

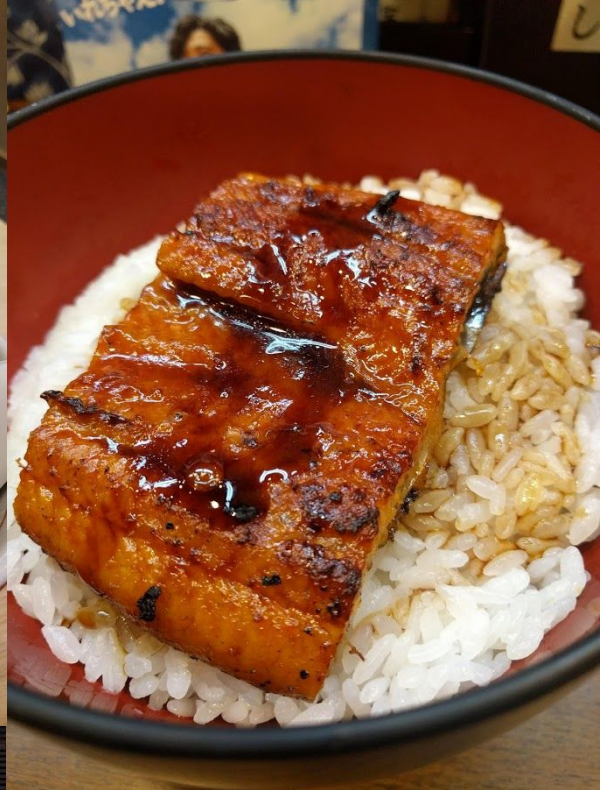


Pivotal.

Continuous Deployment With Spinnaker & Kubernetes

Paul Czarkowski

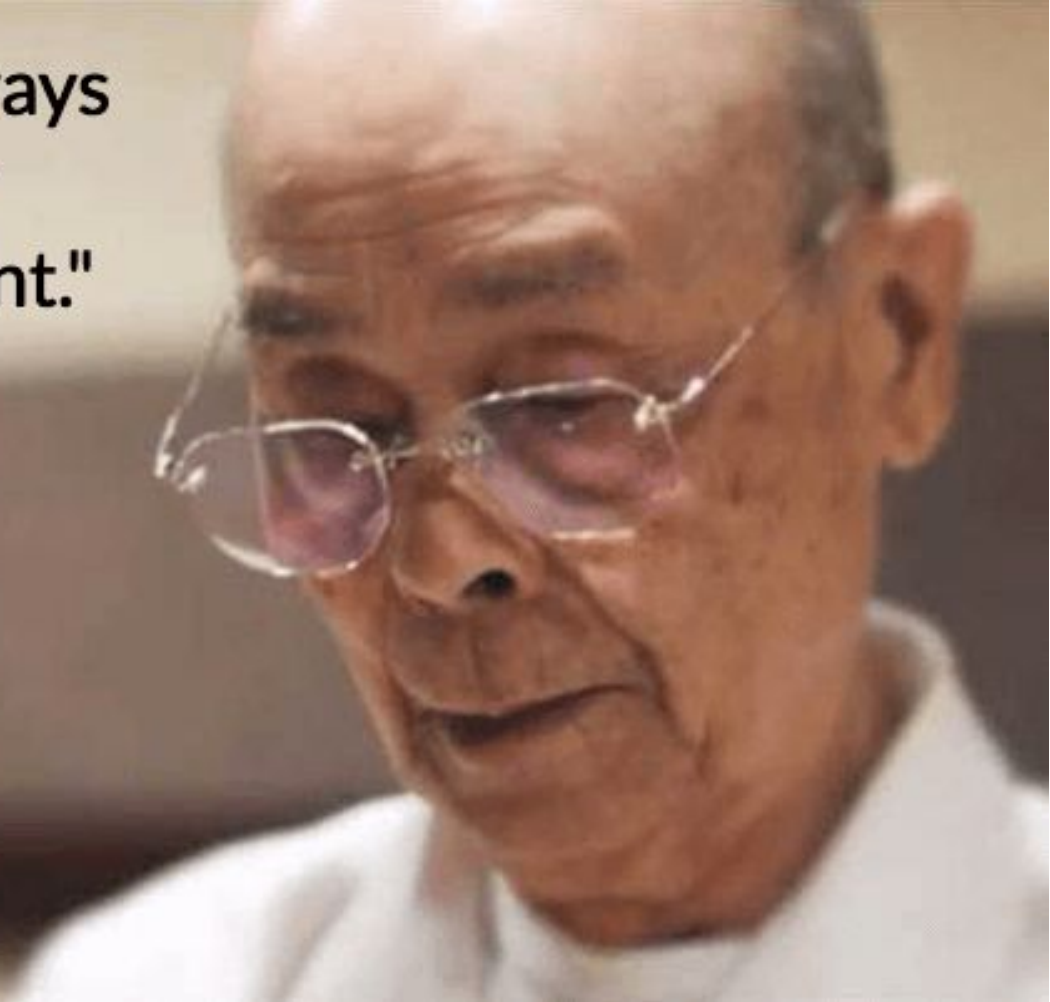
Twitter: @pczarkowski







**"There's always
room for
improvement."**



政

善

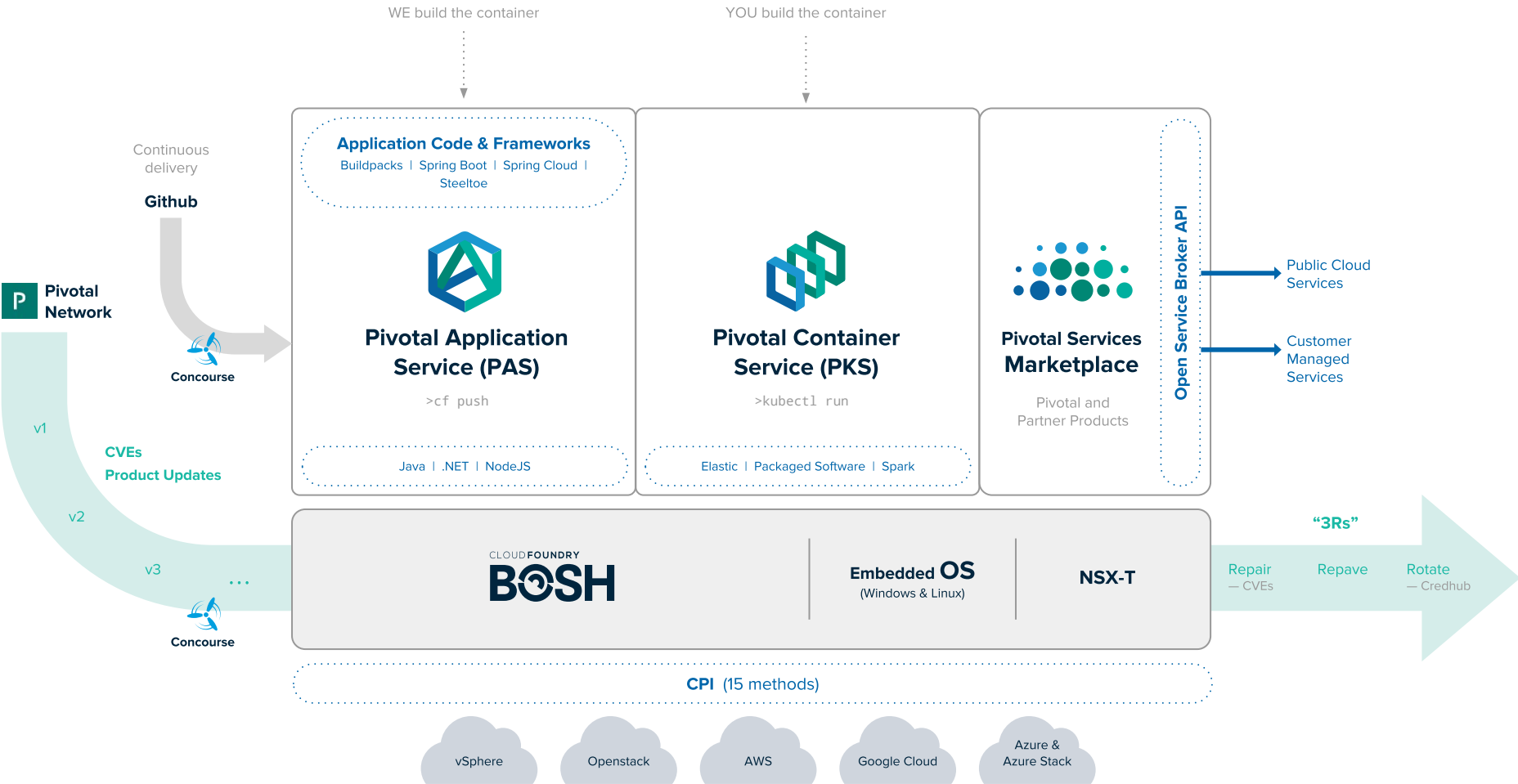
Agenda

- ~~Who am I?~~
- Pivotal Container Service (PKS)
- Kubernetes 101
- Kubernetes Manifests
- Helm Charts
- Spinnaker
- Q+A



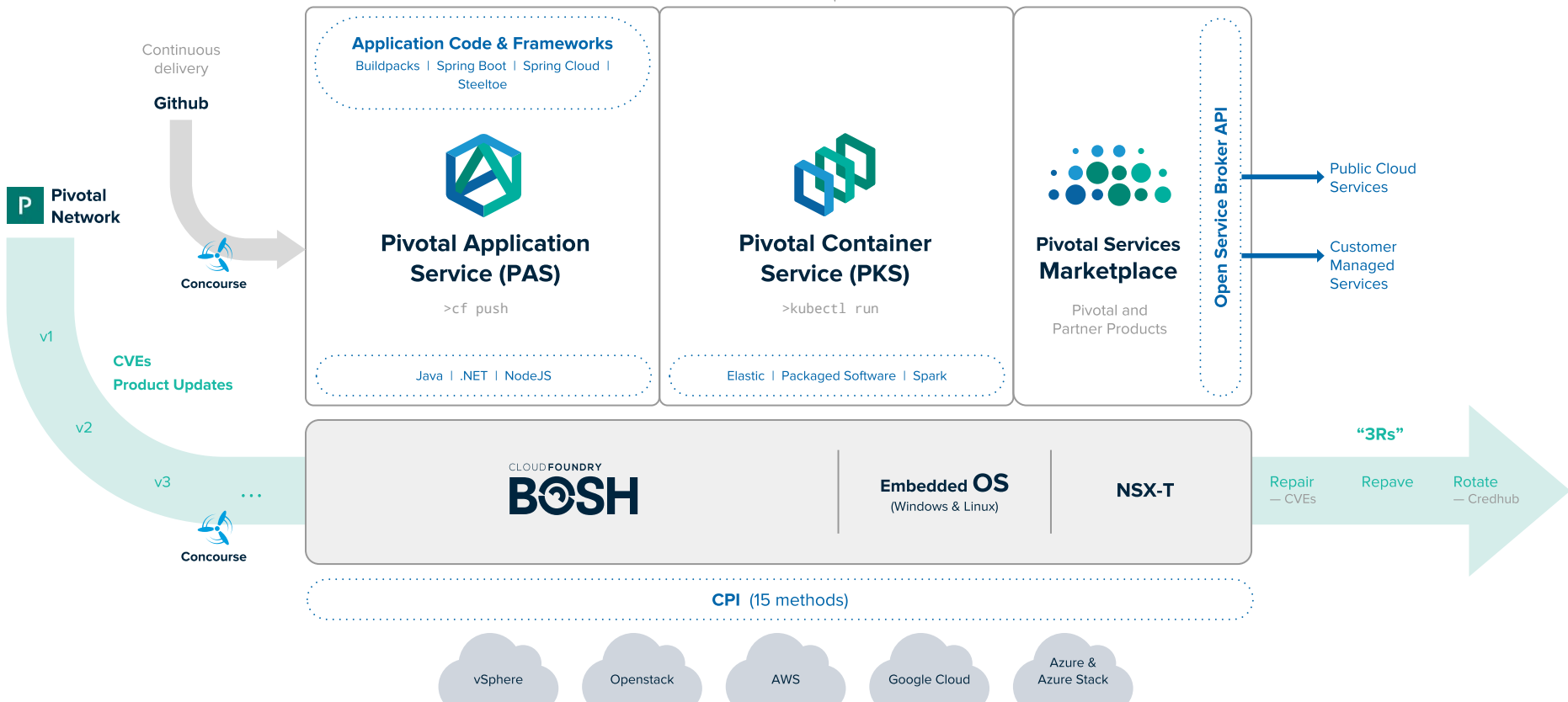
A group of people in a workshop setting. A man on the left is pointing at a wall covered in sticky notes. A group of people are sitting on stools in the center, listening. A man on the right is standing with his arms crossed, also listening. The background shows a modern office or workshop environment with large windows and a blackboard.

Pivotal Container Service



WE build the container

YOU build the container



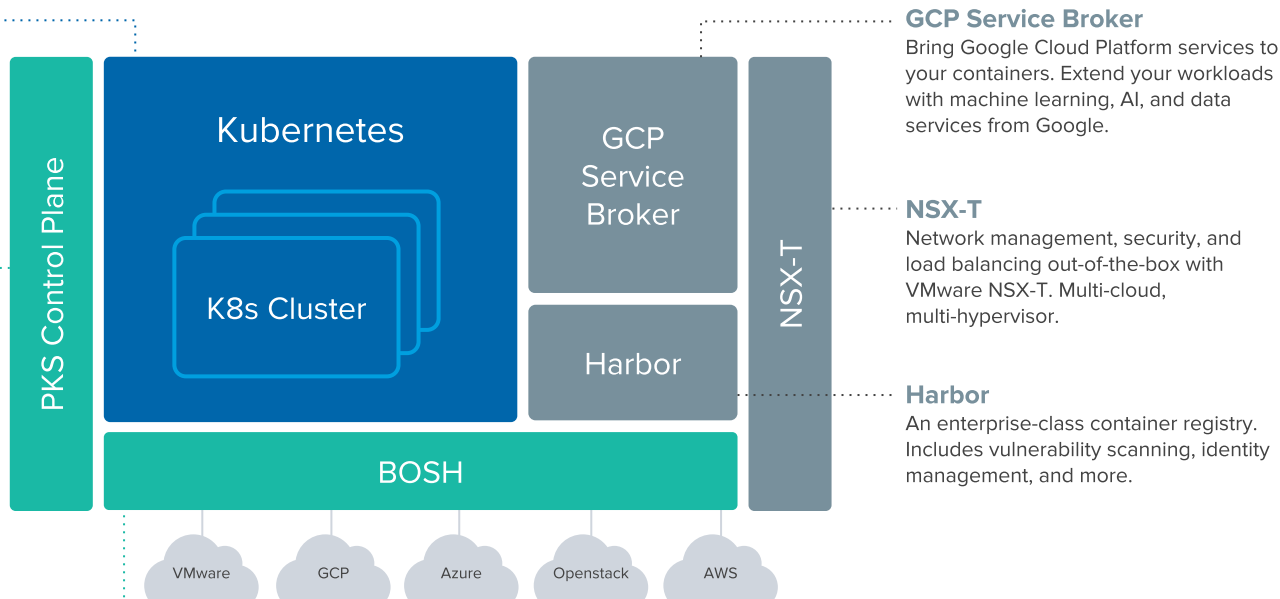
Enterprise-Grade Kubernetes

Built with open-source Kubernetes

Constant compatibility with the latest stable release of Google Kubernetes Engine—no proprietary extensions.

PKS Control Plane

Use the PKS CLI and API to create, operate, and scale your clusters.



GCP Service Broker

Bring Google Cloud Platform services to your containers. Extend your workloads with machine learning, AI, and data services from Google.

NSX-T

Network management, security, and load balancing out-of-the-box with VMware NSX-T. Multi-cloud, multi-hypervisor.

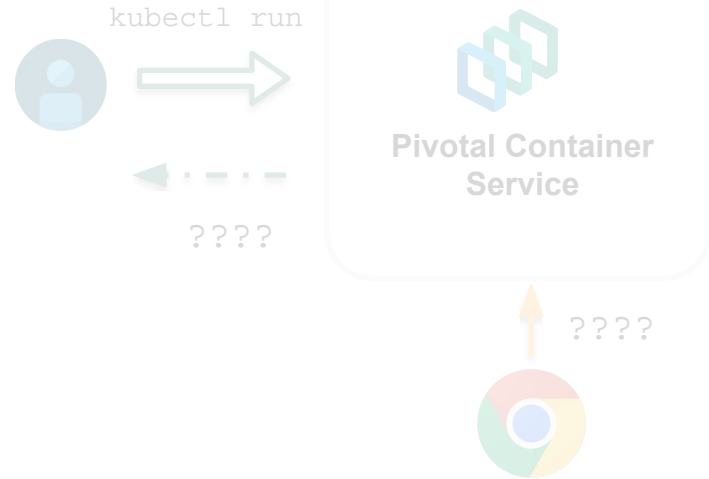
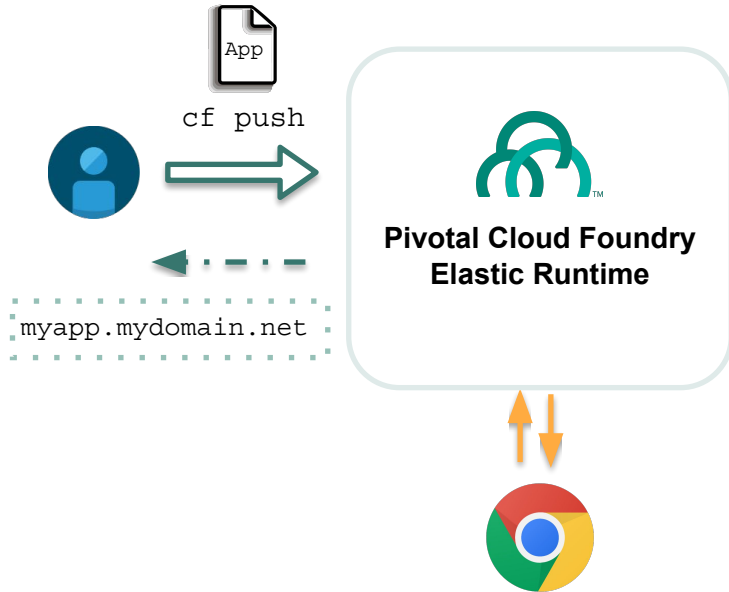
Harbor

An enterprise-class container registry. Includes vulnerability scanning, identity management, and more.

BOSH

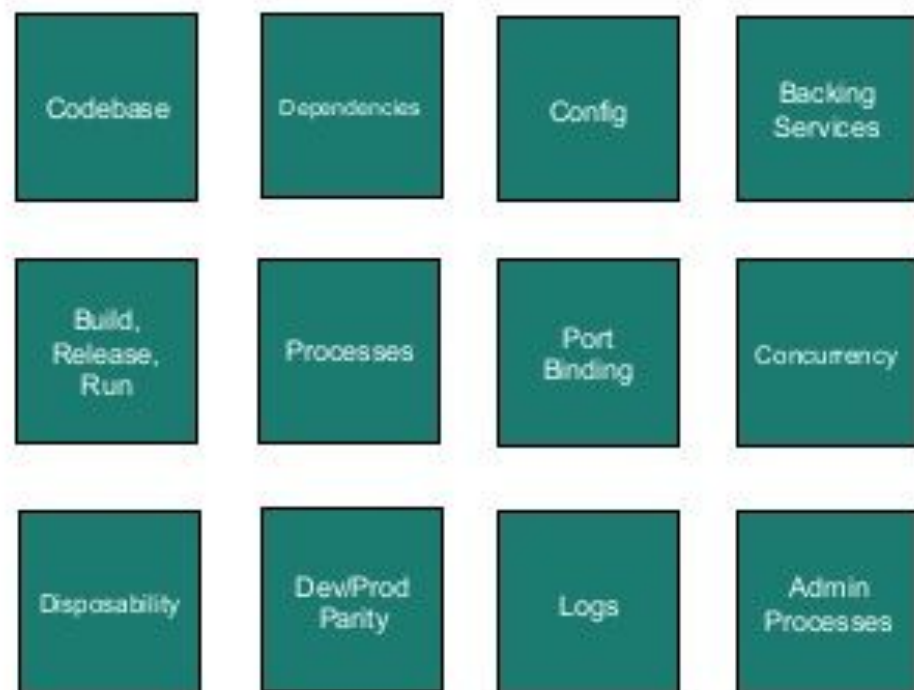
Reliable and consistent operational experience for any cloud.

App / Container Deployment, Services & Routing



Use 12 factor app principles to create cloud ready applications

- A set of best practices for developing and deploying cloud-native software.
- Practices translate into platform features and workflow requirements.

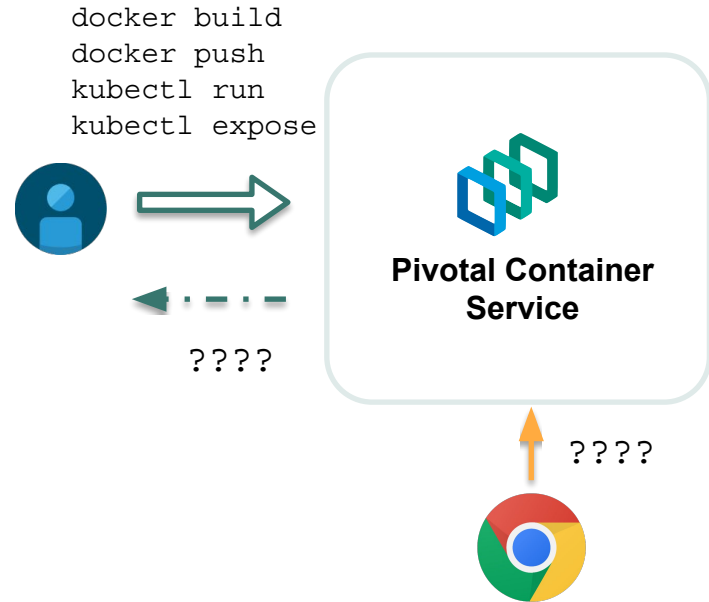


Use 12 factor app principles to create cloud ready applications

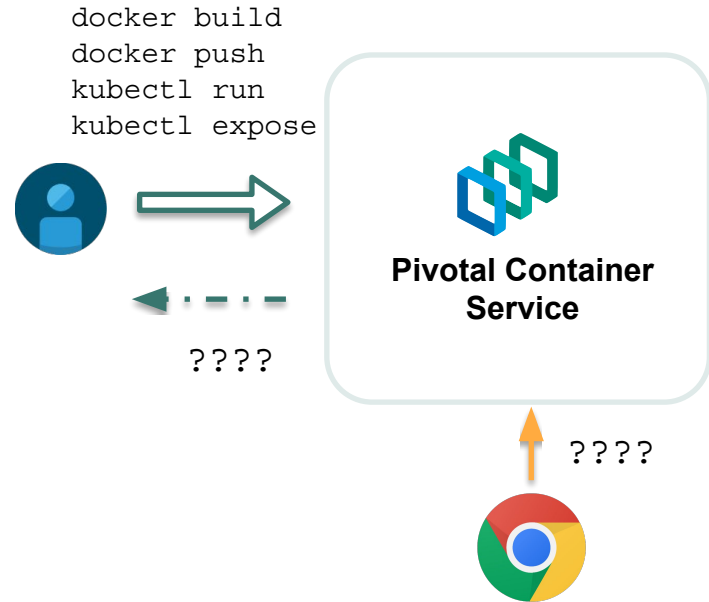
- A set of best practices for developing and deploying cloud-native software.
- Practices translate into platform features and workflow requirements.



App / Container Deployment, Services & Routing



App / Container Deployment, Services & Routing

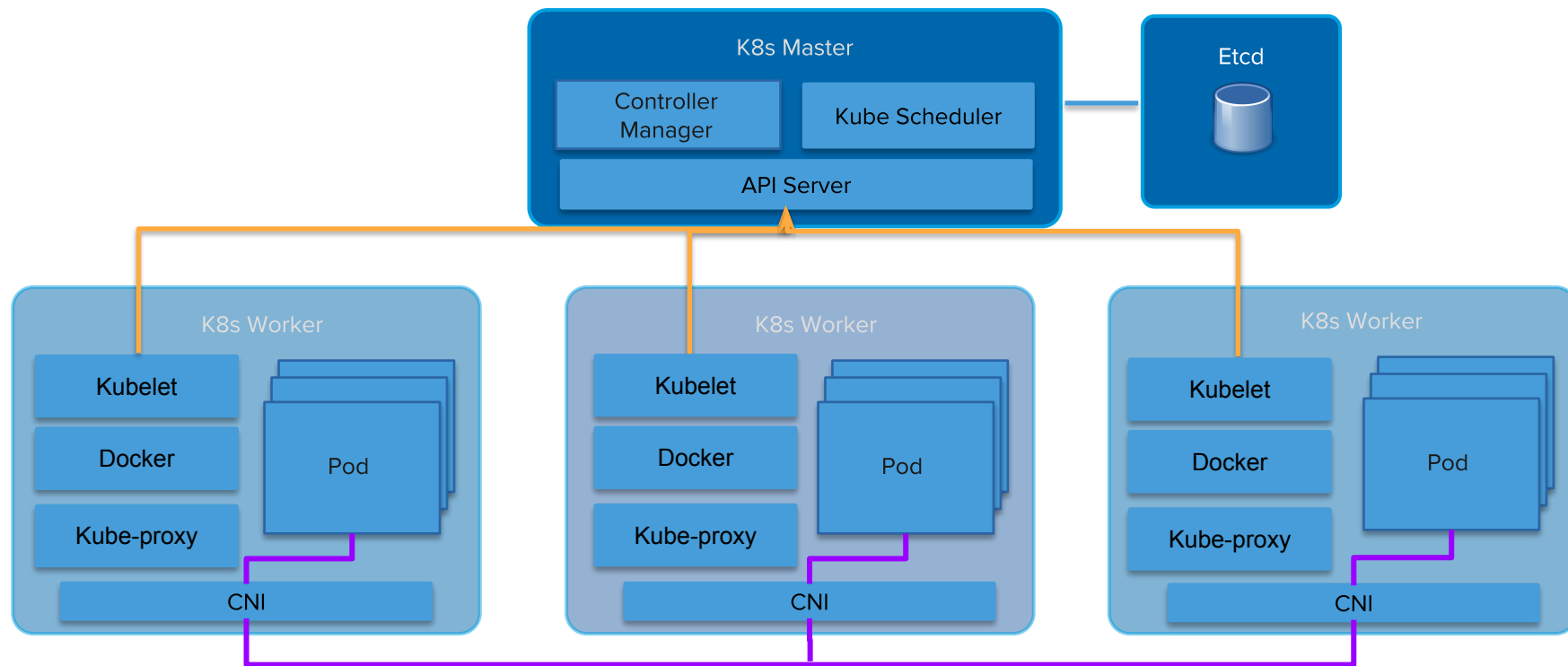




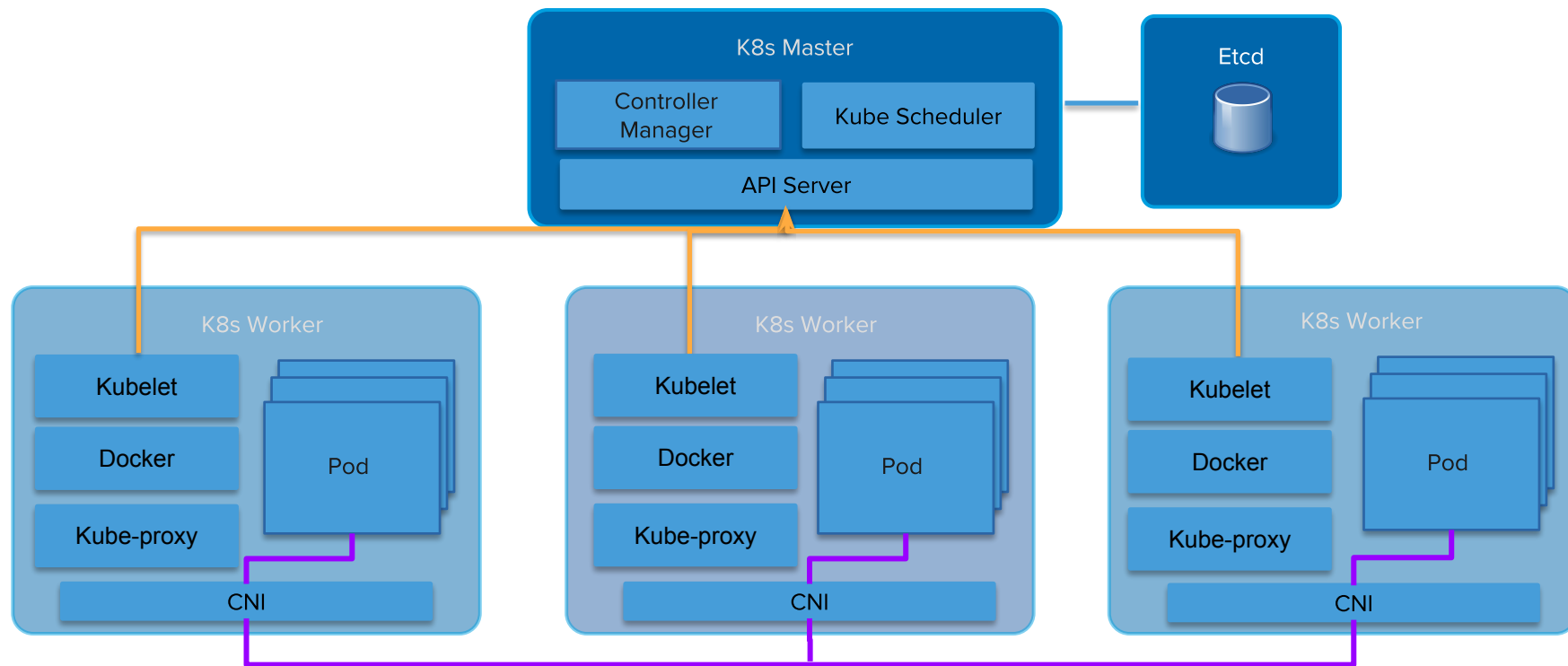
A group of people are in a workshop or meeting room. On the left, a man stands and points with a marker at a wall covered in many sticky notes. In the center, three people are seated on stools, looking towards the man pointing. On the right, two more people are seated at a table, also looking in the same direction. The room has a modern, open-plan feel with large windows in the background. The entire image is overlaid with a semi-transparent dark blue filter.

Kubernetes 101

Logical Kubernetes Architecture



Logical Kubernetes Architecture



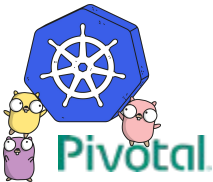
```
$ kubectl --help
```

kubectl controls the Kubernetes cluster manager.

Find more information at: <https://kubernetes.io/docs/reference/kubectl/overview/>

Basic Commands (Beginner):

create	Create a resource from a file or from stdin.
expose	Take a replication controller, service, deployment or pod and expose it as a new Kubernetes Service
run	Run a particular image on the cluster
set	Set specific features on objects



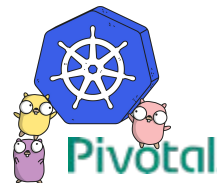
Kubernetes Manifest

apiVersion:

kind:

metadata:

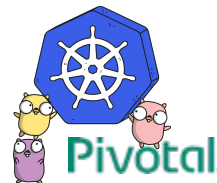
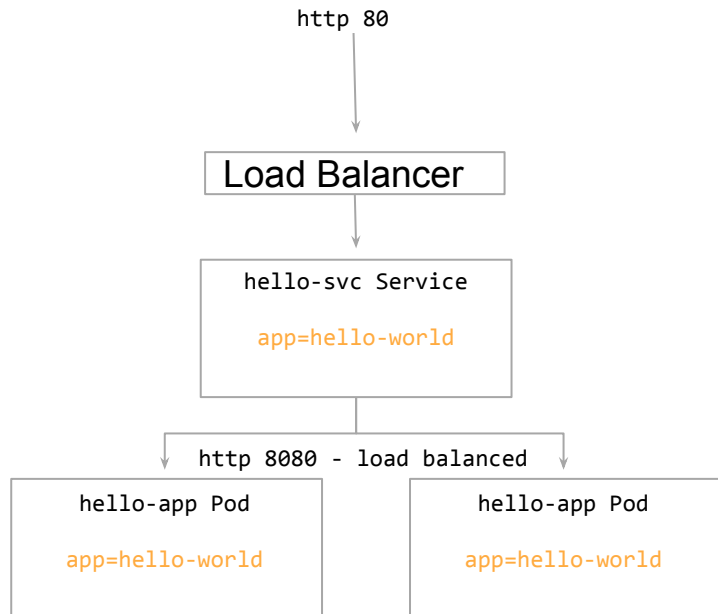
spec:



Kubernetes Manifest

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  labels:
    app: hello-world
  name: hello-app
spec:
  replicas: 2
  template:
    metadata:
      labels:
        app: hello-world
    spec:
      containers:
        - image: paulczar/hello-world
          name: hello-world
```

```
apiVersion: v1
kind: Service
metadata:
  name: hello-svc
spec:
  ports:
    - port: 80
      protocol: TCP
      targetPort: 8080
  selector:
    app: hello-world
  type: LoadBalancer
```



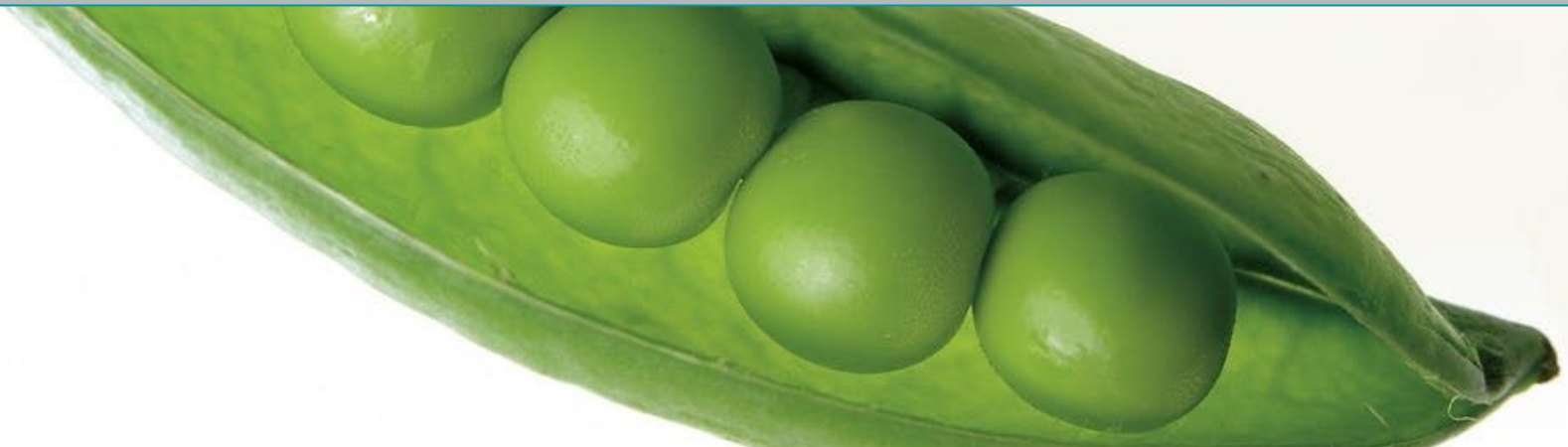


RESOURCES





POD



```
$ kubectl run hello \  
  --image=paulczar/go-hello-world
```

- `kubectl run` created a *deployment* “`deployments.apps/hello`”

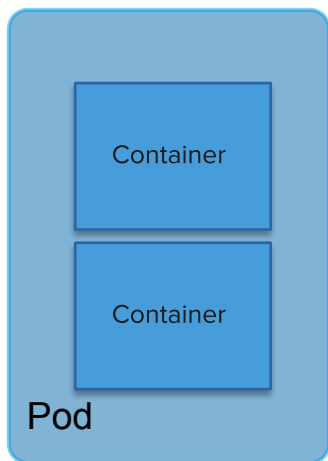
NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
<code>deployment.apps/hello</code>	1	1	1	1	1m

- The deployment created a *replicaset* “`replicaset.apps/hello-64f6bf9dd4`”

NAME	DESIRED	CURRENT	READY	AGE
<code>replicaset.apps/hello-64f6bf9dd4</code>	1	1	1	1m

- Which created a *pod* “`pod/hello-64f6bf9dd4-tq5dq`”

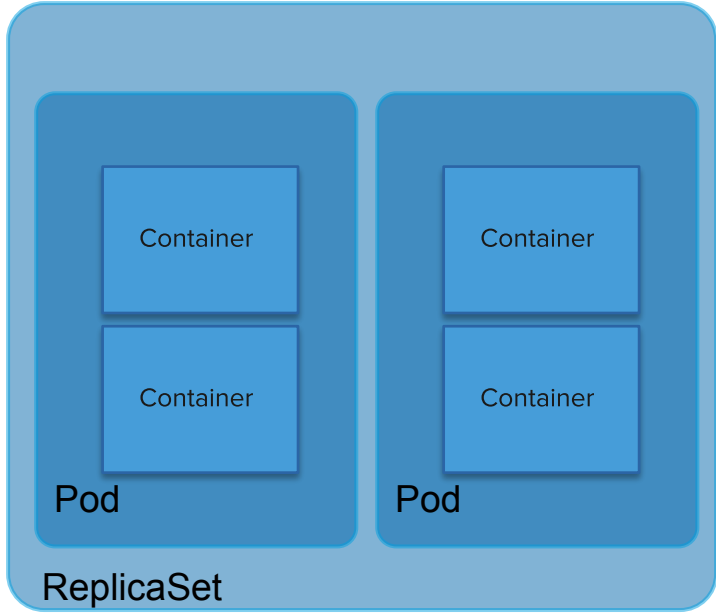
NAME	READY	STATUS	RESTARTS	AGE
<code>pod/hello-64f6bf9dd4-tq5dq</code>	1/1	Running	0	2s



Pod

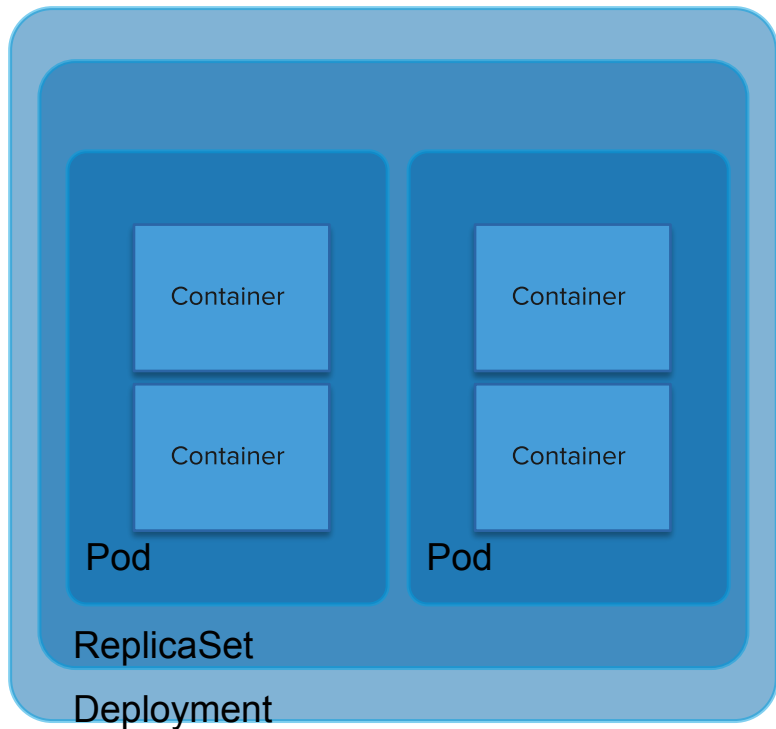
one or more application containers that are **tightly coupled**, sharing **network** and **storage**.

*Example: a web front-end **Pod** that consists of an NGINX container and a PHP-FPM container with a shared unix socket and a “init” container to transform their config files based on environment variables.*



ReplicaSet

Extends Pod resource to run and maintain a specific number of copies of a pod.



Deployment

a controller that ensures a set number of **replicas** of a **Pod** is running and provides **update and upgrade workflows** for your **Pods**.

Example: cloud native Node app that scales horizontally and upgrades 2 pods at a time.

```
$ kubectl scale --replicas=3 \  
deployment/hello
```

```
$ kubectl scale --replicas=3 deployment/hello
deployment.extensions/hello scaled
```

```
$ kubectl get all
```

NAME	READY	STATUS	RESTARTS	AGE
pod/hello-64f6bf9dd4-2bndq	1/1	Running	0	15m
pod/hello-64f6bf9dd4-4kq9l	0/1	ContainerCreating	0	2s
pod/hello-64f6bf9dd4-8lkcs	1/1	Running	0	5s

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/hello	3	3	2	3	16m

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/hello-64f6bf9dd4	3	3	2	16m

```
$ kubectl edit deployment hello
```

```
...
```

```
spec:
```

```
  containers:
```

```
  - env:
```

```
    - name: MESSAGE
```

```
      value: HELLO I LOVE YOU!!!!
```

```
  image: paulczar/go-hello-world
```

```
  imagePullPolicy: Always
```

```
  name: hello
```

```
$ kubectl get all
```

NAME	READY	STATUS	RESTARTS	AGE
pod/hello-5c75b546c7-4lwnn	1/1	Running	0	1m
pod/hello-5c75b546c7-bwxxq	1/1	Running	0	1m
pod/hello-5c75b546c7-sl2pg	1/1	Running	0	1m

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/hello	3	3	3	3	23m

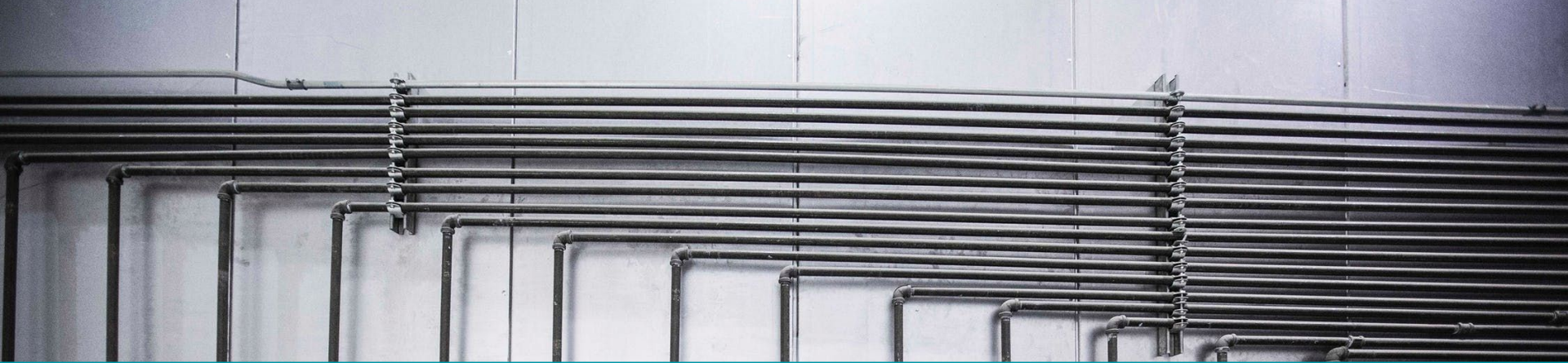
NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/hello-5c75b546c7	3	3	3	1m
replicaset.apps/hello-64f6bf9dd4	0	0	0	23m

```
$ kubectl port-forward deployment/hello 8080
```

```
Forwarding from 127.0.0.1:8080 -> 8080
```

```
$ curl localhost:8080
```

```
<html><head><title>HELLO I LOVE YOU!!!!</title></head><body>HELLO I LOVE  
YOU!!!!</body></html>
```

Service



```
$ kubectl expose deployment \  
    hello --type=LoadBalancer \  
    --port 80 --target-port 8080
```

```
kubect1 expose deployment hello
```

- creates a *service* with a *ClusterIP* that acts as an internal loadbalancer to all *Pods* in the “hello” *deployment*

```
--type=LoadBalancer
```

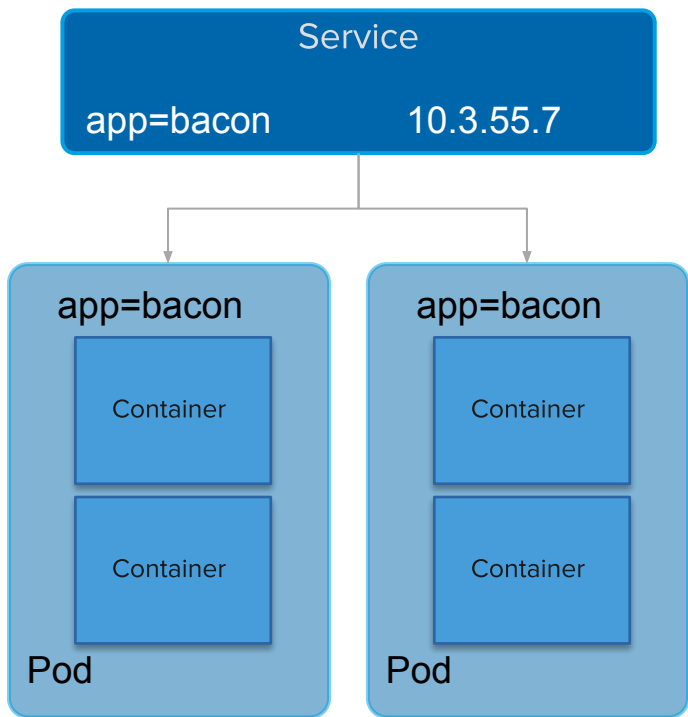
- Creates a *NodePort*
- Configures a *LoadBalancer* to access the *Pods* via the *NodePort*

```
$ kubect1 get services
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
hello	LoadBalancer	10.39.248.123	35.184.17.129	80:30468/TCP	5m

```
$ curl 35.184.17.129
```

```
<html><head><title>HELLO I LOVE YOU!!!!</title></head><body>HELLO I LOVE  
YOU!!!!</body></html>
```

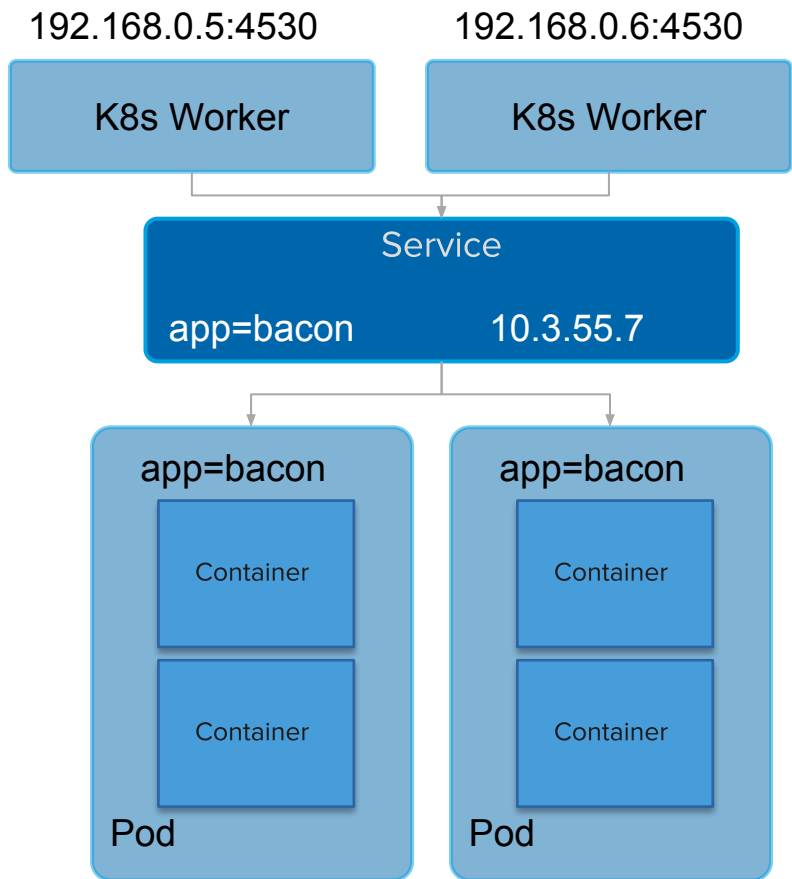


Service

track **Pods** based on metadata and provides connectivity and service discovery (DNS, Env variables) for them.

Type

ClusterIP (default) exposes service on a **cluster-internal IP**.

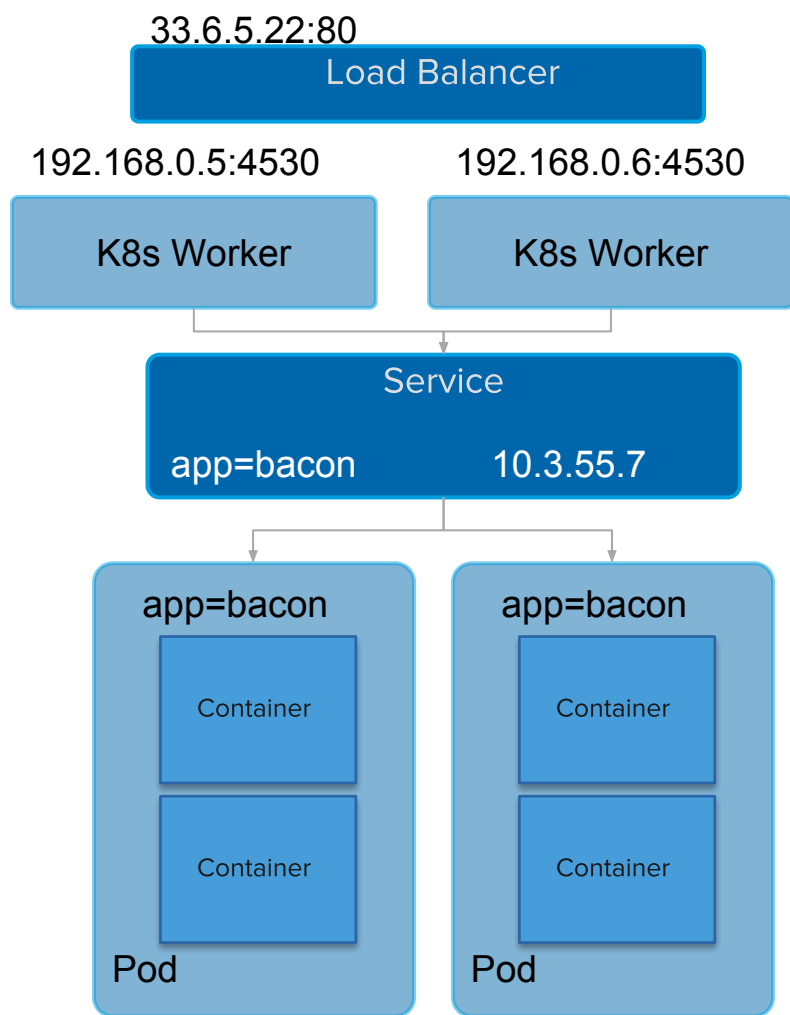


Service

track **Pods** based on metadata and provides connectivity and service discovery (DNS, Env variables) for them.

Type

NodePort extends **ClusterIP** to expose services on each node's IP via a **static port**.

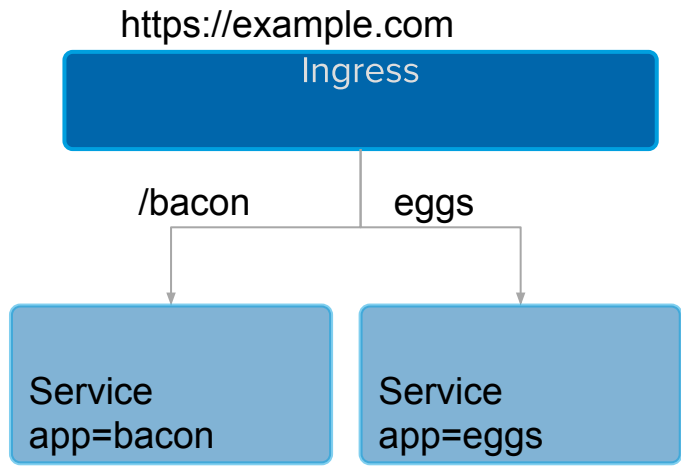


Service

track **Pods** based on metadata and provides connectivity and service discovery (DNS, Env variables) for them.

Type

LoadBalancer extends **NodePort** to configure a cloud provider's load balancer using the **cloud-controller-manager**.



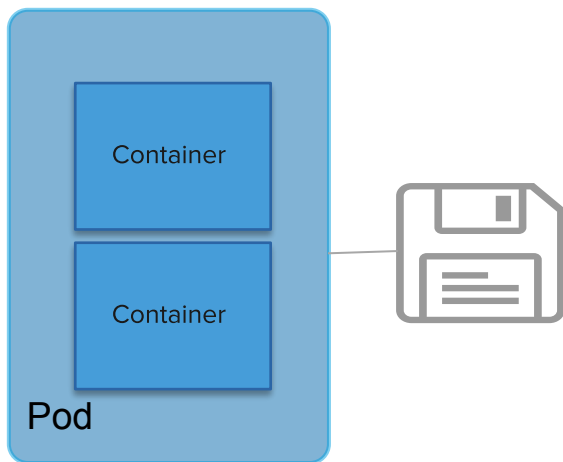
Ingress

a controller that manages an external entity to provide load balancing, SSL termination and name-based virtual hosting to services based on a set of rules.



Volume





Volume

Is [effectively] a **Directory**, possibly with data in it, available to **all containers** in a Pod.

Usually **Shares lifecycle** of a Pod (Created when Pod is created, destroyed when Pod is destroyed).

Persistent Volumes outlive Pods.

Can be mounted from local disk, or from a network storage device such as a EBS volume, iscsi, NFS, etc.



Config Map / Secret



```
$ kubectl create configmap hello \  
  --from-file=index.html
```

```
kubectl create configmap hello --from-file=index.html
```

- creates a *configmap* called “*hello*” containing the contents *index.html*

```
$ kubectl get configmap hello -o yaml
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: hello
data:
  index.html: "<html>\n<head>\n\t<title>Hello to my
friends</title>\n</head>\n<body>\n\tHello
to my friends\n</body>\n</html>\n\n"
```



```
kubectl create secret generic hello --from-file=index.html
```

- creates a *secret* called “*hello*” containing a *base64* hash of contents *index.html*

```
$ kubectl get secret hello -o yaml
```

```
apiVersion: v1
```

```
kind: Secret
```

```
metadata:
```

```
  name: hello
```

```
data:
```

```
  index.html:
```

```
PGh0bWw+CjxoZWFKPgoJPHRpdGx1Pkh1bGxvIHRvIG15IGZyaWVuZHM8L3RpdGx1Pgo8L2h1YWQ+Cjxib2R5  
PgoJSGVsbG8gdG8gbXkgZnJpZW5kcwo8L2JvZHK+CjwvaHRtbD4KCg==
```

ConfigMaps/Secrets (user-data)

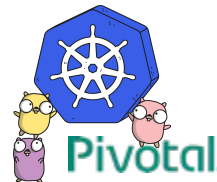
Provides **key-value pairs** to be injected into a **pod** much like user-data is injected into a Virtual Machine in the cloud.

Allows you to do **last minute configuration** of applications running on Kubernetes such as setting a database host, or a admin password.

ConfigMaps store values as **strings**, **Secrets** store them as **byte arrays** (serialized as base64 encoded strings).

Secrets are [currently] **not encrypted** by default. This is likely to **change**.

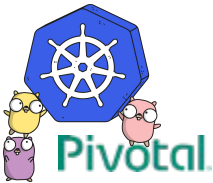
Can be injected as files in a Volume, or as Environment Variables.

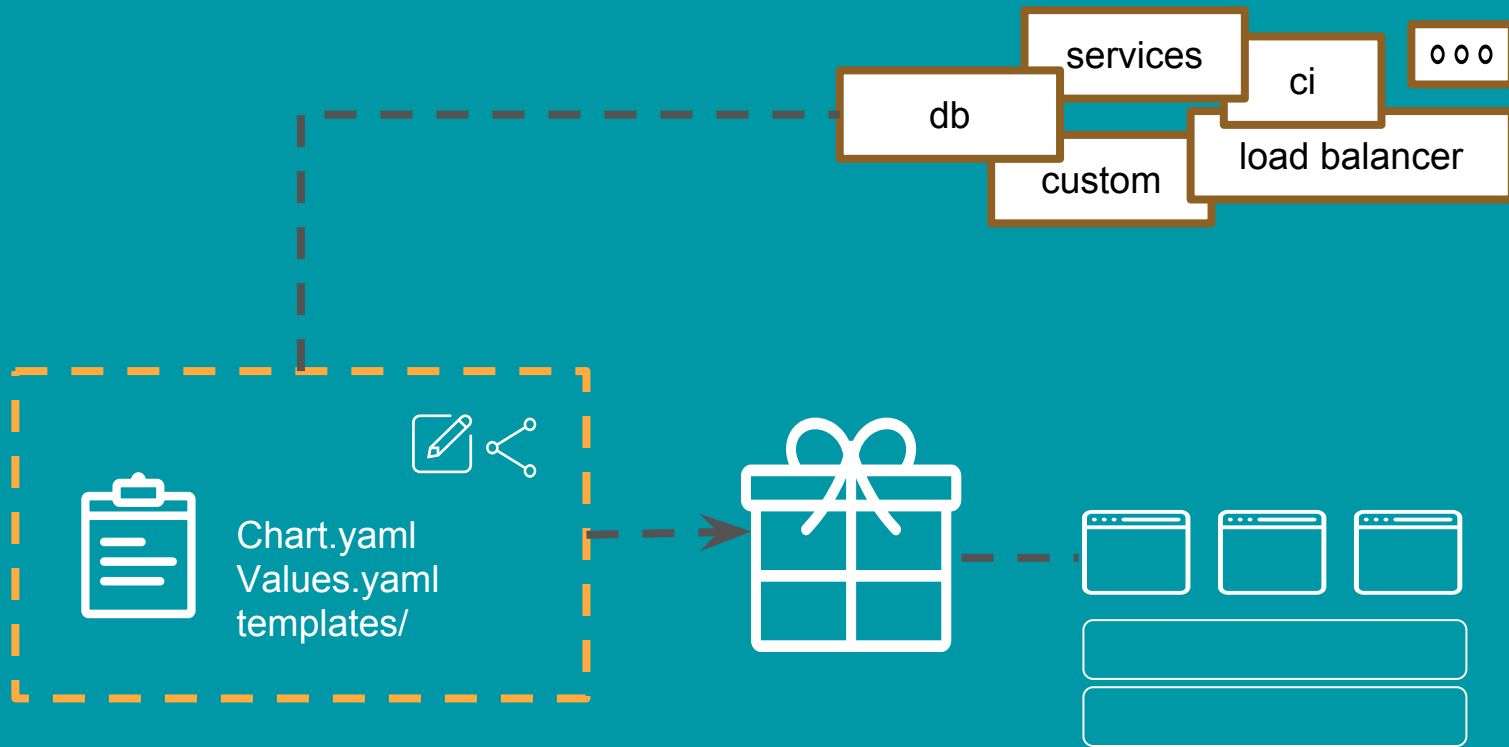


A group of people are in a workshop or meeting room. A man on the left is standing and pointing at a wall covered in many sticky notes. Several other people are sitting on stools, looking towards him. A man on the right is standing with his arms crossed, also looking towards the group. The room has a modern, open-plan feel with large windows in the background.

Helm

Helm is the **best** way to
find, share, and use
software built for ***Kubernetes***





 Secure | <https://hub.kubeapps.com>



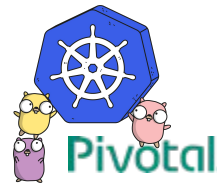
Kubeapps Hub

Discover & launch great
Kubernetes-ready apps

Search charts

Wordpress, Jenkins, Kubeless...

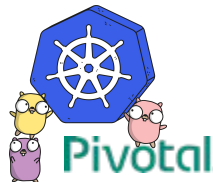
231 charts ready to deploy



```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: {{ .Chart.name }}-app
  labels:
    app: {{ .Chart.name }}
...
...
spec:
  containers:
    - image: paulczar/hello-world
      name: hello-world
      volumeMounts:
        - name: config
          mountPath: /etc/hello
  volumes:
    - name: config
      configMap:
        name: {{ .Chart.name }}-cm
```

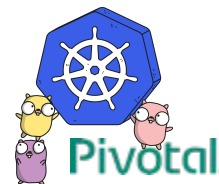
```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Chart.name }}-cm
data:
  db: {{ .Value.db }}
```

```
apiVersion: v1
kind: Service
metadata:
  name: {{ .Chart.name }}-svc
  labels:
    app: {{ .Chart.name }}-world
spec:
  ports:
    - port: {{ .Value.port }}
      protocol: TCP
      targetPort: 8080
  selector:
    app: {{ .Chart.name }}-world
  type: NodePort
```



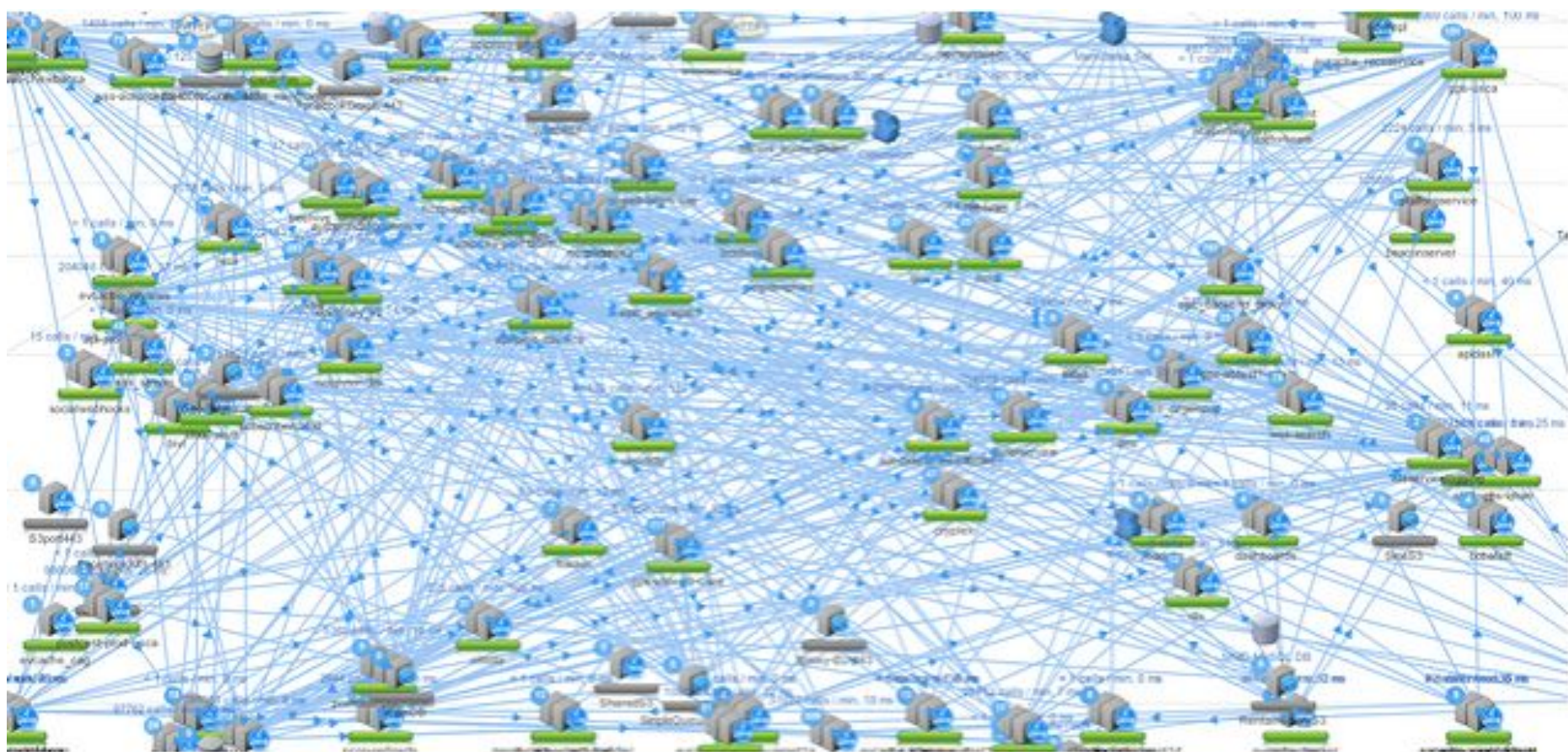
```
$ helm install --name staging . \  
  --set db='user:pass@staging.mysql/dbname'
```

```
$ helm install --name production . \  
  --set db='user:pass@production.mysql/dbname'
```

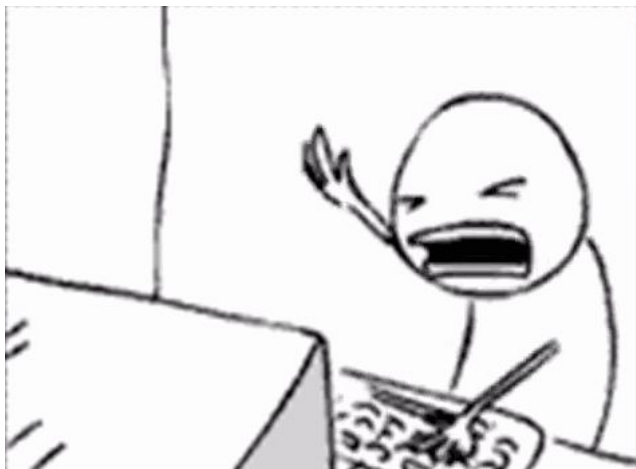


A group of people in a workshop setting. One person is standing and pointing at a wall covered in sticky notes. Several others are sitting on stools, looking towards the speaker. The scene is dimly lit with a blue tint.

Spinnaker



<https://medium.com/netflix-techblog/announcing-ribbon-tying-the-netflix-mid-tier-services-together-a89346910a62>



Pivotal

<https://giphy.com/gifs/frustrated-keyboard-g8GfH3i5F0hby>

Amazon EC2





+



Cluster Management

- Server Group
- Cluster
- Applications
- Load Balancer
- Firewall

Pipelines

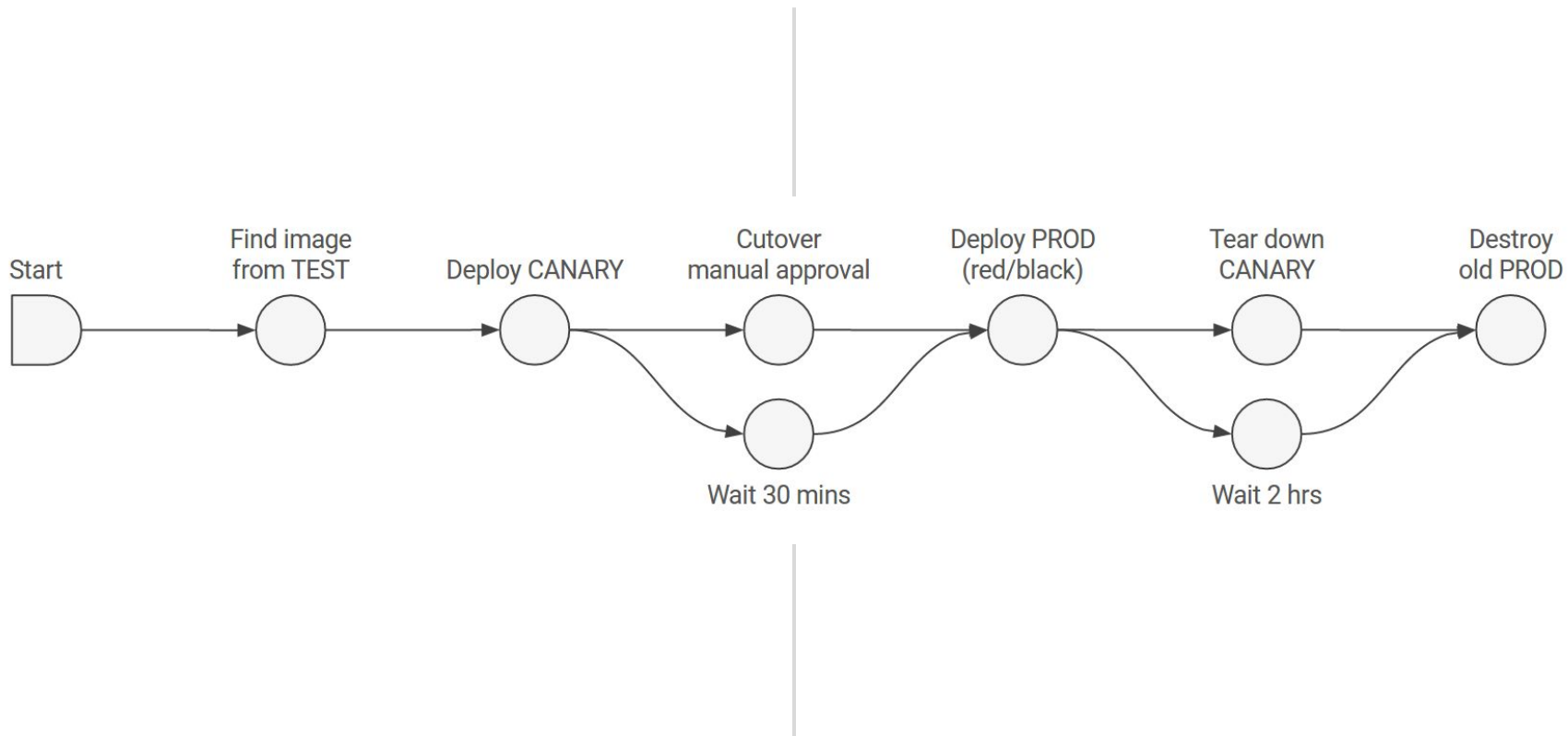
- Pipeline
- Stage
- Deployment Strategies

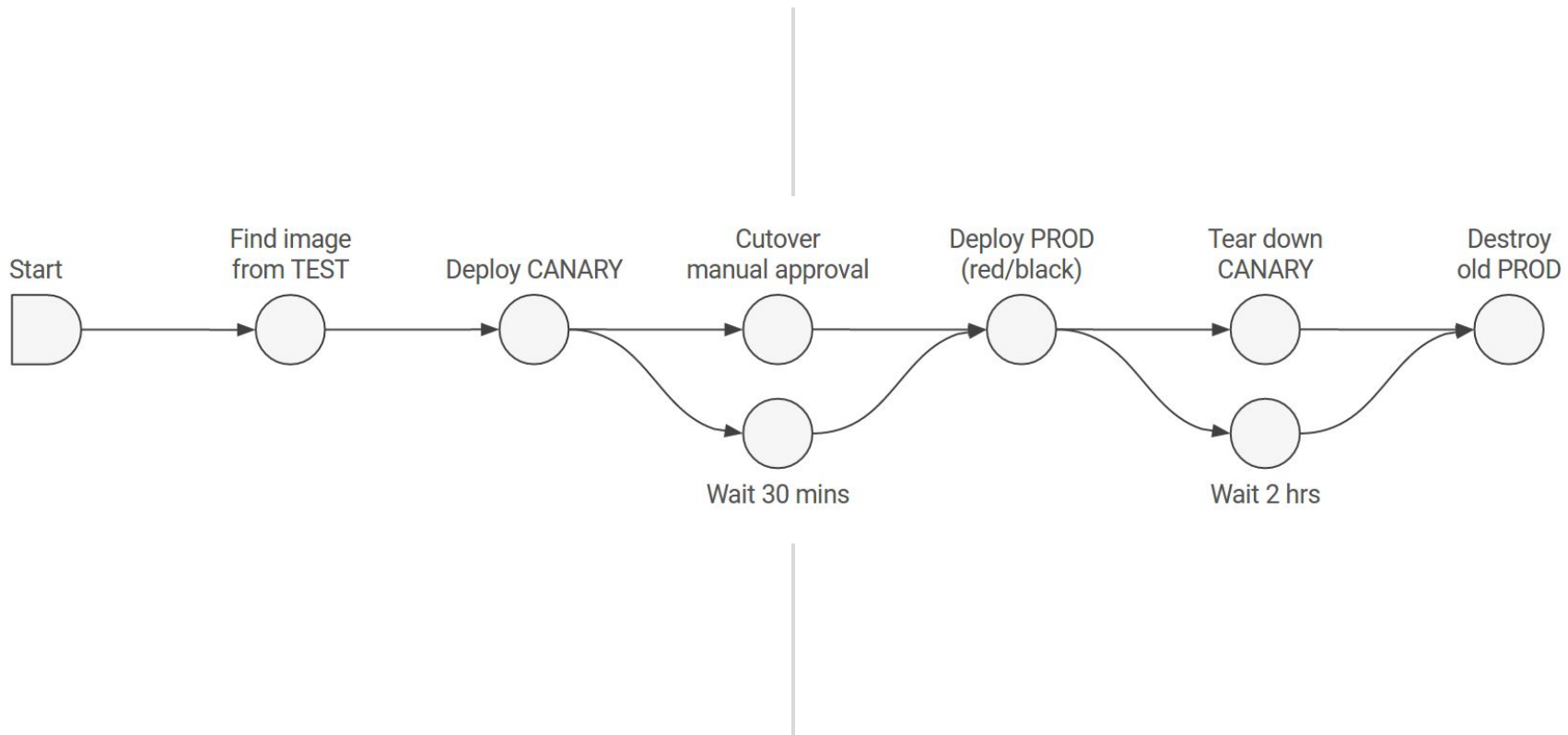
Multi-Cloud Inventory

- Server Group
- Cluster
- Applications
- Load Balancer
- Firewall

Actions and Reactions

- Pipeline
- Stage
- Deployment Strategies

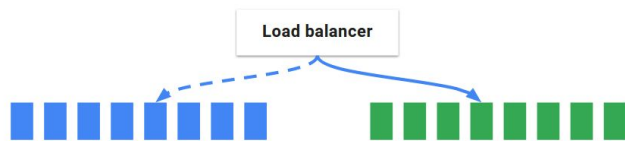




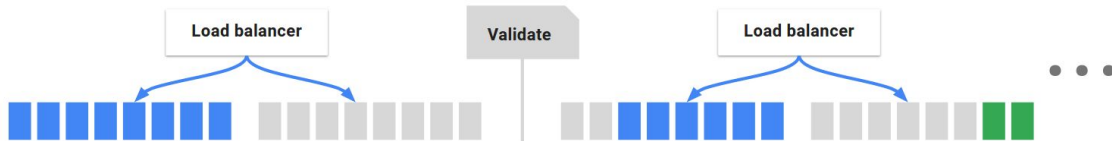
Deployment Strategies

- Serve
- Cluster
- Applic
- Load
- Firew

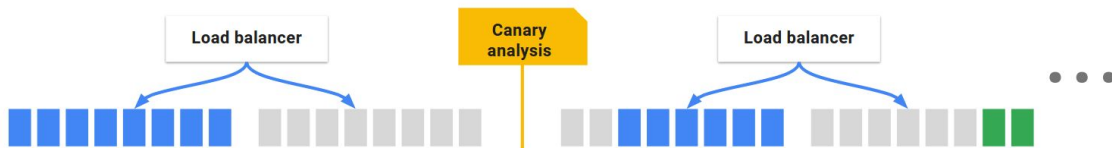
Red/black
(Blue/green)

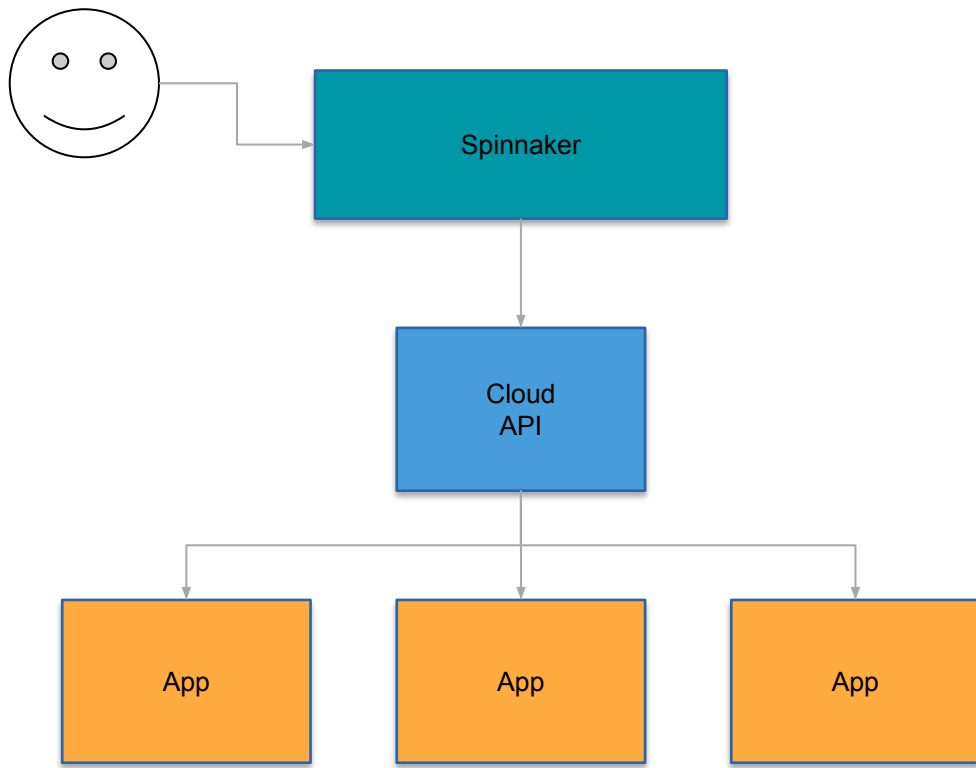


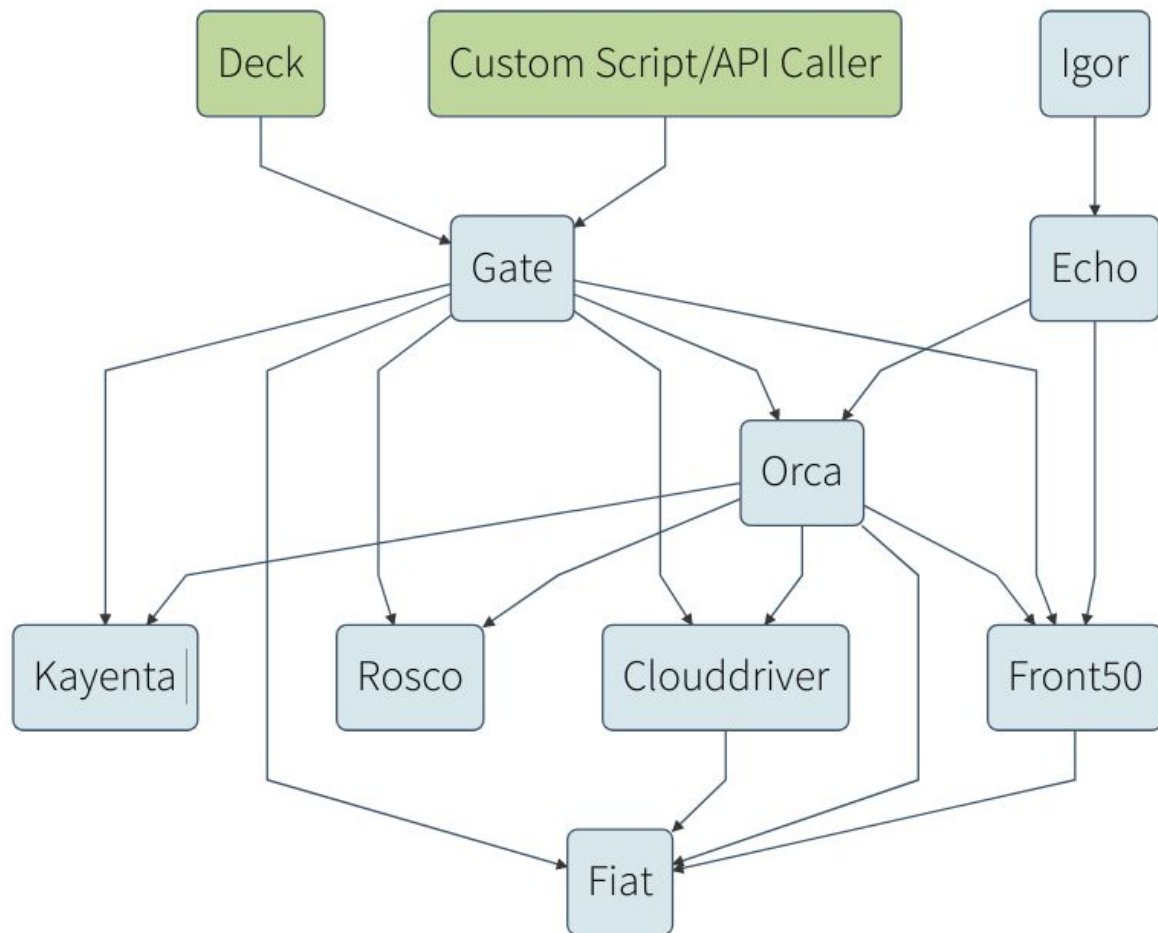
Rolling red/black



Canary







Halyard





