Kubernetes
the Deltatre way

8 JUN, 5:30 PM

ogr officine grandi riparazioni

newesis e Professional | Have Fun!

tag Talent Garden

Kubernetes advanced topics & Kind

Deltatre Innovation Lab

CARLO ALBERTO SCAGLIA (DELTATRE)
SUPPORTED BY
MASSIMILIANO GIOVAGNOLI (KLOUDOPS)

Carlo Alberto Scaglia

Linkedin:
https://www.linkedin.com/in/carloalbertoscaglia/

GitHub:
https://github.com/CarloAlbertoS

Father of one, passionate about music, basket and Lego I started to work for deltatre in 2006 as a Video Operator and fell in love with IT world.

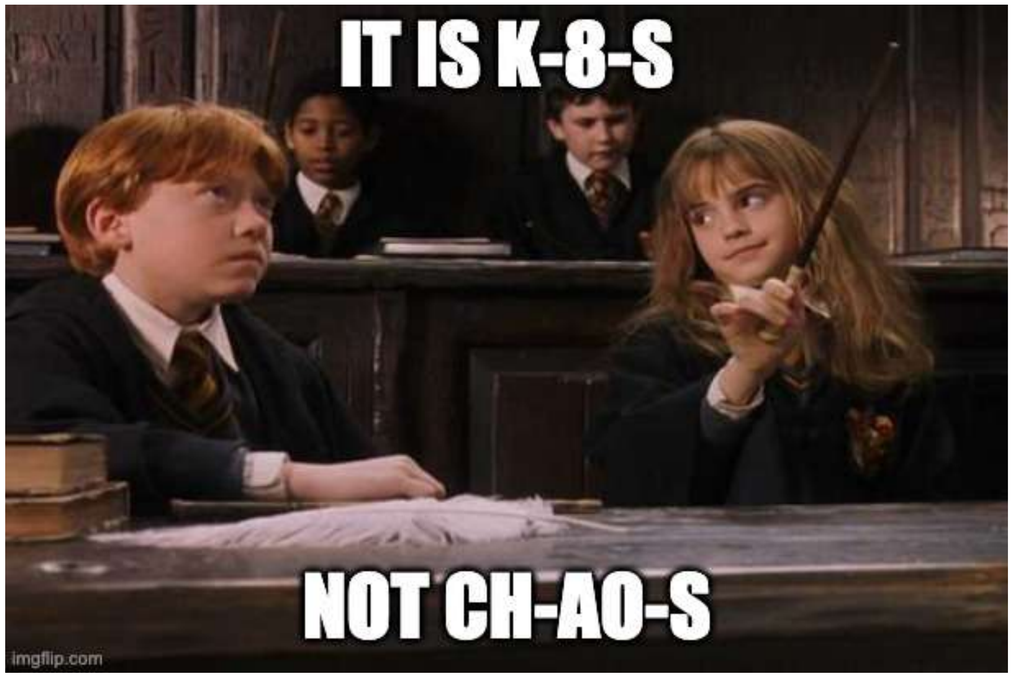
Fourteen years later I still see myself as a Rookie and try to push myself as far as I can with latest technologies on the field.

Introduction

deltatre

Fasten your seat belts, it’s time to k8s!

- ❑ What we saw in the previous session
- ❑ Secure your cluster: AGGIUNGI TU QUALCOSA
- ❑ Our Apps are up and running: cool, but who’s driving?
- ❑ »I love k8s, but you know it doesn’t guarantee persistance» and other nice fairy tales
- ❑ It’s a KIND of Magic!



Admission Controller

«You shall not pass!!» (Gandalf teaching admission controllers to Balrog)

What are they?

Admission Controllers are a set of pre shipped plugins which intercept authenticated request to the APIs and perform controls on them before (eventually) passing them the API Server.

Admission Controller are essential for many of the advanced configurations in k8s (for example quota management, Service Account creation and PVC resize)

How to use them and what do they do?

In the latest versions of k8s many of those are already enabled and active once the cluster is initiated, so no additional action is required, while the remaining one can be enabled in the api-server configuration.

They act based on two different roles (which can be combined): modify and validate.

Plugin List and Specifications

Default

NamespaceLifecycle, LimitRanger, ServiceAccount, TaintNodesByCondition, Priority, DefaultTolerationSeconds, DefaultStorageClass, StorageObjectInUseProtection, PersistentVolumeClaimResize, RuntimeClass, CertificateApproval, CertificateSigning, CertificateSubjectRestriction, DefaultIngressClass, ResourceQuota

Available

AlwaysAdmit, AlwaysDeny, AlwaysPullImages, DenyEscalatingExec, DenyExecOnPrivileged, EventRateLimit, ExtendedResourceToleration, ImagePolicyWebhook, LimitPodHardAntiAffinityTopology, NamespaceAutoProvision, NamespaceExists, NodeRestriction, OwnerReferencesPermissionEnforcement, PersistentVolumeLabel, PodNodeSelector, PodPreset, PodSecurityPolicy, PodTolerationRestriction, SecurityContextDeny

Webhooks

MutatingAdmissionWebhook, ValidatingAdmissionWebhook,

Enabled by default, they give additional flexibility by adding two customizable Webhook (one per each role) which can accept external REST APIs as input to decouple custom logics.

Advantages

- ❑ Security
- ❑ Remote Control
- ❑ Ease of use
- ❑ Configuration Management

Pod Security Policy

With this AC you have the full control on what can run inside the pod and effectively limit containers capabilities

- The policy can be defined as a YAML, so it can be replicated in all deployments easily
- It requires a Role (or a ClusterRole) and a binding to be used
- No, it is not related to a portable console

Control Aspect	Field Names
Running of privileged containers	`privileged`
Usage of host namespaces	`hostPID`, `hostIPC`
Usage of host networking and ports	`hostNetwork`, `hostPorts`
Usage of volume types	`volumes`
Usage of the host filesystem	`allowedHostPaths`
White list of FlexVolume drivers	`allowedFlexVolumes`
Allocating an FSGroup that owns the pod's volumes	`fsGroup`
Requiring the use of a read only root file system	`readOnlyRootFilesystem`
The user and group IDs of the container	`runAsUser`, `runAsGroup`, `supplementalGroups`
Restricting escalation to root privileges	`allowPrivilegeEscalation`, `defaultAllowPrivilegeEscalation`
Linux capabilities	`defaultAddCapabilities`, `requiredDropCapabilities`, `allowedCapabilities`
The SELinux context of the container	`seLinux`
The Allowed Proc Mount types for the container	`allowedProcMountTypes`
The AppArmor profile used by containers	annotations
The seccomp profile used by containers	annotations
The sysctl profile used by containers	`forbiddenSysctls`, `allowedUnsafeSysctls`

Pod Security Policy - Example

PSP YAML

```yaml
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: my-psp
spec:
  privileged: false  # Prevents creation of privileged Pods
  seLinux:
    rule: RunAsAny
  supplementalGroups:
    rule: RunAsAny
  runAsUser:
    rule: RunAsAny
  fsGroup:
    rule: RunAsAny
  volumes:
  - '*'
```

CLUSTERROLE AND BINDING

```yaml
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: my-clusterrole
rules:
- apiGroups:
  - policy
  resources:
  - podsecuritypolicies
  verbs:
  - use
  resourceNames:
  - my-psp
```

```yaml
# Bind the ClusterRole to the desired set of service accounts.
# Policies should typically be bound to service accounts in a namespace.
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: my-rolebinding
  namespace: my-namespace
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: my-clusterrole
subjects:
# Example: All service accounts in my-namespace
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:serviceaccounts
# Example: A specific service account in my-namespace
- kind: ServiceAccount # Omit apiGroup
  name: default
  namespace: my-namespace
```

Limit Range

It defines the amount of resources which can be assigned to pods and containers in terms of CPU, RAM and Volumes

- The policy can define both the constraint values (min/max) and the default ones if not specified inside the deployments
- It can also define a min/max storage requests per PVC
- It can enumerates namespaces constraints (it works in combination with resourceQuota AC)

```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: mem-limit-range
spec:
  limits:
  - default:
      memory: 512Mi
    defaultRequest:
      memory: 256Mi
    type: Container
```

```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: cpu-min-max-demo-lr
spec:
  limits:
  - max:
      cpu: "800m"
    min:
      cpu: "200m"
    type: Container
```

Resource Quota

It defines the amount of resources which can be assigned to namespaces in terms of CPU, RAM, Pods and Volumes

- The policy can define both the constraint values (min/max) and the default ones if not specified inside the deployments
- It can also define a min/max storage requests per PVC
- It defines the limit of the sum of the resources requested

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: mem-cpu-demo
spec:
  hard:
    requests.cpu: "1"
    requests.memory: 1Gi
    limits.cpu: "2"
    limits.memory: 2Gi
```

Storage Classes and Volume Claims

Storage Classes

With a Storage class you can map your existing storage (both physical and cloud based) inside the cluster, defining for example different quality of service levels and pass everything as a resource to your deployments.

- Each Storage Class must contain a provisioner and a value for reclaimPolicy (default is «delete»)
- Additional Parameters are specific for provisioner

Volume Plugin	Internal Provisioner	Config Example
AWSElasticBlockStore	✓	AWS EBS
AzureFile	✓	Azure File
AzureDisk	✓	Azure Disk
CephFS	-	-
Cinder	✓	OpenStack Cinder
FC	-	-
FlexVolume	-	-
Flocker	✓	-
GCEPersistentDisk	✓	GCE PD
Glusterfs	✓	Glusterfs
iSCSI	-	-
Quobyte	✓	Quobyte
NFS	-	-
RBD	✓	Ceph RBD
VsphereVolume	✓	vSphere
PortworxVolume	✓	Portworx Volume
ScaleIO	✓	ScaleIO
StorageOS	✓	StorageOS
Local	-	Local

Storage Classes Examples

GCP (Huey)

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: slow
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard
  fstype: ext4
  replication-type: none
```

Azure (Dewey)

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: slow
provisioner: kubernetes.io/azure-disk
parameters:
  skuName: Standard_LRS
  location: eastus
  storageAccount: azure_storage_account_name
```

AWS (Louie)

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: slow
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1
  iopsPerGB: "10"
  fsType: ext4
```

Persistent Volumes And Claims

- The PersistentVolume subsystem provides an API for users and administrators that abstracts details of how storage is provided from how it is consumed.
- A *PersistentVolume* (PV) is a storage element in the cluster that has been provisioned by an administrator (static) or provisioned using Storage Classes (dynamic). PVs are volume plugins like Volumes, but with an independent lifecycle from the individual Pods that uses them.
- A *PersistentVolumeClaim* (PVC) is a request for storage performed by the user. It is a specific resource inside the cluster and relies on PVs to allocate the space needed.
- **NOTE**: on managed system like AKS the number of available Volumes per node varies with the selected size of the virtual machine, so pay attention to it when sizing your environment!

Persistent Volumes

- Name: the name of the volume (NOTE: must be a valid DNS subdomain)
- Capacity: the size of the Volume
- volumeMode: Filesystem (default), block
- Reclaim Policy: retain, recycle (wiping), delete
- StorageClassName: the StorageClass used
- MountOption: specific options (NOTE: no validation, misconfiguration result in failures)
- accessModes: press the screen to know more about

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0003
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  storageClassName: slow
  mountOptions:
    - hard
    - nfsvers=4.1
  nfs:
    path: /tmp
    server: 172.17.0.2
```

Access Modes

Access Mode defines how the Volume can be used by Pods. It is very important to note that ONLY one mode can be used per time by all pods connected to the volume, even if it supports more than one.

- ReadWriteOnce (RWO) – the volume can be mounted as read-write by a single node
- ReadOnlyMany (ROX) – the volume can be mounted read-only by many nodes
- ReadWriteMany (RWX) – the volume can be mounted as read-write by many nodes

Volume Plugin	ReadWriteOnce	ReadOnlyMany	ReadWriteMany
AWSElasticBlockStore	✓	-	-
AzureFile	✓	✓	✓
AzureDisk	✓	-	-
CephFS	✓	✓	✓
Cinder	✓	-	-
CSI	depends on the driver	depends on the driver	depends on the driver
FC	✓	✓	-
FlexVolume	✓	✓	depends on the driver
Flocker	✓	-	-
GCEPersistentDisk	✓	✓	-
Glusterfs	✓	✓	✓
HostPath	✓	-	-
iSCSI	✓	✓	-
Quobyte	✓	✓	✓
NFS	✓	✓	✓
RBD	✓	✓	-
VsphereVolume	✓	-	- (works when Pods are collocated)
PortworxVolume	✓	-	✓
ScaleIO	✓	✓	-
StorageOS	✓	-	-

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myclaim
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 8Gi
  storageClassName: slow
  selector:
    matchLabels:
      release: "stable"
    matchExpressions:
      - {key: environment, operator: In, values: [dev]}
```

Persistent Volume Claims

The Persistent Volume Claims is what the Pod uses to claim a portion of a Persistent Volume

- The name needs to be a valid DNS subdomain
- storageClassName: a PVC can reclaim only PVs matching the Storage Class specified
- Selectors are label used to fine match PVs available

Pod Claim

The PVC is the only information which is passed directly to the Pod

- PVC has to exist in the same namespace of the Pod
- Since PVCs are namespaces resources, so ROX and RWX access modes are possible ONLY within the same namespace

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
    - name: myfrontend
      image: nginx
      volumeMounts:
      - mountPath: "/var/www/html"
        name: mypd
  volumes:
    - name: mypd
      persistentVolumeClaim:
        claimName: myclaim
```

Resources management

Segregation, Quota and Limits

- Used to create logical segregation of applications
- Can be used to define multiple environments
- Can be used to manage multi-tenant clusters
- Permissions (roles and roles binding) are namespaced
- RBAC can be used to allow users access only on certain namespaces

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: my-nice-namespace
  labels:
    app: my-nice-app
```

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: my-nice-quota
  namespace: my-nice-namespace
spec:
  hard:
    requests.cpu: "1"
    requests.memory: 2Gi
    limits.cpu: "4"
    limits.memory: 4Gi
    pods: 4
```

```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: my-nice-range
  namespace: my-nice-namespace
spec:
  limits:
  - default:
      cpu: 1
      memory: 1024Mi
    defaultRequest:
      cpu: 0.25
      memory: 256Mi
    type: Container
```

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - ipBlock:
        cidr: 172.17.0.0/16
        except:
        - 172.17.1.0/24
    - namespaceSelector:
        matchLabels:
          project: myproject
    - podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 6379
  egress:
  - to:
    - ipBlock:
        cidr: 10.0.0.0/24
    ports:
    - protocol: TCP
      port: 5978
```

```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-allow-my-nice-app
  namespace: my-nice-namespace
spec:
  podSelector:
    matchLabels:
      app: database
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          app: my-nice-app
```

Segregation, Quota and Limits

- Namespace have no limits, no quota and not filters by default
- Policies can be applied to a Namespace to
  - Define default resource requests and limits
  - Define resource quotas
  - Define network access rules

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-ingress
spec:
  podSelector: {}
  policyTypes:
  - Ingress
```

```
...
ingress:
- from:
  - namespaceSelector:
      matchLabels:
        user: alice
  - podSelector:
      matchLabels:
        role: client
...
```

```
...
ingress:
- from:
  - namespaceSelector:
      matchLabels:
        user: alice
  podSelector:
    matchLabels:
      role: client
...
```

Probes

Liveness Probe

Indicates whether the Container is running. If the liveness probe fails, the kubelet kills the Container, and the Container is subjected to its restart policy. If a Container does not provide a liveness probe, the default state is Success.

Readiness Probe

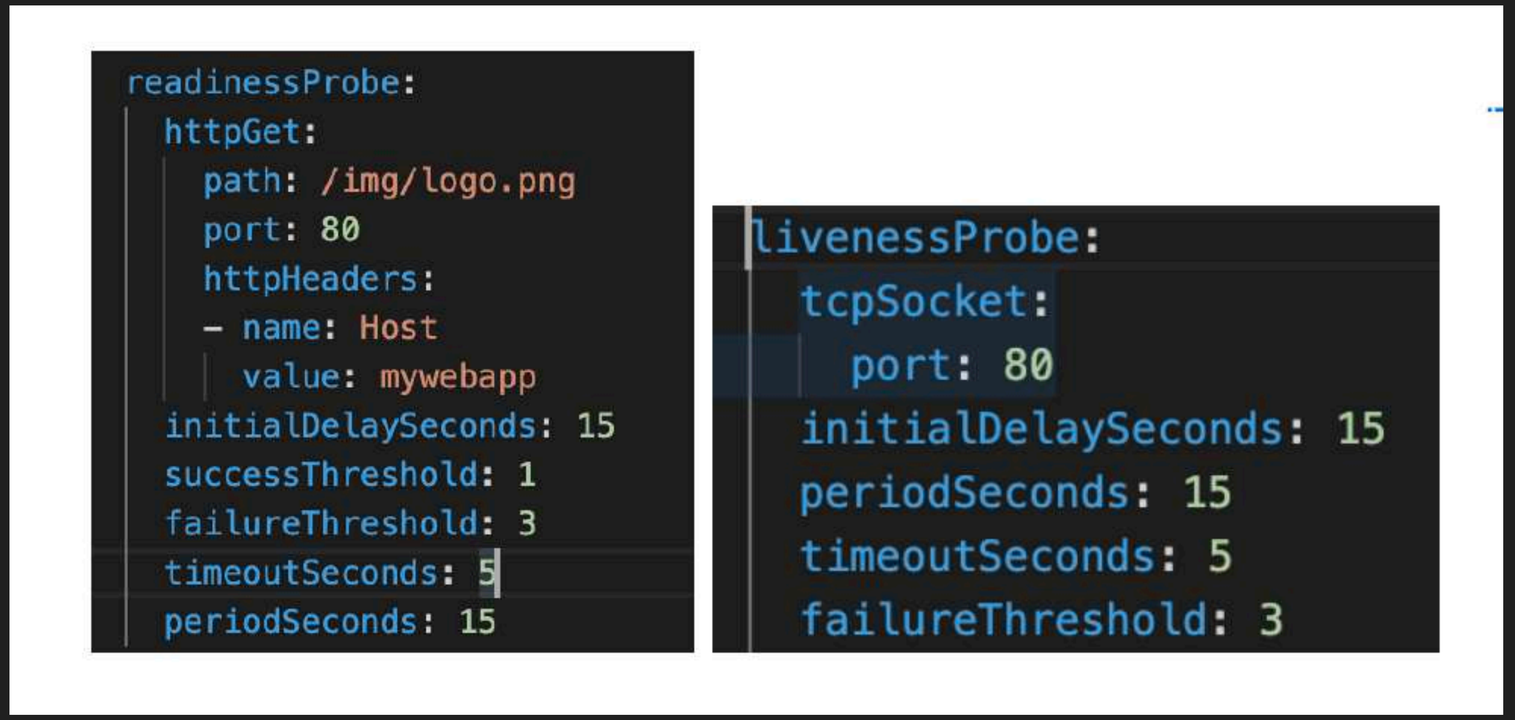
Indicates whether the Container is ready to serve requests. If the readiness probe fails, the endpoints controller removes the Pod's IP address from the endpoints of all Services that match the Pod. The default state of readiness before the initial delay is Failure. If a Container does not provide a readiness probe, the default state is Success.

Startup Probe

Indicates whether the application within the Container is started. All other probes are disabled if a startup probe is provided, until it succeeds. If the startup probe fails, the kubelet kills the Container, and the Container is subjected to its restart policy. If a Container does not provide a startup probe, the default state is Success.

Probes Actions

- ExecAction: Executes a specified command inside the Container. The diagnostic is considered successful if the command exits with a status code of 0.
- TCPSocketAction: Performs a TCP check against the Container's IP address on a specified port. The diagnostic is considered successful if the port is open.
- HTTPGetAction: Performs an HTTP Get request against the Container's IP address on a specified port and path. The diagnostic is considered successful if the response has a status code greater than or equal to 200 and less than 400.



«Now I know I've got to Run away I've got to Get away» (Tainted Love by Depeche Mode)

What is Taint?

A taint is a metadata assigned to a Node. It consists of a key, a value for the key, and an effect. The key and value can be anything and it works in opposition to NodeAffinity pods property.

How do it effectively acts?

There are three behaviour in scheduling pods on tainted nodes:

PreferNoSchedule: taint effect to configure the scheduler to prefer not to schedule intolerant pods to the tainted node, but it will schedule it if no better option available

NoSchedule: taint effect to configure the scheduler not to schedule intolerant pods to the tainted node, but a running pod will not be modified

NoExecute: taint effect to configure the scheduler not to schedule intolerant pods to the tainted node and to evict running pods

Tolerations

- a toleration is a metadata assigned to a POD. It consists of the key name, an operator, a value and, optional, an effect. If effect is provided it must be PreferredNoSchedule, NoSchedule or NoExecute and should match the effect on the taint. If effect is omitted then the toleration will match any taint with any effect as long as the key and value match.
- Equal: value of the operator of the toleration to mark the toleration active if the value match the key value on the taint
- Exists: value of the operator of the toleration to mark the toleration active if the key exist, independently from the value

Node Selector and Affinity

- NodeSelector: Node selectors are a field of PodSpec that signals the scheduler to prefer or require nodes with certain labels or conditions when scheduling a pod

- Affinity: is a property of PodSpec, and there are three types of affinity: nodeAffinity, podAffinity and podAntiAffinity, it provides a mechanism for either attracting it to or repelling it from other pods and nodes. It is possible to set rule as “soft”/“preference” rather than a hard requirement, so if the scheduler can’t satisfy it, the pod will still be scheduled

- NodeAffinity: Node affinity is conceptually similar to nodeSelector, it allows you to constrain which nodes your pod is eligible to be scheduled on, based on labels on the node. Node affinity syntax supports the following operators: In, NotIn, Exists, DoesNotExist, Gt, Lt. You can use NotIn and DoesNotExist together

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: with-node-affinity
spec:
  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
  containers:
  - name: with-node-affinity
    image: k8s.gcr.io/pause:2.0
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: with-pod-affinity
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: security
            operator: In
            values:
            - S1
        topologyKey: failure-domain.beta.kubernetes.io/zone
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 100
        podAffinityTerm:
          labelSelector:
            matchExpressions:
            - key: security
              operator: In
              values:
              - S2
          topologyKey: failure-domain.beta.kubernetes.io/zone
  containers:
  - name: with-pod-affinity
    image: k8s.gcr.io/pause:2.0
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  nodeSelector:
    disktype: ssd
```

Node Selector and Affinity

- PodAffinity and PodAntiAffinity: Inter-pod affinity and anti-affinity allow you to constrain which nodes your pod is eligible to be scheduled based on labels on pods that are already running on the node rather than based on labels on nodes. The legal operators for pod affinity and anti-affinity are In, NotIn, Exists, DoesNotExist

- requiredDuringSchedulingIgnoredDuringExecution: the POD will not not be scheduled if affinity rule not satisfied, running POD are not touched.

- preferredDuringSchedulingIgnoredDuringExecution: the POD will not not be scheduled if affinity rule not satisfied unless there is no better alternative, running POD are not touched.

Node Selector and Affinity

> **Note:** Inter-pod affinity and anti-affinity require substantial amount of processing which can slow down scheduling in large clusters significantly. We do not recommend using them in clusters larger than several hundred nodes.

> **Note:** Pod anti-affinity requires nodes to be consistently labelled, in other words every node in the cluster must have an appropriate label matching `topologyKey`. If some or all nodes are missing the specified `topologyKey` label, it can lead to unintended behavior.

Don’t tell us we didn’t warn you ☺

Pimp My Pods

Static Pods

Static Pods are common pods from a deployment perspective, but they are managed directly from the kubelet deamon on a specific node rather than API server.

They are linked to specific nodes and to create them a restart of the kubelet deamon is necessary

Static Pods are defined directly in the Kubelet configuration file and can be managed in two different ways:

- FileSystemHosted: with this configuration the manifest YAML for the static pod exist directly on the filesystem of the desired node and by populate the **–pod-manifest-path** inside the kubelet config file, the deamon will periodically scan the folder to create the resource

- Web-hosted: differently from the previous method, in this case the manifest location for the deamon to refer to is a remote webserver. The check mechanism remain the same, but in this case the correct key inside the kubelet config is **–pod-manifest-path**

To let the API server capable of retrieve the status of the Static Pod, the kubelet always create a mirror pod which can be seen (if PSP are set in the correct way).

You will be then able to see Static Pod with the normal set of commands, but if you try to operate on it you will see your commands won't have any effect on it since the API server does not control it.

StatefulSets

StatefulSets are a special kind of deployments which maintain the same approach to ensure identical container specs, but add the possibility to assign a sticky identity to Pods. The identification of each pod will be stable during future rescheduling.

Stateful Sets works in pair with volume persistance.

As per the documentation, StetefulSets provide:

- stable, unique network identifiers
- Stable, persistent storage
- Ordered, graceful deployment and scaling
- Ordered, automated rolling updates

There are some limitation as well, which we report here:

- The storage for a given Pod must either be provisioned by a PersistentVolume Provider based on the requested storage class, or pre-provisioned by an admin.
- Deleting and/or scaling a StatefulSet down will *not* delete the volumes associated with the StatefulSet. This is done to ensure data safety, which is generally more valuable than an automatic purge of all related StatefulSet resources
- StatefulSets do not provide any guarantees on the termination of pods when a StatefulSet is deleted. To achieve ordered and graceful termination of the pods in the StatefulSet, it is possible to scale the StatefulSet down to 0 prior to deletion.

Headless Service

Headless service are a specific kind of services which doesn't use load-balancing and single service IP for the Pods it manages.

How the DNS routing to Pods is managed depends on how selectors are defined

To define a service as headless it is enough to specify «none» in clusterIP specification.

The two possibilities to define the DNS selectors: are the following

- With Selectors: the endpoints controller creates Endpoints records in the API, and modifies the DNS configuration to return records (addresses) that point directly to the pods
- Without Selectors: the endpoints controller does not create Endpoints records, but it looks for and configure DNS CNAME or A records related to the service itself.

Examples

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  ports:
  - port: 80
    name: web
  clusterIP: None
  selector:
    app: nginx
---
```

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  selector:
    matchLabels:
      app: nginx # has to match .spec.template.metadata.labels
  serviceName: "nginx"
  replicas: 3 # by default is 1
  template:
    metadata:
      labels:
        app: nginx # has to match .spec.selector.matchLabels
    spec:
      terminationGracePeriodSeconds: 10
      containers:
      - name: nginx
        image: k8s.gcr.io/nginx-slim:0.8
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: www
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
  - metadata:
      name: www
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: "my-storage-class"
      resources:
        requests:
          storage: 1Gi
```

Multi Containers Pod designing patterns

They are both Pods: Can you guess the difference?

Single Malt

Blended

Sidecar

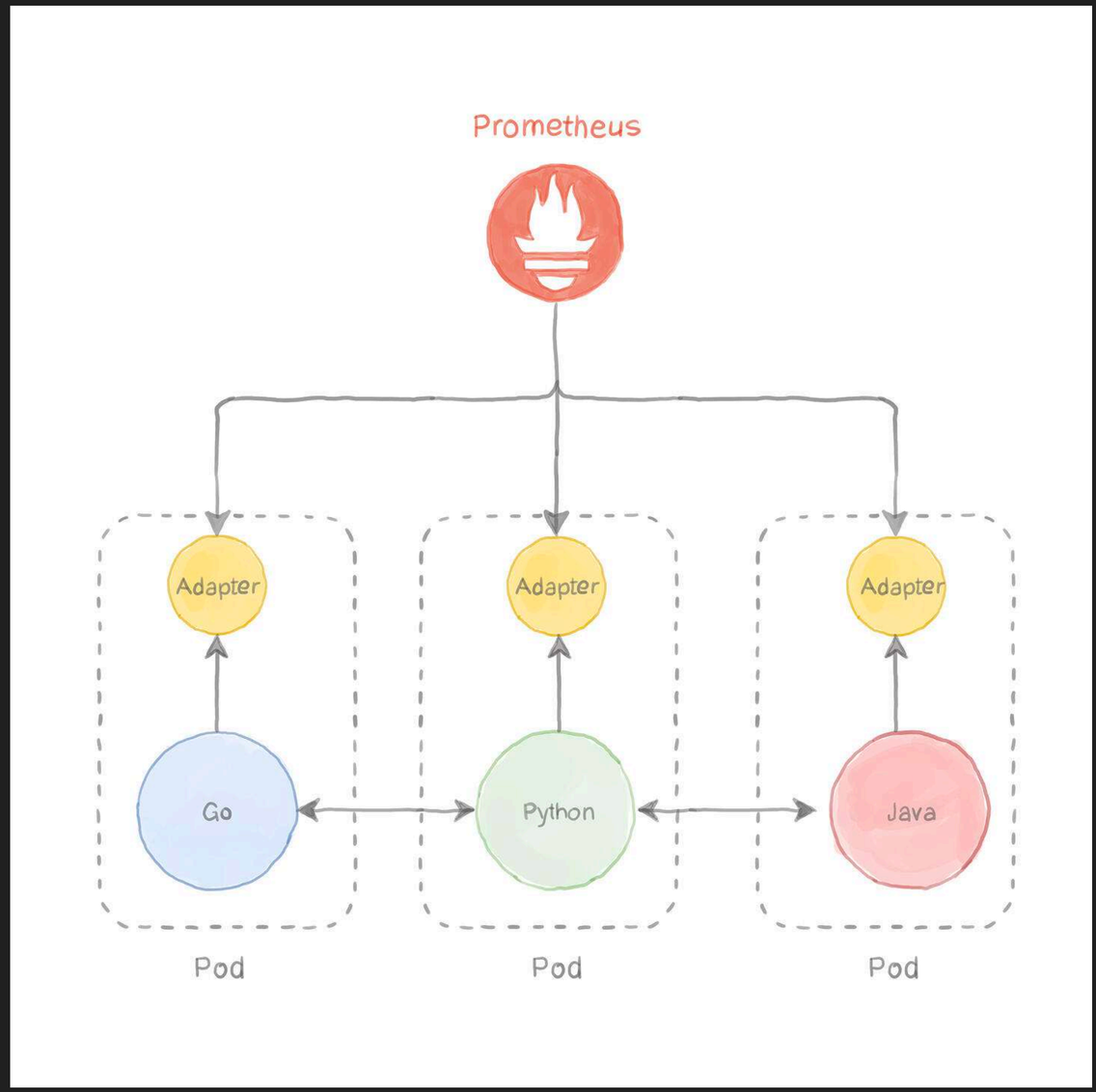
When we speak of “sidecar” pattern we refer to a setup where multiple containers shares the same set of resources (network and storage) and the additional containers run in combination of the of the main one to enrich its features or perform activities based on the its output.

Some examples of sidecar containers are APM/log shipping agents, CI/CD operators, database replication managers.

Adapter

The concept of adapter is very similar to the the sidecar one, but it is specifical to containers which have the purpose of translate the output of your application(s) in a specific different format.

A typical example for this is the Prometheus adapter

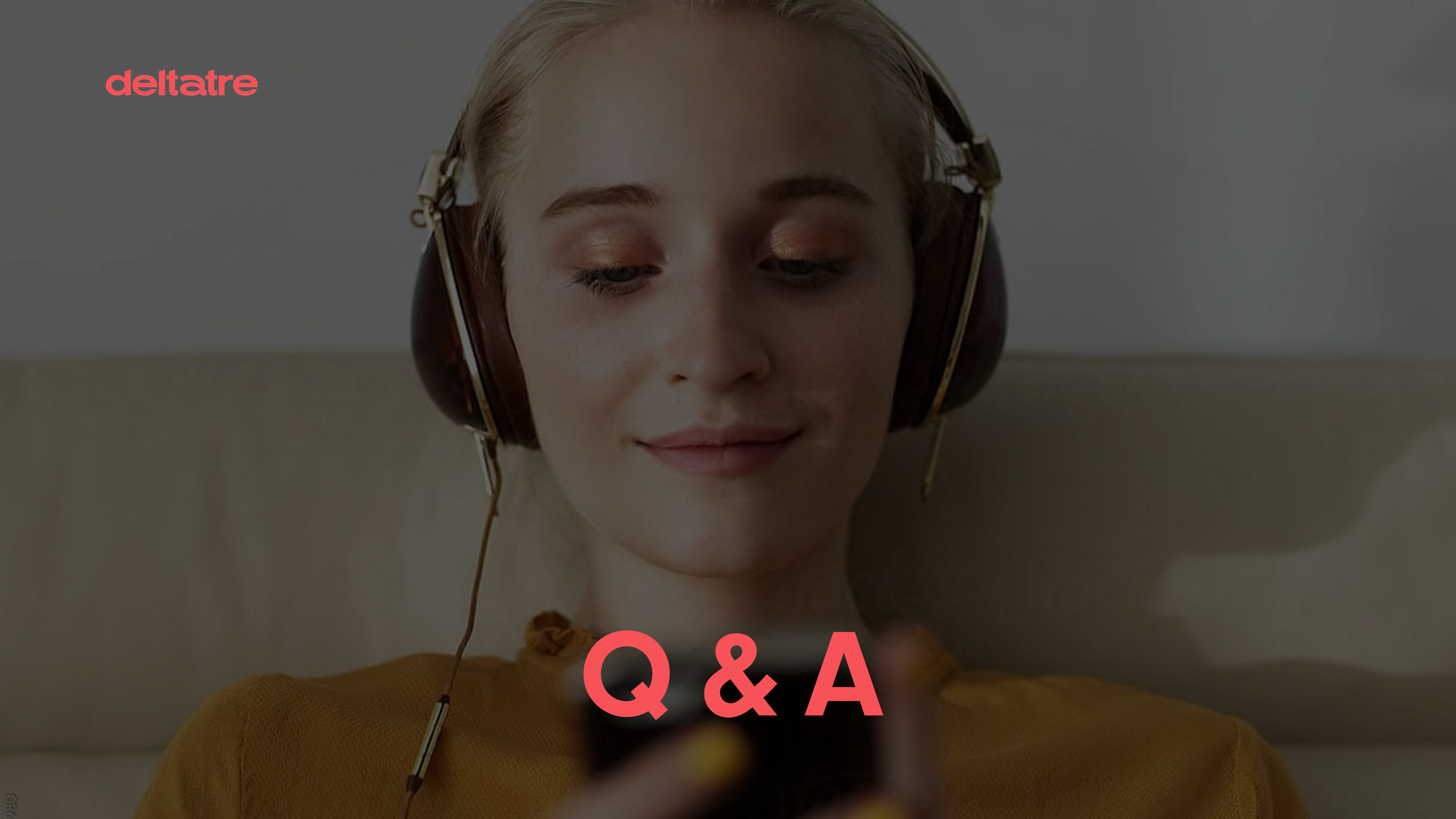


Adapter

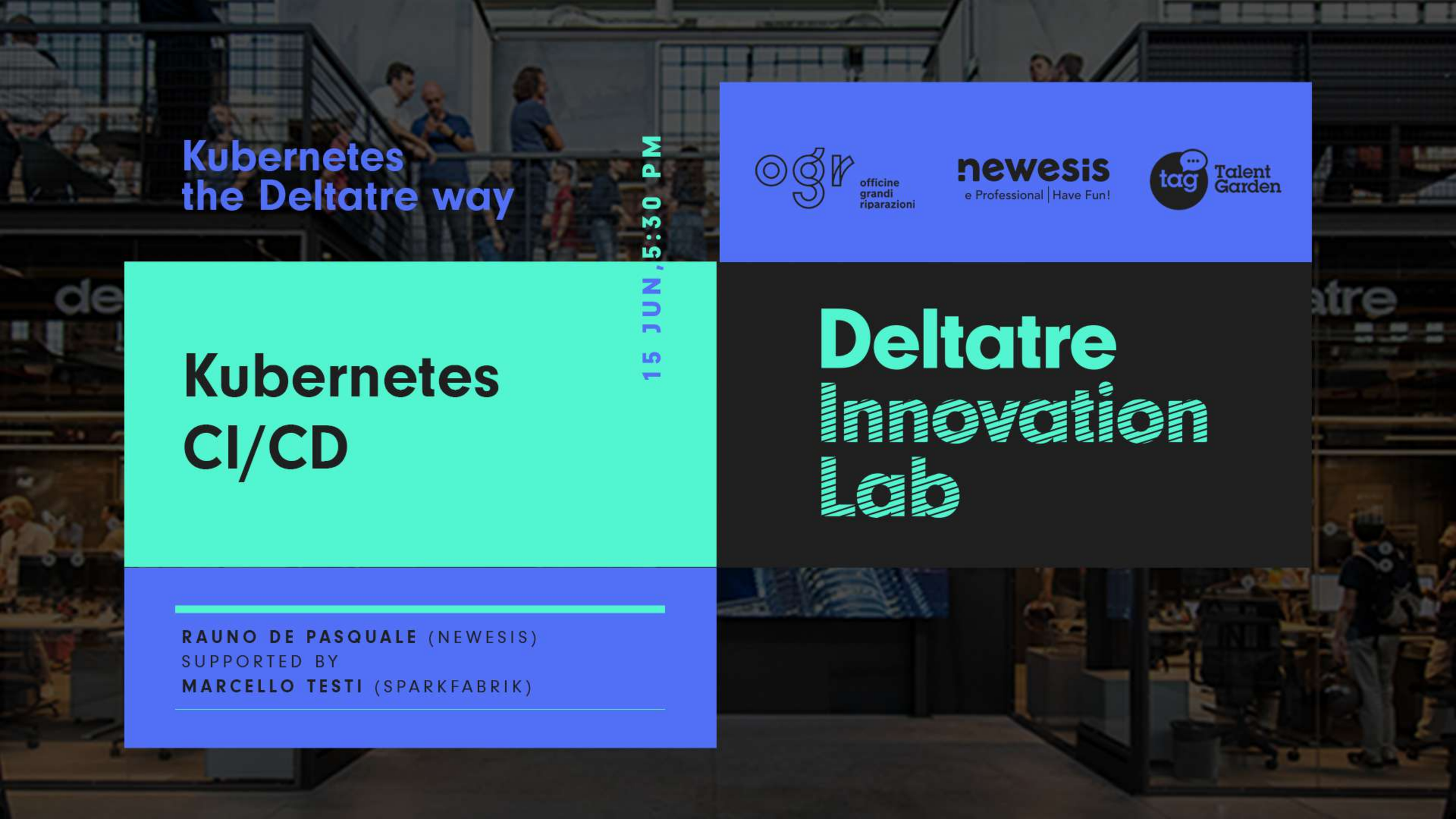
DON’T TRY THIS AT HOME!!!

DEMO TIME

deltatre



Q & A


Kubernetes
the Deltatre way
15 JUN, 5:30 PM
ogr officine grandi riparazioni
newesis e Professional | Have Fun!
tag Talent Garden
Kubernetes
CI/CD
Deltatre
Innovation
Lab
RAUNO DE PASQUALE (NEWESIS)
SUPPORTED BY
MARCELLO TESTI (SPARKFABRIK)