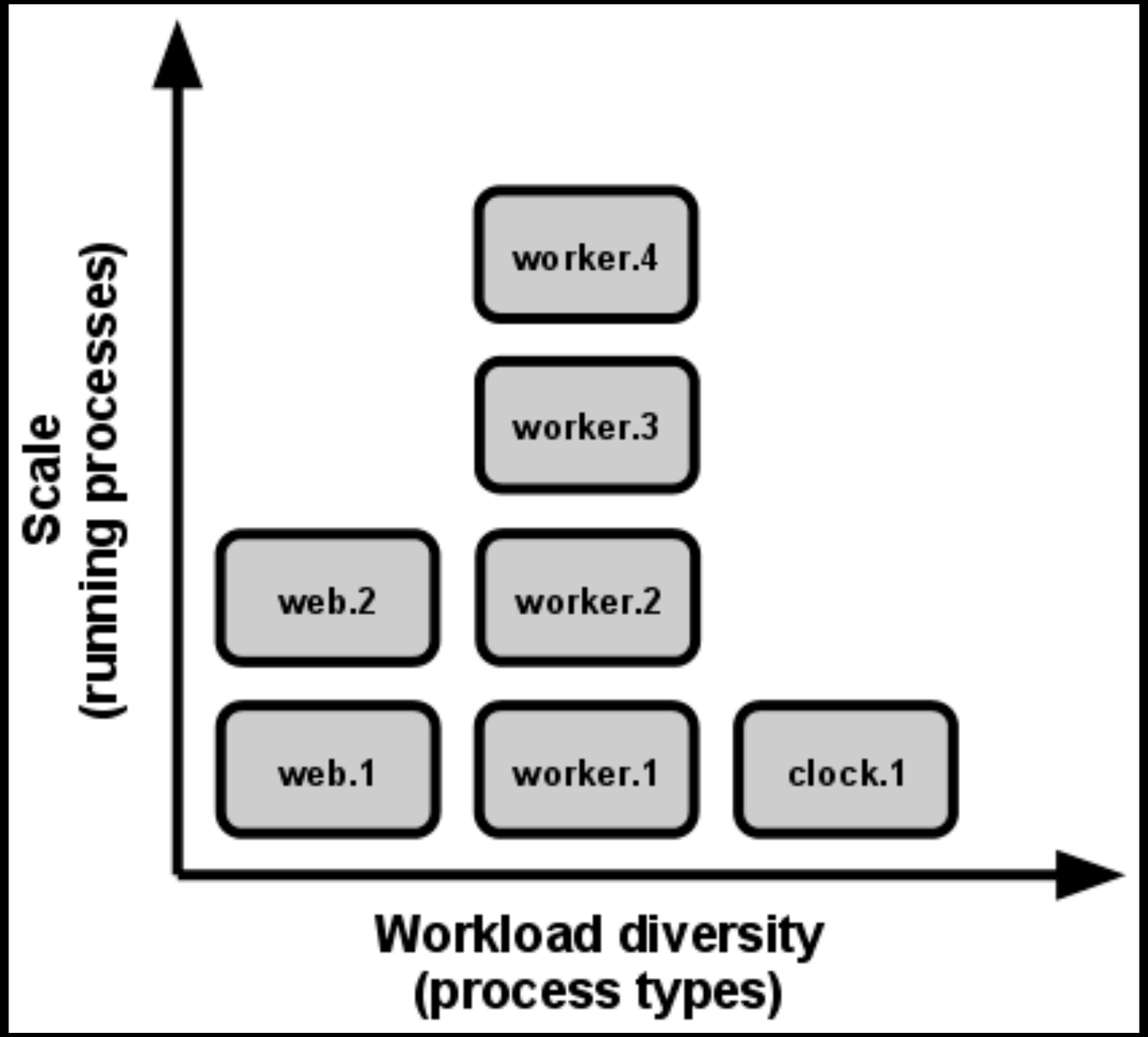
https://docs.docker.com/userguide/dockerlinks/



VIII. Concurrency

— Scale out via the process model

MOAR PROCESSES
LESS MULTIPLEXING


Scale
(running processes)
worker.4
worker.3
web.2
worker.2
web.1
worker.1
clock.1
Workload diversity
(process types)


NANOSERVICES?
http://www.etsy.com/shop/sharpwriter
HEUSER 2012

Horizontal Scale Out Architecture

Delegate Process Management

IX. Disposability

— Maximize robustness with fast startup and graceful shutdown

Start NOW?
Stop NOW?
NO PROBLEM

Minimize Start Up Time


MICROSERVICES!
http://www.etsy.com/shop/sharpwriter
NEUSER 2012

Graceful Shutdown

CRASH WELL

X. Dev/prod parity

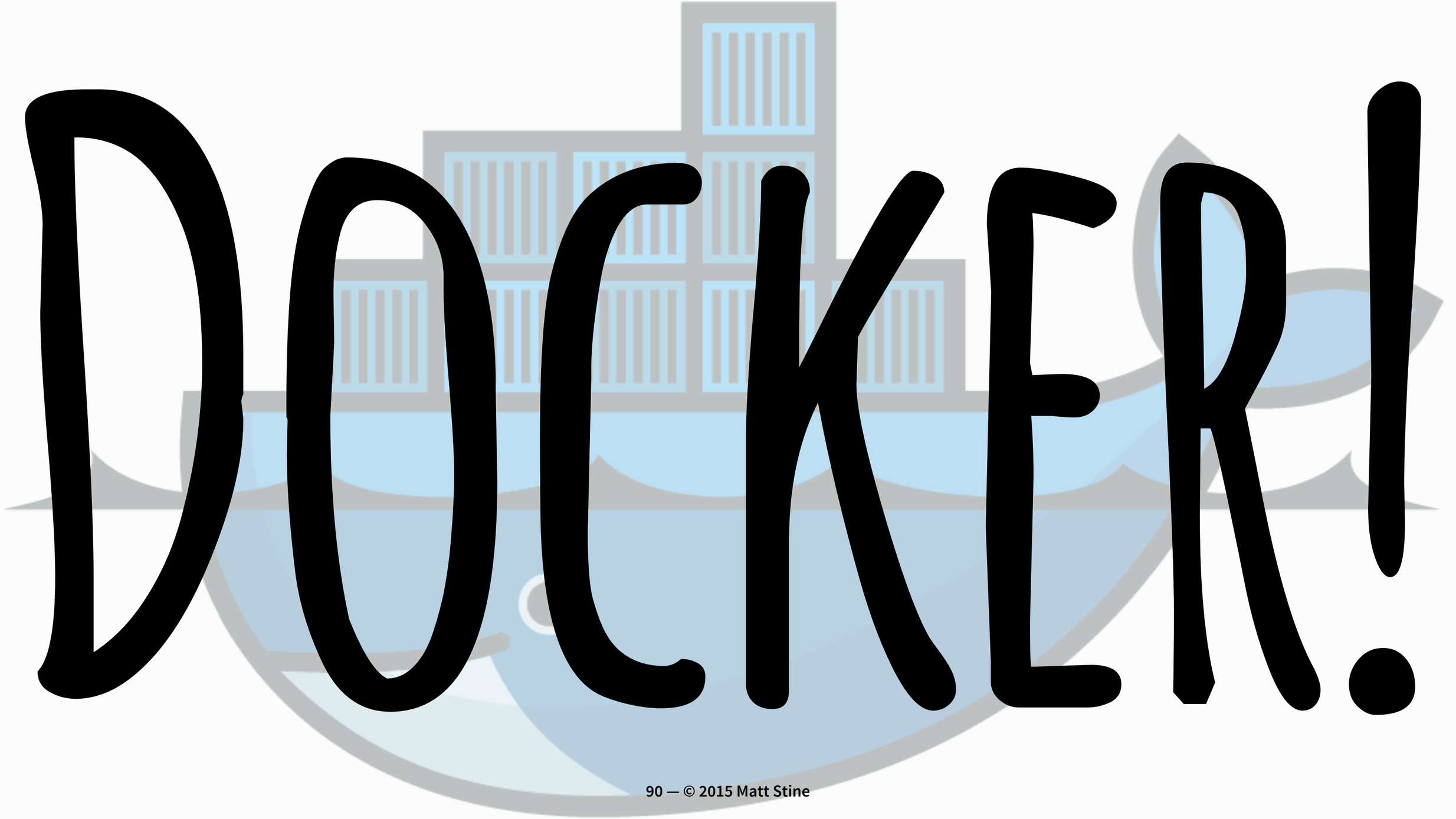
— Keep development, staging, and production as similar as possible

Time Gap

People Gap

Tools Gap

Lean Toward Identical Backing Services


DOCKER!

XI. Logs

— Treat logs as event streams

Stop Managing Log Files!

STDOUT

STDERR

DEV:
LOG TAILS

Prod: Log Capture

Capture / Aggregate:

- Logplex https://github.com/heroku/logplex
  - Fluent https://github.com/fluent/fluentd

Index / Analyze:

- Splunk http://splunk.com
- Logstash http://logstash.net

XII. Admin processes

— Run admin/management tasks as one-off processes

Things like:

- Database Migrations
- Console / REPL
- One-off Scripts

Same:

- Environment (ship admin code!)
- Dependency Isolation (Maven/Gradle)

Spring Boot Shell Demo

Thanks!

Matt Stine (@mstine)

- This Presentation: https://github.com/mstine/nfjs_2015/tree/master/BuildingTwelveFactorApps

Credits

- The Twelve-Factor App: http://12factor.net/
- Elasticity: http://www.flickr.com/photos/karen_d/2944127077
- Ephemerality: http://www.flickr.com/photos/smathur/852322080