@MELANIECEBULA / SEPT 2018 / ONS EUROPE

Services for All

How to Empower Engineers (with Kubernetes)

Outline

1. A brief history of configuration at Airbnb
2. Generating Kubernetes files
3. Creating a kubectl wrapper
4. Extending and customizing Kubernetes
5. Beyond Kubernetes

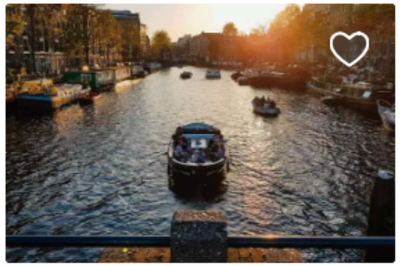
Start planning your trip


Homes


Experiences


Restaurants

Experiences near your home in Amsterdam



BOAT RIDE
Amsterdam Experience Cruise
$59 per person
4.89 ★★★★★ 836

FOOD WALK
The all Dutch food & history tour
$91 per person
4.91 ★★★★★ 196

PHOTO WALK
Capture the city & you on a photo walk
$35 per person
4.83 ★★★★★ 132

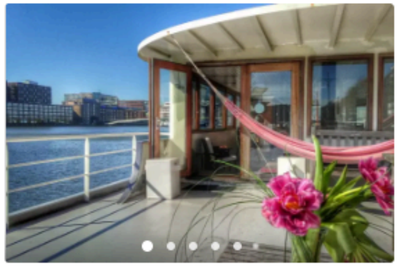
BOAT RIDE
Early morning- Canals all to ourselves
$42 per person
4.89 ★★★★★ 565

BOAT RIDE
Romantic Waters Boattour
$35 per person
4.87 ★★★★★ 231

BIKE RIDE
Jewish tour & visit to AnneFrank house
$43 per person
4.79 ★★★★★ 24

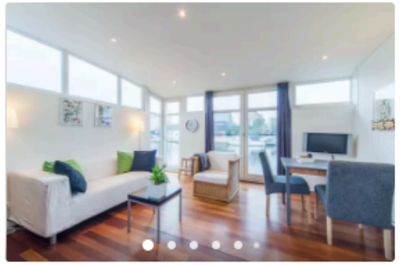
Show all (118) >

Where to stay

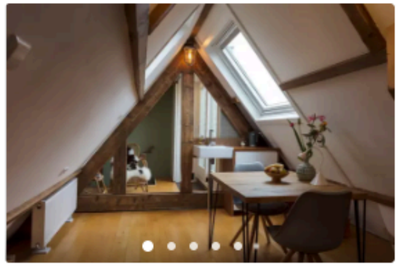


PRIVATE ROOM · 1 BED
LUXURY INDEPENDENT STUDIO on SHIP : free bikes!
$162 per night · Free cancellation
★★★★★ 334 · Superhost

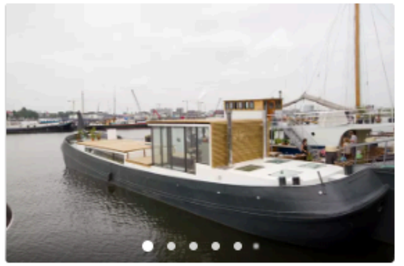
PRIVATE ROOM · 2 BEDS
Rebel - Private Room
$126 per night · Free cancellation
★★★★★ 551 · Superhost



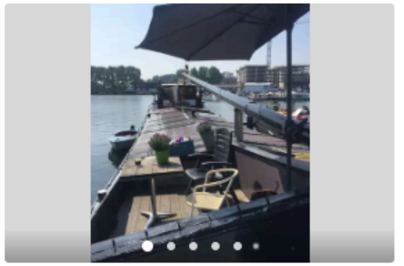
PRIVATE ROOM · 1 BED
Bed & Boat, apartment on houseboat. Free bikes.
$145 per night · Free cancellation
★★★★★ 328 · Superhost



PRIVATE ROOM · 1 BED
Private Attic Studio/Roofterrace
$90 per night · Free cancellation
★★★★★ 370 · Superhost



PRIVATE ROOM · 1 BED
Experience a houseboat in Amsterdam
$145 per night · Free cancellation
★★★★★ 247 · Superhost



PRIVATE ROOM · 1 BED
Authentic houseboat with privacy and comfort
$128 per night · Free cancellation
★★★★★ 373 · Superhost

Show all (2000+) >

What is airbnb?

AN ONLINE MARKETPLACE FOR SHARING HOMES AND EXPERIENCES

81k
cities

191+
countries

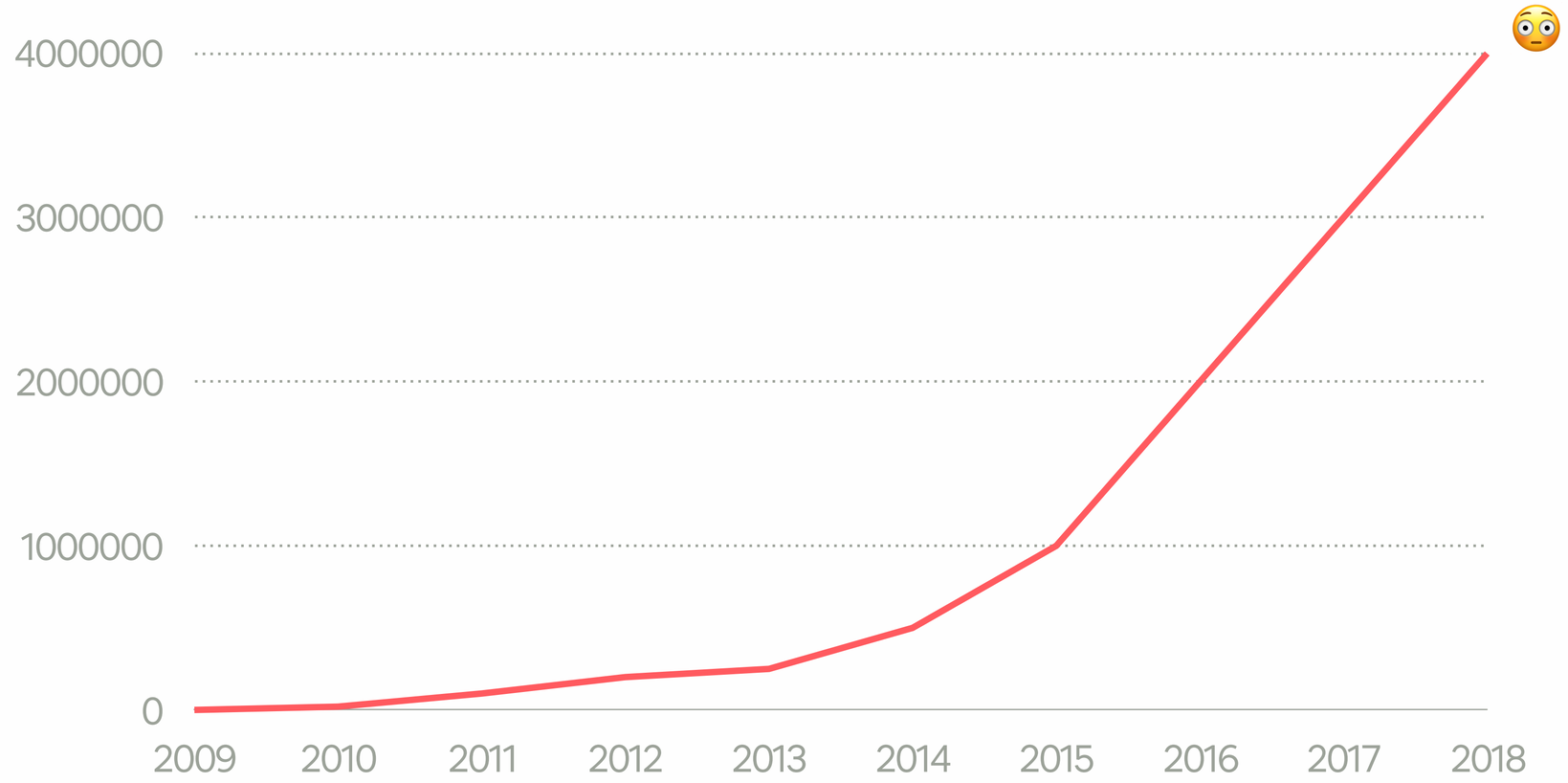
5 mil
homes


@MELANIECEBULA
Who am I?

A BRIEF HISTORY

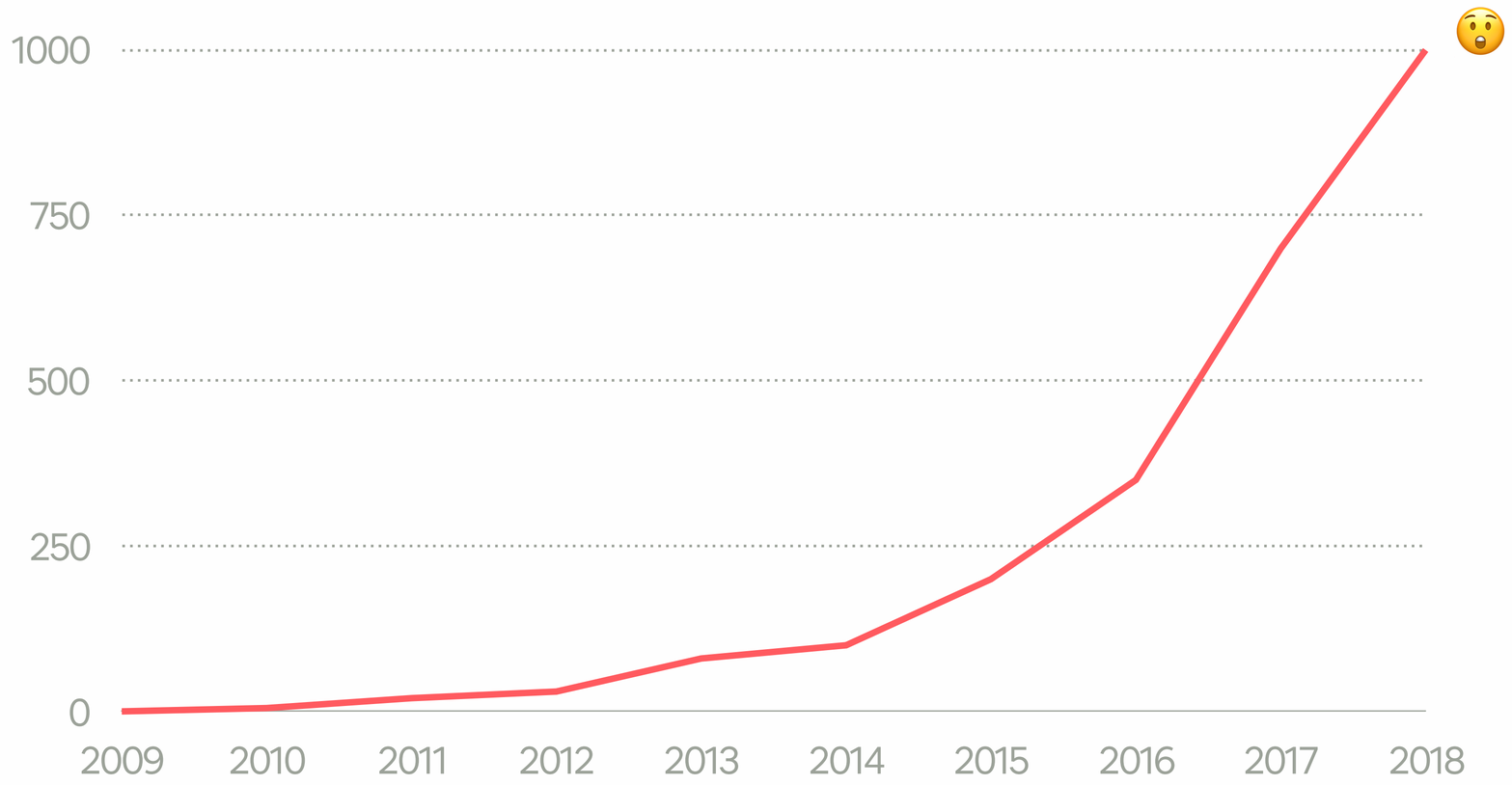
Why Microservices?

MONOLITH LOC



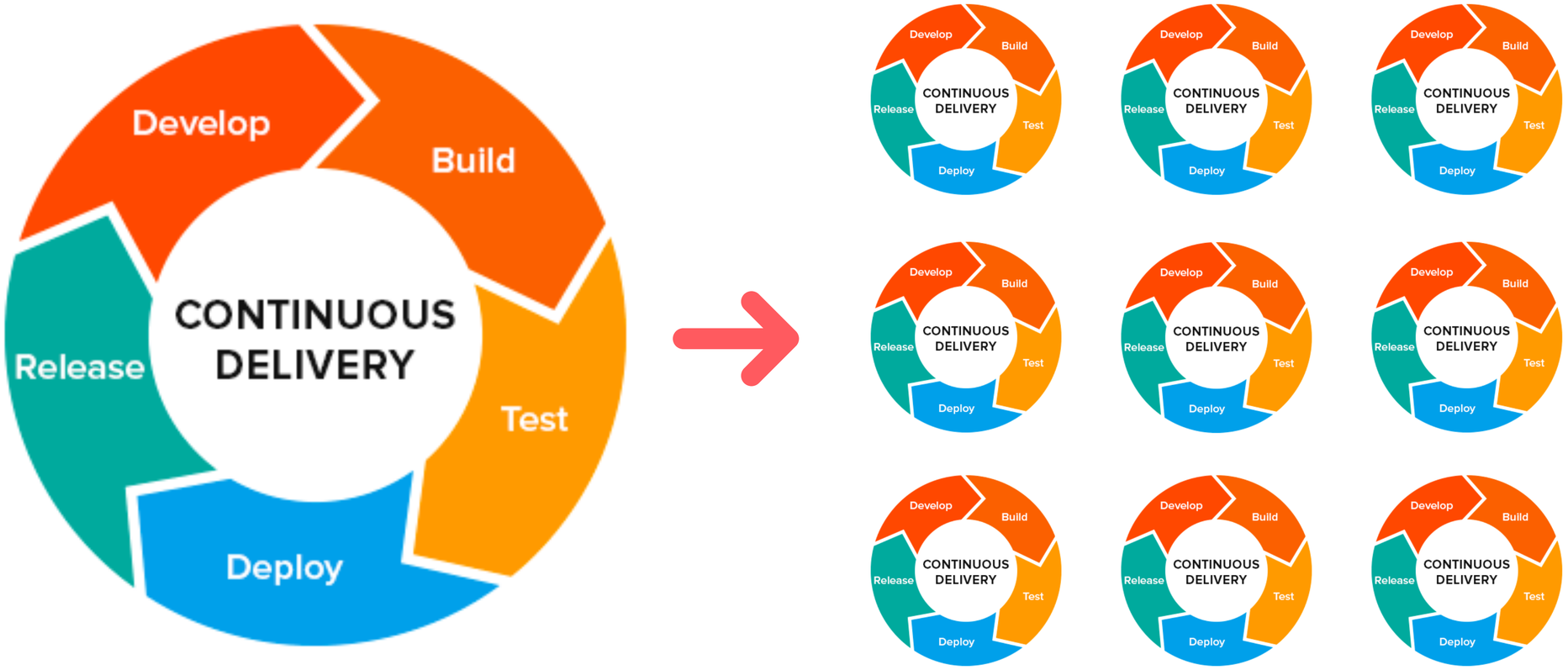
Why Microservices?

ENGINEERING TEAM



Why Microservices?

SCALING CONTINUOUS DELIVERY

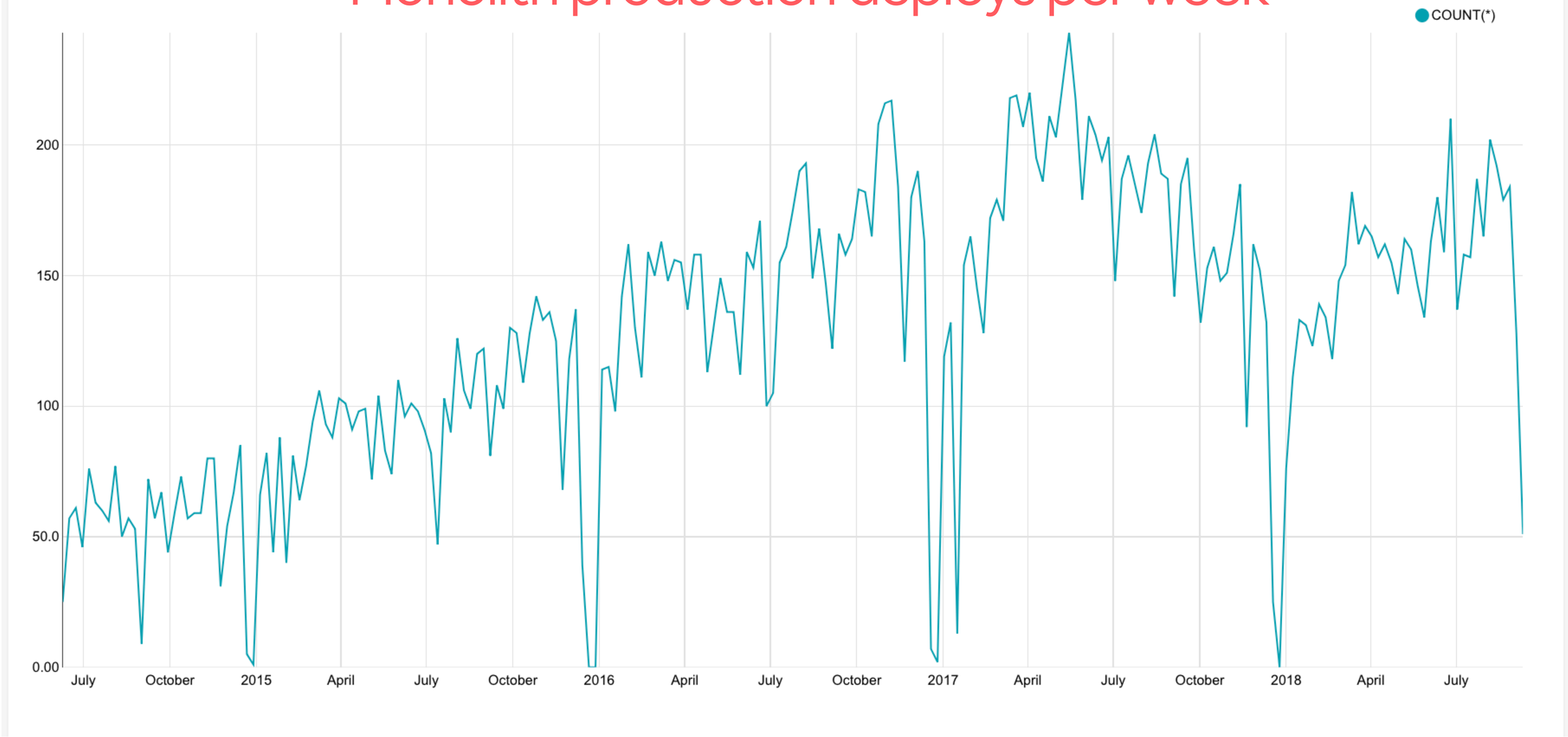


Why Microservices?

Deploys per week (all apps, all environments)

Why Microservices?

Monolith production deploys per week



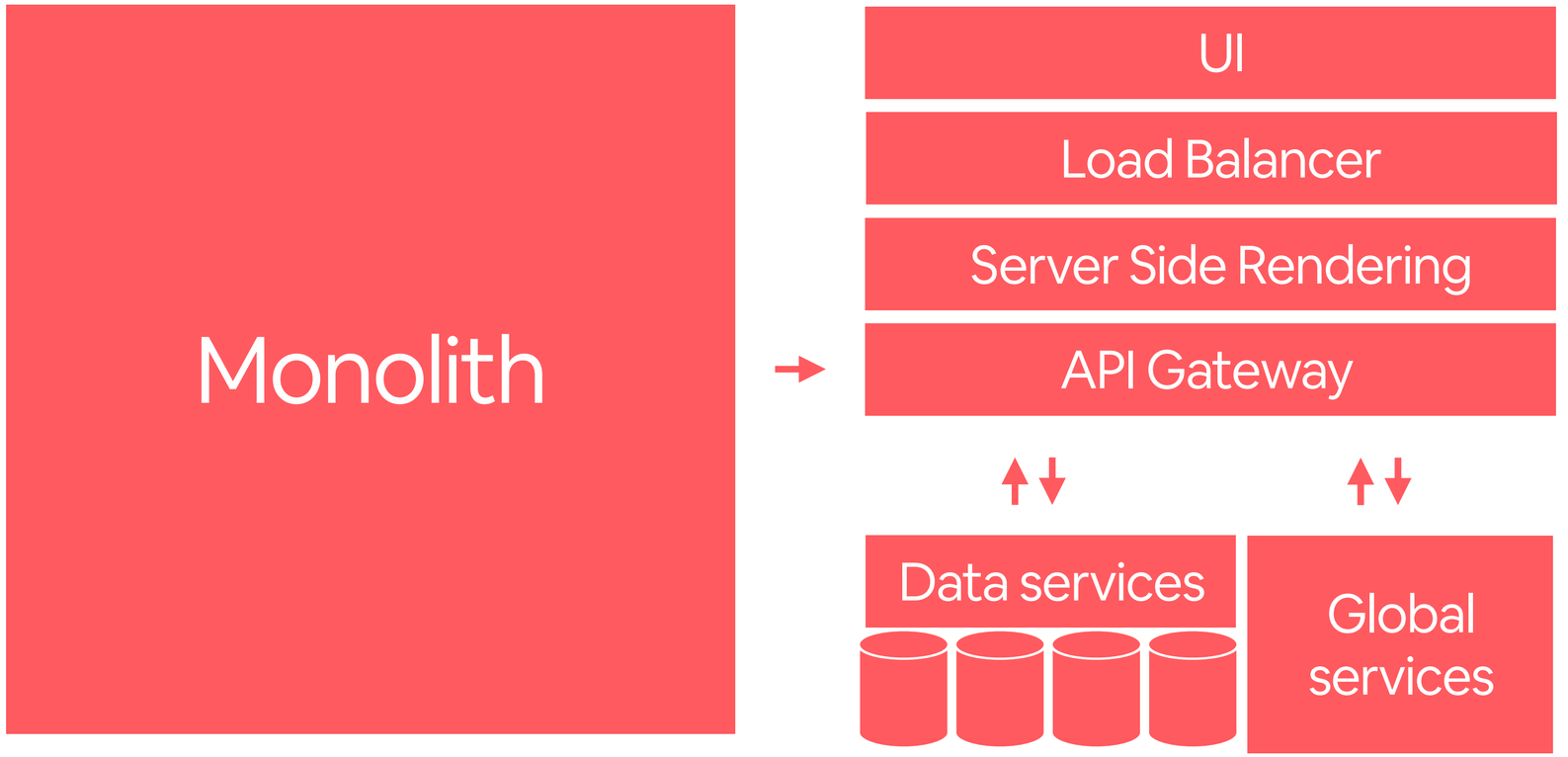
Why Microservices?

120,000

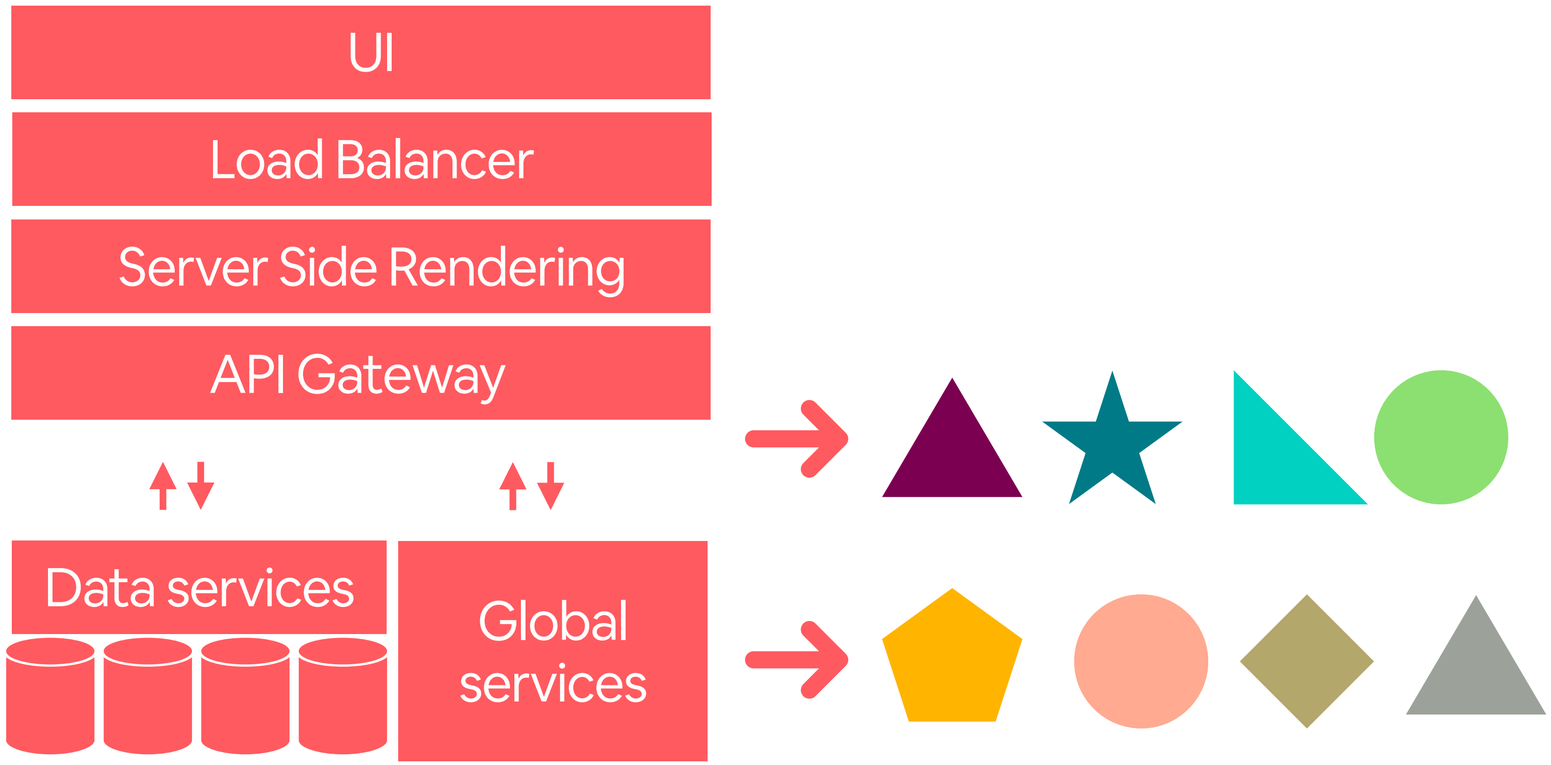
production deploys

per year

Splitting the Monolith

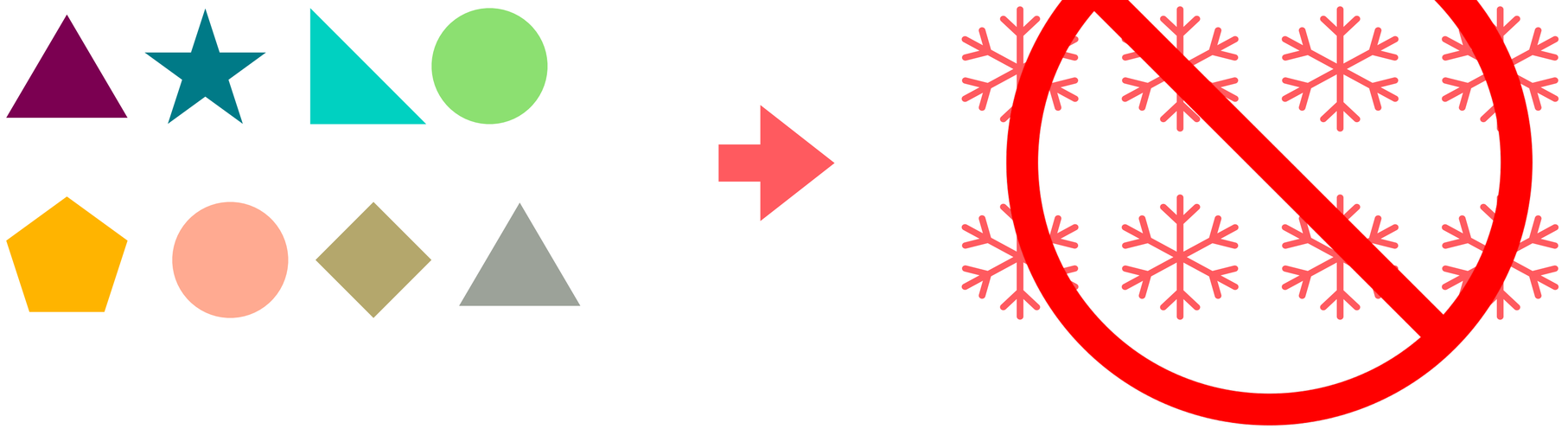


Splitting the Monolith



Standardize Service Creation

NO SNOWFLAKES

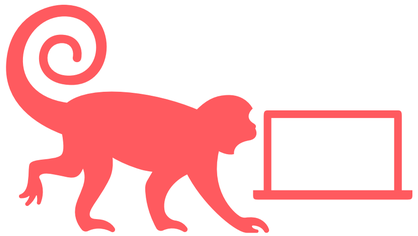


Standardize Service Creation

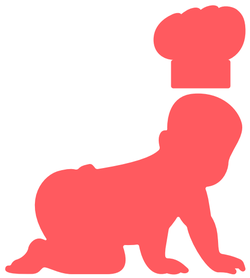
OKAY BUT HOW?

CONFIGURATION IS KEY

Evolution of Configuration Management



Manually configuring boxes



Automating configuration of applications with Chef

Automating configuration and orchestration of containerized applications with Kubernetes

KUBERNETES OUT-OF-THE-BOX

What Makes Kubernetes Awesome

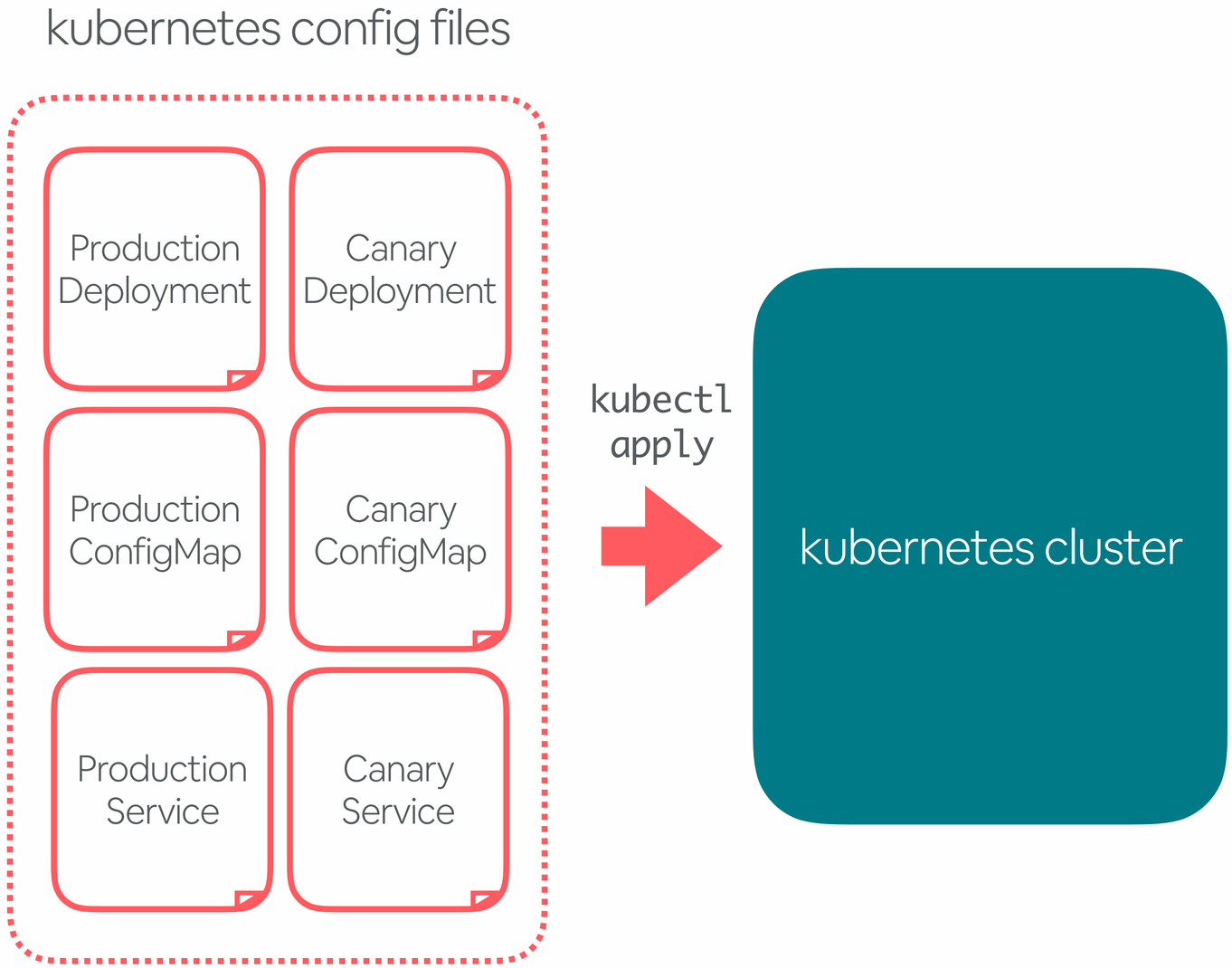
- immutable, reproducible containers
- designed for a microservices architecture
- human-readable format
- declarative
- efficient scheduling
- extensible API

What Makes Kubernetes NOT Awesome

- significant set up cost
- complex configuration and concepts
- tooling is not developer friendly
- open issues

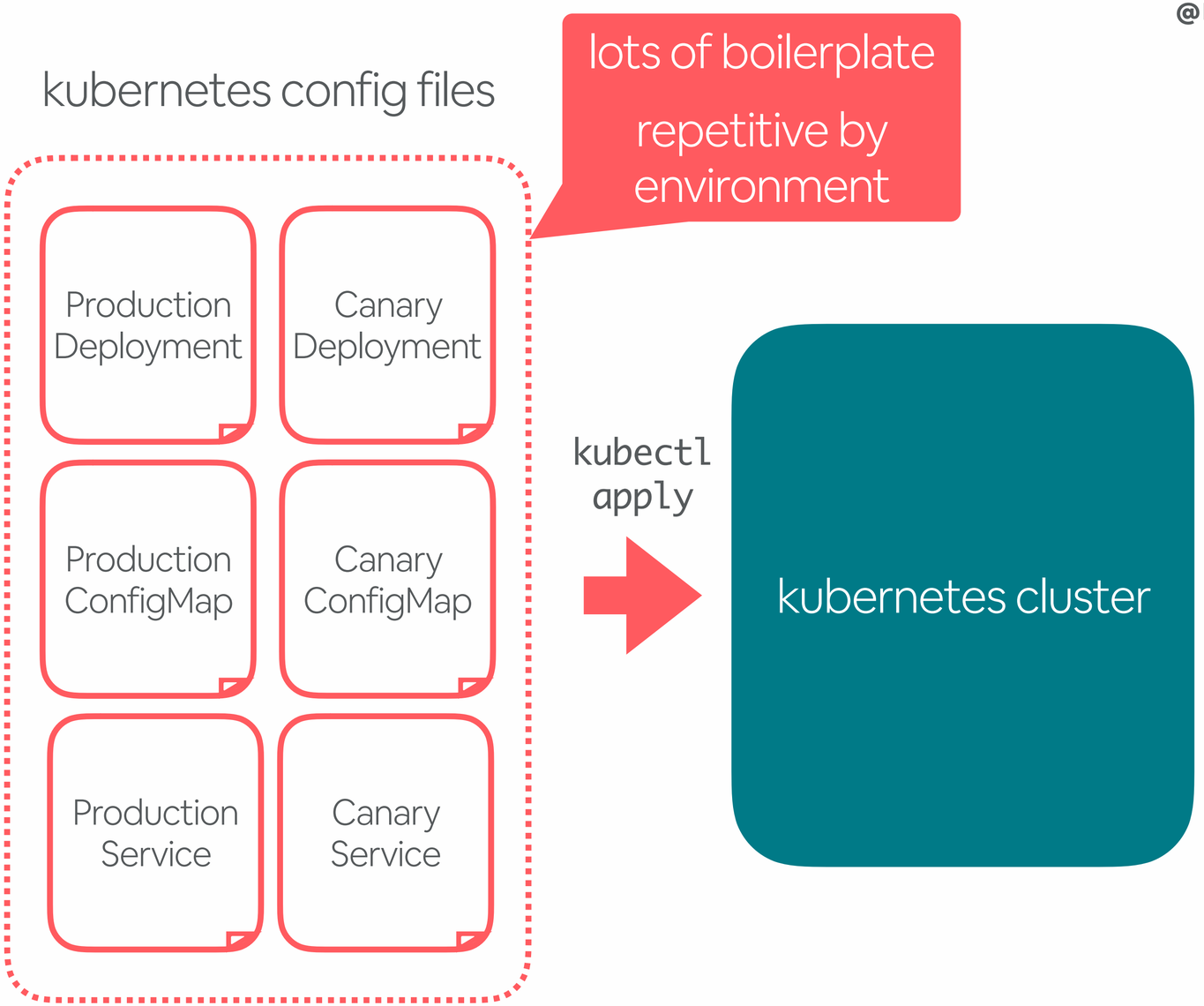
GENERATING KUBERNETES FILES

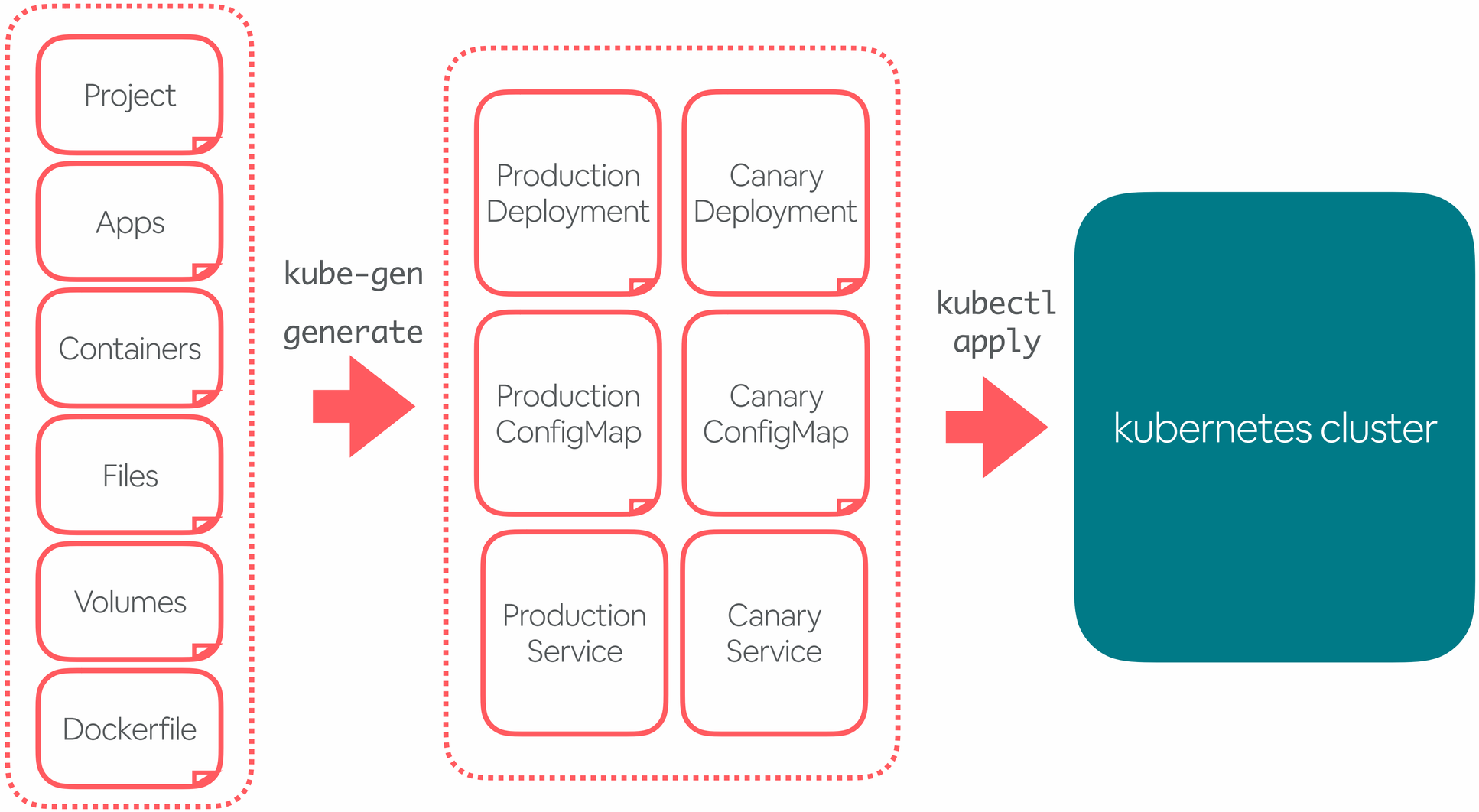
kubernetes

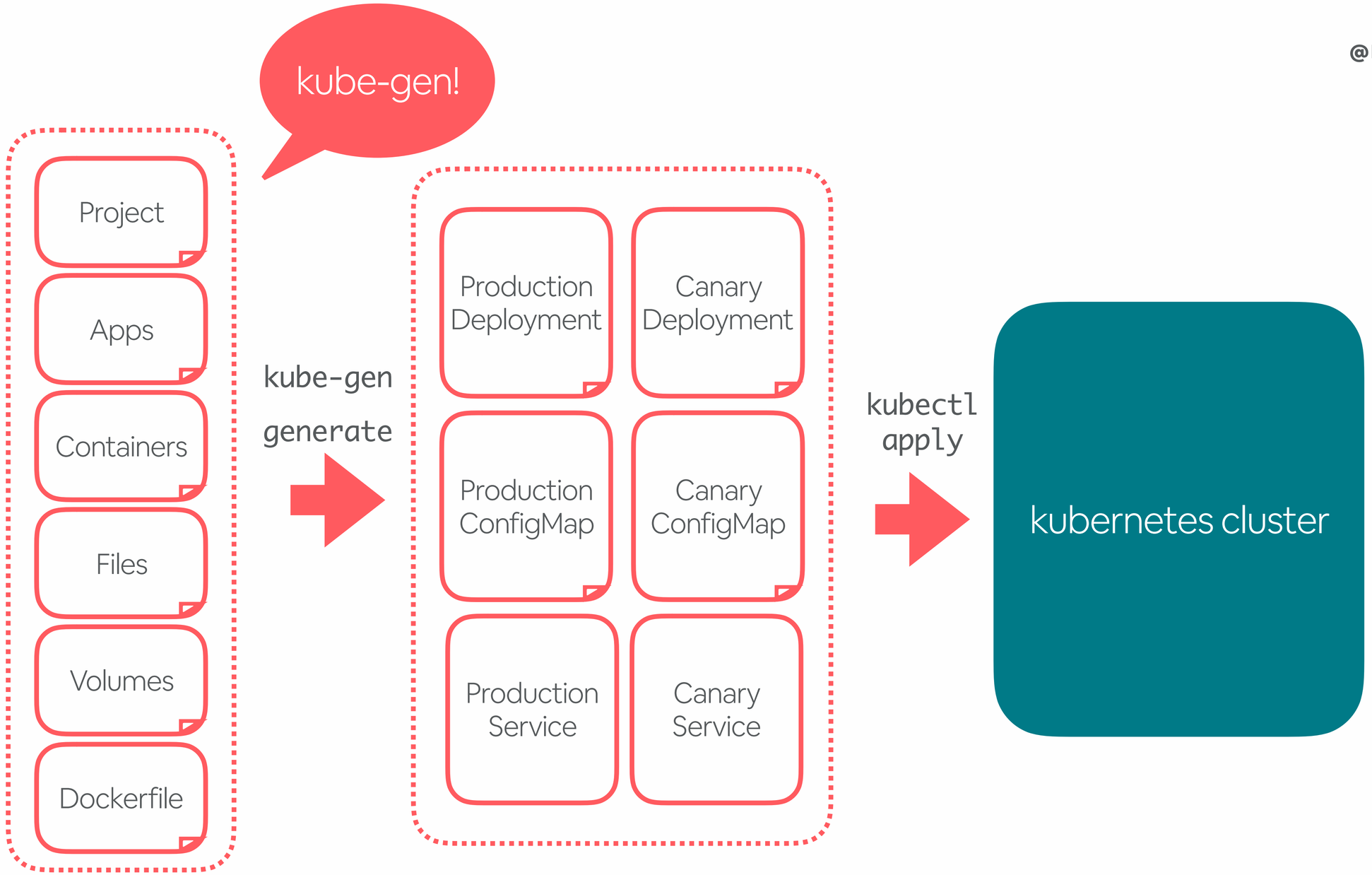

kubernetes config files
Production Deployment
Canary Deployment
Production ConfigMap
Canary ConfigMap
Production Service
Canary Service
kubectl apply
kubernetes cluster

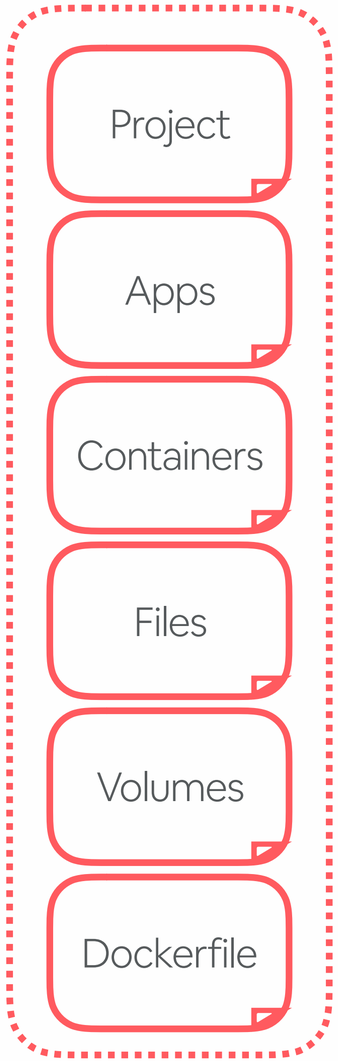
kubernetes

IS REPETITIVE




Project
Apps
Containers
Files
Volumes
Dockerfile
kube-gen
generate
Production Deployment
Canary Deployment
Production ConfigMap
Canary ConfigMap
Production Service
Canary Service
kubectl
apply
kubernetes cluster


kube-gen!
Project
Apps
Containers
Files
Volumes
Dockerfile
kube-gen
generate
Production Deployment
Canary Deployment
Production ConfigMap
Canary ConfigMap
Production Service
Canary Service
kubectl
apply
kubernetes cluster



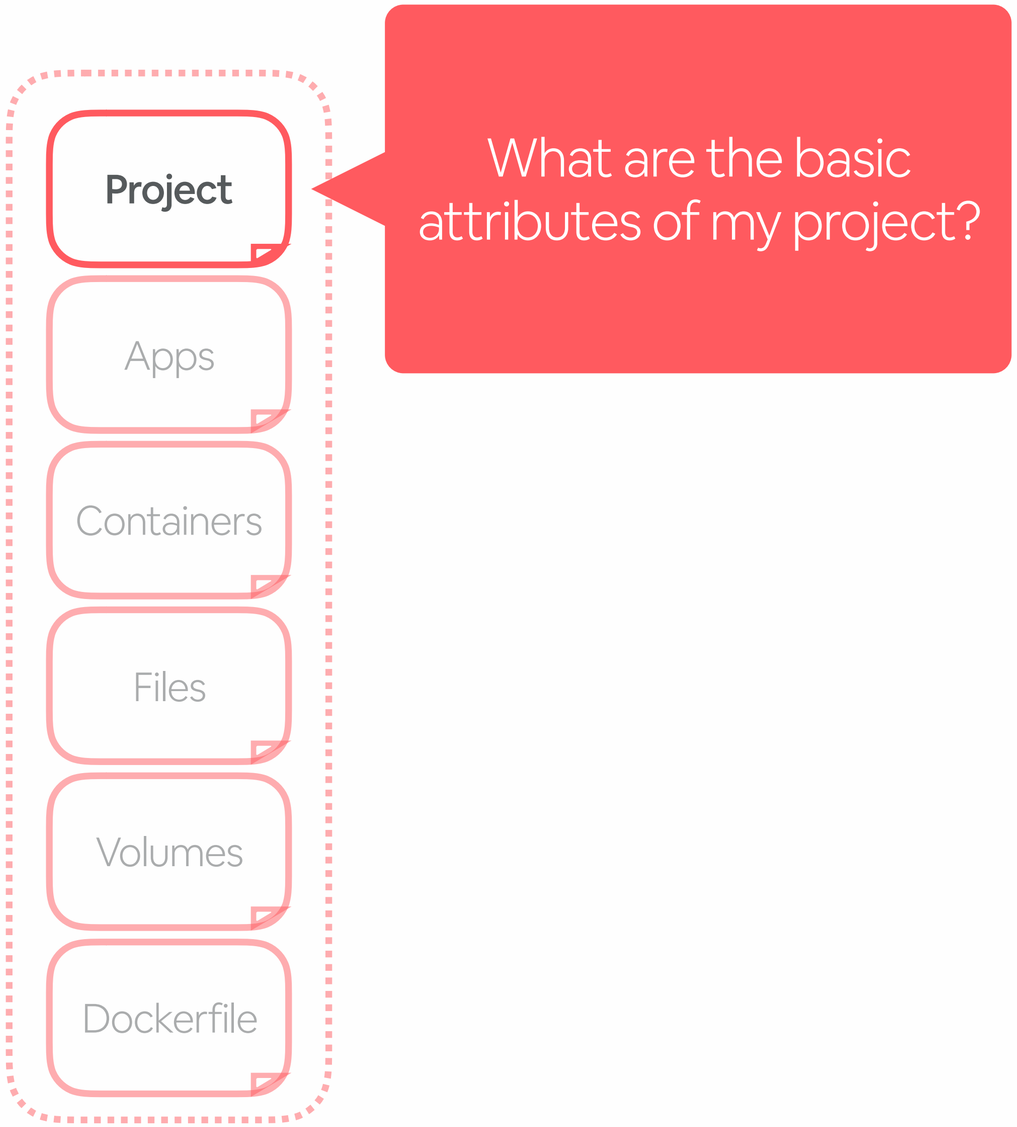
kube-gen

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/
```

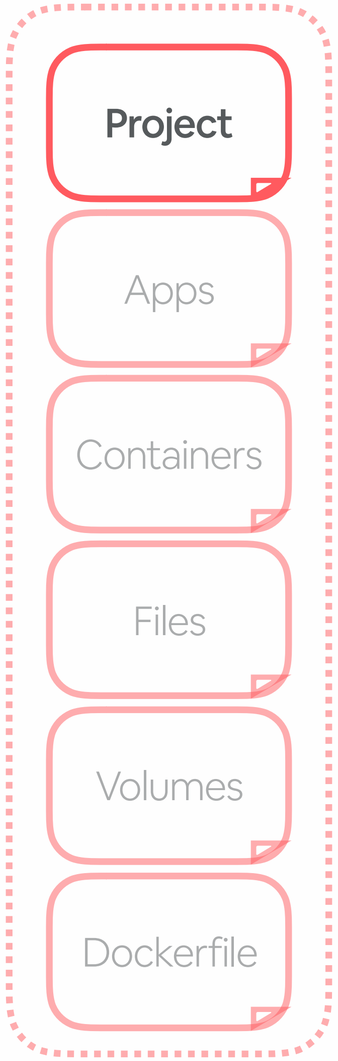
kube-gen

THE PROJECT FILE



kube-gen

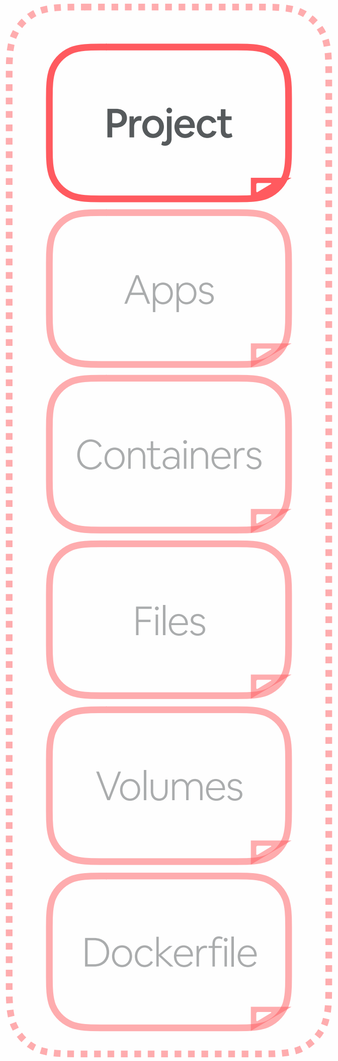
THE PROJECT FILE




```
# under _infra/kube/kube-gen.yml
version: 5.1.0
project:
  name: bonk
environments:
  production:
    params:
      replicas: 10
      port: 6585
  staging:
    params:
      replicas: 2
    port: 6586
users:
    melanie_cebula
```

kube-gen

THE PROJECT FILE

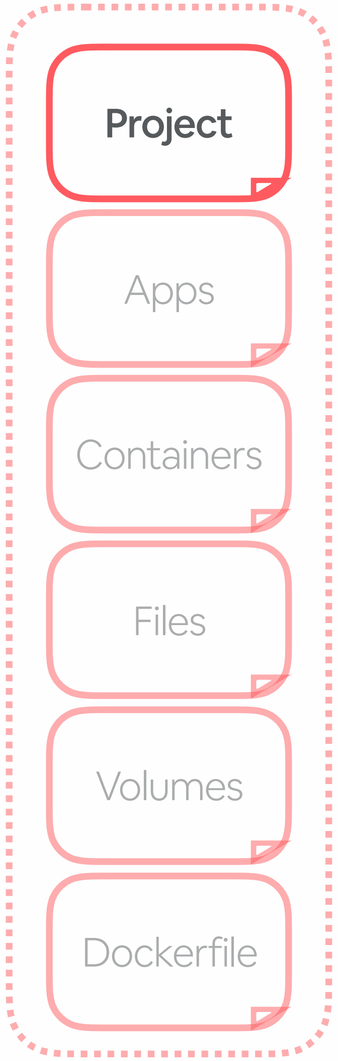



```
# under _infra/kube/kube-gen.yml
version: 5.1.0
project:
  name: bonk
environments:
  production:
    params:
      replicas: 10
      port: 6585
  staging:
    params:
      replicas: 2
    port: 6586
users:
  melanie_cebula
```

becomes an admin role binding file!

kube-gen

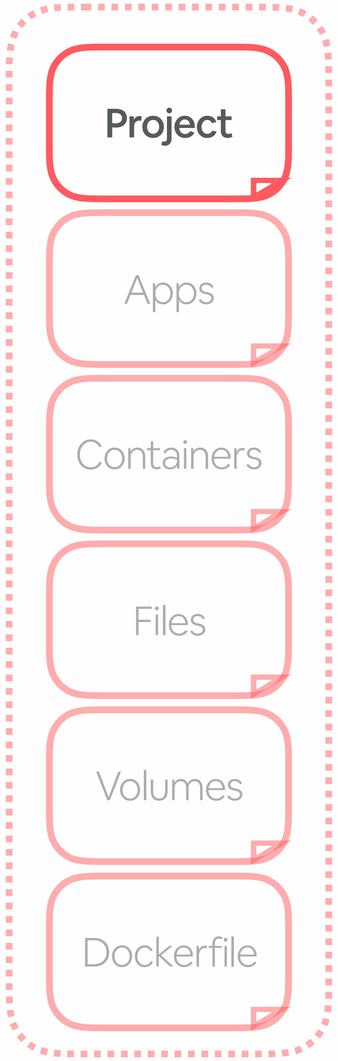
GENERATED ADMIN ROLEBINDING FILE




```
# under generated/bonk/bonk-production/admin-role-binding.yml

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

  creationTimestamp: null

  name: bonk-production-admin-role-binding

  namespace: bonk-production

roleRef:

  apiGroup: rbac.authorization.k8s.io

  kind: ClusterRole

  name: admin

subjects:

- kind: User

  name: melanie_cebula
```

kube-gen

THE PROJECT FILE

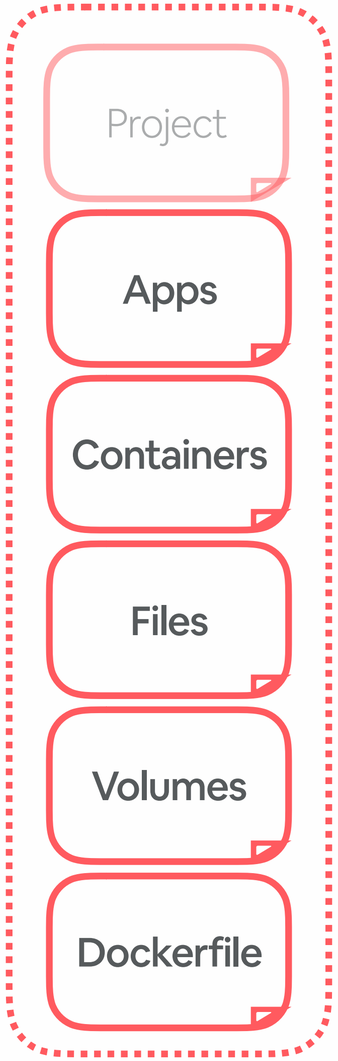



```
# under _infra/kube/kube-gen.yml
version: 5.1.0
project:
  name: bonk
environments:
  production:
    params:
      replicas: 10
      port: 6585
  staging:
    params:
      replicas: 2
    port: 6586
users:
    melanie_cebula
```

environments can be accessed by all the other files!

kube-gen

TEMPLATE VARIABLES

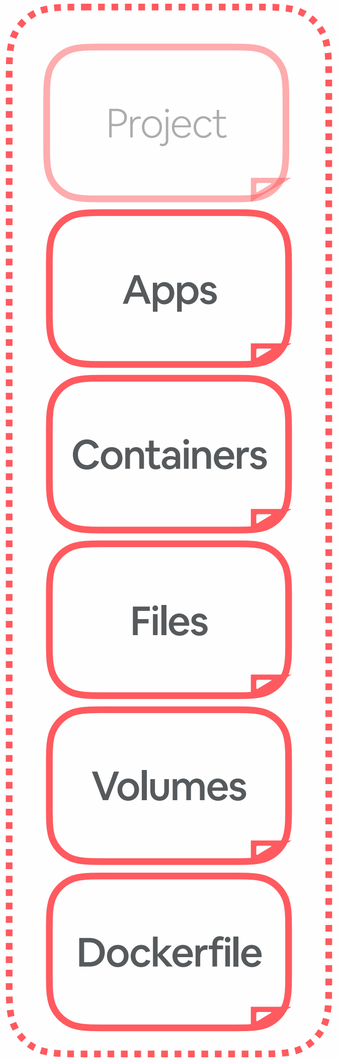


- different concepts map to different yaml files under `_infra/kube`
- environment params defined in the project are accessed with go templating:
- ex: `{{ .Env.Params.replicas }}`

```
# under _infra/kube/apps/bonk.yml
workload:
  deployment:
    replicas: {{ .Env.Params.replicas }}
    strategy:
      rollingUpdate
```

kube-gen

TEMPLATE VARIABLES



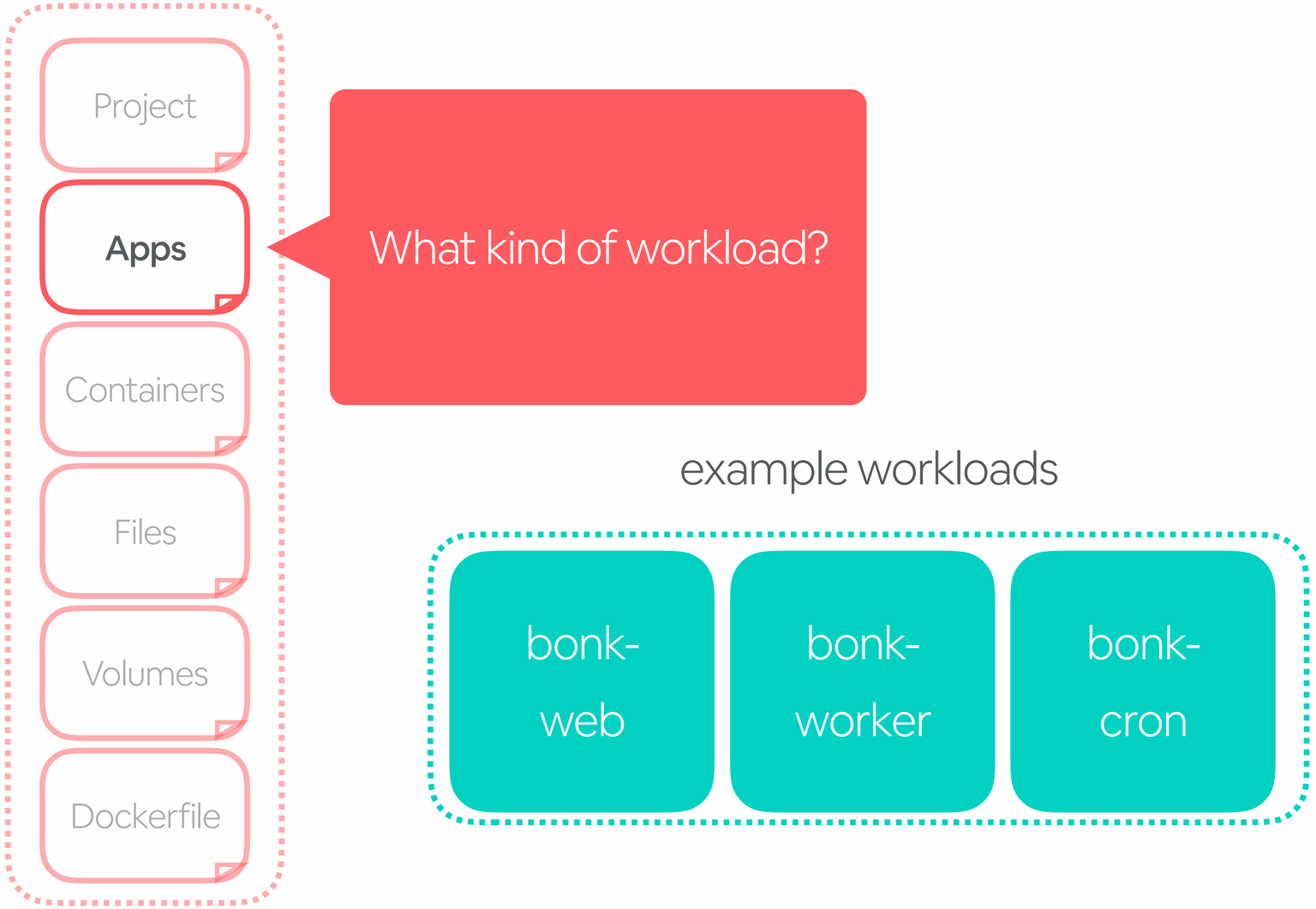
- `{{ }}` signals go templating
- kube-gen will generate all the files for the different environments, and replace each `.Env.Params.replicas` with the appropriate value from the project file
- ex: staging has 2 replicas, production has 10 replicas

```
# under _infra/kube/apps/bonk.yml
workload:
  deployment:
    replicas: {{ .Env.Params.replicas }}
    strategy:
      rollingUpdate
```

from the project file!

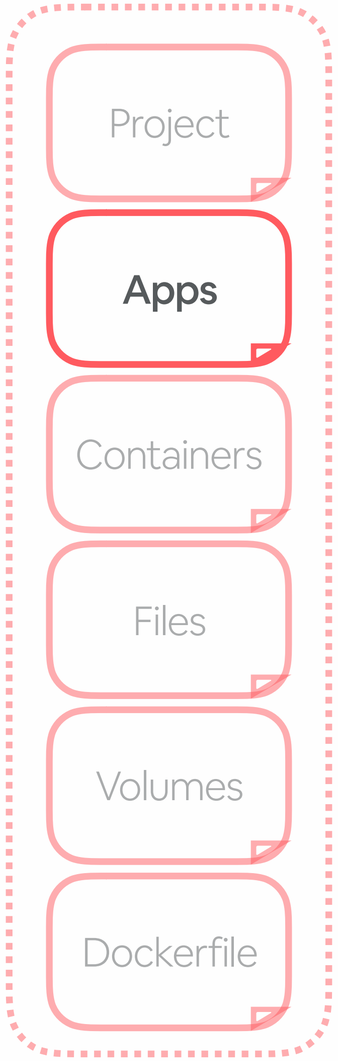
kube-gen

APP FILES



kube-gen

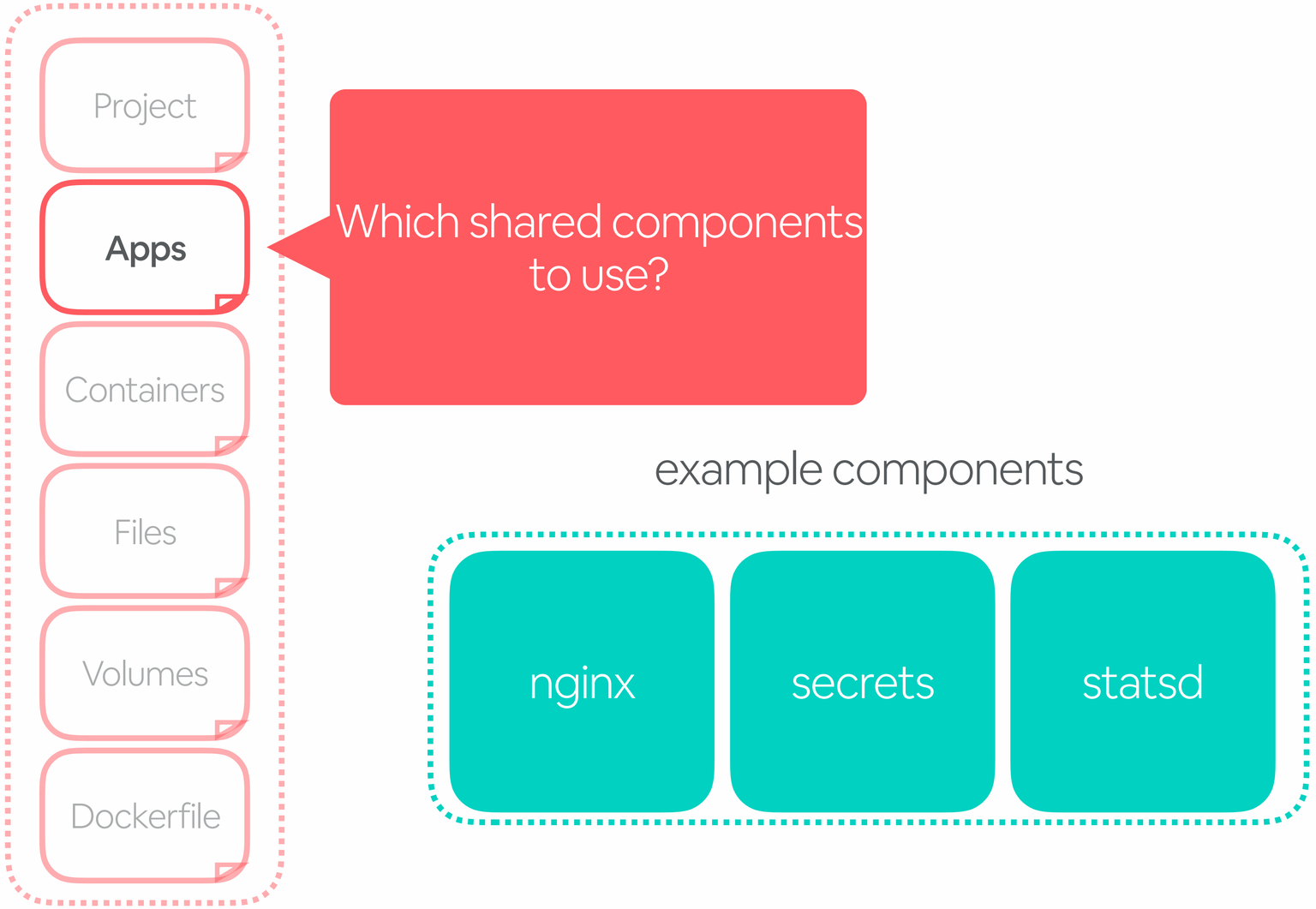
APP FILES




```
# under _infra/kube/apps/bonk-web.yml
workload:
  deployment:
{{ if eq .Env.Name “development” }}
   strategy:
      rollingUpdate:
        maxUnavailable: 1
{{ else }}
    autoscaling:
      minReplicas: {{ .Env.Params.minReplicas }}
      maxReplicas: {{ .Env.Params.maxReplicas }}
{{ end }}
```

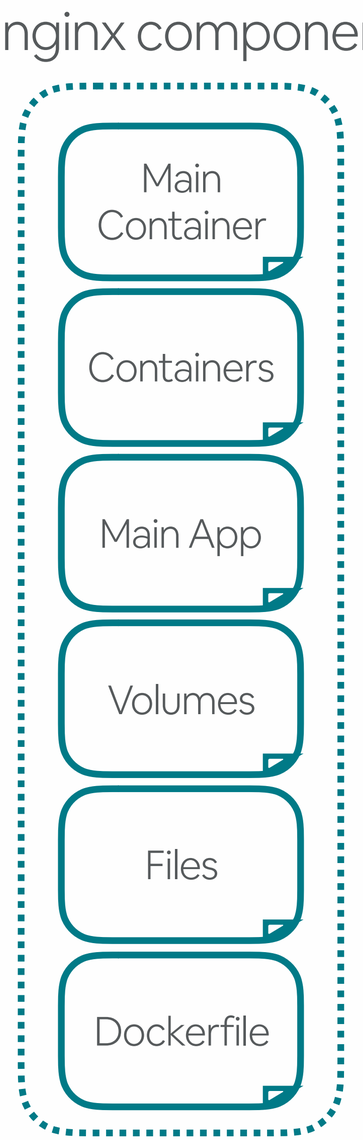
kube-gen

COMPONENTS



kube-gen

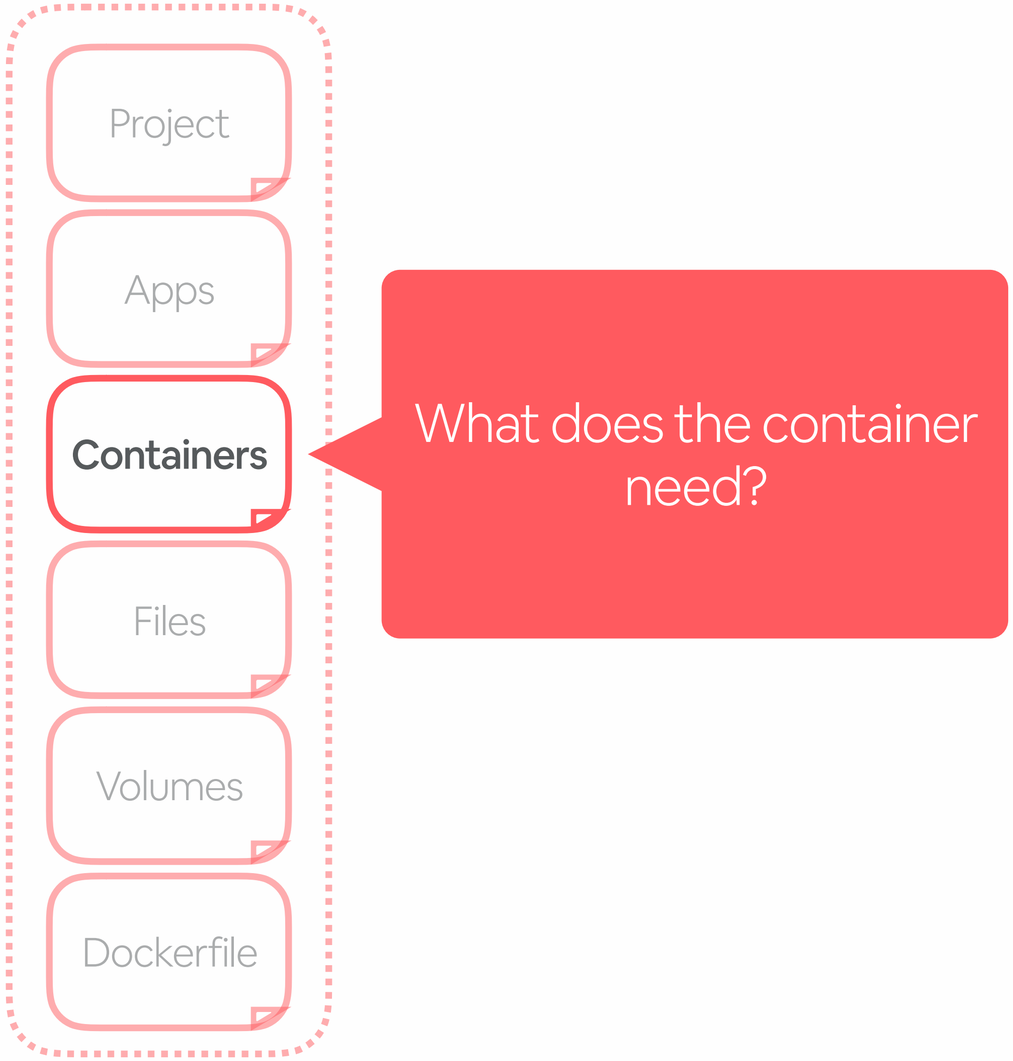
COMPONENTS



- common patterns are abstracted into a component
- apps enable components
- component yaml merged into project
- may require params to be set
- may set default params

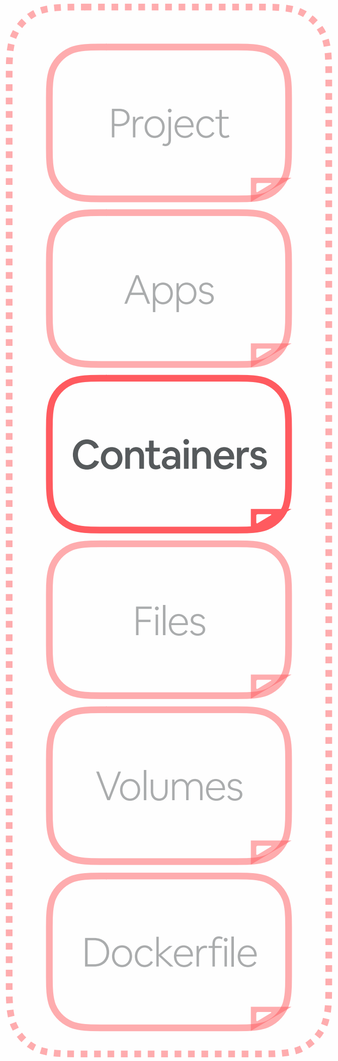
kube-gen

CONTAINER FILES



kube-gen

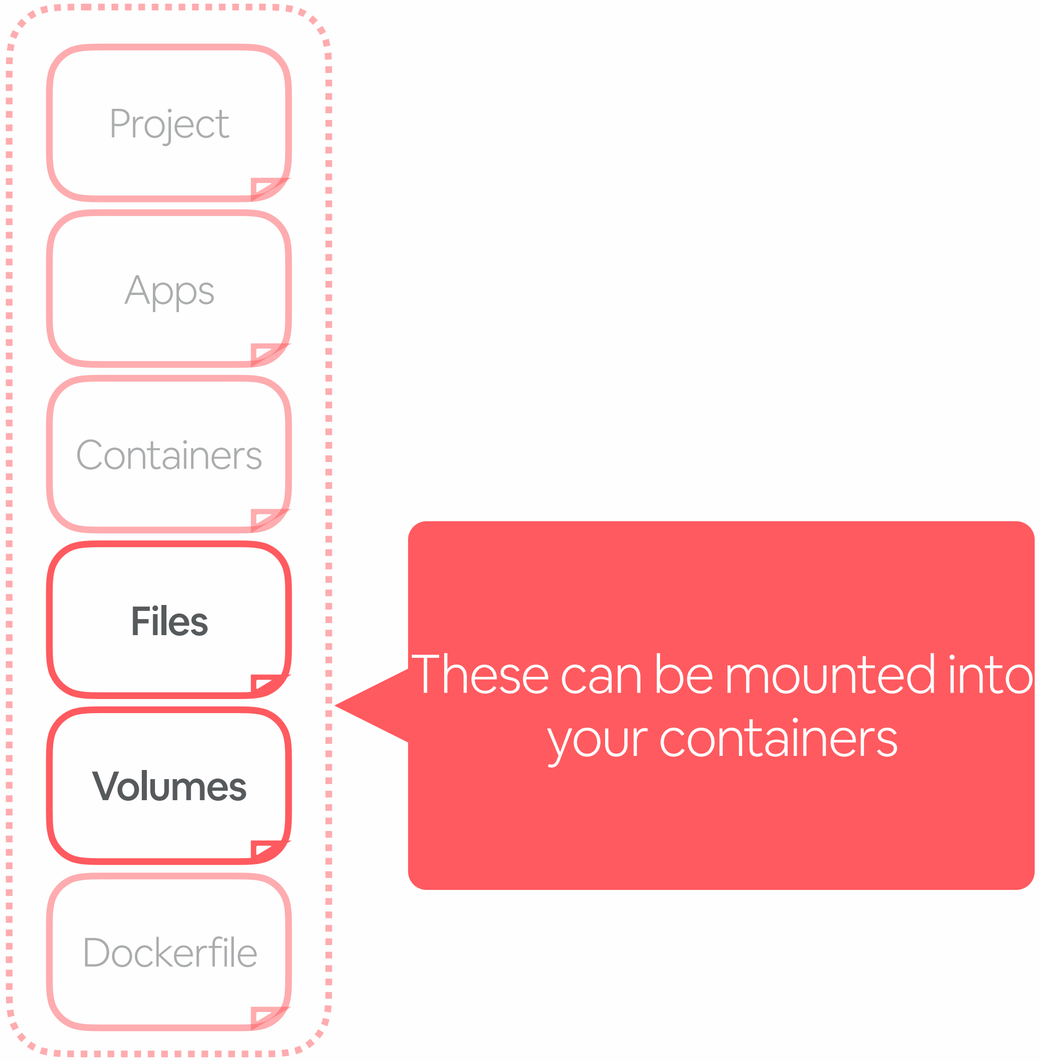
CONTAINER FILES



- image to use
- command and args to run
- ENV variables to pass in
- resource requests and limits for container
- file and volume mounts used by this container

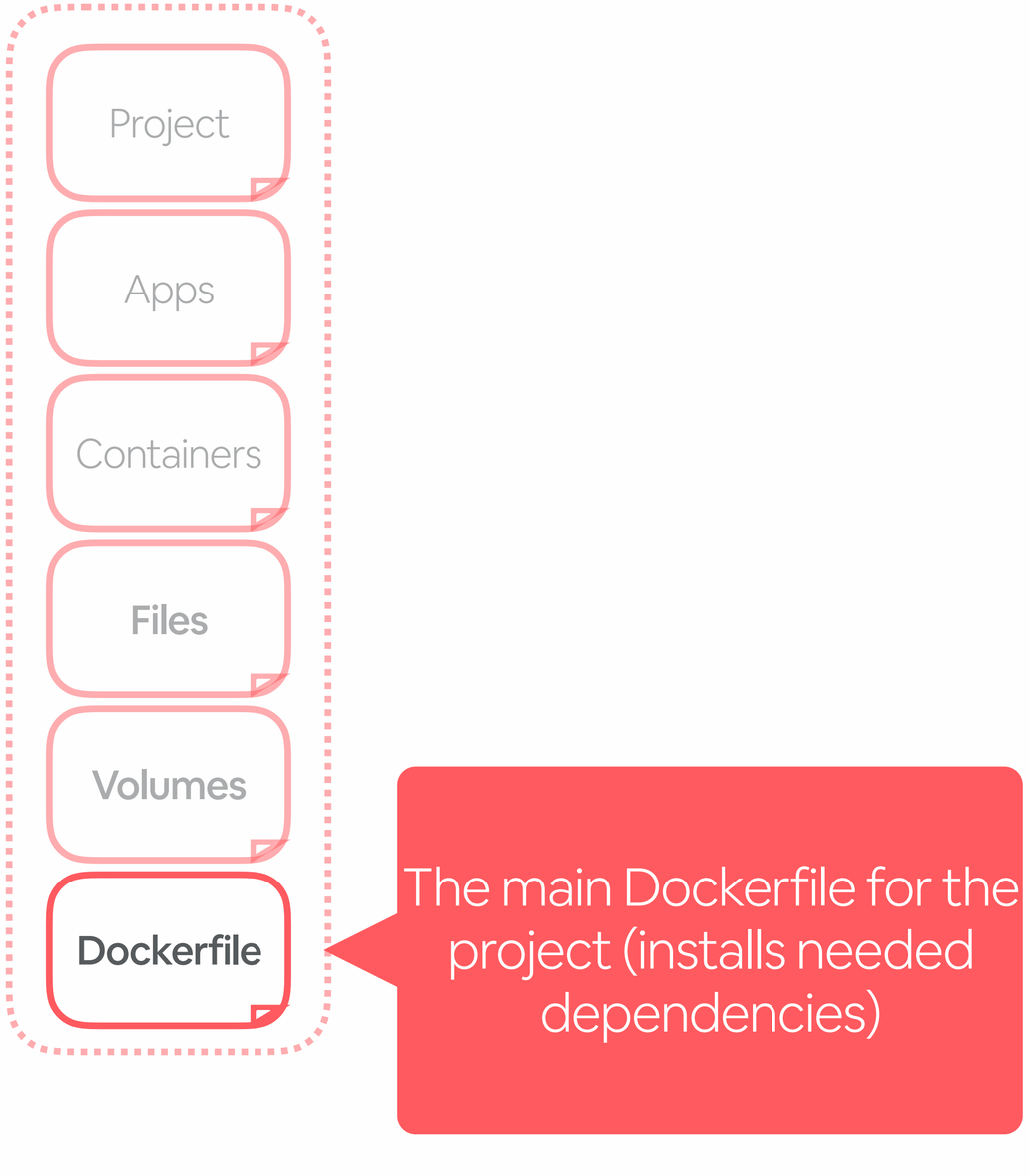
kube-gen

OTHER FILES



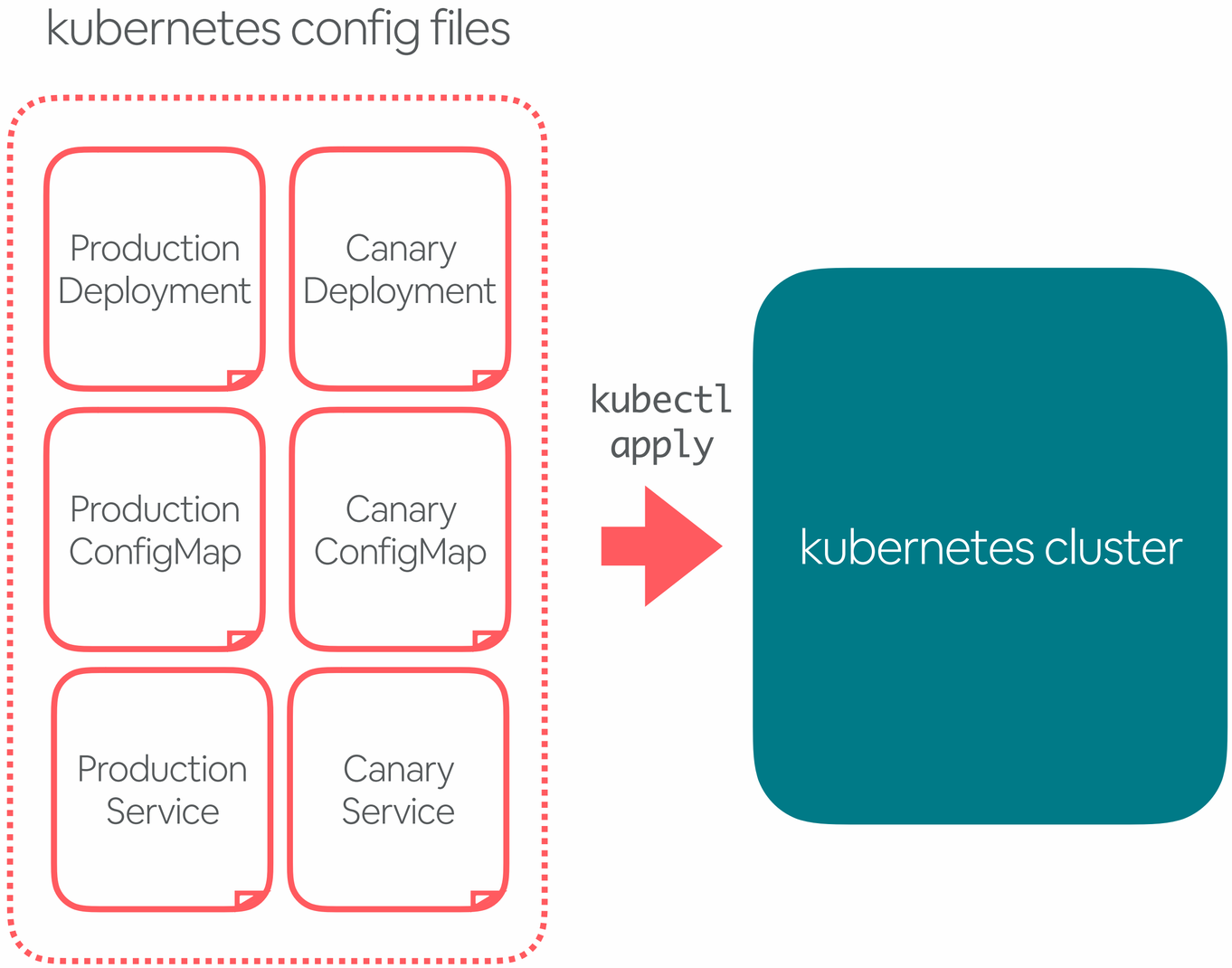
kube-gen

OTHER FILES



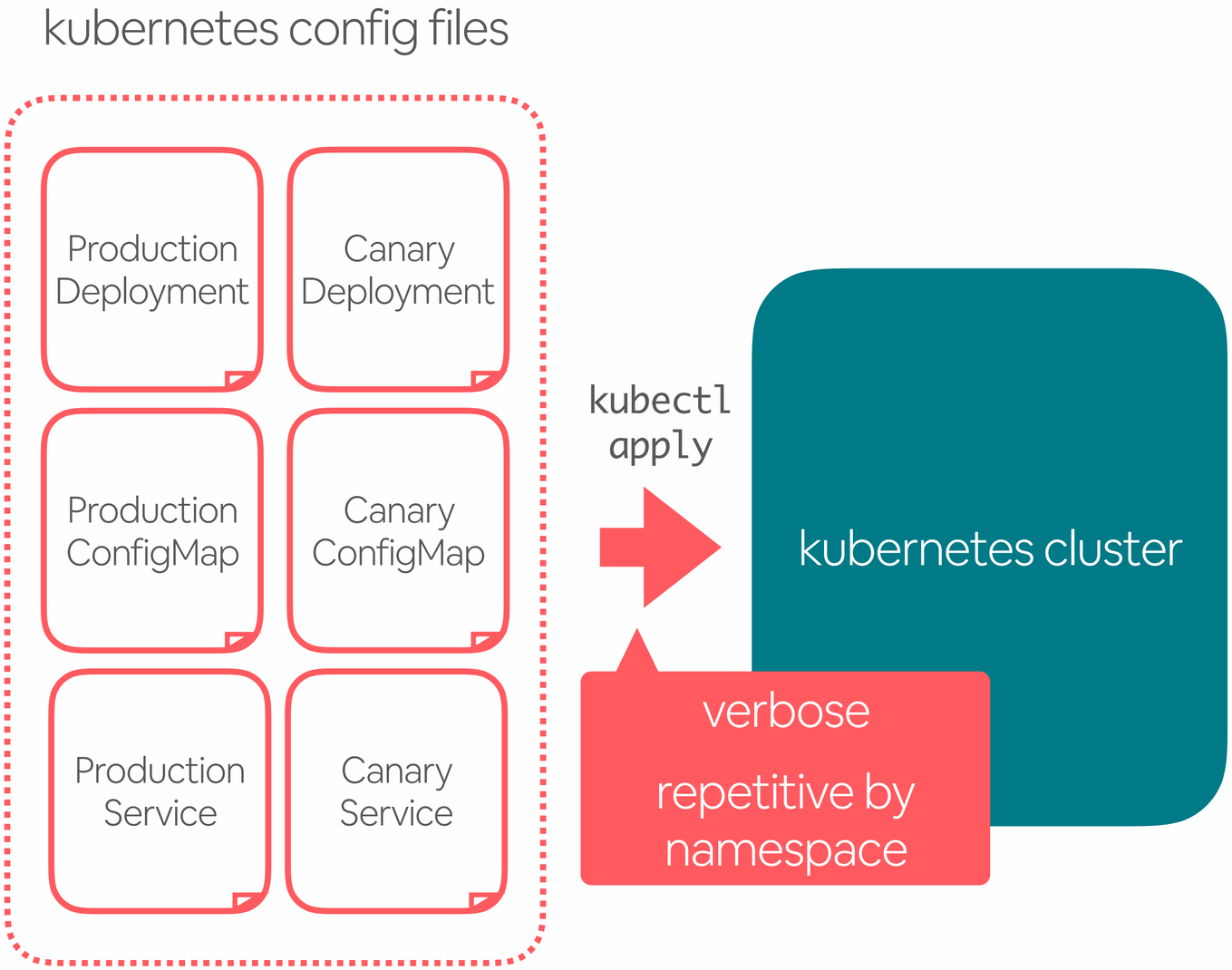
KUBECTL WRAPPER

kubernetes


kubernetes config files
Production Deployment
Canary Deployment
Production ConfigMap
Canary ConfigMap
Production Service
Canary Service
kubectl apply
kubernetes cluster

kubectl

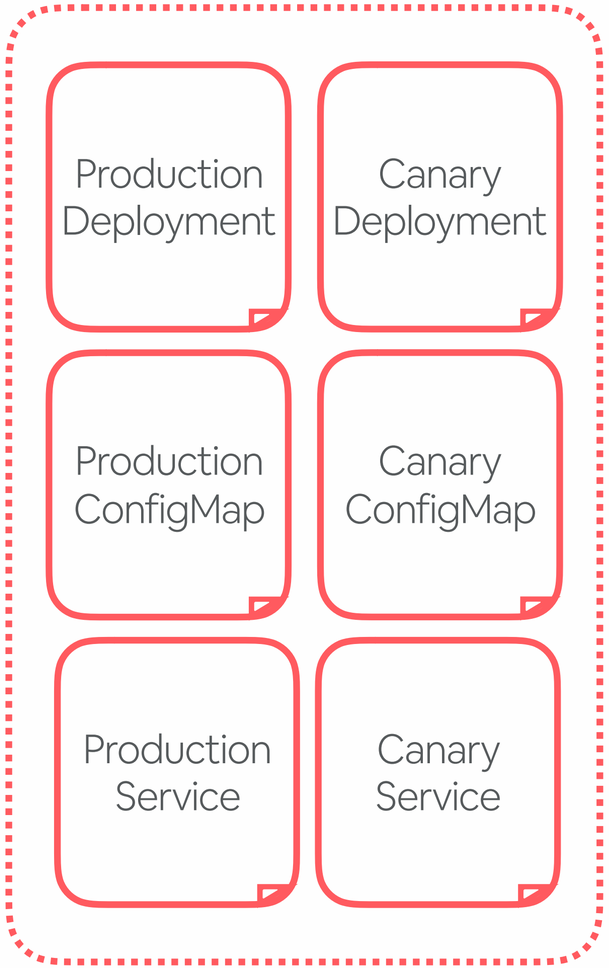
IS VERBOSE



k tool

KUBECTL WRAPPER

kubernetes config files



k tool

THE ALL PURPOSE CLI WRAPPER

@MELANIECEBULA

k tool

USES ENV VARS

- Runs in the project home directory:

```
$ cd /path/to/bonk
$ k status
```

- Environment variables for arguments:

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

- Prints the command that it will execute:

```
$ k status ENV=staging
kubectl get pods --namespace=bonk-staging
```

k tool

SIMPLIFIES BUILD AND DEPLOYS

- `k generate` transforms kube-gen files to kubernetes files
- `k build` performs 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=statsd-proxy
```

- `k diagnose`: shows status and logs for each failing container of your pod, and shows failure events

EXTENDING KUBERNETES

Extending Kubernetes

WITH CUSTOM CONTROLLERS

- **admission controller:**
  - require project ownership (via annotations)
  - prevent namespace creation under incorrect cluster
  - prevent deployment of objects with known security holes
- **deployment pruner:**
  - automatically rotate old pods
  - rotates over-provisioned availability zones

Extending Kubernetes

WITH GRACEFUL TERMINATION

- it is a common anti-pattern for clients of a service to not retry on connection failures
- as a k8s service deploys, requests may be sent to terminating pods and cause a spike in 500s
- use graceful termination to mitigate this

Extending Kubernetes

WITH GRACEFUL TERMINATION

```
# under _infra/kube/apps/bonk.yml
# wait up to 180 seconds before “kill -9” the pods
terminationGracePeriodSeconds: 180
```

```
# under _infra/kube/containers/container.yml
 # wait up to 120 seconds before shutting down
lifecycle:
  preStop:
    exec:
      command:
        - /bin/sleep
        - “120”
```

Extending Kubernetes

WITH GRACEFUL TERMINATION

```
# under _infra/kube/apps/bonk.yml
# wait up to 180 seconds before “kill -9” the pods
terminationGracePeriodSeconds: 180
```

```
# under _infra/kube/containers/container.yml
 # wait up to 120 seconds before shutting down
lifecycle:
  preStop:
    exec:
      command:
        - /bin/sleep
        - “120”
```

this gives our service discovery container time to mark this pod as unhealthy

Extending Kubernetes

WITH GRACEFUL TERMINATION

```
# under _infra/kube/apps/bonk.yml
# wait up to 180 seconds before “kill -9” the pods
terminationGracePeriodSeconds: 180
```

```
# under _infra/kube/containers/container.yml
 # wait up to 120 seconds before shutting down
lifecycle:
  preStop:
    exec:
      command:
        - /bin/sleep
        - “120”
```

you can do more sophisticated server shutdown here too!

Extending Kubernetes

WITH GRACEFUL TERMINATION

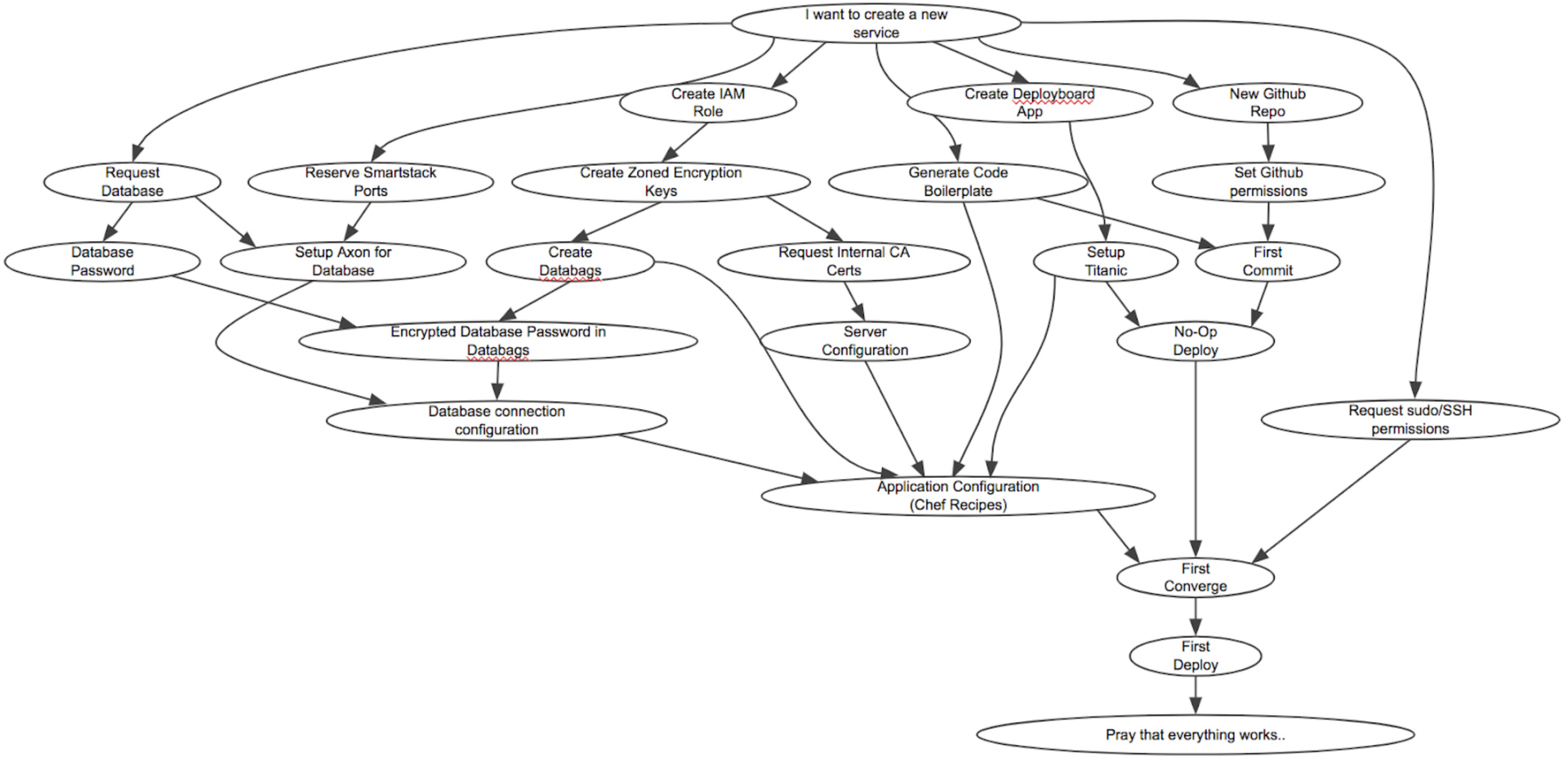
```
# under _infra/kube/apps/bonk.yml
# wait up to 180 seconds before “kill
terminationGracePeriodSeconds: 180
```

after the preStop command finishes, the containers have an additional 60sec to exit gracefully

```
# under _infra/kube/containers/container.yml
 # wait up to 120 seconds before shutting down
lifecycle:
  preStop:
    exec:
      command:
        - /bin/sleep
        - “120”
```

BEYOND KUBERNETES

Creating a new service (before)



Everything about a service should be in one place, and managed with one process.

Configuration

LIVES IN ONE PLACE

Everything you need to know about your service can be found in one place

- All configuration lives in _infra alongside project code
- Edit code and configuration with one pull request
- Easy to add new configuration
- Configuration statically validated as part of CI

Configuration

LIVES IN ONE PLACE

What we support:

- kube-gen files
- continuous integration files (with containers)
- documentation (markdown)
- databag secrets and keys
- development (legacy)
- AWS IAM roles
- internal tool configuration (UI/UX)
- legacy service discovery configuration
- project ownership and metadata
- .. and more!

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 changes atomically

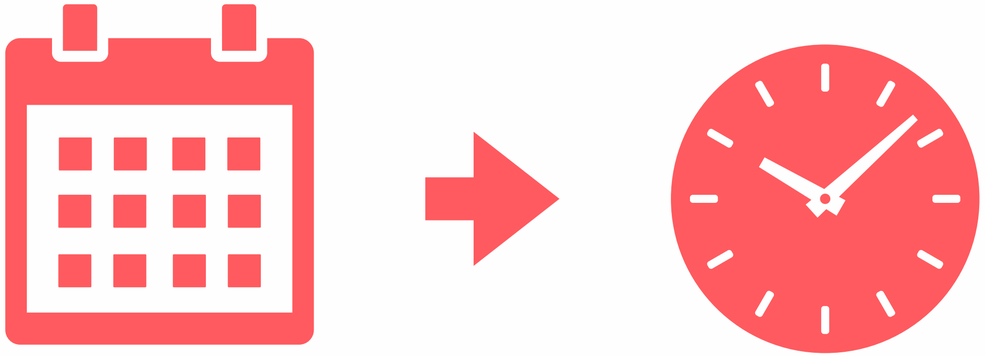
GENERATING SERVICE BOILERPLATE

Reduce service boilerplate

WITH GENERATORS!

- generators make best practices the default
- collection of config generators and language-specific generators
- uses thor for better control (update, review, commit)
- set up a “hello world” service with just one command

Reduce time to “hello world”



2 weeks

2 minutes

Takeaways

- Store configuration alongside project code
- Abstract away your infrastructure with generators
- Abstract away complex tooling with a wrapper CLI
- Configuration and tools should set defaults
- Standardize on one process for storing and applying configuration changes