Kubernetes for you

ENSIAS – IT-Holic

2nd of April

Karim Benzidane, Ph.D

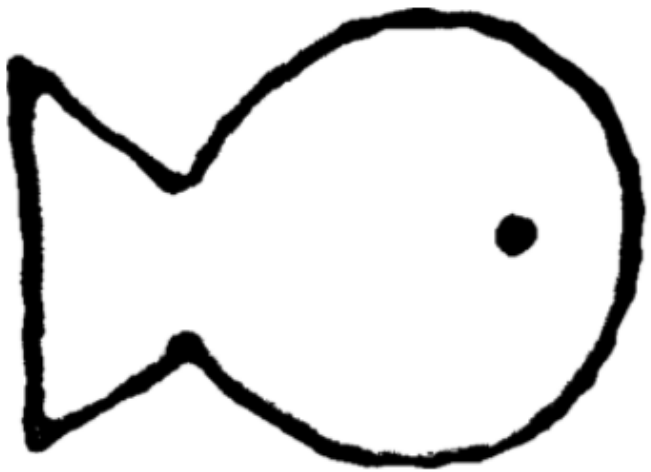
@The_Mirak

@cncg_casa

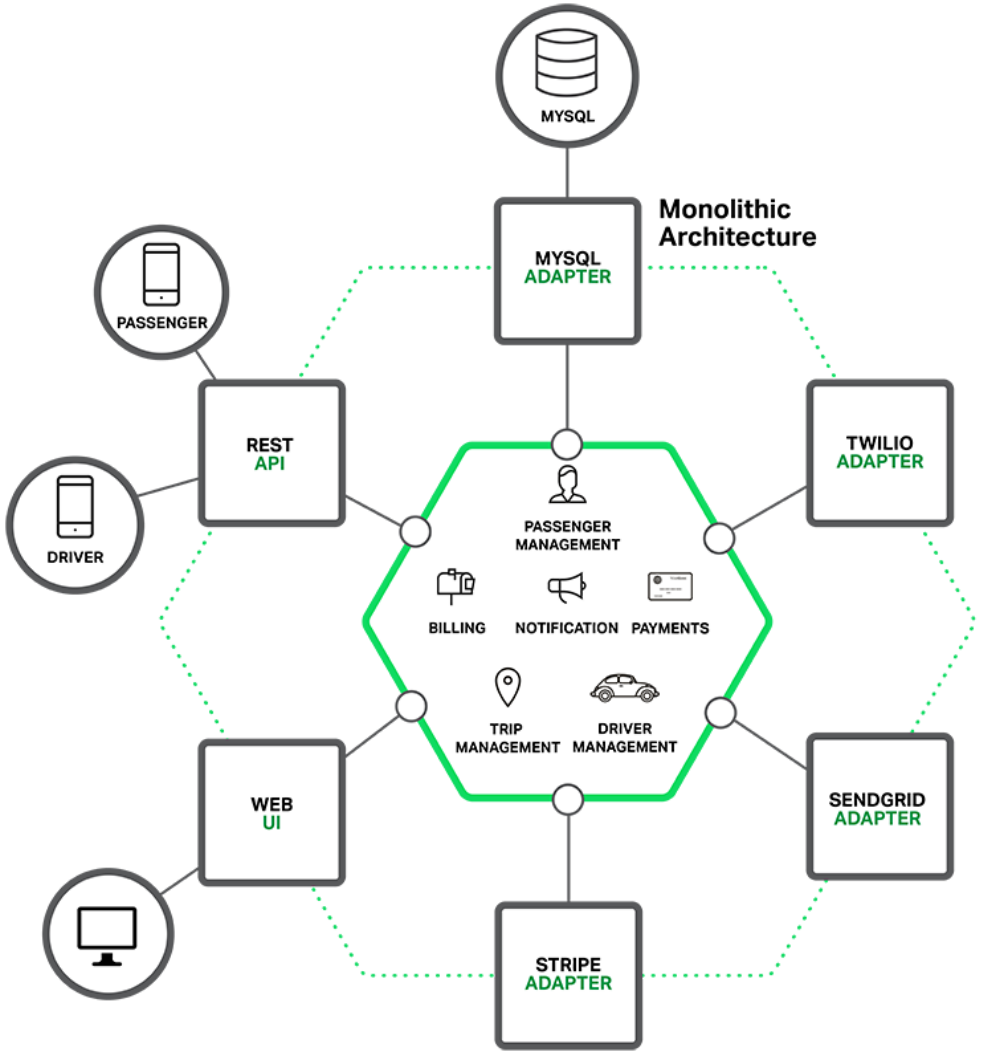
Join at

slido.com

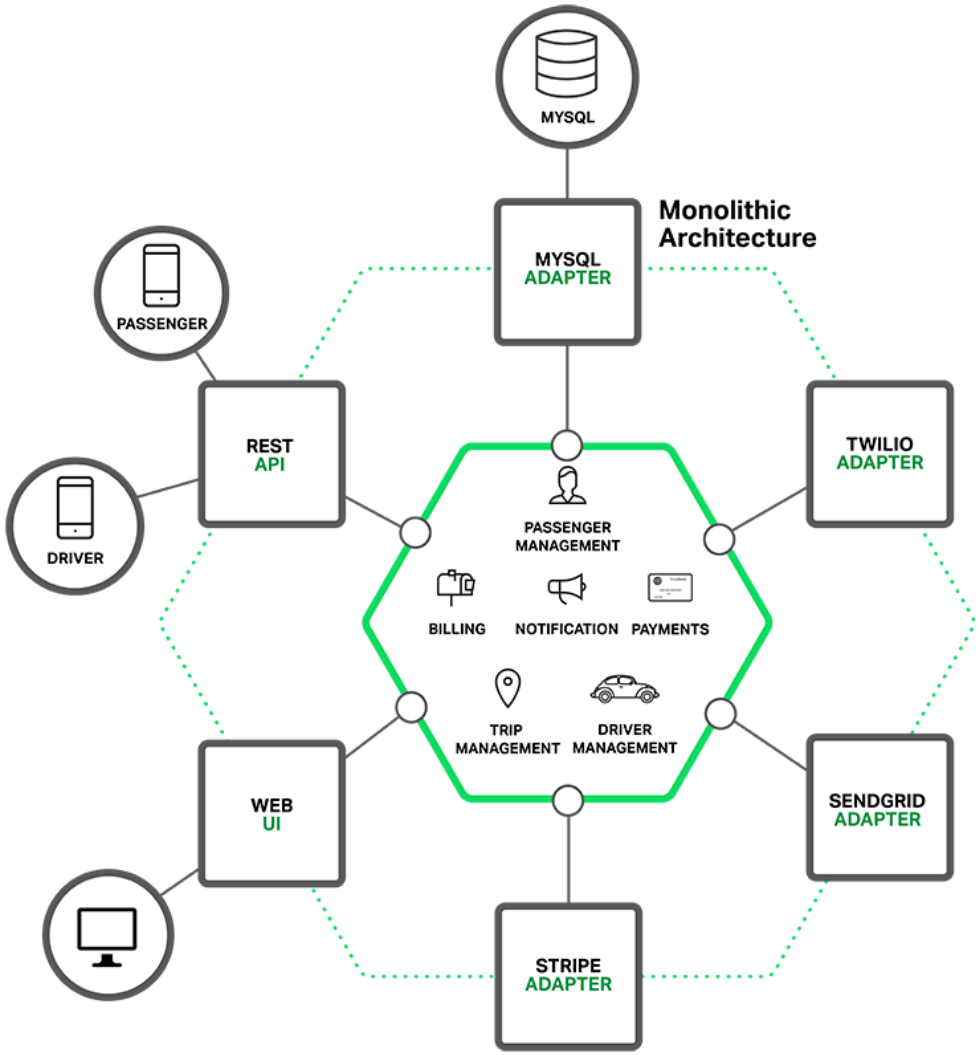
#316 667

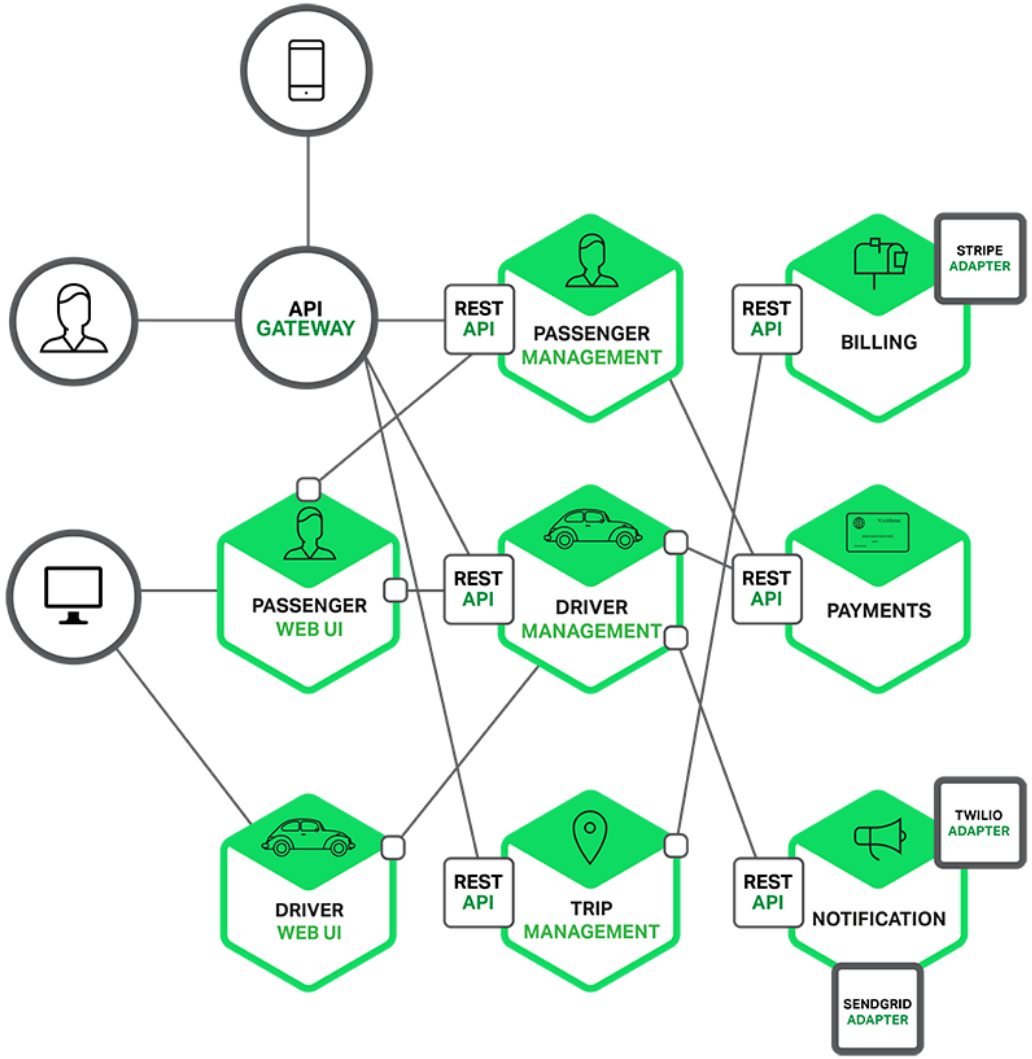


General Assumption

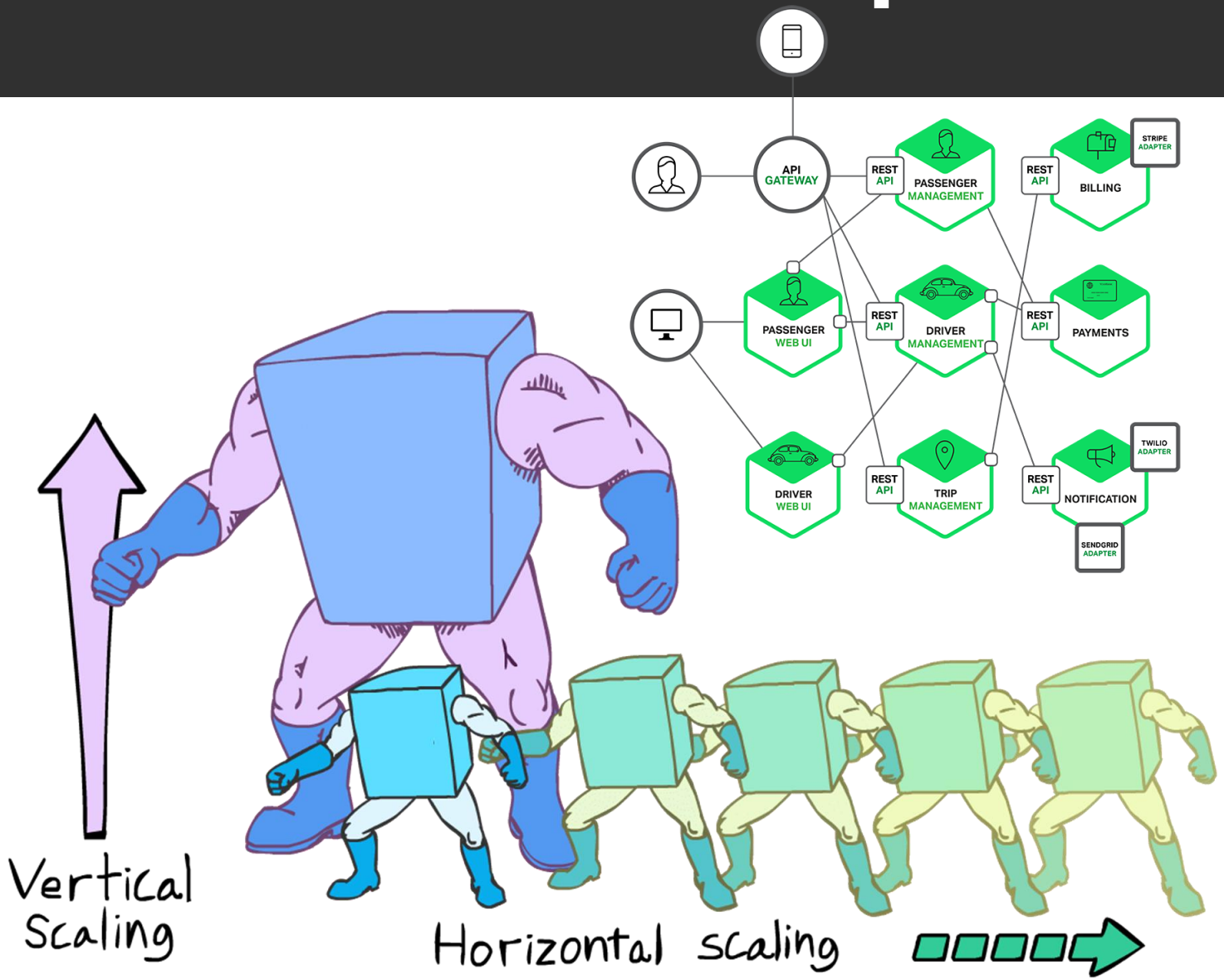


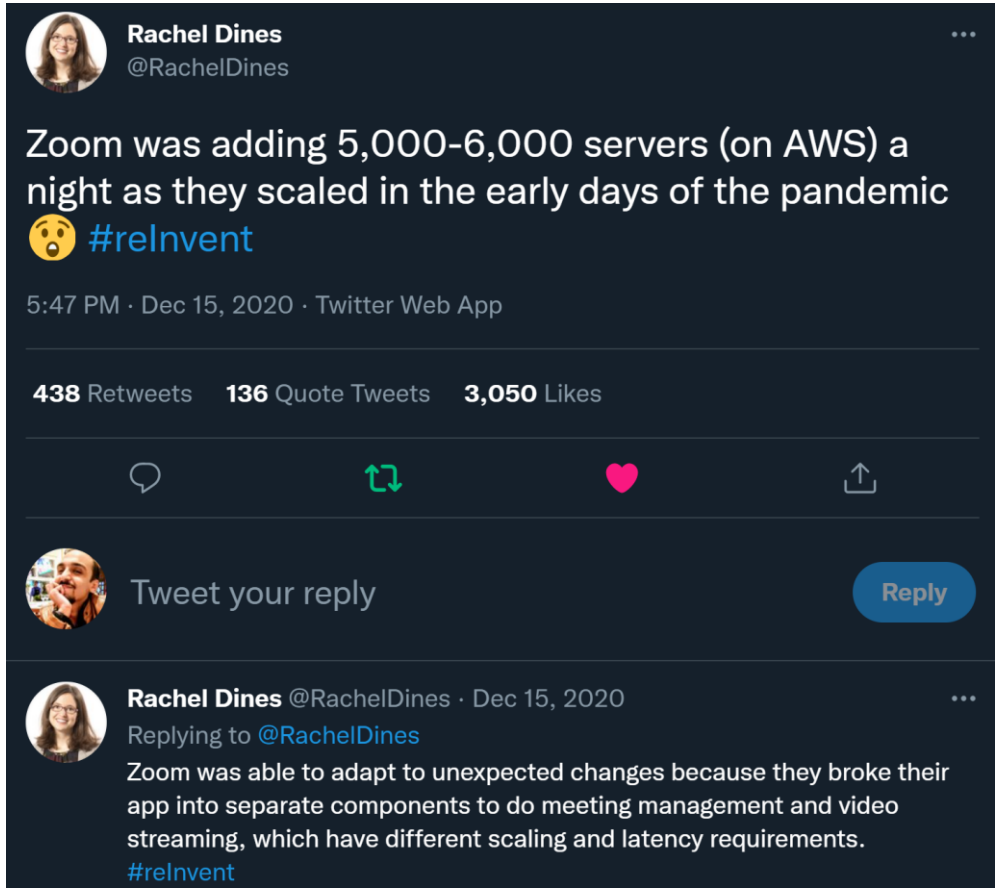
General Assumption

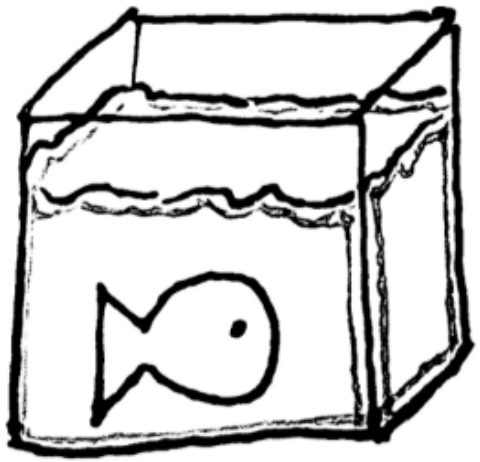




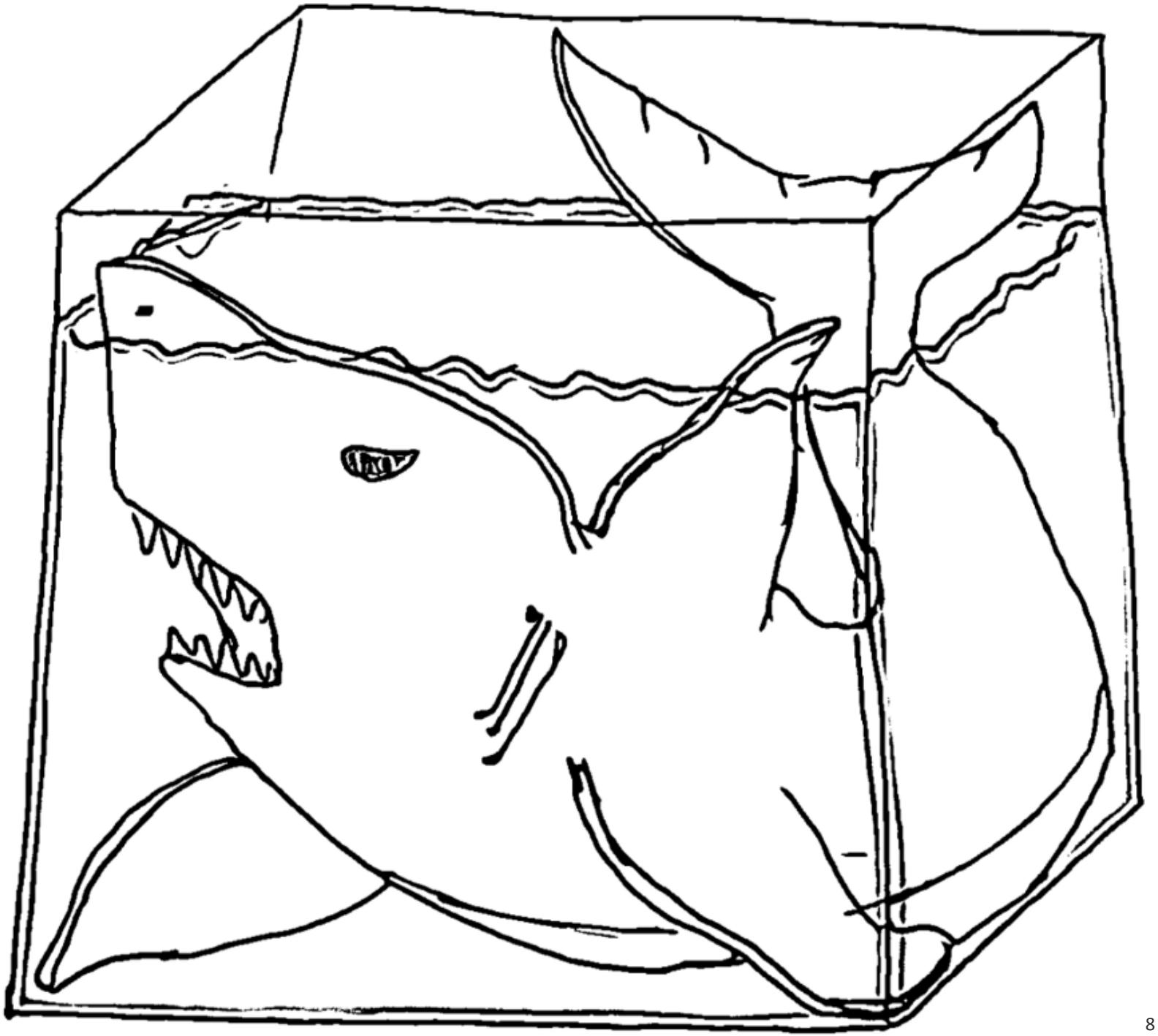
General Assumption







containers



General Assumption

- As always, microservices also have **drawbacks**.

- When the **number** of those components **increases**, deployment-related **decisions** become increasingly **difficult**
  - the number of deployment **combinations increase**,
  - the number of **inter-dependencies between** the components **increases** by an even greater factor.

General Assumption

- Microservices need to **find** and **talk** to **each other**.
- Someone or something **needs to configure** all of them **properly** to **enable** them to **work together** as a single system.
- It becomes increasingly **difficult** to **configure**, **manage**, and **keep** the **whole** system **running smoothly**.

General Assumption

- **Ideally**, we want **developers** to **deploy** applications **themselves without** having to **wait** for the **ops people**.

- But **deploying** an **application** often **requires** an **understanding** of the **underlying infrastructure**.

- Developers **don’t** always **know** those **details**.

General Assumption

- It’s much **harder** to **figure out where** to **put** each of those **components** to achieve **high resource utilization** and thereby keep the **hardware costs down**.

Doing all this manually is hard work, Automation is a must

- This **automation** includes **automatic scheduling** of those **components** to our servers, **automatic configuration**, **supervision**, and **failure-handling**.

In 2006 Google developed **cGroups**.

Today, everything at Google runs in a container.

On average we launch

5 billion

containers every week


RENA
MONROVIA
msc
RESCUE

In the mid 2000s Google developed **Borg**.

In the mid 2000s Google developed **Borg**.

In 2014 Google developed and open sourced **Kubernetes**

General Assumption

- After having kept **Borg** and **Omega secret** for a whole decade,

- In **2014** Google introduced **Kubernetes**, an **open-source** system based on the **experience** gained through **Borg**, **Omega**, and other **internal Google systems**.

General Assumption

This is where Kubernetes comes in.

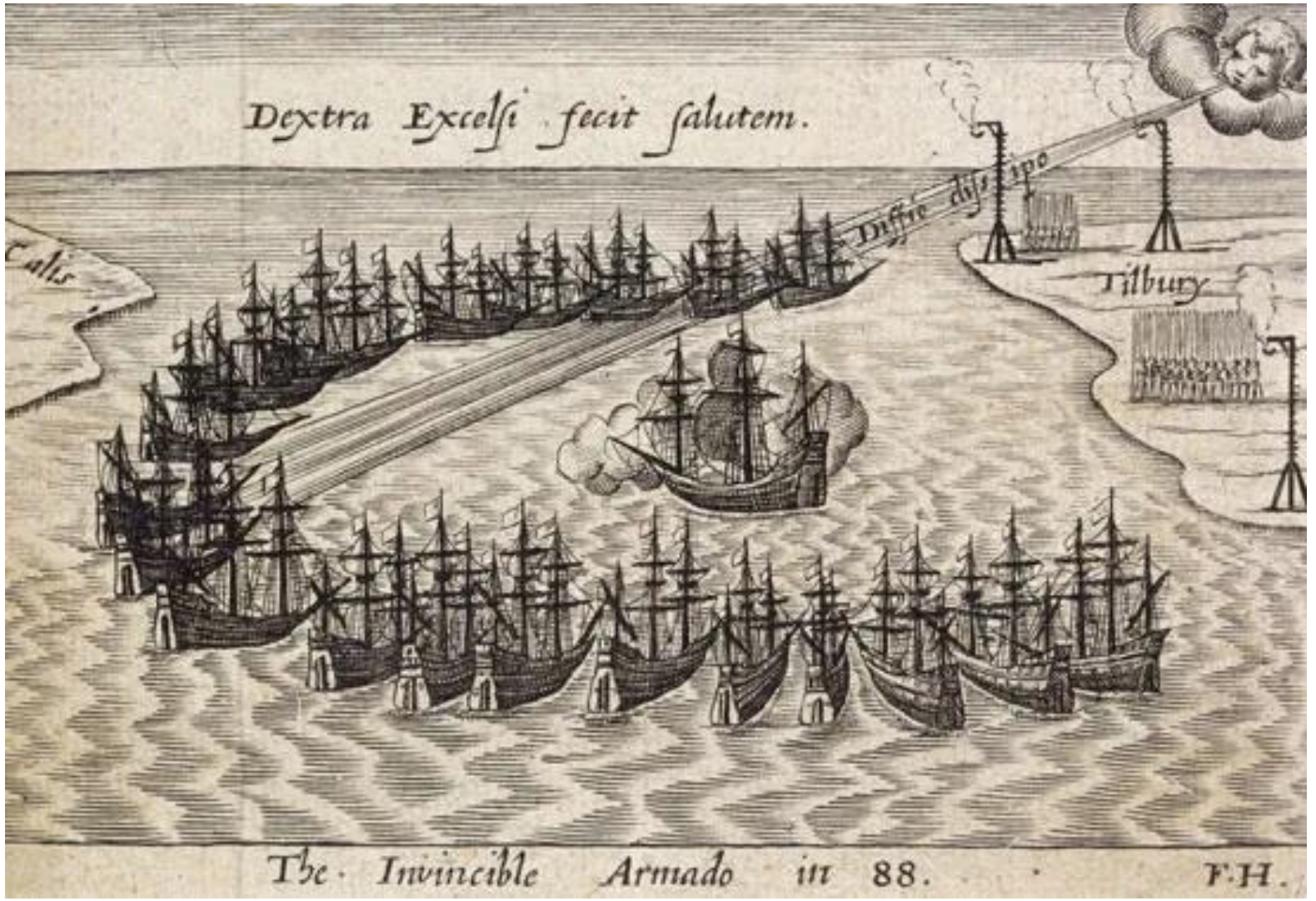
What is Kubernetes !?

Kubernetes is **Greek** for pilot or **helmsman**

What is Kubernetes !?

- A helmsman
  - **maintains** the **course** of the **ship**,
  - **carries out** the **orders given** by the **captain**
  - **reports back** the ship's heading.

- Kubernetes
  - **steers applications**
  - **reports** on their **status**



What is Kubernetes !?

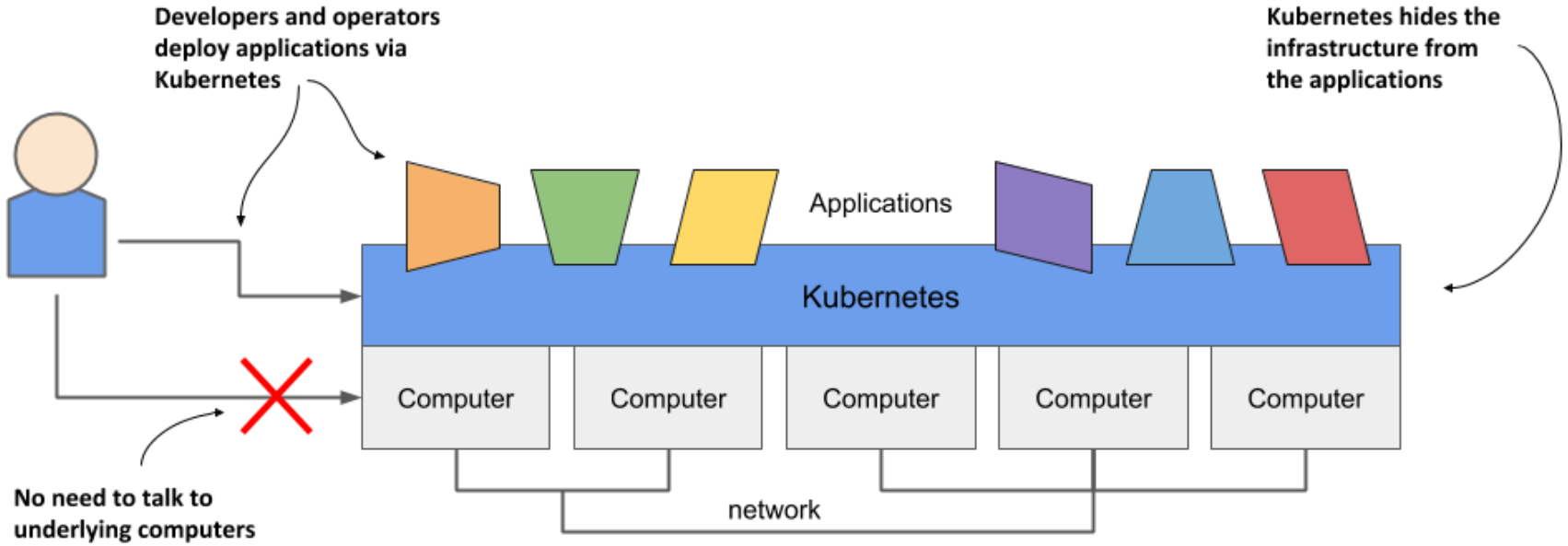
- Kubernetes enables **developers** to **deploy** their **applications themselves** and as often as they want

- The **focus** for **sysadmins shifts** from **supervising individual apps** to mostly **supervising** and **managing Kubernetes** and the **rest** of the **infrastructure**
  - while **Kubernetes** itself **takes care** of the **apps**.

What is Kubernetes !?

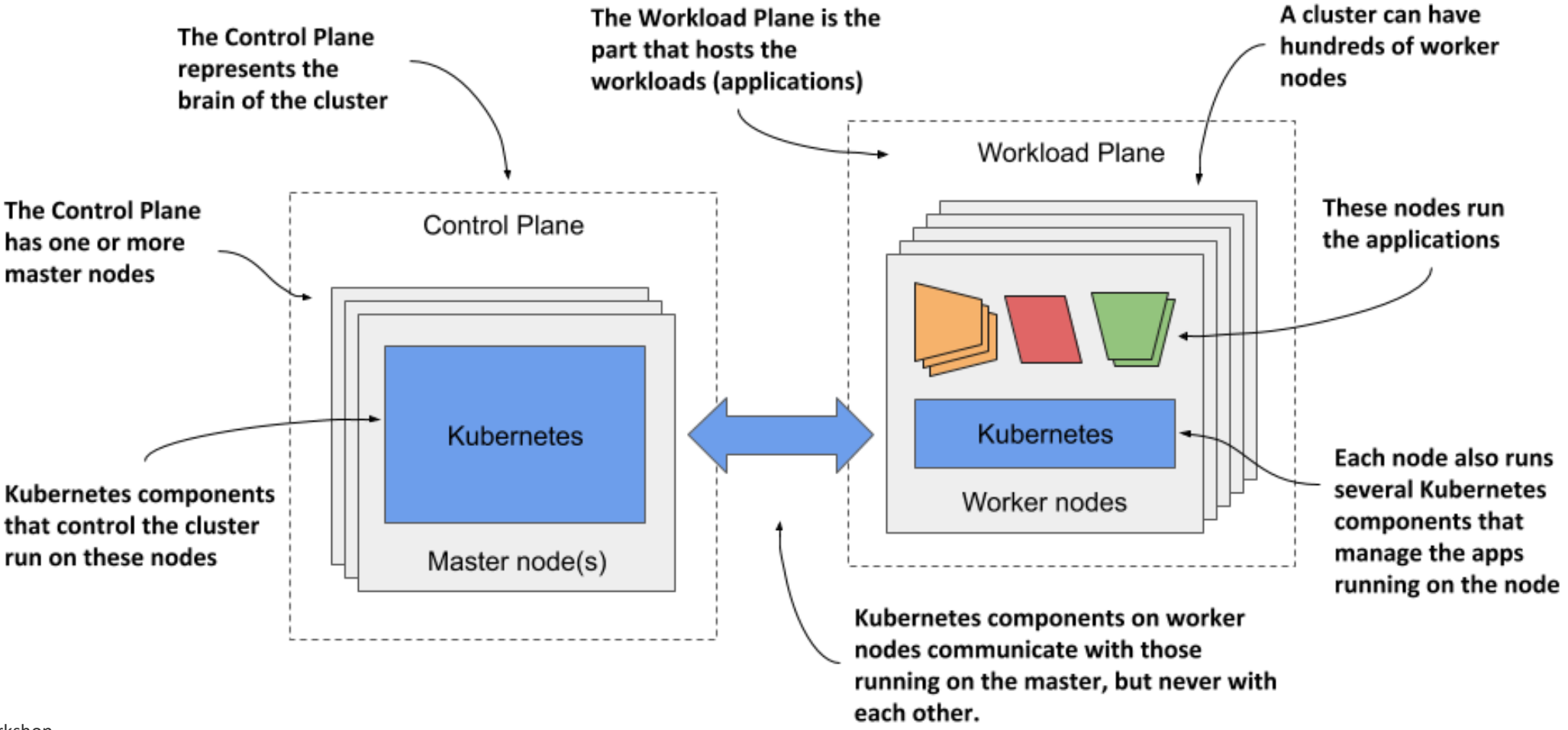
- It **abstracts** away the **underlying infrastructure**

- This **simplifies** the **development**, **deployment**, and **management** for both **development** and the **operations** teams.



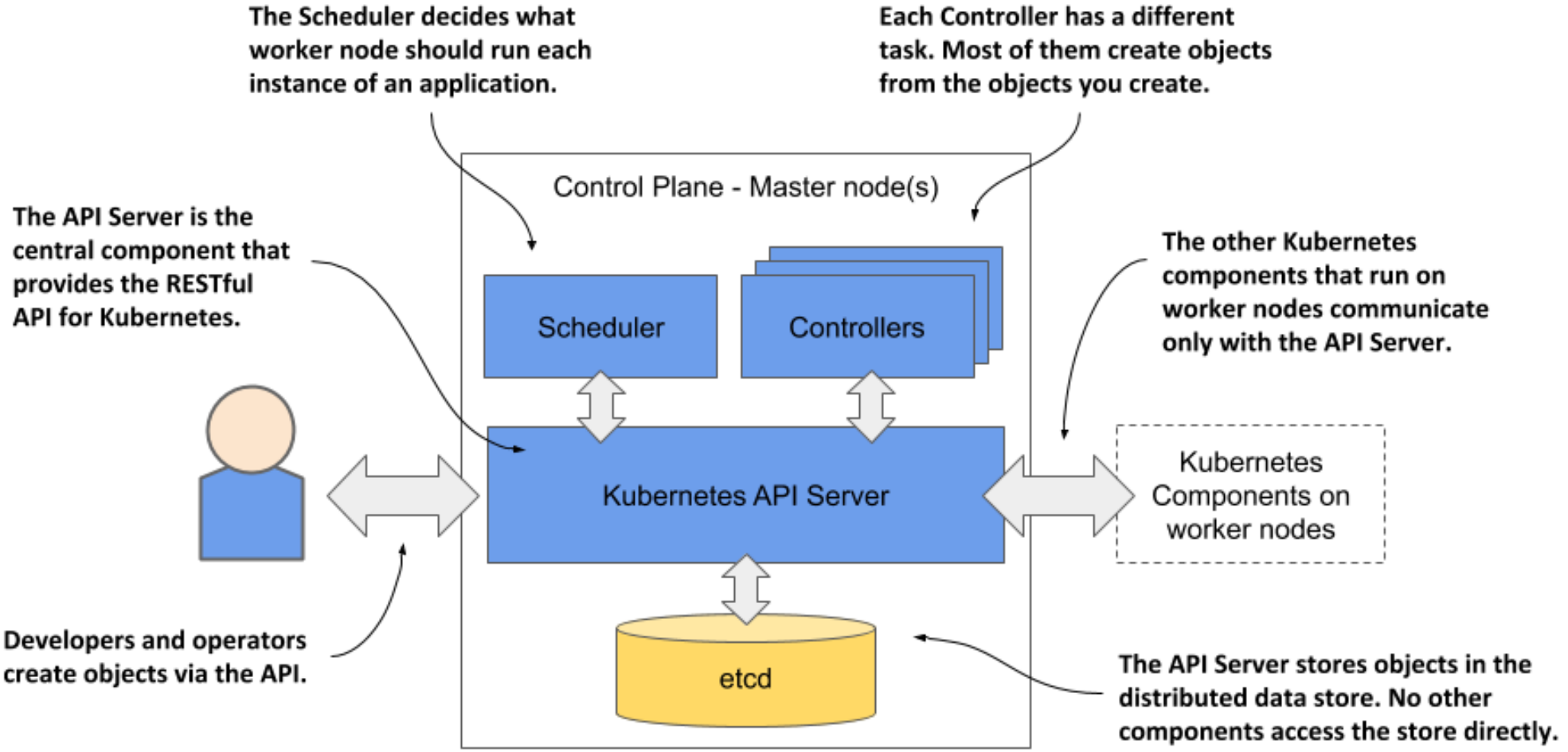
What is Kubernetes !?

Architecture overview



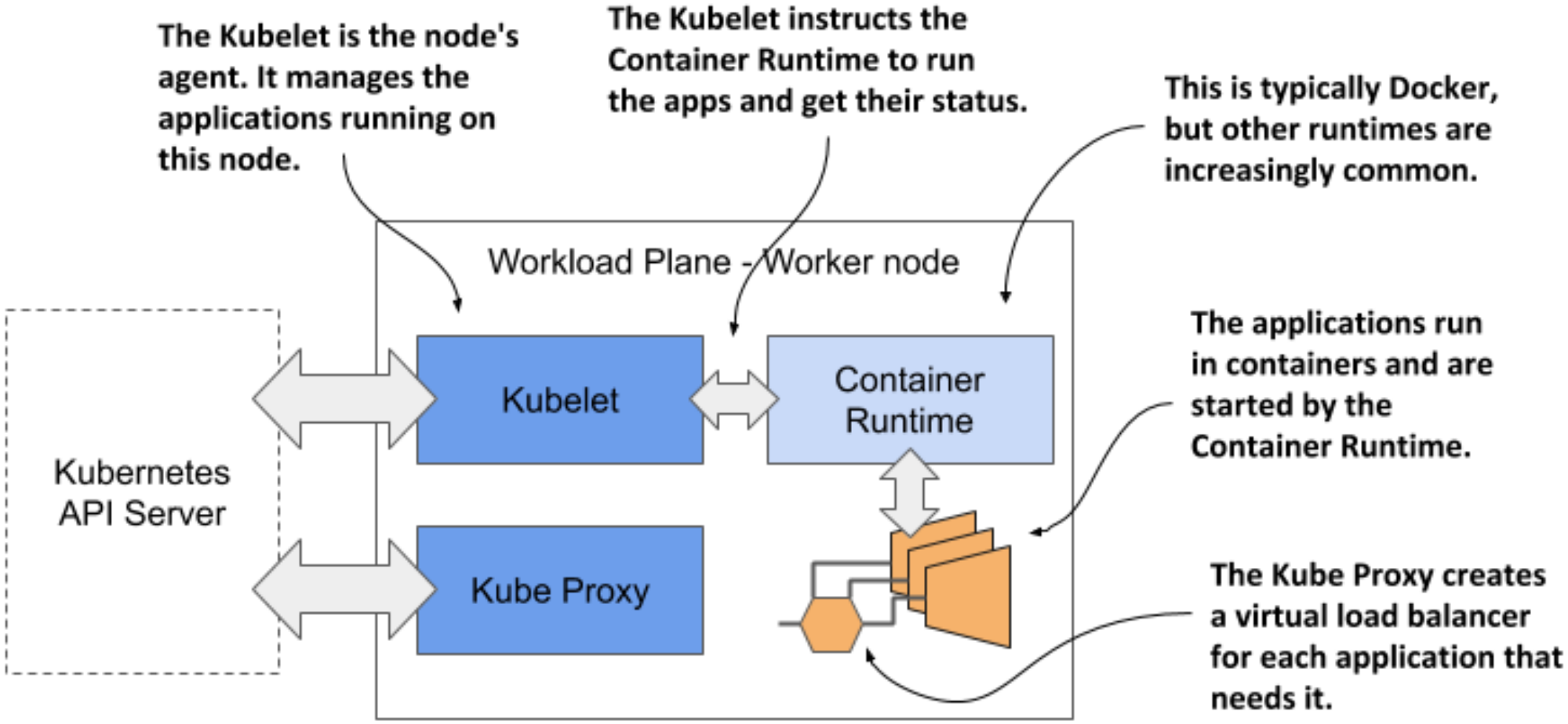
What is Kubernetes !?

The Control Plane

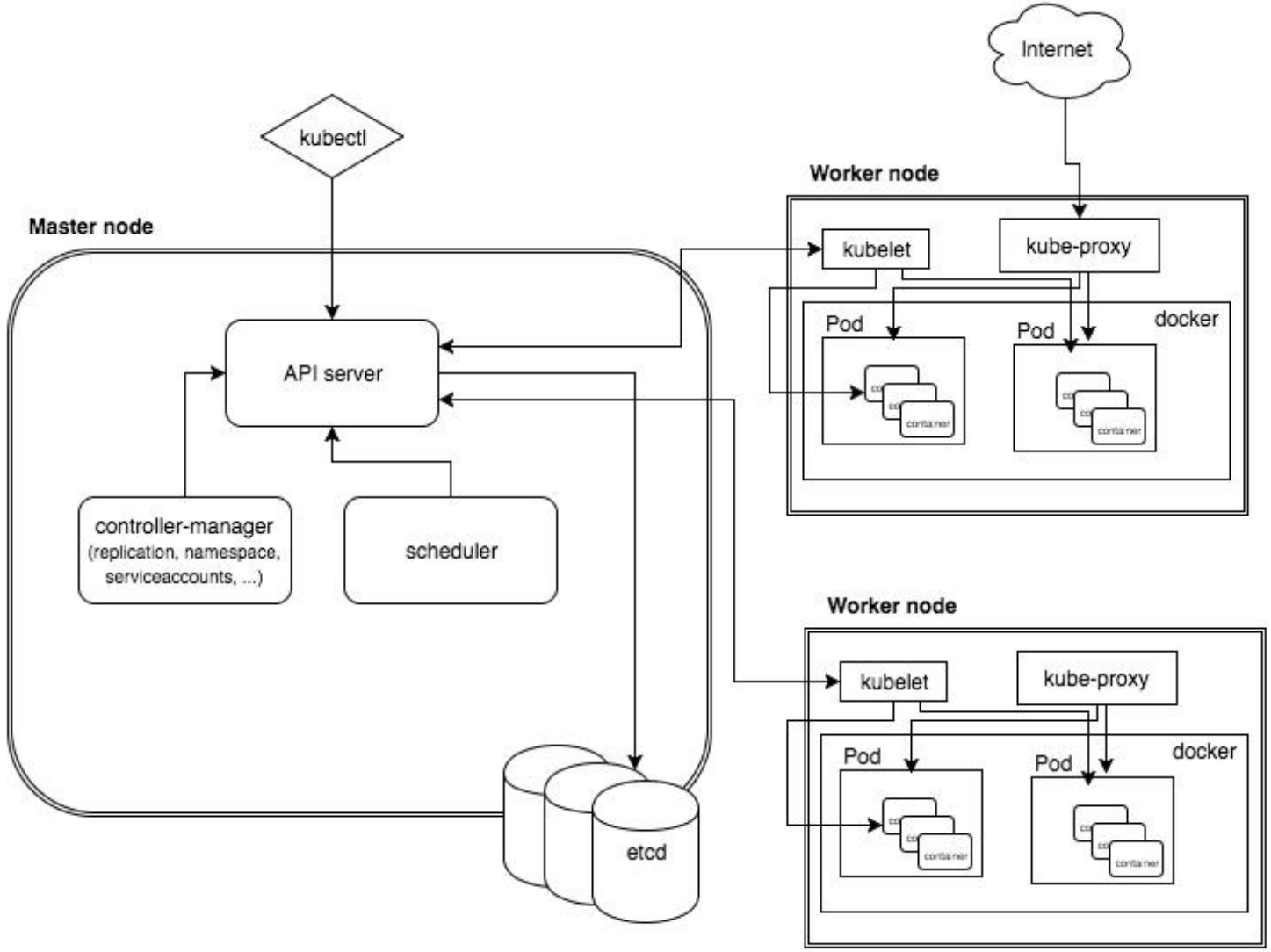


What is Kubernetes !?

The workload plane



What is Kubernetes !?



What is Kubernetes !?

The benefits

*"**Kubernetes** does the things that the very **best system administrator would do**: **automation**, **failover**, **centralized logging**, **monitoring**. It takes what we've learned in the DevOps community and makes it the default, out of the box.*

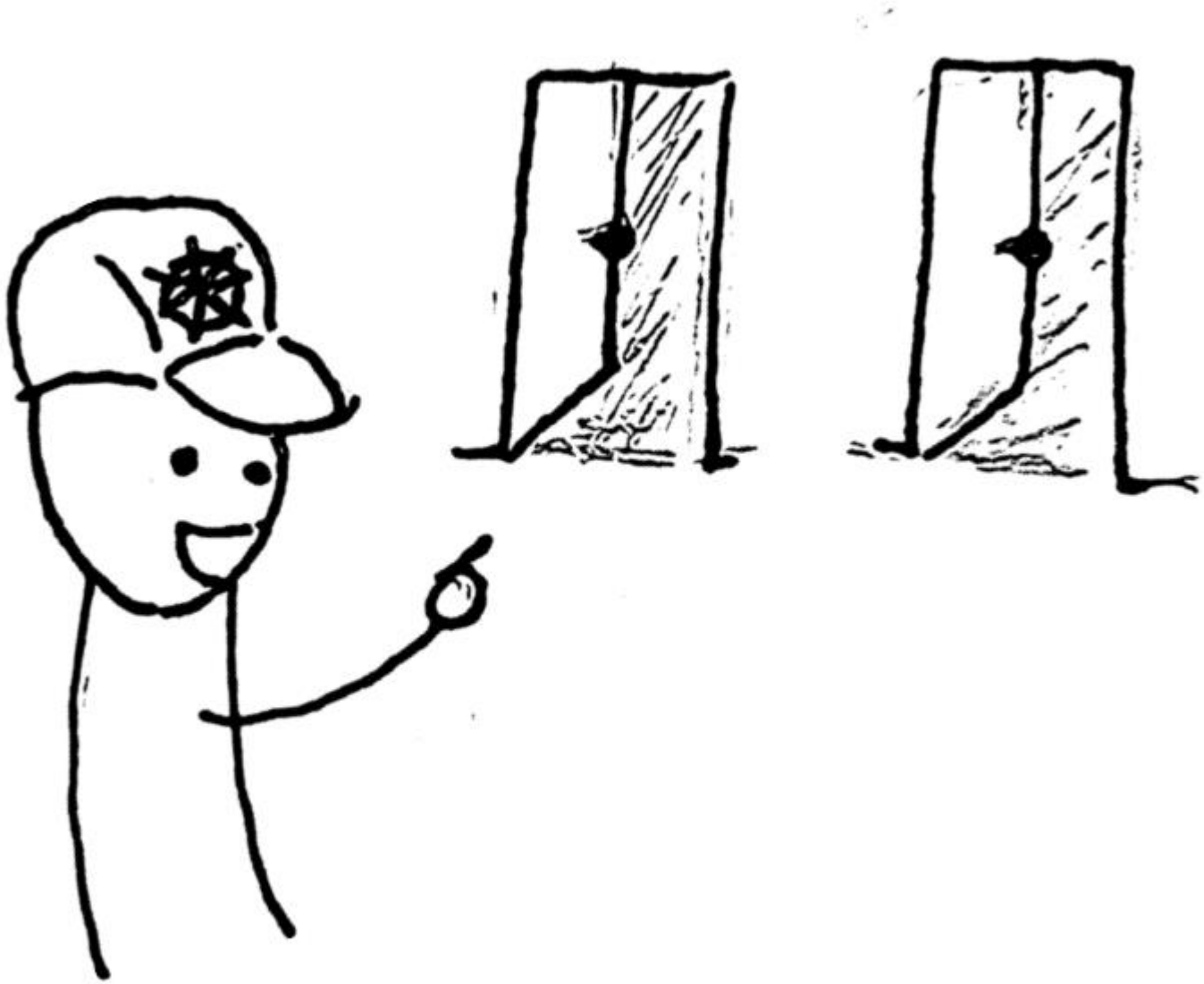
—Kelsey Hightower

What is Kubernetes !?

The benefits

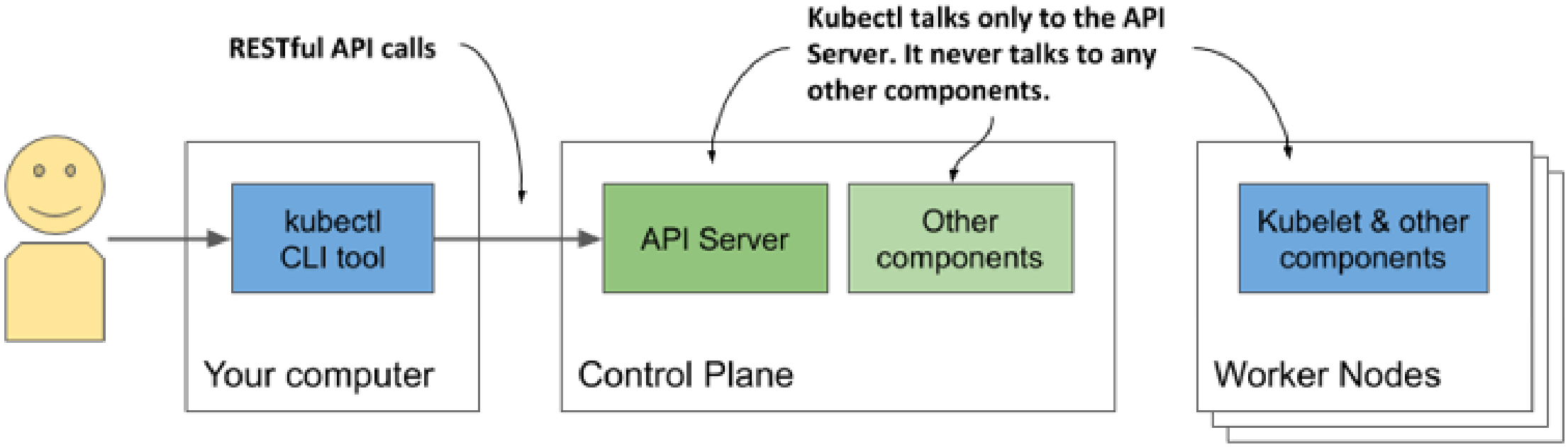
*“Everything **Fails** All the Time.*

—Werner Vogels



Interacting with Kubernetes

- **kubectl** is the **officially supported command-line** tool for accessing the Kubernetes API.
- It can be **installed** on **Linux**, **macOS**, or **Windows**.



Interacting with Kubernetes

Completion for kubectl

- kubectl command can also **output shell completion code** for **both** the **bash** and the **zsh shell**.

- It enables **tab completion** of **not only command** names but also the **object names**.

- With tab **completion**, things are much **easier**. Only **pressTAB** after **typing** the first few **characters** of **each token**

Interacting with Kubernetes

kubeconfig Configuration File

- Such configuration files are referred to as **kubeconfig** files.

- Note that **kubeconfig** is a **generic way** to **refer** to **kubectl** configuration files and that it is **not** the **name** of the **config file**.

- kubectl uses such files to **store** the **information needed** for us to **choose** a **cluster** and **communicate** with its **API** server.

Interacting with Kubernetes

kubeconfig Configuration File

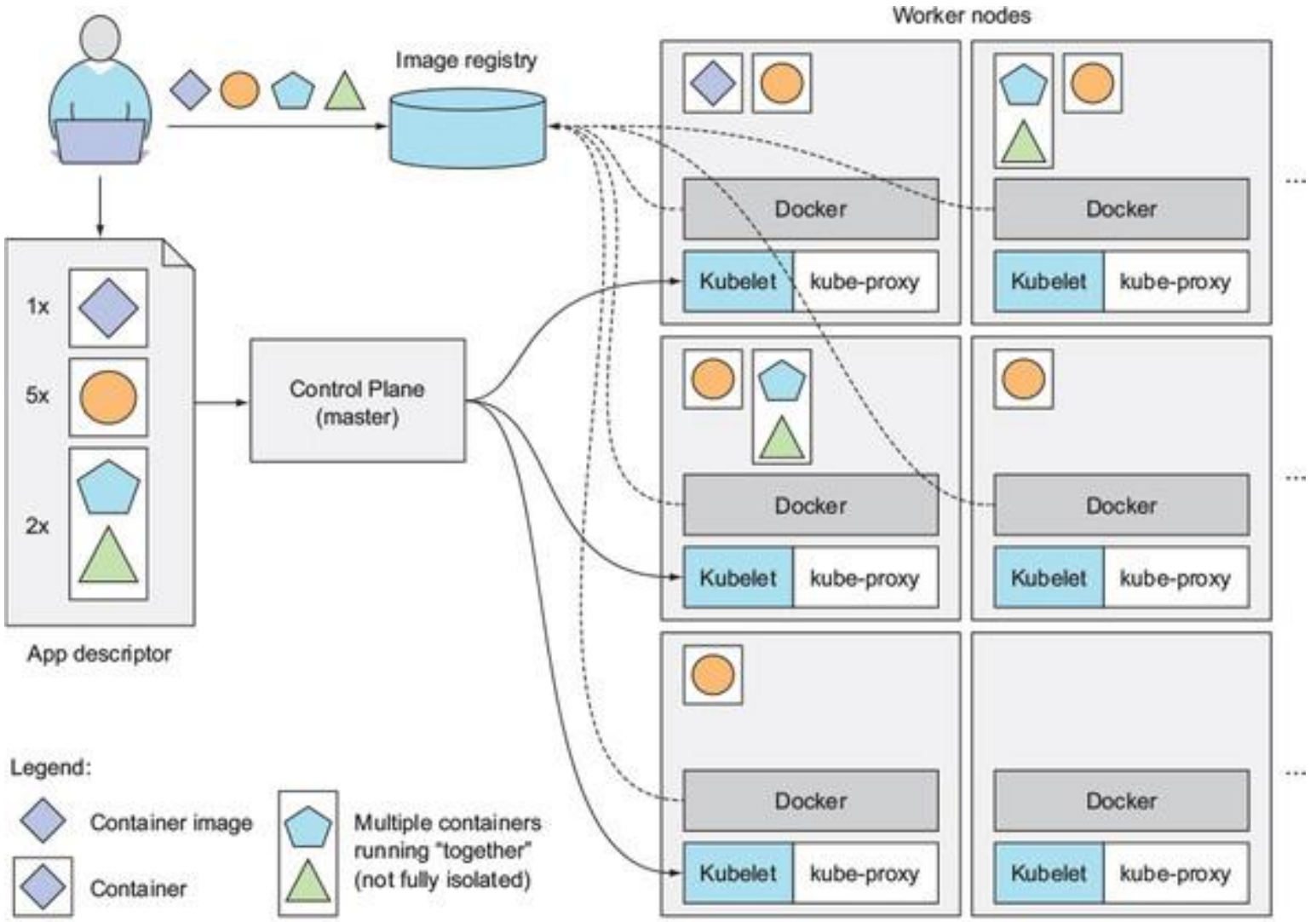
- By **default**, kubectl looks for the file in the **$HOME/.kube directory.**

- In most scenarios, specify a **KUBECONFIG** environment variable or use the **--kubeconfig** flag to specify the **kubeconfig** files.

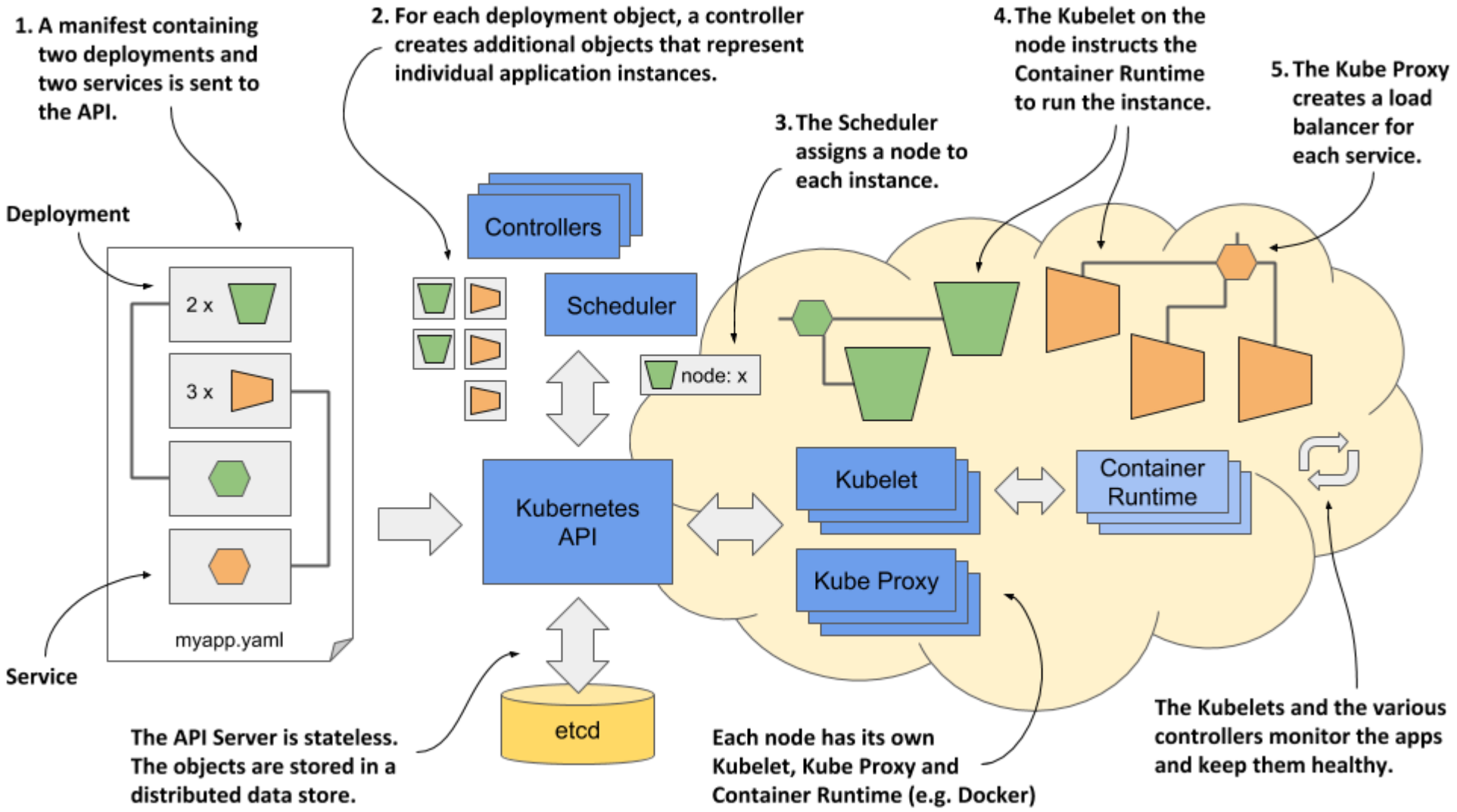
Running an application in Kubernetes

- **Everything** in Kubernetes is **represented** by an **object**.
- These **objects** can be **created** and **retrieved** via the **Kubernetes API**.
- These **objects** are usually **defined** in one or more **manifest** files in either **YAML** or **JSON format**.

Running an application in Kubernetes



Running an application in Kubernetes



Running an application in Kubernetes

- Kubernetes **continuously makes sure** that the **deployed state** of the **application** always **matches** the **provided description**.

- Kubernetes also **reports** the **status** of the **application** by **updating** the **object** that **represents** the application **instance**.

- The other **controllers monitor** these **objects** and **ensure** that **applications** are **moved** to **healthy nodes if** their **nodes fail**.

Running an application in Kubernetes

- Kubernetes **allows clients** to easily **find containers** that **provide** a **specific service**

- Kubernetes will **expose all** of them at a **single static IP** address and **expose** that **address** to all **applications running** in the **cluster**.

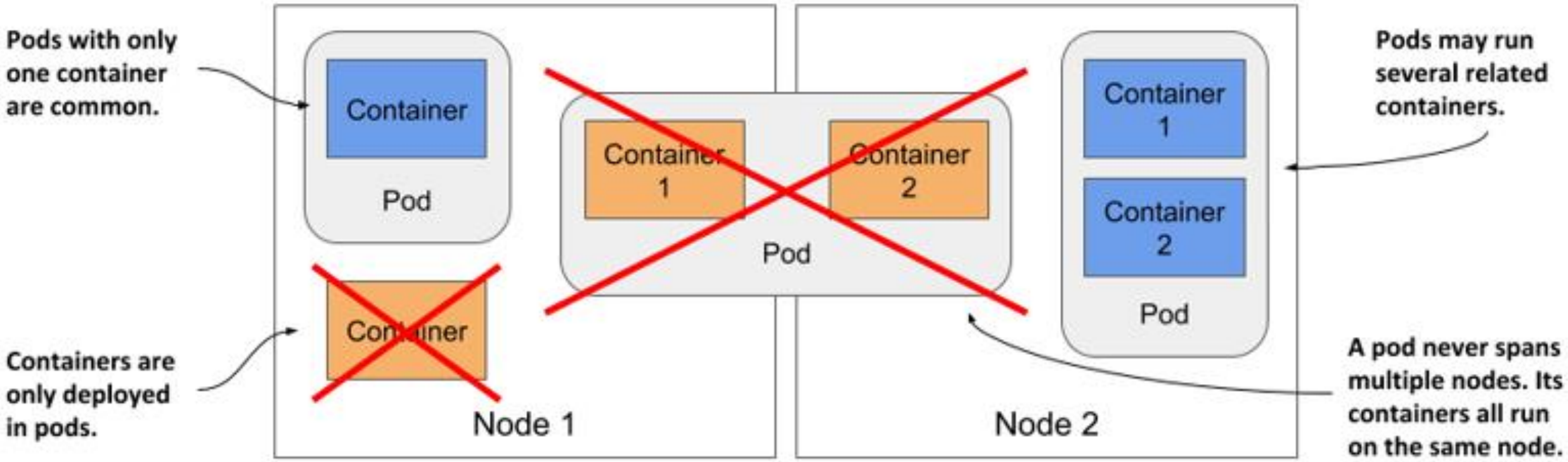
- This is done **through environment variables**, but **clients** can also **look up** the **service** IP through good old **DNS**.

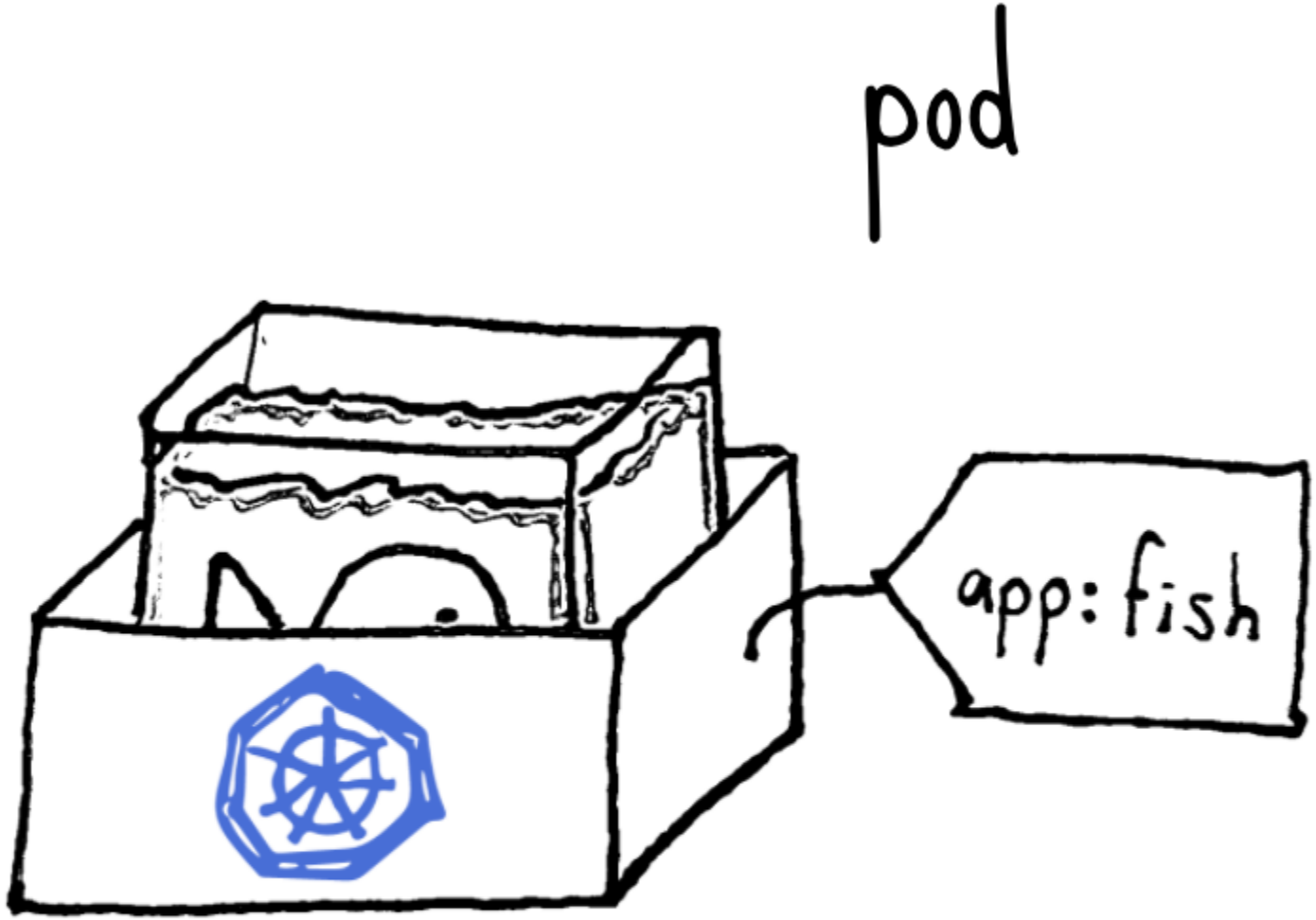
Pods

- A **container** is **not** the **smallest unit** of **deployment** in **Kubernetes**.
- A POD is the **smallest object**, that you **can create** in kubernetes.

Pods

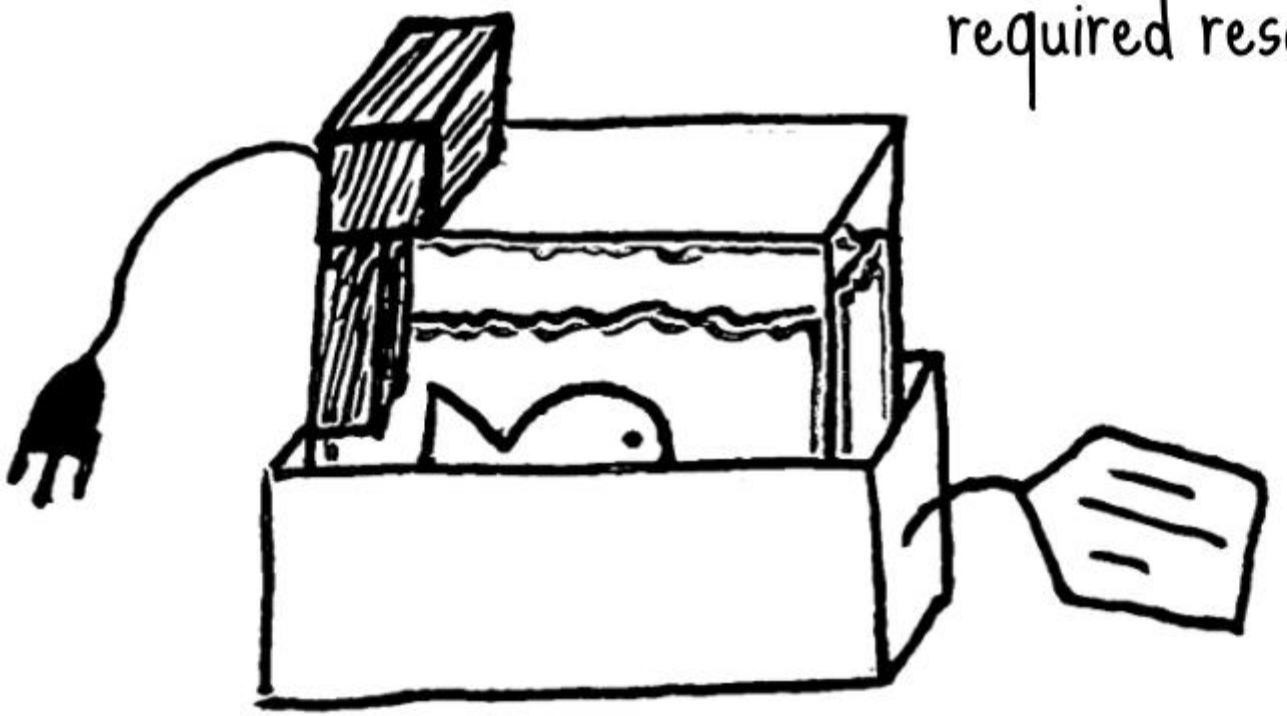
Pods

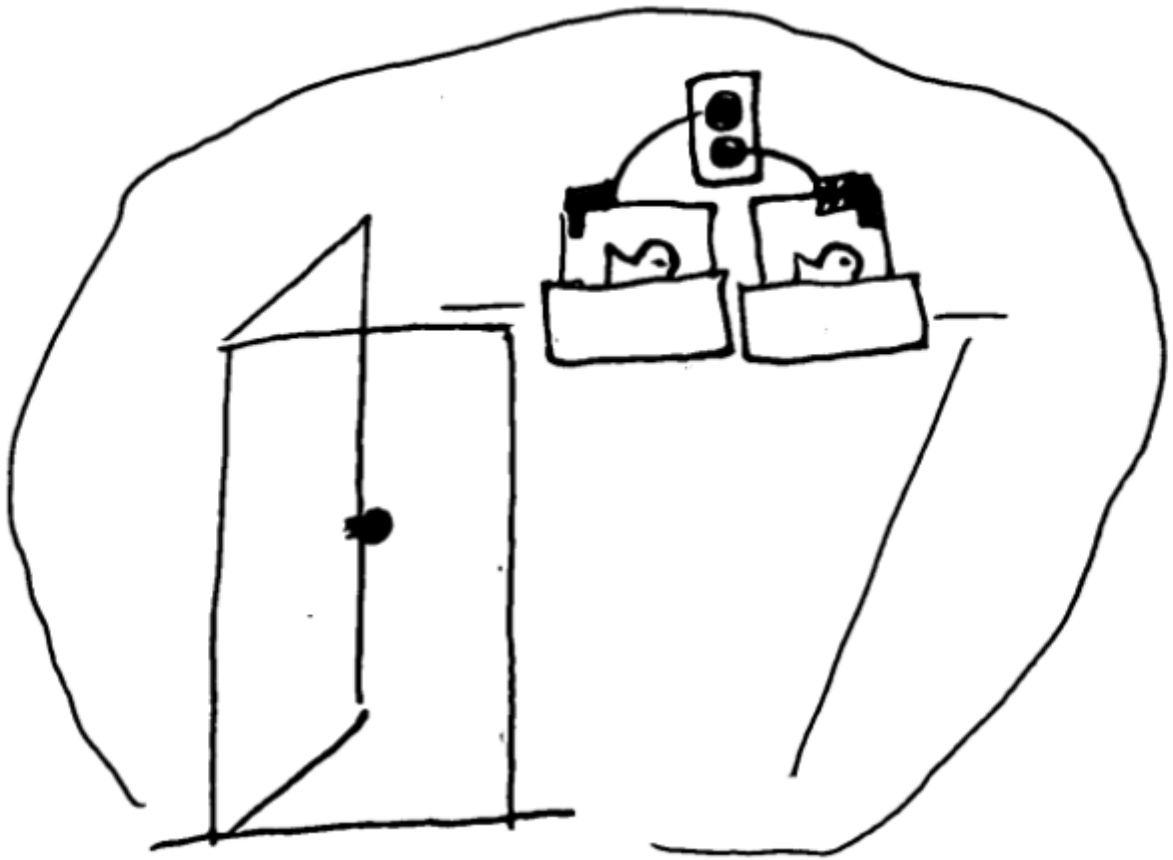

Pods with only one container are common.
Container
Pod
Container 1
Container 2
Pod
Container
Containers are only deployed in pods.
Node 1
Container 1
Container 2
Pod
Pods may run several related containers.
A pod never spans multiple nodes. Its containers all run on the same node.
Node 2

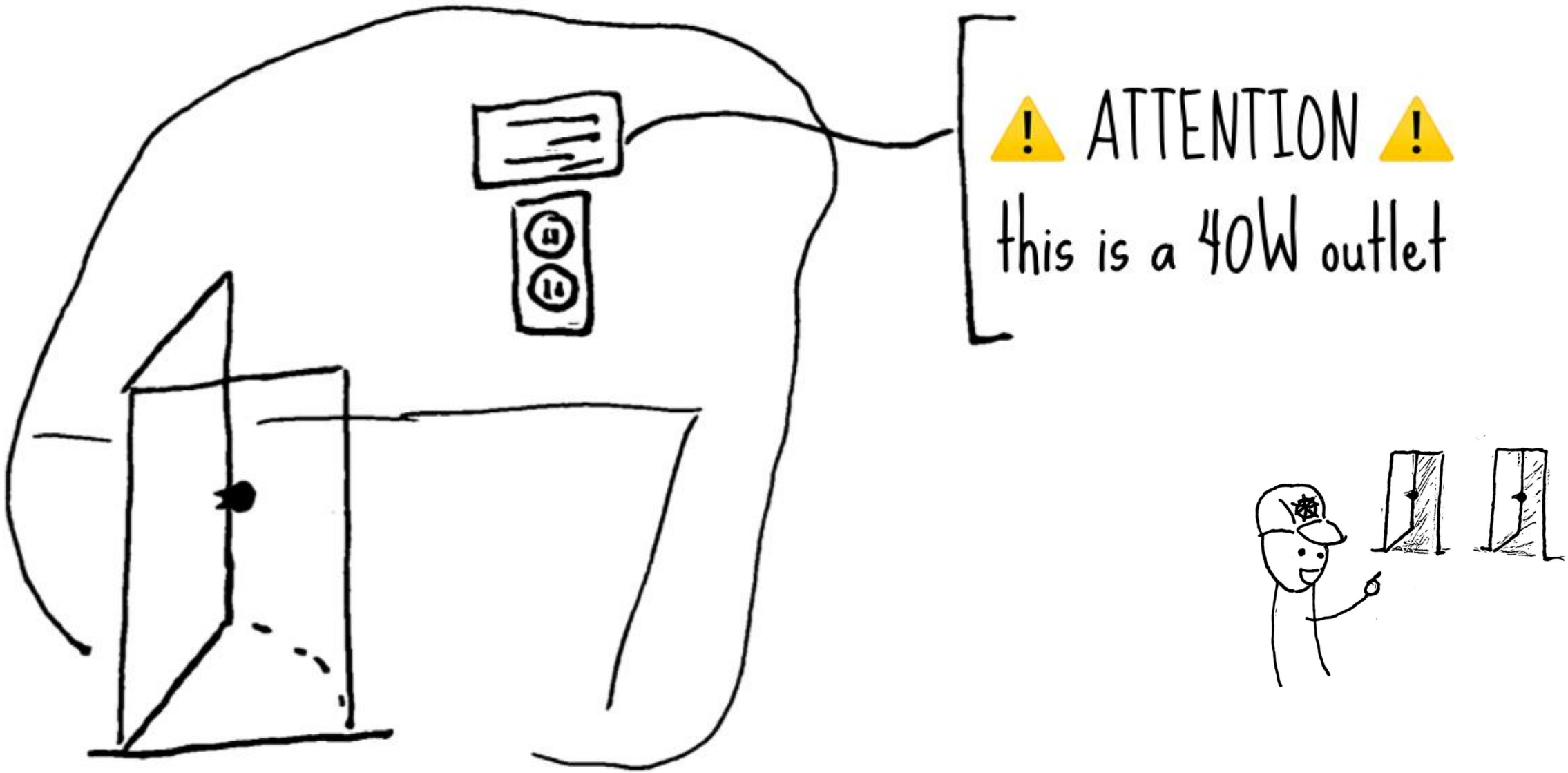

pod
app: fish

water filter:

required resources: 60W



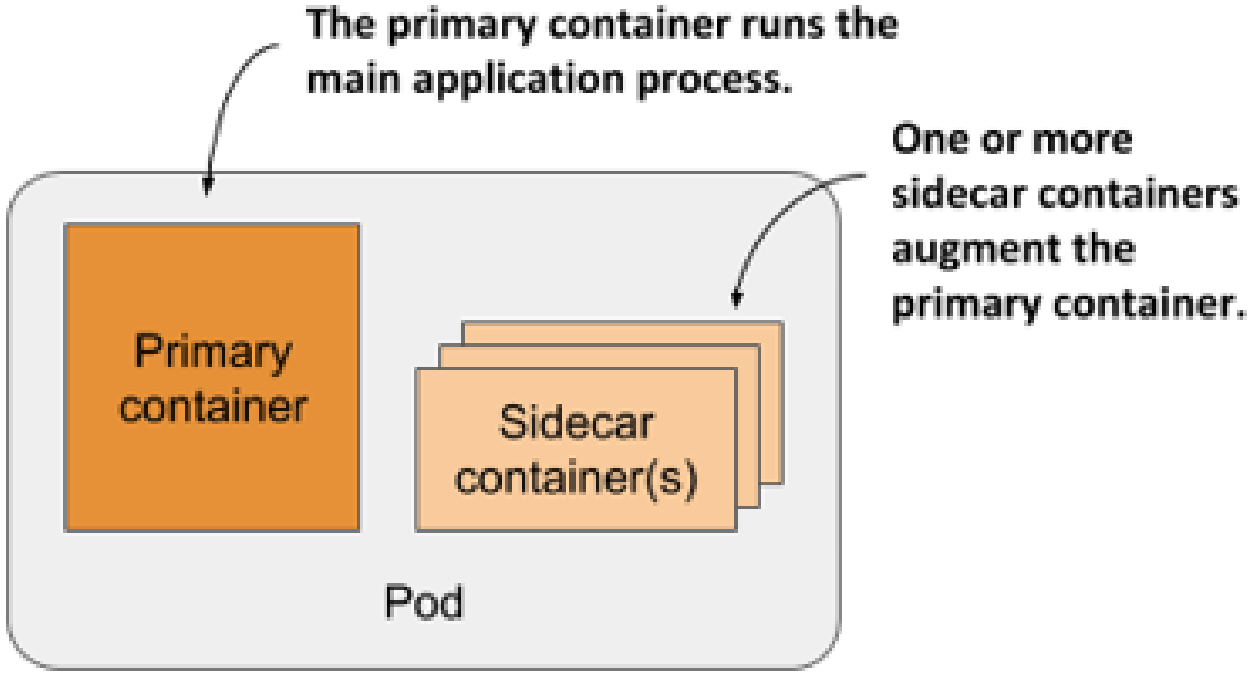



⚠ ATTENTION ⚠
this is a 40W outlet

Pods

Multiple containers in a pod

- **Multiple containers** into a **single pod** is used for **applications that consists** of **one main process** and **one** or **more complementary processes**



Pods

Pod creation with cli

- The simplest way to create a Pod is via the imperative kubectl run command:

$ kubectl run nginx --image nginx

- To see the status of the pod

$ kubectl get pods

- To delete the pod

$ kubectl delete pods/nginx

Pods

Pod creation with manifest

```
apiVersion: v1
kind: Pod
metadata:
  name: demo-app
spec:
  containers:
  - name: demo-app
    image: luksa/kubia:1.0
    ports:
    - containerPort: 8080
```

Pods

Pod creation with manifest

```
apiVersion: v1
kind: Pod
metadata:
  name: demo-app
spec:
  containers:
  - name: demo-app
    image: luksa/kubia:1.0
    ports:
    - containerPort: 8080
```

This manifest uses the v1 API version to define the object

Pods

Pod creation with manifest

```
apiVersion: v1
kind: Pod
metadata:
  name: demo-app
spec:
  containers:
  - name: demo-app
    image: luksa/kubia:1.0
    ports:
    - containerPort: 8080
```

The object specified in this manifest is a **Pod**

Pods

Pod creation with manifest

```
apiVersion: v1
kind: Pod
metadata:
  name: demo-app
spec:
  containers:
  - name: demo-app
    image: luksa/kubia:1.0
    ports:
    - containerPort: 8080
```

The name of the pod

Pods

Pod creation with manifest

```
apiVersion: v1
kind: Pod
metadata:
  name: demo-app
spec:
  containers:
  - name: demo-app
    image: luksa/kubia:1.0
    ports:
    - containerPort: 8080
```

- The name of the container
- Container image to create the container from
- The port the app is listening on

Pods

Pod creation with manifest

```
apiVersion: v1
kind: Pod
metadata:
  name: demo-app
spec:
  containers:
  - name: demo-app
    image: luksa/kubia:1.0
    ports:
    - containerPort: 8080
```

Use the **kubectl apply** command to **launch** a single instance of the pod:

$ kubectl create -f demo-pod.yaml

Pods

TIP

- To create a pod **manifest from scratch**, use the following command to **create** the file and **then edit** it to add **more fields**:

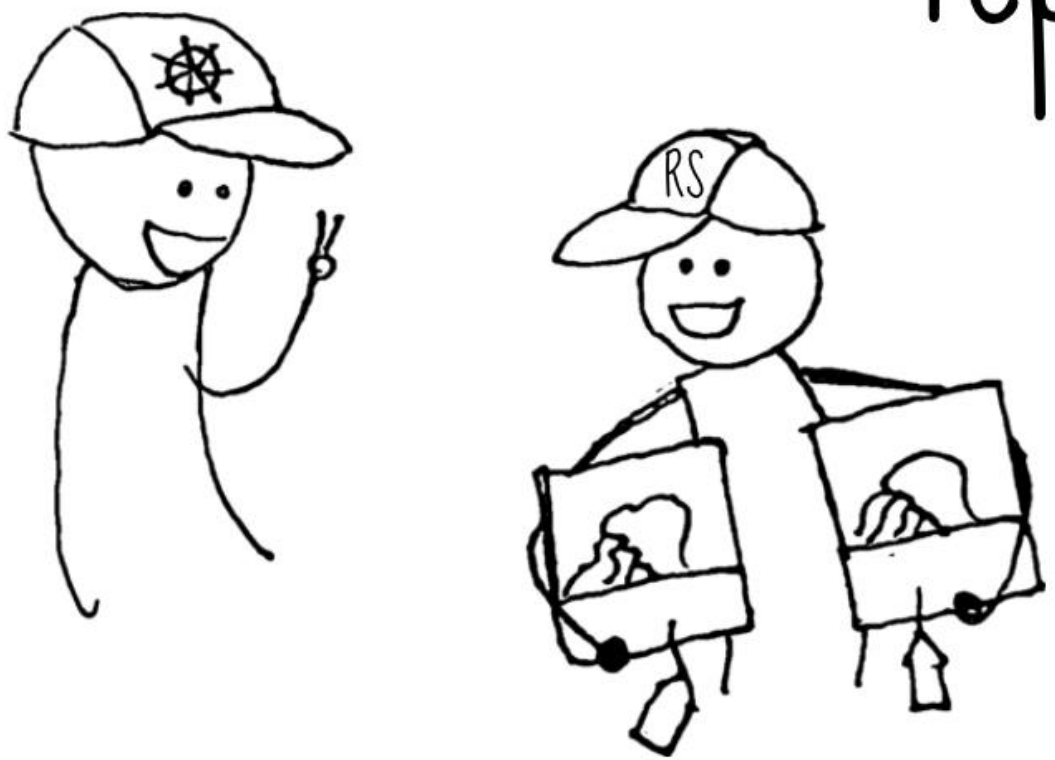
$ kubectl run demo-app --image=luksa/kubia:1.0 --dry-run=client -o yaml > mypod.yaml

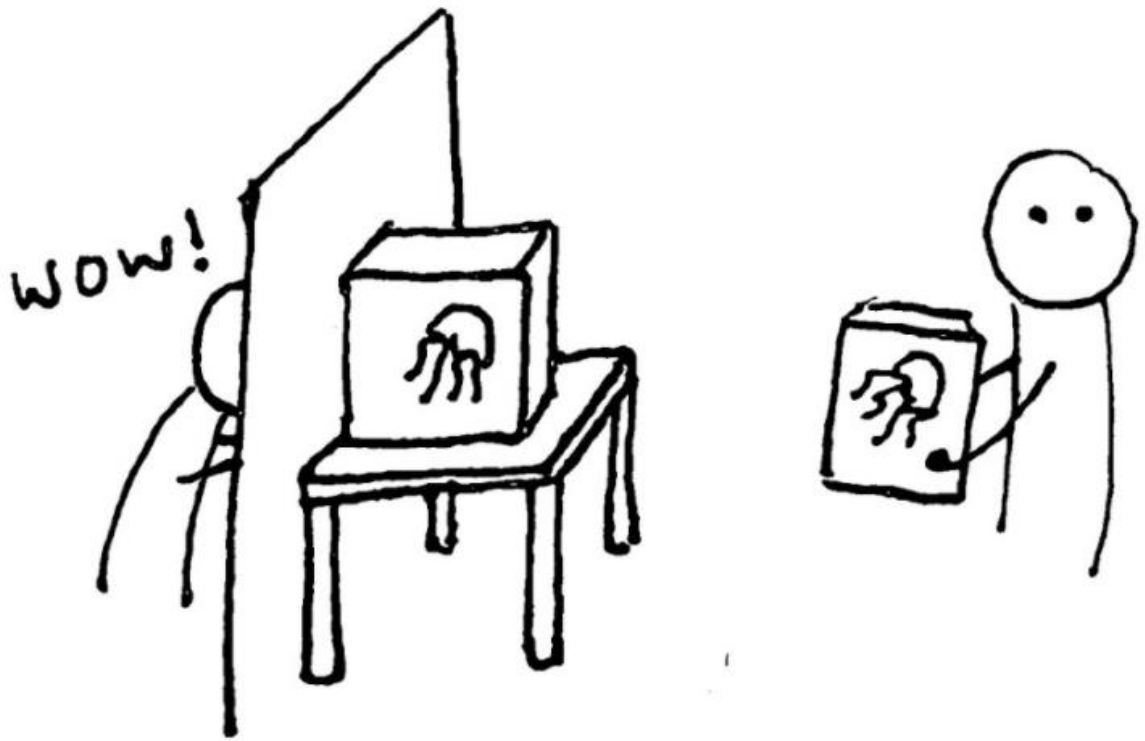
- The **--dry-run=client** flag tells kubectl to output the definition instead of actually creating the object via the API.

- Now we know how to:
  - **package** our **app components** into containers,
  - **group them** into pods,

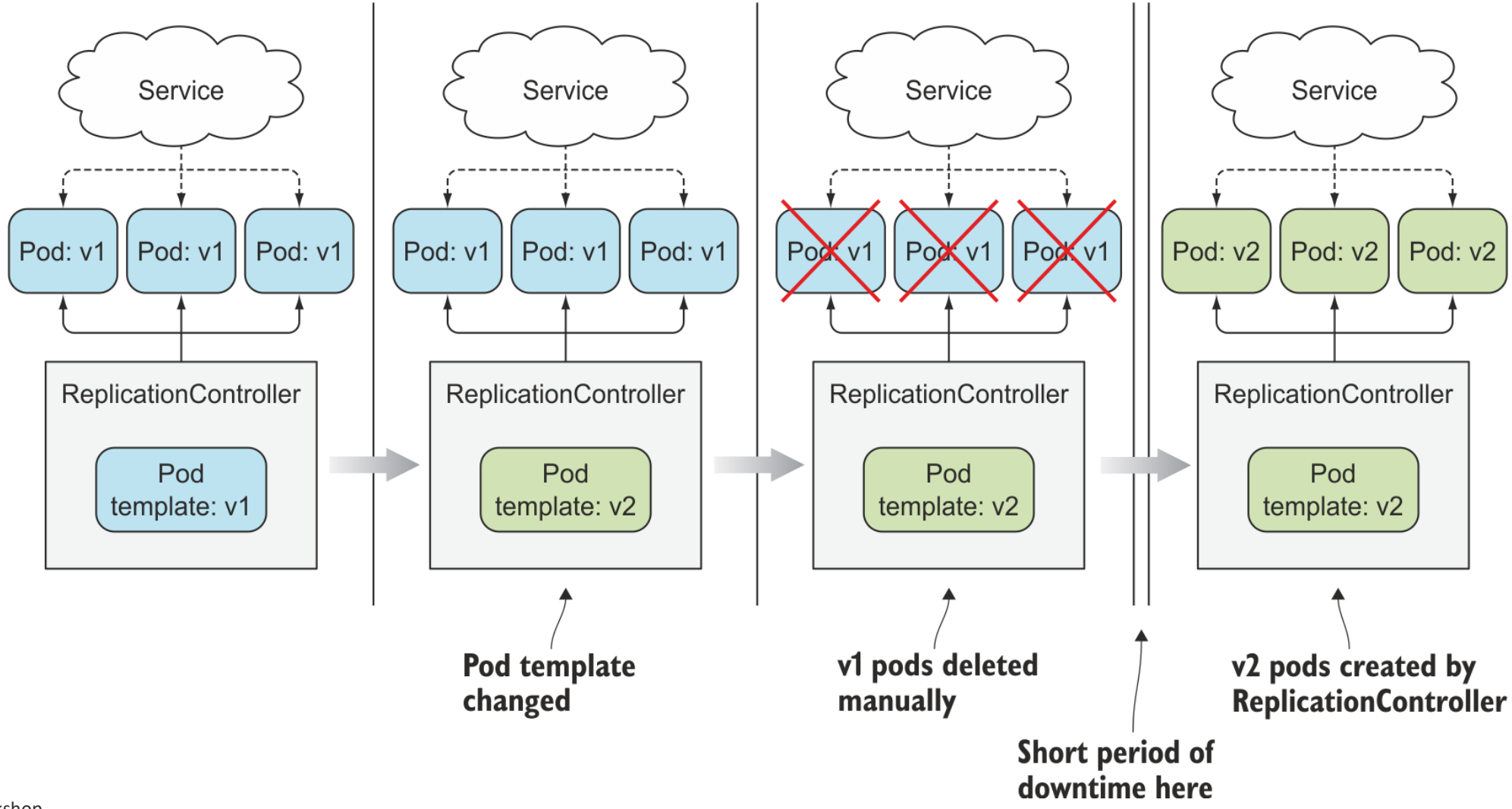
- Let’s see how we can **manage** them and **update** them.

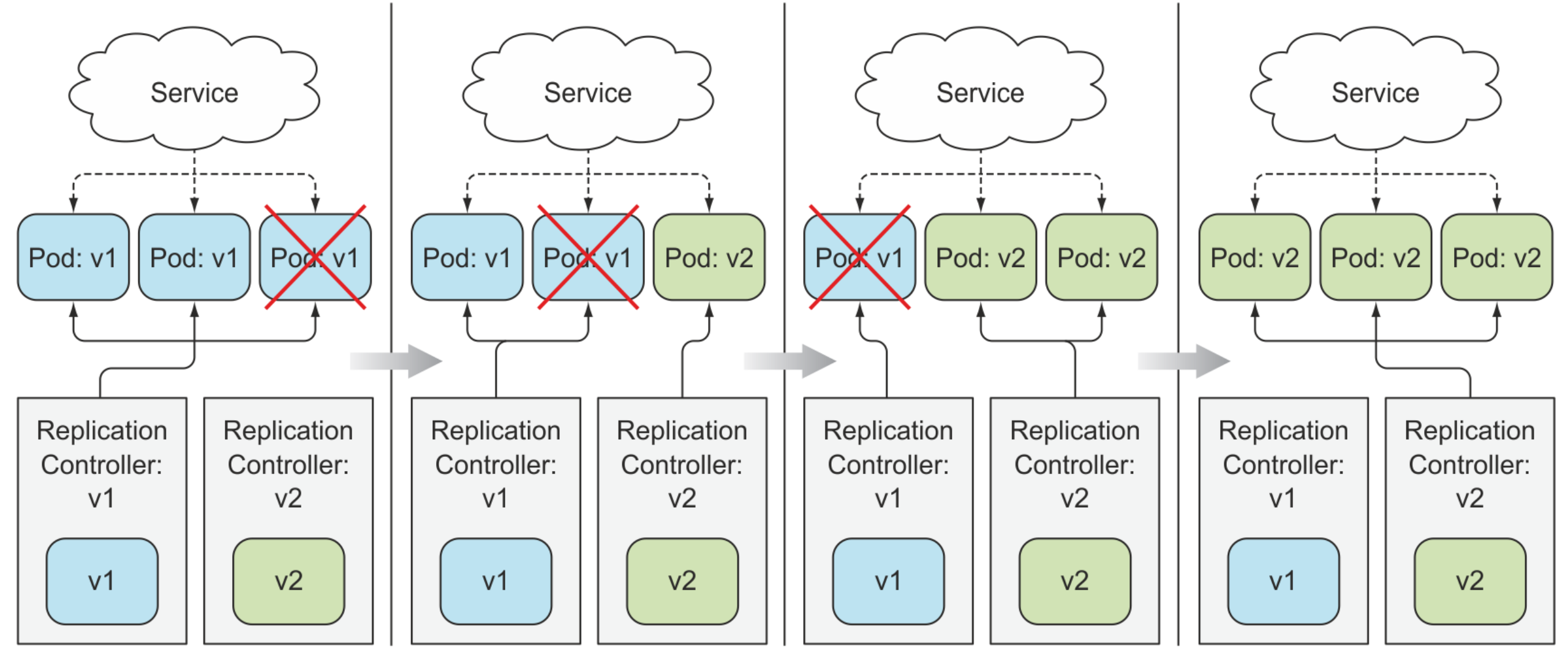
replica set




WOW!

- There are two ways of **updating** pods


Service
Pod: v1
Pod: v1
Pod: v1
ReplicationController
Pod
template: v1
Service
Pod: v1
Pod: v1
Pod: v1
ReplicationController
Pod
template: v2
Pod template
changed
Service
Pod: v1
Pod: v1
Pod: v1
ReplicationController
Pod
template: v2
v1 pods deleted
manually
Short period of
downtime here
Service
Pod: v2
Pod: v2
Pod: v2
ReplicationController
Pod
template: v2
v2 pods created by
ReplicationController

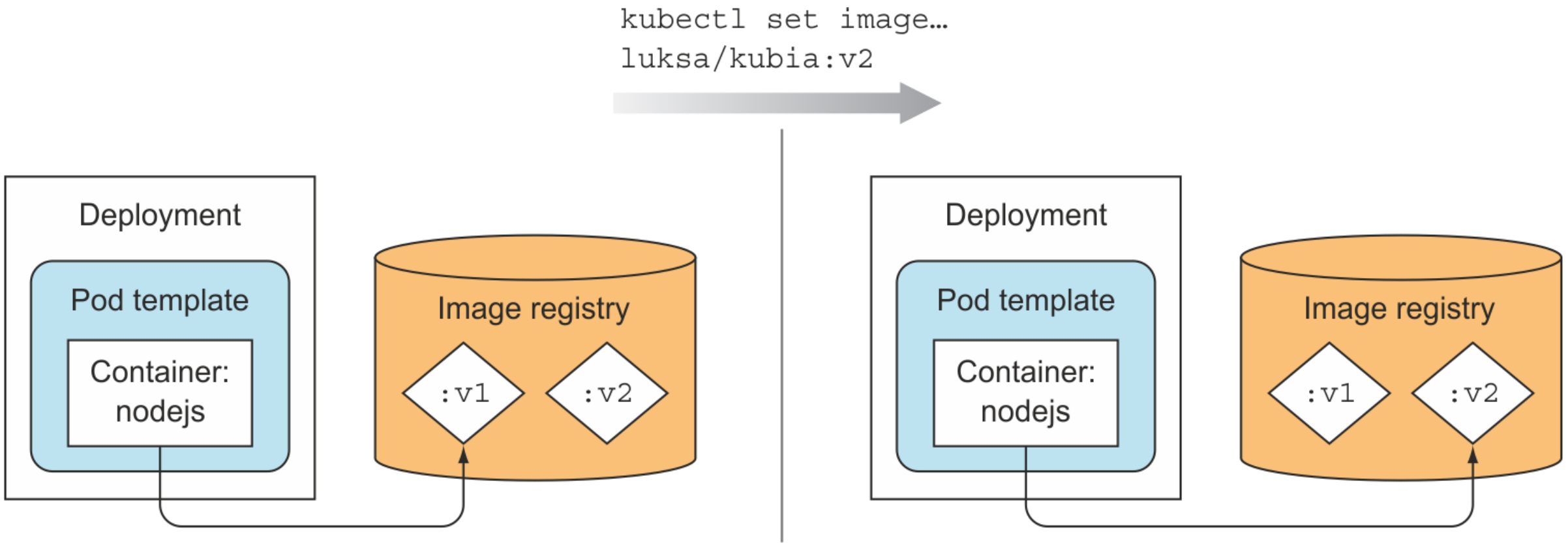

Service
Pod: v1
Pod: v1
Pod: v1
Replication Controller: v1
v1
Replication Controller: v2
v2
Service
Pod: v1
Pod: v1
Pod: v2
Replication Controller: v1
v1
Replication Controller: v2
v2
Service
Pod: v1
Pod: v2
Pod: v2
Replication Controller: v1
v1
Replication Controller: v2
v2
Service
Pod: v2
Pod: v2
Pod: v2
Replication Controller: v1
v1
Replication Controller: v2
v2

Deployments

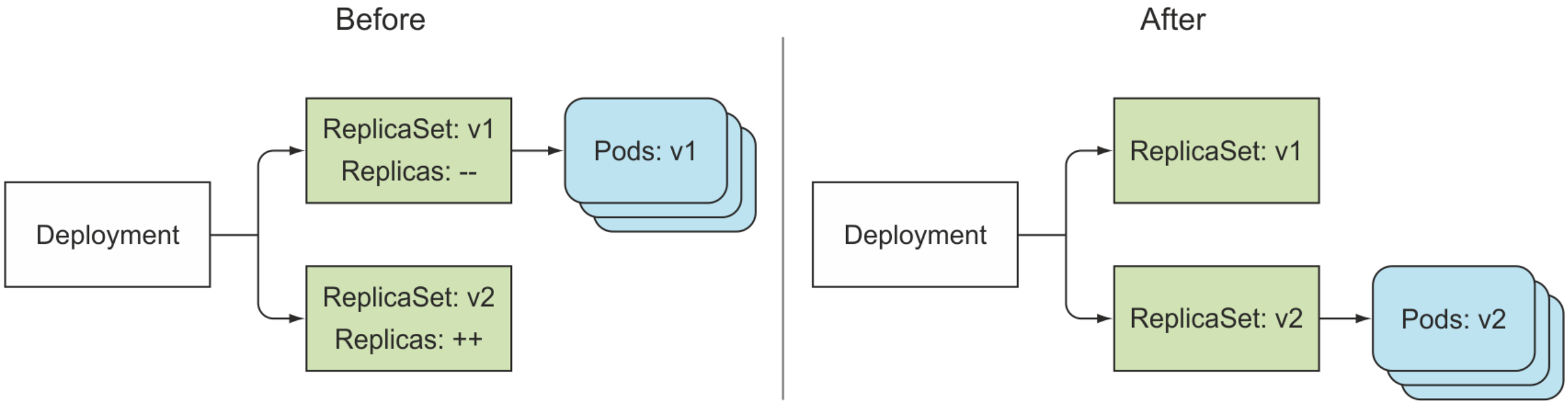
- A Deployment is a **higher-level resource** meant for **deploying applications** and **updating**

- When using a Deployment, the **actual pods** are **created** and **managed** by the Deployment's **ReplicaSets**, **not** by the **Deployment** directly

Deployments

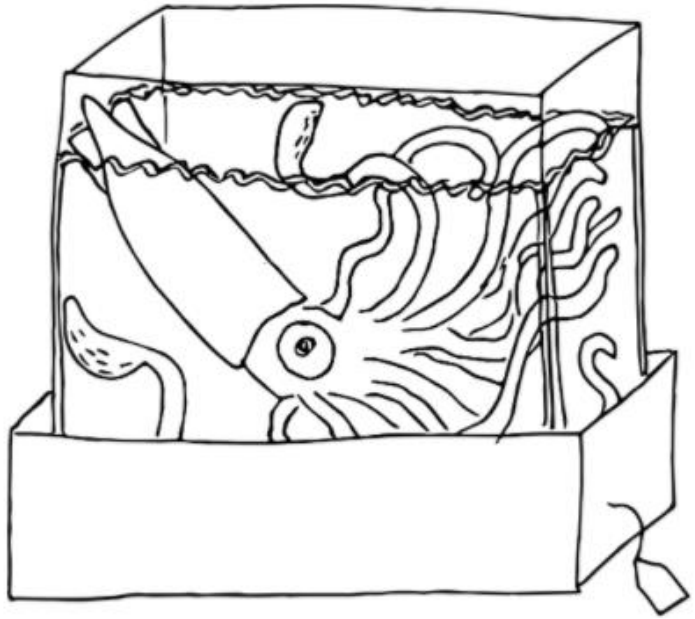


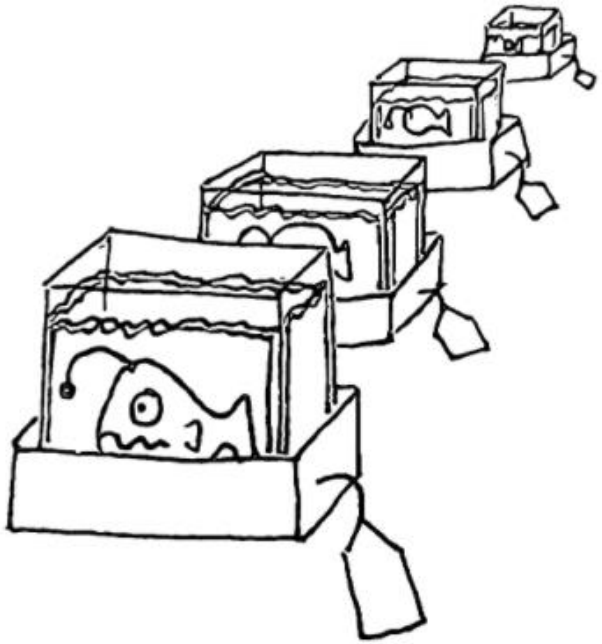
Deployments

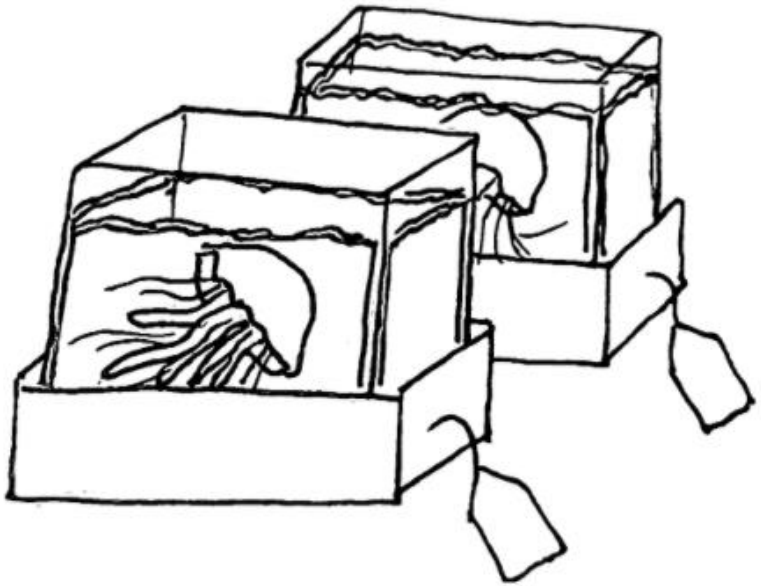


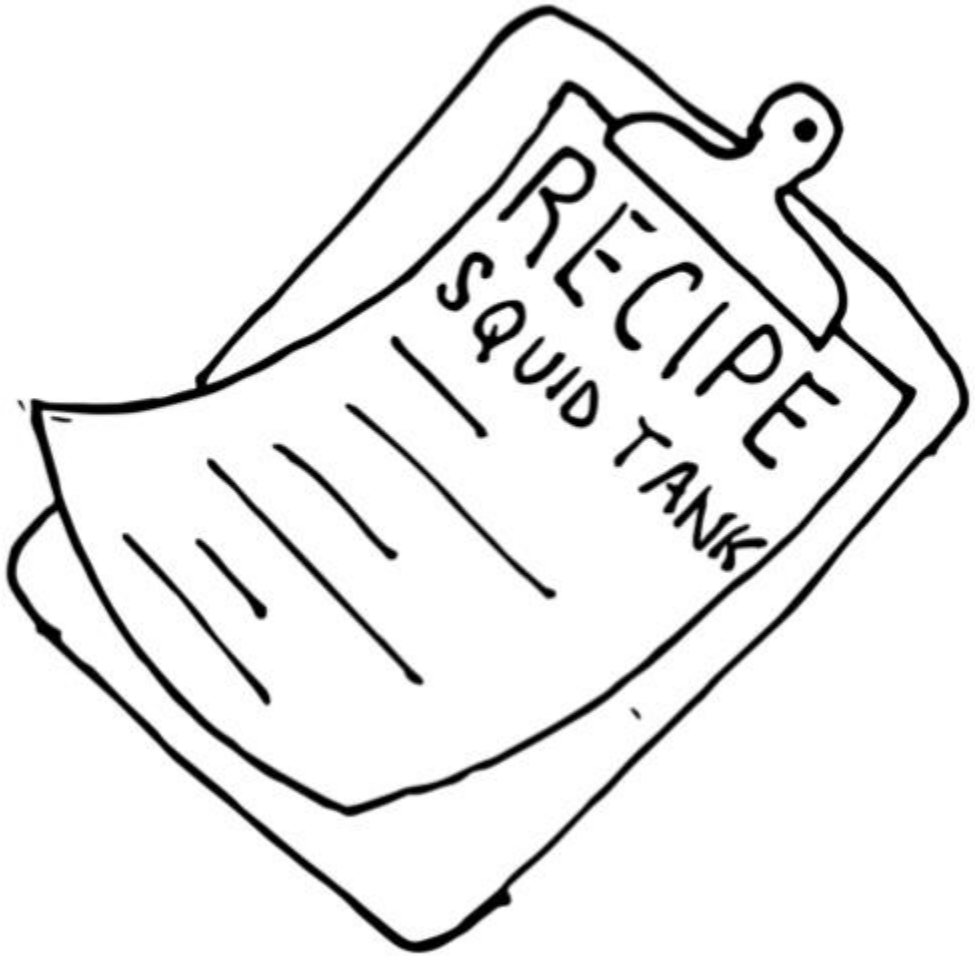
deployment

the Deep Sea exhibit








RECIPE
SQUID TANK

Deployments

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kubia
spec:
  replicas: 3
  template:
    metadata:
      name: kubia
      labels:
        app: kubia
    spec:
      containers:
      - image: luksa/kubia:v1
        name: nodejs
  selector:
    matchLabels:
      app: kubia
```

Deployments

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kubia
spec:
  replicas: 3
  template:
    metadata:
      name: kubia
      labels:
        app: kubia
    spec:
      containers:
      - image: luksa/kubia:v1
        name: nodejs
  selector:
    matchLabels:
      app: kubia
```

Deployments

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kubia
spec:
  replicas: 3
  template:
    metadata:
      name: kubia
      labels:
        app: kubia
    spec:
      containers:
      - image: luksa/kubia:v1
        name: nodejs
  selector:
    matchLabels:
      app: kubia
```

Deployments

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kubia
spec:
  replicas: 3
  template:
    metadata:
      name: kubia
      labels:
        app: kubia
    spec:
      containers:
      - image: luksa/kubia:v1
        name: nodejs
  selector:
    matchLabels:
      app: kubia
```

Deployments

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kubia
spec:
  replicas: 3
  template:
    metadata:
      name: kubia
      labels:
        app: kubia
    spec:
      containers:
      - image: luksa/kubia:v1
        name: nodejs
  selector:
    matchLabels:
      app: kubia
```

Deployments

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kubia
spec:
  replicas: 3
  template:
    metadata:
      name: kubia
      labels:
        app: kubia
    spec:
      containers:
      - image: luksa/kubia:v1
        name: nodejs
  selector:
    matchLabels:
      app: kubia
```

```
$ kubectl create -f kubia-deployment-v1.yaml --record
```

- --record **records** the command in the **revision** history

```
$ kubectl get deployment
$ kubectl describe deployment
$ kubectl rollout status deployment kubia
$ kubectl get po
$ kubectl get replicasets
```

Deployments

Rolling back – Simulation

```
$ kubectl set image deployment kubia nodejs=luksa/kubia:v3
$ kubectl rollout status deployment kubia
$ watch kubectl get po
$ kubectl rollout undo deployment kubia
```

Deployments

Rolling back – Simulation

- **Rolling back** a rollout is **possible** because Deployments keep a **revision history**.
- The **history** is stored in the **underlying ReplicaSets**.
- When a **rollout completes**, the **old ReplicaSet isn't deleted**, and this **enables rolling back** to any revision

Deployments

Rolling back – Simulation

- The revision **history** can be **displayed** with the kubectl rollout history command:

```
$ kubectl rollout history deployment kubia
```

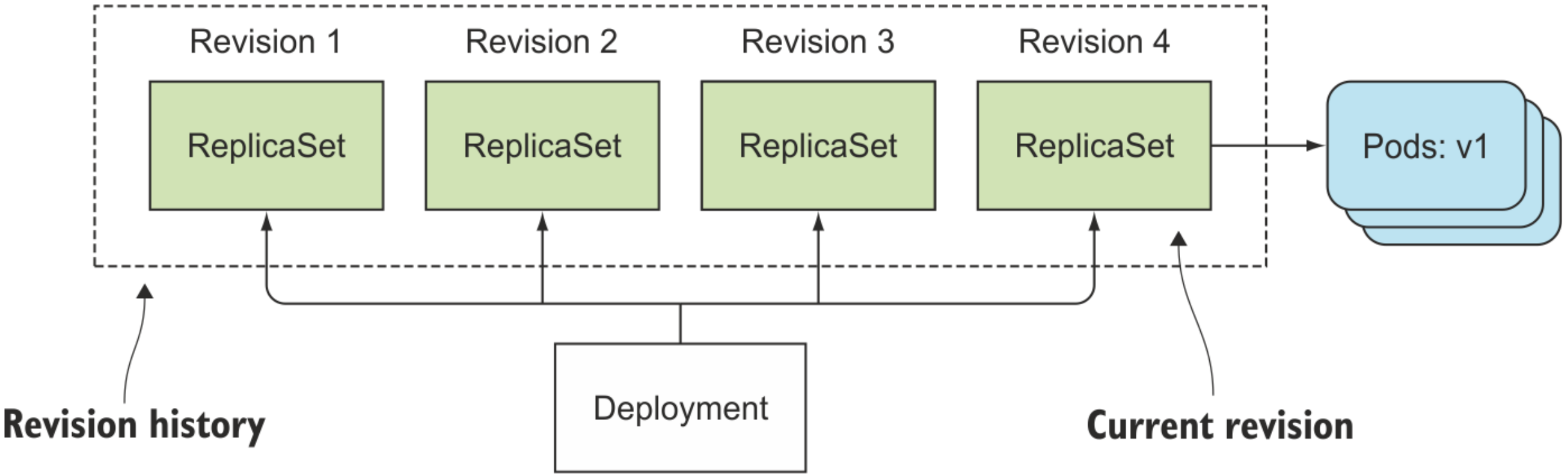
- Without --**record**, the **CHANGE-CAUSE** column in the revision history would be **empty**.

- To roll **back** to a **specific revision** by specifying the revision in the undo command:

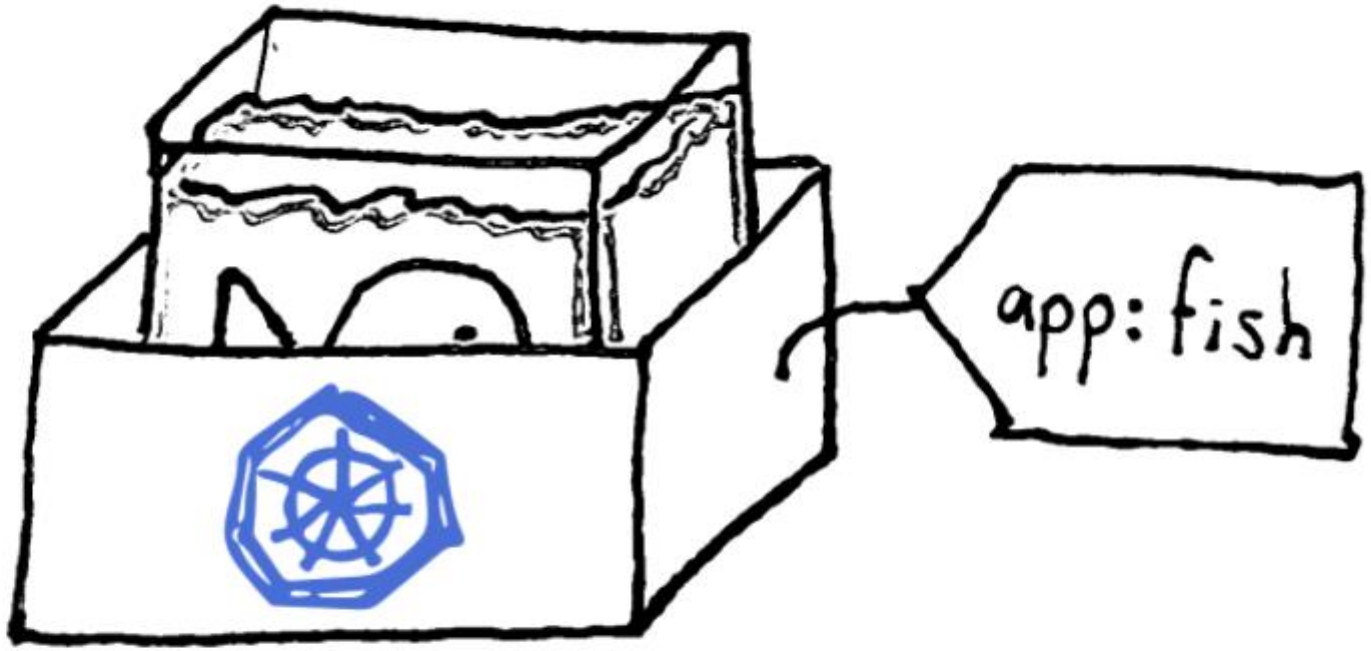
```
$ kubectl rollout undo deployment kubia --to-revision=1
```

Deployments

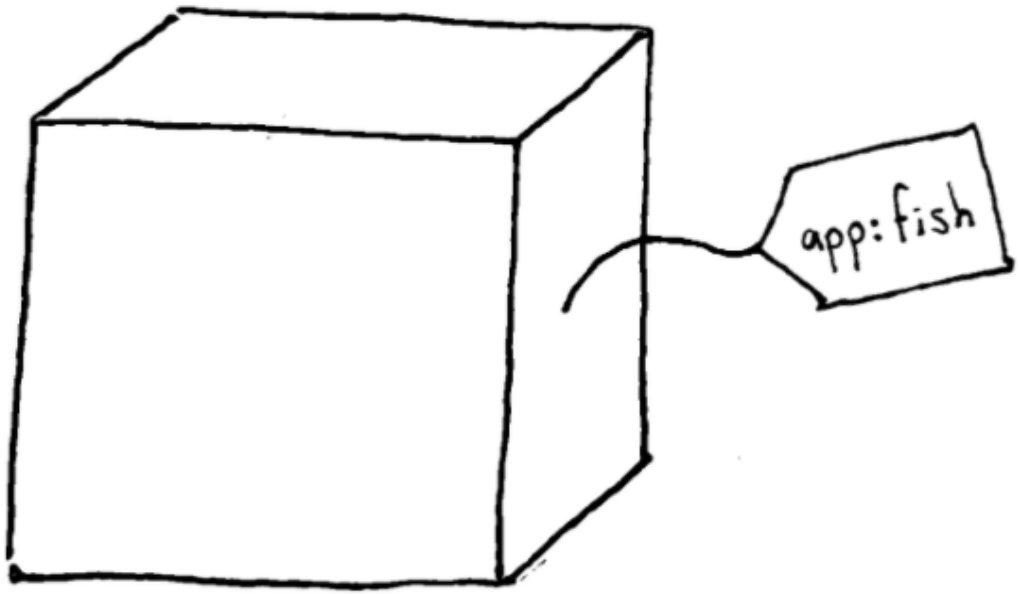
Rolling back – Simulation

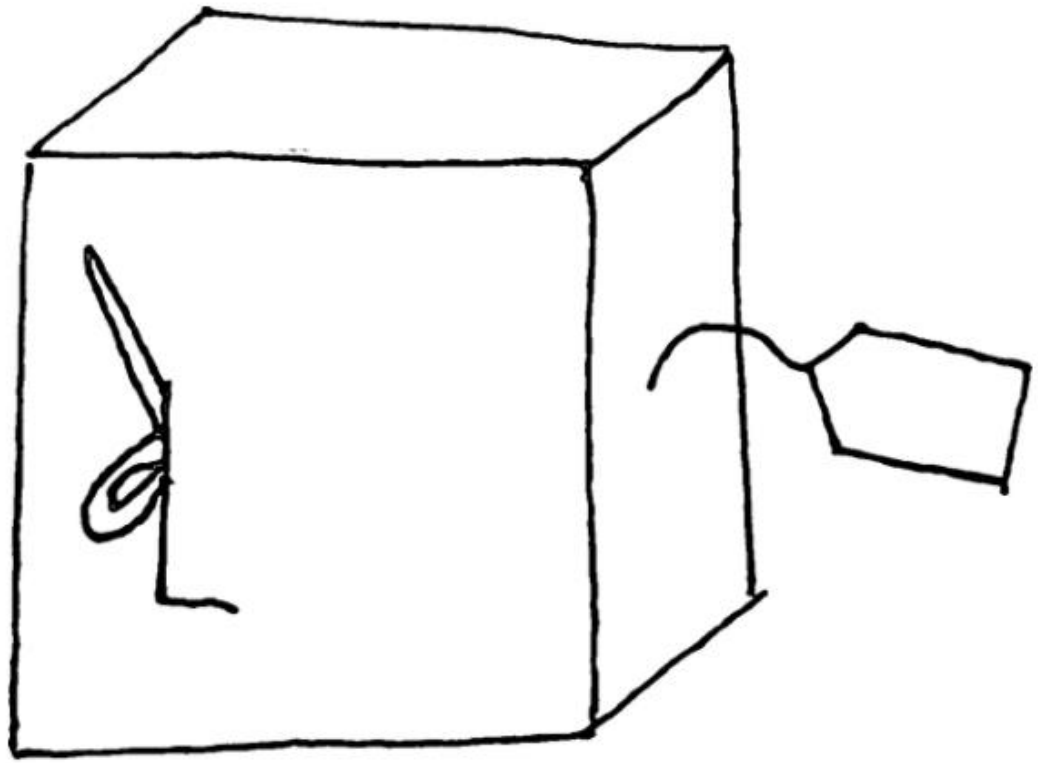


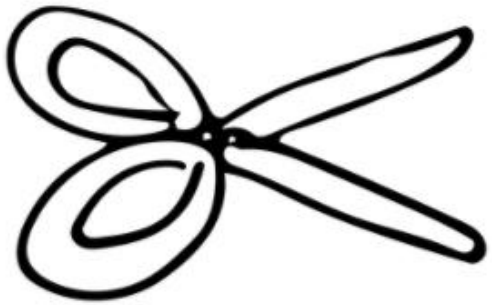
pod



pod







service

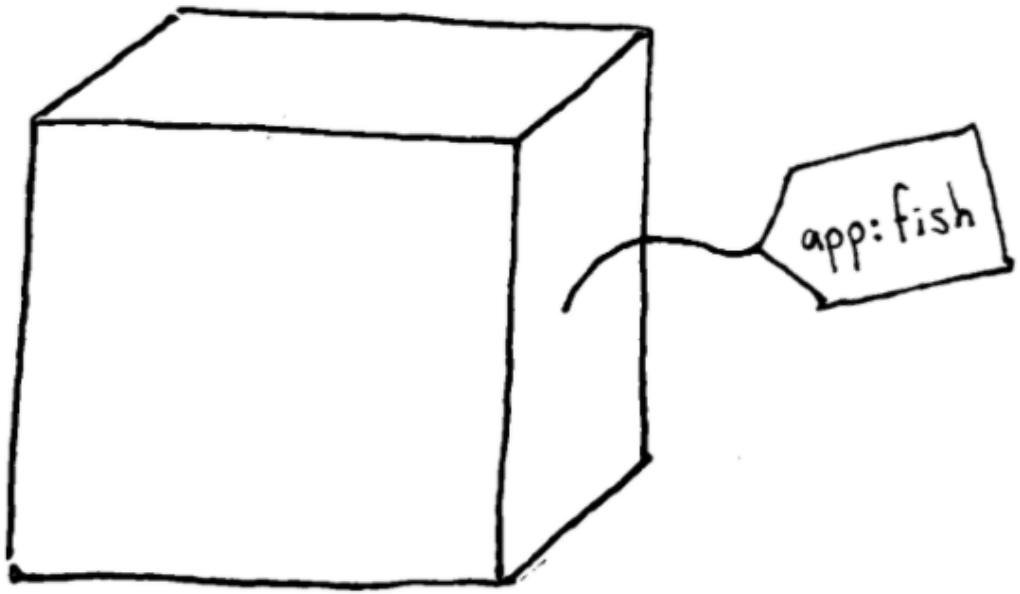
Services

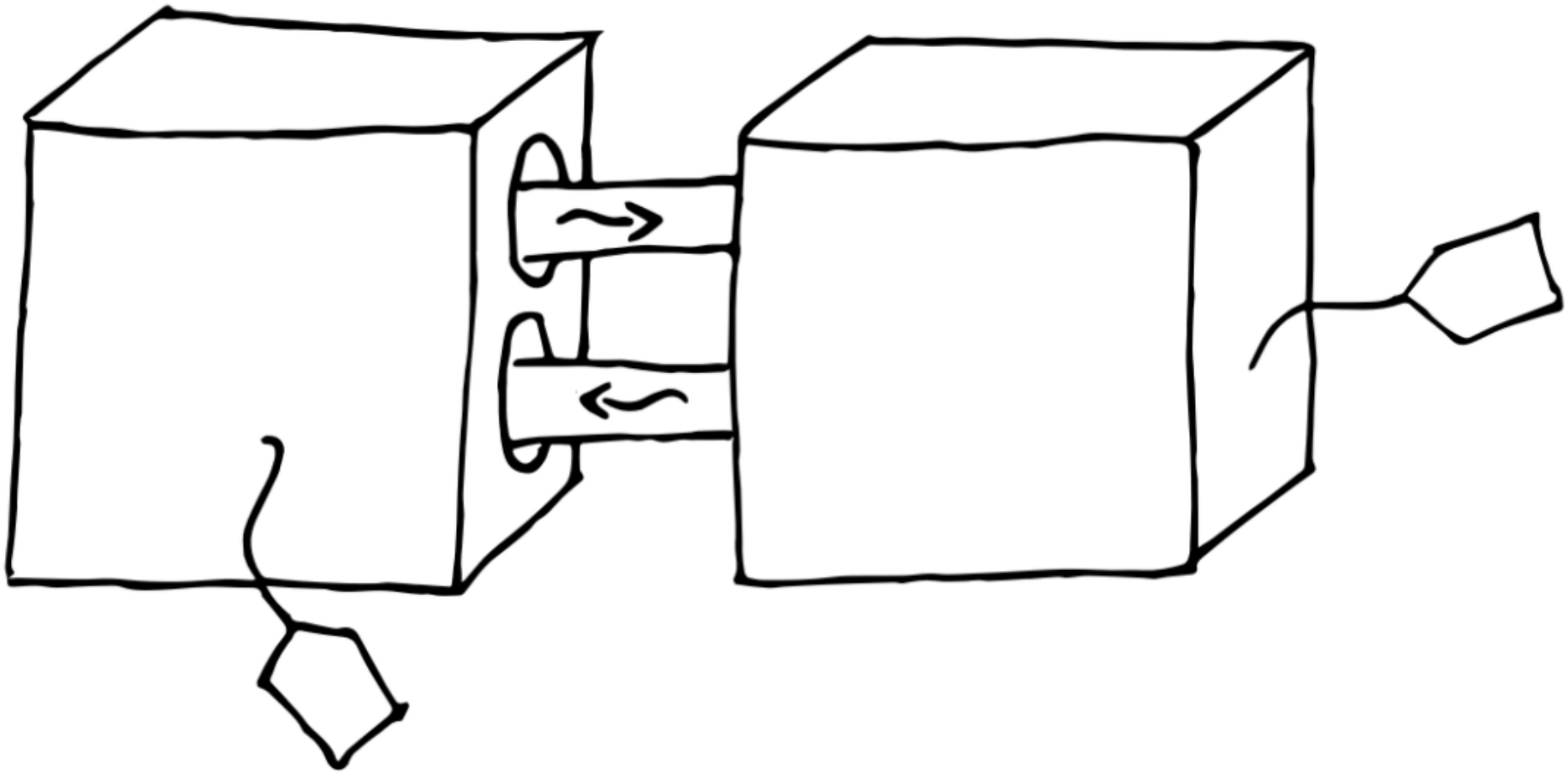
- A Kubernetes **Service** is a **resource** that **makes** a **single**, **constant point of entry** to a **group** of **pods** providing the **same service**.

- Each **service has an IP** address and **port** that **never change** while the service **exists**.

- Clients can **open connections to that IP** and **port**, and those connections are then **routed** to **one** of the **pods** backing that service.

pod





services between pods

Services

Service creation – via manifest

```
apiVersion: v1
kind: Service
metadata:
  name: kubia
spec:
  ports:
  - port: 80
    targetPort: 8080
  selector:
    app: kubia
```

Services

Service creation – via manifest

```
apiVersion: v1
kind: Service
metadata:
  name: kubia
spec:
  ports:
  - port: 80
    targetPort: 8080
  selector:
    app: kubia
```

- The **port** this **service** will be **available on**
- The **container port** the **service** will **forward to**

Services

Service creation – via manifest

```
apiVersion: v1
kind: Service
metadata:
  name: kubia
spec:
  ports:
  - port: 80
    targetPort: 8080
  selector:
    app: kubia
```

- All **pods** with the **app=kubia label** will be **part** of this **service**.

Services

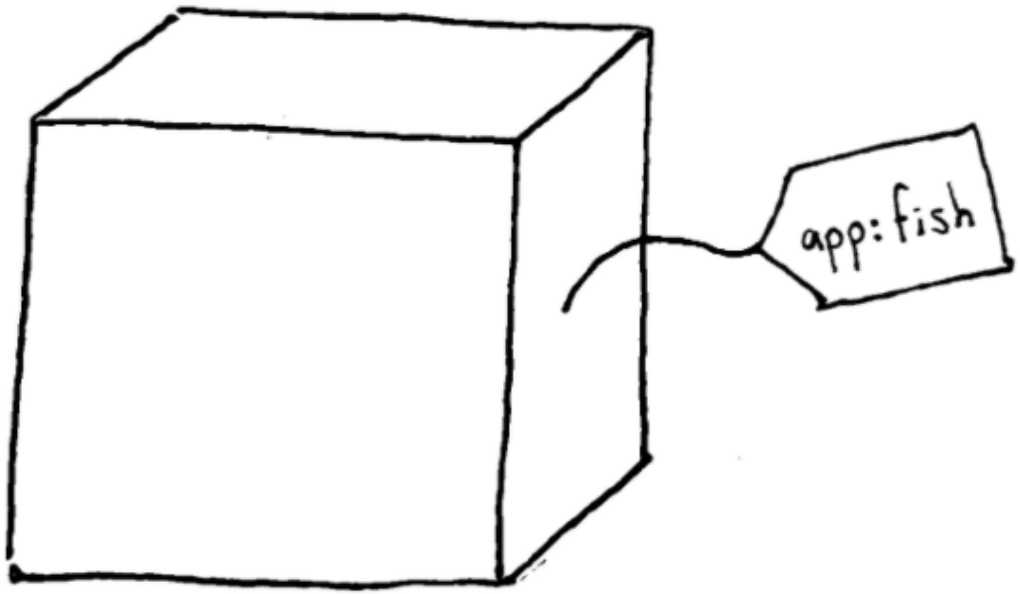
Service creation – via manifest

```
apiVersion: v1
kind: Service
metadata:
  name: kubia
spec:
  ports:
  - port: 80
    targetPort: 8080
  selector:
    app: kubia
```

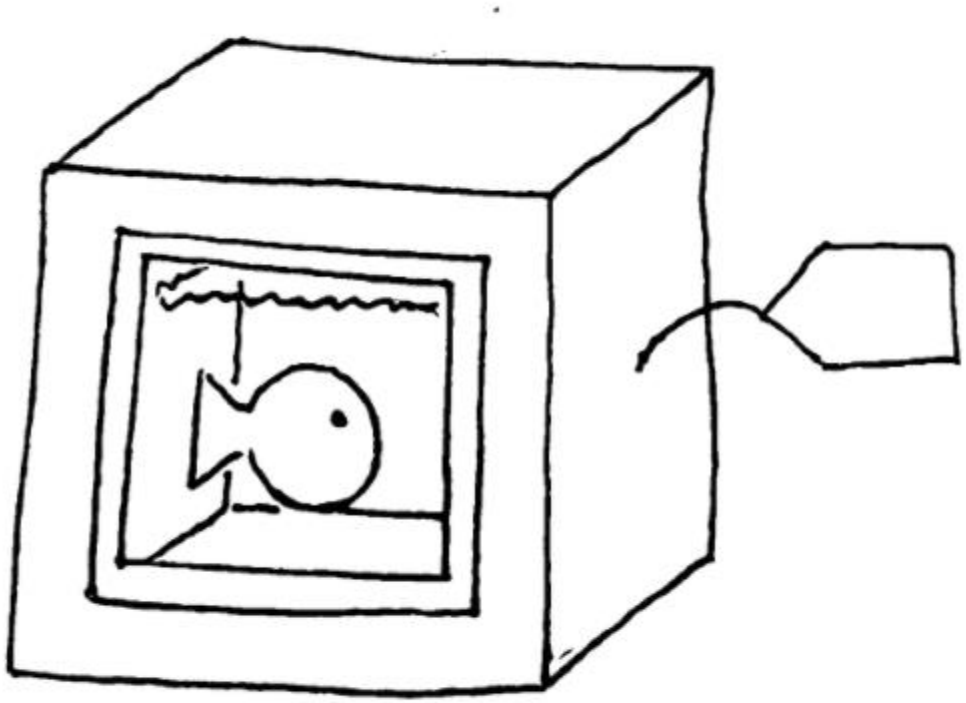
```
$ kubectl create –f kubia-svc.yaml

$ kubectl get svc
```

- This is a **clusterIP** service,
- It's only **accessible** from **inside** the **cluster**.
- The primary **purpose** of this **service** is **exposing groups** of pods to **other pods** in the **cluster**

pod



exposed pod



Services

NodePort service

- With NodePort, Kubernetes **reserve** a **port** on **all** its **nodes**
  - (the **same port** number is used across all of them)

- Kubernetes will **forward incoming connections** to the **pods** that are **part** of the **service**.

Services

NodePort service

```
apiVersion: v1
kind: Service
metadata:
  name: kubia-nodeport
spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: 8080
    nodePort: 30123
  selector:
    app: kubia
```

Services

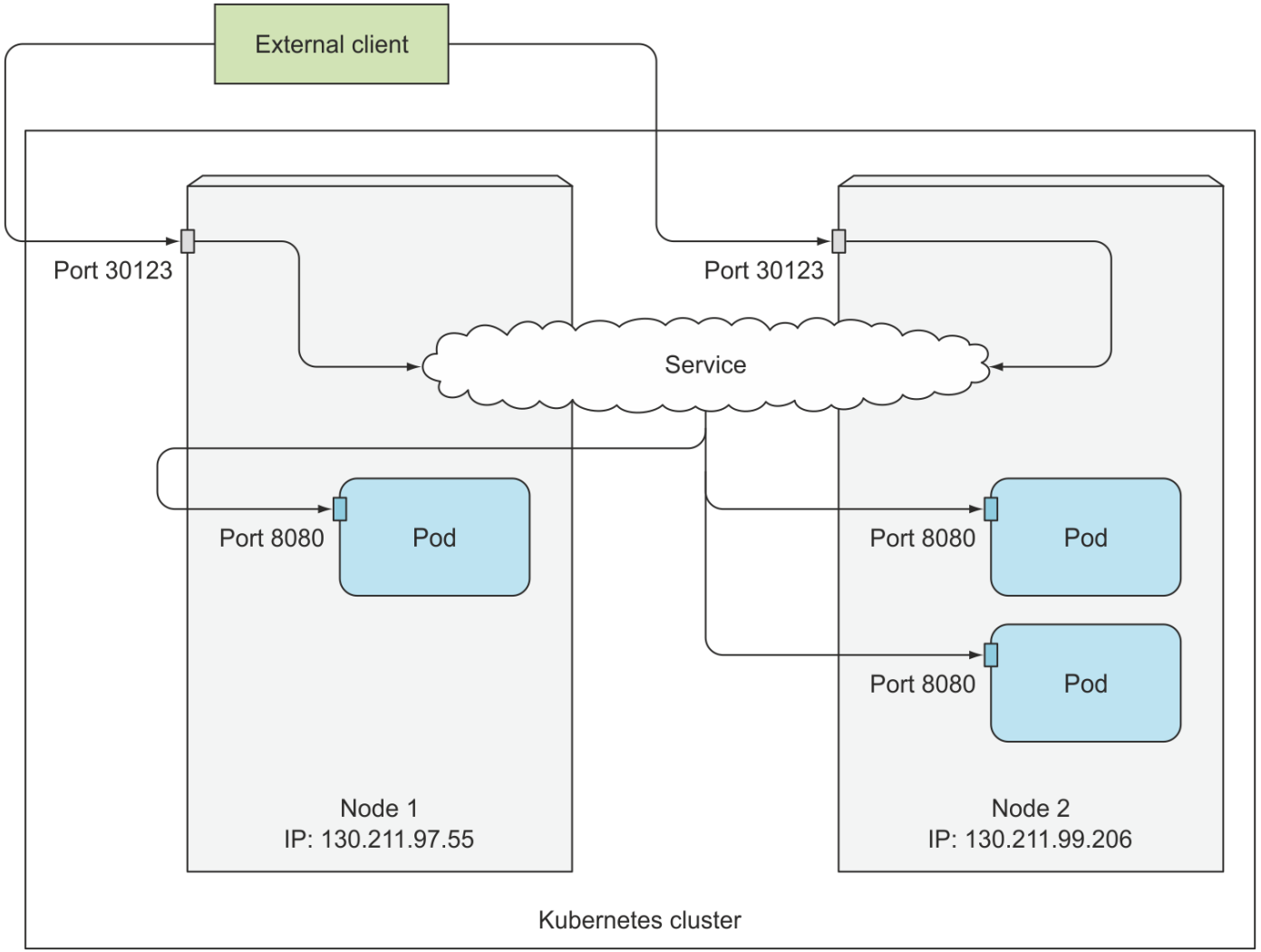
NodePort service

```
apiVersion: v1
kind: Service
metadata:
  name: kubia-nodeport
spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: 8080
    nodePort: 30123
  selector:
    app: kubia
```

Services

NodePort service

```
apiVersion: v1
kind: Service
metadata:
  name: kubia-nodeport
spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: 8080
    nodePort: 30123
  selector:
    app: kubia
```



Services

NodePort service

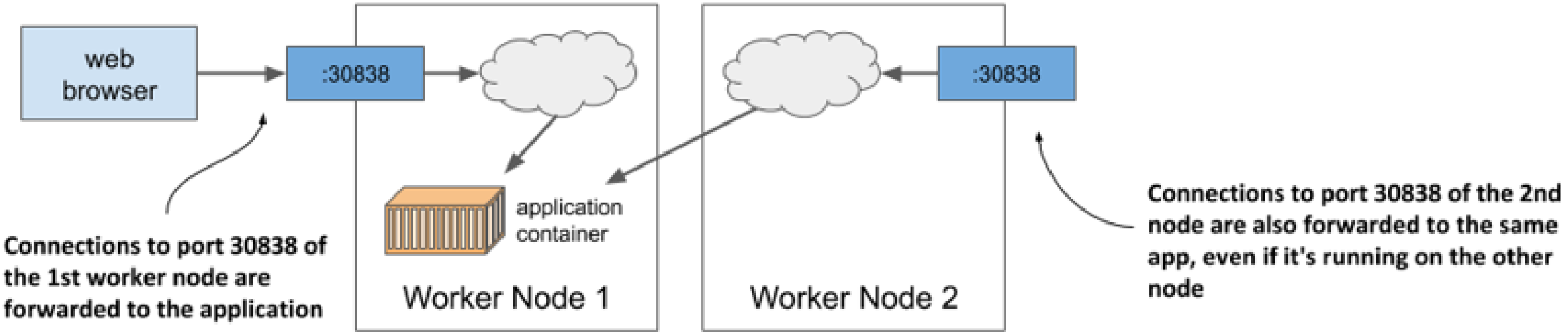
```
apiVersion: v1
kind: Service
metadata:
  name: kubia-nodeport
spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: 8080
    nodePort: 30123
  selector:
    app: kubia
```

```
$ kubectl create –f kubia-svc-nodeport.yaml

$ kubectl get svc kubia-nodeport
```

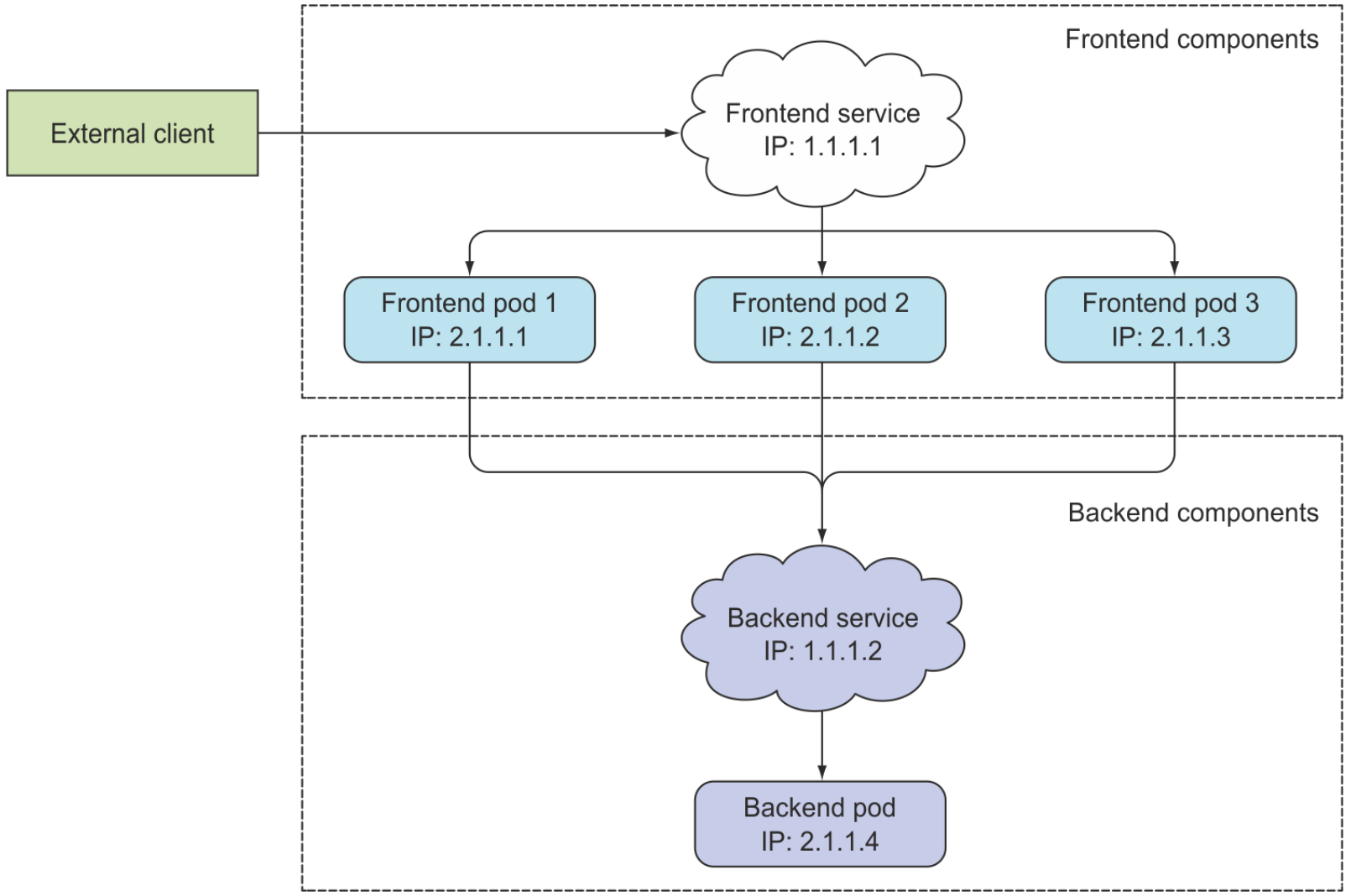
- The service is **accessible** at the following addresses:
  - < Service IP>: :80
  - <1st node's IP>:30123
  - <2nd node's IP>:30123

Services

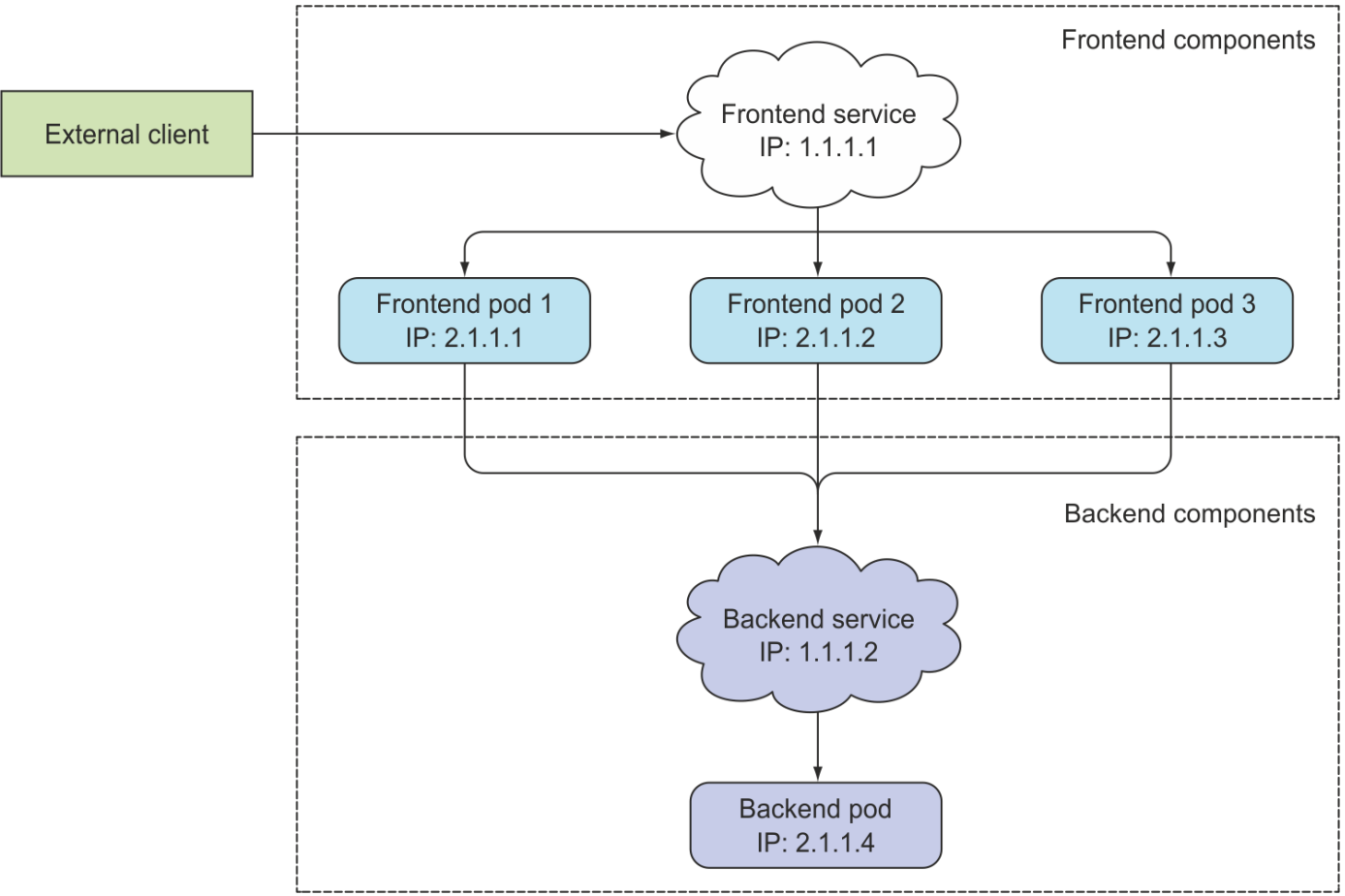
NodePort service



Services


Frontend components
External client
Frontend service
IP: 1.1.1.1
Frontend pod 1
IP: 2.1.1.1
Frontend pod 2
IP: 2.1.1.2
Frontend pod 3
IP: 2.1.1.3
Backend components
Backend service
IP: 1.1.1.2
Backend pod
IP: 2.1.1.4

Services



- A **service** created for the **frontend** pods and configuring it to be **accessible from outside** the cluster
  - This will **expose** a **single**, constant **IP** address through which **external clients** can **connect** to the pods.

- A **service** created for the **backend** pod,
  - This will create a **stable address** for the **backend** pod.
  - The service **address doesn't change** even if the **pod's IP** address **changes**


namespace: EDB
Ingress
https://api.edbtransfer.tech
Service
proxy-service:9999
Deployment
ReplicaSet
Pod
proxy:VERSION
container
Service
backoffice-service:8084
Service
client-service:8083
Service
backoffice-service:8082
Service
transfer-service:8081
Deployment
ReplicaSet
Pod
backoffice:VERSION
container
Pod
backoffice:VERSION
container
Deployment
ReplicaSet
Pod
client:VERSION
container
Pod
client:VERSION
container
Deployment
ReplicaSet
Pod
backoffice:VERSION
container
Pod
backoffice:VERSION
container
Deployment
ReplicaSet
Pod
transfer-service:VERSION
container
Pod
transfer-service:VERSION
container
Service
postgres:5432
Statefulset
Pod
master
postgres-0
container
Pod
postgres-1
container
Pod
postgres-2
container

Before Kubernetes and Virtualization

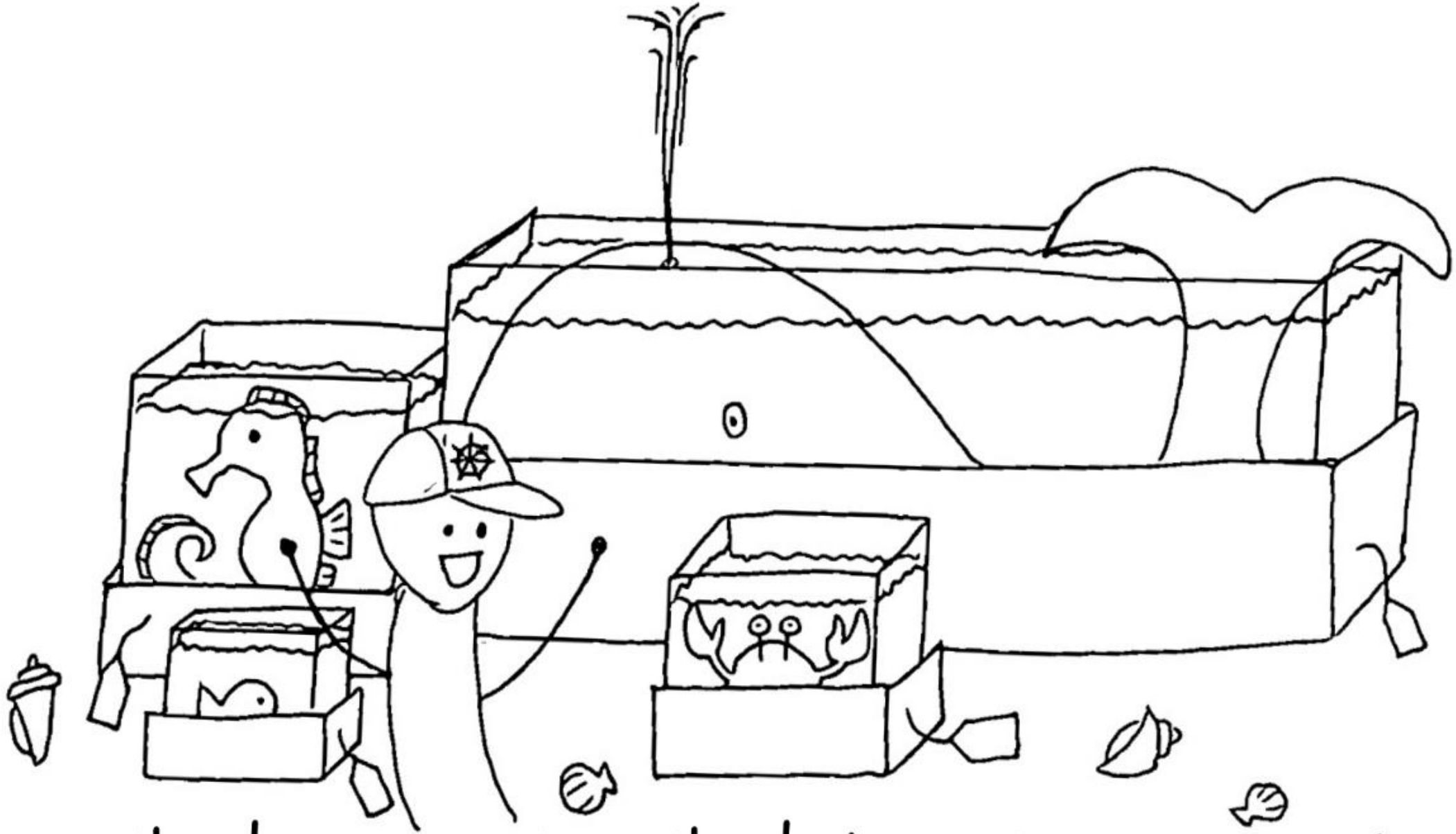
- Application
- Operating System
- Server

After Kubernetes

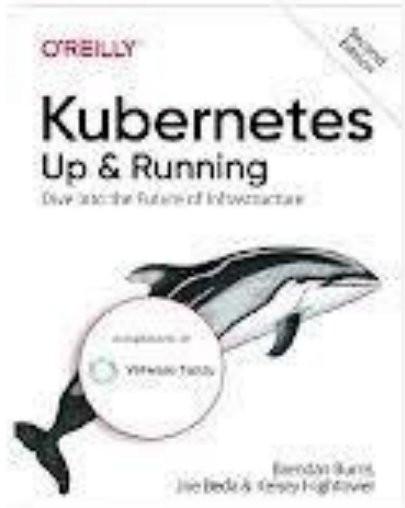
- Load Balancer
- Ingress
- Kube-Proxy
- Service Mesh
- Side Car
- Application
- Operating System
- Virtualization
- Server

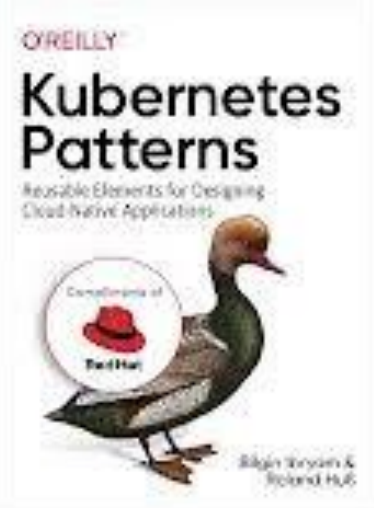

Running a bare metal Kubernetes cluster in production isn't stressing me out anymore.
— Mark, 22 years old
I'm pretty sure the application is somewhere around here
LOAD BALANCER
iNGRESS
KUBE-PROXY
Service Mesh
SIDE CAR
APPLICATION

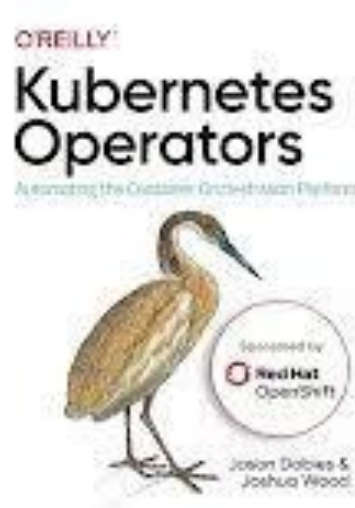
Thank you

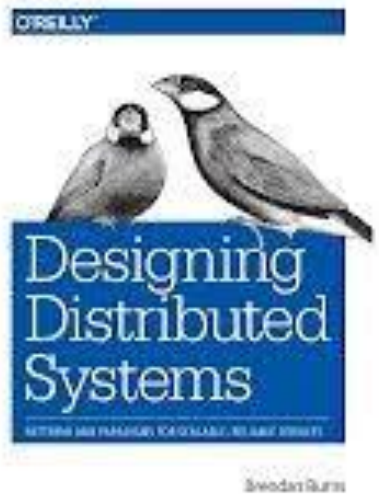


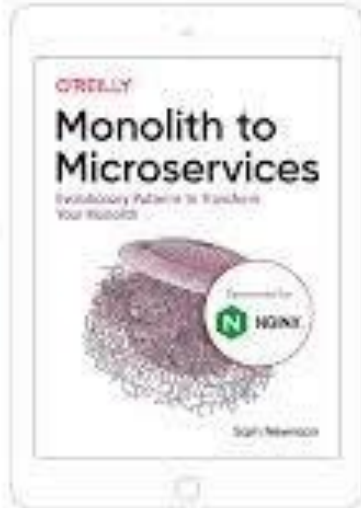
thanks for visiting the kubernetes aquarium!

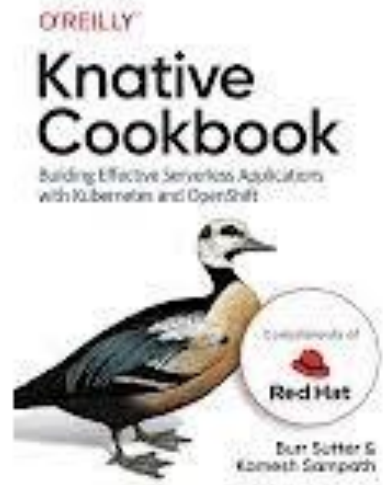

O'REILLY
Kubernetes
Up & Running


O'REILLY
Kubernetes
Patterns
Reusable Elements for Designing
Cloud Native Applications


O'REILLY
Kubernetes
Operators


O'REILLY
Designing
Distributed
Systems


O'REILLY
Monolith to
Microservices


O'REILLY
Knative
Cookbook
Building Effective Serverless Applications
with Kubernetes and OpenShift