

9 Steps to Awesome with Kubernetes

Burr Sutter (burrsutter.com)

github.com/burrsutter/9stepsawesome

Upcoming 3 hour classes/workshops

9 Steps to Awesome with Kubernetes

February 5, 2019

<https://www.safaribooksonline.com/live-training/courses/9-steps-to-awesome-with-kubernetes/0636920231363/>

March 12, 2019 <https://www.safaribooksonline.com/live-training/courses/9-steps-to-awesome-with-kubernetes/0636920231783/>

Istio on Kubernetes: Enter the Service Mesh

February 7, 2019

<https://www.safaribooksonline.com/live-training/courses/istio-on-kubernetes-enter-the-service-mesh/0636920231318/>

March 14, 2019

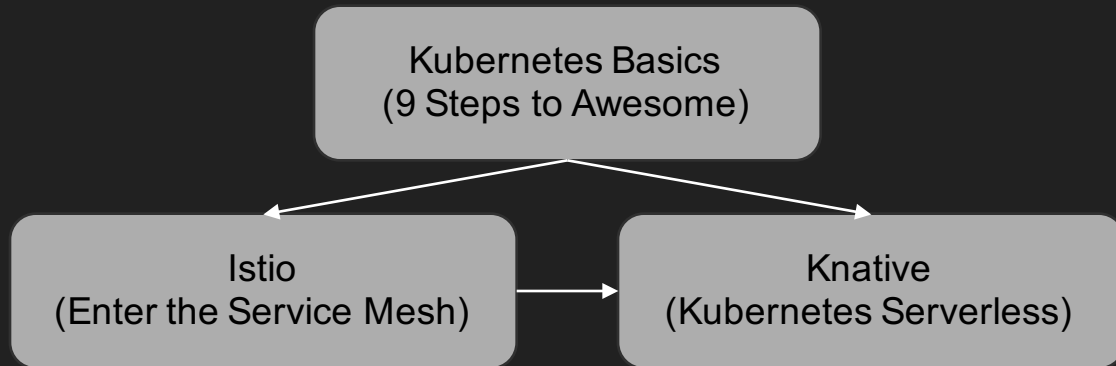
<https://www.safaribooksonline.com/live-training/courses/istio-on-kubernetes-enter-the-service-mesh/0636920231745/>

New 3-hour Deep Dive

Kubernetes Serverless with Knative

March 15, 2019

<https://www.safaribooksonline.com/live-training/courses/kubernetes-serverless-with-knative/0636920257226/>



Recording from Devoxx BE 2018:

https://www.youtube.com/watch?v=ZpbXSdzp_vo



"excellent stuff.. worth 3 hours spending to watch this video.. Really loved it. Will give a complete picture on Kubernetes and how it works" - Pradeep

Setup

https://github.com/burrsutter/9stepsawesome/blob/master/1_installation_started.adoc

Tips on the Virtualization Drivers

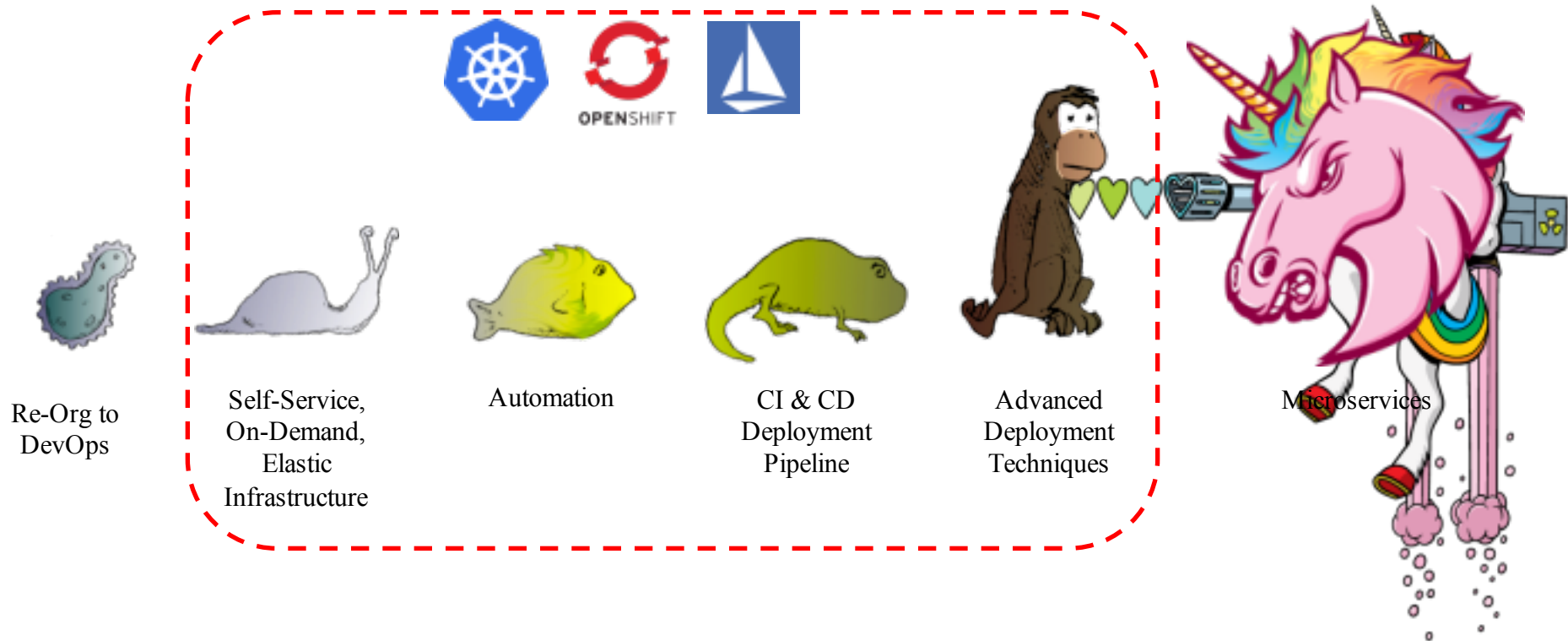
<https://docs.okd.io/latest/minishift/getting-started/setting-up-virtualization-environment.html>

<https://github.com/kubernetes/minikube/blob/master/docs/drivers.md>



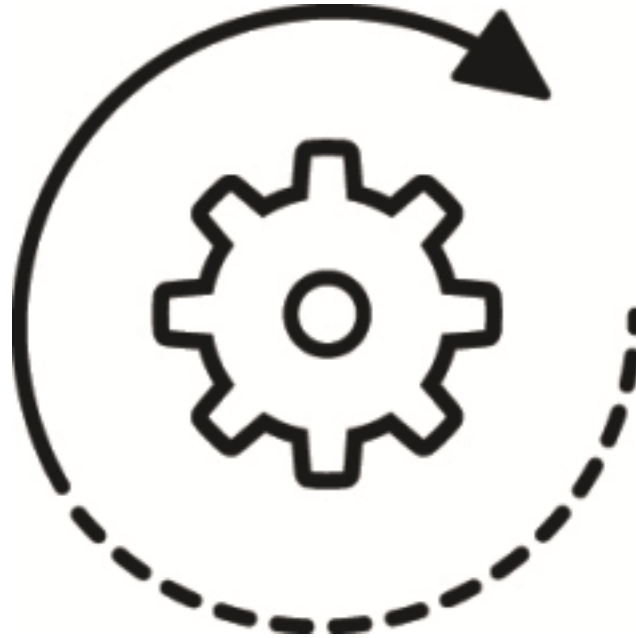
<http://www.g-pop.net/fistoffury.htm>

Your Journey to Awesomeness



Agility

Continuous Delivery, Deployment, Improvement



3 Month Deployment Cycle
3 Months BEFORE you gain Feedback and Learn

9 Steps

0 - Introduction

1 - Installation & Getting Started

2 - Building Images, Running Containers

3 - oc/kubectl exec magic

4 - Logs

5 - Configuration & Environment

6 - Service discovery & load-balancing

7 - Live & Ready

8 - Rolling updates, Canaries, Blue/Green

~~9 - Debugging~~ Databases

Bonus Items

Step 0: Introduction

A Challenge

Have you ever had “/” vs “\” break your app? Or perhaps needed a unique version of a JDBC driver? Or had a datasource with a slightly misspelled JNDI name? Or received a patch for the JVM or app server that broke your code?

Containerize
Your
App

.war or .ear	
Custom Configuration	JDBC driver, datasource, JMS queue, users
Application Server	Weblogic 10.x.y, Tomcat 6.x.y, JBoss EAP 6.x.y
Java Virtual Machine	Java 1.6.6_45 or Java 1.7.0_67
Operating System	Linux Kernel Version & Distribution
Server Hardware	

Email

MyApp.war has been tested with the following

On my Windows 7 desktop

JDK 1.8.43

Wildfly 9

Configuration:

Datasource: MySQLDS

Tested with: mysql-connector-java-5.1.31-bin.jar

Production Environment

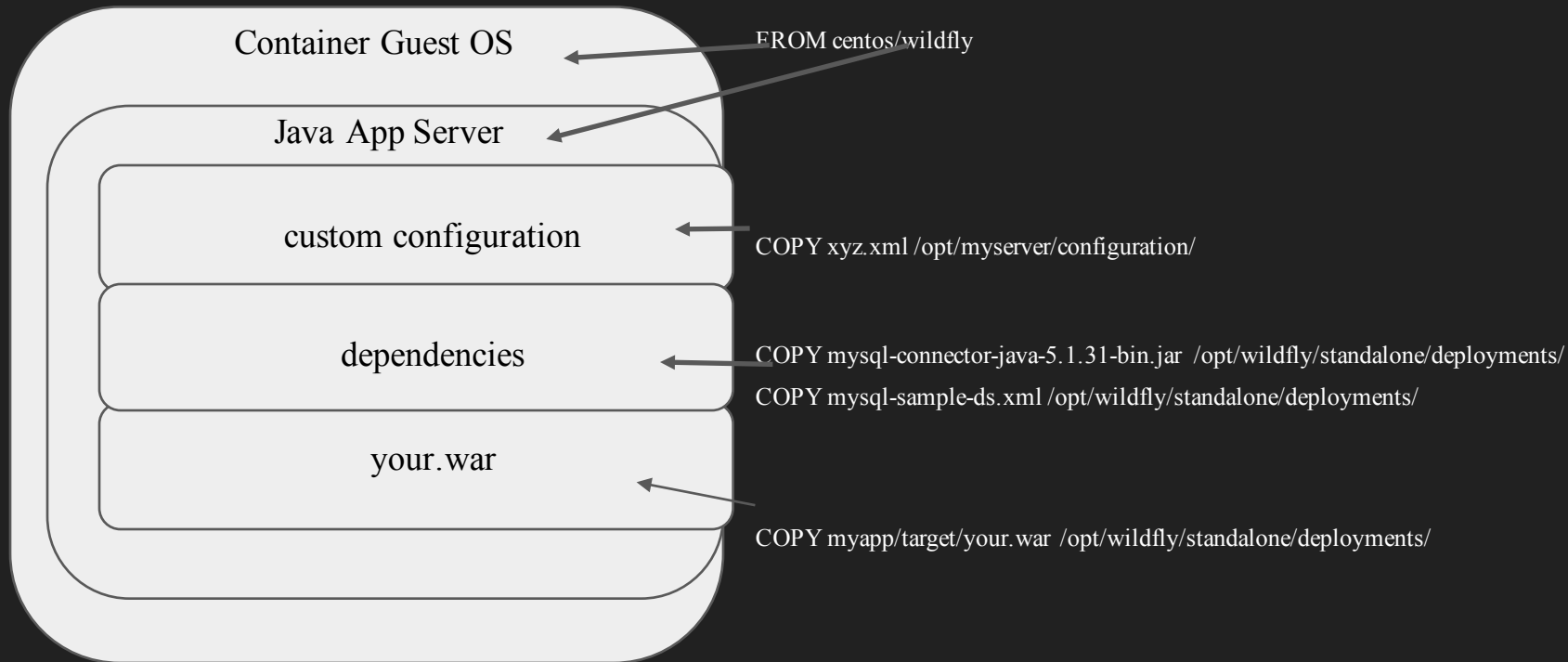
Red Hat Enterprise Linux 6.2

JRE 1.7.3

WebSphere 8.5.5

Oracle 9

Dockerfile



DevOps Challenges for Multiple Containers

- How to scale?
- How to avoid port conflicts?
- How to manage them on multiple hosts?
- What happens if a host has trouble?
- How to keep them running?
- How to update them?
- Rebuild Container Images?





<https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>

Meet Kubernetes

- Greek for “Helmsman,” also the root of the word “Governor” (from latin: gubernator)
- Container orchestrator
- Supports multiple cloud and bare-metal environments
- Inspired by Google’s experience with containers
- Open source, written in Go
- Manage applications, not machines



OPENSIFT

Key Capabilities

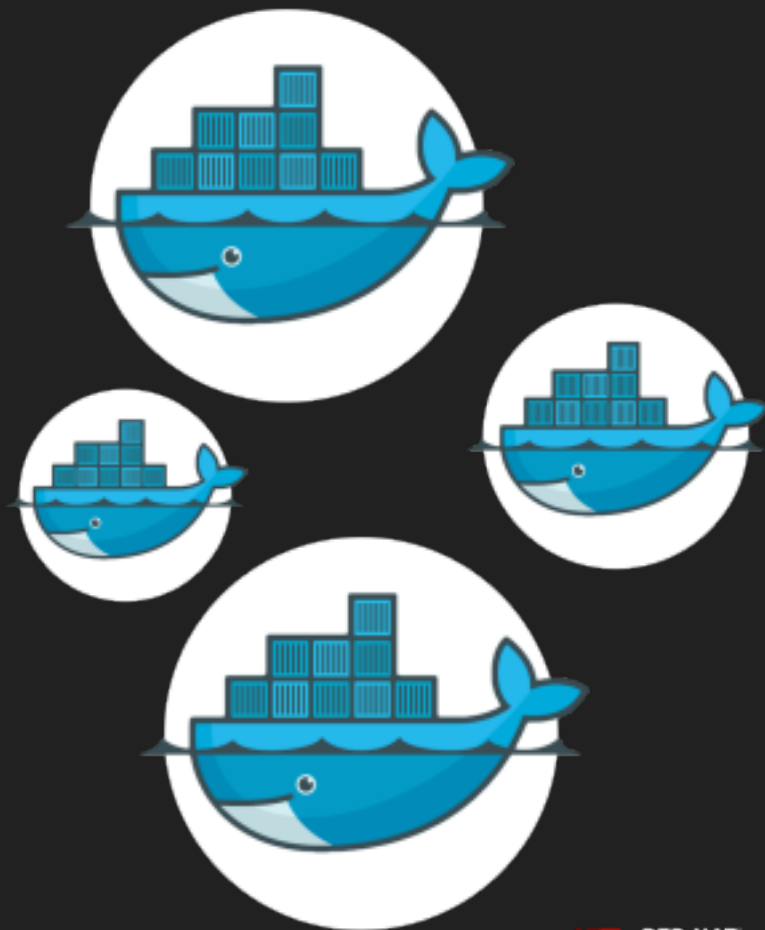
- Self-healing
- Horizontal Manual & Auto Scaling
- Automatic Restarting
- Scheduled across hosts
- Built-in load-balancer
- Rolling upgrades

Pods

A group of whales is commonly referred to as a pod and a pod usually consists a group of whales that have bonded together either because of biological reasons or through friendships developed between two or more whales.

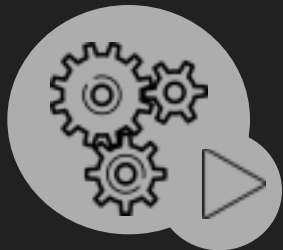
In many cases a typical whale pod consists of anywhere from 2 to 30 whales or more.*

*<http://www.whalefacts.org/what-is-a-group-of-whales-called/>



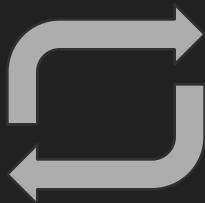
Kubernetes Terms

Pod



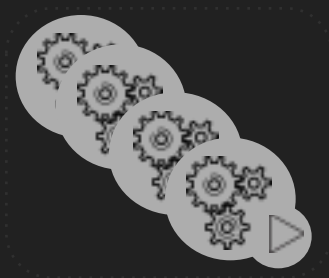
- ✓ 1+ containers
- ✓ Shared IP
- ✓ Shared storage (ephemeral)
- ✓ Shared resources
- ✓ Shared lifecycle

Replicaset/ Deployment



- ✓ The Desired State - replicas, pod template: health checks, resources, image

Service



- ✓ Grouping of pods (acting as one) has stable virtual IP and DNS name

Persistent Volume



- ✓ Network available storage
- ✓ PVs and PVCs

Label



- ✓ Key/Value pairs associated with Kubernetes objects (env=production)

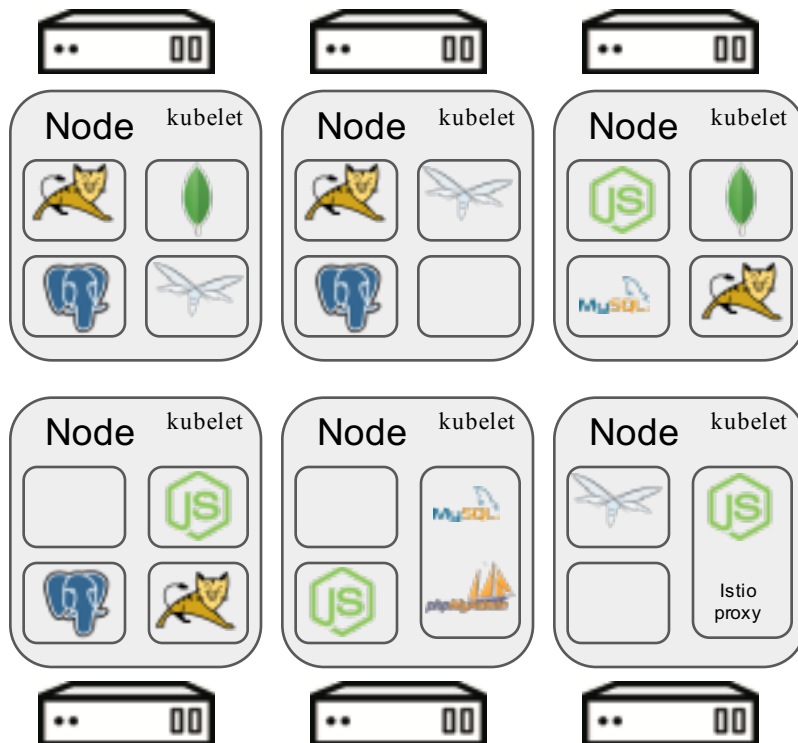
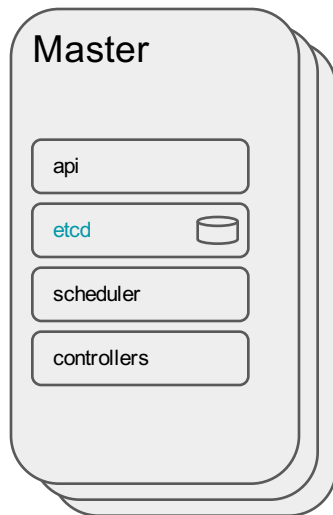
Kubernetes Cluster



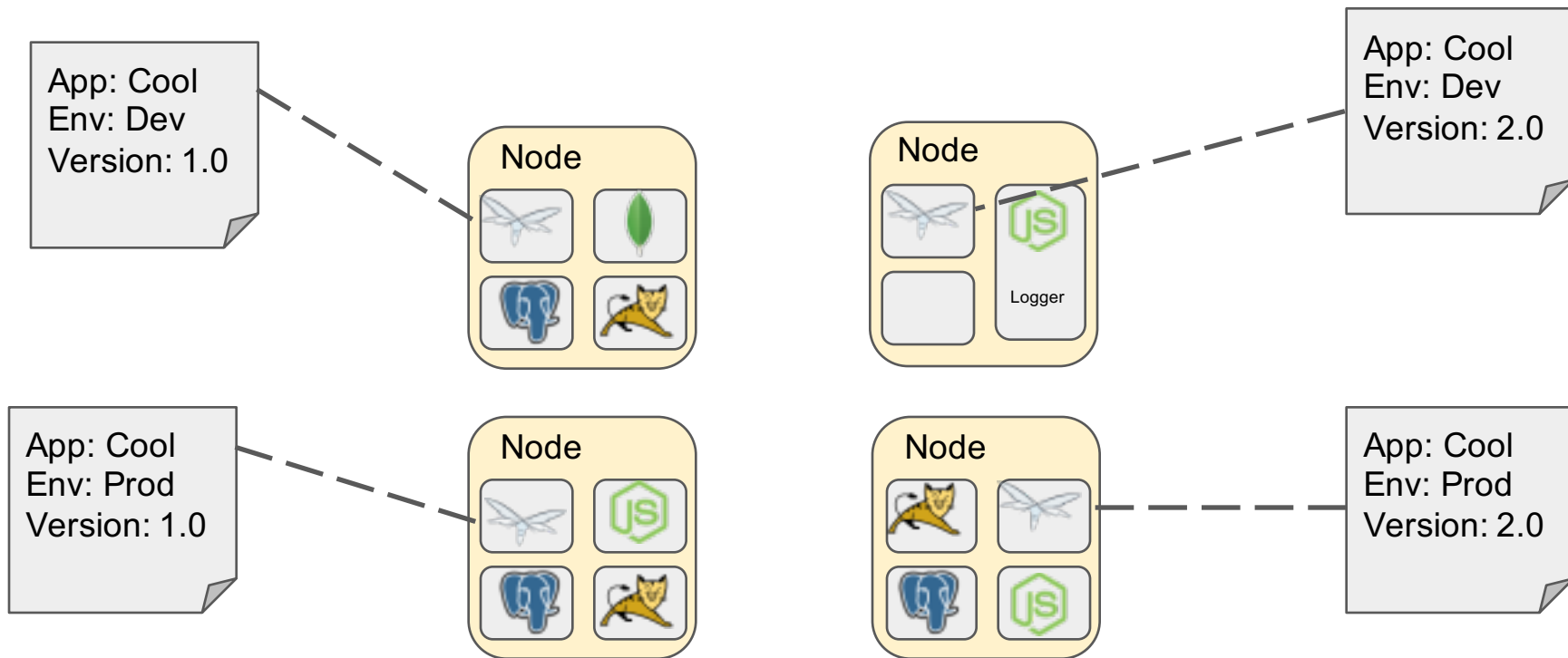
Dev



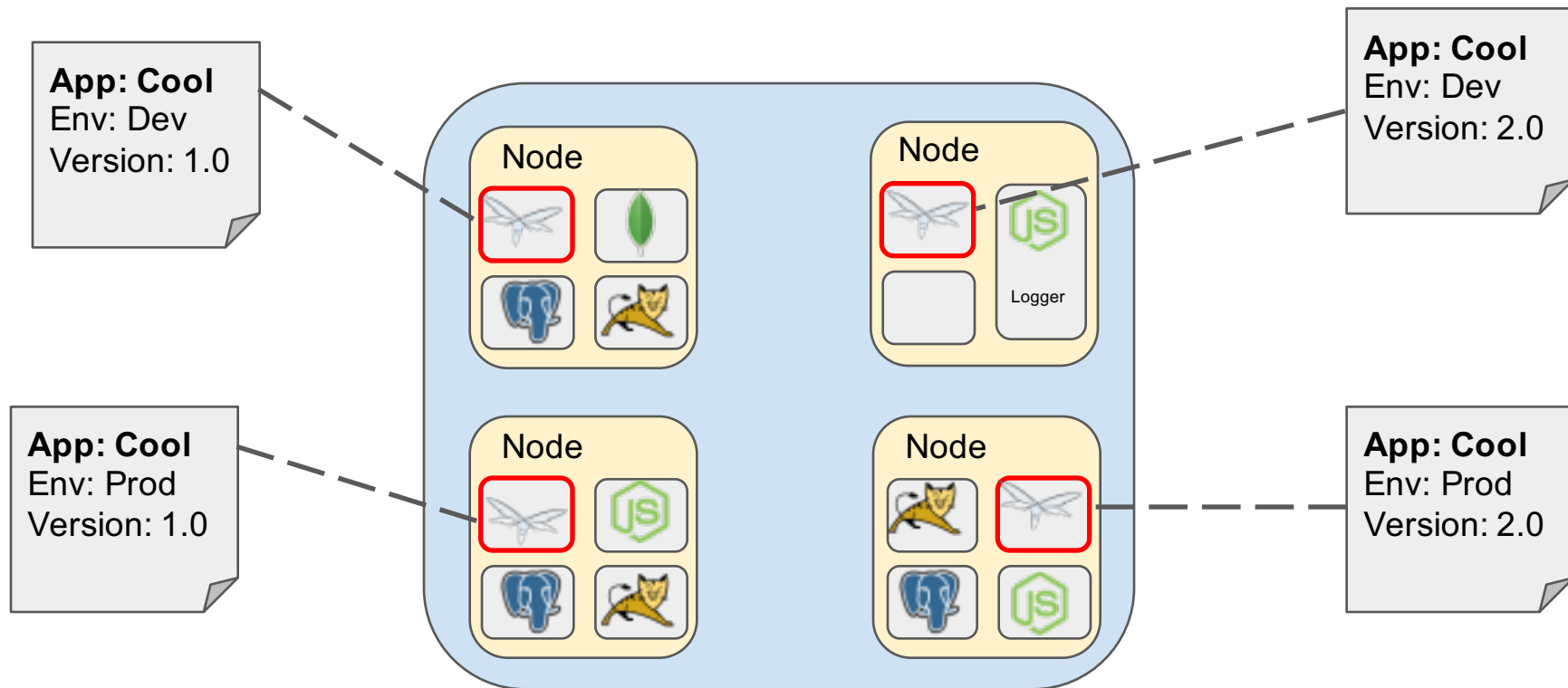
Ops



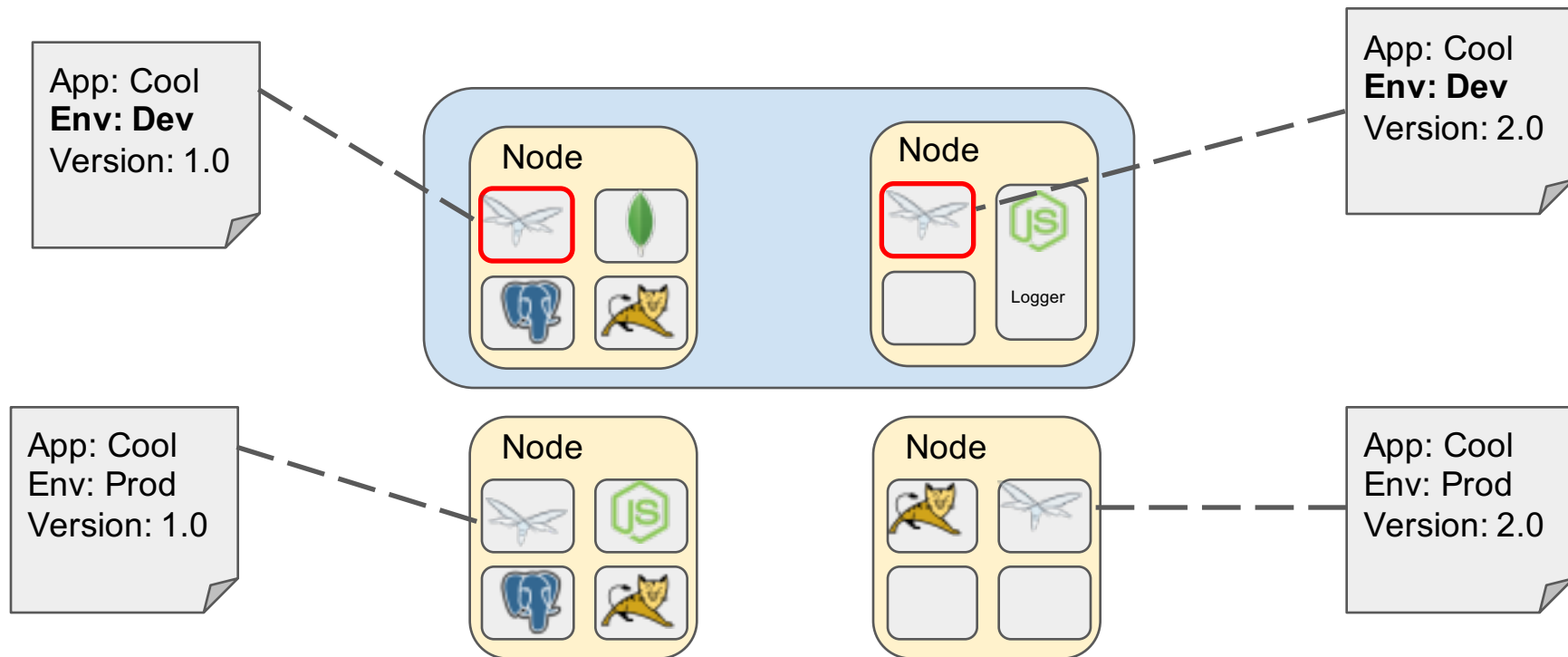
Labels



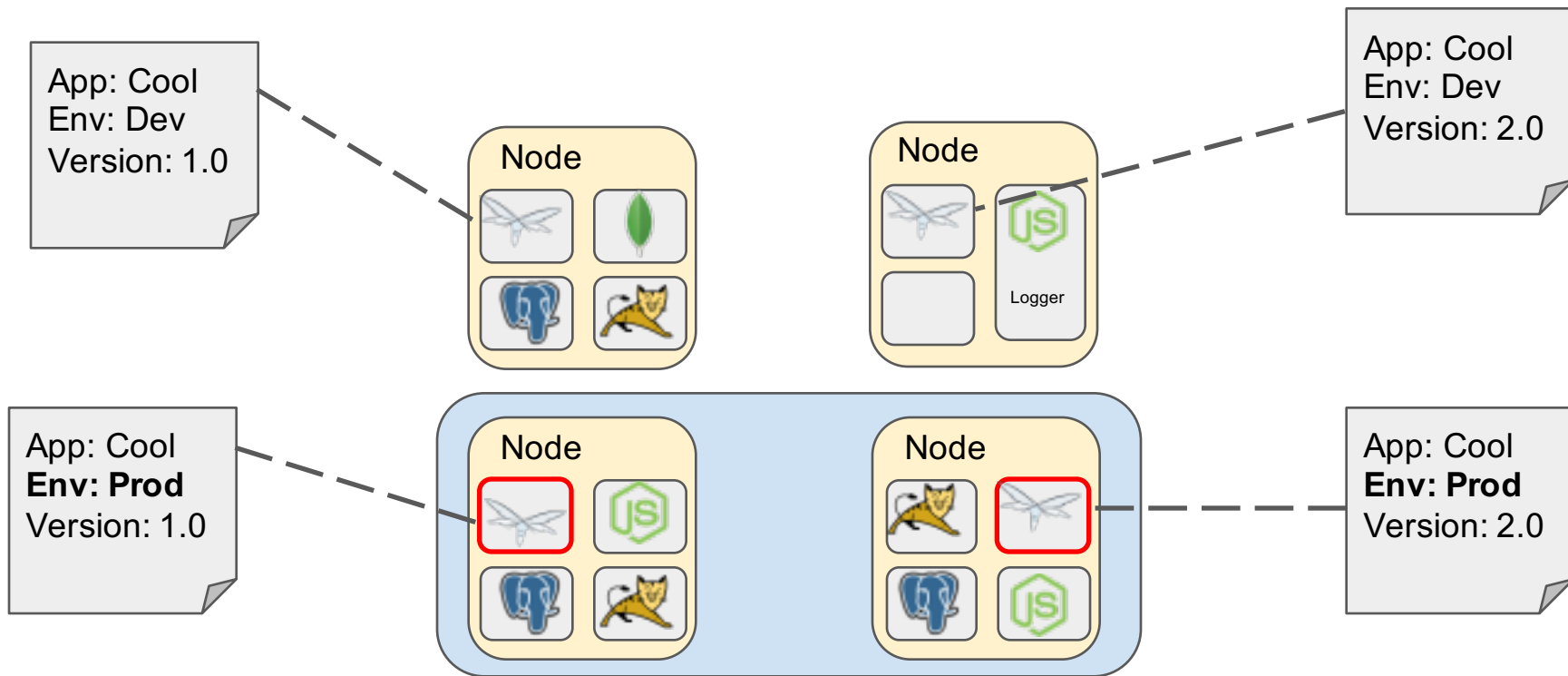
Labels App:Cool



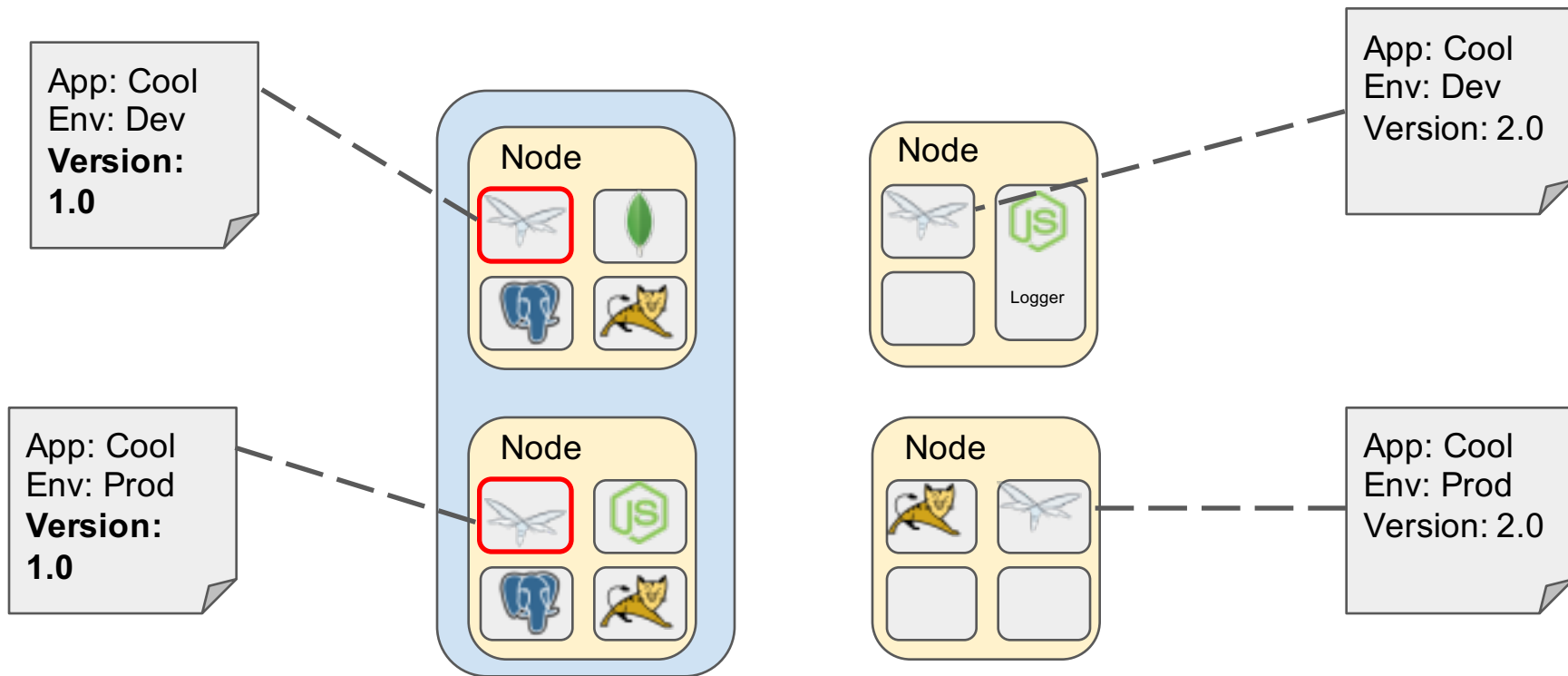
Labels Env:Dev

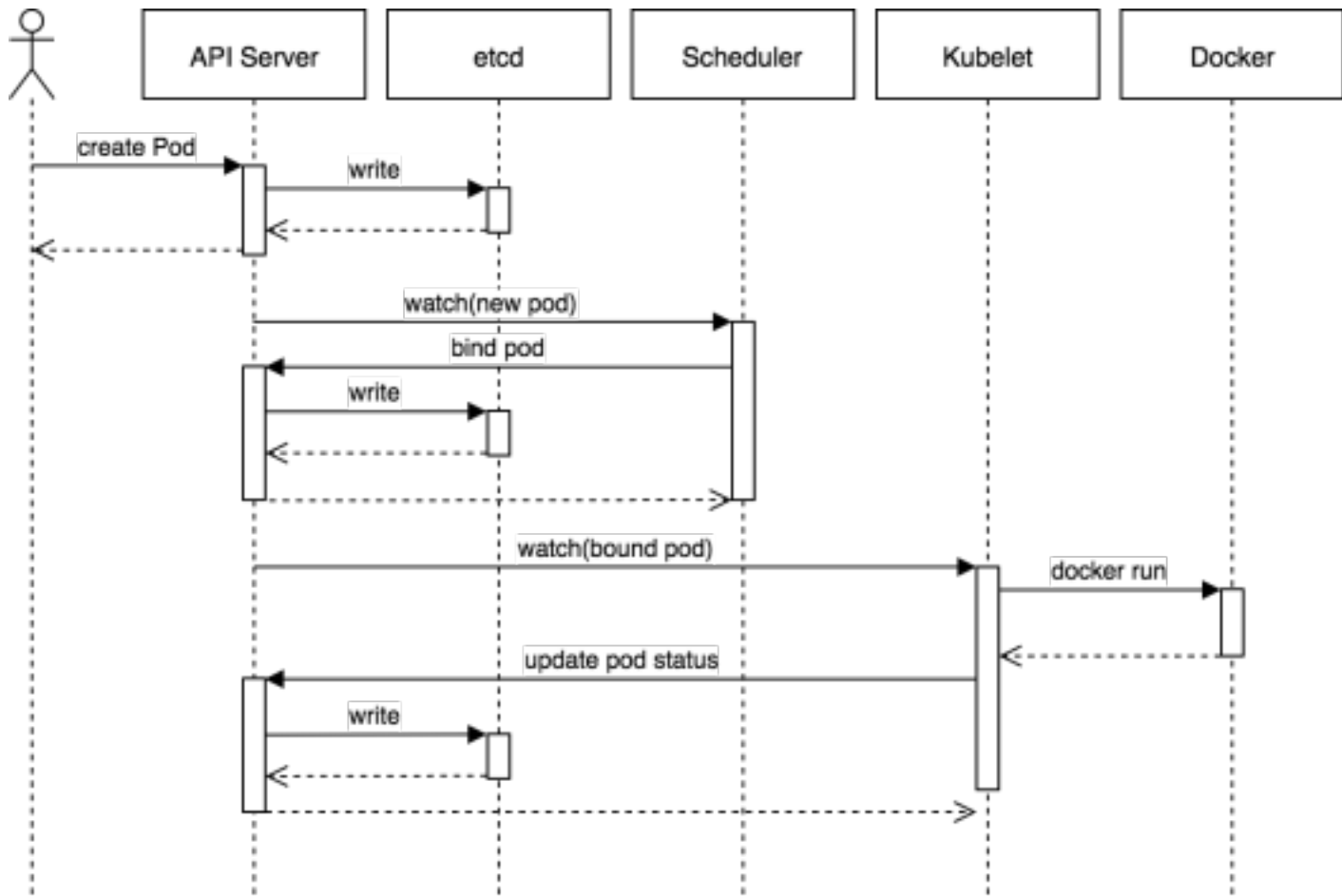


Labels Env:Prod



Labels Version:1.0





<https://dzone.com/articles/kubernetes-lifecycle-of-a-pod>

<https://blog.heptio.com/core-kubernetes-jazz-improv-over-orchestration-a7903ea92ca>

kubectl commands

<https://kubernetes.io/docs/user-guide/kubectl/>

<https://kubernetes.io/docs/reference/kubectl/cheatsheet>

```
kubectl get namespaces
```

```
kubectl get pods -n mynamespace
```

```
kubectl run myvertex --image=burr/myvertex:v1 --port=8080
```

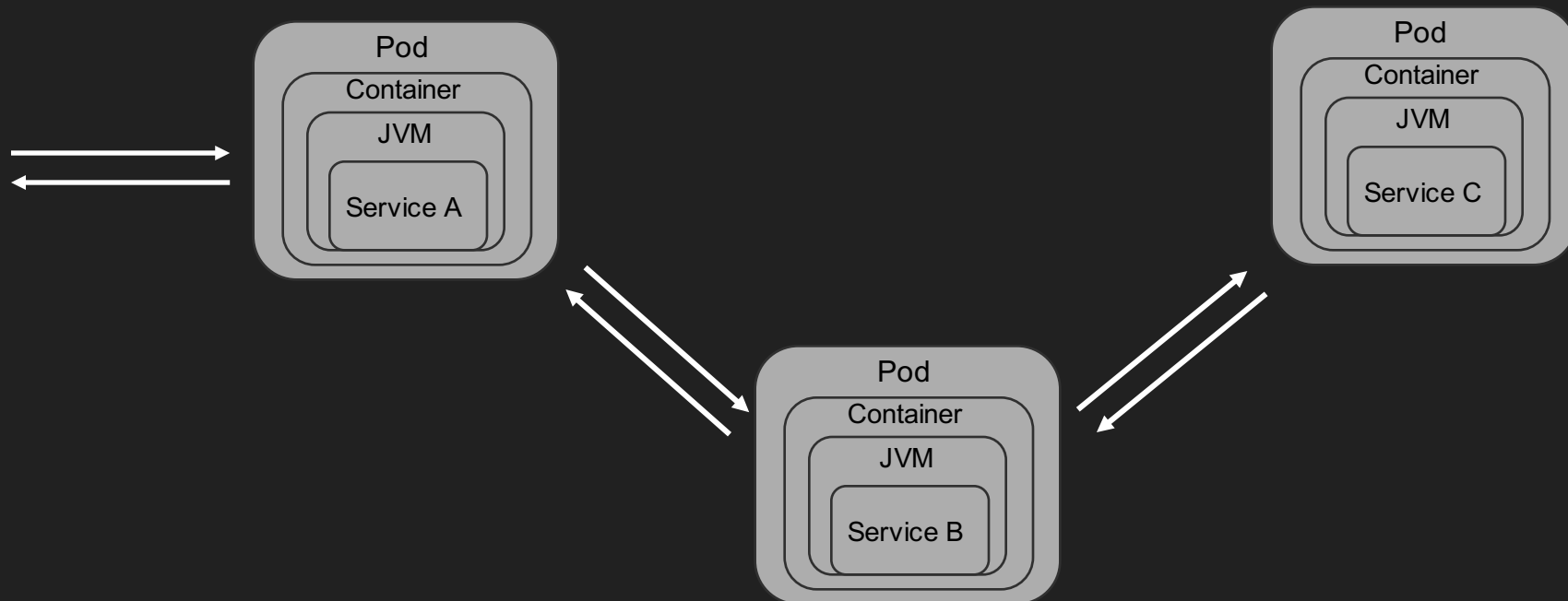
```
kubectl logs myvertex-kk605
```

```
kubectl expose deployment --port=8080 myvertex --type=LoadBalancer
```

```
kubectl scale deployment myvertex --replicas=3
```

```
kubectl set image deployment/myvertex myvertex=burr/myvertex:v2
```

Microservices == Distributed Computing



java -jar myapp.jar

DropWizard

www.dropwizard.io

JAX-RS API

First to market

DropWizard Metrics



Vert.x

vertx.io

Reactive
Async
non-blocking

RxJava

vertx run
myhttp.java



Spring Boot

[spring.io/projects/
spring-boot](http://spring.io/projects/spring-boot)

Spring API
(@RestController)

‘Starter’ POMs:
start.spring.io



Thorntail

thorntail.io

MicroProfile.io

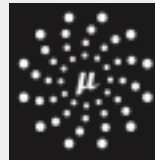
‘Starter’ POMs:
thorntail.io/generator



Micronaut

micronaut.io

"Compile-time"
dependency
injection





- Defines **open source** Java **microservices** specifications
- Industry Collaboration - Red Hat, IBM, Payara, Tomitribe, London Java Community, SouJava, Oracle, Hazelcast, Fujitsu, SmartBear...
- **WildFly Swarm** is **Red Hat's** implementation
- Minimum footprint for Enterprise Java cloud-native services (v1.3) :

JSON-P 1.0

Health
Check

JWT
Propagation

Config

OpenAPI

REST
Client

CDI 1.2

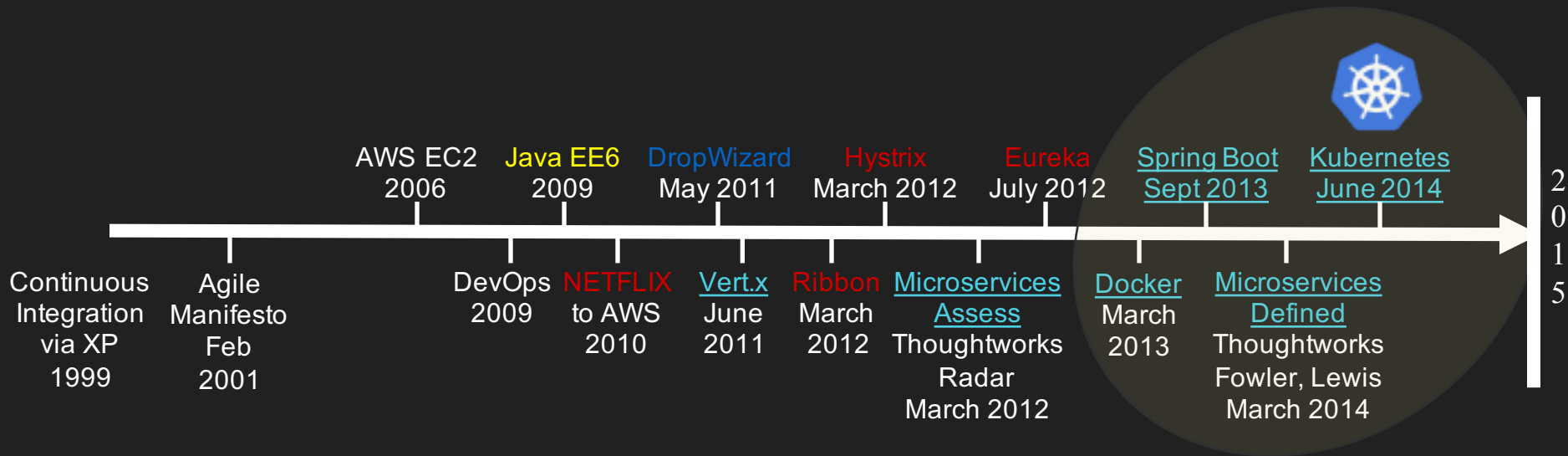
JAX-RS 2.0

Fault
Tolerance

Metrics

Open
Tracing

History of Microservices



NETFLIX | OSS





2015

Launch 1000+
Containers

Audience
Claims a
Container

<https://www.youtube.com/watch?v=GCtpncA0Ea0&feature=youtu.be&t=1031>

Java Microservices Platform circa 2015



NETFLIX Ribbon



Better Microservices Platform circa 2016



Better Microservices Platform circa 2017

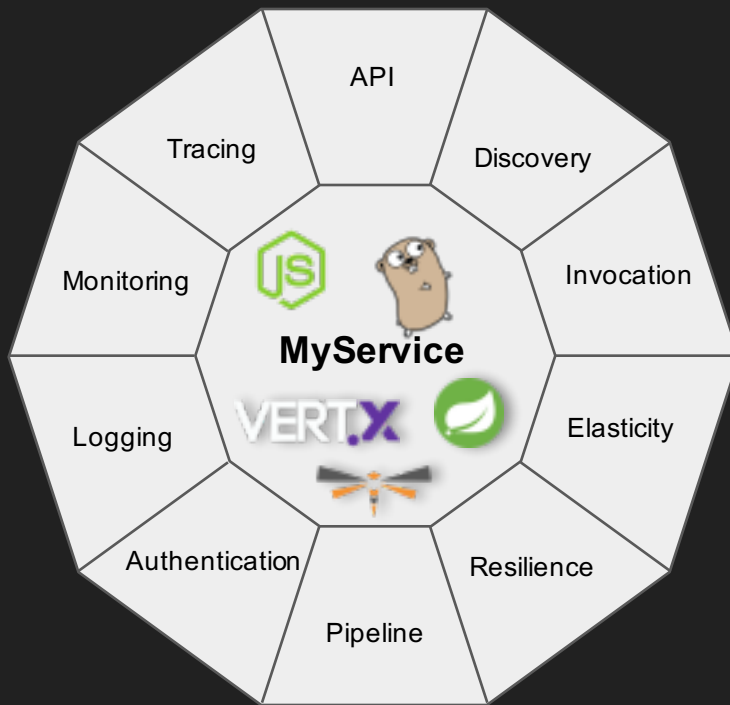


Better Microservices Platform circa 2018



Polyglot Microservices Platform circa 2018





OPENSIFT

Polyglot Microservices Platform

9 Steps

Step 1: Installation

Step 1: Installation

Lots of Options

1. Localhost Development
 - a. [minikube](#) ([kubectl](#)) ([docs](#))
 - b. [minishift](#) ([oc](#)) ([docs](#))
2. Hosted Kubernetes Cluster
 - a. [GKE from Google Cloud Platform](#)
 - b. [AKS from Microsoft Azure](#)
 - c. [EKS from Amazon Web Services](#)
3. Many more... <https://kubernetes.io/docs/setup/pick-right-solution/#turnkey-cloud-solutions>

Step 1 Demo

https://github.com/burrsutter/9stepsawesome/blob/master/1_installation_started.adoc

Step 2: Building Images

Running your App on Kubernetes

1. Find a **base Image**: Docker Hub, Quay.io, gcr.io, access.redhat.com/containers
2. Craft your **Dockerfile**
3. Build your Image: `docker build -t mystuff/myimage:v1 .`
4. `kubectl create -f myDeployment.yml`
5. `kubectl create -f myService.yml`
6. Expose a URL via your Kubernetes distribution's load-balancer

Step 2: Building Images

Options Include:

- A. **docker build** then `kubectrl run` or `kubectrl create -f deploy.yml`
- B. [Fabric8 maven plugin](#) ([fabric8.io](#))
- C. [Jib](#) - Maven/Gradle plugin
- D. [s2i](#) - source to image
- E. No d-o-c-k-e-r
 - a. Red Hat's [podman](#), Google's [kaniko](#), Uber's [makisu](#)
- F. [Buildpacks](#) - similar to Heroku & Cloud Foundry

Managing Yaml

Options Include:

- A. [Helm Charts](#) - but for Tiller (Tiller going away in 3)
- B. [Kompose](#) - converts docker-compose.yml to kubernetes yaml
- C. ~~Ksonnet~~ - templating for Kubernetes manifests ([archived](#))
- D. Kapitan - templating for Kubernetes manifests ([helm](#), [kapitan](#), [kustomize](#))
- E. Kustomize - templating for Kubernetes manifests ([Helm vs Kustomize](#))

Dockerfile for Java projects

```
FROM fabric8/java-jboss-openjdk8-jdk:1.4.0
ENV JAVA_APP_DIR=/deployments
EXPOSE 8080 8778 9779
COPY target/my.jar /deployments/
```

<https://github.com/fabric8io-images/java/tree/master/images/jboss/openjdk8/jdk>

docker build, kubectl run

```
minikube(docker-env) or minishift(docker-env)
docker build -t burr/myimage:v1 .
docker run -it -p 8080:8080 burr/myimage:v1
curl $(minishift ip):8080
# now run it on Kubernetes
kubectl run myapp --image burr/myimage:v1 --port 8080
kubectl expose deployment --port=8080 myapp --type=LoadBalancer
oc expose service myapp
curl myapp-stuff.$(minishift ip).nip.io
# scale up
kubectl scale --replicas=2 deploy/myapp
# create an updated image
docker build -t burr/myimage:v2 .
# rollout update
kubectl set image deployment/myapp myapp=burr/myimage:v2
```

Fabric8 Maven Plugin

<https://maven.fabric8.io/#fabric8:setup> (no **Dockerfile**, **Deployment.yml**)

```
oc new-project stuff (or kubectl create namespace)
```

```
mvn clean compile package
```

```
mvn io.fabric8:fabric8-maven-plugin:3.5.40:setup
```

```
mvn fabric8:deploy
```

Do NOT Java + Docker == FAIL

Slides: bit.ly/javadockerfail

[Recording from JBCNConf 2017](#)

```
docker run -m 100MB openjdk:8u121 java -  
XshowSettings:vm -version
```

```
docker run -m 100MB openjdk:8u131 java -  
XX:+UnlockExperimentalVMOptions -  
XX:+UseCGroupMemoryLimitForHeap -XshowSettings:vm -  
version
```

Step 2 Demo

https://github.com/burrsutter/9stepsawesome/blob/master/2_building_running.adoc
<https://github.com/burrsutter/kube4docker>

Step 3: oc or kubectl exec

Step 3: oc or kubectl exec

"ssh" into your containers and explore

```
kubectl get pods --namespace=microworld
```

```
kubectl exec -it --namespace=microworld $POD cat /sys/fs/cgroup/memory/memory.limit_in_bytes
```

Or

```
kubectl exec -it --namespace=microworld microspringboot1-2-nz8f8 /bin/bash
```

```
ps -ef | grep java
```

Note: the following apply if using the fabric8 generated image, otherwise consult your Dockerfile

```
java -version
```

```
javac -version
```

```
# now find that fat jar
```

```
find / -name *.jar
```

```
cd /deployments (based on use of the fabric8 maven plugin)
```

```
ls
```

```
exit
```

Step 3 Demo

https://github.com/burrsutter/9stepsawesome/blob/master/3_kubectl_exec.adoc
<https://github.com/burrsutter/javadockerfail>

Step 4: logs

Step 4: logs

```
System.out.println("Where am I?");
```

```
Or console.log("Node logs");
```

```
kubectl get pods
```

```
kubectl logs microspringboot1-2-nz8f8
```

```
kubectl logs microspringboot1-2-nz8f8 -p # last failed pod
```

```
OR ./kubetail.sh
```

<https://raw.githubusercontent.com/johanhaleby/kubetail/master/kubetail>

```
OR stern (brew install stern)
```

<https://github.com/wercker/stern>

```
OR kail (https://github.com/boz/kail)
```

Step 4 Demo

https://github.com/burrsutter/9stepsawesome/blob/master/4_logs.adoc

Step 5: env and configmaps

Step 5: env vars & configmaps

An app's config is everything that is likely to vary between deploys (staging, production, developer environments, etc). [12 Factor Apps](#)

```
kubectl set env deployment/myboot DBCONN="jdbc:sqlserver://45.91.12.123:1443;user=MyUserName;password=*****;"
```

```
kubectl create cm my-config --from-env-file=config/some.properties
```

Step 5 Demo

https://github.com/burrsutter/9stepsawesome/blob/master/5_configuration.adoc

Step 6: service discovery & load-balancing

Step 6: Service Discovery

1. Services are internal to the cluster and can be mapped to pods via a label selector
2. Just refer to a Service by its name, it is just DNS

```
String url = "http://producer:8080/";
```

```
ResponseEntity<String> response =  
    restTemplate.getForEntity(url, String.class);
```


Step 6 Demo

https://github.com/burrsutter/9stepsawesome/blob/master/6_discovery.adoc

Step 7: Live and Ready

Step 7: Live and Ready

```
kubectl create -f Deployment.yml
```

```
resources:
  requests:
    memory: "300Mi"
    cpu: "250m" # 1/4 core
  limits:
    memory: "400Mi"
    cpu: "1000m" # 1 core
livenessProbe:
  httpGet:
    port: 8080
    path: /
  initialDelaySeconds: 10
  periodSeconds: 5
  timeoutSeconds: 2
readinessProbe:
  httpGet:
    path: /health
    port: 8080
  initialDelaySeconds: 10
  periodSeconds: 3
```

Step 7 Demo

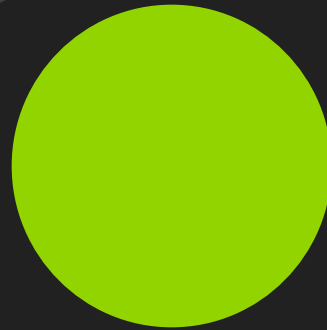
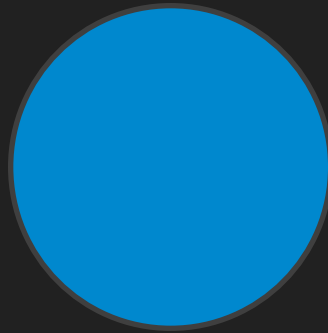
https://github.com/burrsutter/9stepsawesome/blob/master/7_live_ready.adoc

<https://github.com/redhat-developer-demos/popular-movie-store/blob/master/src/main/java/org/workspace7/moviestore/controller/HomeController.java#L158-L159>

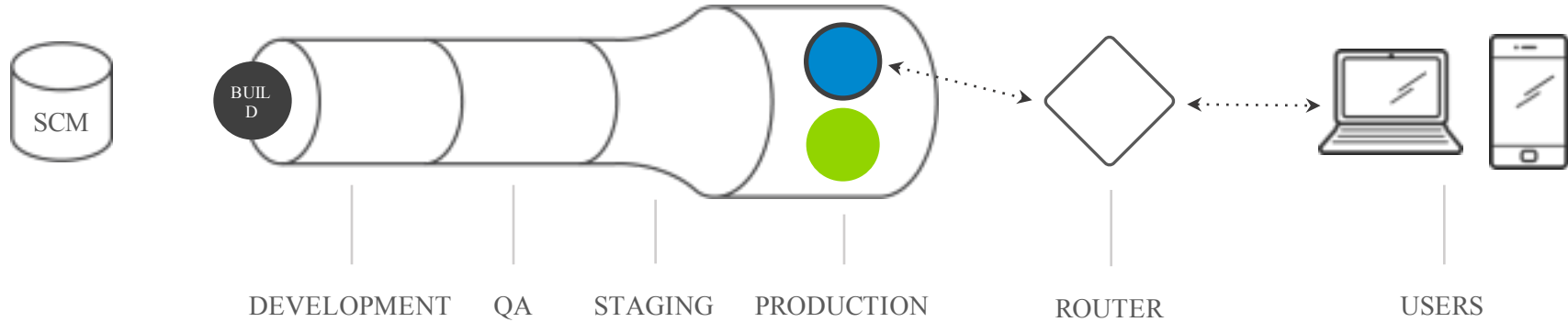
<https://github.com/redhat-developer-demos/popular-movie-store/blob/master/src/main/fabric8/deployment.yml#L38-L42>

Step 8: Rolling Updates, Blue/Green Canary

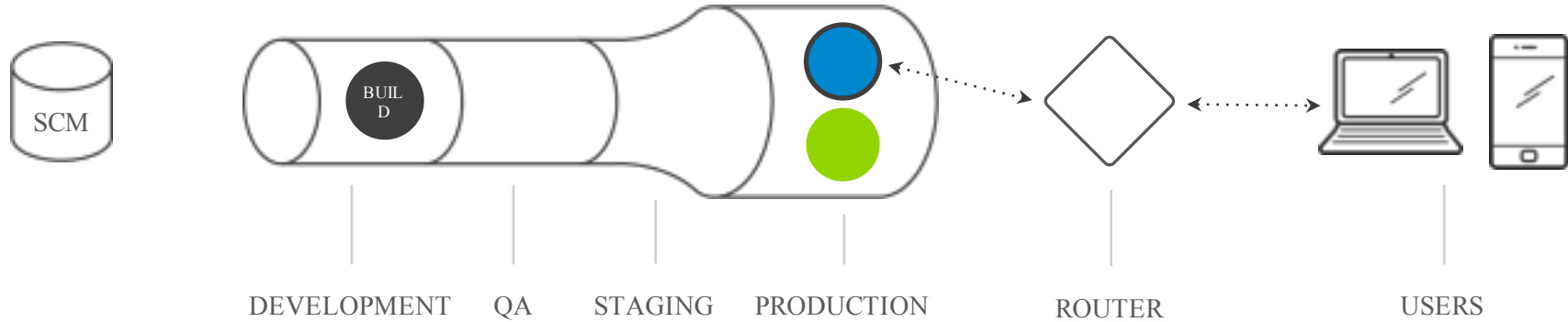
Blue/Green Deployment



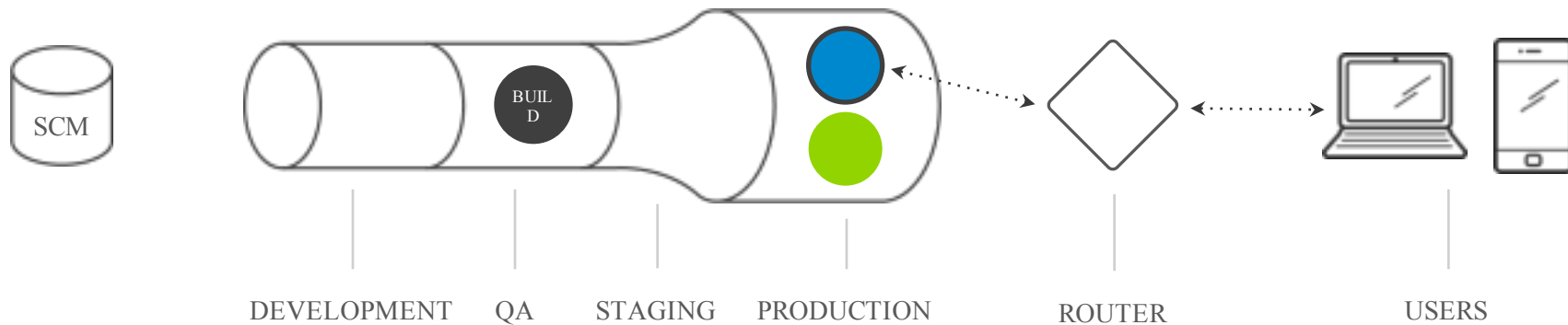
Blue/Green Deployment



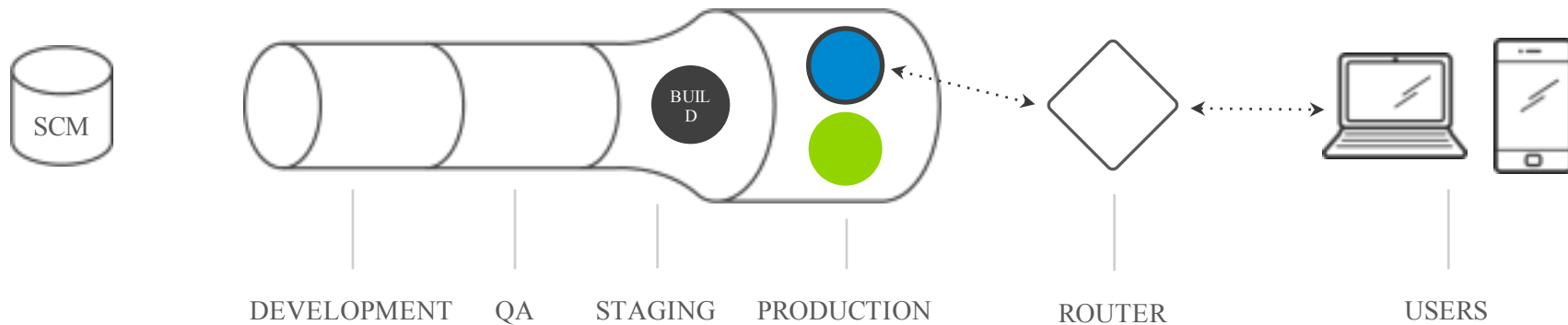
Blue/Green Deployment



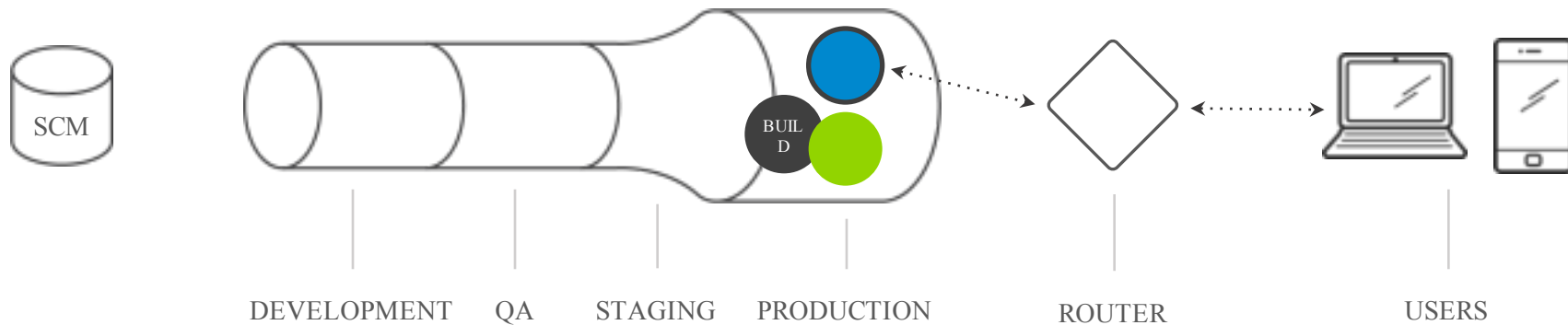
Blue/Green Deployment



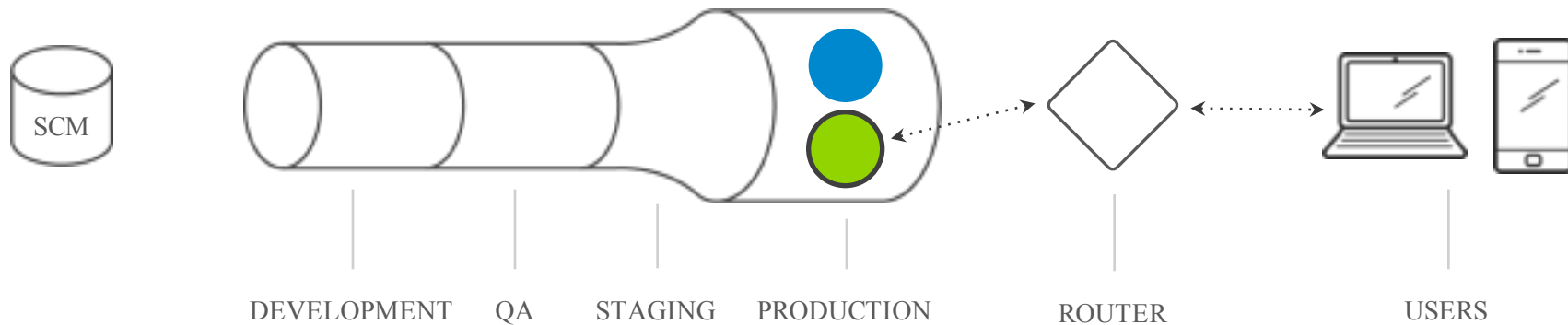
Blue/Green Deployment



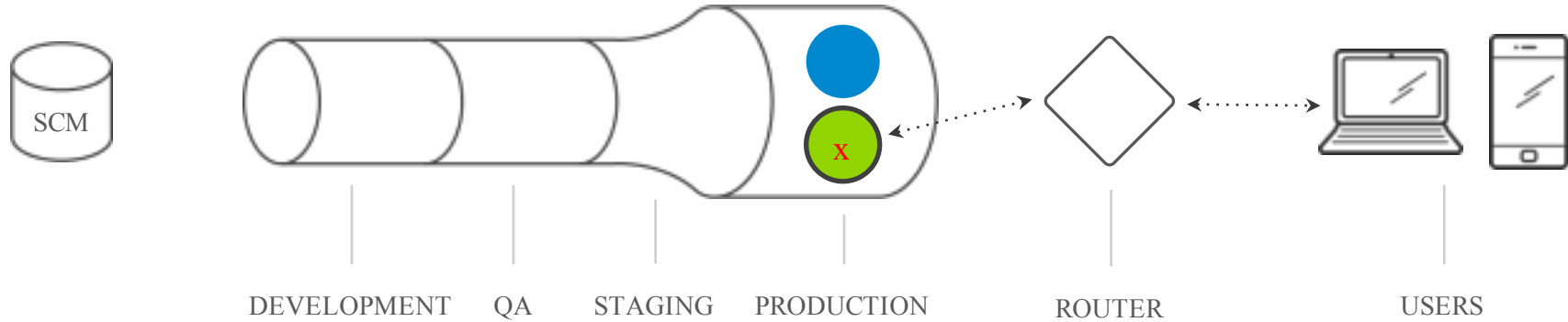
Blue/Green Deployment



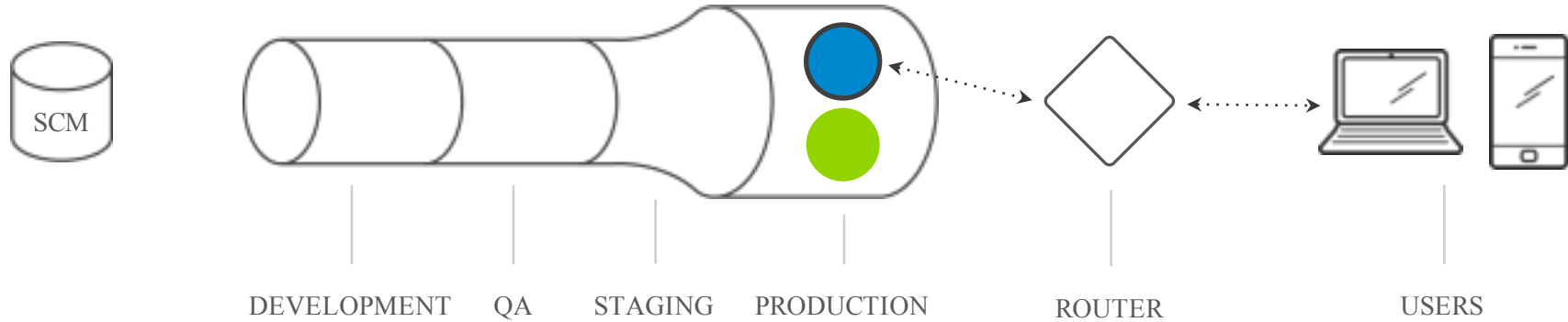
Blue/Green Deployment



Blue/Green Deployment



Blue/Green Deployment



Step 8: Blue/Green & Canary

bit.ly/msa-tutorial

Note: We plan to retire this tutorial

OpenShift Web Console | Microservices Frontend | Helloworld-MSA (Microservices Architecture)

frontend-helloworld-msa.app.burr.red

Red Hat - Hello world MSA (Microservices Architecture)

SSO | Browser as a client | **API Gateway** | Service chaining | Hystrix Dashboard | Jaeger Dashboard

Using an API Gateway

Refresh Results

API Gateway

- Blue - Aloha mai aloha-3-3kt96
- Blue - Hola de hola-2-xtd8w
- Blue - Clà de ola-3-1vwzr
- Blue - Bonjour2 de bonjour-3-4gv0f

Refresh Results

```
graph LR; Driver[Driver] <--> Internet((Internet)); Internet <--> APIGateway[API Gateway]; APIGateway --> HOLA[HOLA (JAX-RS)]; APIGateway --> Bonjour[Bonjour]; APIGateway --> Aloha[Aloha]; APIGateway --> Clà[Clà];
```

Step 8 Demo

https://github.com/burrsutter/9stepsawesome/blob/master/8_deployment_techniques.adoc

New Step 9: Databases

Step 9: Databases

1. [Persistent Volume](#)
2. [Persistent Volume Claim](#)
3. Deployment
4. Service

https://github.com/burrsutter/9stepsawesome/blob/master/9_databases.adoc

Old Step 9: Debugging

Step 9: Debugging

<https://maven.fabric8.io/#fabric8:debug>

```
mvn fabric8:deploy
```

```
mvn fabric8:debug
```

<https://code.visualstudio.com/blogs/2017/09/28/java-debug>

<https://github.com/VeerMuchandi/openshift-local/blob/master/DebuggingUsingIDE.md>

Step 9 Demo

https://github.com/burrsutter/9stepsawesome/blob/master/9_debugging.adoc

Bonus: Istio



Istio - Sail

(Kubernetes - Helmsman or ship's pilot)

learn.openshift.com/service-mesh

bit.ly/istio-intro

bit.ly/istio-tutorial

Next Generation - Service Mesh

Code Independent

- Intelligent Routing and Load-Balancing
 - Canary Releases
 - Dark Launches
- Distributed Tracing
- Circuit Breakers
- Fine grained Access Control
- Telemetry, metrics and Logs
- Fleet wide policy enforcement

What is Knative?



"Kubernetes-based platform to build, deploy, and manage modern **serverless** workloads."

"Essential **base primitives** for all"

"Knative provides a set of **middleware components** that are essential to build modern, source-centric, and **container-based applications** that can run anywhere: on premises, in the cloud, or even in a third-party data center"

Bonus: Eclipse Che

Container Native IDE

Allows you to launch a browser-based IDE inside of a Linux container that matches your production environment

<https://www.eclipse.org/che/docs/kubernetes-single-user.html>

<https://www.eclipse.org/che/docs/openshift-single-user.html>

Try it bit.ly/che-workshop

Create - OpenShift.io x OpenShift Eclipse Che | myvert: x Vert.x HTTP Booster x

Secure | https://che-burrrzinga-che.8a09.starter-us-east-2.openshiftapps.com/che/myvertxhello5-atanh

Workspace Project Edit Assistant Run Git Profile Help #1 00:56

Projects Explorer

- vertx
- myvertxhello5 (master)
 - src
 - licenses
 - main
 - fabric8
 - java
 - io.openshift.booster
 - HttpApplication.java
 - resources
 - test
 - target
 - jenkinsfile
 - LICENSE
 - README.adoc

```
29 // Retrieve the port from the configuration, default to 8080.
30 config().getInteger("http.port", 8080).ar -> {
31     if (ar.succeeded()) {
32         System.out.println("Server starter on port " + ar.result().actualPort());
33     }
34     future.handle(ar.mapEmpty());
35 });
36
37
38
39 private void greeting(RoutingContext rc) {
40     String name = rc.request().getParam("name");
41     if (name == null) {
42         name = "World";
43     }
44
45     JsonObject response = new JsonObject()
46         .put("content", String.format(template, name));
47
48
49 }
```

Debug

Breakpoints:

- HttpApplication.java:40

Frames:

- vert.x-eventloop-thread-0@3 in group "main": RUNNING
- greeting(RoutingContext):40, HttpApplication
- handle():-1, HttpApplication
- handleContext(RoutingContext):217, io.vertx.ext.web.impl.RouteImpl
- iterateNext():78, io.vertx.ext.web.impl.RoutingContextImplBase
- next():118, io.vertx.ext.web.impl.RoutingContextImpl

Variables:

- @httpApplication.java:40
- template="Hello, %s!"
- context=instance of io.vertx.core
- vertx=instance of io.vertx.core
- rc=instance of io.vertx.ext.web

OpenJDK 64-Bit Server VM 1.8.0_144

Events Processes Debug



STRIMZI

<http://strimzi.io/>

Apache Kafka on Kubernetes & OpenShift

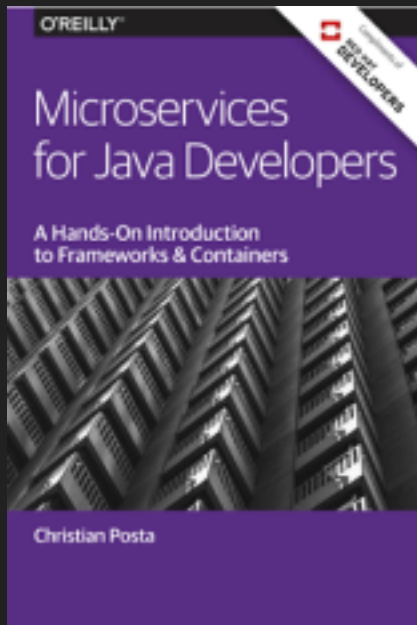
Postgres on Kubernetes



<https://info.crunchydata.com/blog/using-kubernetes-deployments-for-running-postgresql>

Free Resources

bit.ly/javamicroservicesbook



Free eBooks from developers.redhat.com

Microservices Introductory Materials

Demo: bit.ly/msa-tutorial

Slides: bit.ly/microservicesdeepdive

Video Training: bit.ly/microservicesvideo

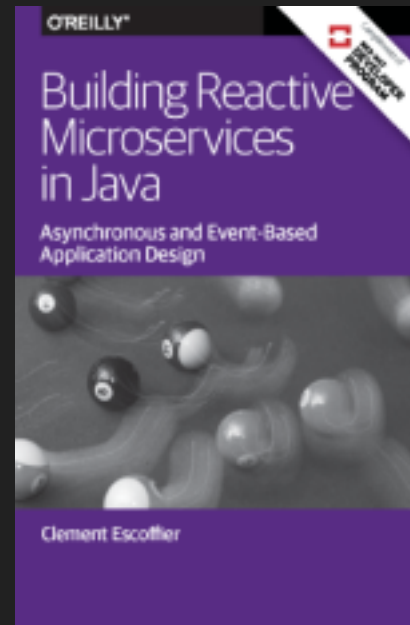
[Kubernetes for Java Developers](#)

Advanced Materials

bit.ly/istio-tutorial

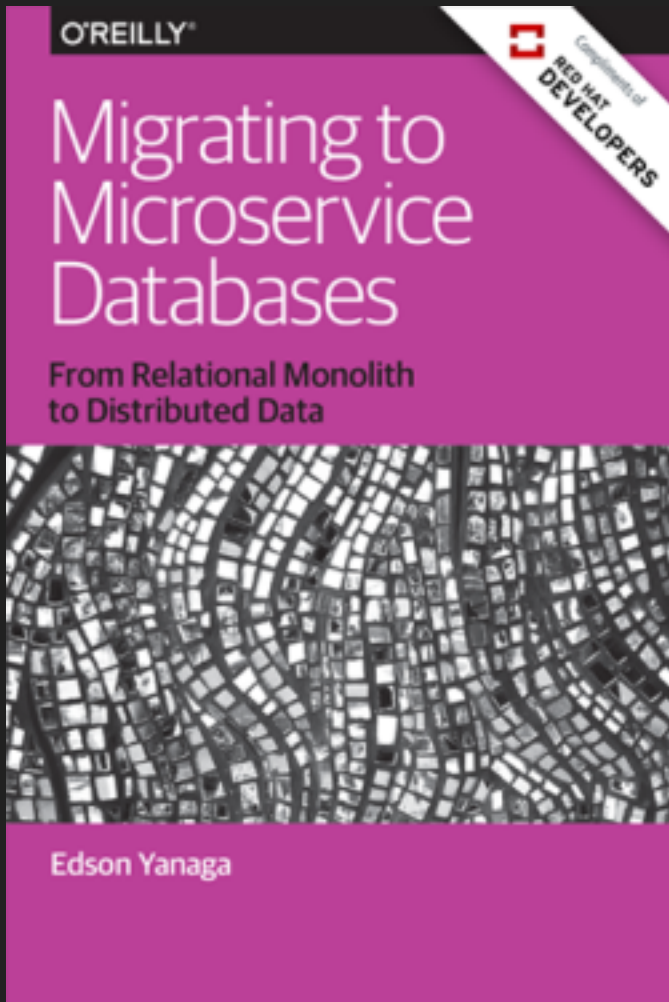
learn.openshift.com/service-mesh

bit.ly/reactivemicroservicesbook

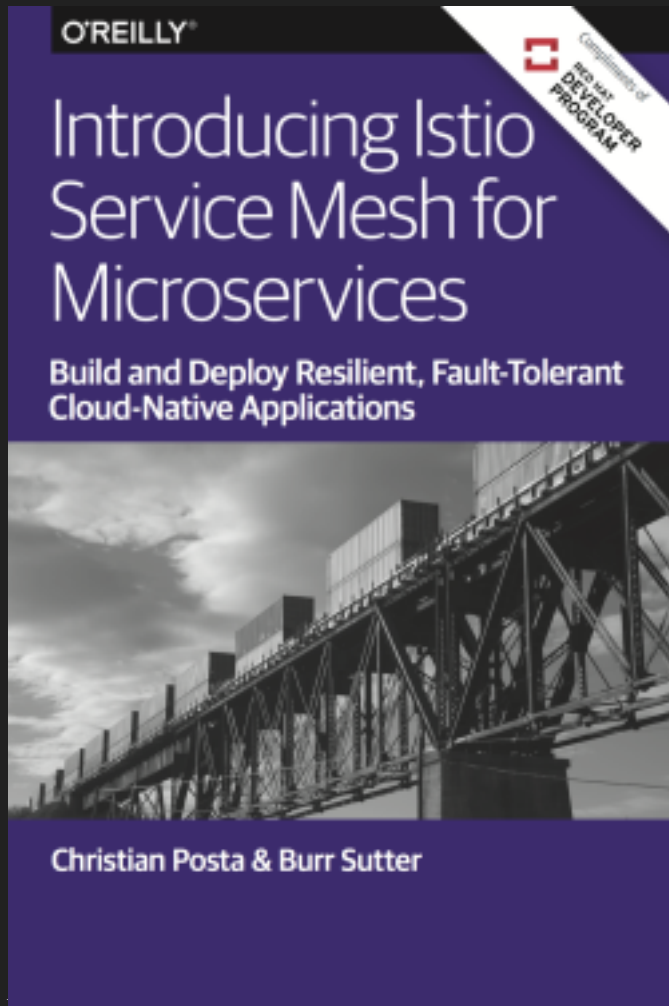


bit.ly/faas-tutorial

learn.openshift.com/serverless



bit.ly/mono2microdb

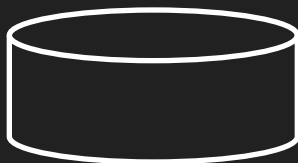
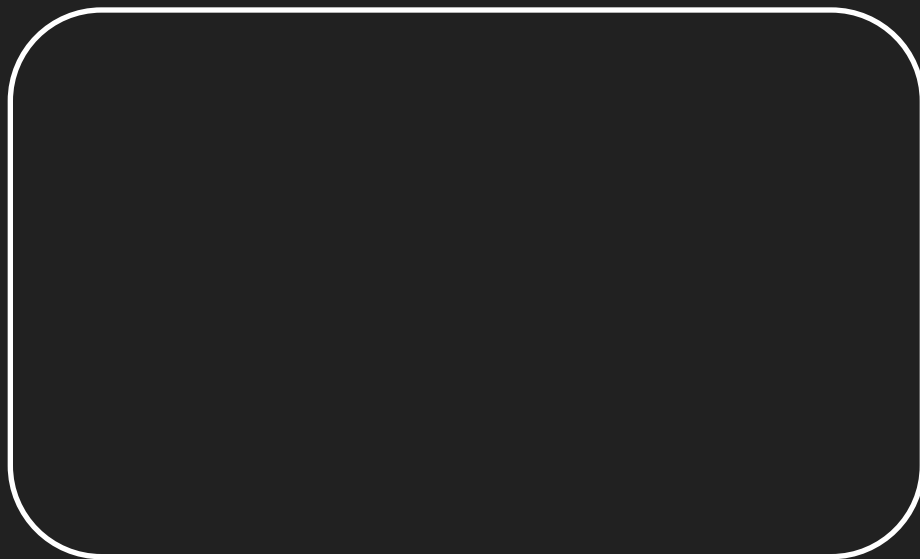


bit.ly/istio-book

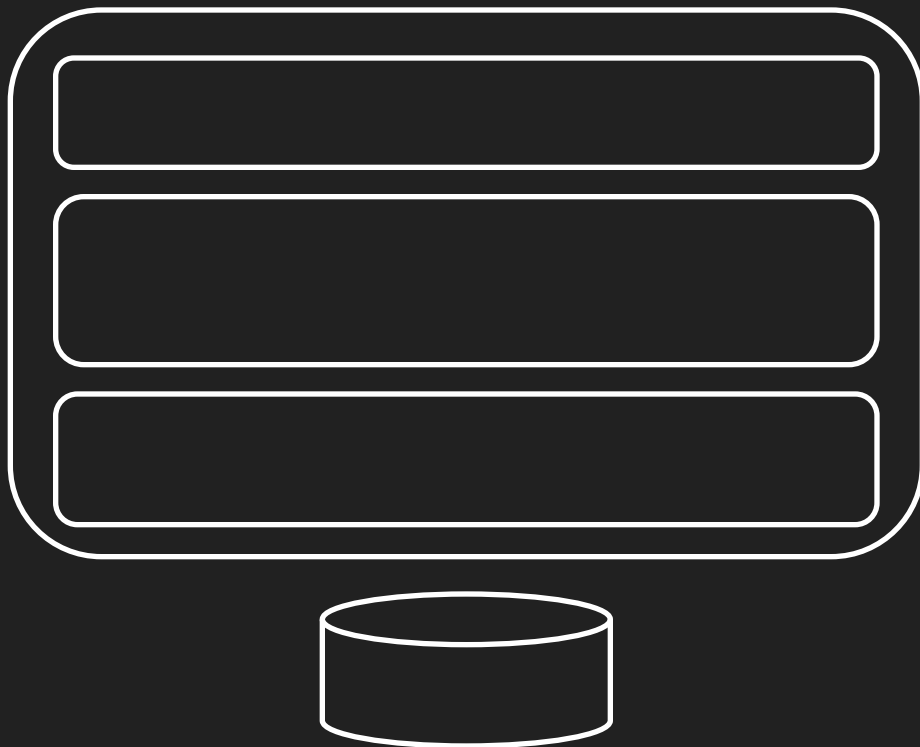
The End (Istio is Next)

Backup Content

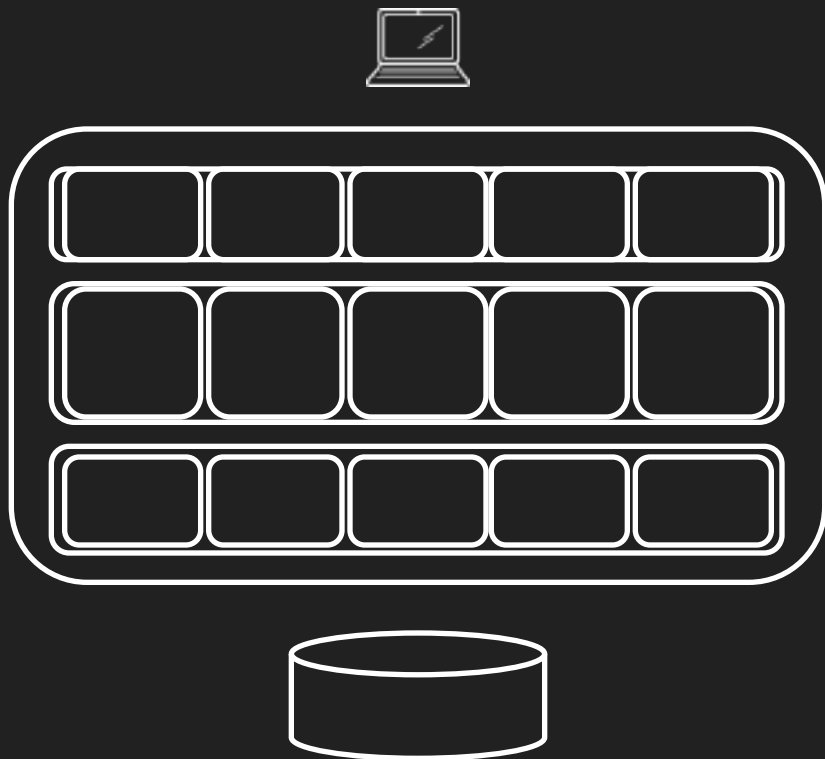
Application on Whiteboard



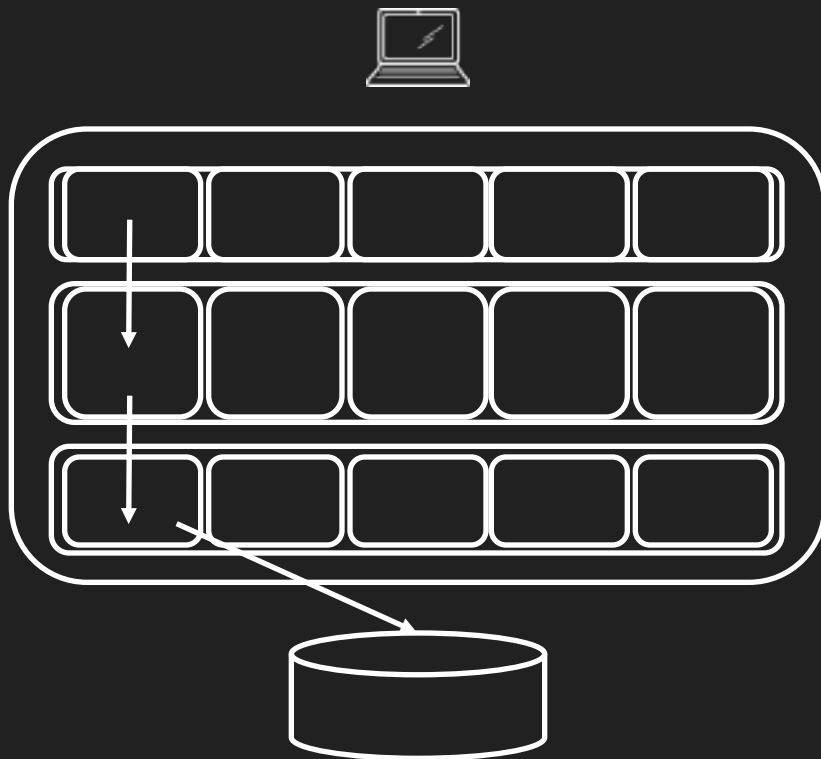
Whiteboard had 3 Tiers



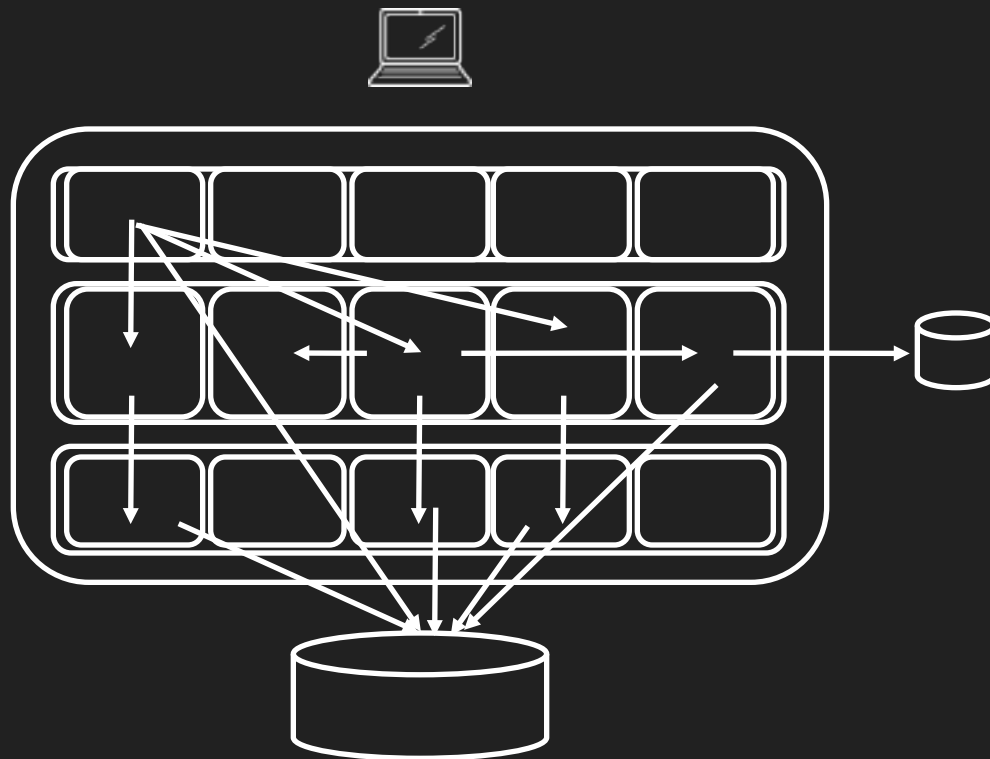
UI, Logic, Data



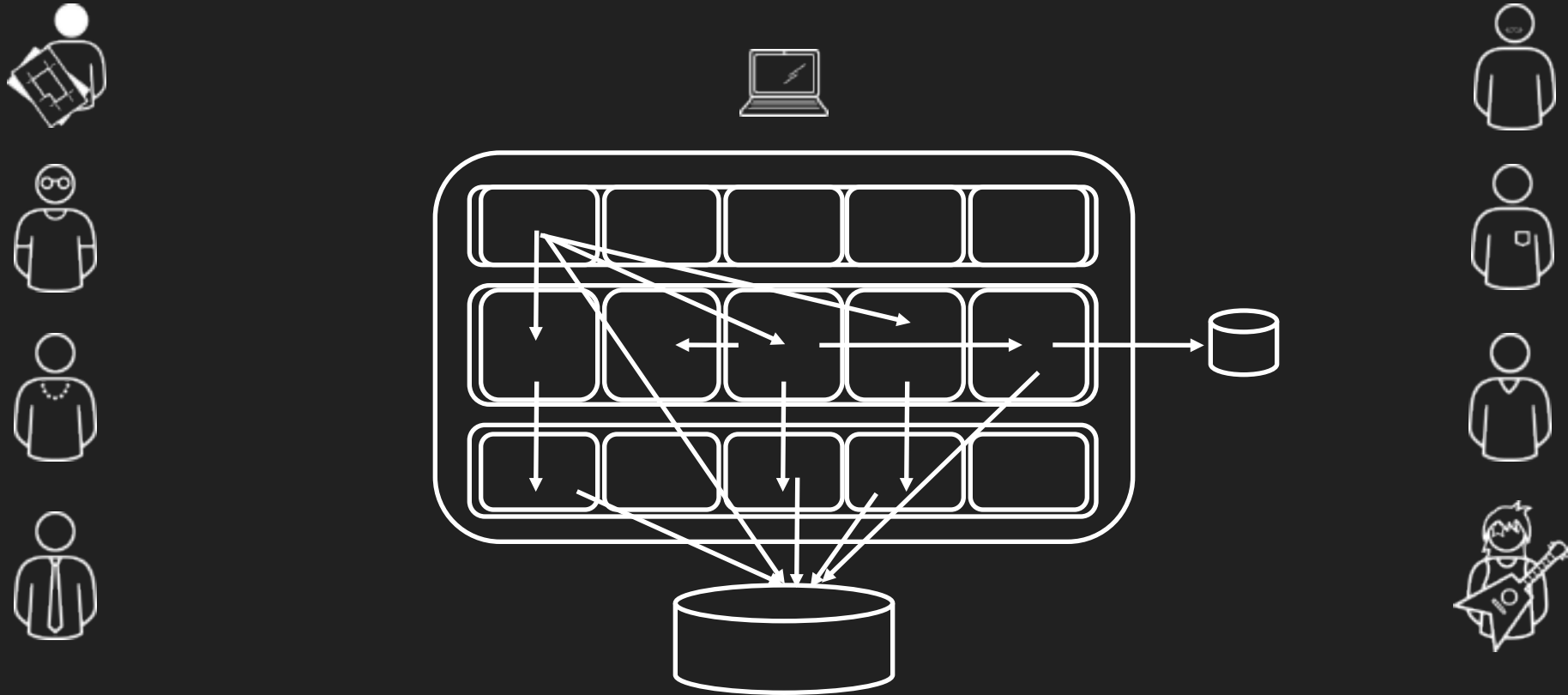
Clean Architecture



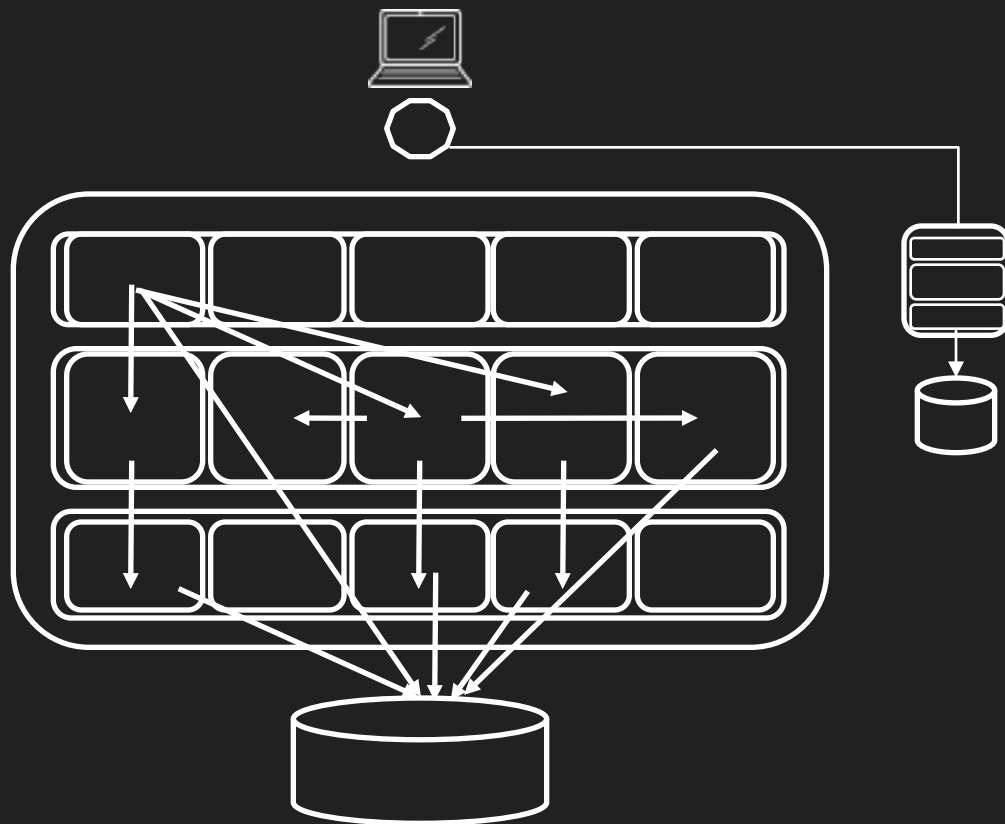
Real Life: Non-Majestic Monolith



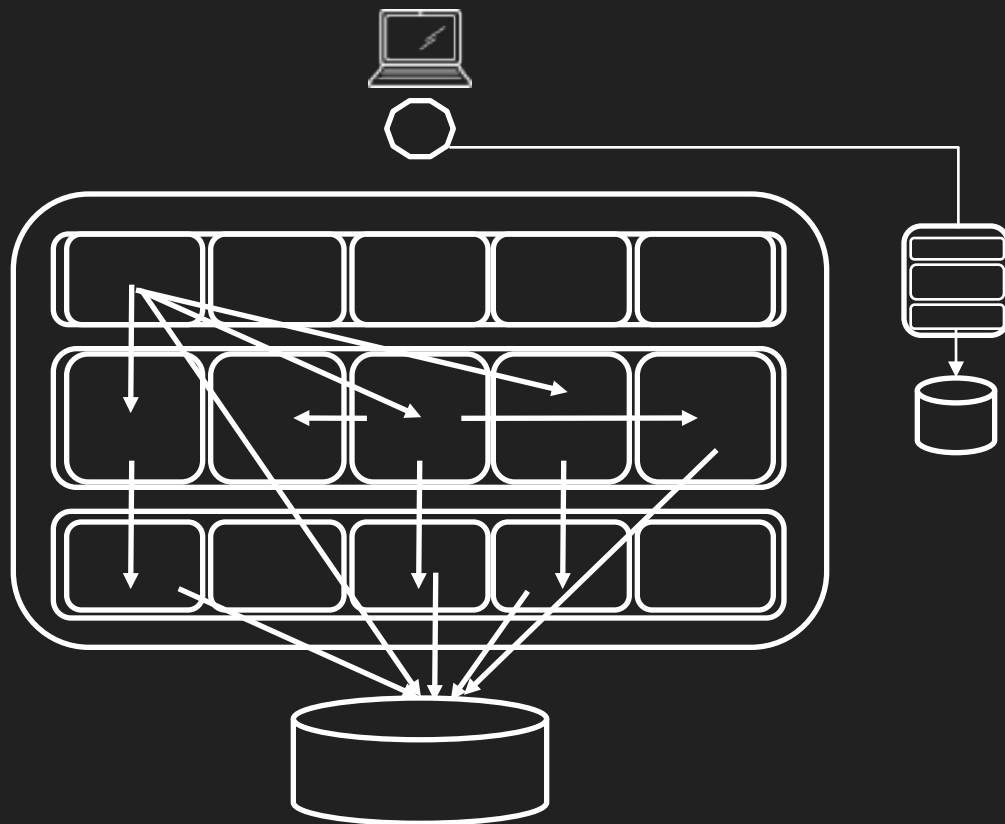
Large Team



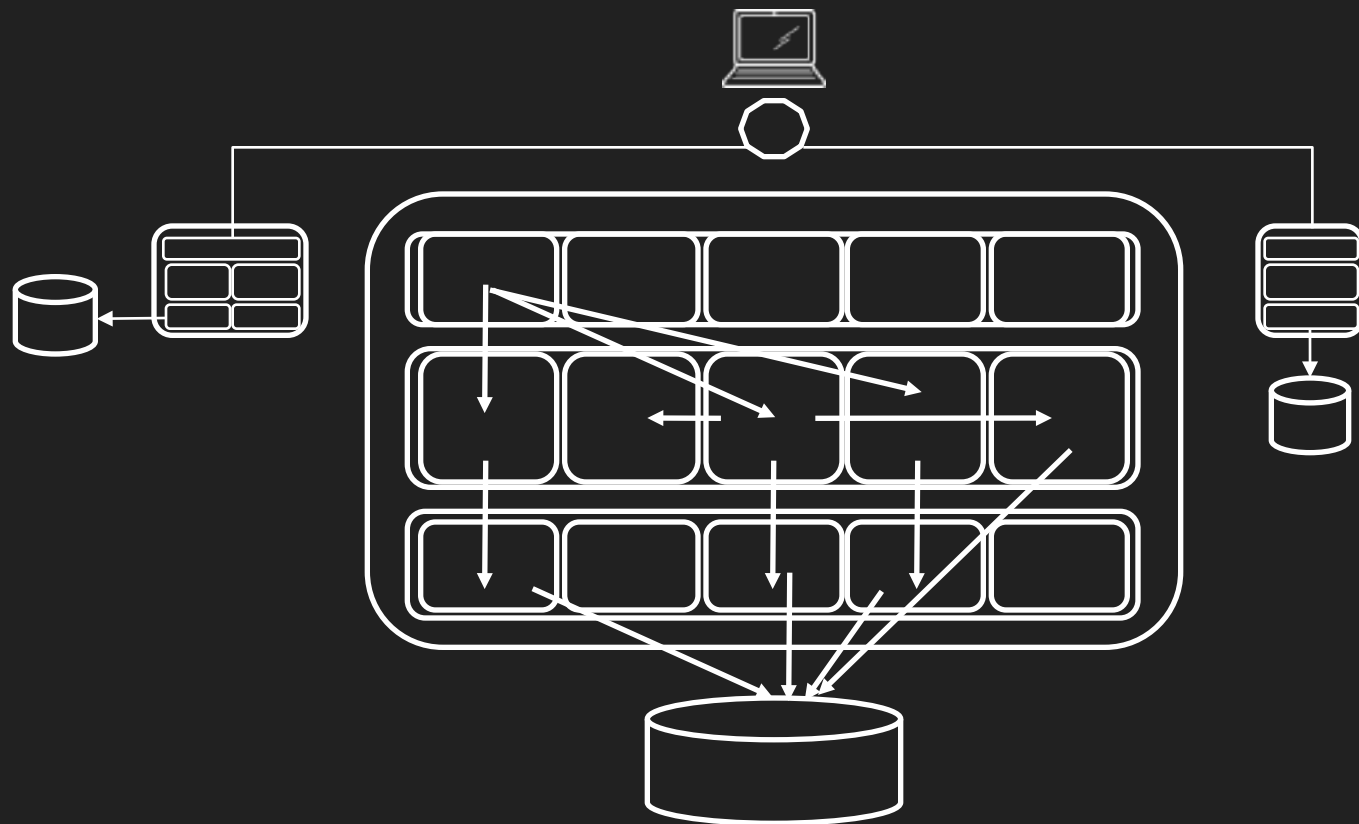
Strangle



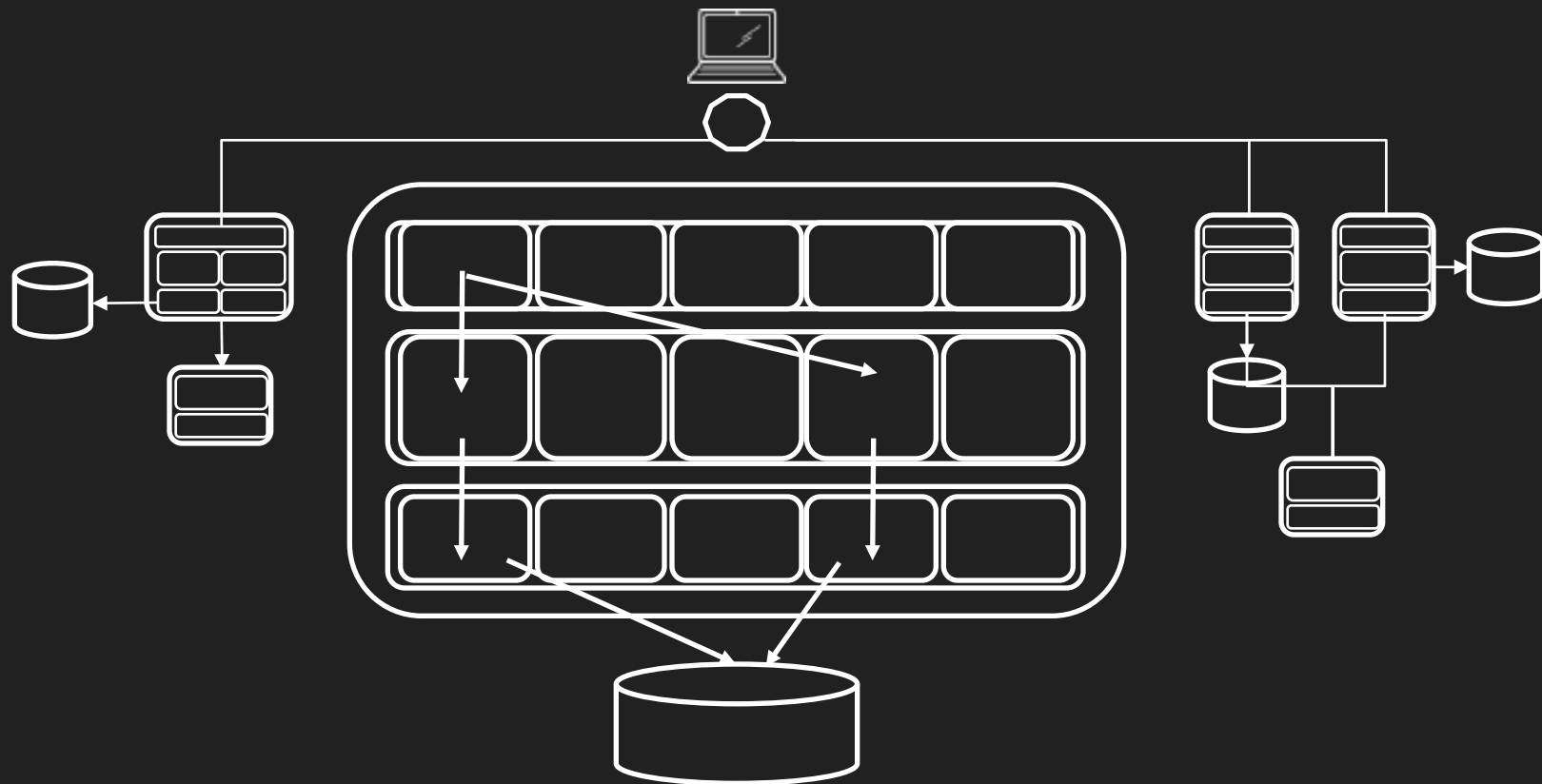
Strangle Hug



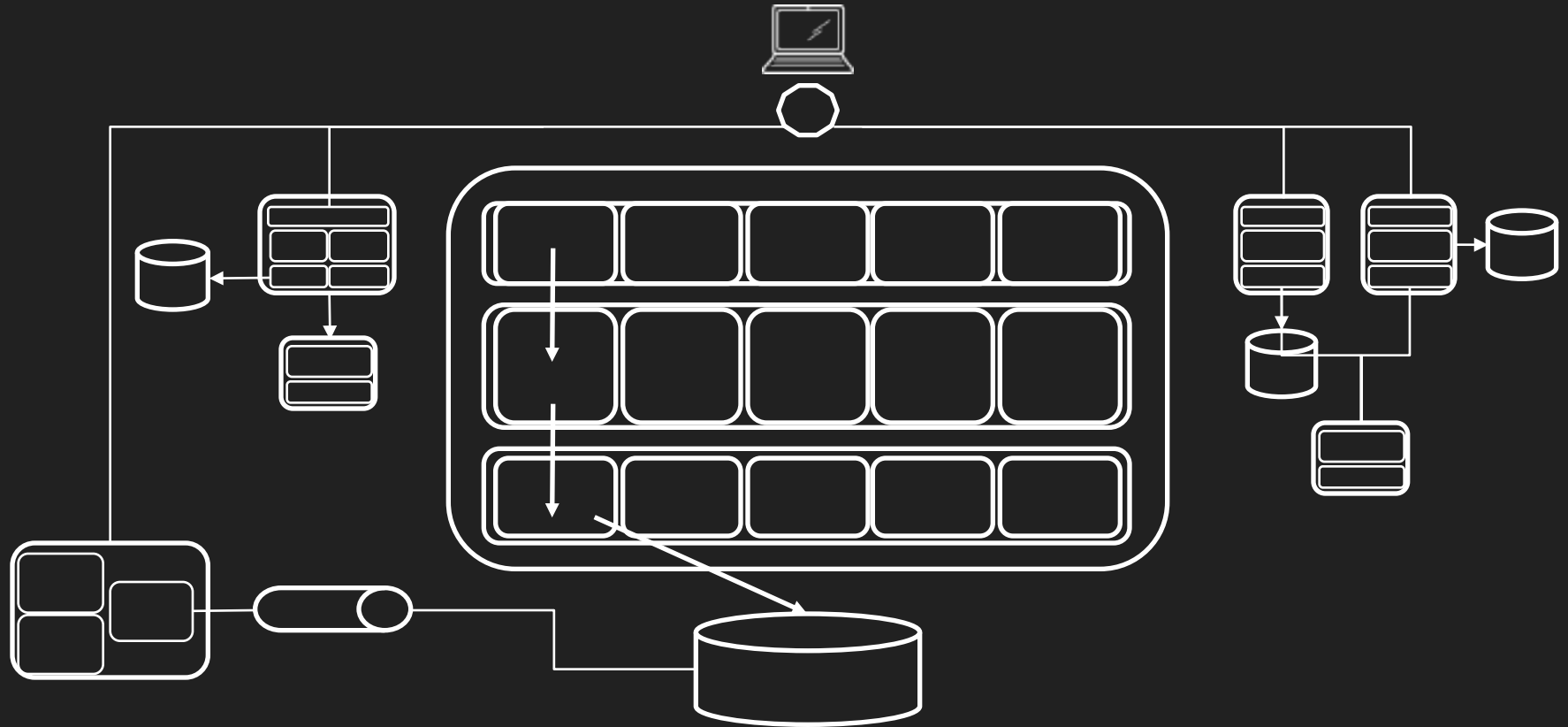
Friend Hug



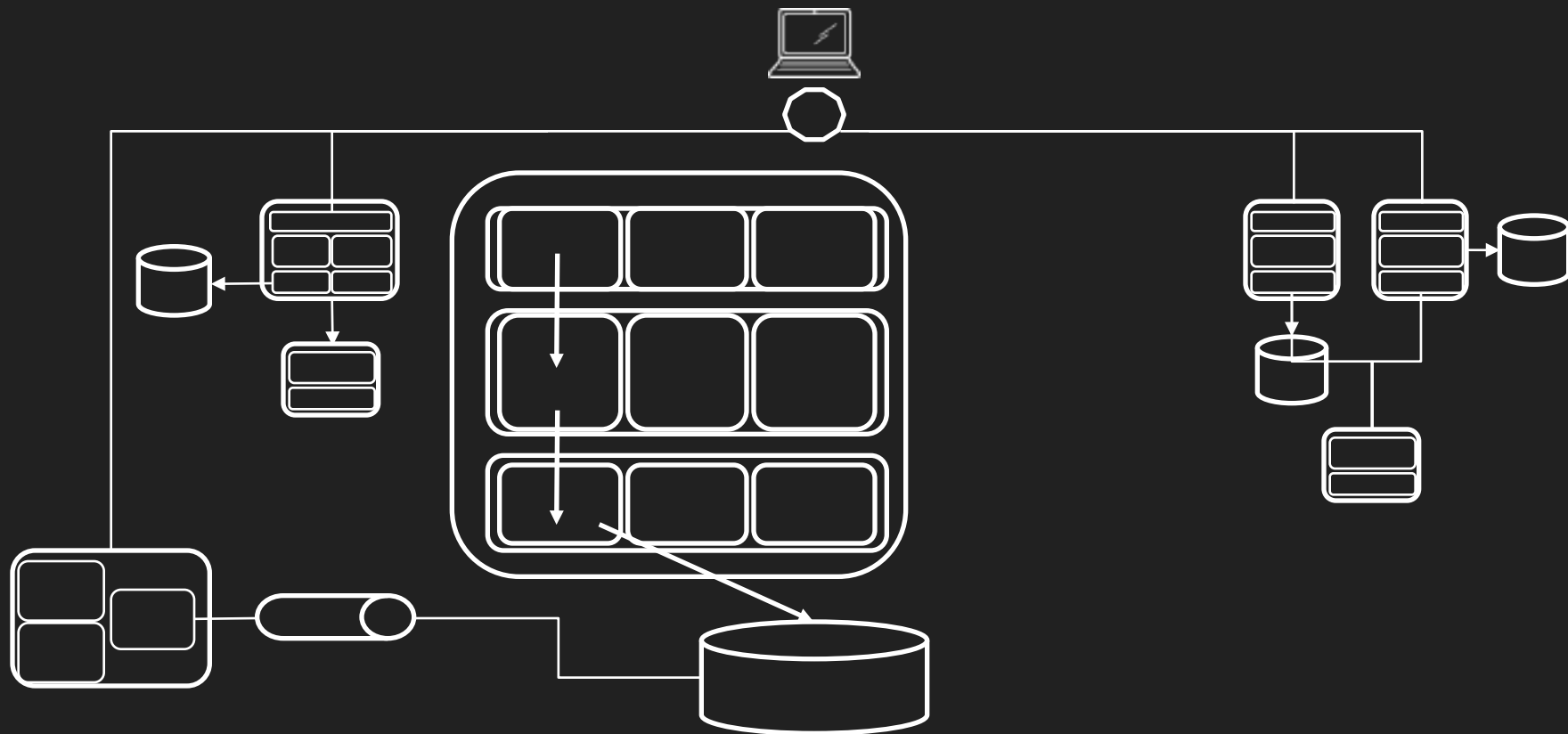
Family Hug



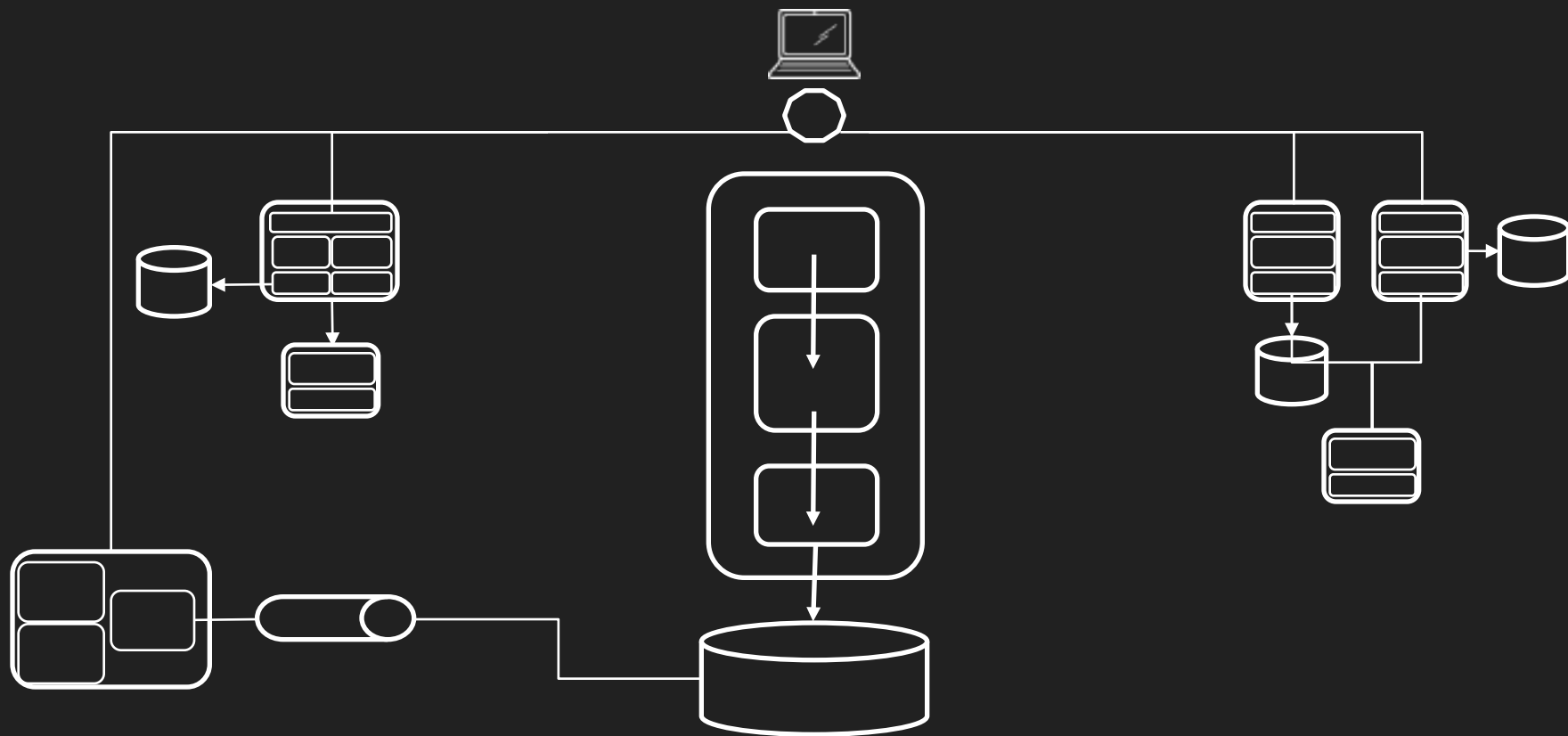
Bear Hug



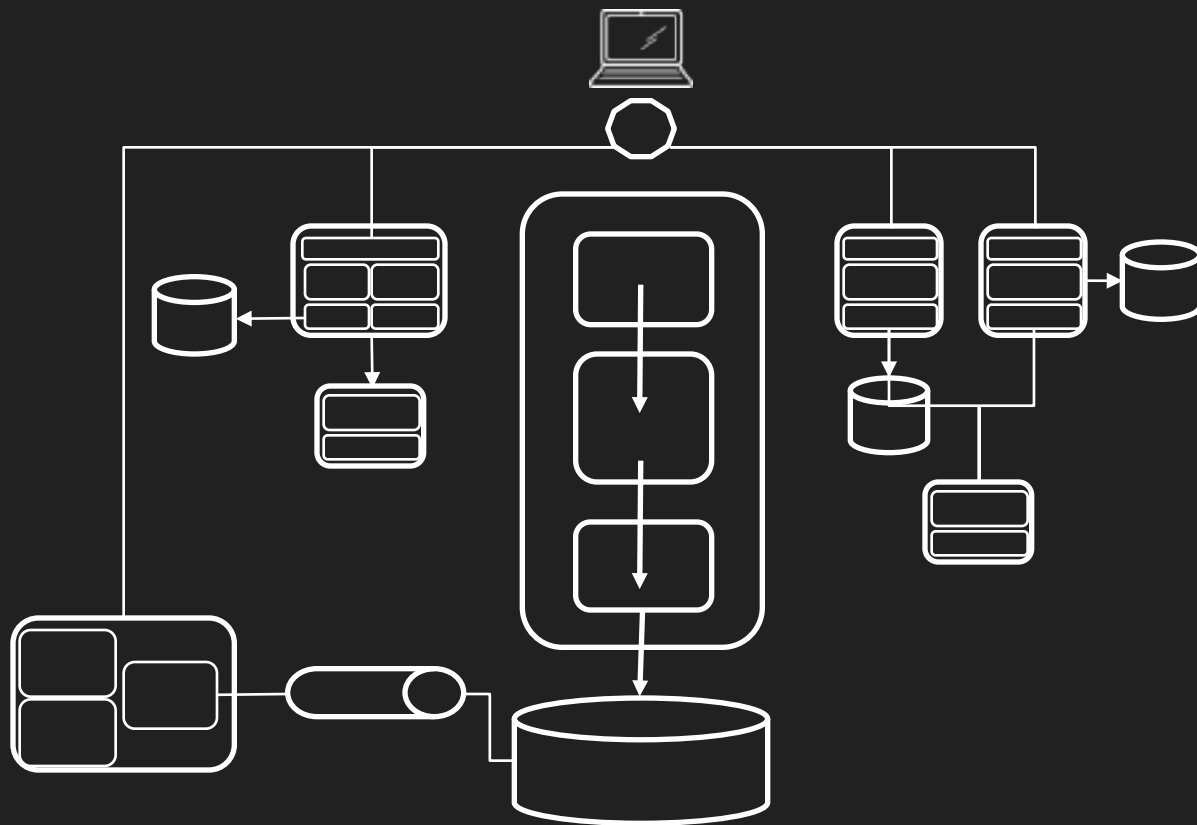
"Refactor"



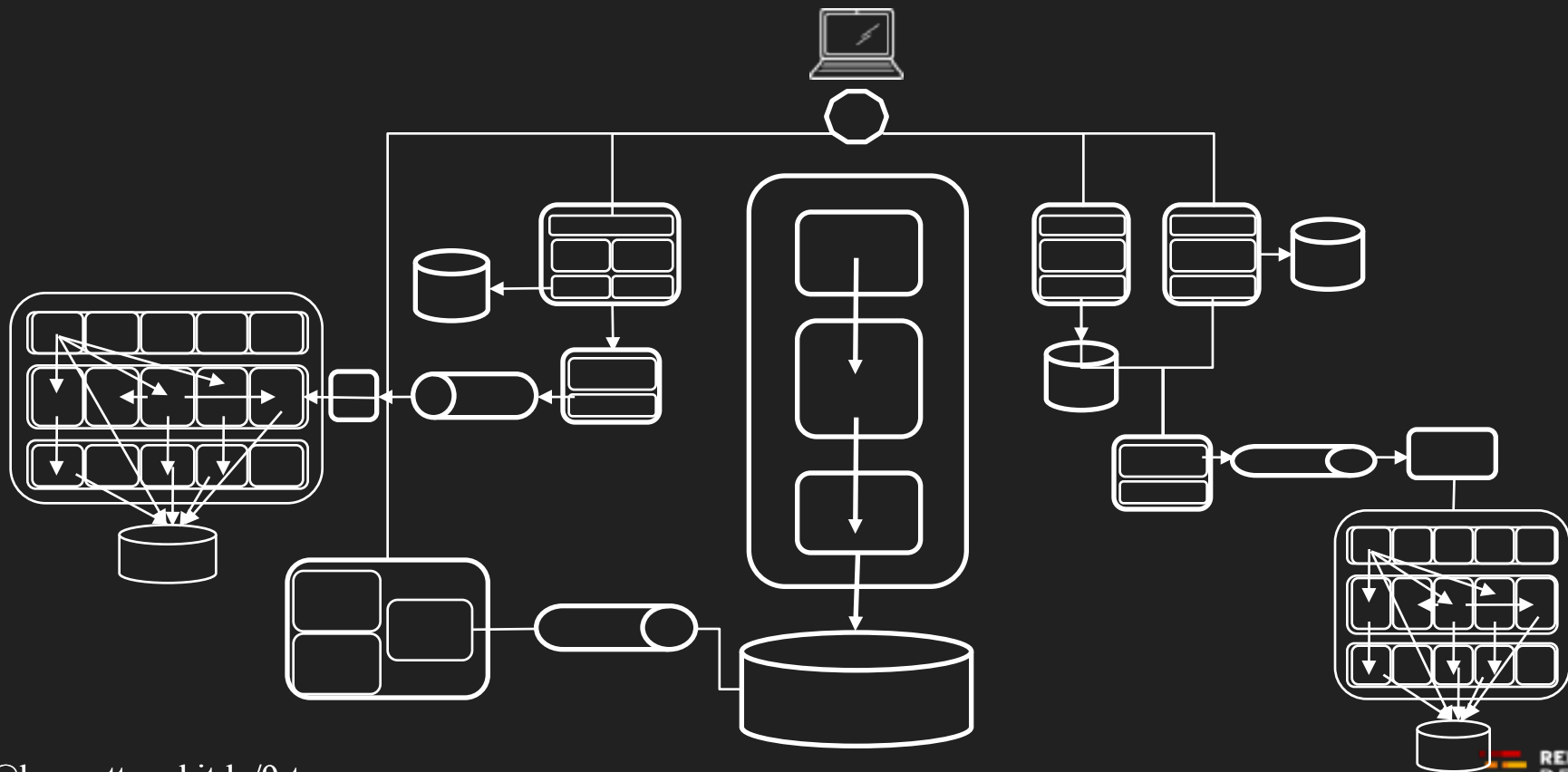
Squeeze



Streamline



Embrace Others



Raffle Rules (applicable in the real)

1. Follow: @burrsutter 
2. With picture of the session
3. Mention @burrsutter
4. With hashtag #devoxx