

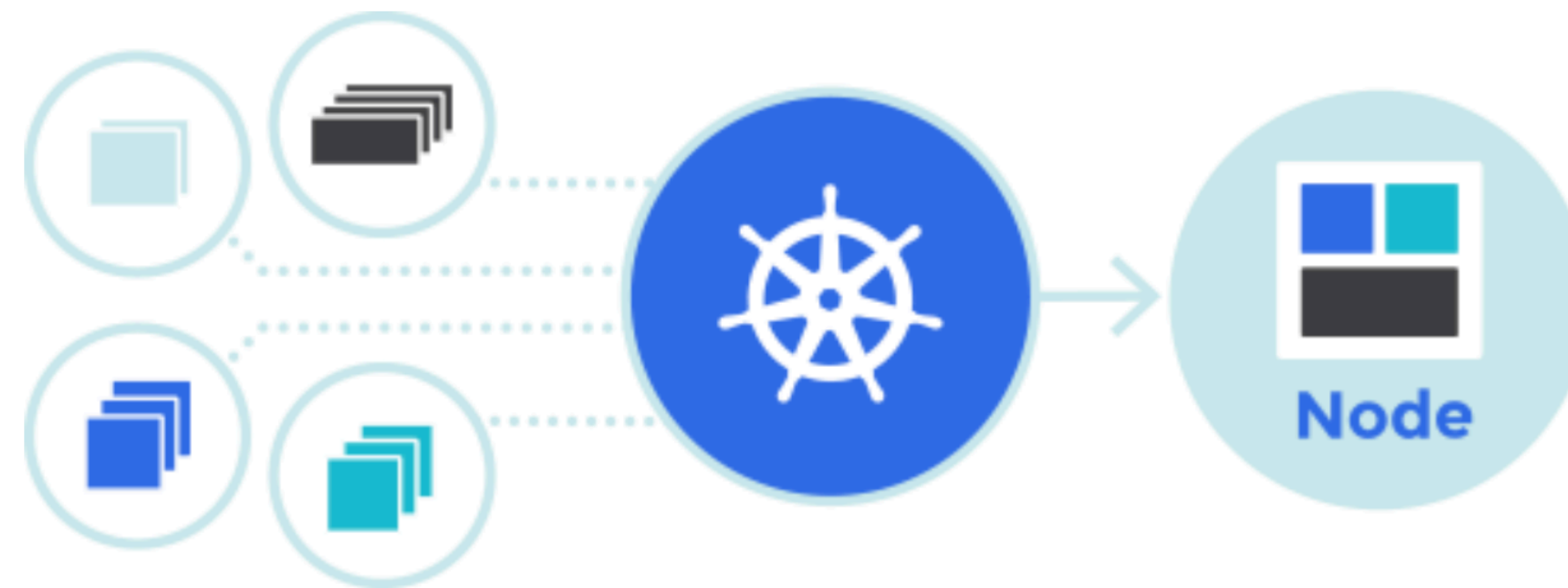
@MELANIECEBULA / MARCH 2019 / CON LONDON

Developing Kubernetes Services

at Airbnb Scale



What is kubernetes?



Kubernetes (k8s) is an open-source system for automating deployment, scaling, and management of containerized applications.

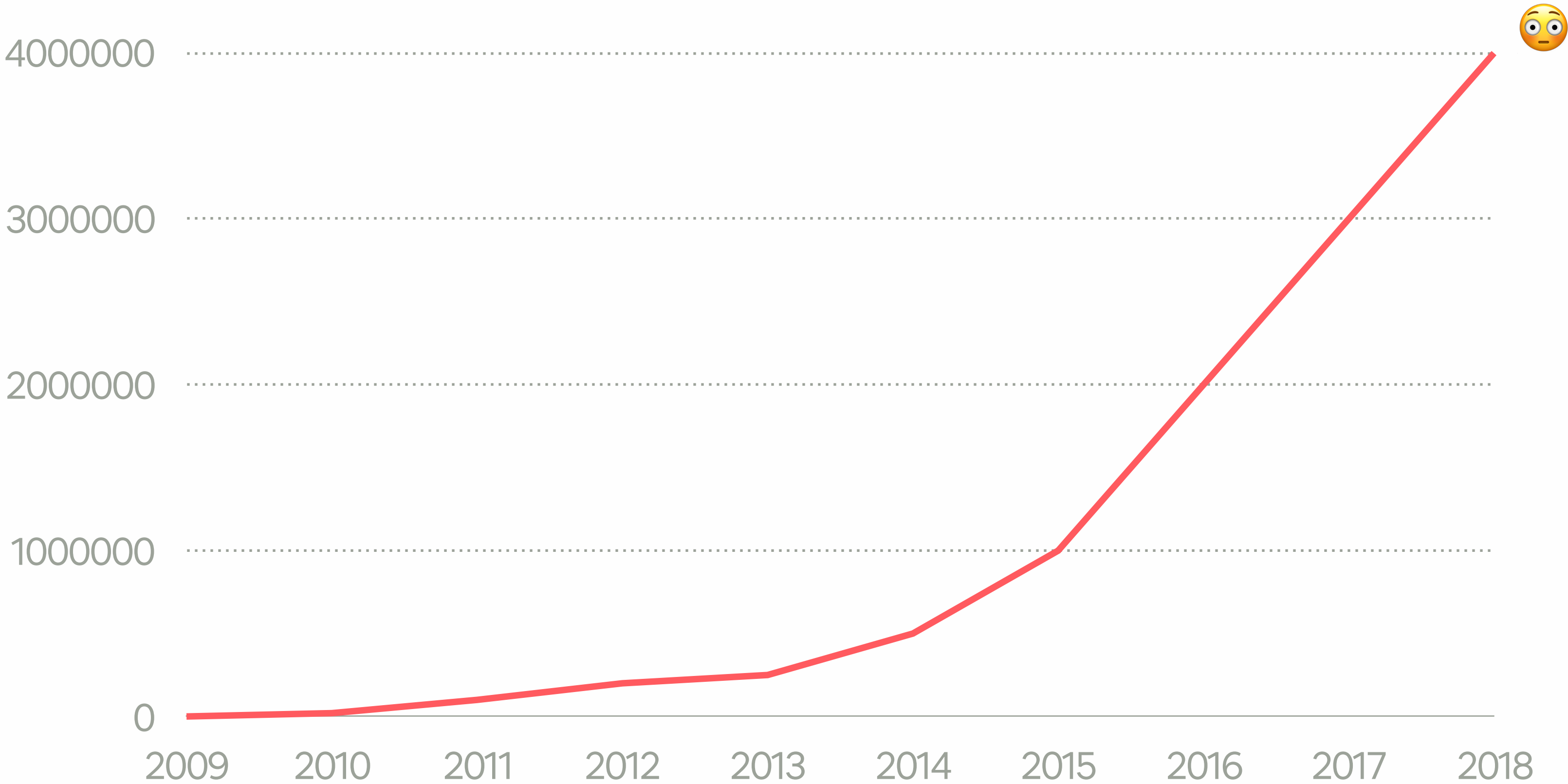
Who am I?

A BRIEF HISTORY

Why Microservices?

@MELANIECEBULA

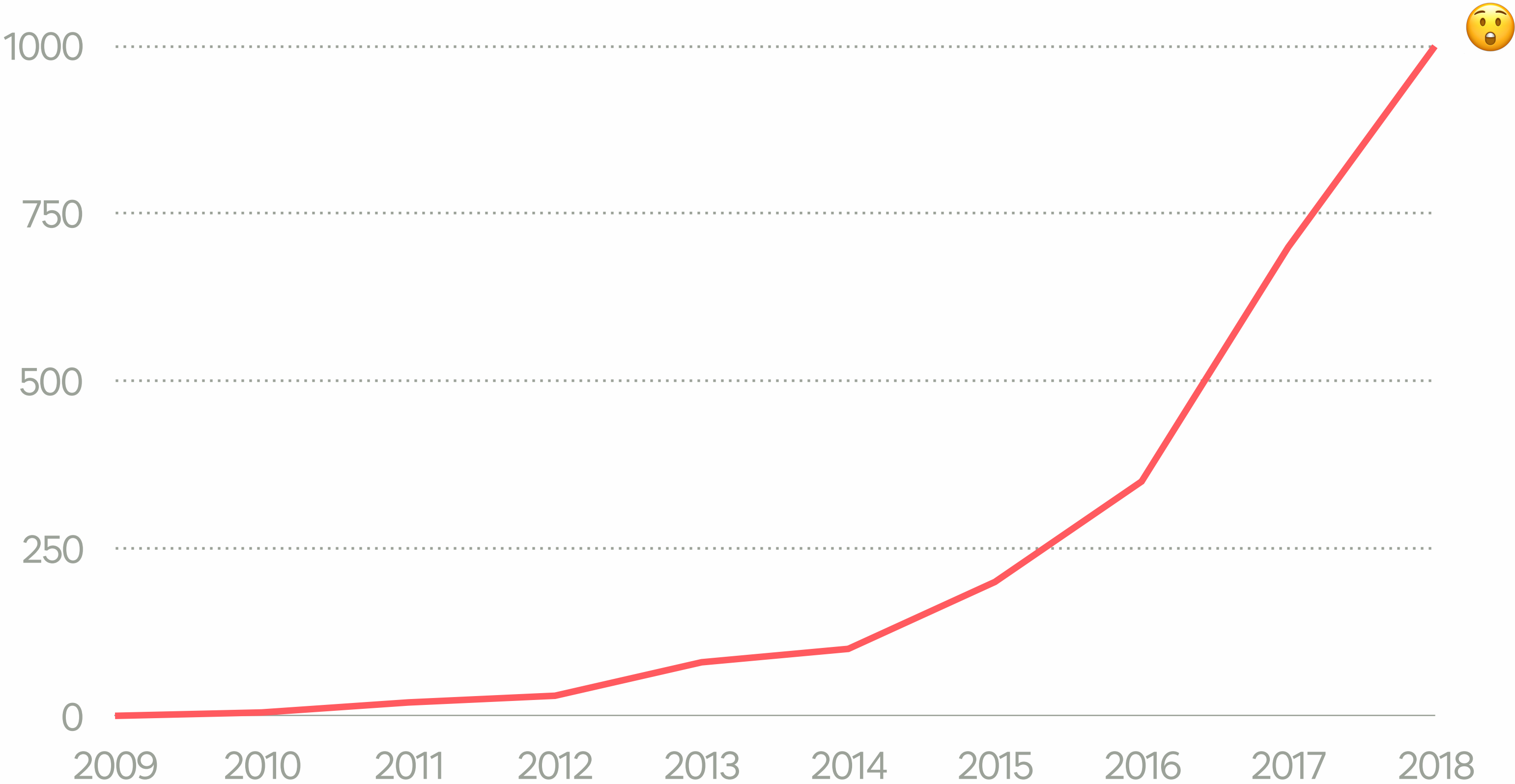
MONOLITH LOC



Why Microservices?

@MELANIECEBULA

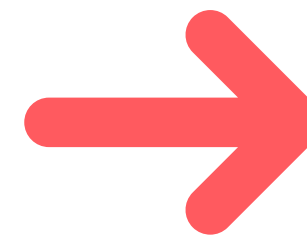
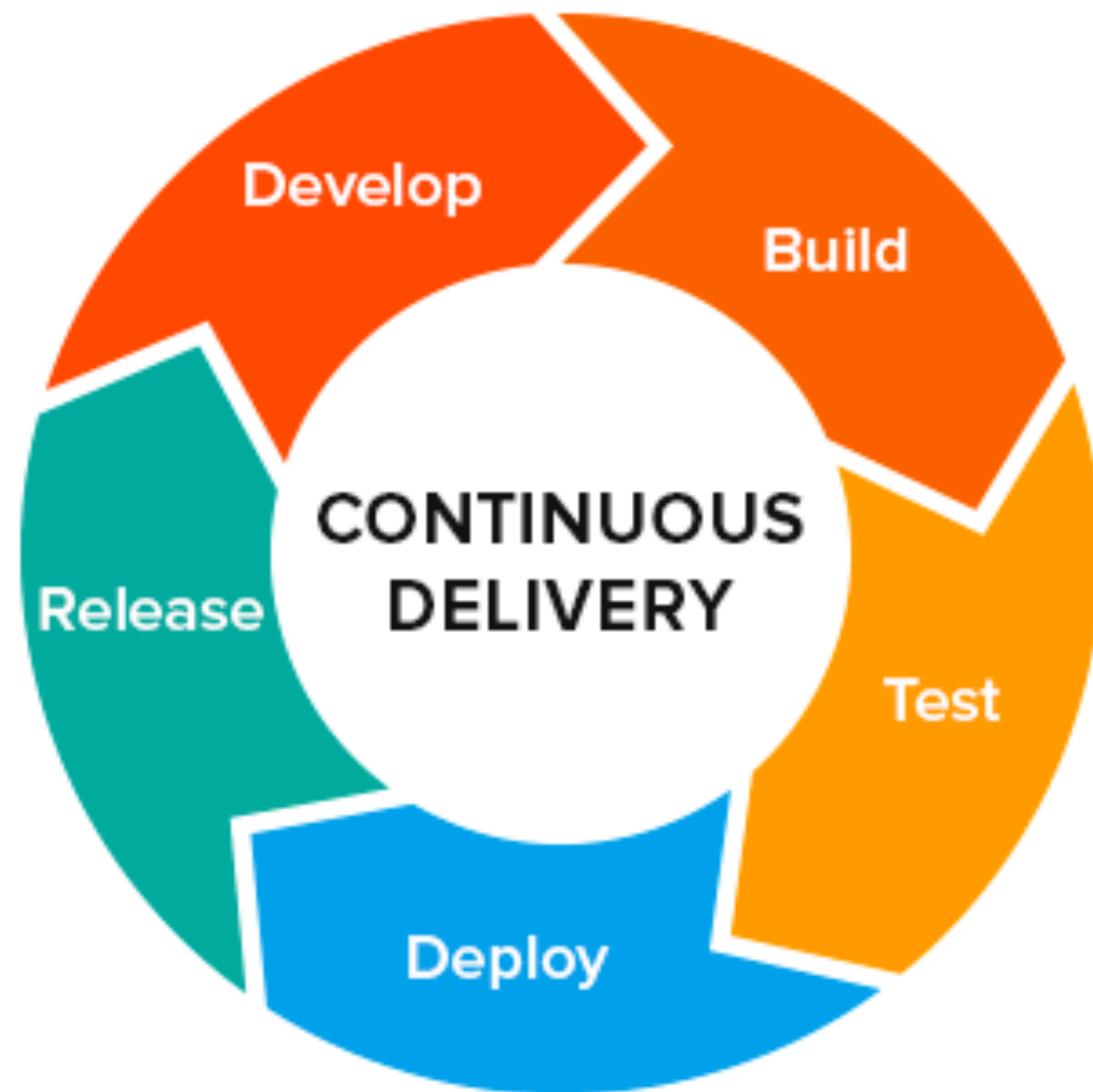
ENGINEERING TEAM



Why Microservices?

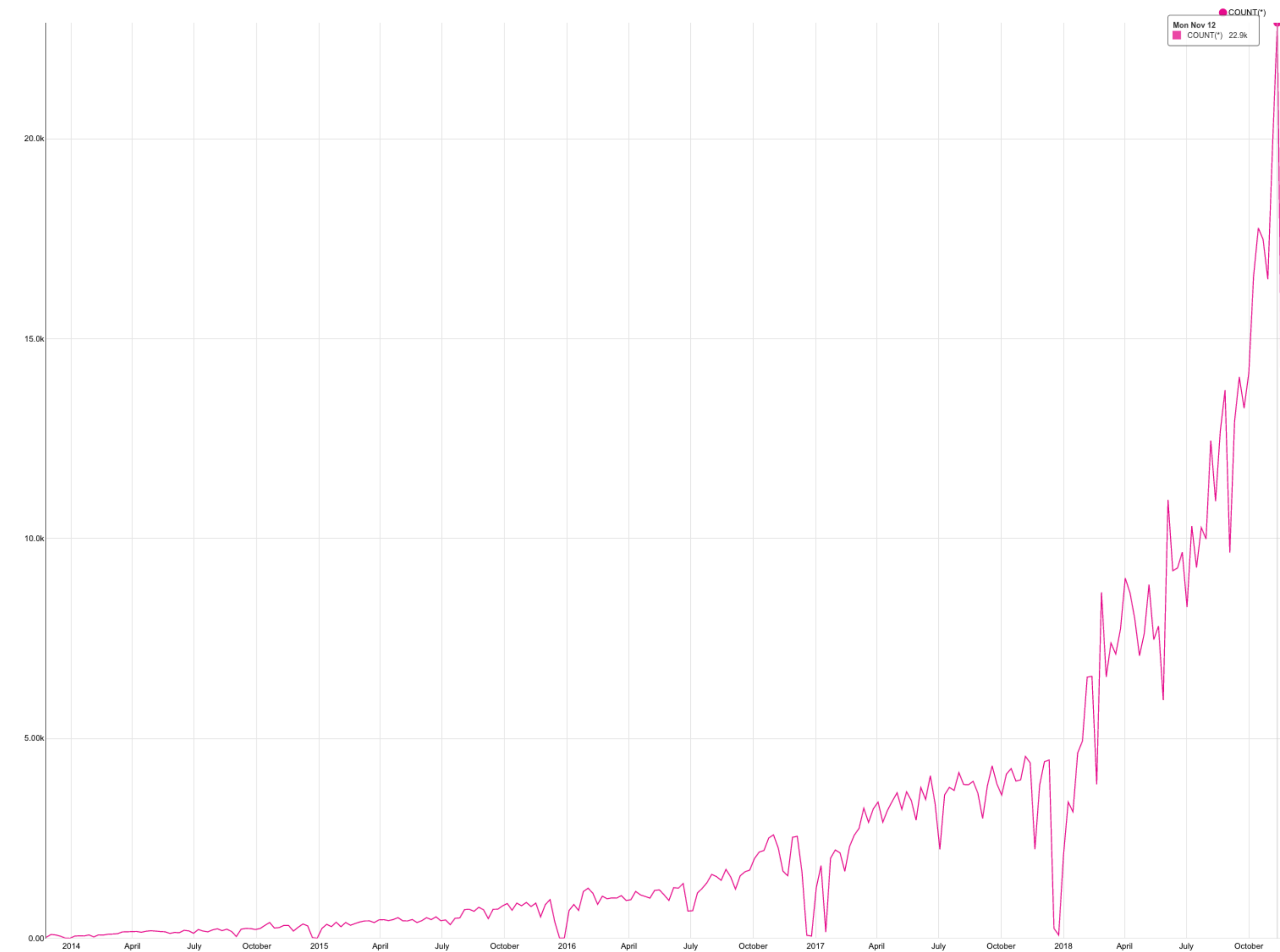
SCALING CONTINUOUS DELIVERY

@MELANIECEBULA



Why Microservices?

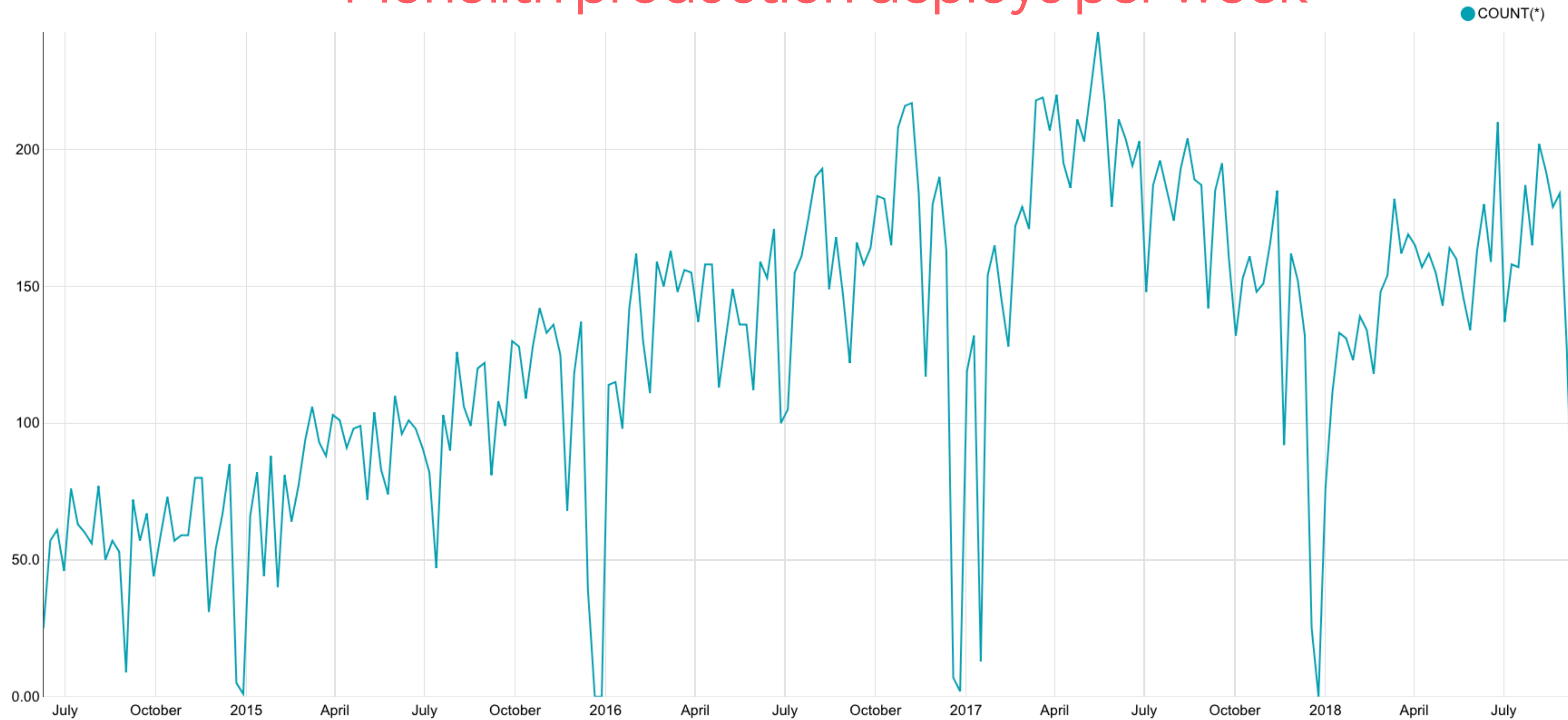
Deploys per week (all apps, all environments)



Why Microservices?

@MELANIECEBULA

Monolith production deploys per week



Why Microservices?

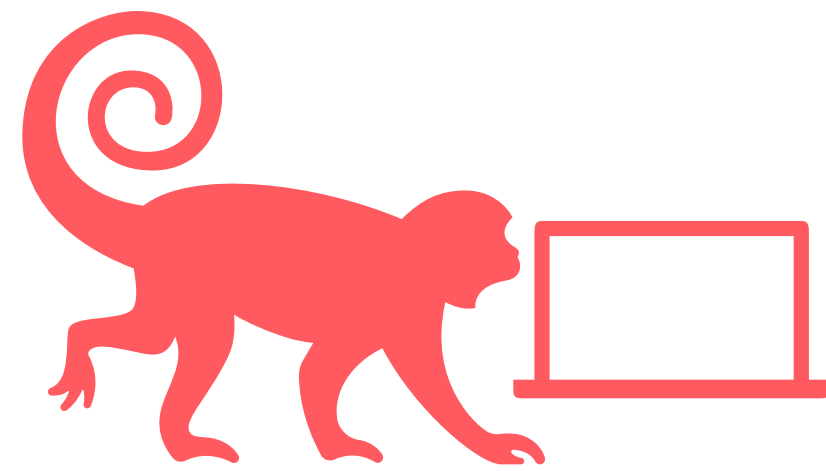
@MELANIECEBULA

125,000
production deploys
per year

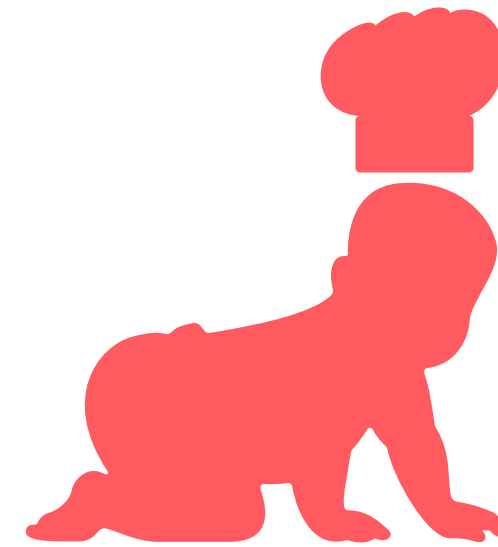
Why kubernetes?

@MELANIECEBULA

EVOLUTION OF CONFIGURATION MANAGEMENT



Manually configuring
boxes



Automate
configuration of
applications with Chef



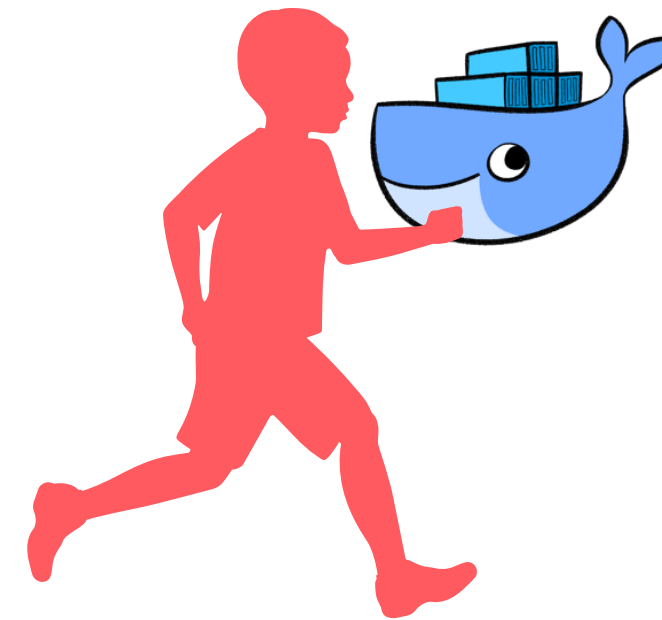
Automate
configuration and
orchestration of
containerized
applications with
Kubernetes

Why kubernetes?

@MELANIECEBULA



- declarative
- efficient scheduling
- extensible API



- portable
- immutable
- reproducible



- human-friendly data
- standard format

TODAY

50% of services
in kubernetes

250+ critical services
in kubernetes

Challenges with kubernetes?

@MELANIECEBULA

- complex configuration
- complex tooling
- integrating with your current infrastructure
- open issues
- scaling
- ... and more!

Challenges with kubernetes?

- complex configuration
- complex tooling
- integrating with your current infrastructure
- open issues
- scaling
- ... and more!



solvable problems!

you are not alone.

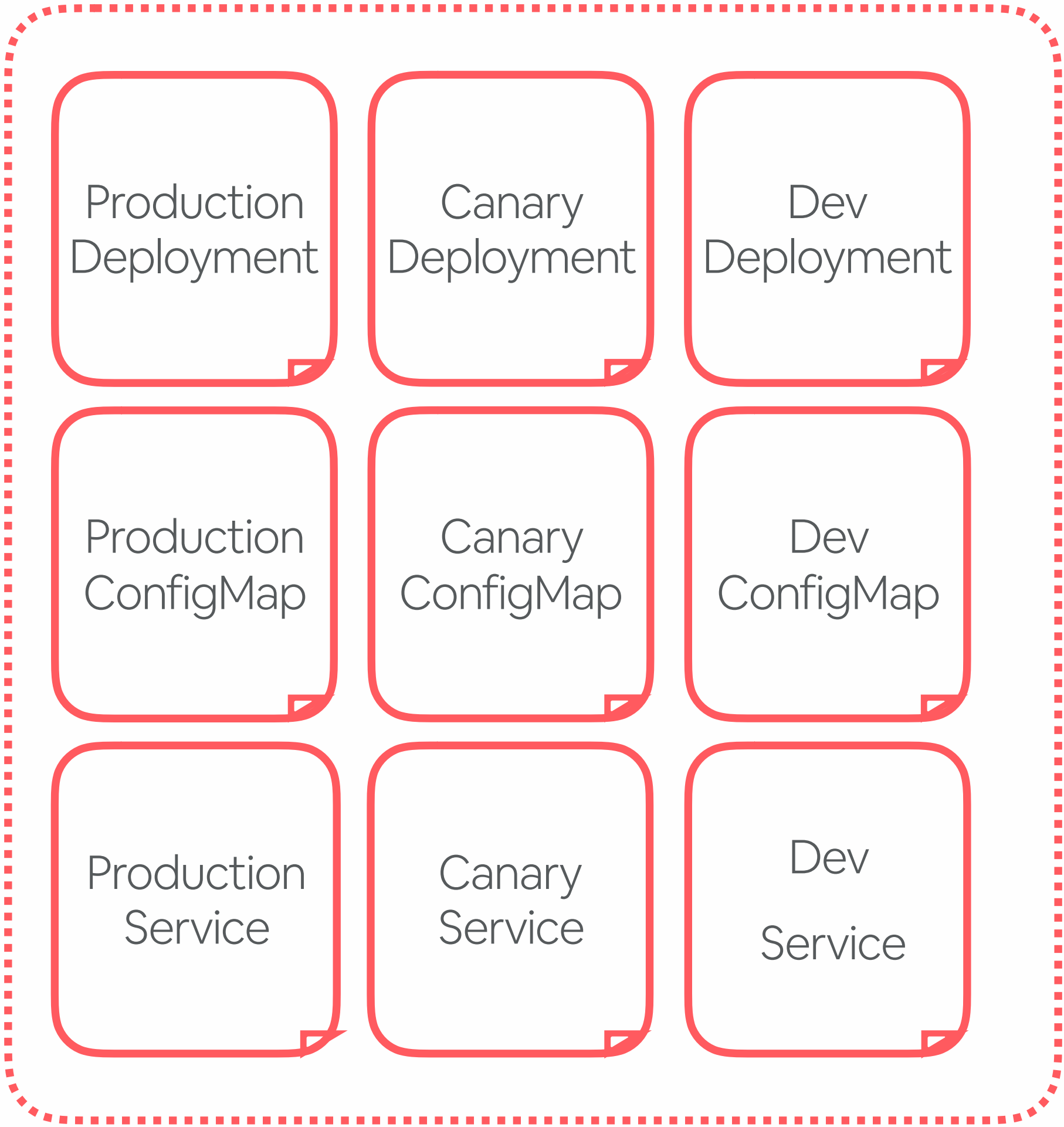
Solutions?

@MELANIECEBULA

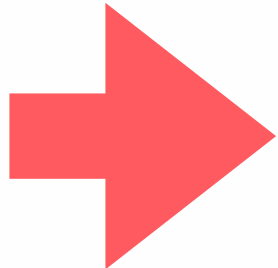
1. abstract away k8s configuration
2. generate service boilerplate
3. version + refactor configuration
4. opinionated kubectl
5. custom ci/cd + validation

ABSTRACT AWAY CONFIGURATION

kubernetes config files

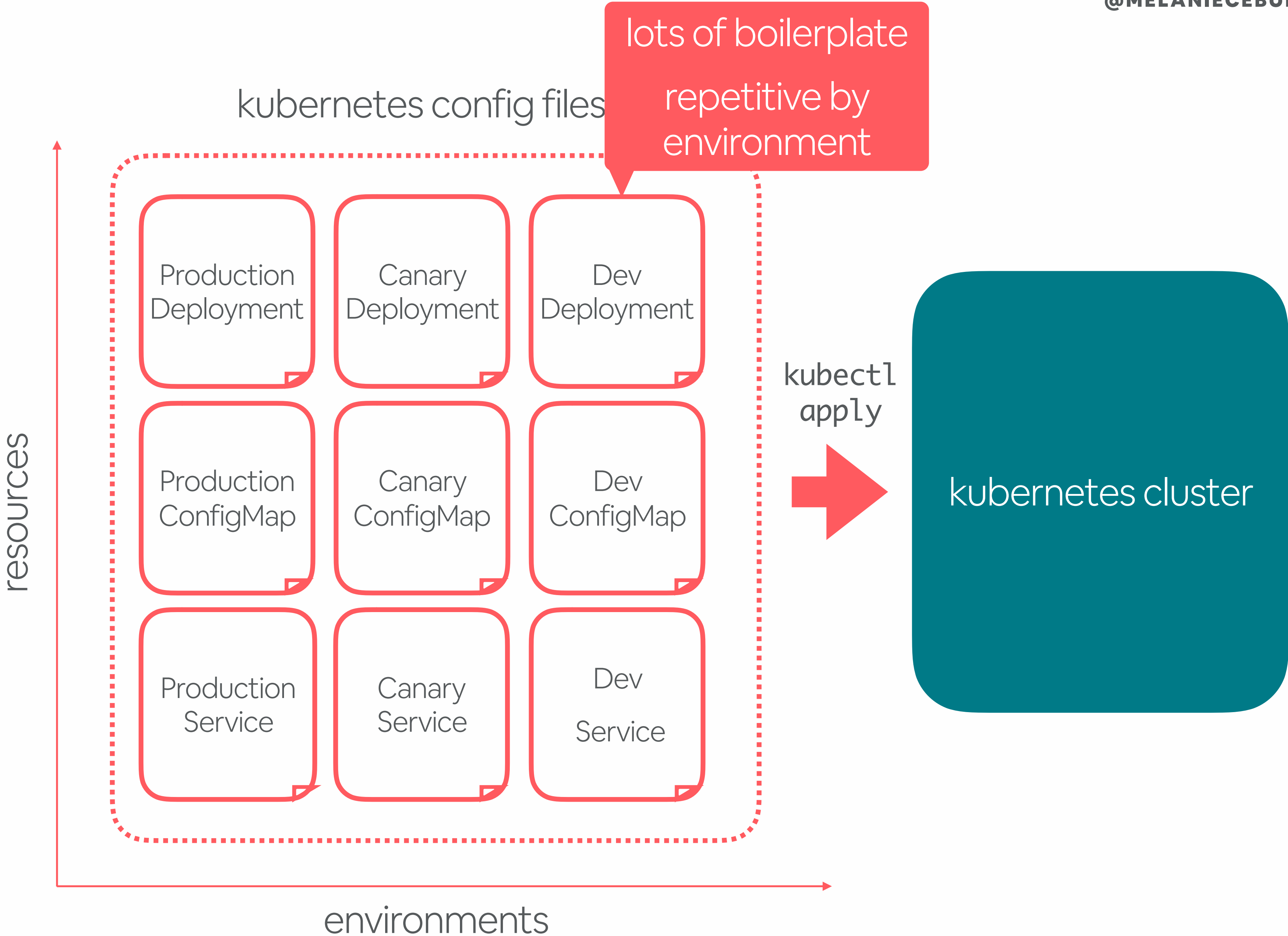


kubectl
apply



kubernetes

kubernetes



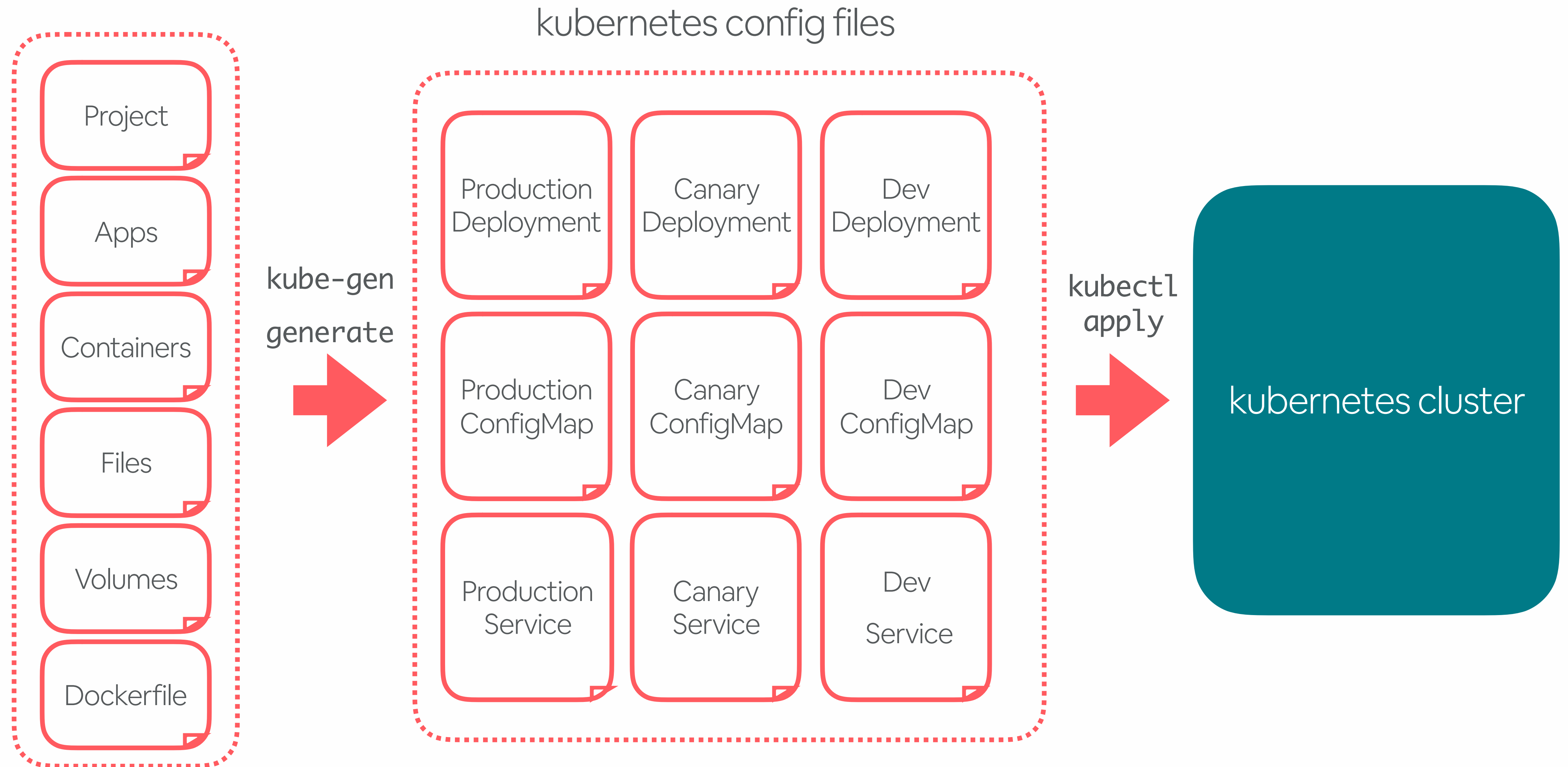
Reducing k8s boilerplate

OUR REQUIREMENTS

- **Prefer templating** over file inheritance
- Input should be **templated YAML** files
- Make it easier to **migrate 100s of legacy services**
- Make it easier to **retrain 1000 engineers**

generating k8s configs

@MELANIECEBULA



generating k8s

@MELANIECEBULA

kube-gen!

kubernetes config files

Project

Apps

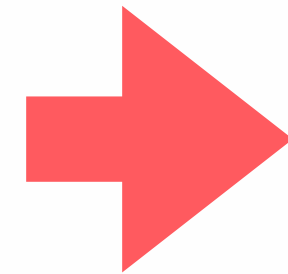
Containers

Files

Volumes

Dockerfile

kube-gen
generate



Production
Deployment

Canary
Deployment

Dev
Deployment

Production
ConfigMap

Canary
ConfigMap

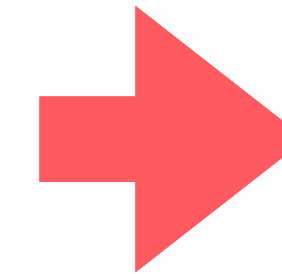
Dev
ConfigMap

Production
Service

Canary
Service

Dev
Service

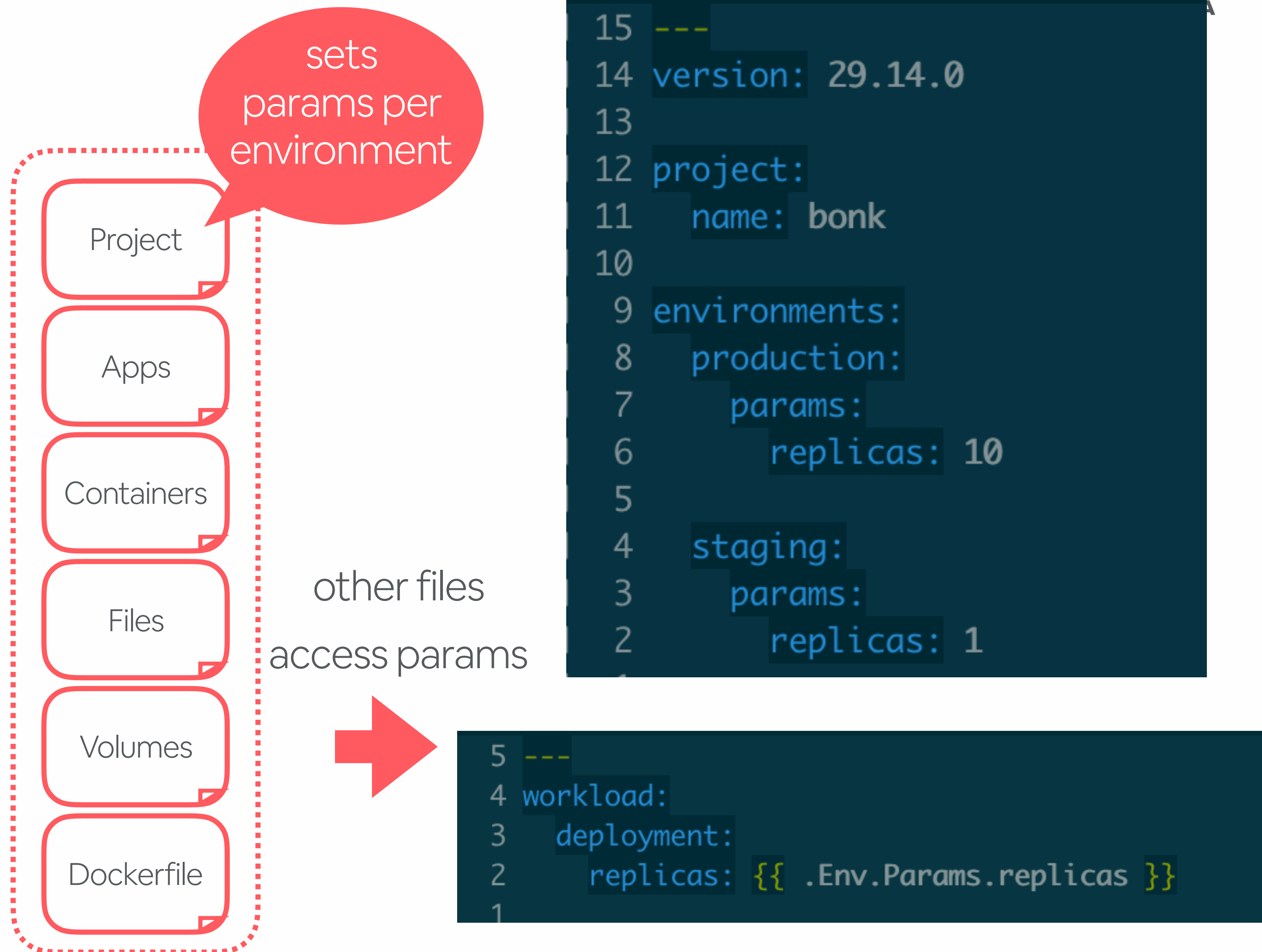
kubectl
apply



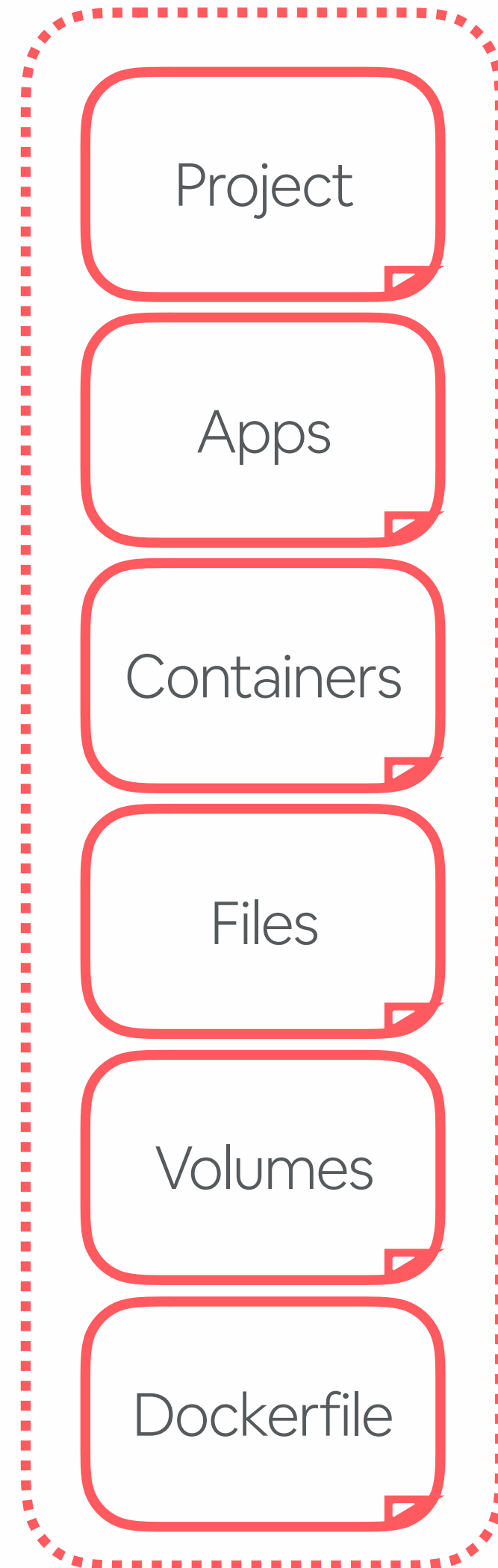
kubernetes cluster

Reducing k8s boilerplate

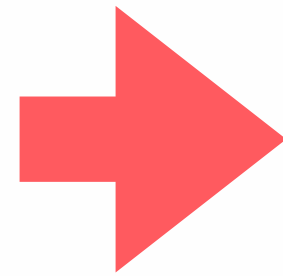
WHAT WE WENT WITH



generating k8s configs

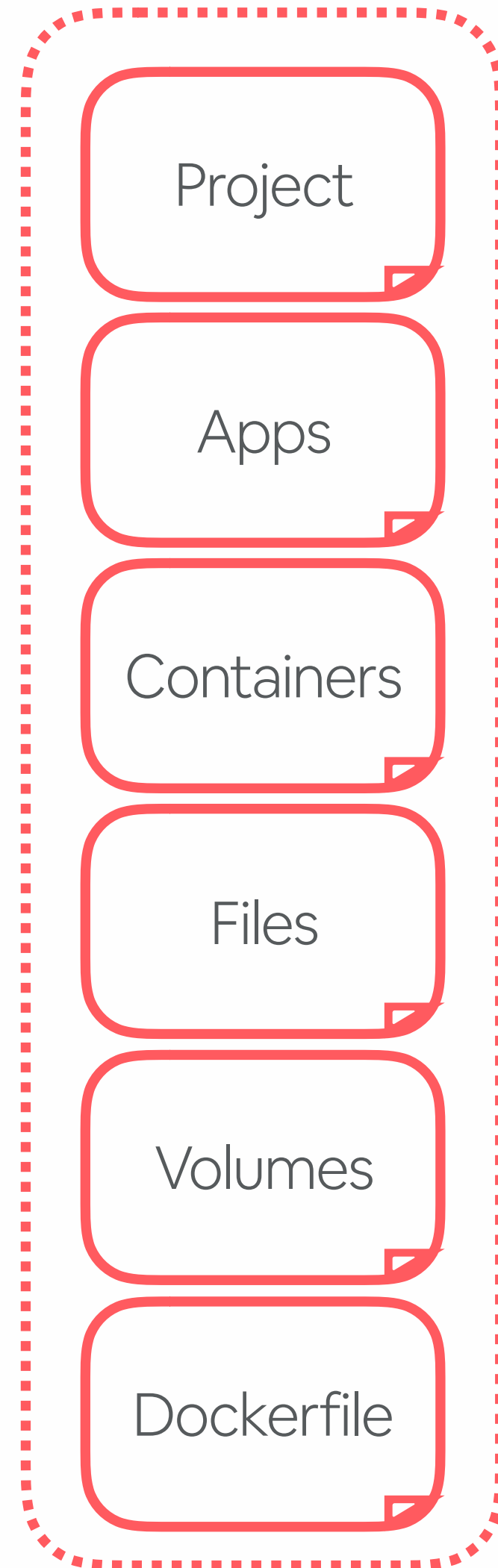


kube-gen
generate

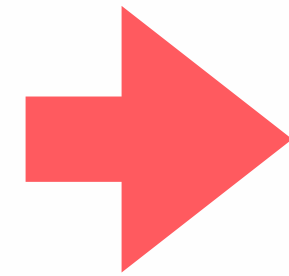


```
/Users/melanie_cebula/onetouch-codelabs/projects/bonk/generated-apps/bonk/  
▸ bonk-canary/  
▸ bonk-development/  
▾ bonk-production/  
    bonk-production-admin-role-binding.yml  
    bonk-production-databag-bonk-configmap.yml  
    bonk-production-deployment.yml  
    bonk-production-mini-announcer-configmap.yml  
    bonk-production-service-config-map-configmap.yml  
    bonk-production-service.yml  
    bonk-production-synapse-configmap.yml  
    bonk-production-zoned-key-configmap.yml  
▸ bonk-staging/
```

generating k8s configs



kube-gen
generate

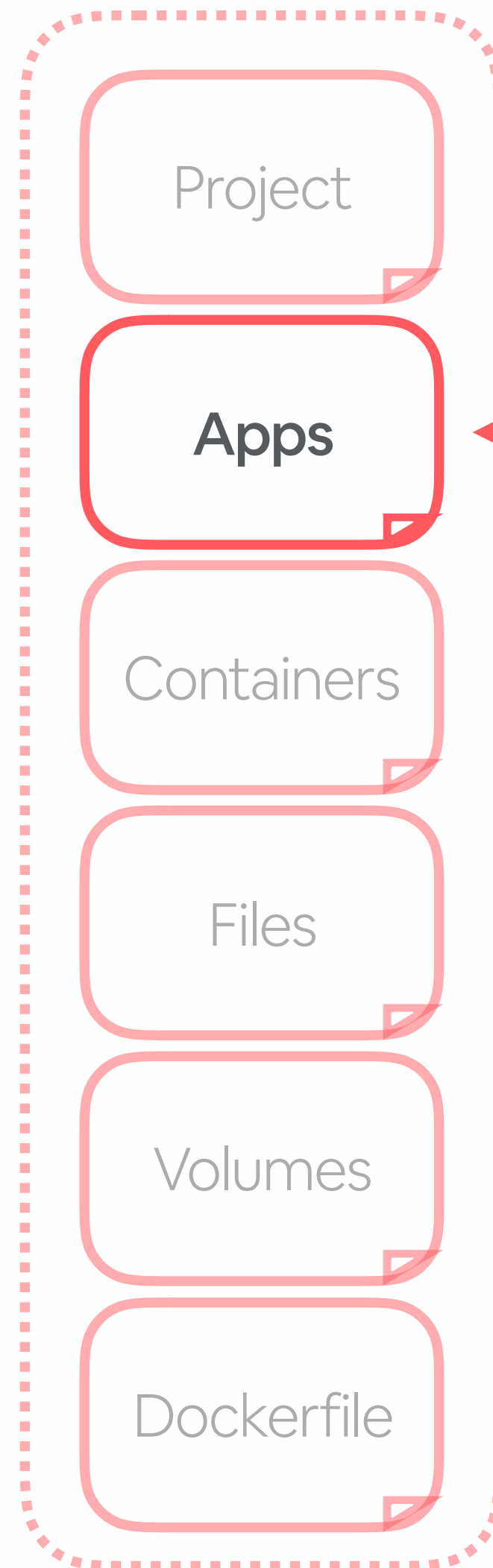


```
/Users/melanie_cebula/onetouch-codelabs/projects/bonk/generated-apps/bonk/  
└─ bonk-canary/  
└─ bonk-development/  
└─ bonk-production/  
    bonk-production-a  
    bonk-production-d  
    bonk-production-d  
    bonk-production-mini-announcer-configmap.yml  
    bonk-production-service-config-map-configmap.yml  
    bonk-production-service.yml  
    bonk-production-synapse-configmap.yml  
    bonk-production-zoned-key-configmap.yml  
└─ bonk-staging/
```

standardized
namespaces based on
environments!

kube-gen

COMPONENTS



Which shared components
to use?

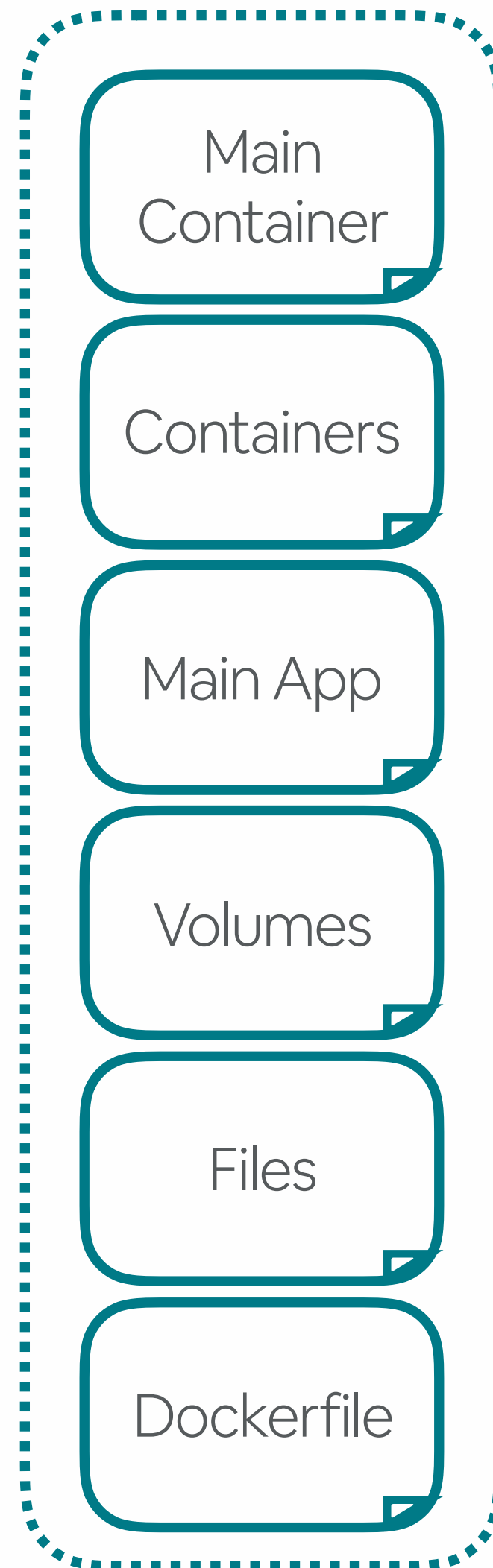
example components



kube-gen

COMPONENTS

nginx component



- common pattern abstracted into component
- component yams is merged into project on generate
- components can require or set default params

Reducing k8s boilerplate

OPEN SOURCE OPTIONS

1. Combine with package management (ex: helm)
2. Override configuration via file inheritance (ex: kustomize)
3. Override configuration via templating (ex: kapitan)



Takeaways

- Reduce kubernetes boilerplate
- Standardize on environments and namespaces

GENERATE SERVICE BOILERPLATE

Everything about a service is in one place in git, and managed with one process.

Configuration

LIVES IN ONE PLACE

```
/Users/melanie_cebula/bonk/
└─ _infra/
   ├── ci/
   ├── docs/
   ├── keys/
   ├── kube/
   ├── secrets/
   ├── airtlab.yml
   ├── aws.yml
   ├── deployboard.yml
   ├── dyno.yml
   └── project.yml
└─ app/
└─ bin/
└─ config/
└─ db/
└─ lib/
└─ log/
└─ public/
└─ spec/
└─ tmp/
└─ vendor/
   ├── config.ru
   ├── Gemfile
   ├── Gemfile.lock
   ├── Rakefile
   ├── README.md
   └── unicorn.rb
```

@MELANIECEBULA

Everything about a service is in one place in git

- All configuration lives in `_infra` alongside project code
- Edit code and configuration with one pull request
- Easy to add new configuration
- Statically validated in CI/CD

Configuration

LIVES IN ONE PLACE

```
/Users/melanie_cebula/bonk/
└─ _infra/
   ├── ci/
   ├── docs/
   ├── keys/
   ├── kube/
   ├── secrets/
   │   ├── airtlab.yml
   │   ├── aws.yml
   │   ├── deployboard.yml
   │   ├── dyno.yml
   │   └── project.yml
   ├── app/
   ├── bin/
   ├── config/
   ├── db/
   ├── lib/
   ├── log/
   ├── public/
   ├── spec/
   ├── tmp/
   └── vendor/
       ├── config.ru
       ├── Gemfile
       ├── Gemfile.lock
       ├── Rakefile
       ├── README.md
       └── unicorn.rb
```

@MELANIECEBULA

What we support:

- kube-gen files
- framework boilerplate
- API boilerplate
- CI/CD
- docs
- AWS IAM roles
- project ownership
- storage
- .. and more!

Configuration

LIVES IN ONE PLACE

```
/Users/melanie_cebula/bonk/  
└─ _infra/  
  ├── ci/  
  ├── docs/  
  ├── keys/  
  ├── kube/  
  ├── secrets/  
  │   ├── airtlab.yml  
  │   ├── aws.yml  
  │   ├── deployboard.yml  
  │   ├── dyno.yml  
  │   └── project.yml  
  ├── app/  
  ├── bin/  
  ├── config/  
  ├── db/  
  ├── lib/  
  ├── log/  
  ├── public/  
  ├── spec/  
  ├── tmp/  
  └── vendor/  
      ├── config.ru  
      ├── Gemfile  
      ├── Gemfile.lock  
      ├── Rakefile  
      ├── README.md  
      └── unicorn.rb
```

this “hello world”
service was created
in one command

Configuration

LIVES IN ONE PLACE

```
/Users/melanie_cebula/bonk/
```

```
▼ _infra/
```

```
▶ ci/
```

```
▶ docs/
```

```
▶ keys/
```

```
▶ kube/
```

```
▶ secrets/
```

```
airlab.yml
```

```
aws.yml
```

```
deployboard.yml
```

```
dyno.yml
```

```
project.yml
```

```
▶ app/
```

```
▶ bin/
```

```
▶ config/
```

```
▶ db/
```

```
▶ lib/
```

```
▶ log/
```

```
▶ public/
```

```
▶ spec/
```

```
▶ tmp/
```

```
▶ vendor/
```

```
config.ru
```

```
Gemfile
```

```
Gemfile.lock
```

```
Rakefile
```

```
README.md
```

```
unicorn.rb
```

collection of config
generators (ex: docs,
ci)

Configuration

LIVES IN ONE PLACE

```
/Users/melanie_cebula/bonk/  
└─ _infra/  
  ├── ci/  
  ├── docs/  
  ├── keys/  
  ├── kube/  
  ├── secrets/  
  │   ├── airtlab.yml  
  │   ├── aws.yml  
  │   ├── deployboard.yml  
  │   ├── dyno.yml  
  │   └── project.yml  
  ├── app/  
  ├── bin/  
  ├── config/  
  ├── db/  
  ├── lib/  
  ├── log/  
  ├── public/  
  ├── spec/  
  ├── tmp/  
  └── vendor/  
      ├── config.ru  
      ├── Gemfile  
      ├── Gemfile.lock  
      ├── Rakefile  
      ├── README.md  
      └── unicorn.rb
```

collection of
framework-specific
generators (ex: Rails,
Dropwizard)

Configuration

CAN BE GENERATED

```
/Users/melanie_cebula/bonk/
└─ _infra/
   ├── ci/
   ├── docs/
   ├── keys/
   ├── kube/
   ├── secrets/
   ├── airtlab.yml
   ├── aws.yml
   ├── deployboard.yml
   ├── dyno.yml
   └── project.yml
└─ app/
└─ bin/
└─ config/
└─ db/
└─ lib/
└─ log/
└─ public/
└─ spec/
└─ tmp/
└─ vendor/
   ├── config.ru
   ├── Gemfile
   ├── Gemfile.lock
   ├── Rakefile
   ├── README.md
   └── unicorn.rb
```

@MELANIECEBULA

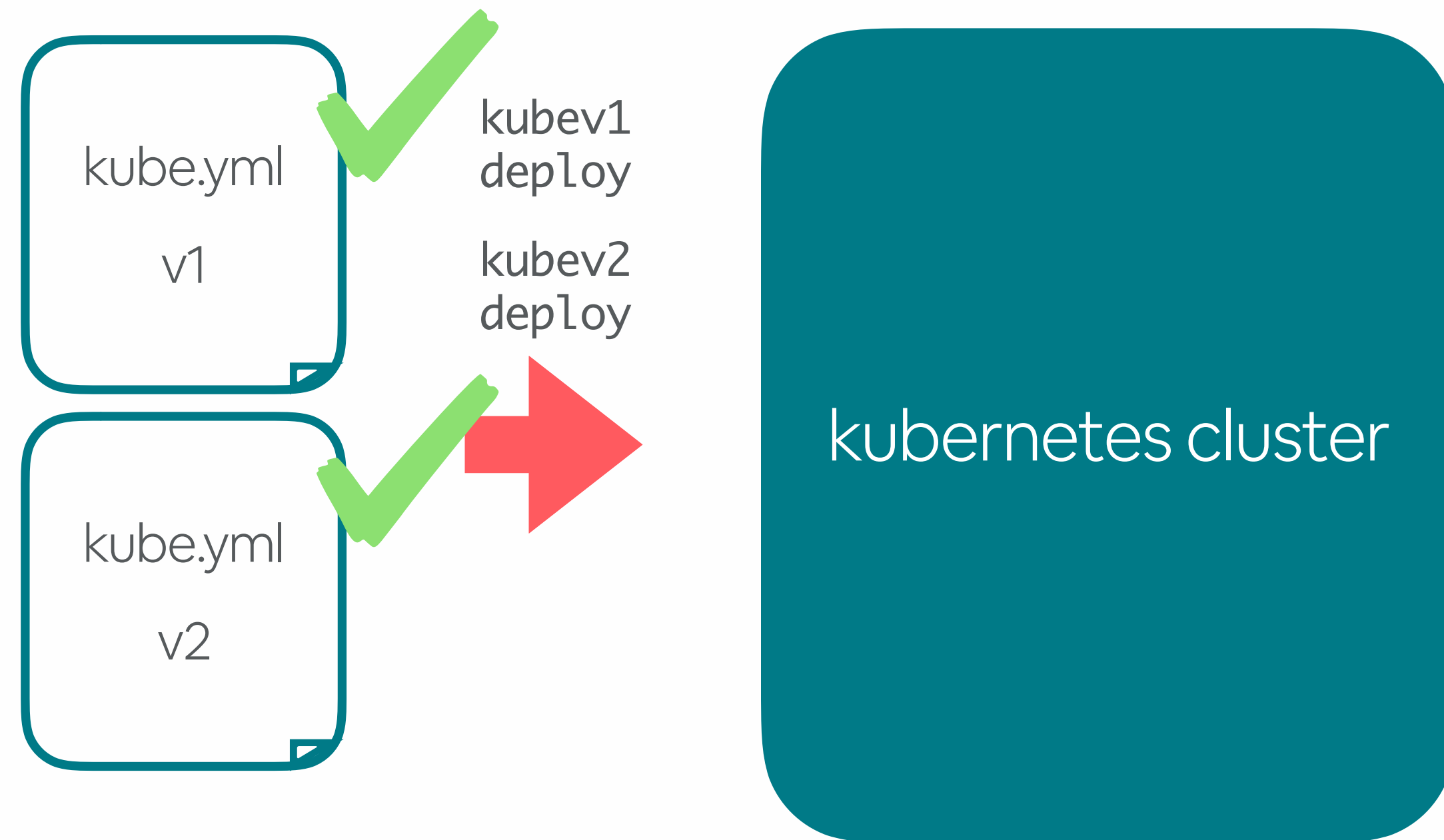
- make best practices the *default* (ex: deploy pipeline, autoscaling, docs)
- run generators individually or as a group
- support for review, update, commit

Takeaways

- Everything about a service should be in one place in git
- Make best practices the *default* by generating configuration

VERSION CONFIGURATION

Why do we version our kube configuration?



- add support for something new (ex: k8s version)
- want to change something (ex: deployment strategy)
- want to drop support for something (breaking change)
- know which versions are bad when we make a regression 😅
- support release cycle and cadence

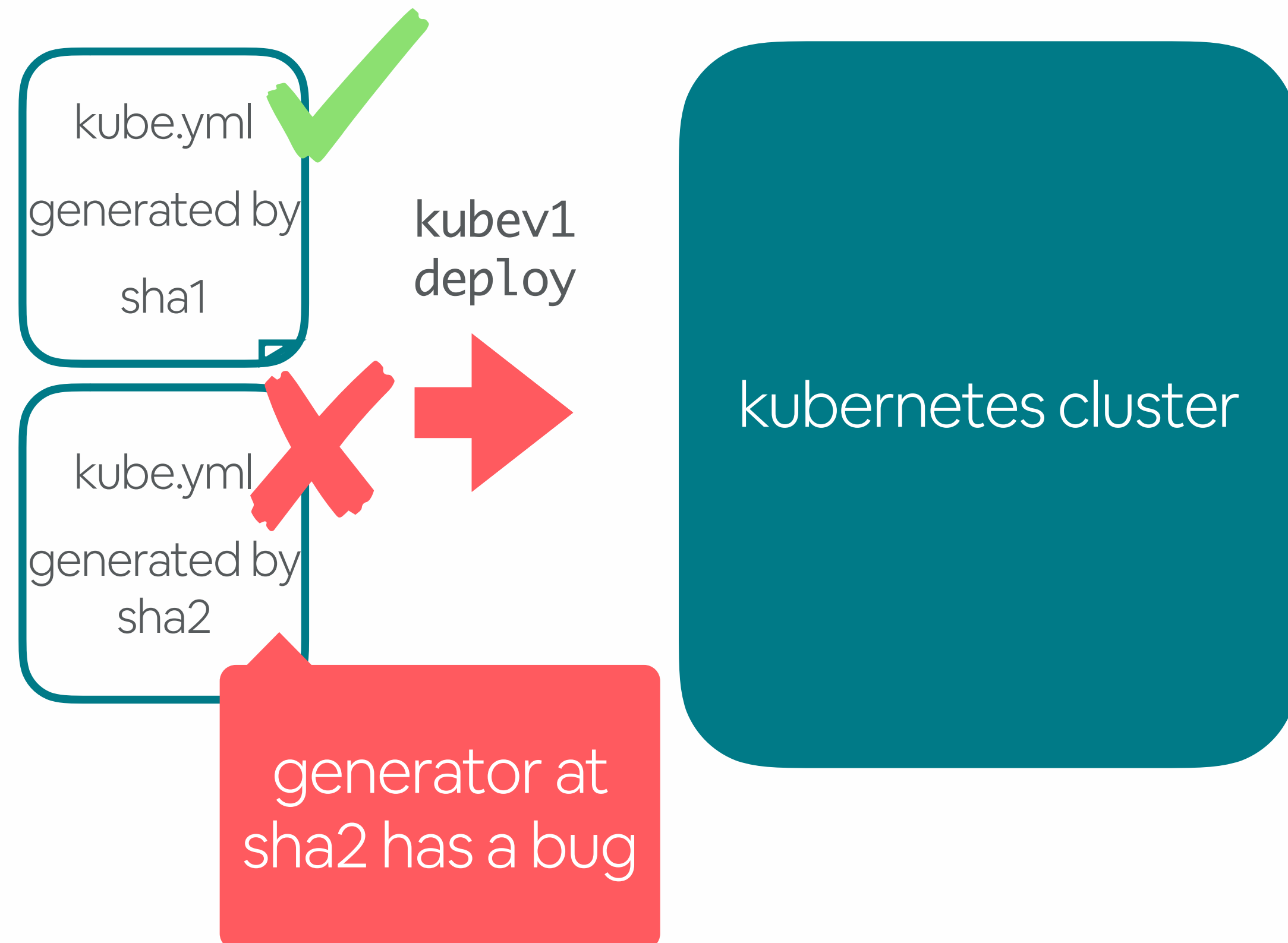
How do we version our kube configuration?

```
21 ---
20 version: 29.14.0
19
18 project:
17   name: bonk
16
15 common:
14   all:
13     params:
12       replicas: 1
11       port: 11002
10
9   production:
8     params:
7       iamRole: bonk-production
6
5   staging:
4     params:
3       iamRole: bonk-staging
2       port: 11003
1
```

1. version field
2. publish binaries for each version
3. channels point to binaries (ex: stable)
4. generate and apply using the appropriate binary

bonk kube-gen.yml

Why do we version our generated configuration?



- what our project generators generate changes over time
- best practices change
- and bugs in the generators are found! 😅

How do we version our generated configuration?

```
15 ---
14 # Generated by: onetouch-gen v1.2.0 (titanic: sha 4455be8f built 2019-01-04 01:22:12 UTC)
13 environments:
12   production:
11     - iam:
10       role_name: bonk-production
9       trusted_entities:
8         - Chef-KubernetesMinion
7       attached_policies: []
6   staging:
5     - iam:
4       role_name: bonk-staging
3       trusted_entities:
2         - Chef-KubernetesMinion
1       attached_policies: []
```

generator tags
generated files with
version, sha, and
timestamp

REFACTOR CONFIGURATION

Why do we refactor configuration?

FOR HUNDREDS OF SERVICES

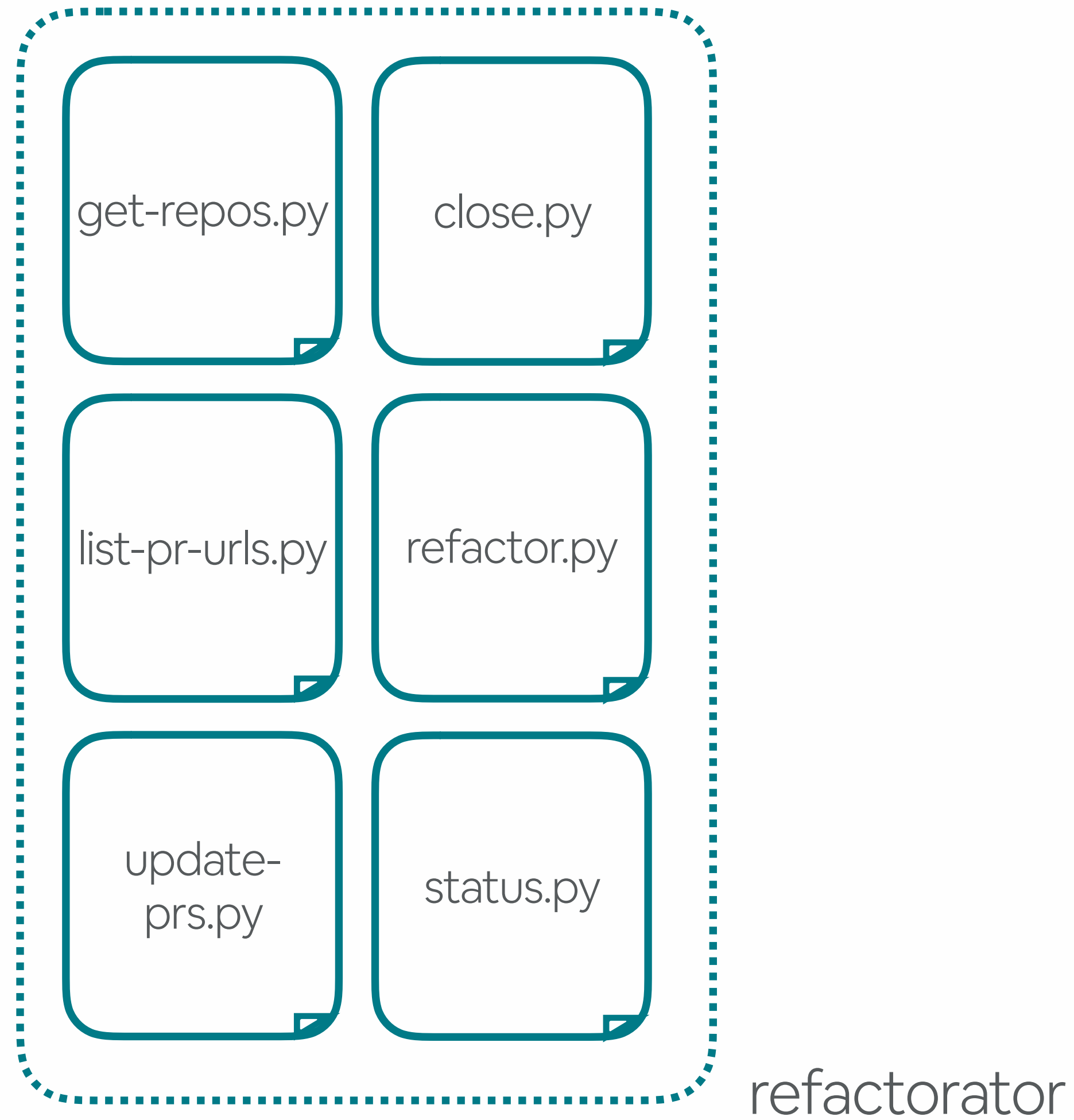
- services should be *up-to-date* with latest best practices
- update configuration to the latest supported versions
- apply security patches to images
- configuration migrations should be automated

(we don't want to manually refactor)

@MELANIECEBULA

250+ critical services
in kubernetes

How do we refactor configuration?



- collection of general purpose scripts
- scripts are modular
- scripts cover the lifecycle of a refactor

The lifecycle of a refactor

Run Refactor

Checks out repo, finds project, runs refactor job, tags owners, creates PR

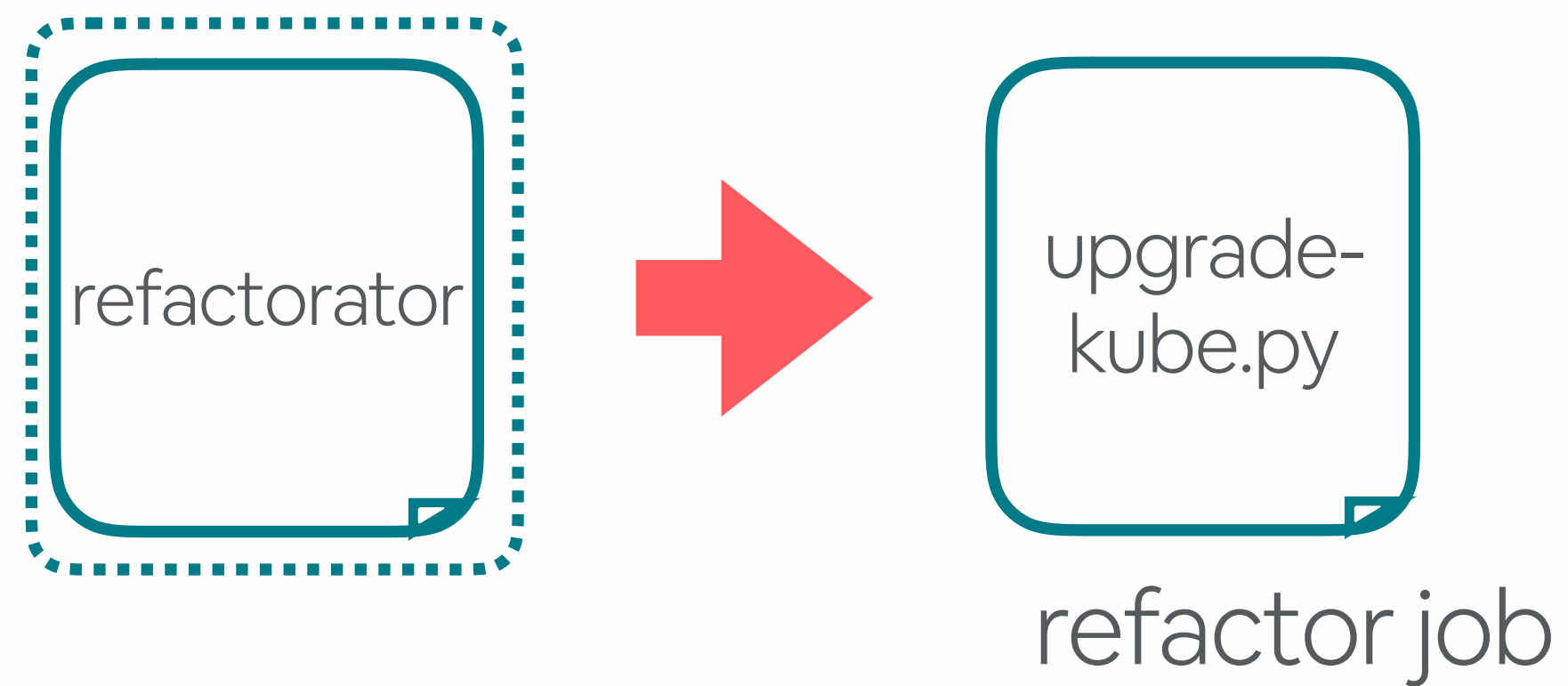
Update

Comments on the PR, reminding owners to verify, edit, and merge the PR

Merge

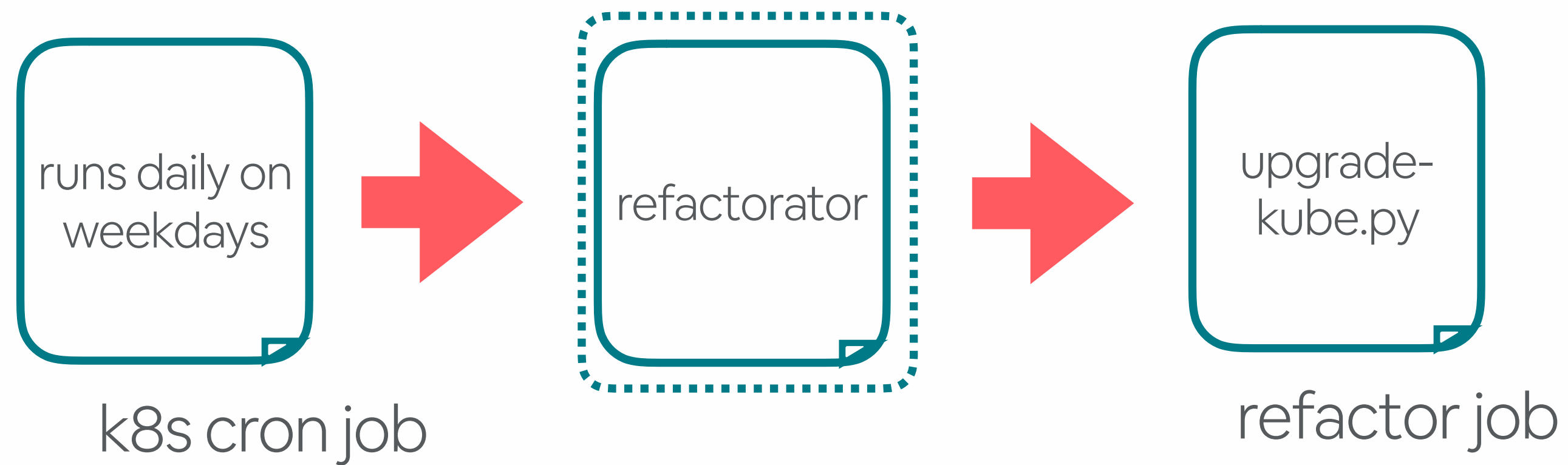
Merges the PR with different levels of force

How do we refactor configuration?



- refactorator will run a refactor for all services given a *refactor job*
- refactor job updates `_infra` file(s)
- ex: upgrade kube version to stable

Bumping stable version



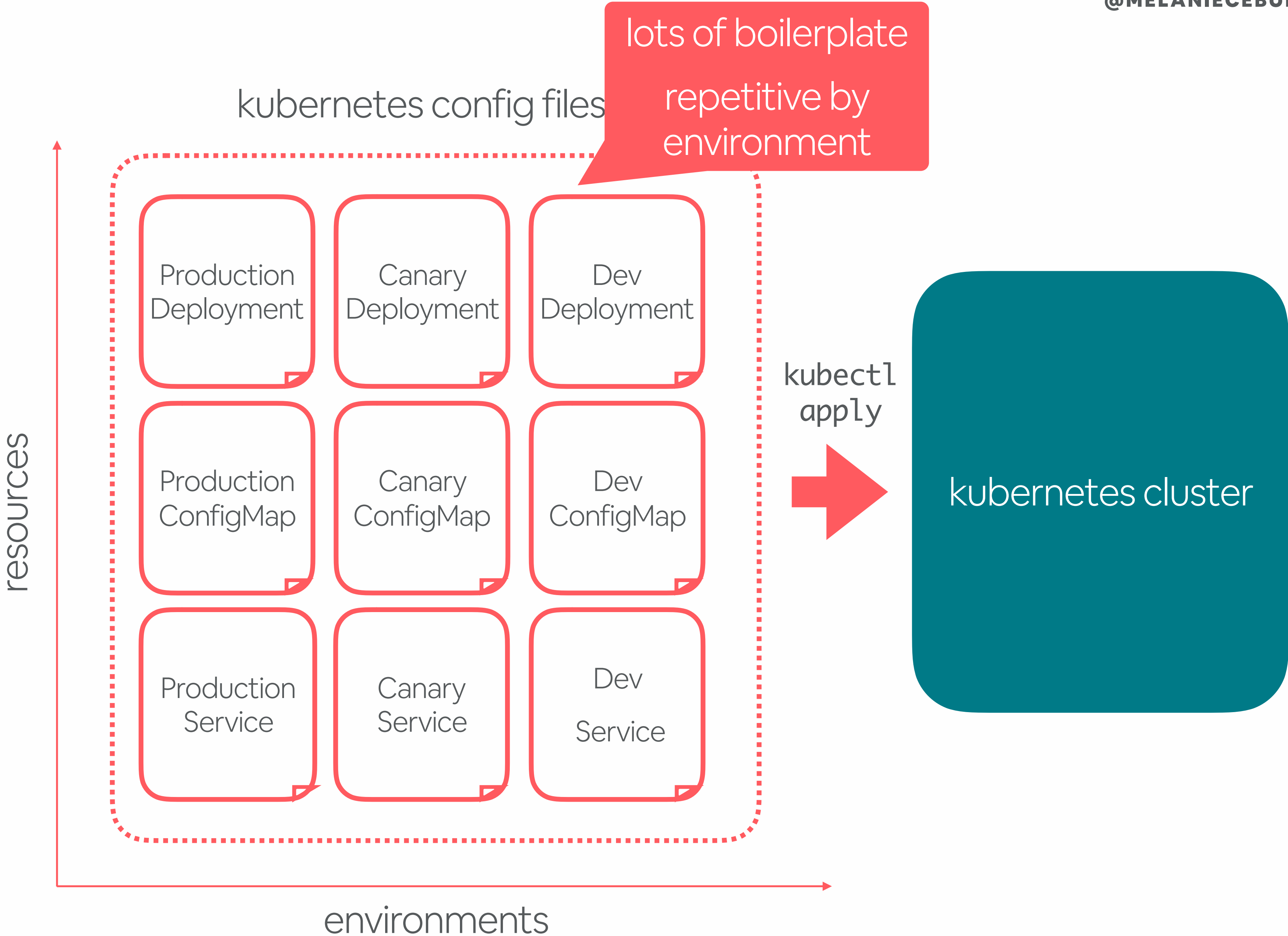
- bump stable version
- cron job uses refactorator and with the upgrade-kube.py refactor job to create PRs
- another cron job handling updating and merging PRs

Takeaways

- Configuration should be versioned and refactored automatically.

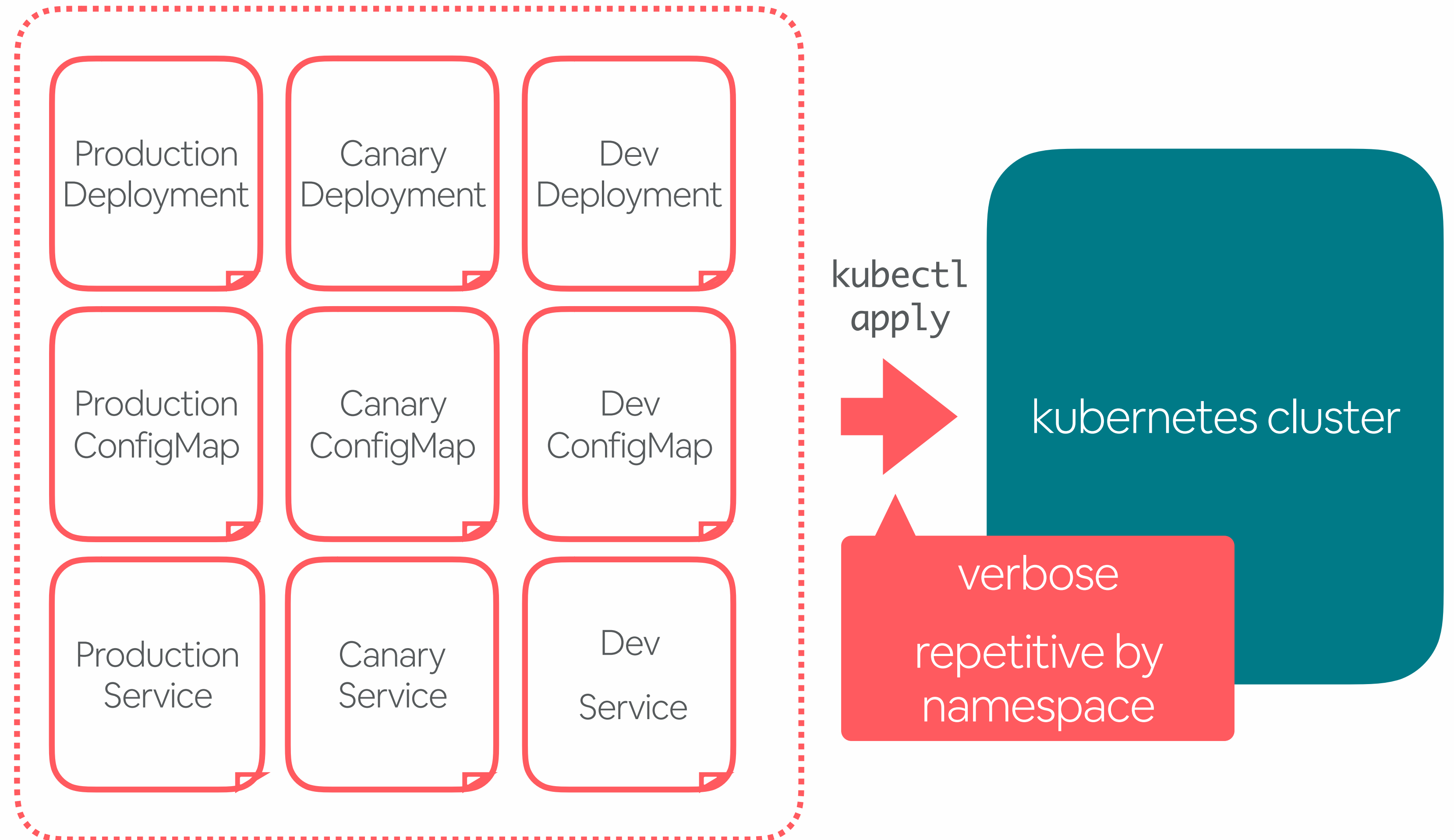
OPINIONATED KUBECTL

kubernetes



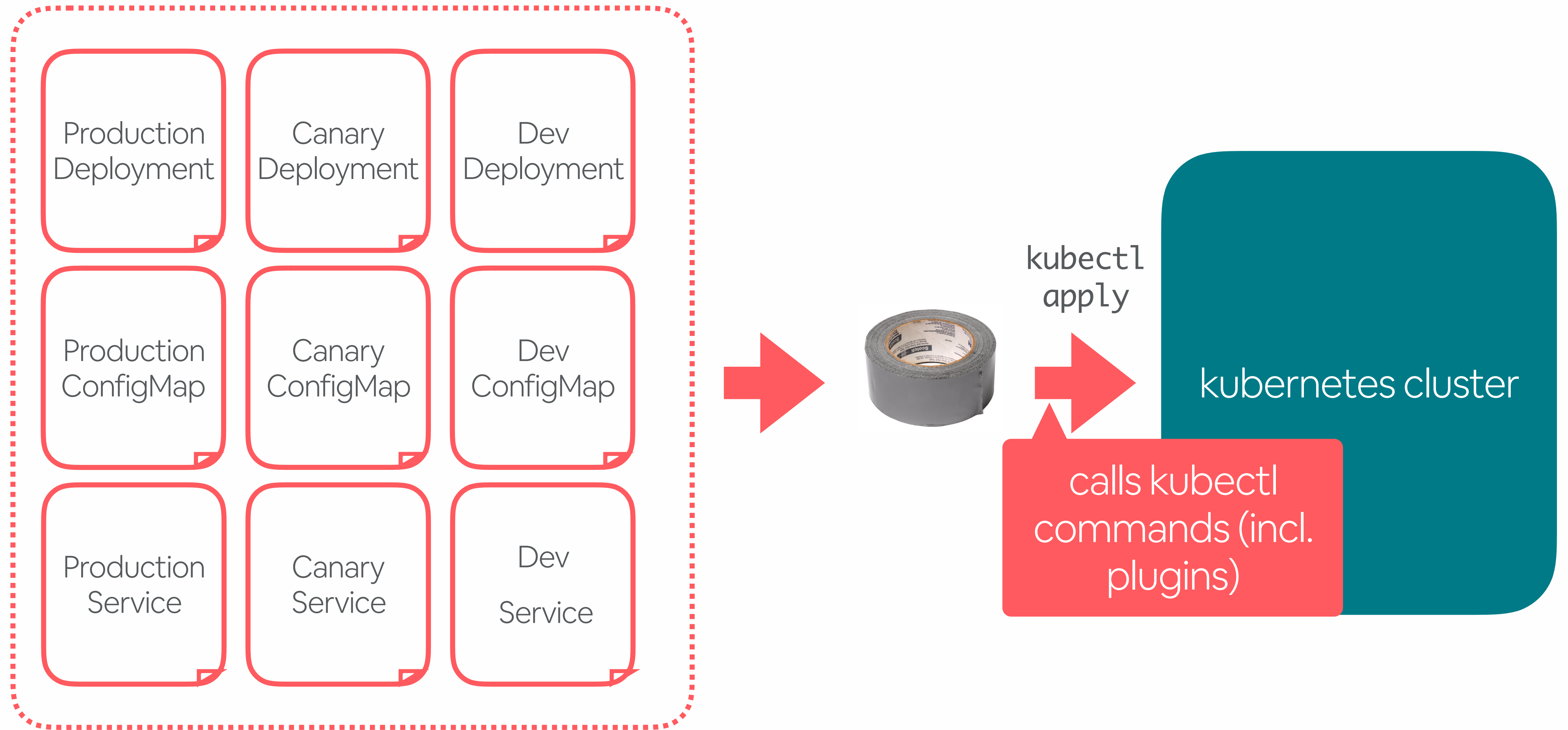
kubernetes

kubernetes config files



ktool

KUBECTL WRAPPER



k tool

OPINIONATED KUBECTL

@MELANIECEBULA



ktool

USES ENV VARS

- Runs in the project home directory:

```
$ cd /path/to/bonk
```

```
$ k status
```

- Environment variables for arguments:

```
$ k status ENV=staging
```

- Prints the command that it will execute:

```
$ k status ENV=staging
```

```
kubectl get pods --namespace=bonk-staging
```

standardized
namespaces!

k tool

SIMPLIFIES BUILDS AND
DEPLOYS

- `k generate` generates kubernetes files
- `k build` performs project build, docker build and docker push with tags
- `k deploy` creates namespace, applies/replaces kubernetes files, sleeps and checks deployment status
- can chain commands; ex: `k all`

k tool

A DEBUGGING TOOL

- defaults to random pod, main container:
`$ k ssh ENV=staging`
- specify particular pod, specific container:
`$ k logs ENV=staging POD=... CONTAINER=bonk`
- automates debugging with `k diagnose ENV=staging`

k tool

A DEBUGGING TOOL

- defaults to random pod, main container:

```
$ k ssh ENV=staging
```

- specify particular pod, specific container

```
$ k logs ENV=staging POD=... COM
```

- automates debugging with `k diagnose ENV=staging`

call kubectl diagnose

What are kubectl plugins?

Extend kubectl with plugins



FEATURE STATE: Kubernetes v1.13  beta

This guide demonstrates how to install and write extensions for [kubectl](#). By thinking of core `kubectl` commands as essential building blocks for interacting with a Kubernetes cluster, a cluster administrator can think of plugins as a means of utilizing these building blocks to create more complex behavior. Plugins extend `kubectl` with new sub-commands, allowing for new and custom features not included in the main distribution of `kubectl`.

What are kubectl plugins?

Installing kubectl plugins

A plugin is nothing more than a standalone executable file, whose name begins with `kubectl-`. To install a plugin, simply move this executable file to anywhere on your PATH.

k diagnose

SETUP

```
---
name: bonk

command:
- heyo

{{if ne .Env "port" "8080"}}
ports:
- containerPort: {{.Env.Params.port}}
{{end}}
```

deploy bonk
service with failing
command

```
melanie_cebula@melanie-cebula ~/d/bonk> kubectl get pods --namespace=bonk-staging
NAME                                READY   STATUS    RESTARTS   AGE
bonk-staging-64f6f7bb7c-4dtvs       8/8     Running   0           2h
bonk-staging-64f6f7bb7c-rzhwr       8/8     Running   0           2h
bonk-staging-c5946c7d-28k5z         6/8     CrashLoopBackOff   0           2h
```

new pod in
CrashLoopBackoff

k diagnose

MANUALLY

```
melanie_cebula@melanie-cebula ~/d/bonk> kubectl get pods --namespace=bonk-staging -o=yaml | grep "ready: false" -B 10 -A 5
lastState:
  terminated:
    containerID: docker://b02432874bb542773cab26f558a056344f460d467b5ffb5d7b292588c4c3b9e7
    exitCode: 127
    finishedAt: "2019-02-22T01:14:57Z"
    message: |
      oci runtime error: container_linux.go:247: starting container process caused "exec: \"heyo\": executable file not found in $PATH"
    reason: ContainerCannotRun
    startedAt: "2019-02-22T01:14:57Z"
name: bonk
ready: false
restartCount: 43
state:
  waiting:
    message: Back-off 5m0s restarting failed container=bonk pod=bonk-staging-c5946c7d-28k5z_bonk-staging(e2c...733c)
    reason: CrashLoopBackOff
```

1. use “get pods -o=yaml” and look for problems

2. grab logs for unready container

```
melanie_cebula@melanie-cebula ~/d/bonk> kubectl logs --namespace=bonk-staging bonk-staging-c5946c7d-28k5z -c bonk
container_linux.go:247: starting container process caused "exec: \"heyo\": executable file not found in $PATH"
```

k diagnose

MANUALLY

```
melanie_cebula@melanie-cebula ~/d/bonk> kubectl --namespace=bonk-staging get events -o=yaml --field-selector='involvedObject.name=bonk-staging-c5946c7d-28k5z' 2>/dev/null | grep reason -B 10
kind: Event
lastTimestamp: "2019-02-22T01:29:01Z"
message: Back-off restarting failed container
metadata:
  creationTimestamp: "2019-02-21T22:14:03Z"
  name: bonk-staging-c5946c7d-28k5z.15857ffd5310408e
  namespace: bonk-staging
  resourceVersion: "2425662024"
  selfLink: /api/v1/namespaces/bonk-staging/events/bonk-staging-c5946c7d-28k5z.15857ffd5310408e
  uid: ff46b92b-3625-11e9-95cd-1229ad3a733c
reason: BackOff
--
kind: Event
lastTimestamp: "2019-02-22T01:34:04Z"
message: "Readiness probe failed: health checks failed: \nbonk \n"
metadata:
  creationTimestamp: "2019-02-21T22:04:04Z"
  name: bonk-staging-c5946c7d-28k5z.1585800008314b1a
  namespace: bonk-staging
  resourceVersion: "2425732343"
  selfLink: /api/v1/namespaces/bonk-staging/events/bonk-staging-c5946c7d-28k5z.1585800008314b1a
  uid: 9a1630af-3624-11e9-95cd-1229ad3a733c
reason: Unhealthy
```

3. get k8s events
related to this pod

kubectl podevents

KUBECTL PLUGIN

```
#!/bin/bash
```

```
# get events for pod in namespace
```

```
NAMESPACE=$1
```

```
POD=$2
```

```
kubectl get events --namespace=$NAMESPACE -o=yaml --field-selector="involvedObject.name=$POD" |  
grep "count\|firstTimestamp\|message\|lastTimestamp\|reason"
```

kubectl podevents
plugin

kubectl diagnose

USES COBRA GO CLI

spf13 / cobra

<> Code

! Issues 158

🔗 Pull requests 47

A Commander for modern Go CLI interactions

```
// defines CLI command and flags

var Namespace string

var rootCmd = &cobra.Command{

    Use: "kubectl diagnose -namespace<namespace>"

    Short: "diagnoses a namespace with pods in CrashLoopBackOff"

    Run: func(cmd *cobra.Command, arg []string) {

        // Fill in with program logic

    }

}

func Execute() {

    rootCmd.Flags().StringVarP(&Namespace, "namespace", "n", "")

    rootCmd.MarkFlagRequired("namespace")

    if err := rootCmd.Execute(); err != nil {

        fmt.Println(err)

        os.Exit(1)

    }

}
```

@MELANIECEBULA

kubectl diagnose

USES K8S CLIENT-GO

kubernetes / client-go

Code

Issues 44

Pull requests

Go client for Kubernetes.

```
// get pods (assume Namespace is defined)
kubeconfig := filepath.Join(os.Getenv("HOME"), ".kube", "config")
config, err := clientcmd.BuildConfigFromFlags("", kubeconfig)
if err != nil { ... }

clientset, err := kubernetes.NewForConfig(config)
if err != nil { ... }

pods, err :=
clientset.CoreV1().Pods(Namespace).List(m

fmt.Printf("There are %d pods in the names
len(pods.Items), Namespace)

for _, pod := range pod.Items {

    podName := pod.Name

    for _, c := range pod.Status.ContainerStatuses {

        if c.Ready != true {

            // print c.LastTerminatedState and c.State

        }

    }

}
```

uses k8s client-go
and Namespace
param to get pods

kubectl diagnose

USES K8S CLIENT-GO

kubernetes / client-go

Code

Issues 44

Pull requests

Go client for Kubernetes.

@MELANIECEBULA

```
// get pods (assume Namespace is defined)
kubeconfig := filepath.Join(os.Getenv("HOME"), ".kube", "config")
config, err := clientcmd.BuildConfigFromFlags("", kubeconfig)
if err != nil { ... }

clientset, err := kubernetes.NewForConfig(config)
if err != nil { ... }

pods, err :=
clientset.CoreV1().Pods(Namespace).List(metav1.ListOptions{})

fmt.Printf("There are %d pods in the namespace %s\n",
len(pods.Items), Namespace)

for _, pod := range pod.Items {

    podName := pod.Name

    for _, c := range pod.Status.ContainerStatuses {

        if c.Ready != true {

            // print c.LastTerminatedState and c.State

        }

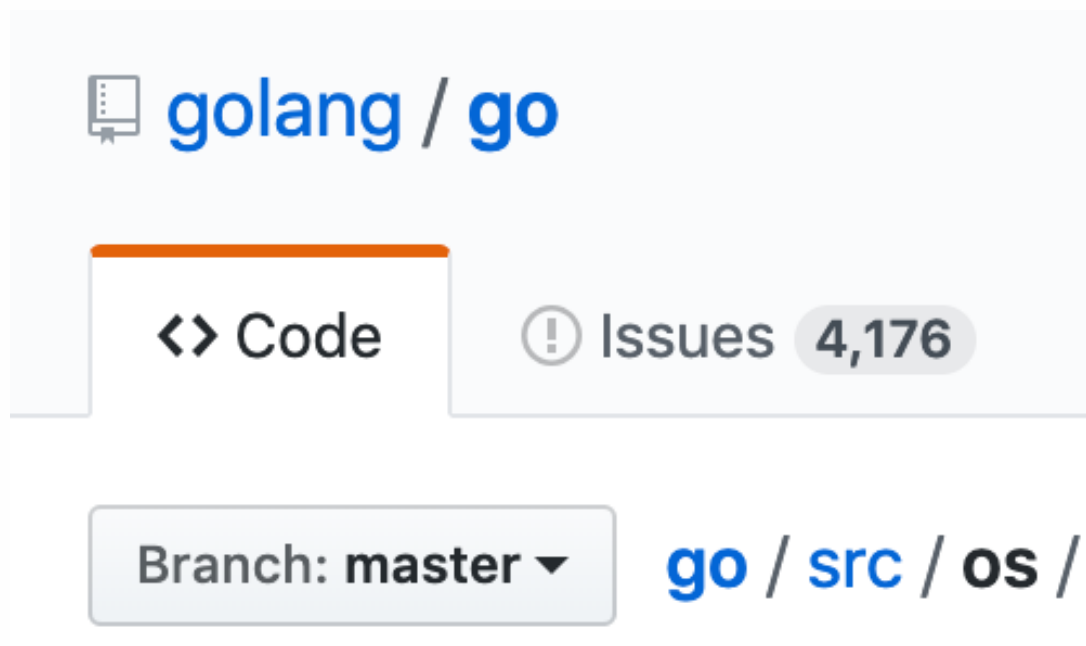
    }

}
```

prints info for all
unready containers

kubectl diagnose

USES OS/EXEC (WHEN LAZY)



```
// get pod events for namespace and pod
cmd := exec.Command("kubectl", "podevents", Namespace, podName)
var out bytes.Buffer
var stderr bytes.Buffer
cmd.Stdout = &out
cmd.Stderr = &stderr
err := cmd.Run()
if err != nil {
    fmt.Println(fmt.Sprintf(err) + ": " + stderr.String())
    log.Fatal(err)
} else {
    fmt.Println("Events: \n" + out.String())
}

// also grab logs
cmd = exec.Command("kubectl", "logs", podname, fmt.Sprintf("--
namespace=%s", Namespace), "-c", "bonk")
```

podevents kubectl
plugin

kubectl diagnose

GO KUBECTL PLUGIN

```
melanie_cebula@melanie-cebula ~/a/r/k-diagnose> kubectl diagnose --namespace=bonk-staging
There are 3 pods in the namespace bonk-staging
Pod bonk-staging-64f6f7bb7c-4dtvs in namespace bonk-staging is healthy
Pod bonk-staging-64f6f7bb7c-rzhwr in namespace bonk-staging is healthy
Container bonk is unready with restart count 42
LastState:
  exit code: 127
  signal: 0
  message: oci runtime error: container_linux.go:247: starting container process caused "exec: \"heyo\": executable file not found in $PATH"

  reason: ContainerCannotRun
State:
  message: Back-off 5m0s restarting failed container=bonk pod=bonk-staging-c5946c7d-28k5z_bonk-staging(e2a319bf-3623-11e9-95cd-1229ad3a733c)
  reason: CrashLoopBackOff
Container mini-announcer is unready with restart count 0
Pod bonk-staging-c5946c7d-28k5z in namespace bonk-staging is unhealthy
Events:
  count: 664
  firstTimestamp: "2019-02-21T21:59:32Z"
  message: Back-off restarting failed container
  reason: BackOff
  count: 1137
  firstTimestamp: "2019-02-21T21:59:44Z"
  message: "Readiness probe failed: health checks failed: \nbonk \n"
  reason: Unhealthy

Logs:
container_linux.go:247: starting container process caused "exec: \"heyo\": executable file not found in $PATH"
```


kubectl diagnose

GO KUBECTL PLUGIN

```
melanie_cebula@melanie-cebula ~/a/r/k-diagnose> kubectl diagnose --namespace=bonk-staging
There are 3 pods in the namespace bonk-staging
Pod bonk-staging-64f6f7bb7c-4dtvs in namespace bonk-staging is healthy
Pod bonk-staging-64f6f7bb7c-rzhwr in namespace bonk-staging is healthy
Container bonk is unready with restart count 0
LastState:
  exit code: 127
  signal: 0
  message: oci runtime error: container_linux.go:247: starting container process caused "exec: \"heyo\": executable file not found in $PATH"
  reason: ContainerCannotRun
State:
  message: Back-off 5m0s restarting failed container=bonk pod=bonk-staging-c5946c7d-28k5z_bonk-staging(e2a319bf-3623-11e9-95cd-1229ad3a733c)
  reason: CrashLoopBackOff
Container mini-announcer is unready with restart count 0
Pod bonk-staging-c5946c7d-28k5z in namespace bonk-staging is unhealthy
Events:
  count: 664
  firstTimestamp: "2019-02-21T21:59:32Z"
  message: Back-off restarting failed container
  reason: BackOff
  count: 1137
  firstTimestamp: "2019-02-21T21:59:44Z"
  message: "Readiness probe failed: health checks failed: \nbonk \n"
  reason: Unhealthy

Logs:
container_linux.go:247: starting container process caused "exec: \"heyo\": executable file not found in $PATH"
```

1. unready
container info

kubectl diagnose

GO KUBECTL PLUGIN

```
melanie_cebula@melanie-cebula ~/a/r/k-diagnose> kubectl diagnose --namespace=bonk-staging
There are 3 pods in the namespace bonk-staging
Pod bonk-staging-64f6f7bb7c-4dtvs in namespace bonk-staging is healthy
Pod bonk-staging-64f6f7bb7c-rzhwr in namespace bonk-staging is healthy
Container bonk is unready with restart
LastState:
  exit code: 127
  signal: 0
  message: oci runtime error: container_linux.go:247: starting container process caused "exec: \"heyo\": executable file not found in $PATH"

  reason: ContainerCannotRun
State:
  message: Back-off 5m0s restarting failed container=bonk pod=bonk-staging-c5946c7d-28k5z_bonk-staging(e2a319bf-3623-11e9-95cd-1229ad3a733c)
  reason: CrashLoopBackOff
Container mini-announcer is unready with restart
Pod bonk-staging-c5946c7d-28k5z in namespace bonk-staging is unhealthy
Events:
  count: 664
  firstTimestamp: "2019-02-21T21:59:32Z"
  message: Back-off restarting failed container
  reason: BackOff
  count: 1137
  firstTimestamp: "2019-02-21T21:59:44Z"
  message: "Readiness probe failed: health checks failed: \nbonk \n"
  reason: Unhealthy

Logs:
container_linux.go:247: starting container process caused "exec: \"heyo\": executable file not found in $PATH"
```

1. unready
container info

2. kubectl
podevents

kubectl diagnose

GO KUBECTL PLUGIN

```
melanie_cebula@melanie-cebula ~/a/r/k-diagnose> kubectl diagnose --namespace=bonk-staging
There are 3 pods in the namespace bonk-staging
Pod bonk-staging-64f6f7bb7c-4dtvs in namespace bonk-staging is healthy
Pod bonk-staging-64f6f7bb7c-rzhwr in namespace bonk-staging is healthy
Container bonk is unready with restart
LastState:
  exit code: 127
  signal: 0
  message: oci runtime error: container_linux.go:247: starting container process caused "exec: \"heyo\": executable file not found in $PATH"
  reason: ContainerCannotRun
State:
  message: Back-off 5m0s restarting failed container=bonk pod=bonk-staging-c5946c7d-28k5z_bonk-staging(e2a319bf-3623-11e9-95cd-1229ad3a733c)
  reason: CrashLoopBackOff
Container mini-announcer is unready with restart
Pod bonk-staging-c5946c7d-28k5z in namespace bonk-staging is healthy
Events:
  count: 664
  firstTimestamp: "2019-02-21T21:59:32Z"
  message: Back-off restarting failed container=mini-announcer pod=bonk-staging-c5946c7d-28k5z_bonk-staging(e2a319bf-3623-11e9-95cd-1229ad3a733c)
  reason: BackOff
  count: 1137
  firstTimestamp: "2019-02-21T21:59:44Z"
  message: "Readiness probe failed: heyo\n"
  reason: Unhealthy

Logs:
container_linux.go:247: starting container process caused "exec: \"heyo\": executable file not found in $PATH"
```

1. unready
container info

2. kubectl
podevents

3. pod logs for
unready containers

Takeaways


- Create an opinionated kubectl wrapper
- Automate common k8s workflows with kubectl plugins


CI/CD

Log the changed files of a PR

Build #404 | julia--log-changed-files | 44318ff

Passed in 3m 51s



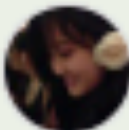
Validations (DEPLOY_PIPELI...

RSpec [ci_required]

RuboCop [ci_required]

Build (PROJECT_NAME=sta...



Build Docs (PROJECT_NAM...

Julia Wang

Created Wednesday at 11:09 AM

Triggered from Webhook



Rebuild

jorb_dispatcher_buildkite --version && jorb_dispatcher_buildkite _infra/ci/dispatch.yml --gl...

Ran in 36s

Waited 9s

i-04e77746cbe296415-75


Validations (DEPLOY_PIPELINES_VALIDATION_VERSION=0.0.4) [ci_required]

run_jorb \{"name\":"Vali...

Ran in 34s

Waited 5s

i-0476de158cb963088-76


RSpec [ci_required]

run_jorb \{"name\":"RSpec\ \[ci_required\]\",\"path\":"_infra/ci/jobs/rspe...

Ran in 41s

Waited 8s

i-0c24219eccd39024a-70


RuboCop [ci_required]

run_jorb \{"name\":"RuboCop\ \[ci_required\]\",\"path\":"_infra/ci/jobs/...

Ran in 37s

Waited 1s

i-0e21324366ac6165e-69


Build (PROJECT_NAME=stamp-collector) [ci_required]

run_jorb \{"name\":"Build\ \[PROJECT_NAME...

Ran in 3m 28s

Waited 3s

i-00ce2dc8ef1db6820-69

Build Docs (PROJECT_NAME=stamp-collector) [ci_required]

run_jorb \{"name\":"Build\ Docs\ \[PROJ...

Ran in 36s

Waited 8s

i-0c24219eccd39024a-69

✓ Build (PROJECT_NAME=stamp-collector) [ci_required] run_jorb \{"name\":"Build\ \ (PROJECT_NAME... Ran in 3m 28s

≡ Log

📄 Artifacts

📦 Agent

⚙️ Environment

+ Expand groups - Collapse groups

```
1 ▶ Running global environment hook
5 ▶ Running global pre-checkout hook
11 ▶ Preparing working directory
14 ▶ Running global checkout
18 ▶ Running global post-checkout
24 ▶ Running commands
42 ▶ Running 'install kube-
106 ▶ Running 'k generate'
160 ▶ Running 'k build'
1003 ▶ Running 'create titanic build info in ./config'
1004 ▶ Running 'set build tags'
1008 ▶ Running 'tar artifact (project only)'
1023 ▶ Running 'upload titanic artifact'
1036 ▶ Running 'k clean'
1053 ▶ Running global pre-exit hook
```

Each step in our CI /CD jobs are
RUN steps in a build Dockerfile

✓ Build (PROJECT_NAME=stamp-collector) [ci_required] run_jorb \{"name\":"Build\ \ (PROJECT_NAME... Ran in 3m 28s

≡ Log

📄 Artifacts

📦 Agent

⚙️ Environment

+ Expand groups - Collapse groups

```
1 ▶ Running global environment hook
5 ▶ Running global pre-checkout hook
11 ▶ Preparing working directory
14 ▶ Running global checkout hook
18 ▶ Running global post-checkout hook
24 ▶ Running commands
42 ▶ Running 'install kube-gen (
100 ▶ Running 'k generate'
160 ▶ Running 'k build'
1003 ▶ Running 'create titanic bui
1004 ▶ Running 'set build tags'
1008 ▶ Running 'tar artifact (project only)'
1023 ▶ Running 'upload titanic artifact'
1036 ▶ Running 'k clean'
1053 ▶ Running global pre-exit hook
```

runs k commands

DEPLOY PROCESS

A single deploy process for every change

Develop

Write code and config
under your project

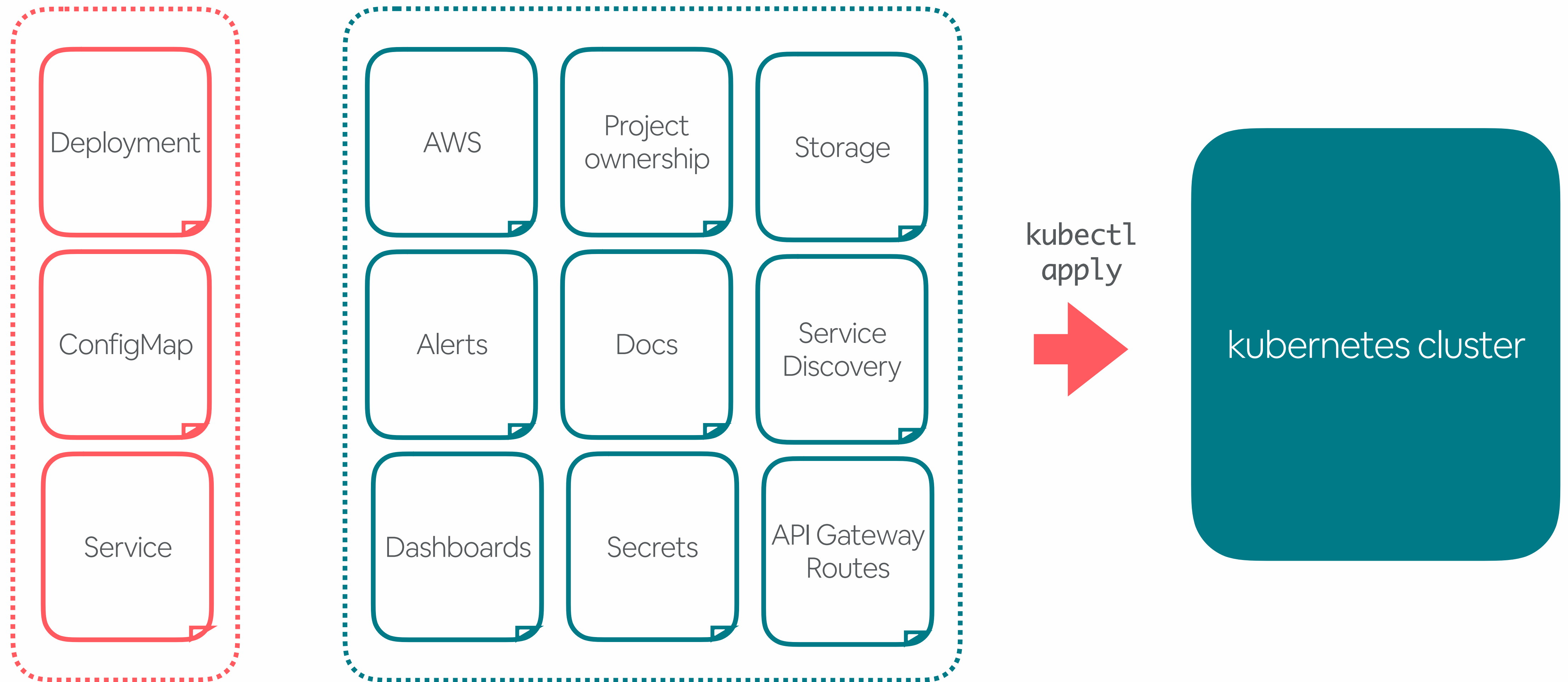
Merge

Open a PR and merge
your code to master

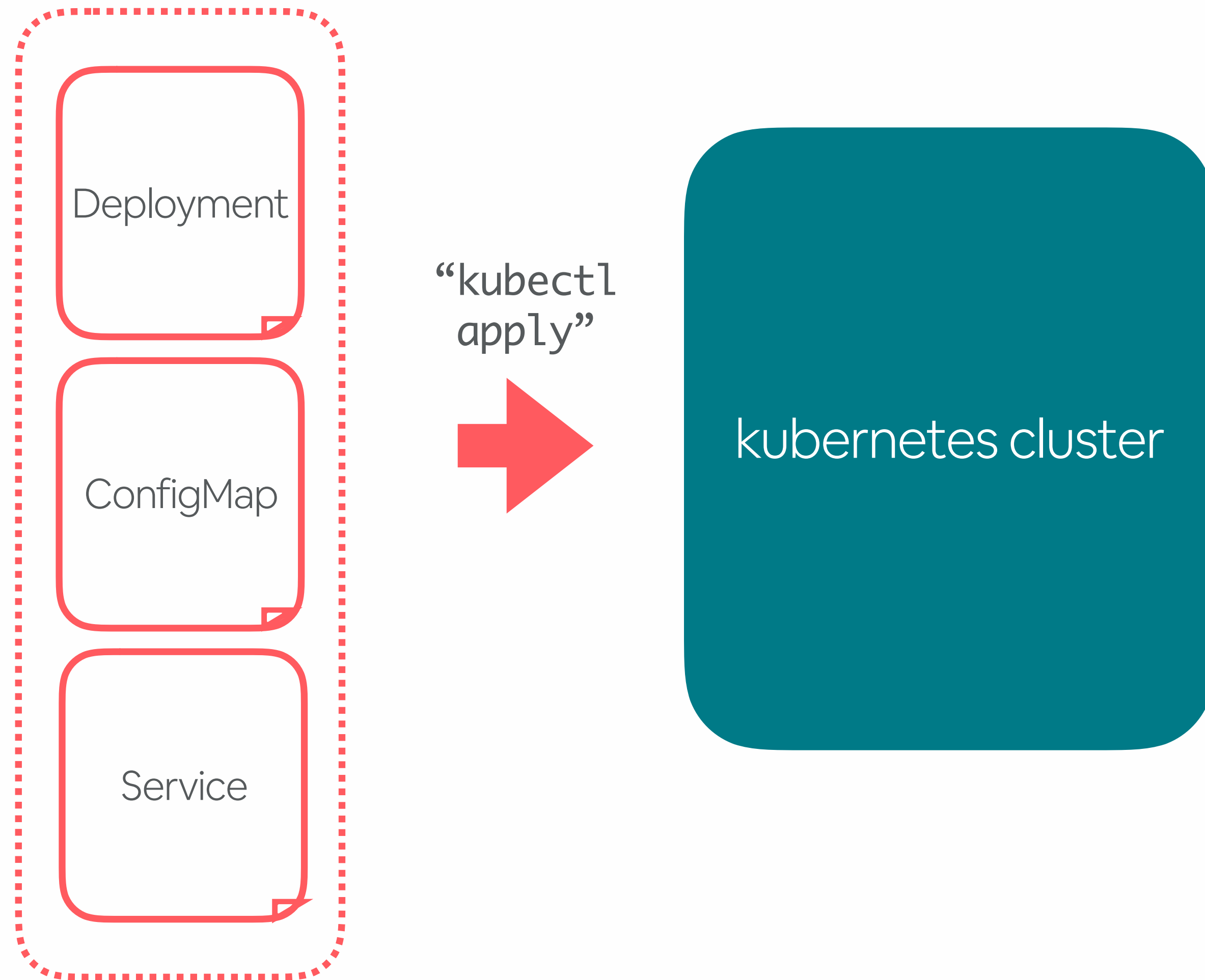
Deploy

Deploy all code and
config changes

A single deploy process for every change



How do we apply k8s configuration?



- kubectl apply all files
- in some cases where apply fails, replace files without force
- always restart pods on deploy to pick up changes
- return atomic success or failure state by sleeping and checking status

How do you always restart pods on deploy?

We add a date label to the pod spec, which convinces k8s to relaunch all pods

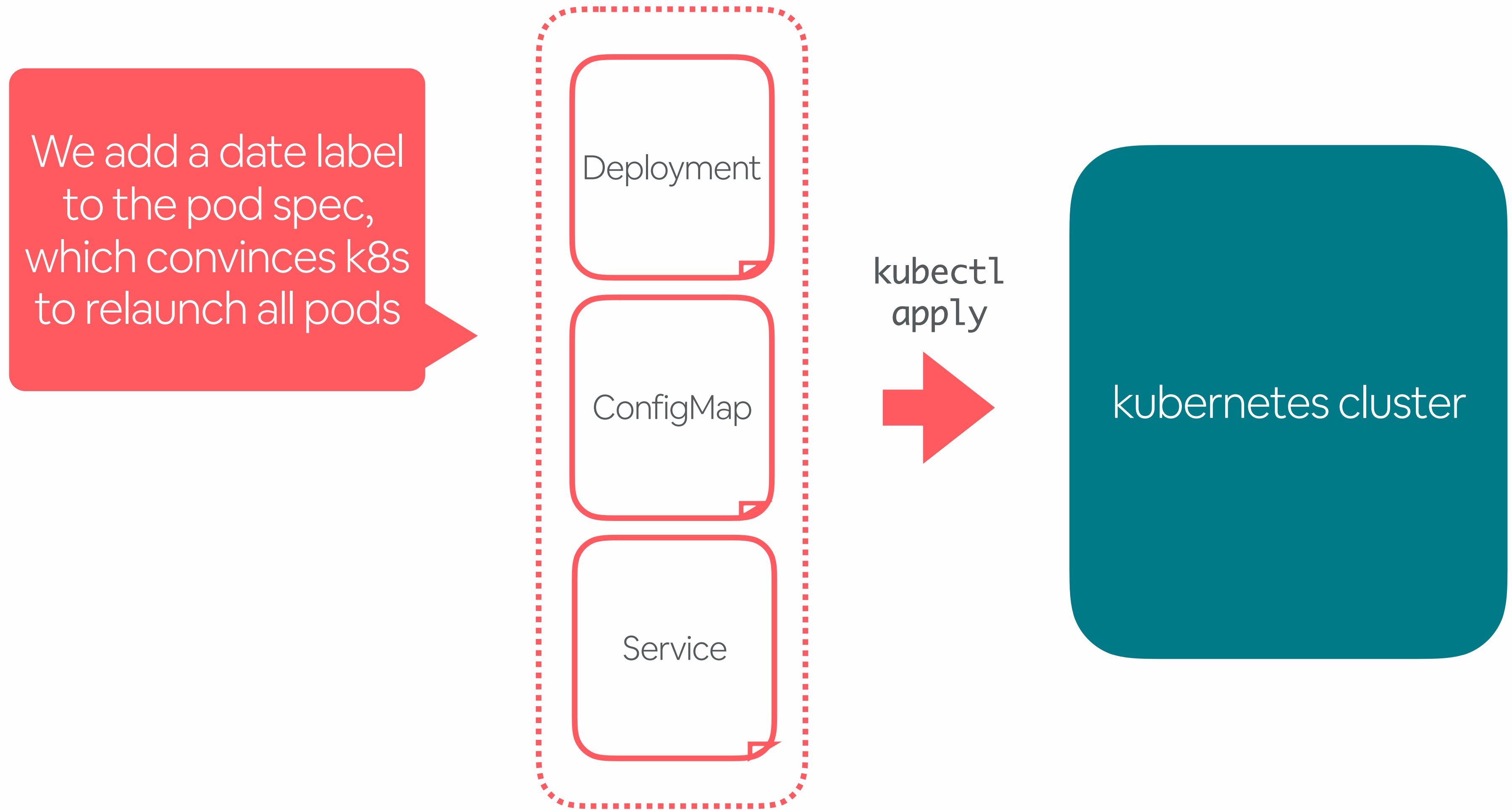
Deployment

ConfigMap

Service

kubectl
apply

kubernetes cluster



How do we apply custom configuration?

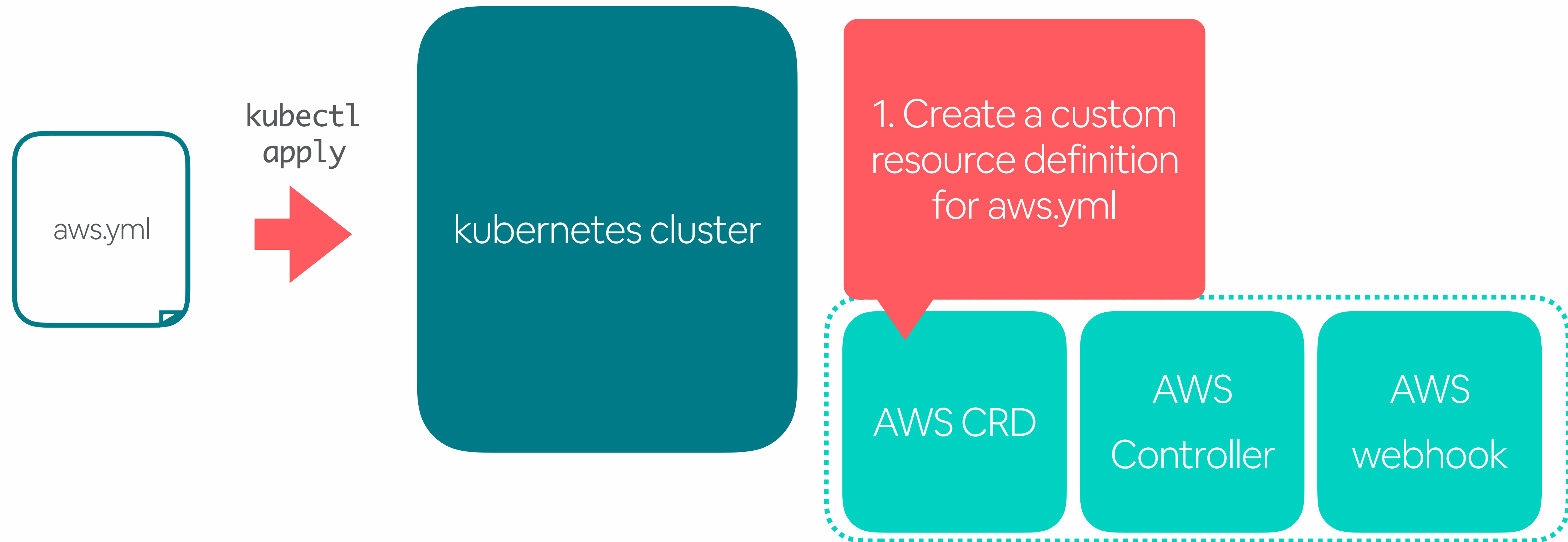
Custom controllers

On their own, custom resources simply let you store and retrieve structured data. When you combine a custom resource with a *custom controller*, custom resources provide a true *declarative API*.

How do we apply custom configuration?



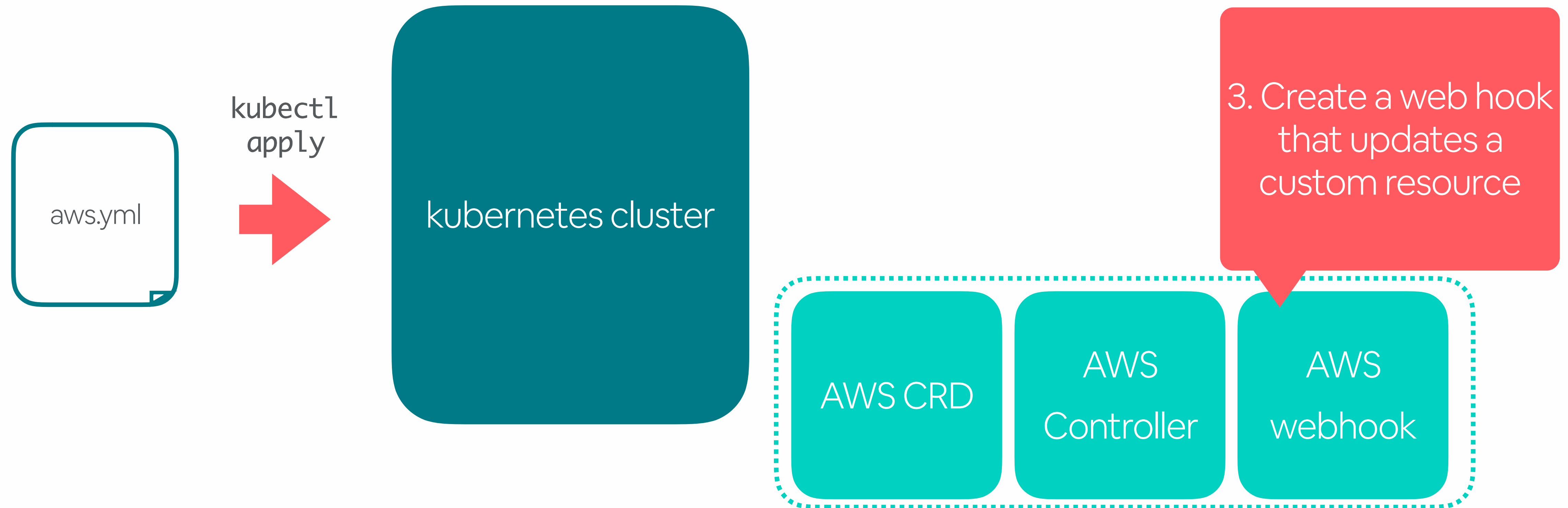
How do we apply custom configuration?



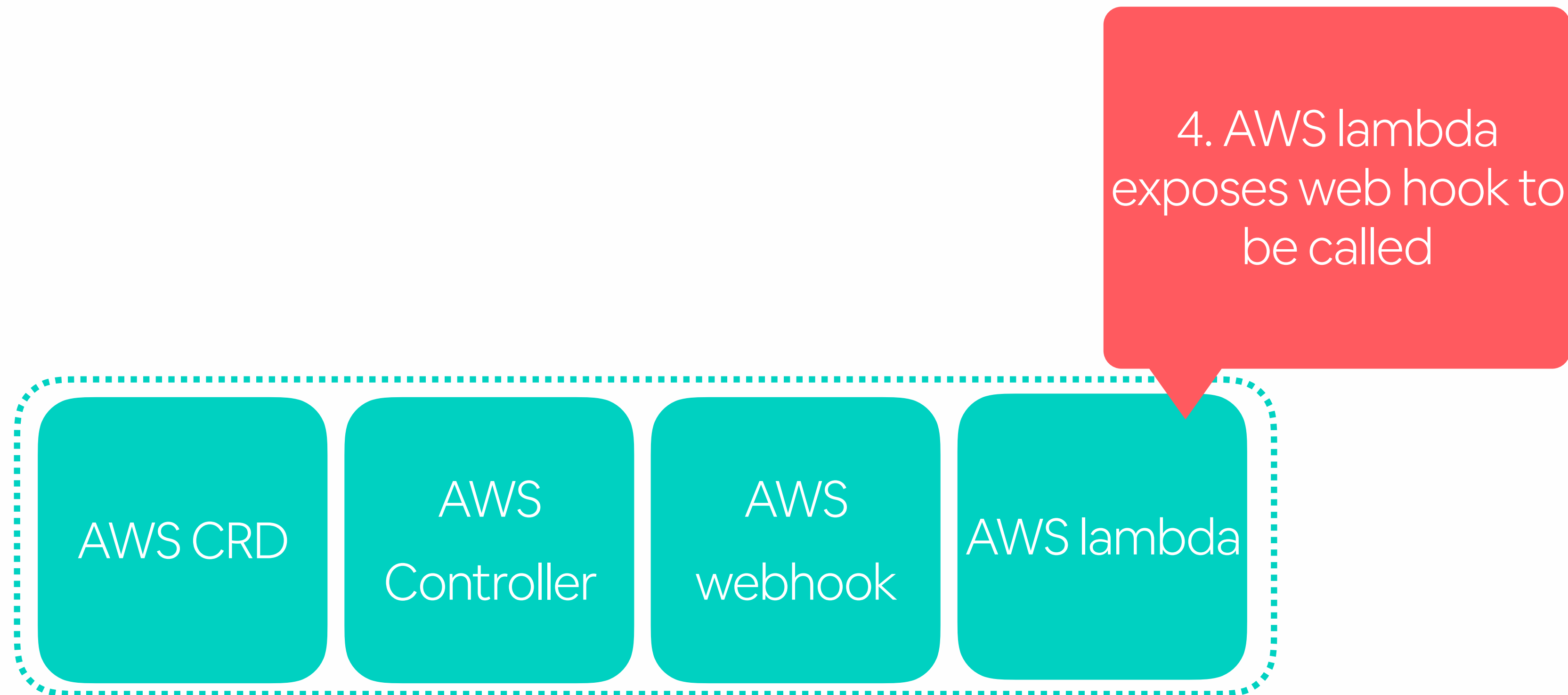
How do we apply custom configuration?



How do we apply custom configuration?



How do we apply custom configuration?



Takeaways

- Code and configuration should be deployed with the same process
- Use custom resources and custom controllers to integrate k8s with your infra

VALIDATION

Configuration

SHOULD BE VALIDATED

```
/Users/melanie_cebula/bonk/
└─ _infra/
   ├── ci/
   ├── docs/
   ├── keys/
   ├── kube/
   ├── secrets/
   ├── airtlab.yml
   ├── aws.yml
   ├── deployboard.yml
   ├── dyno.yml
   └── project.yml
└─ app/
   ├── bin/
   ├── config/
   ├── db/
   ├── lib/
   ├── log/
   ├── public/
   ├── spec/
   ├── tmp/
   └── vendor/
      ├── config.ru
      ├── Gemfile
      ├── Gemfile.lock
      ├── Rakefile
      ├── README.md
      └── unicorn.rb
```

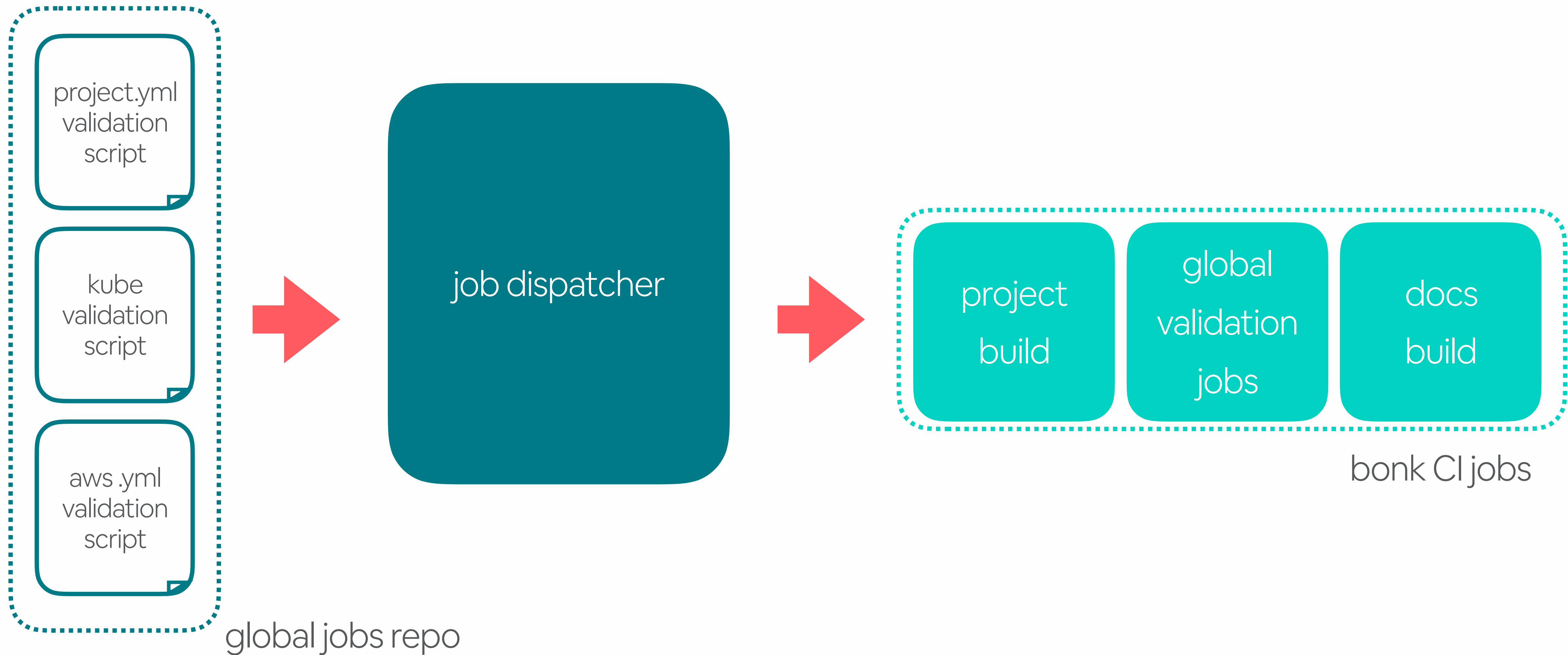
@MELANIECEBULA

- *enforce* best practices
- at build time with validation scripts
- at deploy time with admission controller

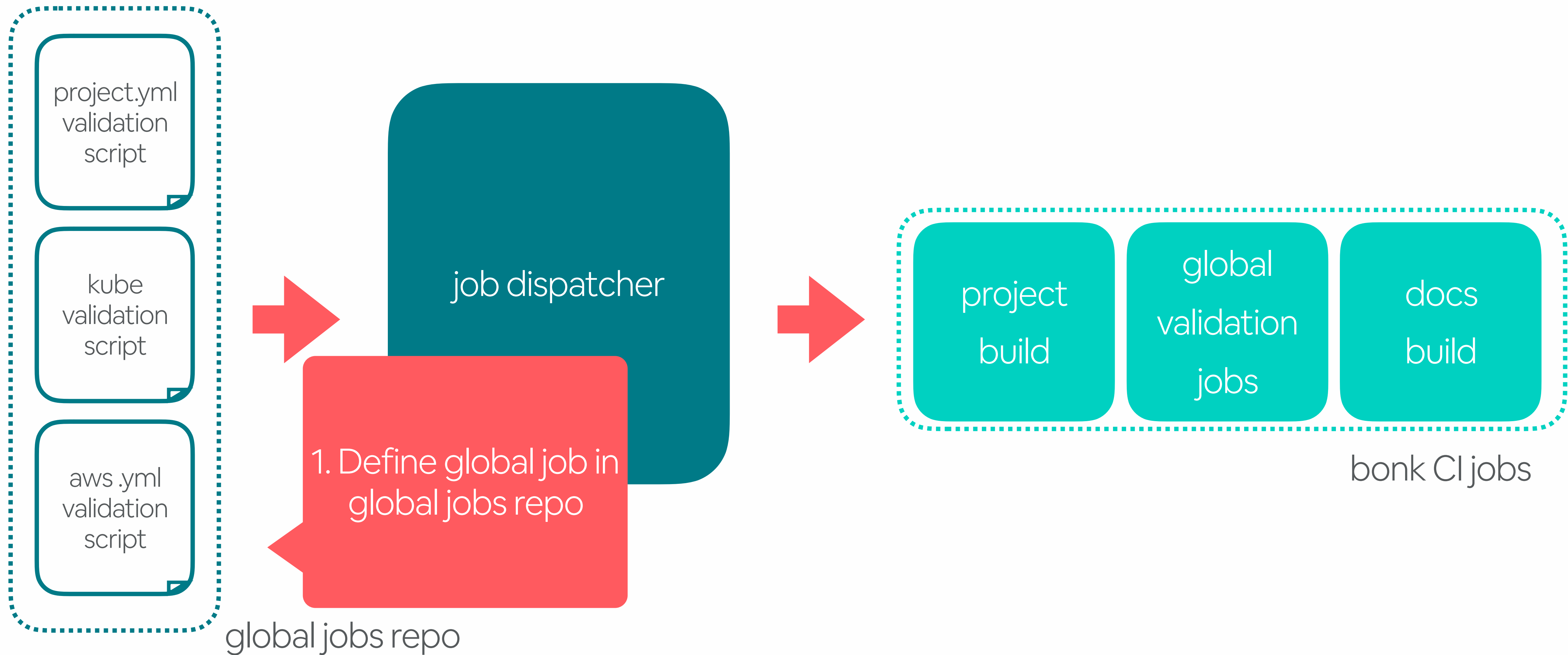
How do we validate configuration at build time?

```
** Invoke scry:validate_project_yaml (first_time)
** Execute scry:validate_project_yaml
I, [2019-02-26T22:47:53.849796 #21991]  INFO -- : Validating project.yml...
I, [2019-02-26T22:47:54.020703 #21991]  INFO -- : Loading project.yml from: /tmp/d20190226-21664-
18tbatz/bonk/projects/bonk/_infra/project.yml
I, [2019-02-26T22:47:54.021592 #21991]  INFO -- : Loaded {"name"=>"bonk", "description"=>"tutorial project",
"slack"=>"onetouch-bootcamp", "teams"=>["Infra - Production Platform - Developer Productivity - Service
Orchestration"], "additional_owners"=>["bruce_sherrod", "melanie_cebula"]}
I, [2019-02-26T22:47:54.021795 #21991]  INFO -- : Done!
Completed Successfully!
```

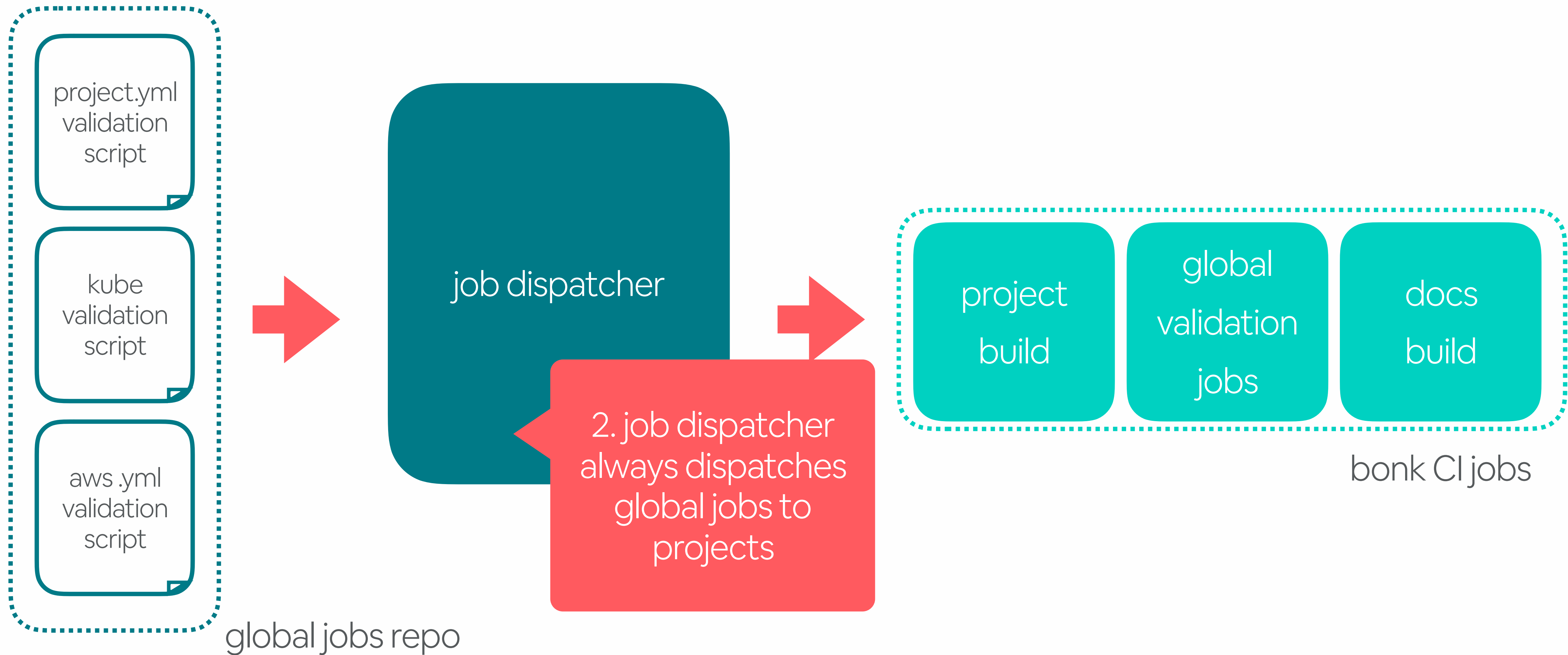
How do we validate configuration at build time?



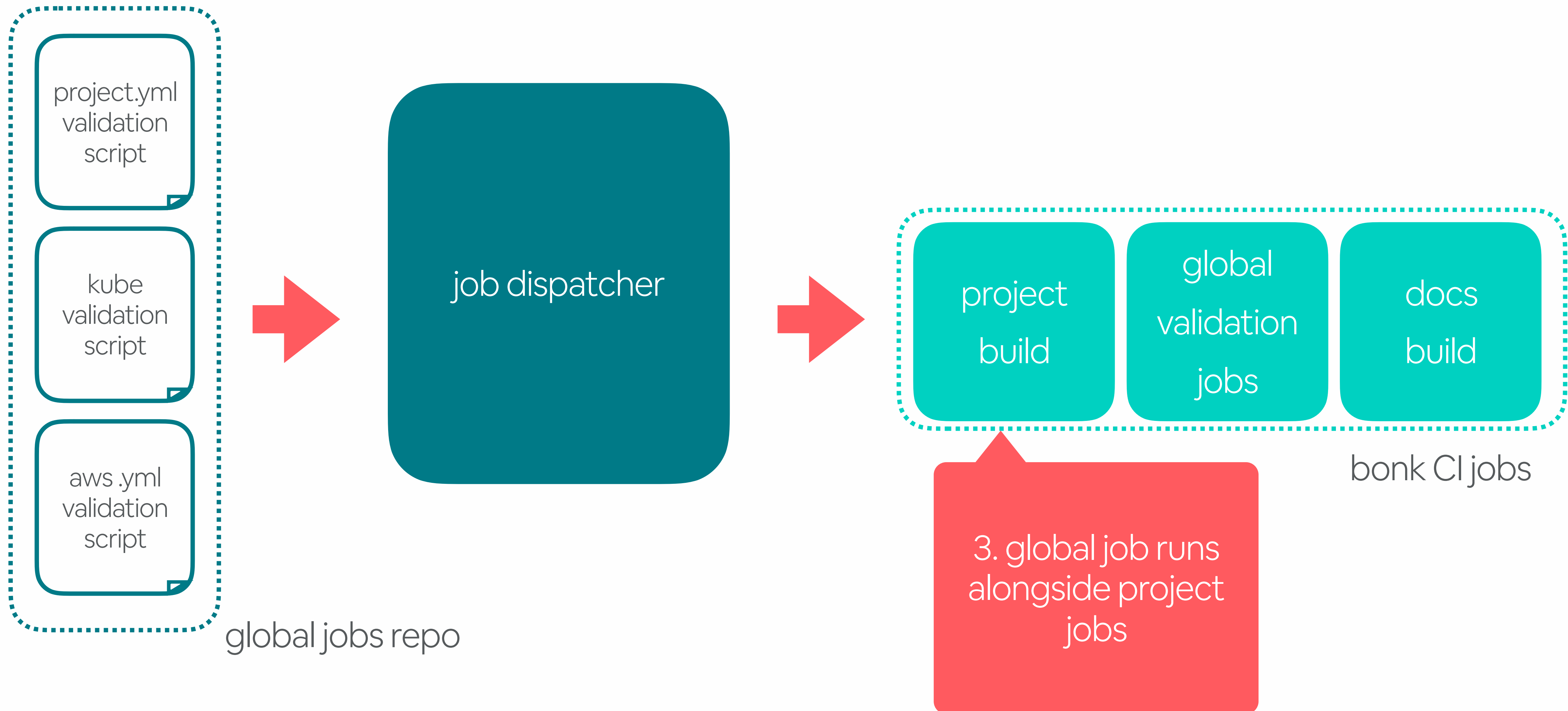
How do we validate configuration at build time?



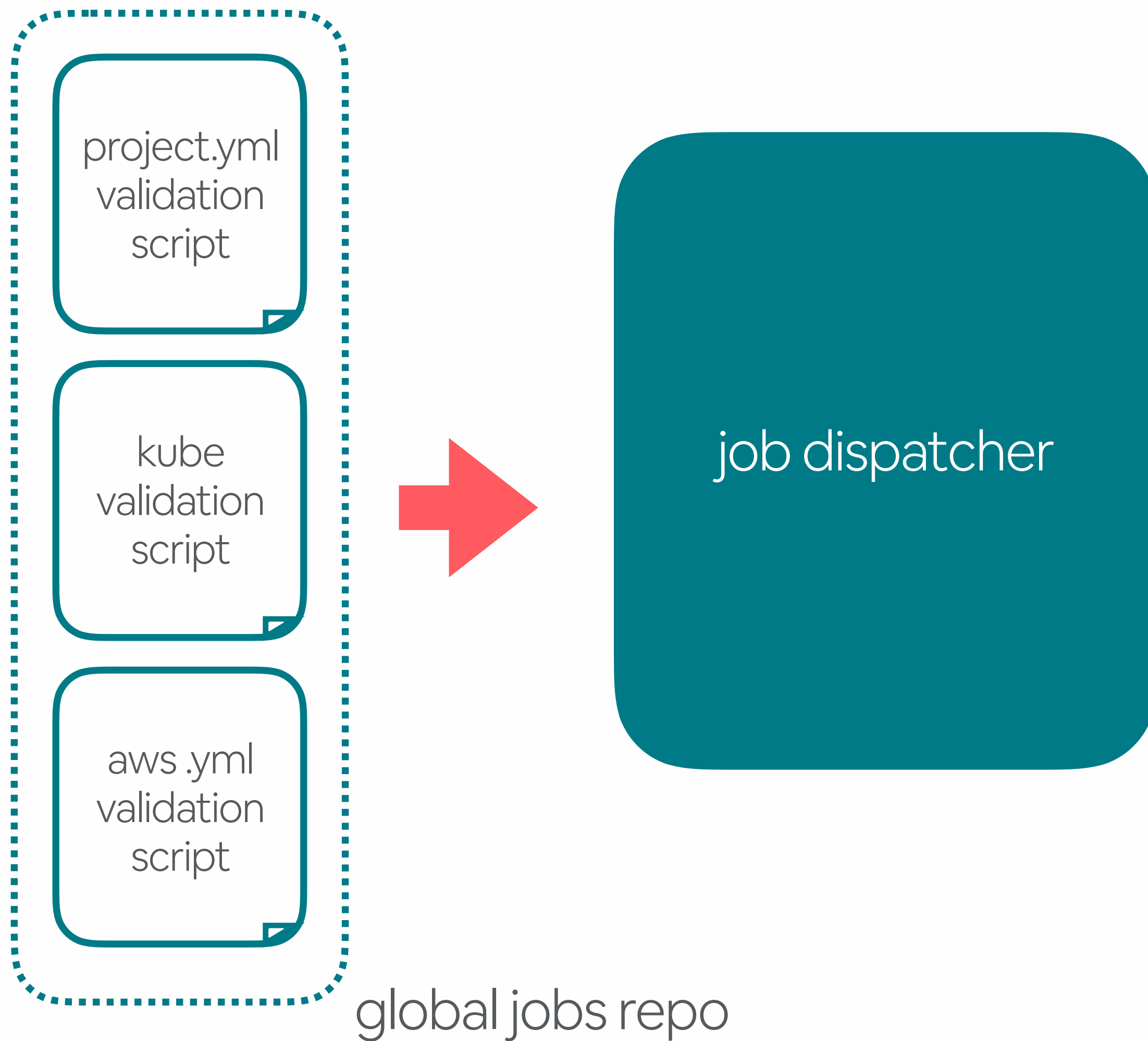
How do we validate configuration at build time?



How do we validate configuration at build time?

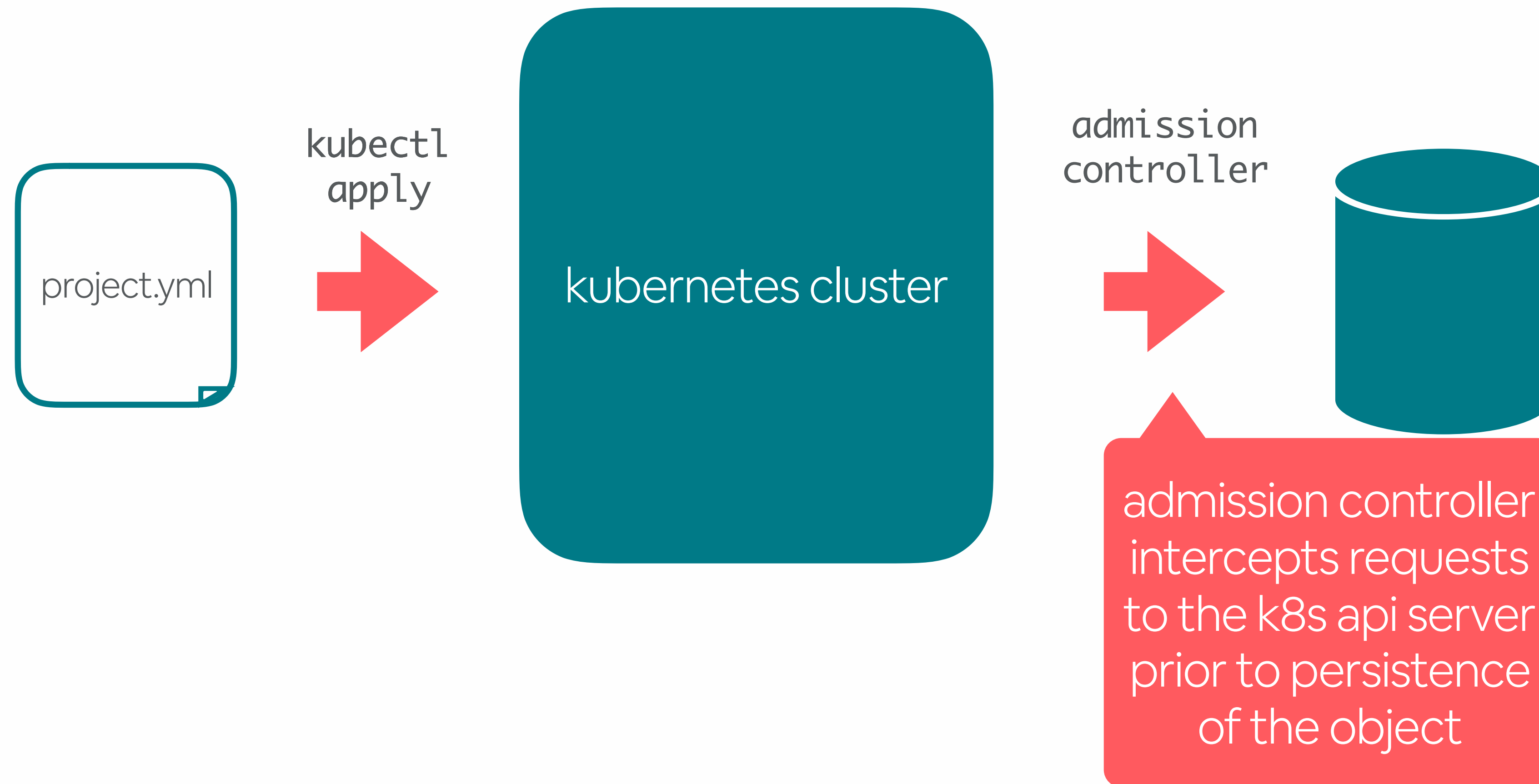


What do we validate at build time?

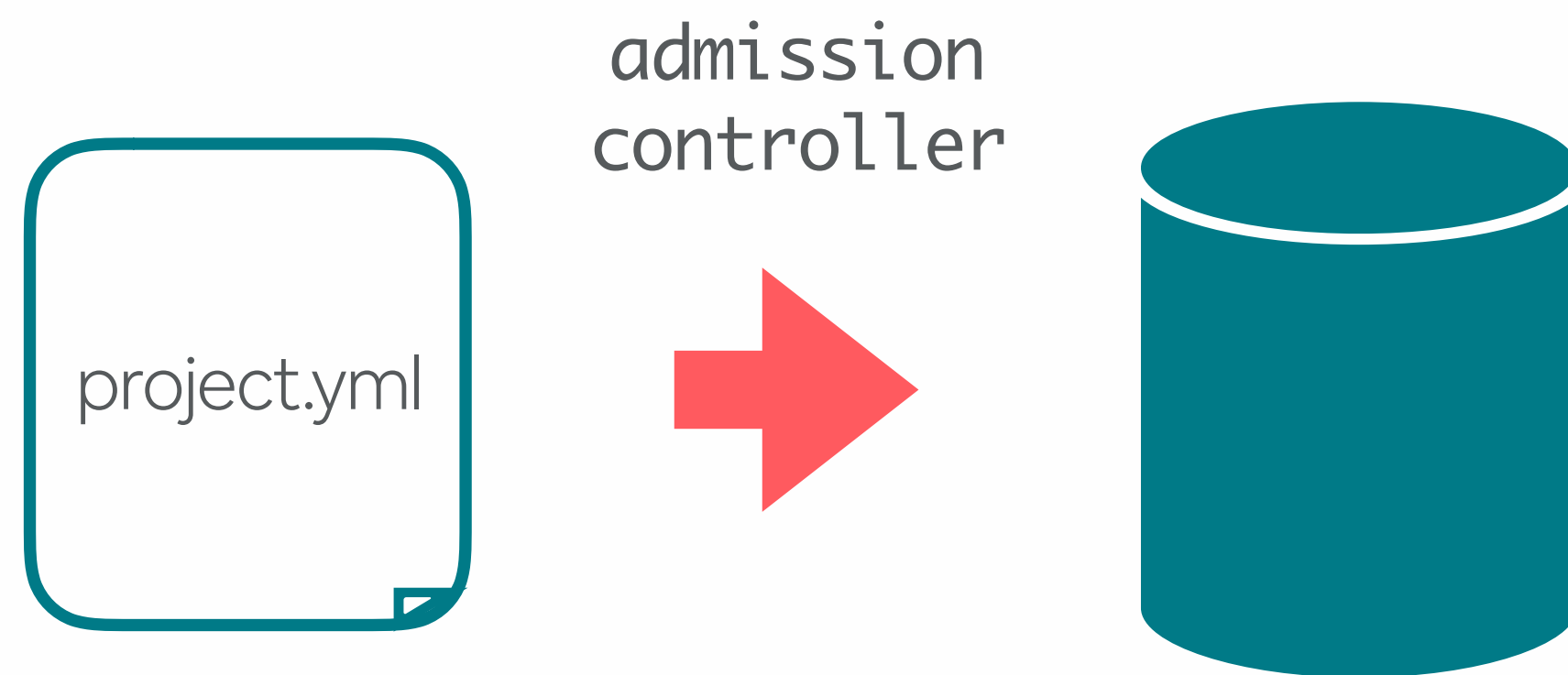


- invalid yaml
- invalid k8s configuration
- bad configuration versions
- max namespace length (63 chars)
- valid project name
- valid team owner in project.yml

How do we validate configuration at deploy time?

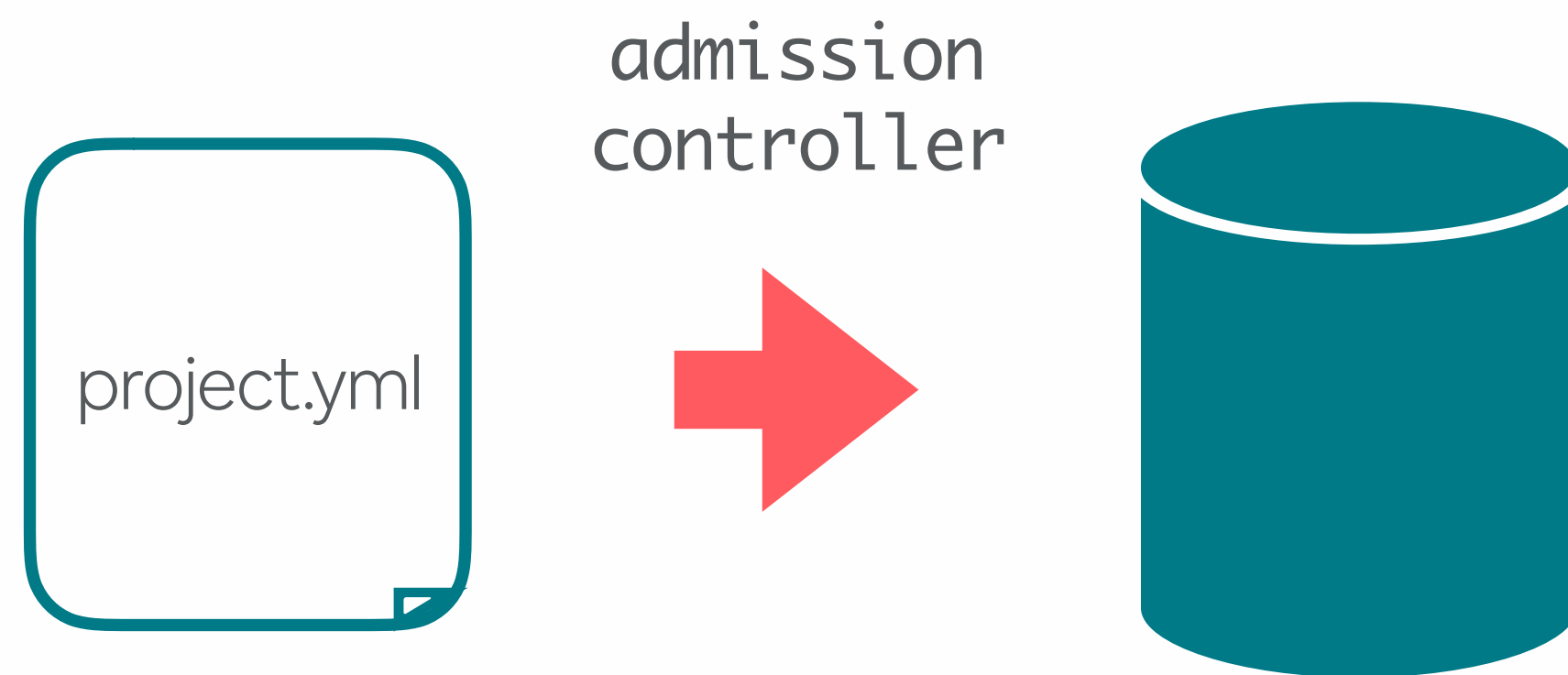


How do we validate configuration at deploy time?



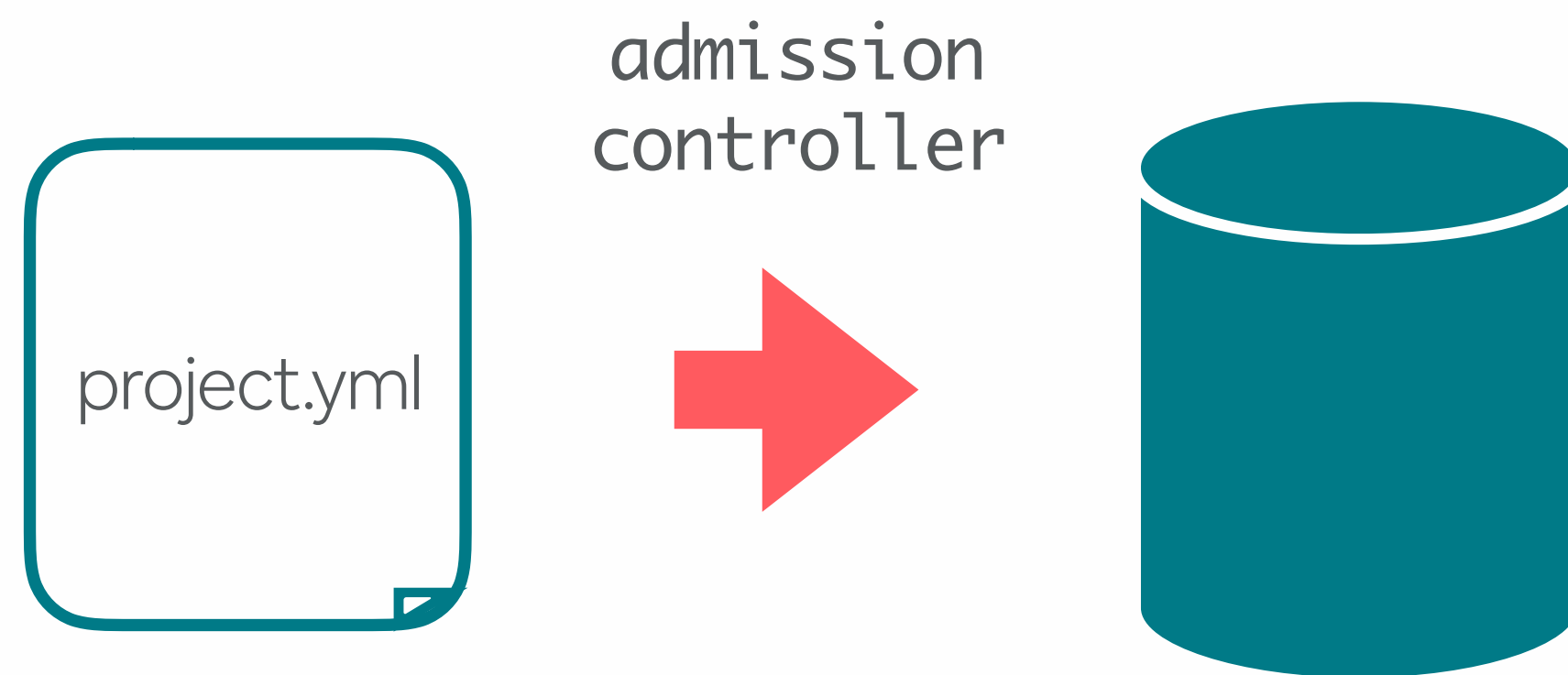
- metadata is encoded as annotations at generate time
- **admission controller** checks for required annotations
- reject any update to resources that are missing required annotations

What do we validate with admission controller?



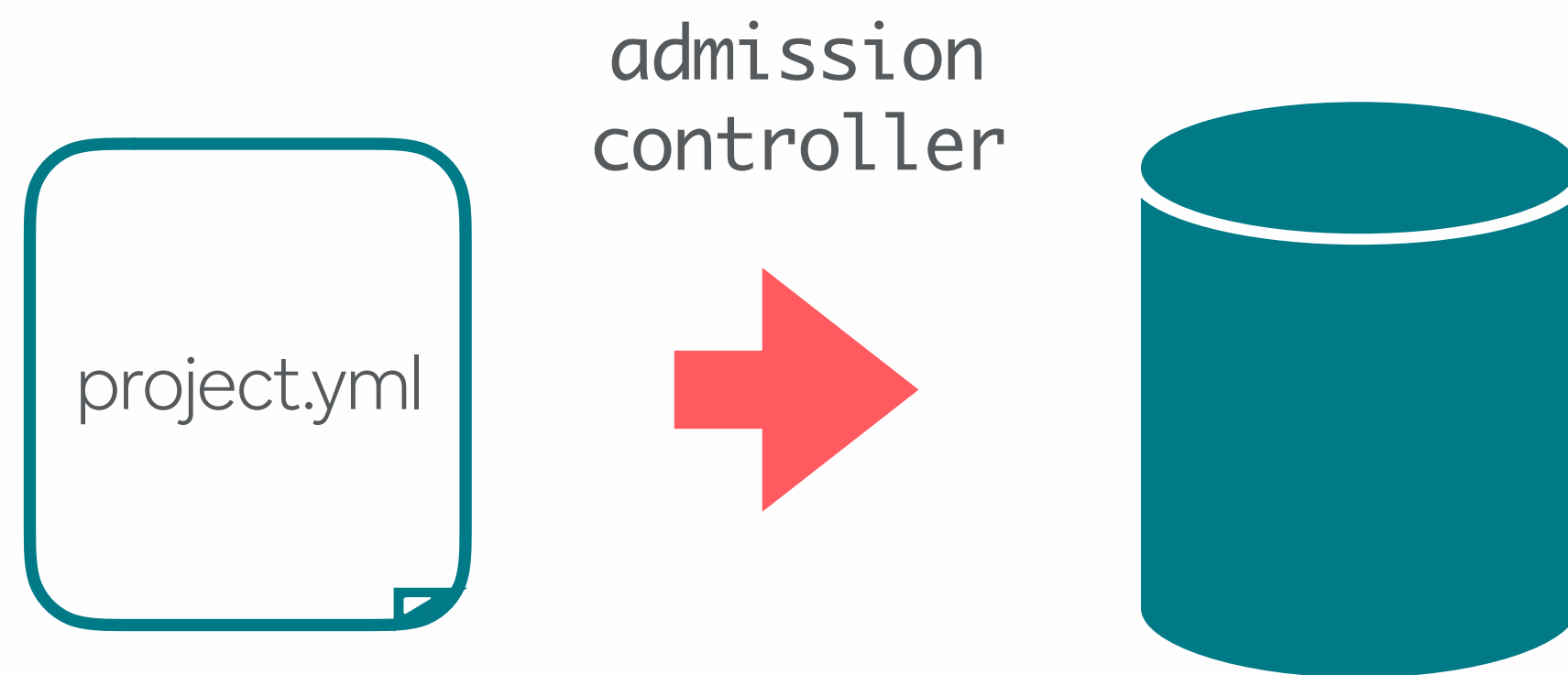
- project ownership annotations
- configuration stored in git
- configuration uses minimally supported version

What do we validate with admission controller?



- production images must be uploaded to production ECR
- prevent deployment of unsafe workloads
- prevent deployment of development namespaces to production clusters

What do we validate with admission controller?



- production images must be uploaded to production ECR
- prevent deployment of unsafe workloads
- prevent deployment of development namespaces to production clusters

standardized
namespaces!

Takeaways

- CI/CD should run the same commands that engineers run locally
- CI/CD should run in a container
- Validate configuration as part of CI/CD

10 Takeaways

1. **Abstract away** complex kubernetes configuration
2. **Standardize** on environments and namespaces
3. Everything about a service should be in **one place in git**
4. **Make best practices the default** by generating configuration
5. Configuration should be **versioned** and **refactored automatically**.
6. **Create an opinionated kubectl wrapper** that automates common workflows
7. **CI/CD should run the same commands** that engineers run locally, in a container
8. Code and configuration should be **deployed with the same process**
9. **Use custom resources and custom controllers** to integrate with your infrastructure
10. **Validate configuration** as part of CI/CD

2019 Challenges

- thousands of services running in k8s
- moving *all* configuration to gitops workflow w/ custom controllers
- scaling the cluster / scaling etcd / multi cluster support
- stateful services / high memory requirements
- tighter integration with kubectl plugins
- ... and more!

