Kubernetes <3 Erlang&Elixir

Dario Freddi, Code Beam Lite Milan, 2018

About me

Old FOSS chap rather new to Elixir/Erlang

Founded ispirata

Just released github.com/astarte-platform


LEFT
EXIT 12
STANDALONE CONTAINERS
ORCHESTRATION AND DETERMINISM

Kubernetes in a nutshell

A cluster for orchestrating Docker containers

Basic currency is a Pod: an expendable instance of one or more containers

Can be quite more advanced than that, but that’s for another talk

Why Kubernetes (or K8S)

Arguably won the orchestration war (RIP other orchestrators)

Using containers is no longer a PITA

Made a boring and lengthy process quite exciting and reliable

Compulsory warnings

I suck at Erlang, and arguably Elixir as well.
So all examples will be in Elixir.

K8S is a complicated beast, and I'll skip some basics due to time constraints.
Don't be afraid to ask! (but consider time is limited :))

Turning a BEAM application into a K8S deployment

How to containerise a BEAM application

Ditch your tooling. I’m serious.

```
FROM elixir:1.6-slim as builder

RUN apt-get -qq update
RUN apt-get -qq install git build-essential curl

RUN mix local.hex --force && \
    mix local.rebar --force && \
    mix hex.info

WORKDIR /app
ENV MIX_ENV prod
ADD . .
RUN mix deps.get
RUN mix release --env=$MIX_ENV

CMD ["./bin/astarte_pairing", "foreground"]
```

```
# Note: keep Debian versions in sync, or dependencies will kill you
FROM debian:jessie-slim
RUN apt-get -qq update

# Set the locale
ENV LANG C.UTF-8

# We need SSL
RUN apt-get -qq install libssl1.0.0

WORKDIR /app
COPY --from=builder /app/_build/prod/rel/astarte_pairing .

CMD ["./bin/astarte_pairing", "foreground"]
```

Multi stage Docker Builds

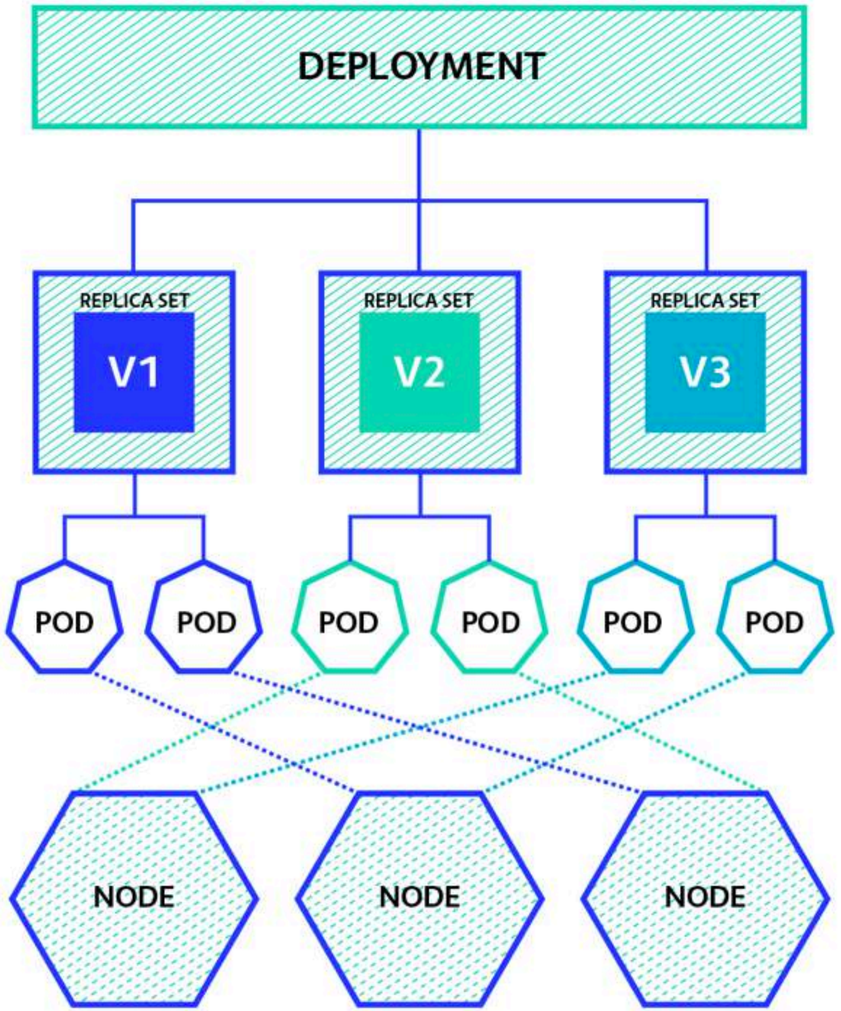
It’s pretty much CI/CD built into Docker itself

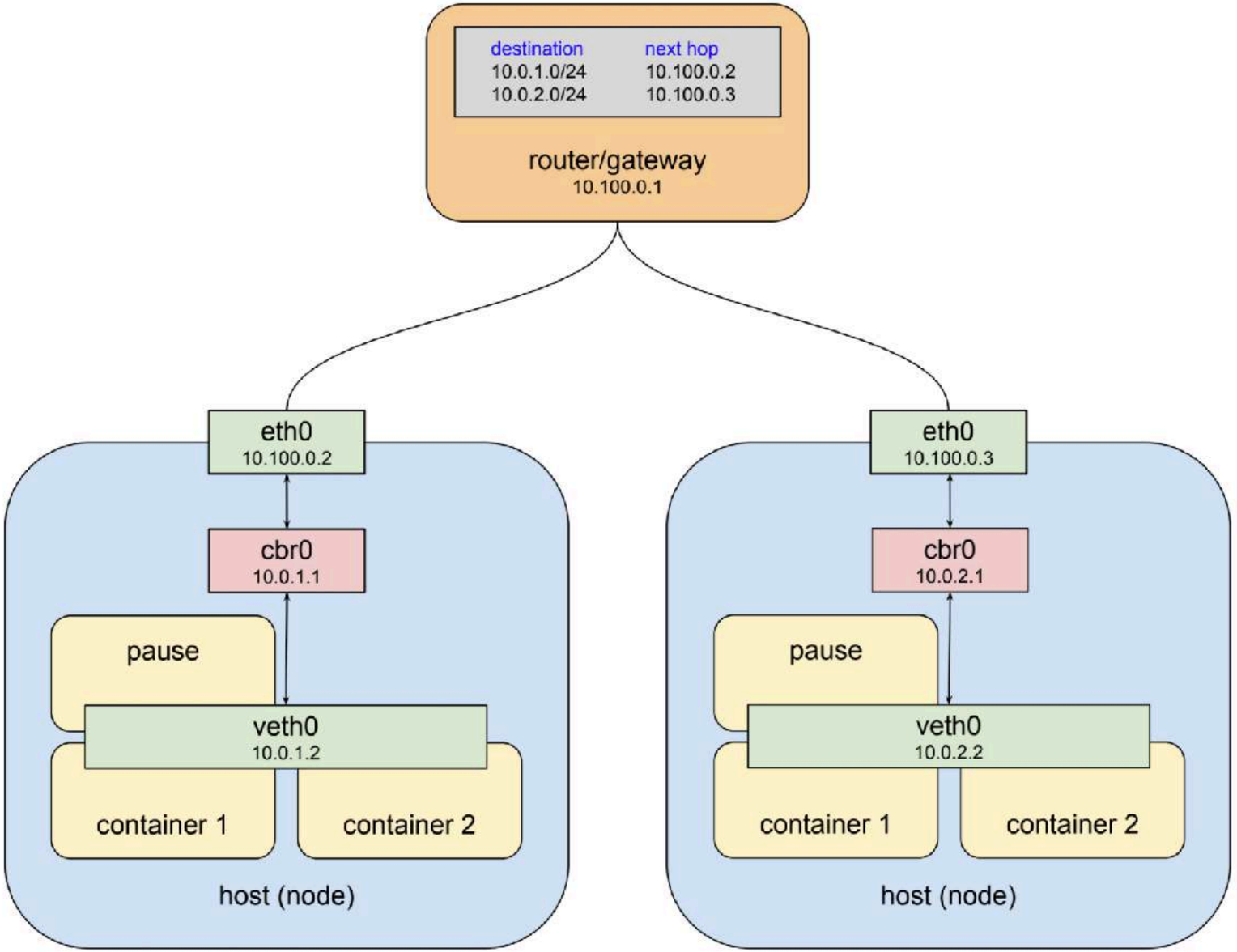
Being careful about adding layers is a thing of the past

Final image clean and small even without resorting to Alpine


ALL
THE
COOL
KIDS

```
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: astarte-pairing-deployment
  namespace: astarte
  labels:
    app: astarte-pairing
spec:
  replicas: 1
  selector:
    matchLabels:
      app: astarte-pairing
  template:
    metadata:
      labels:
        app: astarte-pairing
        astarte-service: pairing
    spec:
      containers:
      - name: astarte-pairing
        image: astarte/astarte_pairing:snapshot
        imagePullPolicy: Always
```


DEPLOYMENT
REPLICA SET
V1
REPLICA SET
V2
REPLICA SET
V3
POD
POD
POD
POD
POD
POD
NODE
NODE
NODE

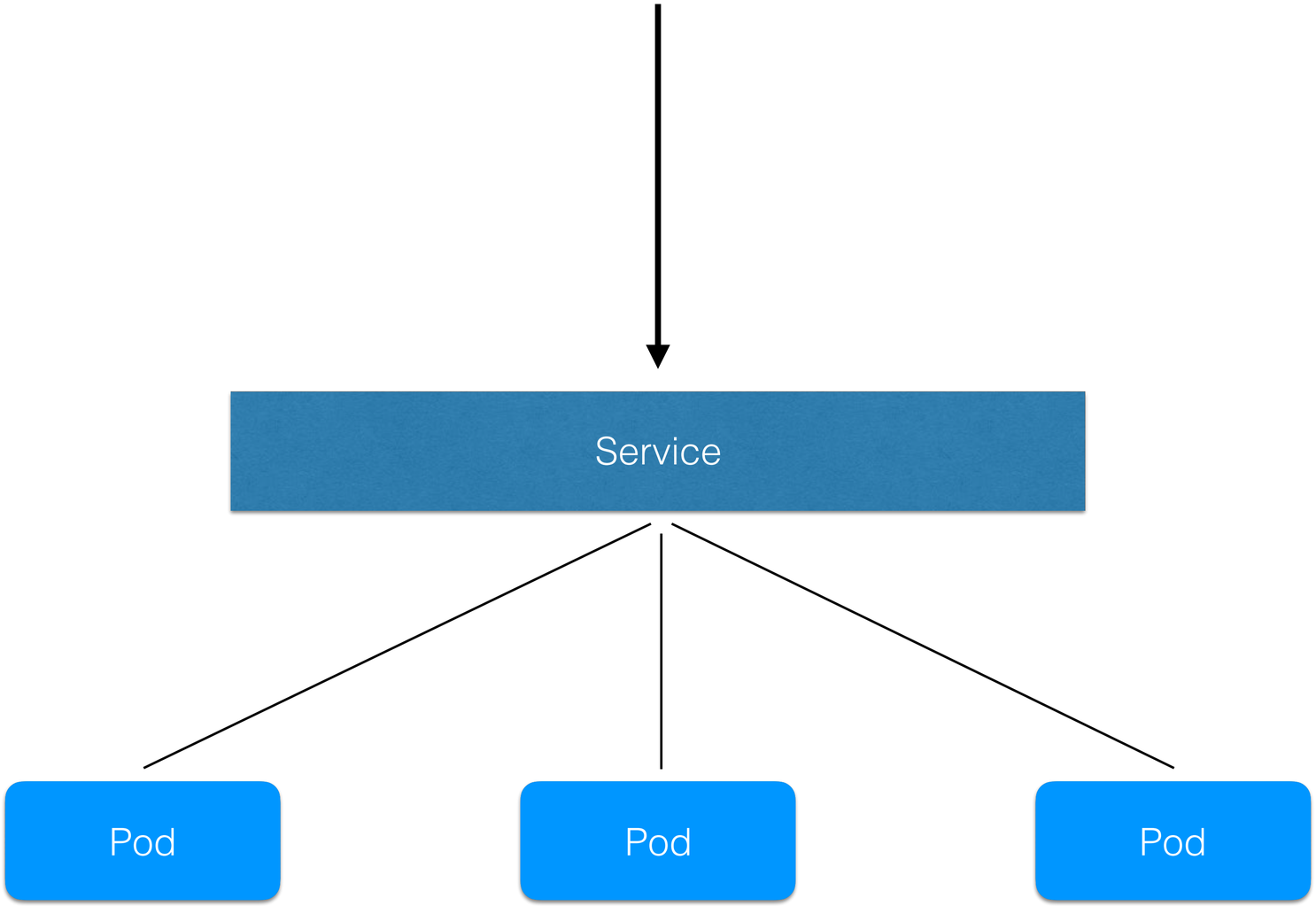

destination
next hop
10.0.1.0/24
10.100.0.2
10.0.2.0/24
10.100.0.3
router/gateway
10.100.0.1
eth0
10.100.0.2
cbr0
10.0.1.1
pause
veth0
10.0.1.2
container 1
container 2
host (node)
eth0
10.100.0.3
cbr0
10.0.2.1
pause
veth0
10.0.2.2
container 1
container 2
host (node)

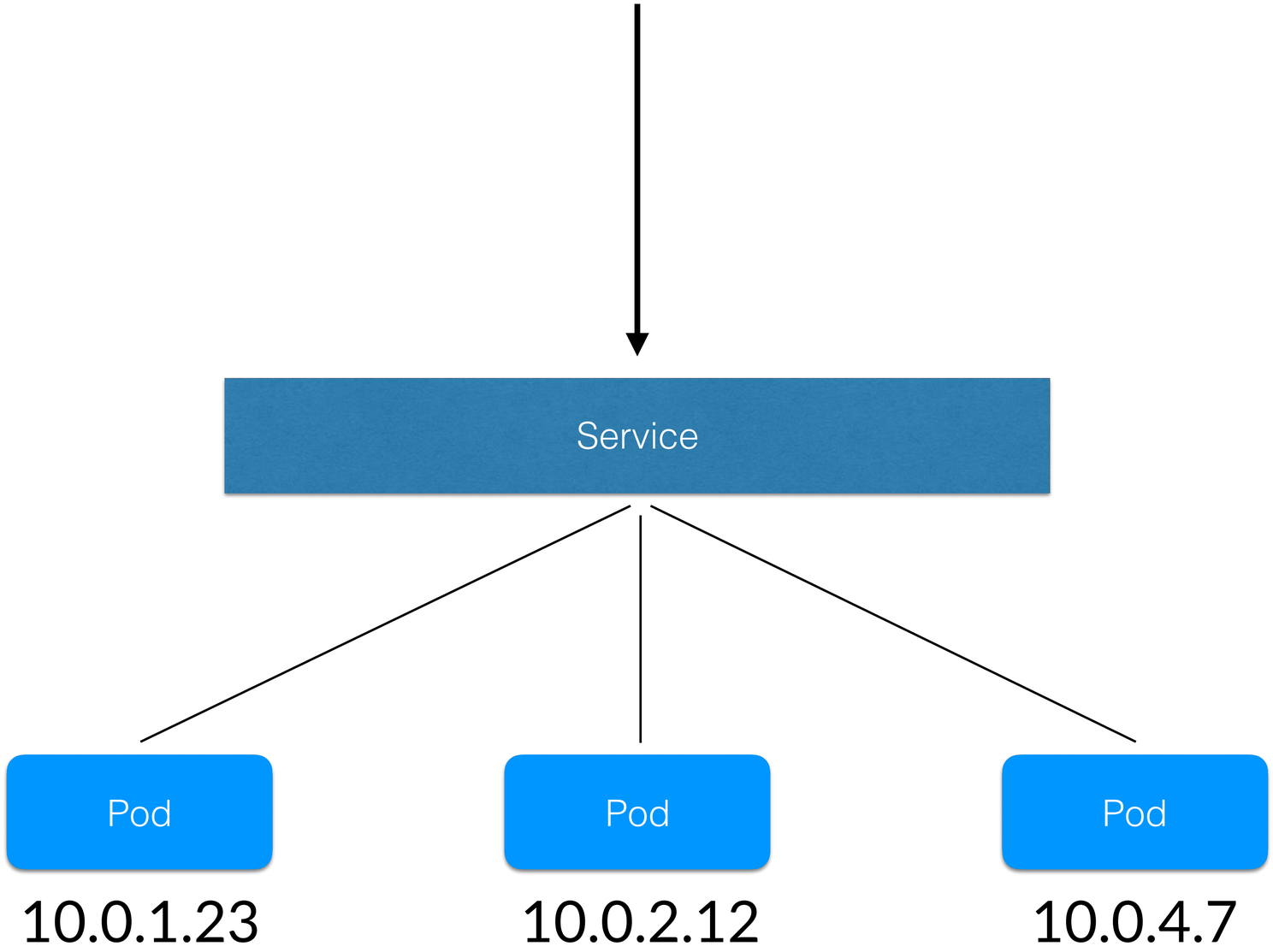
What happens

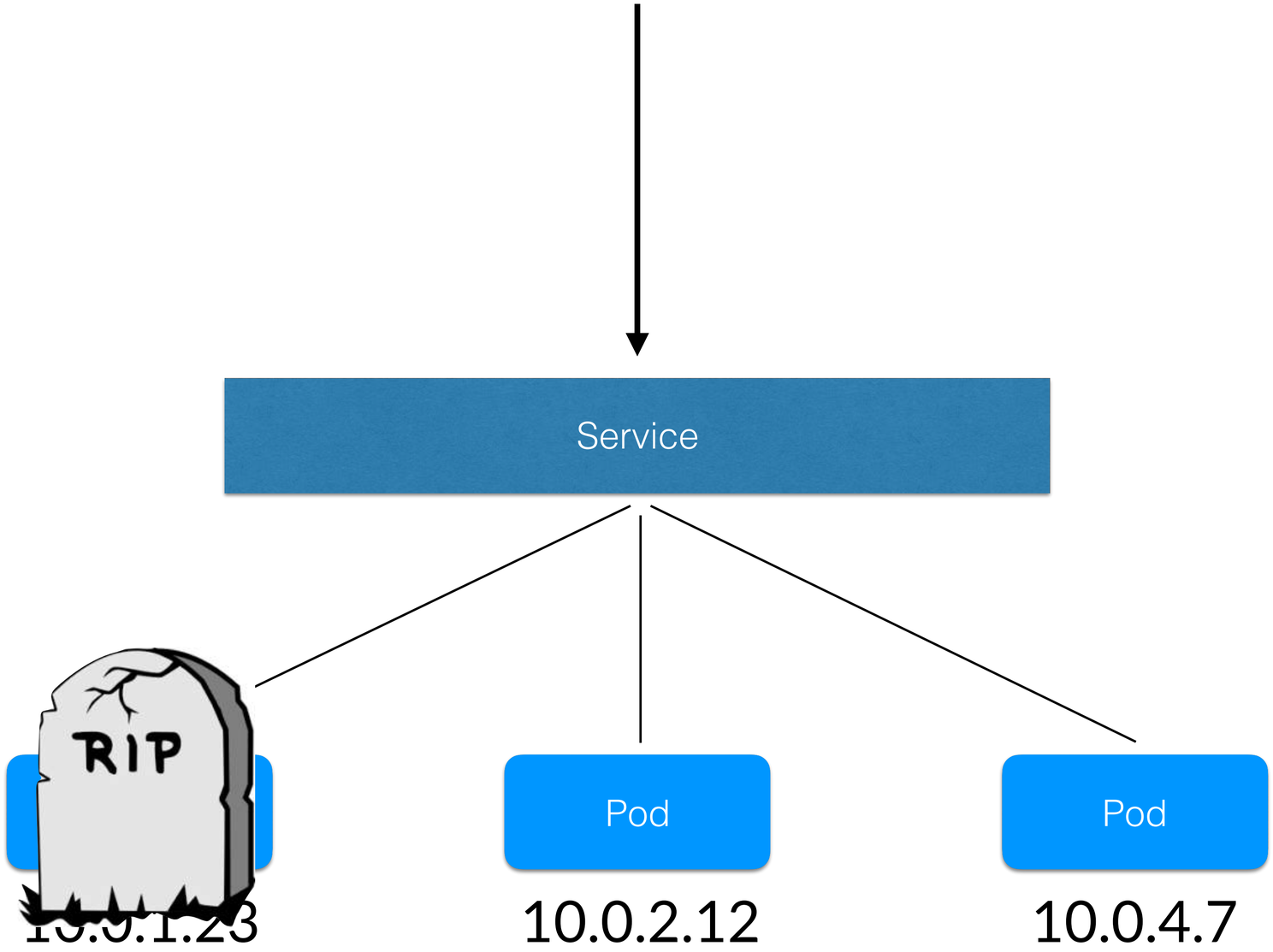
Our container is deployed on a node of our K8S Cluster

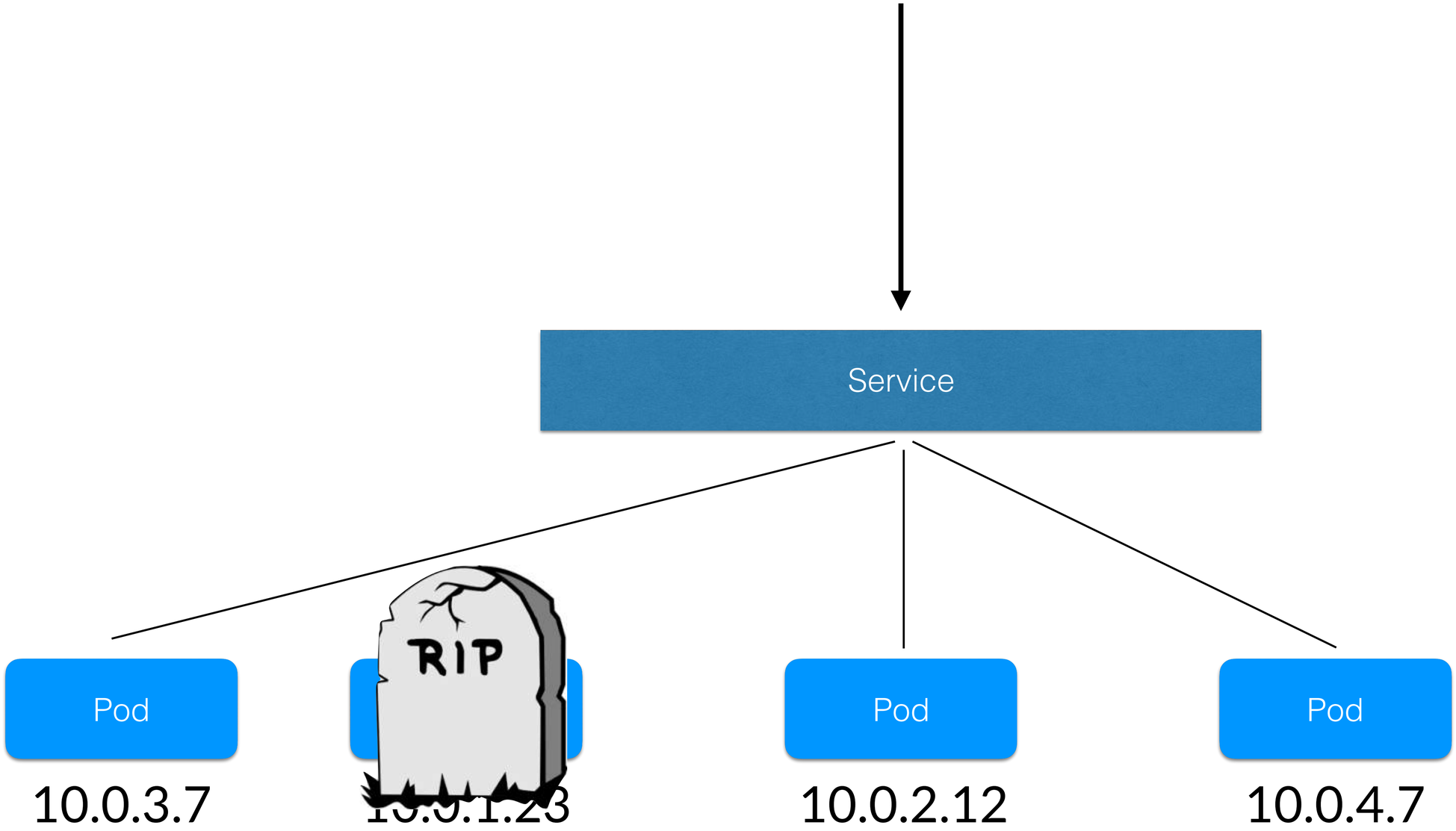
It has an IP inside K8S’ virtual network, and it exposes its ports there

REMEMBER: Pods are 100% volatile, all the time!


Service
Pod
Pod
Pod


Service
Pod
Pod
Pod
10.0.1.23
10.0.2.12
10.0.4.7


Service
RIP
Pod
Pod
10.0.2.12
10.0.4.7

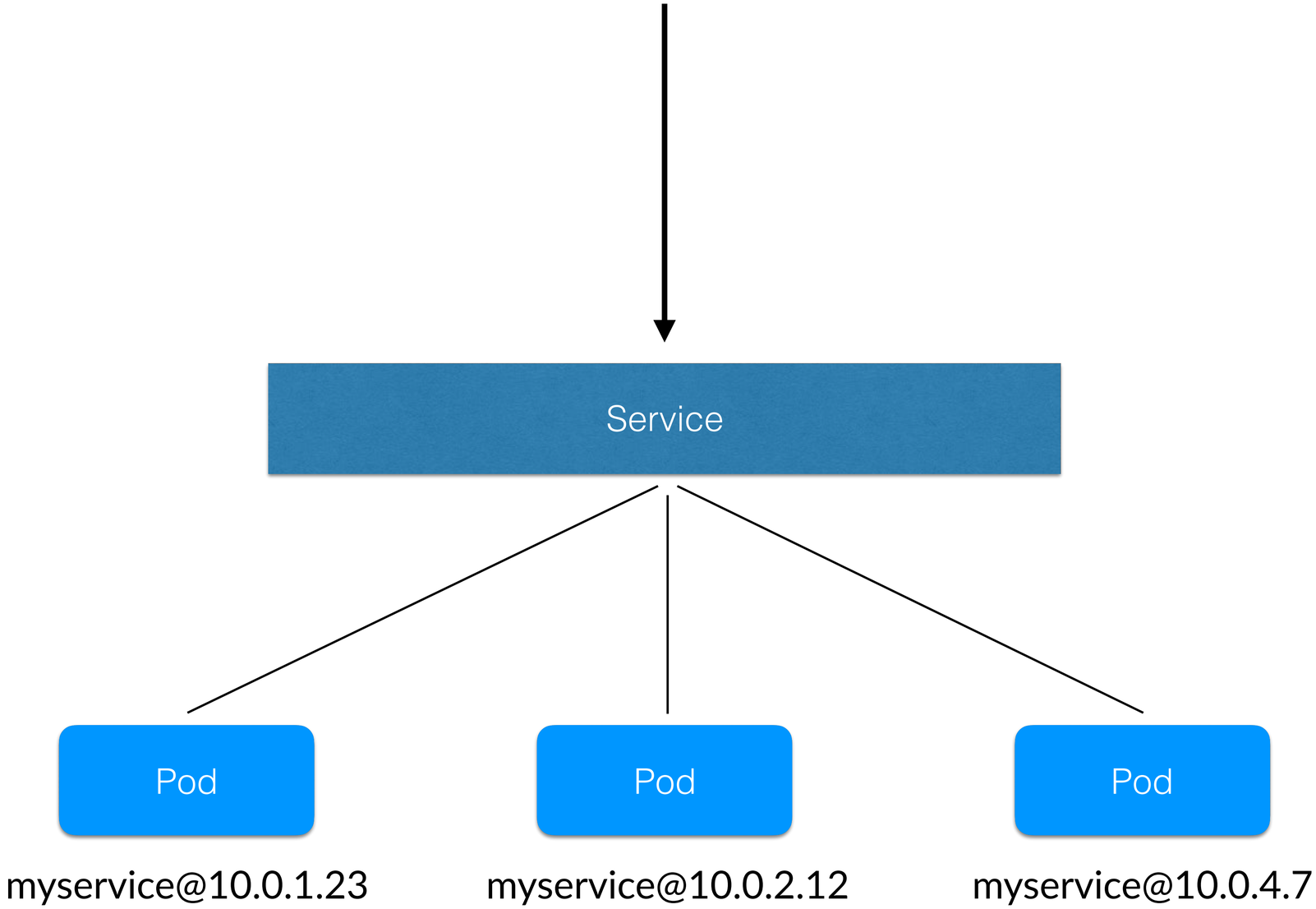

Service
Pod
RIP
Pod
Pod
10.0.3.7
10.0.1.23
10.0.2.12
10.0.4.7

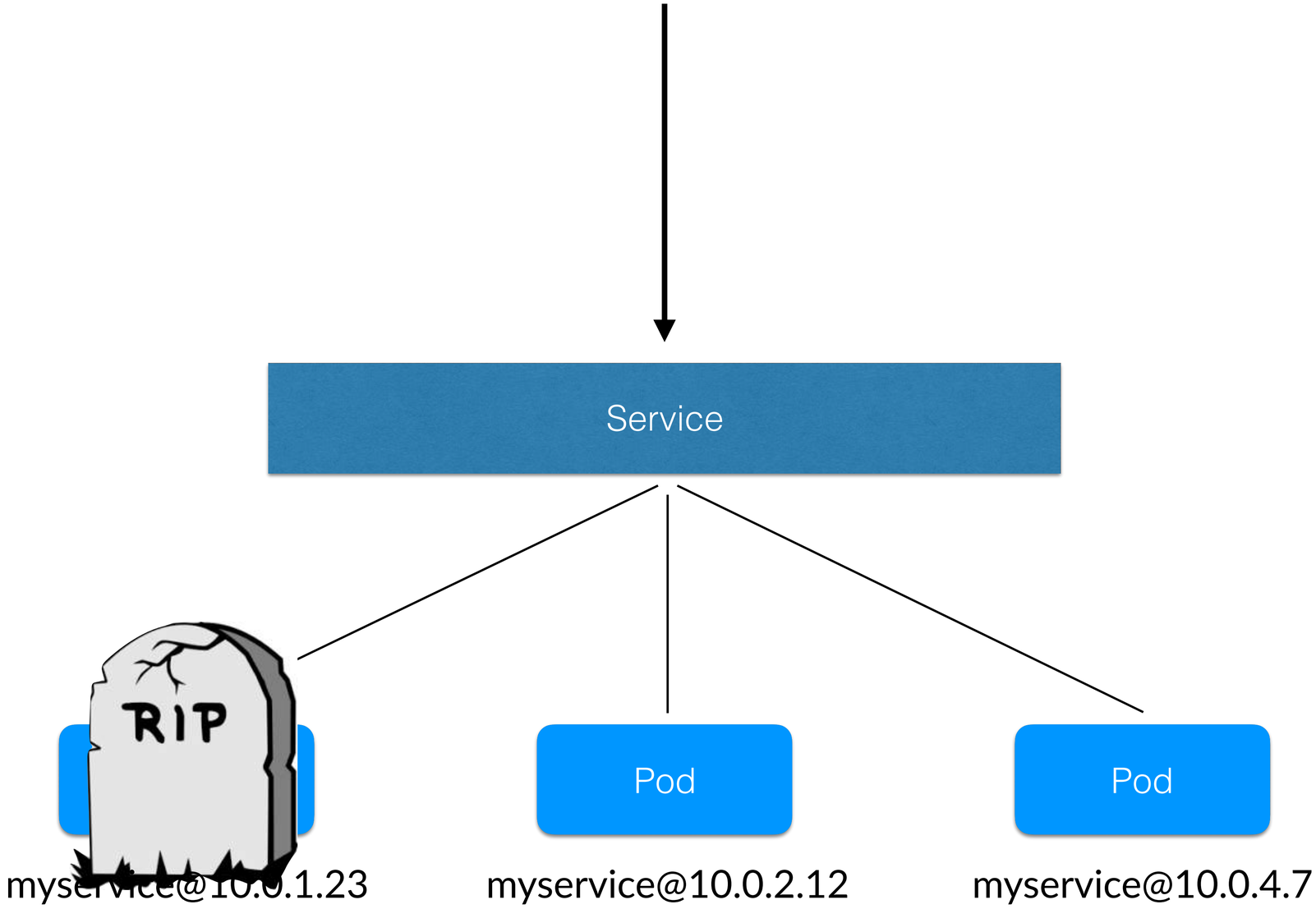
K8S strives to maintain the cluster state

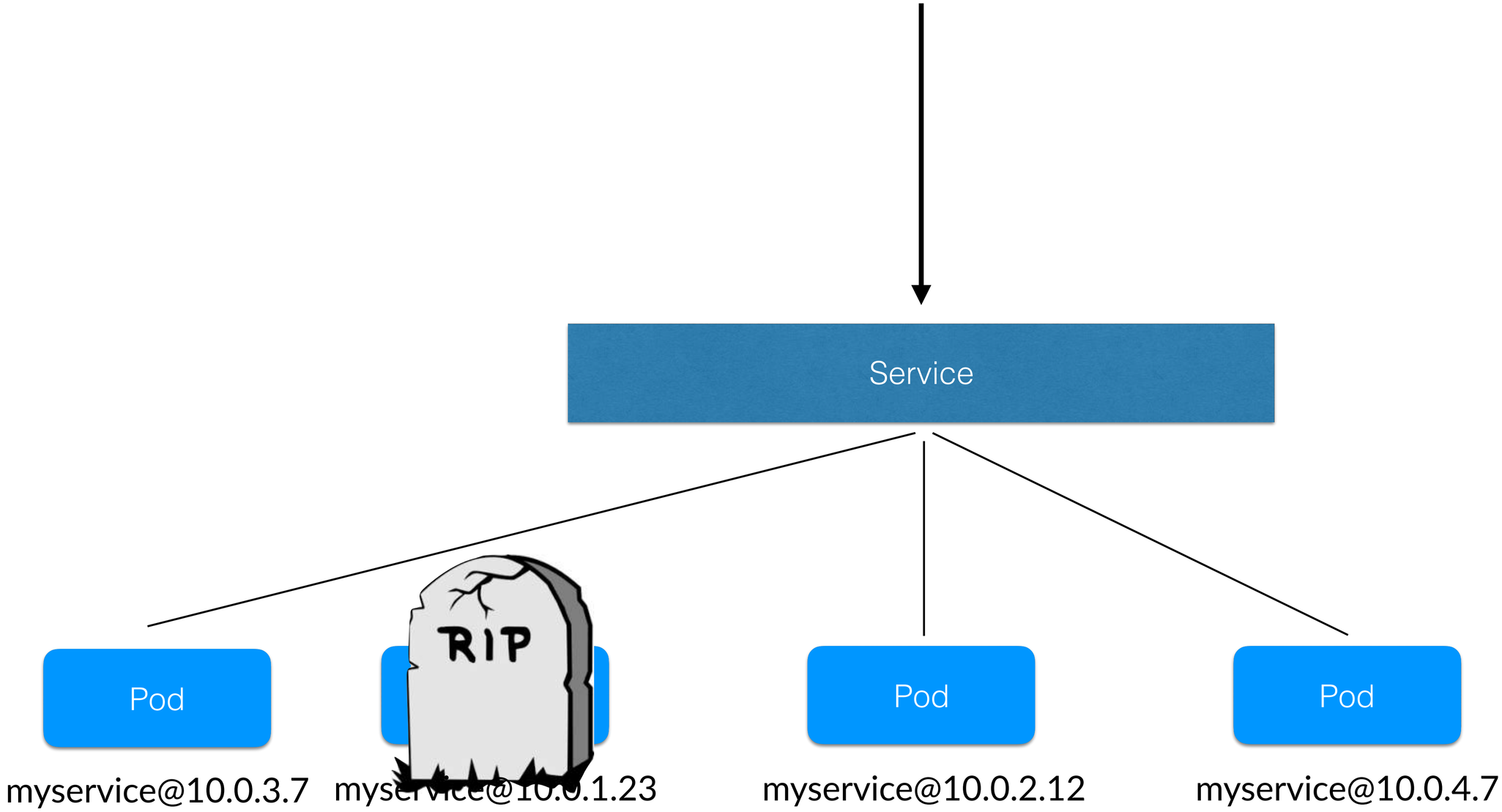
If you ask for X replicas, you'll get X replicas eventually

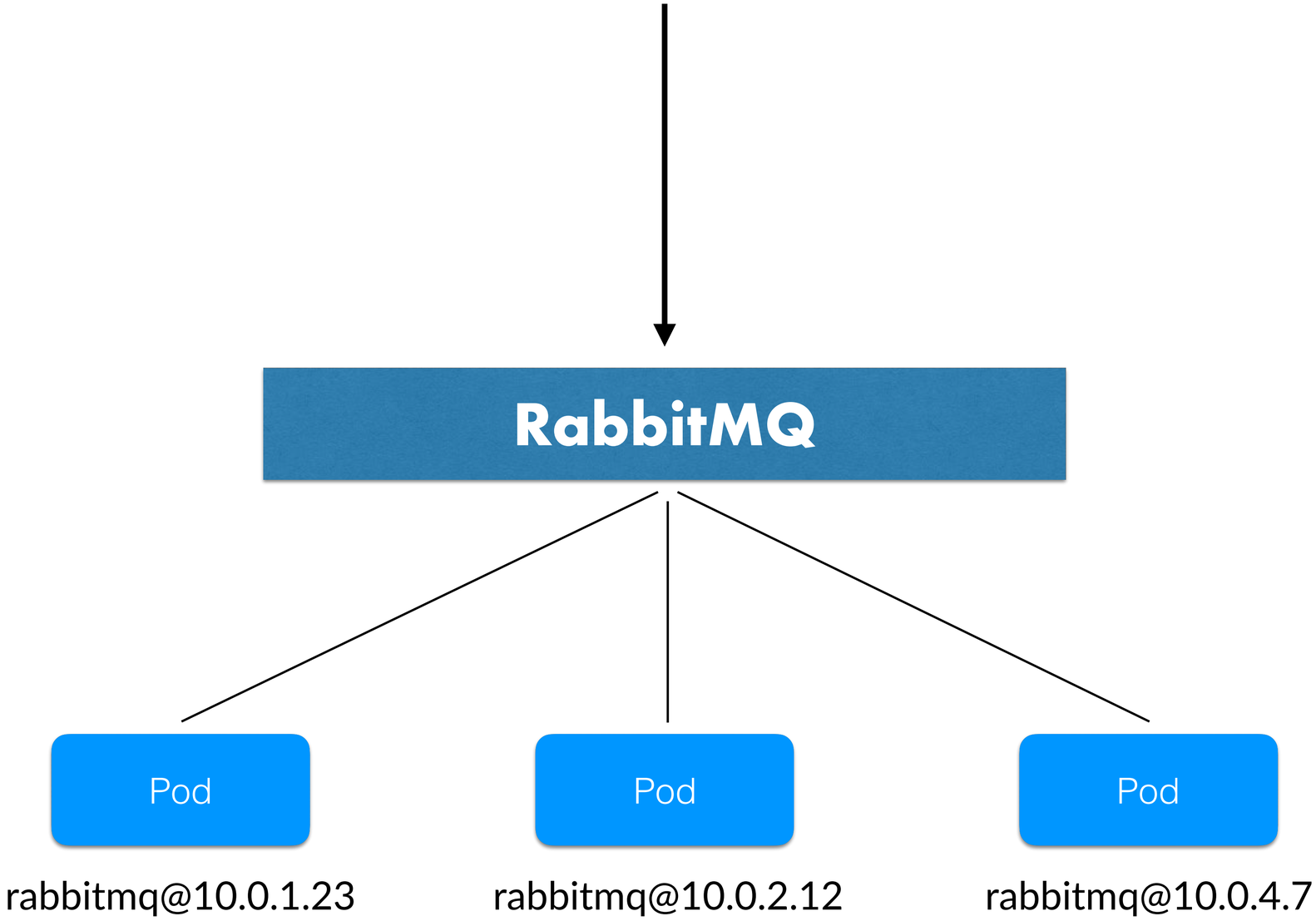
State is ALWAYS constructed using Pods (and secondary resources)

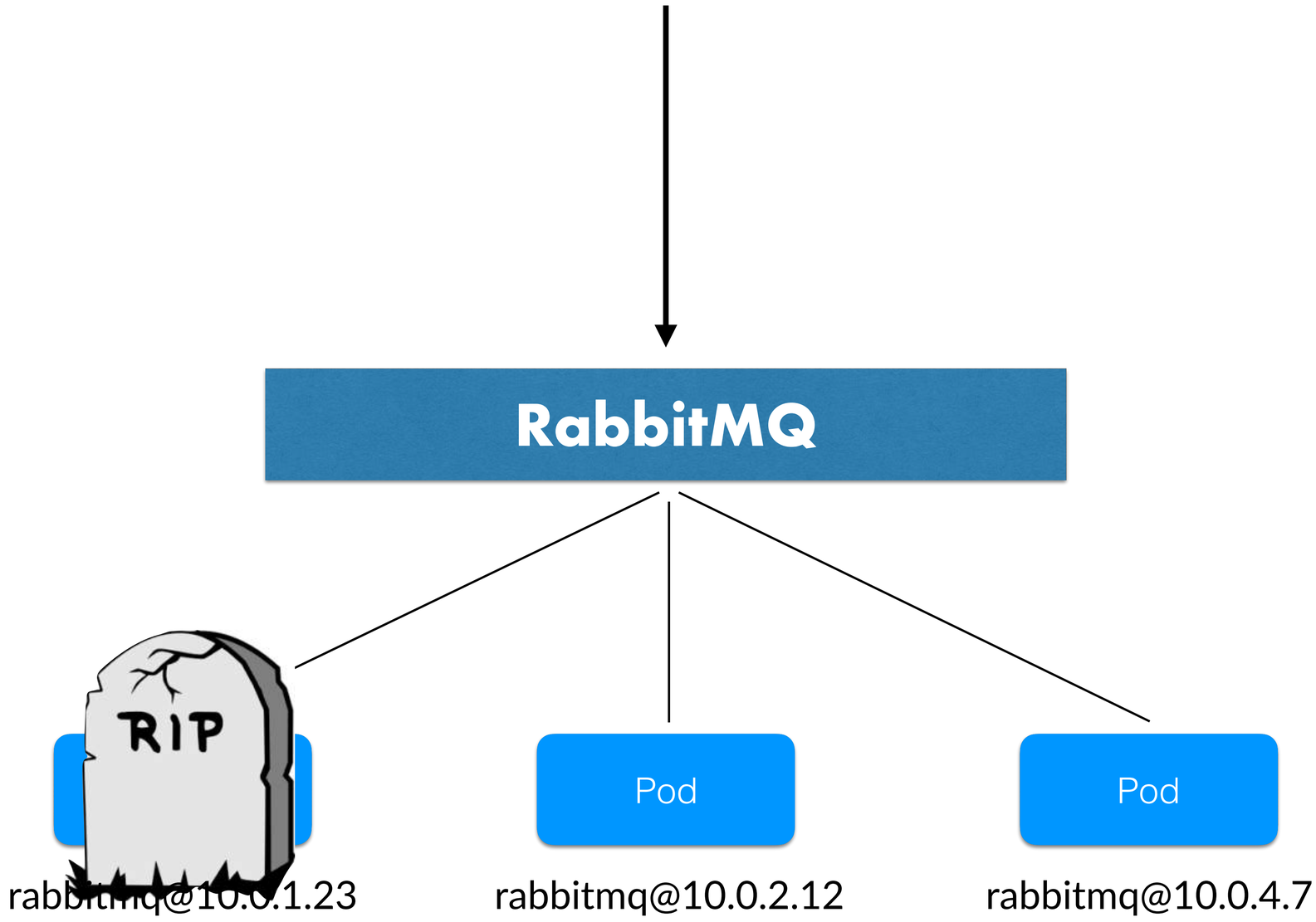
You can't make sure a Pod will always have the same IP

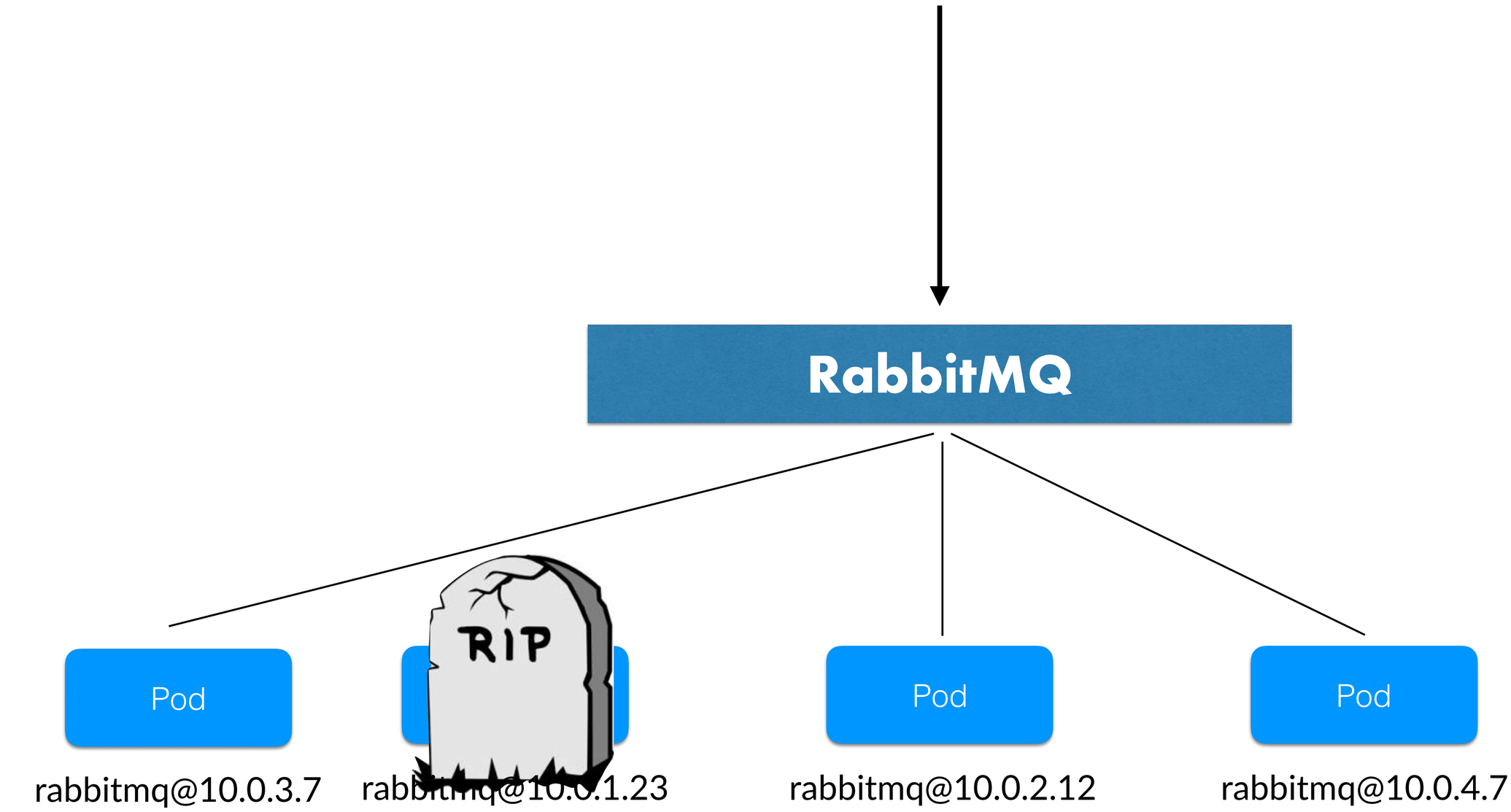

Service
Pod
Pod
Pod
myservice@10.0.1.23
myservice@10.0.2.12
myservice@10.0.4.7


Service
RIP
Pod
Pod
myservice@10.0.1.23
myservice@10.0.2.12
myservice@10.0.4.7


Service
Pod
RIP
Pod
Pod
myservice@10.0.3.7
myservice@10.0.1.23
myservice@10.0.2.12
myservice@10.0.4.7


RabbitMQ
Pod
Pod
Pod
rabbitmq@10.0.1.23
rabbitmq@10.0.2.12
rabbitmq@10.0.4.7


RabbitMQ
RIP
Pod
Pod
rabbitmq@10.0.1.23
rabbitmq@10.0.2.12
rabbitmq@10.0.4.7


RabbitMQ
Pod
RIP
Pod
Pod
rabbitmq@10.0.3.7
rabbitmq@10.0.1.23
rabbitmq@10.0.2.12
rabbitmq@10.0.4.7

You have requested a cluster state that really makes me think

Details

Name: rabbitmq

Namespace: astarte

Labels: app: rabbitmq state: really makes me think

Annotations: kubectl.kubernetes.io/last-applied-configuration

Creation Time: 2018-02-12T22:57 UTC

Images: rabbitmq:3.7

Status

Pods: 3 running

Whoops. Now Rabbit is really confused.

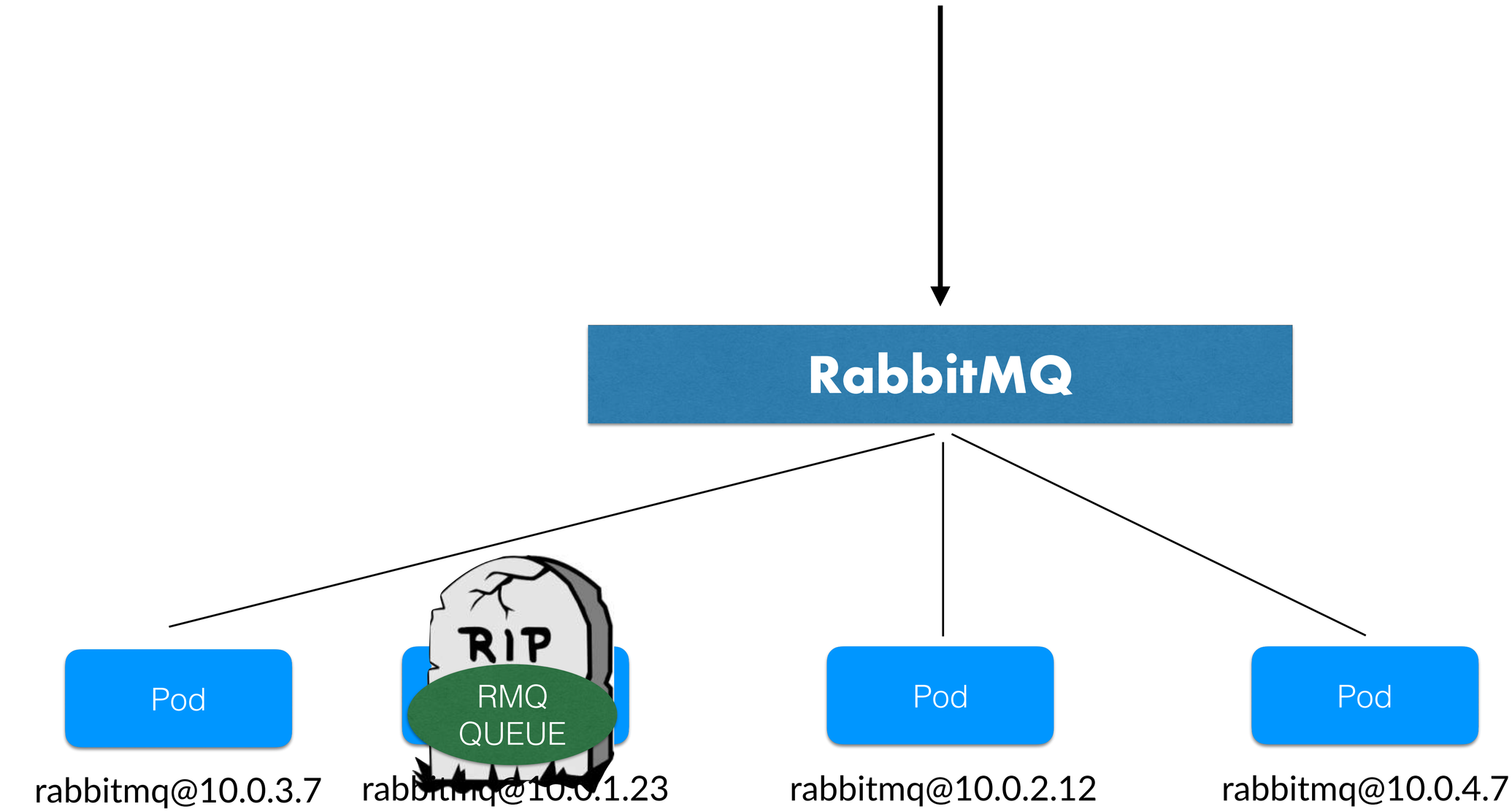
If you configured your cluster in suicide mode, your node is lost forever

(and everything it had with it)

If you didn’t (hopefully), now Rabbit is waiting for a node to come back…

… and it firmly believes a new node joined the cluster

(no, you can’t migrate it from where we are)


RabbitMQ
Pod
RIP
RMQ
QUEUE
Pod
Pod
rabbitmq@10.0.3.7
rabbitmq@10.0.1.23
rabbitmq@10.0.2.12
rabbitmq@10.0.4.7

Let’s get smart.

Pods don’t have deterministic IPs, but CAN have deterministic DNS

Erlang is cool enough to let us use a DNS in node name

K8S should self-heal our cluster, not erase its persistency or make it f*ing weird.

Checklist

1. Get our DNS name
2. Fix our Node Name
3. ???
4. Profit

(there's more, but not enough time :( )

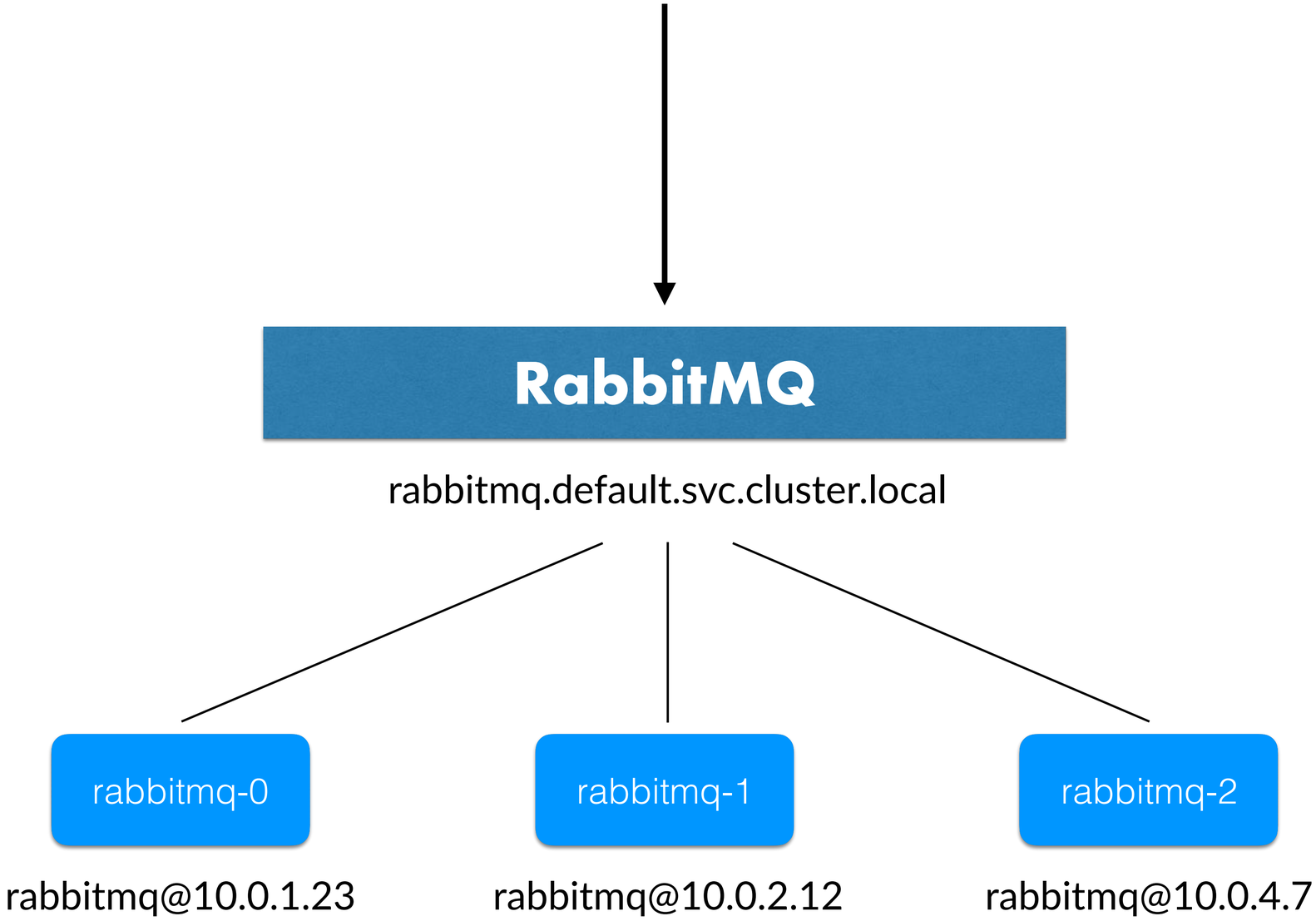
Enter StatefulSets

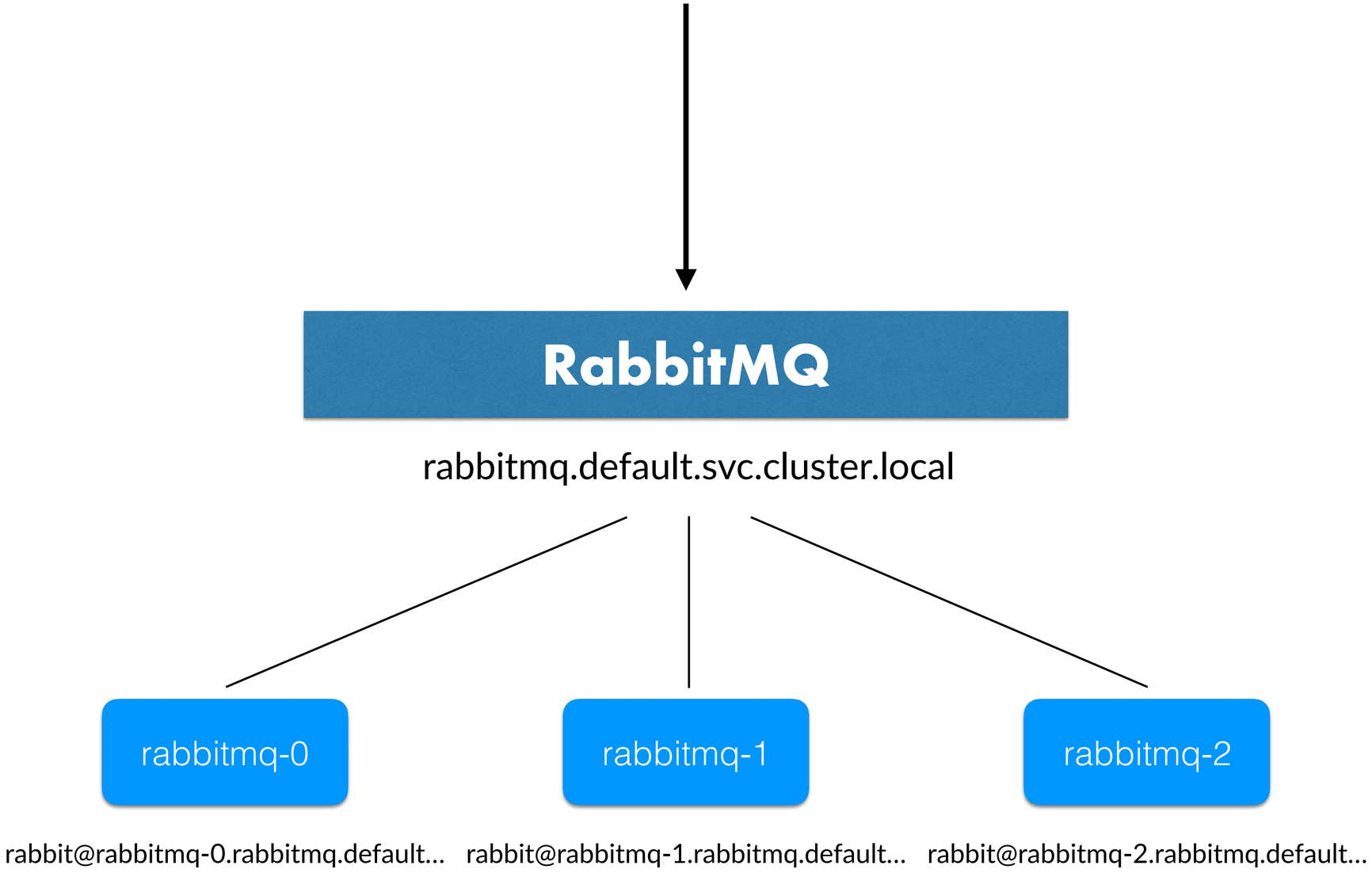
Why do I need this?

Persistent Storage. You can attach volumes in a deterministic name

ReplicaSet on steroids: Pods ALWAYS have the same name

They’re the foundation of our DNS determinism


RabbitMQ
rabbitmq.default.svc.cluster.local
rabbitmq-0
rabbitmq-1
rabbitmq-2
rabbitmq@10.0.1.23
rabbitmq@10.0.2.12
rabbitmq@10.0.4.7


RabbitMQ
rabbitmq.default.svc.cluster.local
rabbitmq-0
rabbitmq-1
rabbitmq-2
rabbit@rabbitmq-0.rabbitmq.default…
rabbit@rabbitmq-1.rabbitmq.default…
rabbit@rabbitmq-2.rabbitmq.default…

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: rabbitmq-config
  namespace: astarte
data:
  enabled_plugins: |
      [rabbitmq_management,rabbitmq_peer_discovery_k8s].
  rabbitmq.conf: |
      ## Clustering
      cluster_formation.peer_discovery_backend  = rabbit_peer_discovery_k8s
      cluster_formation.k8s.host = kubernetes.default.svc.cluster.local
      cluster_formation.k8s.hostname_suffix = .rabbitmq.astarte.svc.cluster.local
      cluster_formation.k8s.address_type = hostname
      cluster_formation.node_cleanup.interval = 10
      cluster_formation.node_cleanup.only_log_warning = true
      cluster_partition_handling = autoheal
      ## queue master locator
      queue_master_locator=min-masters
      ## enable guest user
      loopback_users.guest = false
```

Dude, my app is not RabbitMQ.

ConfigMaps for vm.args!

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: astarte-generic-erlang-configuration
  namespace: astarte
data:
  vm.args: |
      ## Name of the node
      -name ${RELEASE_NAME}@${MY_POD_NAME}.${MY_POD_BASE_DNS}

      ## Cookie for distributed erlang
      -setcookie ${ERLANG_COOKIE}

      # Enable SMP automatically based on availability
      -smp auto

---
```

```
[…]
        volumeMounts:
          - name: config-volume
            mountPath: /beamconfig
            readOnly: true
        env:
          # These are needed for tweaking the Erlang VM
          - name: RELEASE_CONFIG_DIR
            value: /beamconfig
          - name: REPLACE_OS_VARS
            value: "true"
          - name: MY_POD_NAME
            valueFrom:
              fieldRef:
                fieldPath: metadata.name
          - name: RELEASE_NAME
            value: astarte_pairing
          - name: ERLANG_COOKIE
            valueFrom:
              secretKeyRef:
                name: astarte-erlang-cookies
                key: astarte_pairing
    volumes:
      - name: config-volume
        configMap:
          name: astarte-generic-erlang-configuration
          items:
          - key: vm.args
            path: vm.args
```

There’s a lot more tricks, but that’s it.

To find out more of them, check us out!

(please. It’s really new and we need traction)

github.com/astarte-platform

Thanks!

dario@ispirata.com

github.com/astarte-project