Kubernetes for Developers, Architects, & Other People Who Aren't Supposed to Use or Even Know About Kubernetes

An investigation into kubernetes konfusion

March, 2020

@cote

"Kubernetes is a platform for building platforms."

"Nobody will care about Kubernetes in five years."

"If you have a Kubernetes strategy you've already failed."

"Don't even try to run your own Kubernetes cluster."

"Kubernetes 'is now very, very boring.'"

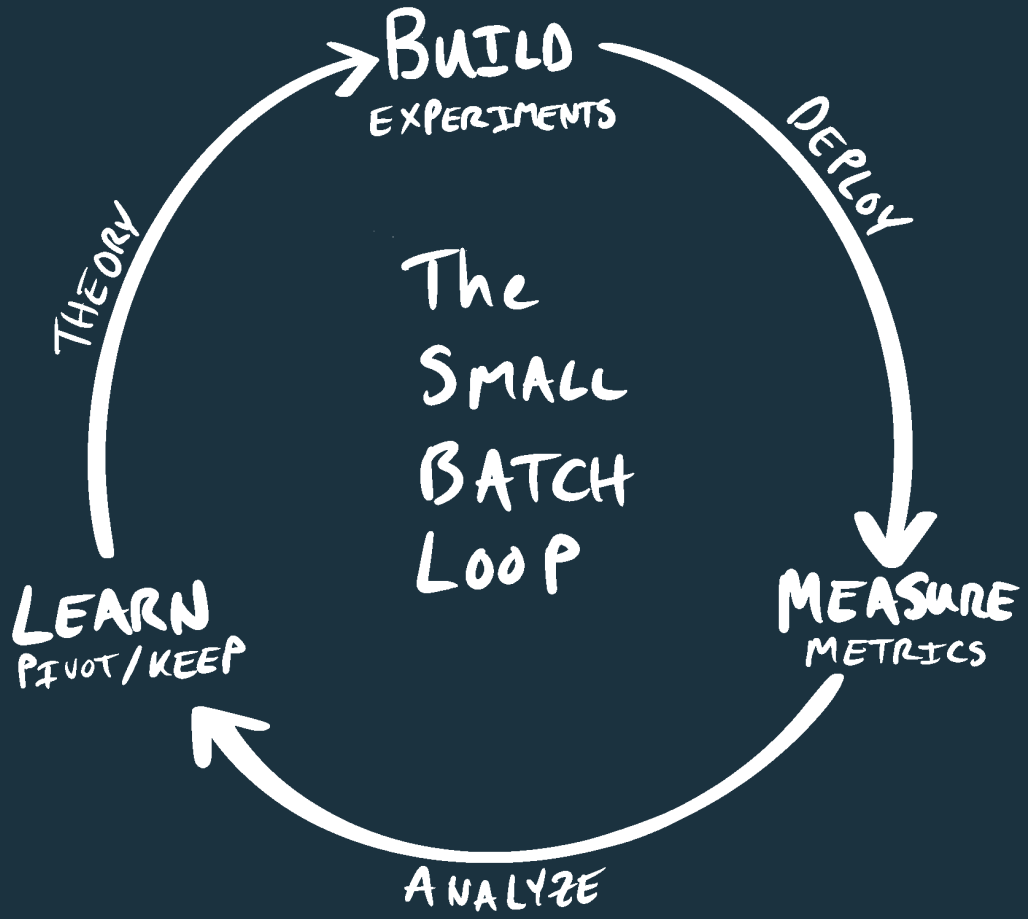
"Containers Will Not Fix Your Broken Culture"

Sources: @kelseyhightower, @quinnypig, @pczarkowski, @cowboyd, @janet_kuo, @bridgetkromhout.

So...as a developer, I should...?

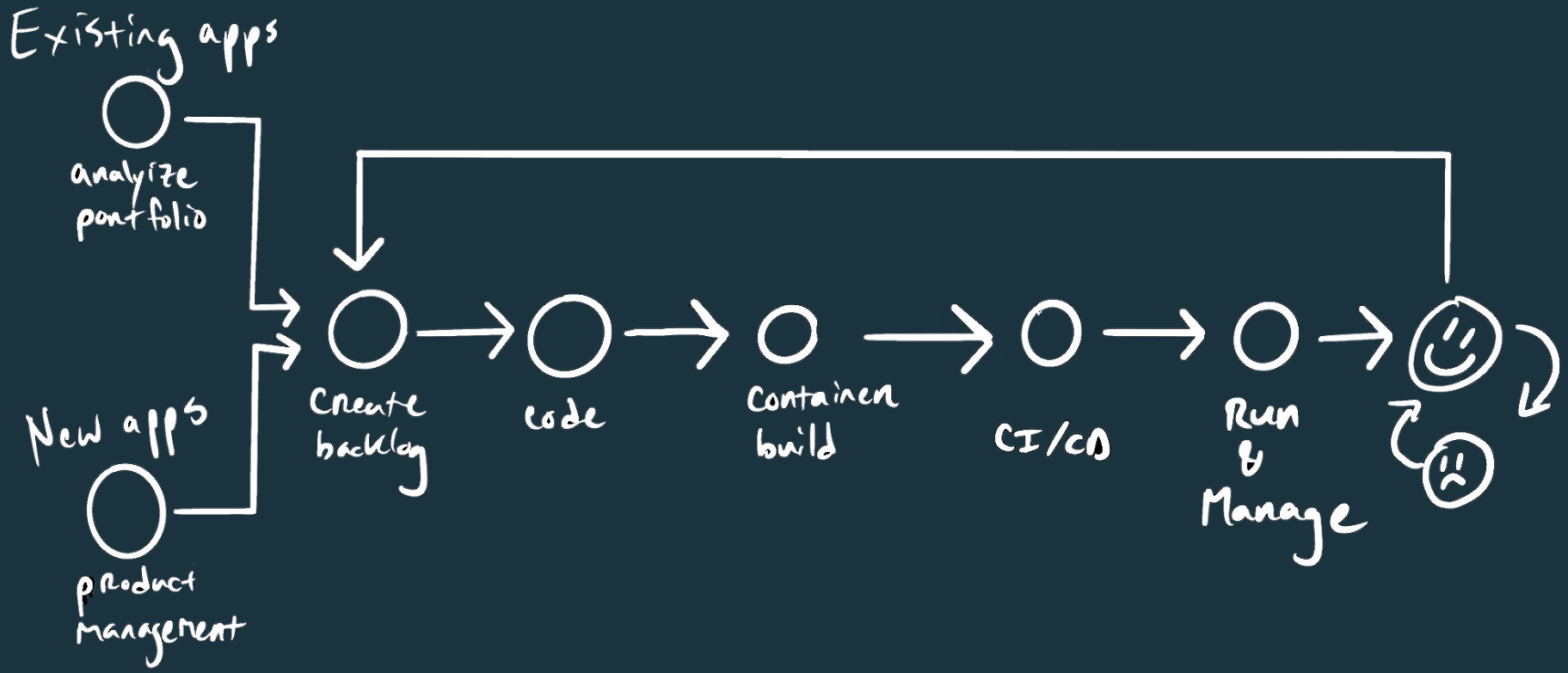
1.) Move pixels on the screen (product)

2.) Remove waste to focus on #1

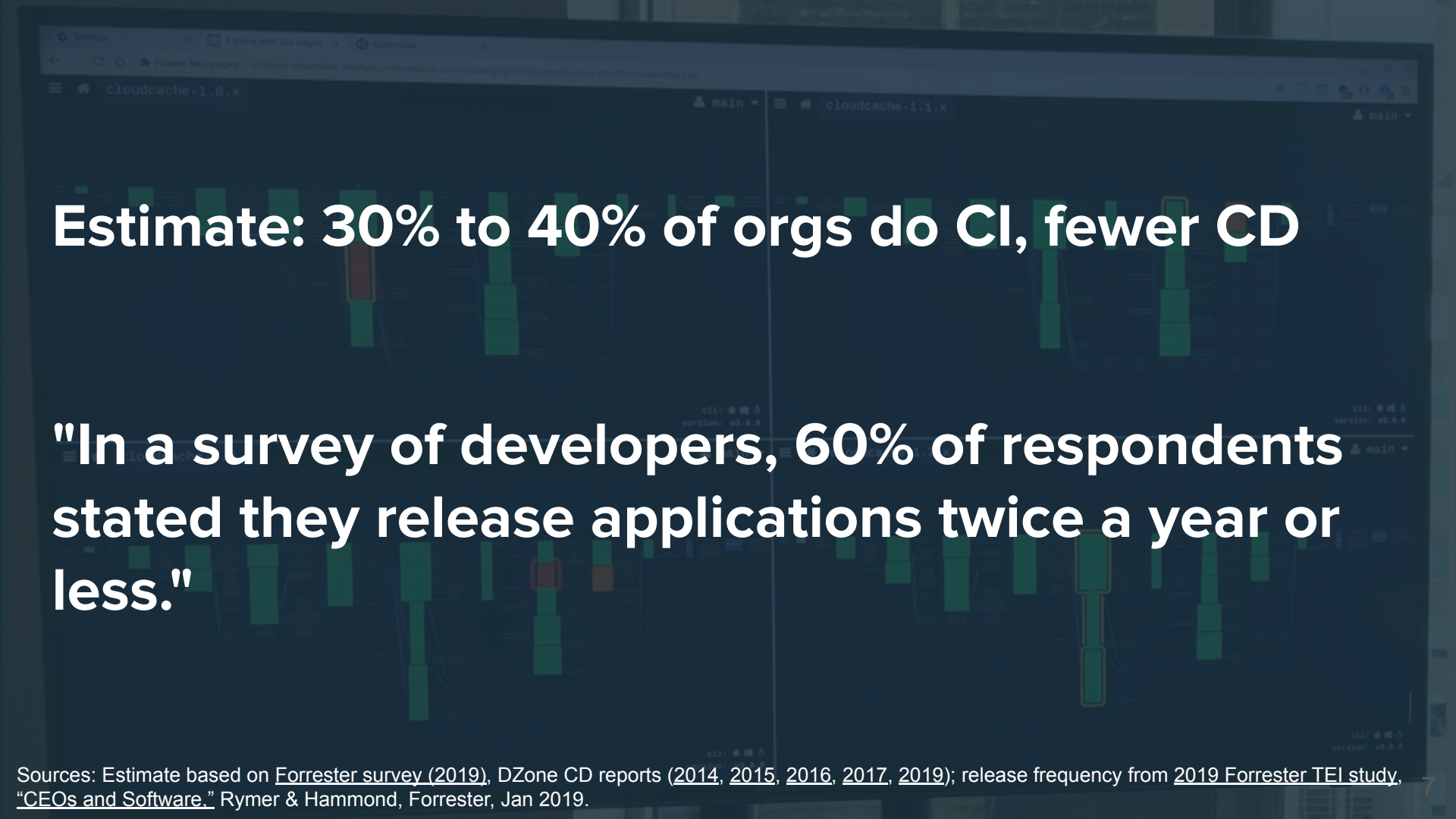


Sources: *Monolithic Transformation*, *The Business Bottleneck*.

Sources: Sophie Seiwald, Daimler, case study in *The Business Bottleneck*. Pics: pictavio & Nick Taylor.



Pic: based on James Watters VMworld 2019 keynote; concept to ka-ching, Edwards and Willis.



Estimate: 30% to 40% of orgs do CI, fewer CD

"In a survey of developers, 60% of respondents stated they release applications twice a year or less."

Sources: Estimate based on Forrester survey (2019), DZone CD reports (2014, 2015, 2016, 2017, 2019); release frequency from 2019 Forrester TEI study, "CEOs and Software." Rymer & Hammond, Forrester, Jan 2019.

In large organizations, governance is key to scaling waste removal.

Governance comes from standardization.

Theory: cloud native platforms can be platforms of good, enterprise governance.


Developers

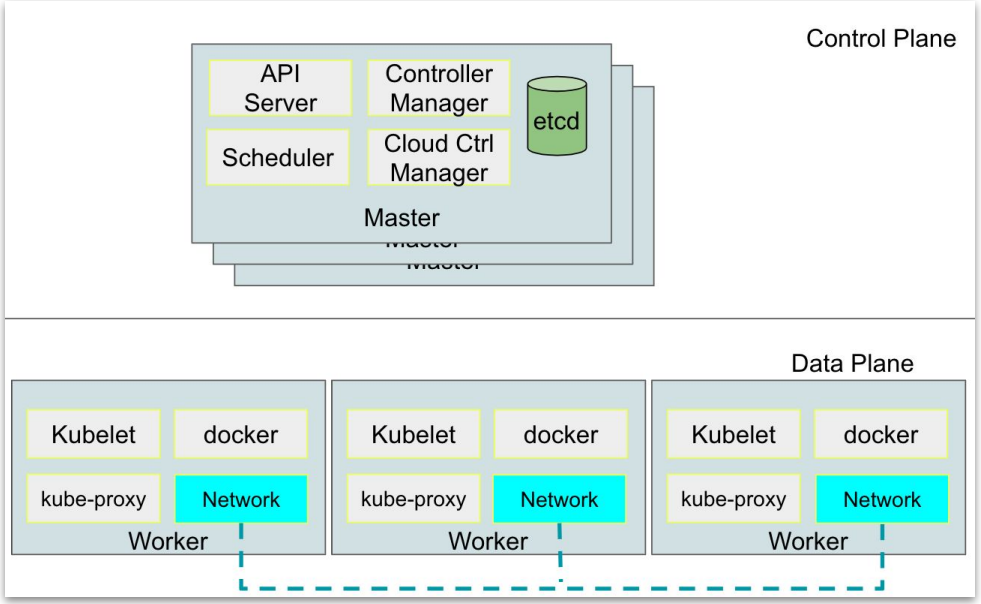
Kubernetes basics

Primitives

- yaml describes & configures
- Containers & pods, run in VMs, in clusters
- Managed by controllers & the scheduler
- System state stored in cluster store & ConfigMap
- "Resources" standardized, self-service

Behavior

- Enforce desired state
- Self-heal with autorestarting
- Scale with ReplicaSets
- Deployment plans, e.g., rolling
- Networking is still magical



You're building distributed apps.

Container-oriented development.

It's microservices!

Building containers

- You build it, you own it
- Endless fiddly stuff
- Standardize on a tool
- Standardize on images/registry
- Tools: spring-boot:buildimage, jib, buildpacks, etc.

More: "Effective Spring on Kubernetes," 2020; "Best Practices to Spring to Kubernetes Easier and Faster," Ray Tsang, 2019.

Pods are like servers...sort of

> [T]he right question to ask yourself when designing Pods is, **"Will these containers work correctly if they land on different machines?" If the answer is "no," a Pod is the correct grouping for the containers.** If the answer is "yes," multiple Pods is probably the correct solution. In the example at the beginning of this chapter, the two containers interact via a local filesystem. It would be impossible for them to operate correctly if the containers were scheduled on different machines."
>
> -*Kubernetes Up & Running*

- One or more containers that share a network and storage. E.g., front-end.
- The thing that's replicated
- E.g., databases pod alone

Packaging

Obviously: containers

Think in terms of "deployments" - pods, with containers, with resources

Declarative changes through yaml

Externalized configuration (ConfigMap or ENV)

Release strategies, e.g., rolling

Use Helm

```yaml
metadata:
  name: musicdeployment
  namespace: default
  labels:
    app: spring-music
spec: # this is the spec for the deployment (inc. things like replicas and how to rollout updates)
  replicas: 1 # bring up and maintain n replicas of the pod
  revisionHistoryLimit: 10 # keep only the last 10 entries in the rollout history
  minReadySeconds: 60 # wait until the first pod has been healthy for 60 seconds before rolling out the next one
      spec: # this is the Pod specification template
        containers: # specify the container types (inc. how they're monitored for health)
        - name: spring-music
          image: benwilcock/spring-music:1.0 # specify the Docker image (you can change versions here)
          imagePullPolicy: IfNotPresent # Would be 'Always' by default, but this can cause extra pulls
          ports: # specify the ports to expose
          - containerPort: 8080 # expose the app on port 8080 in the pod
            protocol: TCP # use the TCP protocol (default, alternative is UDP)
          livenessProbe: # checks if the container (in the pod) is healthy
            httpGet: # this probe will use the http GET method
              path: /actuator/health # call the actuator /health endpoint
              port: 8080 # use the container port 8080
            initialDelaySeconds: 20 # will not get called until 45 seconds after the pod has been created
            timeoutSeconds: 10 # must respond within 1 second
            periodSeconds: 60 # run the probe every 10 seconds
            failureThreshold: 3 # if more than 3 probes fail, restart the container
          readinessProbe: # check if the apploication (or service) is ready to receive traffic
            httpGet: # this probe will use the http GET method
              path: /actuator/health # call the / endpoint to see if we're ready
              port: 8080 # call the container port 8080
            initialDelaySeconds: 30 # check as soon as the pod is 'live'
            periodSeconds: 10 # check every 10 seconds
            failureThreshold: 3 # fail after 3 bad attempts
            successThreshold: 1 # go to 'ready' after 1 successful check
            name: mysqlsecret
            key: database.username
        - name: SPRING_DATASOURCE_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysqlsecret
              key: database.password
        - name: SPRING_DATASOURCE_URL
          valueFrom:
            secretKeyRef:
              name: mysqlsecret
              key: spring.datasource.url
```

Pic: @benbravo73's Spring Music Store in Kubernetes.

Probes & Life-cycle

- livenessProbe
- readinessProbe
- startupProbe

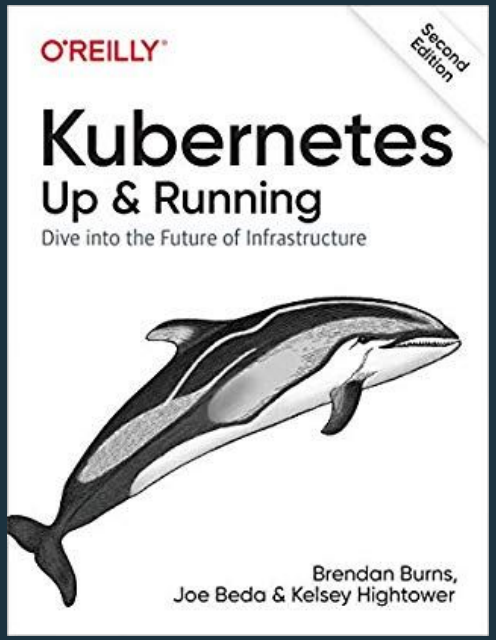
- PostStart
- PreStop
- Log to stdout/err

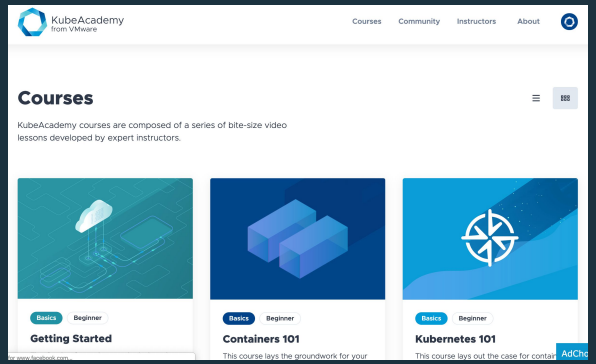
A whole lot more...

Internalized Configuration
Storage
Network Routes
Load Balancers
Secrets
Incident Management, Logs, Observability
Sidecars, Handlers, etc.

Hopefully, a friendly ops person handles these.

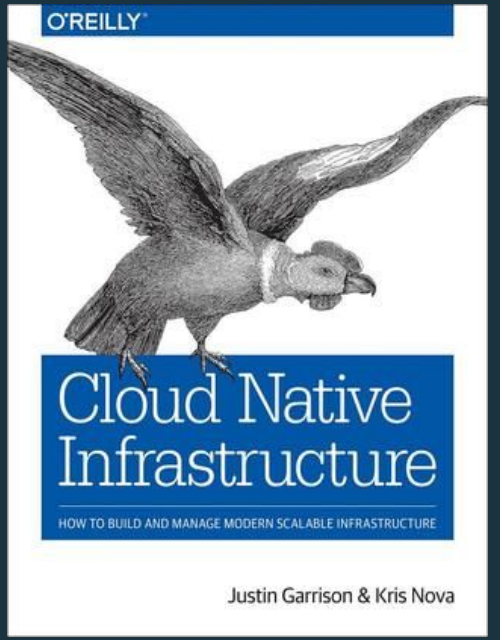
For much better explanations...

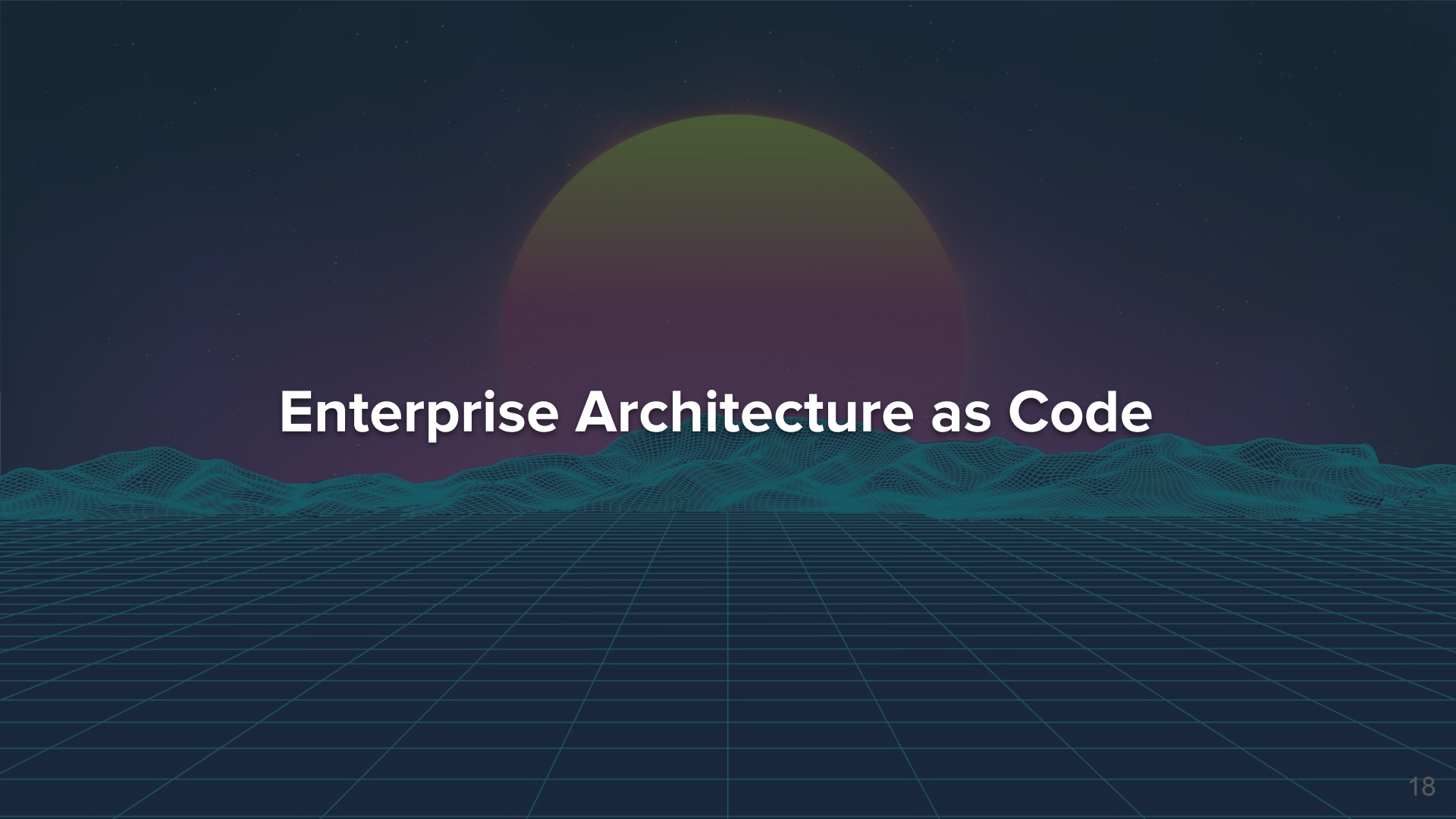




http://Kube.Academy

Spring Cloud Kubernetes




Enterprise Architecture as Code

Use a platform to *implement* enterprise governance

App Logic

Frameworks

Monitoring

Access Ctrl

Runtime

Container

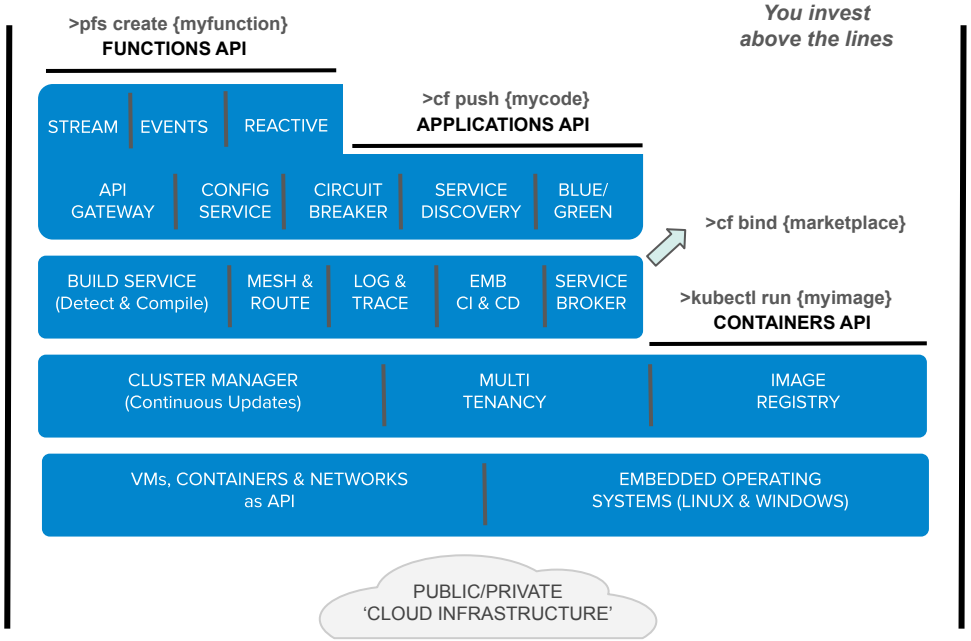
Os

Vm

Network

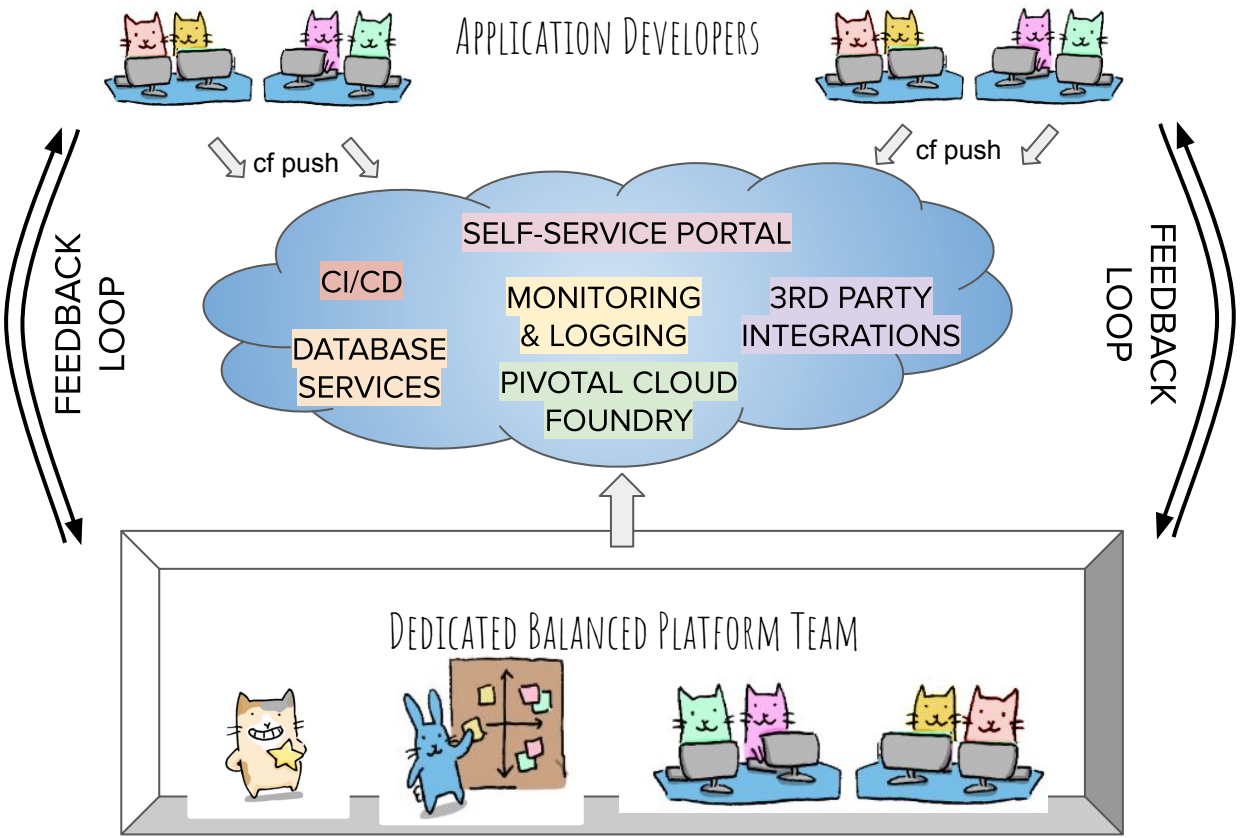
Hw

App Stack



Enterprise Architecture as Code

Enterprise governance through platform as a product

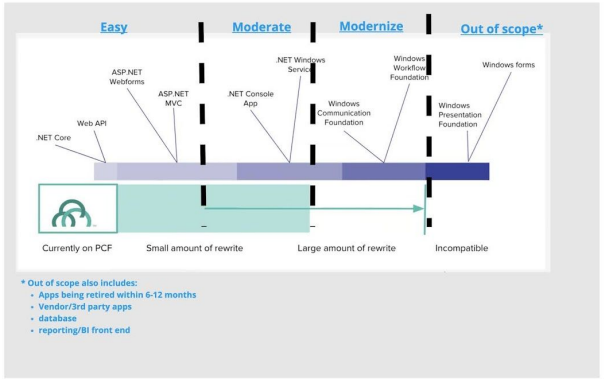


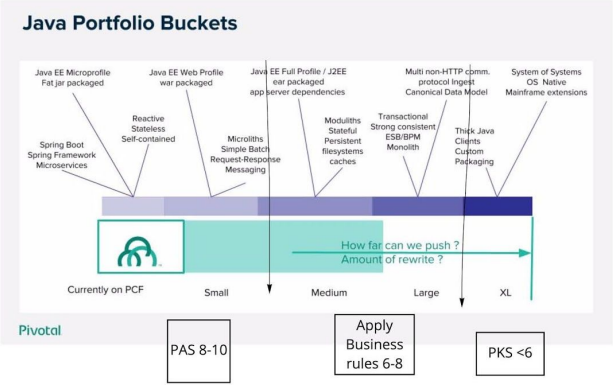
More: "Platform as a Product" presentation, Sep 2019. Pic: @deniseyu21.

The Case for Modernization

- Unique to enterprises
- Years of M&A, tech debt, old business models
- Silent killer of business innovation
- "You can put any garbage you want in a ~~container~~kubernetes"

Pic: @AreasAbandoned; garbage quote adapted from @pczarkowski.



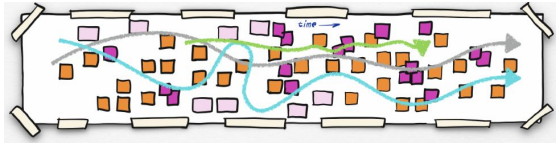


Ongoing portfolio management

- With 1,000's of apps, it will take years
- Evaluate for technical fit
- Evaluate for business value
- Establish criteria for success
- Retro projects, adapt strategy ongoing
- Start small: store finders, single API methods
- Maintain a cookbook
- Do no harm

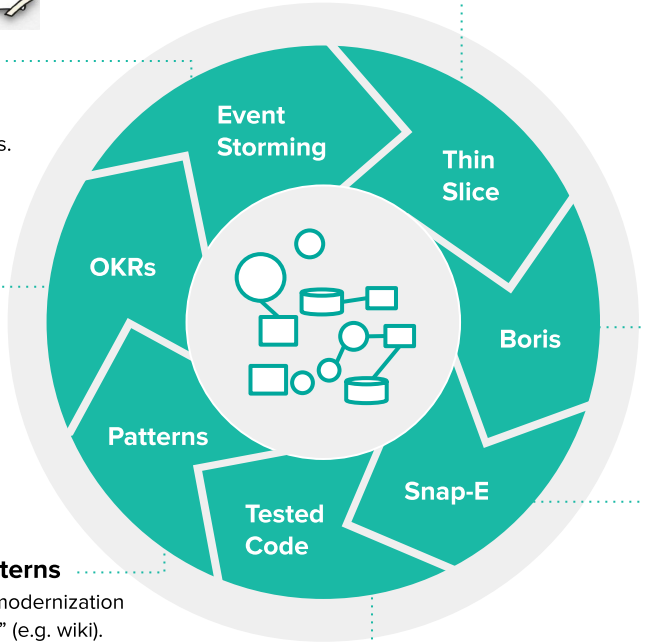
Sources: Rohit Kelapure, Jan 2019; "How AirFrance KLM is Modernizing 2,000 Legacy Applications," Dec 2019; "How T-Mobile Built and Scaled Kubernetes On-Prem for the 2018 Retail Season," Dec 2018.

Deconstructing Monoliths, AppMod example



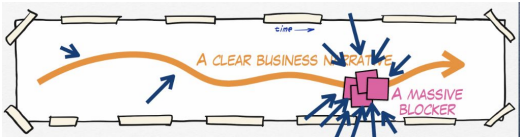
2. Event Storm the system

Highly collaborative modeling tornado!
Make sense of a complex business domain with a common language for different tribes.
Identify trouble spots & starting points.



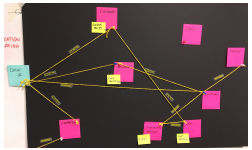
3. Select "Thin Slice(s)" of functionalities

Cut the monolithic business cake and pick a sweet piece to start the modernization work.



4. Drafts the desired notional architecture

Model the process & data flows connecting the business capabilities in sequence.



1. Define Objectives & Key Results

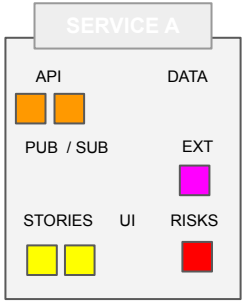
Decide on direction for outcomes & approach.
Agree on objectives and refine key results to measure those.

We will **objective**

as measured by **key results**

5. Fill the backlog

Craft actionable user stories.
Identify APIs, data & connections.



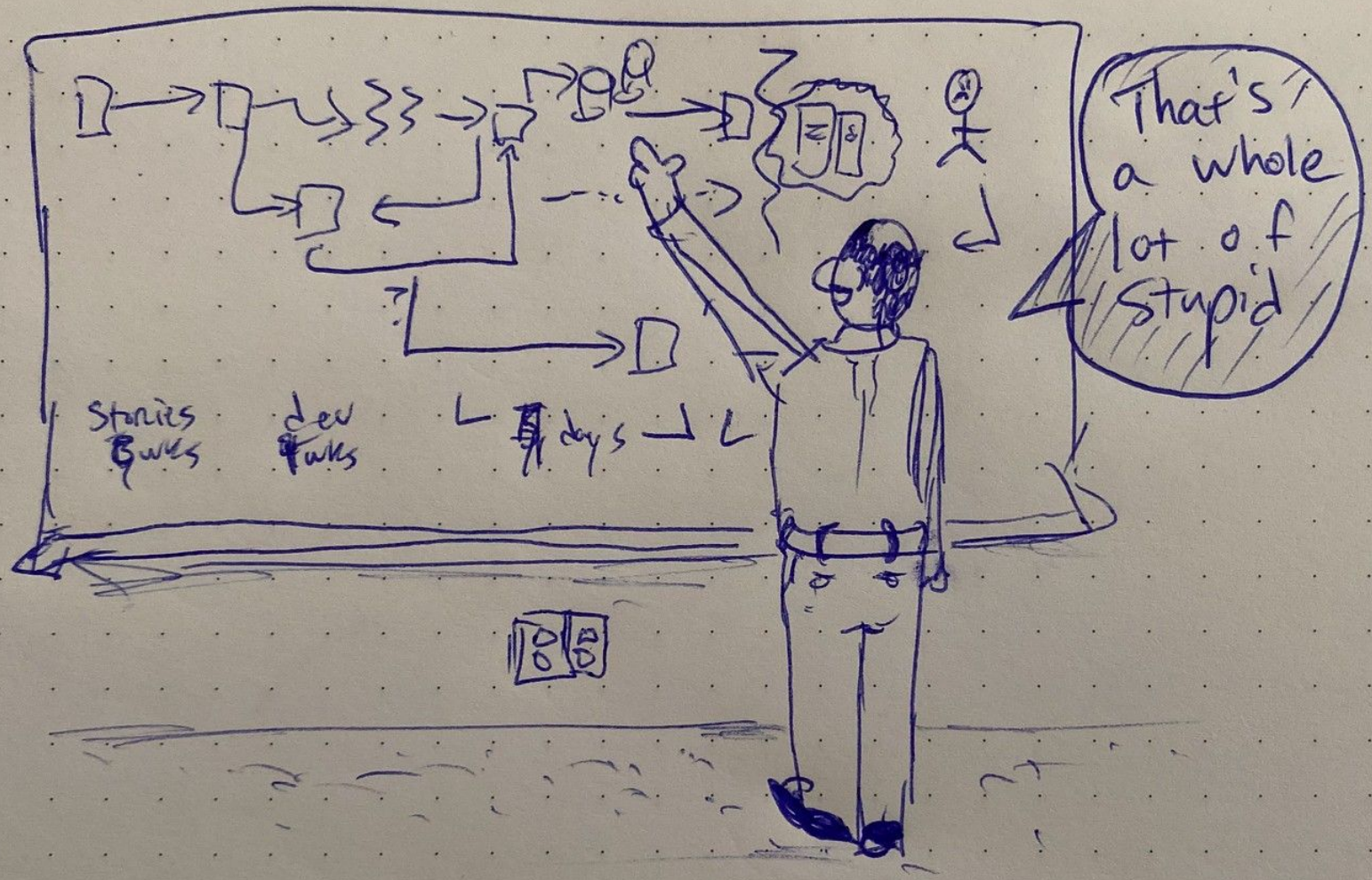
7. Cloud Modernization Patterns

Reuse and document the learned modernization patterns as recipes in a "cookbook" (e.g. wiki).
Consolidate knowledge and accelerate team velocity.

6. Produce tested and working code

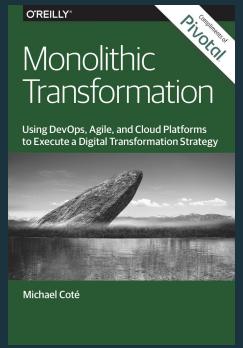
Take low-level design decisions, select technical patterns to use, and implement user stories. Time-bounded set of 1-week iterations.

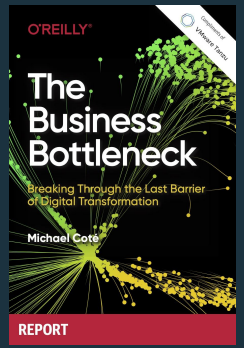
Technical improvements	Business improvements
Daily deploys	11% conversion rate
+30% developer productivity	40% reduced call center volume
+78% operational efficiency	6 months to launch a new business
No weekend work	Software-driven businesses

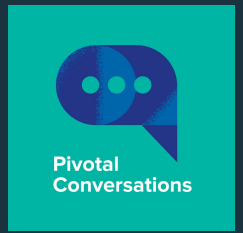

That's a whole lot of stupid
Stories
Bugs
dev
Tasks
days

Coté*

@cote | http://cote.io







* Full name: *Michael* Coté, but no one calls him "Michael" except his wife and mother.
Pic: @pczarkowski.

Externalized Configuration

Specification:

- ConfigMap
- Map the map in deployment
- Cheat with the command line

Access:

- ENV
- Files in container
- Framework, e.g., Spring Cloud Kubernetes

Let's regroup, for developers:

- It's all a cluster of VMs running containers
- ...that work together over a network
- Distributed apps/microservices
- Containerize app components
- Describe app architecture, dependencies & runtime lifecycle in yaml
- Implement probes and lifecycle hooks
- Externalized configuration
- Talk with your ops friends about more

In summary

- Focus on moving pixels on the screen, removing waste
- You'll get a lot of value from a platform
- Kubernetes means making distributed apps, defining runtime
- Enterprise architecture as a code
- Modernization is key for enterprises
- Modernize your portfolio with a deliberate, evolving strategy
- Always focus on the actual software, solving people's problems

Sources: Sophie Seiwald, Daimler, case study in *The Business Bottleneck*. Pic: Luke Kanies