DEPLOYMENTS EXPOSED - SERVICES

SERVICES WITHOUT SELECTORS

Services can also abstract other kinds of backends besides kubernetes pods

For example:

- You want to have an external database cluster in production, but in test you use your own databases.
- You want to point your service to a service in another Namespace or on another cluster.
- You are migrating your workload to Kubernetes and some of your backends run outside of Kubernetes.

The corresponding Endpoints object will not be created.

DEPLOYMENTS EXPOSED - SERVICES

1 ClusterIP

- Exposes the service on a cluster-internal IP
- Makes the service only reachable from within the cluster
- Default service type

2 NodePort

- Exposes the service on each Node's IP at a static port
- You'll be able to contact the NodePort service, from outside the cluster
- <NodeIP>:<NodePort>

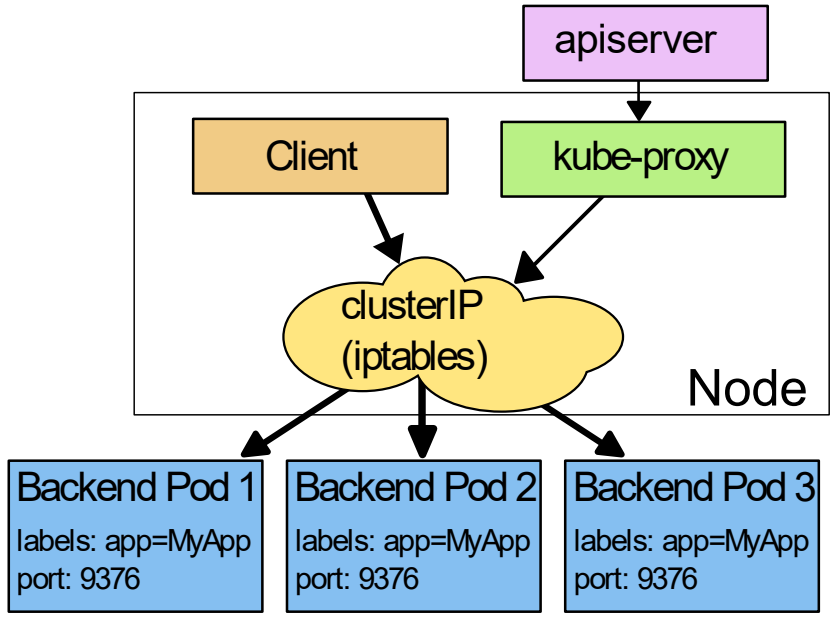
3 LoadBalancer

- Exposes the service externally using a cloud provider's load balancer

DEPLOYMENTS EXPOSED – SERVICES - CLUSTERIP

ClusterIP

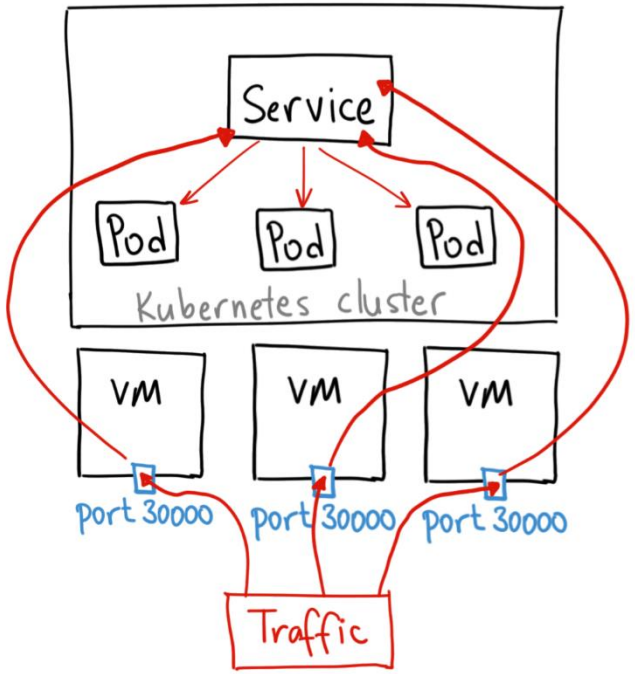
```yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    app: web
  name: web
spec:
  type: ClusterIP
  selector:
    app: web
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
```



DEPLOYMENTS EXPOSED – SERVICES - NODEPORT

NodePort

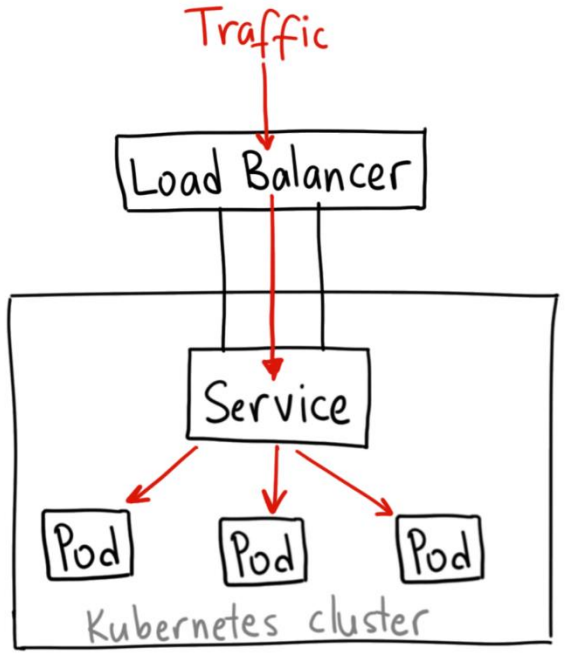
```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: NodePort
  selector:
    app: MyApp
  ports:
    - port: 80
      targetPort: 80
      nodePort: 30007
```



DEPLOYMENTS EXPOSED – SERVICES - LOADBALANCER

LoadBalancer

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
  type: LoadBalancer
```



DEPLOYMENTS EXPOSED - INGRESS

1. Namespaces
2. Pods and Containers
3. Deployment
4. Services
5. **Ingress**
6. Selector, Labels and Ports
6. Network plugins

DEPLOYMENTS EXPOSED - INGRESS

INGRESS

- Provides services with externally-reachable URLs
- Load balance traffic
- Terminates SSL
- Offers name based virtual hosting
- Requires Ingress controller

DEPLOYMENTS EXPOSED - INGRESS

TYPES OF INGRESS

Single Service Ingress

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: single-ingress
spec:
  backend:
    serviceName: testsvc
    servicePort: 80
```

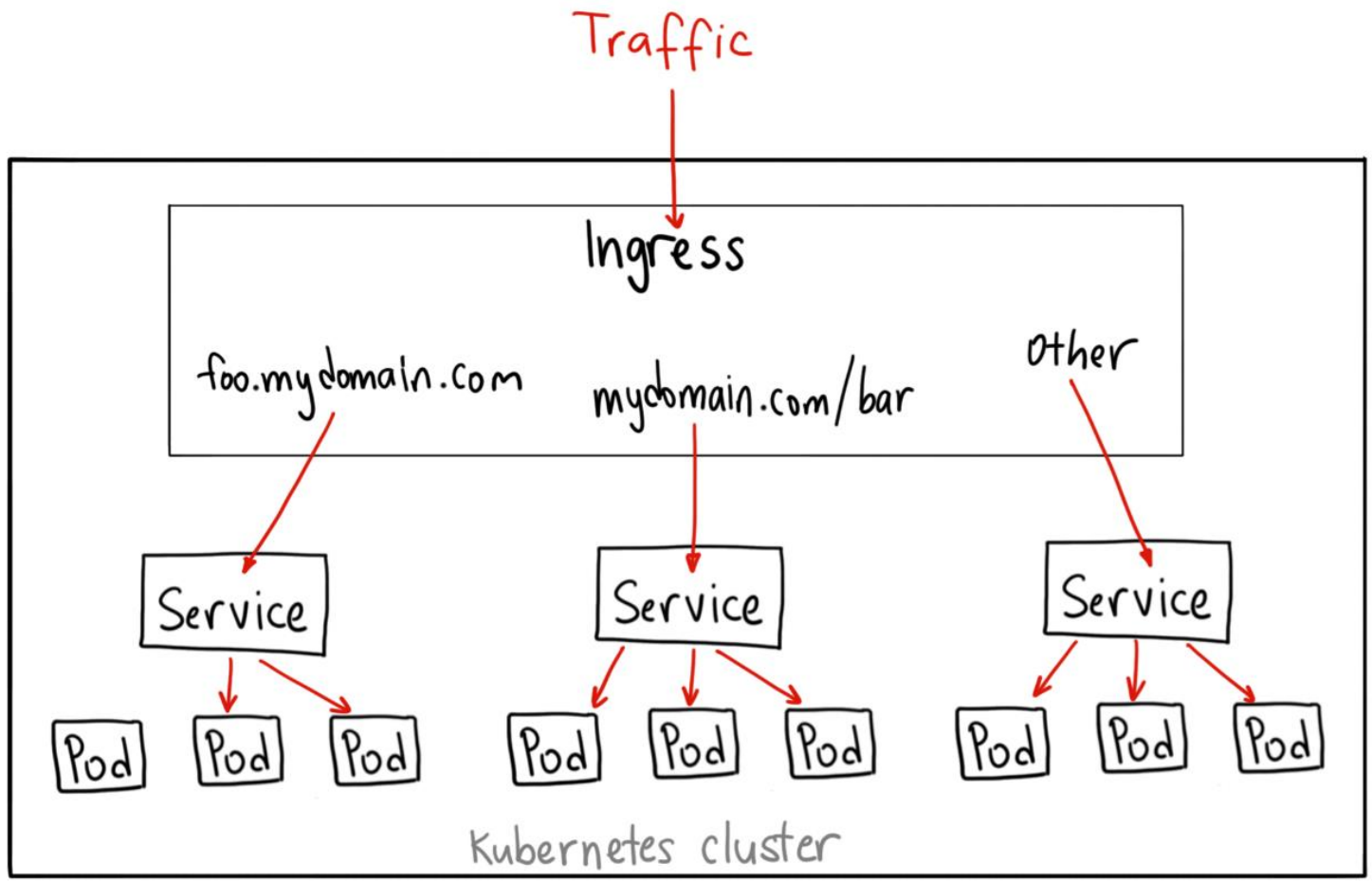
Simple fanout

```
rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /foo
        backend:
          serviceName: s1
          servicePort: 80
      - path: /bar
        backend:
          serviceName: s2
          servicePort: 80
```

Name based virtual hosting

```
rules:
  - host: foo.bar.com
    http:
      paths:
      - backend:
          serviceName: s1
          servicePort: 80
  - host: bar.foo.com
    http:
      paths:
      - backend:
          serviceName: s2
          servicePort: 80
```

DEPLOYMENTS EXPOSED - INGRESS



DEPLOYMENTS EXPOSED - INGRESS

1. Namespaces
2. Pods and Containers
3. Deployment
4. Services
5. Ingress
5. **Selector, Labels and Ports**
6. Network plugins

DEPLOYMENTS EXPOSED - SELECTOR AND LABELS

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: web
  name: web
spec:
  replicas: 3
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
      - image: nginx:latest
        name: nginx
        ports:
        - containerPort: 80
        volumeMounts:
          - name: config-nginx
            mountPath: /etc/nginx/conf.d
      volumes:
        - name: config-nginx
          configMap:
            name: nginx-configmap
```

```yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    app: web
  name: web
spec:
  type: ClusterIP
  selector:
    app: web
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
```

DEPLOYMENTS EXPOSED - SELECTOR AND LABELS

```yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    app: web
  name: web
spec:
  type: ClusterIP
  selector:
    app: web
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
```

```yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-web
  annotations:
    kubernetes.io/ingress.class: "nginx"
spec:
  rules:
  - http:
      paths:
      - path: /
        backend:
          serviceName: web
          servicePort: 80
```

DEPLOYMENTS EXPOSED - PORTS

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: web
  name: web
spec:
  replicas: 3
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
      - image: nginx:latest
        name: nginx
        ports:
        - containerPort: 80
        volumeMounts:
          - name: config-nginx
            mountPath: /etc/nginx/conf.d
      volumes:
        - name: config-nginx
          configMap:
            name: nginx-configmap
```

```yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    app: web
  name: web
spec:
  type: ClusterIP
  selector:
    app: web
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
```

DEPLOYMENTS EXPOSED - PORTS

```yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    app: web
  name: web
spec:
  type: ClusterIP
  selector:
    app: web
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
```

```yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ingress-web
  annotations:
    kubernetes.io/ingress.class: "nginx"
spec:
  rules:
  - http:
      paths:
      - path: /
        backend:
          serviceName: web
          servicePort: 80
```

DEPLOYMENTS EXPOSED - INGRESS

1. Namespaces
2. Pods and Containers
3. Deployment
4. Services
5. Ingress
6. Selector, Labels and Ports
5. **Network plugins**

DEPLOYMENTS EXPOSED – NETWORK PLUGINS

- pods on a node can communicate with all pods on all nodes without NAT
- agents on a node (e.g. system daemons, kubelet) can communicate with all pods on that node

DEPLOYMENTS EXPOSED - FLANNEL

- L2 Network
- Use etcd to store configuration
- VXLAN
- Doesn’t support network policies

DEPLOYMENTS EXPOSED – CALICO

- L3 with BGP
- Network policy
- Calico the Hard Way

https://docs.projectcalico.org/getting-started/kubernetes/hardwa

DEPLOYMENTS EXPOSED – ISTIO

- Traffic Management
- **Observability**
- **Security**
- Extensibility

KUBERNETES QUICK START

#1	Kubernetes Quick Start
#2	Deployments Exposed
#3	Running your App
#4	Application Scaling
#5	Updating an Application
#6	Configuration Files and Specs
#7	Dealing with Storage
#8	Application Configuration
#9	Jobs and Daemons
#10	Stateful Applications
#11	RBAC
#12	HELM

RUNNING YOUR APP – DRY RUNS

1. **Dry runs**
2. Using environment variables
3. Commands and arguments
4. Scheduling
5. Image pull policies
6. Application restarts
7. Application logs

RUNNING YOUR APP – DRY RUNS

DRY RUN

is an option for kubectl utility by using which you can only print the object that would be sent, without sending it.

Kubectl:

- kubectl create deployment nginx1 --image=nginx --dry-run=true
- kubectl create deployment nginx1 --image=nginx --dry-run=true –o json > nginx.yaml

RUNNING YOUR APP – DRY RUNS

1. Dry runs
2. **Using environment variables**
3. Commands and arguments
4. Scheduling
5. Image pull policies
6. Application restarts
7. Application logs

RUNNING YOUR APP – PODS AND CONTAINERS

ENVIRONMENT VARIABLES

When you create a Pod, you can set environment variables for the containers that run in the Pod. To set environment variables, include the **env** or **envFrom** field in the configuration file.

```
spec:
  containers:
  - name: nginx
    image: nginx:1.12
    ports:
    - containerPort: 80
    env:
    - name: DEMO_KUBERNETES
      value: "Hello kubernetes training"
```

RUNNING YOUR APP - PODS AND CONTAINERS

Expose Pod Information to Containers Through Environment Variables

Also you can use environment variables in order to expose information about itself to Containers running in the Pod. Environment variables can expose Pod fields and Container fields.

- metadata.name
- metadata.namespace
- metadata.labels
- metadata.annotations
- spec.nodeName
- spec.serviceAccountName
- status.hostIP
- status.podIP

RUNNING YOUR APP – COMMANDS AND ARGUMENTS

1. Dry runs
2. Using environment variables
3. **Commands and arguments**
4. Scheduling
5. Image pull policies
6. Application restarts
7. Application logs

RUNNING YOUR APP – COMMANDS AND ARGUMENTS

COMMANDS AND ARGUMENTS

- command field in the configuration file
- args field in the configuration file
- cannot be changed after the Pod is created
- override the default command and arguments provided by the container image
- If you define only args, the default image command is used with your new arguments.

```
FROM alpine
ENTRYPOINT ["ping"]
CMD ["www.google.com"]
```

shell:

- docker build -t test docker/
- docker run test
- docker run test epam.com

RUNNING YOUR APP - COMMANDS AND ARGUMENTS

Description	Docker field name	Kubernetes field name
The command run by the container	Entrypoint	command
The arguments passed to the command	Cmd	args

Image Entrypoint	Image Cmd	Container command	Container args	Command run
`[/ep-1]`	`[foo bar]`	<not set>	<not set>	`[ep-1 foo bar]`
`[/ep-1]`	`[foo bar]`	`[/ep-2]`	<not set>	`[ep-2]`
`[/ep-1]`	`[foo bar]`	<not set>	`[zoo boo]`	`[ep-1 zoo boo]`
`[/ep-1]`	`[foo bar]`	`[/ep-2]`	`[zoo boo]`	`[ep-2 zoo boo]`

RUNNING YOUR APP – IMAGE PULL POLICIES

1. Dry runs
2. Using environment variables
3. Commands and arguments
4. **Scheduling**
5. Image pull policies
6. Application restarts
7. Application logs

RUNNING YOUR APP – SCHEDULING

WAYS TO MANAGE SCHEDULING

- All use label selectors to make the selection
- Node Selector
- Node Affinity and anti-affinity
- Taints and Tolerations
- Inter-pod affinity and anti-affinity

RUNNING YOUR APP – SCHEDULING

NODE SELECTOR

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      nodeSelector:
        type: cdp
      containers:
      - name: nginx
        image: nginx:1.12
```

RUNNING YOUR APP – SCHEDULING

NODE AFFINITY

- Not just “AND of exact match”
- Soft/Preference rules
- In, NotIn, Exists, DoesNotExist, Gt, Lt

```
PodSpec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: kubernetes.io/e2e-az-name
            operator: In
            values:
            - e2e-az1
            - e2e-az2
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        preference:
          matchExpressions:
          - key: another-node-label-key
            operator: In
            values:
            - another-node-label-value
```

RUNNING YOUR APP – SCHEDULING

NODE AFFINITY

- Both nodeSelector and nodeAffinity must be satisfied
- Pod can be scheduled onto a node if one of the nodeSelectorTerms is satisfied
- Pod can be scheduled onto a node only if all matchExpressions can be satisfied
- The affinity selection works only at the time of scheduling the pod

RUNNING YOUR APP – SCHEDULING

INTER-POD AFFINITY

```
podAffinity:
requiredDuringSchedulingIgnoredDuring
Execution:
      - labelSelector:
          matchExpressions:
          - key: security
            operator: In
            values:
            - S1
        topologyKey: failure-
domain.beta.kubernetes.io/zone
```

```
podAntiAffinity:
preferredDuringSchedulingIgnoredDuring
Execution:
      - weight: 100
        podAffinityTerm:
          labelSelector:
            matchExpressions:
            - key: security
              operator: In
              values:
              - S2
          topologyKey:
kubernetes.io/hostname
```

RUNNING YOUR APP – SCHEDULING

TAINTS AND TOLERATIONS

- The taint has key key, value value, and taint effect
- Effect can be NoSchedule, PreferNoSchedule, NoExecute
- Toleration has key, operator, value, effect
- Operator can be Exists and Equal

RUNNING YOUR APP – IMAGE PULL POLICIES

1. Dry runs
2. Using environment variables
3. Commands and arguments
4. Scheduling
5. **Image pull policies**
6. Application restarts
7. Application logs

RUNNING YOUR APP – IMAGE PULL POLICIES

PULL POLICIES

- IfNotPresent(DEFAULT)
- Always(:LATEST)
- Never

Note:
You should avoid using :latest tag when deploying containers in production, because this makes it hard to track which version of the image is running and hard to roll back.

RUNNING YOUR APP – APPLICATION RESTARTS

1. Dry runs
2. Using environment variables
3. Commands and arguments
4. Scheduling
5. Image pull policies
6. **Application restarts**
7. Application logs

RUNNING YOUR APP – APPLICATION RESTARTS

APPLICATION RESTARTS

- Always
- OnFailure
- Never
- The default value is Always
- Applies to all Containers in the Pod
- Pods are restarted with an exponential back-off delay (10s, 20s, 40s …)

RUNNING YOUR APP – APPLICATION LOGS

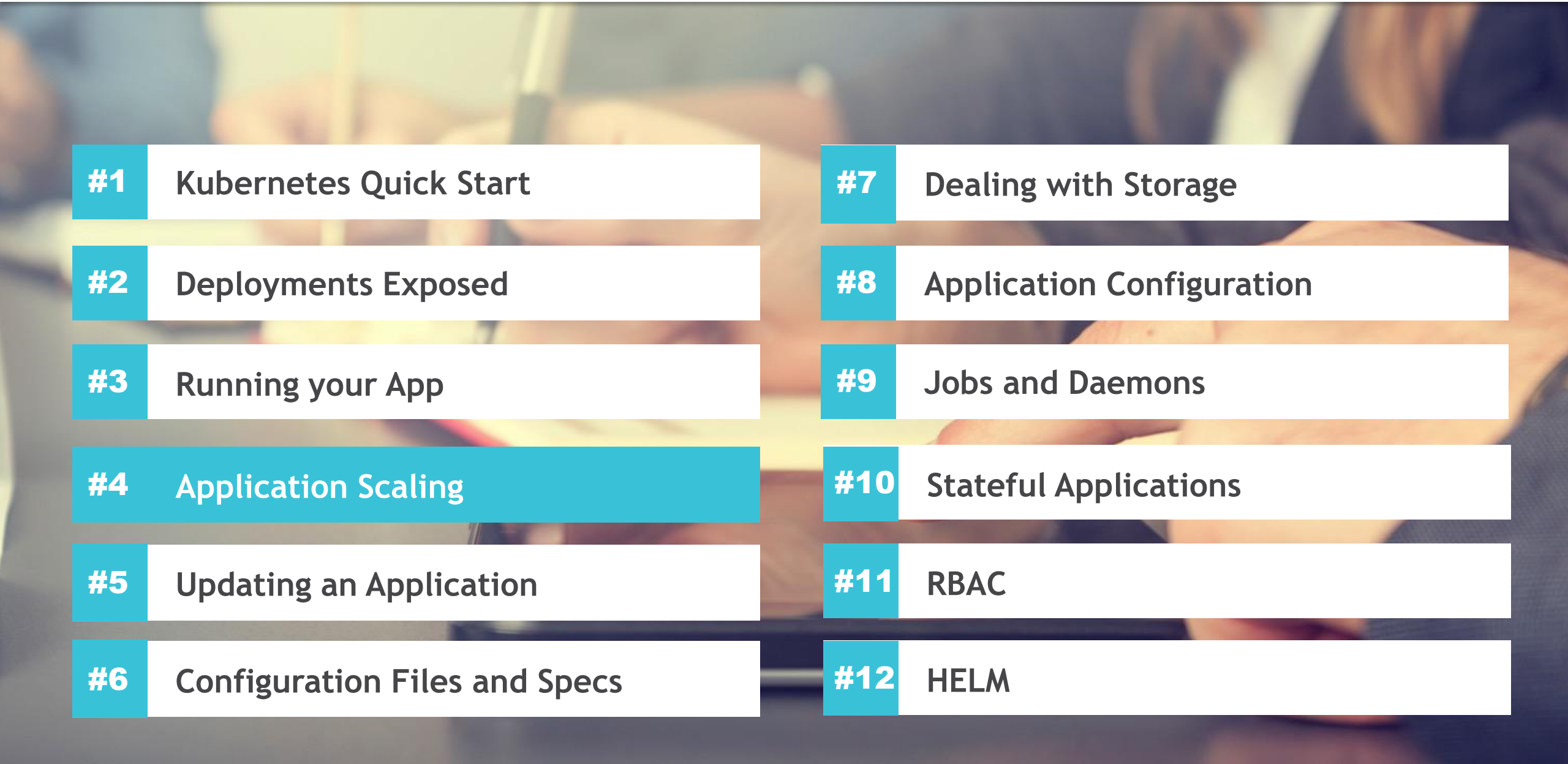
1. Dry runs
2. Using environment variables
3. Commands and arguments
4. Scheduling
5. Image pull policies
6. Application restarts
7. **Application logs**

RUNNING YOUR APP – APPLICATION LOGS

APPLICATION LOGS

- Kubectl logs command retrieves logs from pods
- If the container has crashed use with --previous flag
- If your pod has multiple containers, you should specify container name
- If a container restarts, kubernetes keeps one terminated container with its logs
- If a pod is evicted from the node, all corresponding containers are also evicted, along with their logs
- UI Dashboard logs to retrieve logs from pods

KUBERNETES QUICK START



#1	Kubernetes Quick Start
#2	Deployments Exposed
#3	Running your App
#4	Application Scaling
#5	Updating an Application
#6	Configuration Files and Specs
#7	Dealing with Storage
#8	Application Configuration
#9	Jobs and Daemons
#10	Stateful Applications
#11	RBAC
#12	HELM

APPLICATION SCALING – REPLICA SETS

1. **Replica Sets**
2. Scaling
3. Autoscaling
4. Compute Resources Managing

APPLICATION SCALING – REPLICA SETS

REPLICA SETS

- Can be used independently.
- Using with Deployments do not worry about managing the ReplicaSets
- A ReplicaSet ensures that a specified number of pod replicas are running at any given time.

APPLICATION SCALING – SCALING

1. Replica Sets
2. **Scaling**
3. Autoscaling
4. Compute Resources Managing

APPLICATION SCALING - SCALING

SCALING

- One pod by default
- Scaling to zero
- Proportional scaling

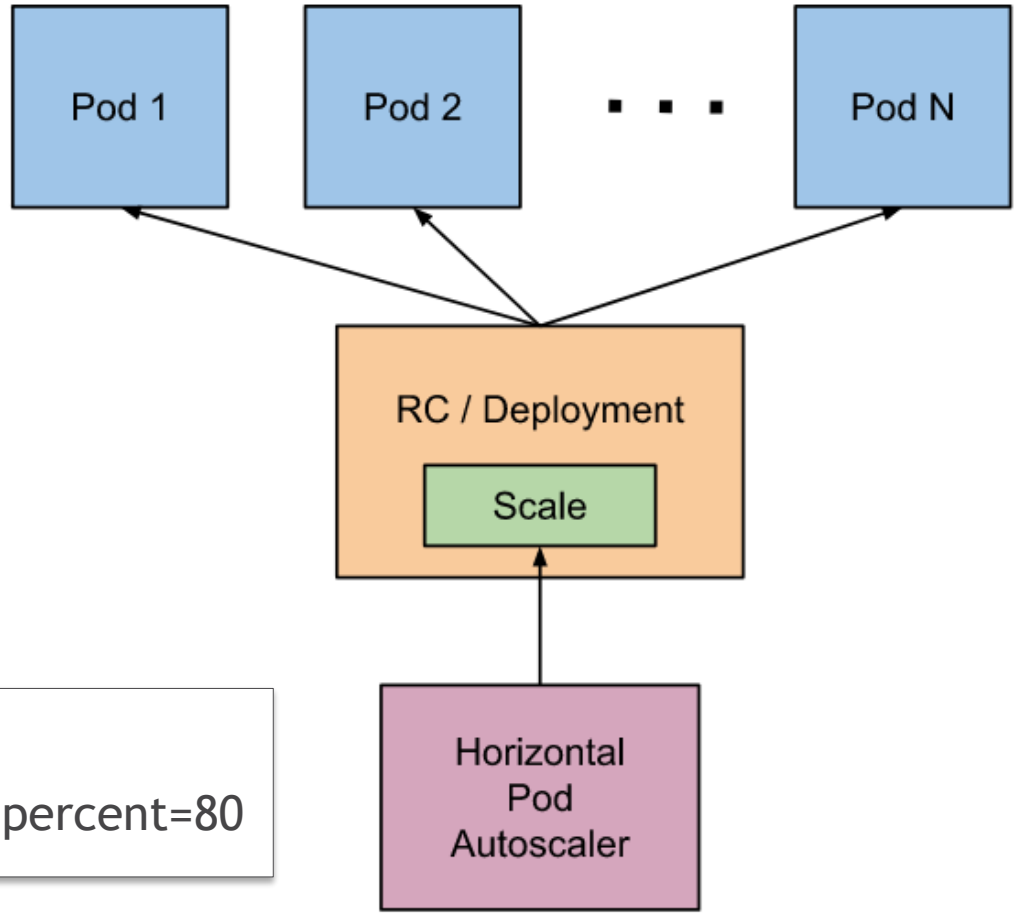
APPLICATION SCALING – AUTOSCALING

1. Replica Sets
2. Scaling
3. **Autoscaling**
4. Compute Resources Managing

APPLICATION SCALING - AUTOSCALING

AUTOSCALING

- is implemented as a Kubernetes API resource and a controller.
- default value of 30 seconds
- obtains the metrics from either the resource metrics API or the custom metrics API
- resource request set require



Kubectl:

- kubectl autoscale deployment nginx --min=2 --max=5 --cpu-percent=80

APPLICATION SCALING – COMPUTE RESOURCES

1. Replica Sets
2. Scaling
3. Autoscaling
4. **Compute Resources Managing**

APPLICATION SCALING - COMPUTE RESOURCES

RESOURCES

```
containers:
  - name: db
    image: mysql
    env:
    - name: MYSQL_ROOT_PASSWORD
      value: "password"
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

QA