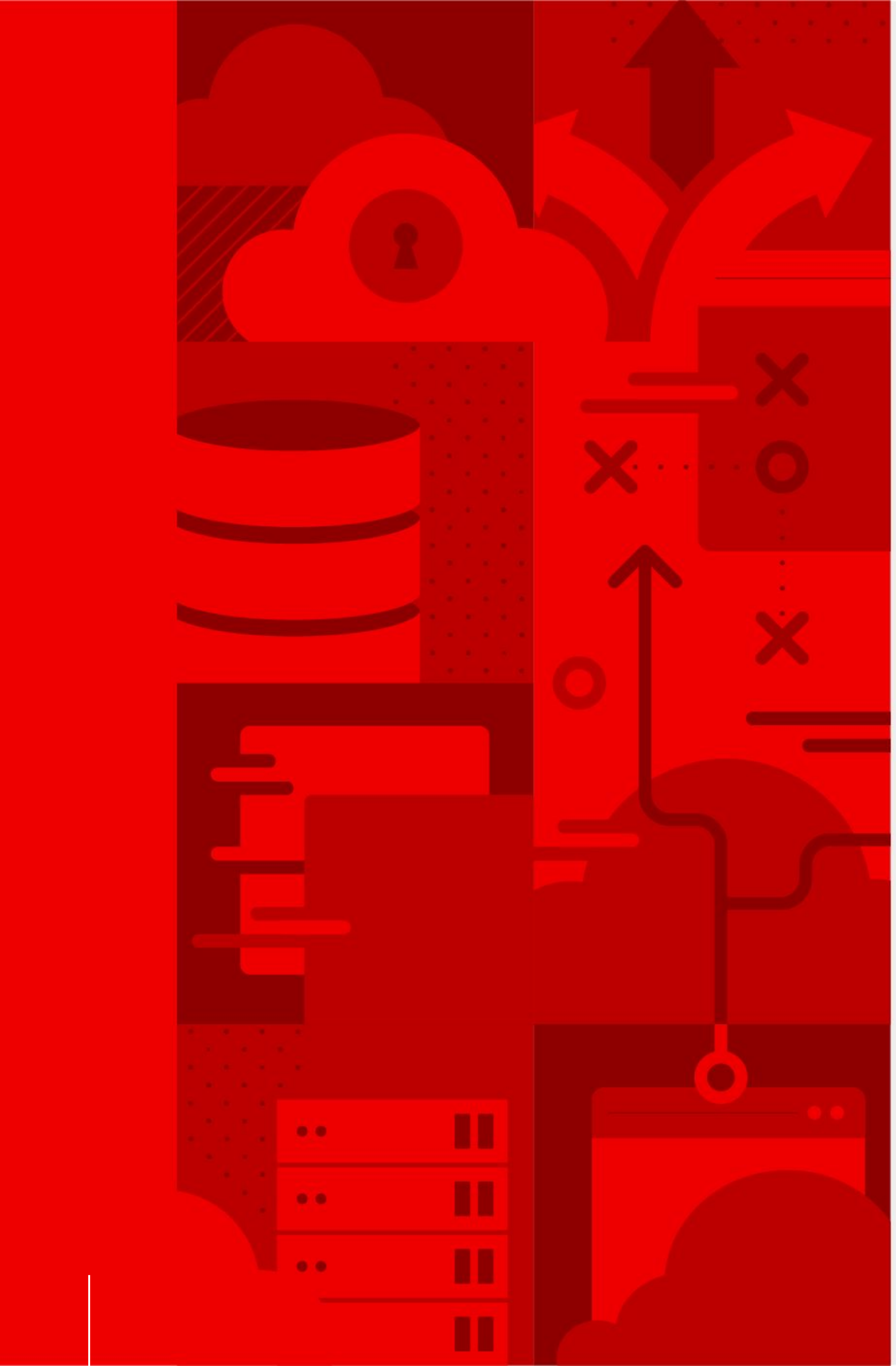


Elastic Kubernetes

Zbyněk Roubalík **@zroubalik**
Principal Software Engineer, Red Hat
OpenShift Serverless & KEDA

Roland Huß **@ro14nd@hachyderm.io**
Senior Principal Software Engineer, Red Hat
OpenShift Serverless Architect

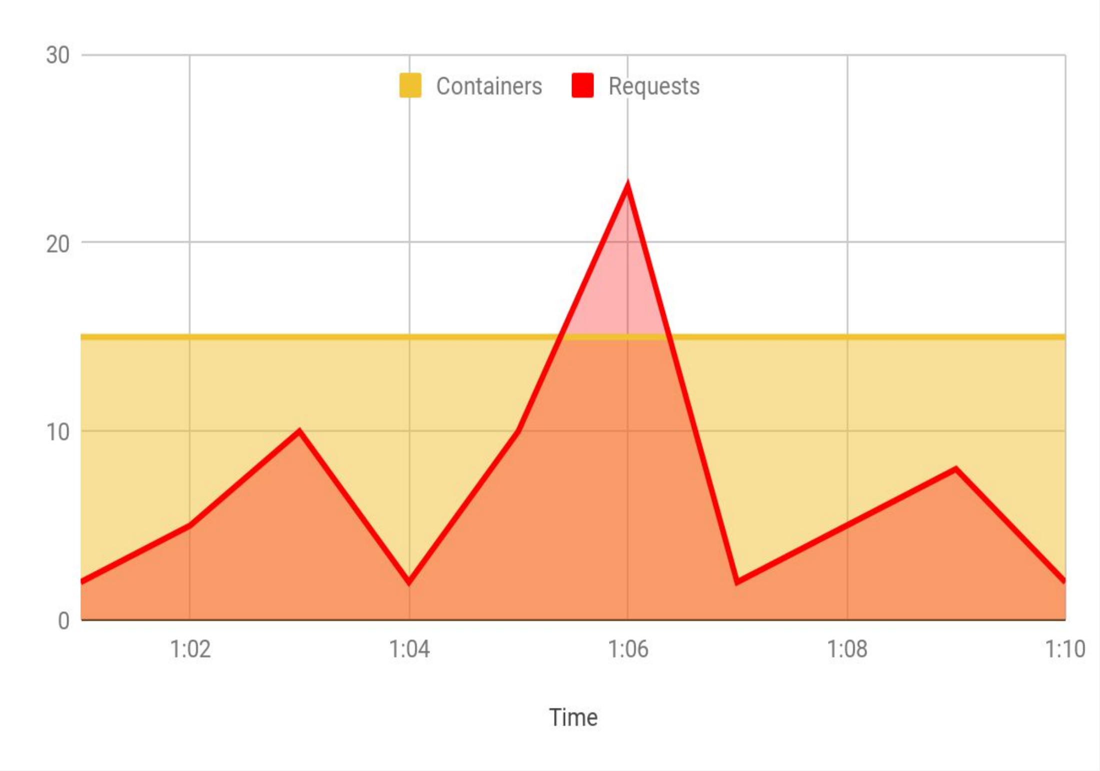
Why “Elastic Kubernetes”?

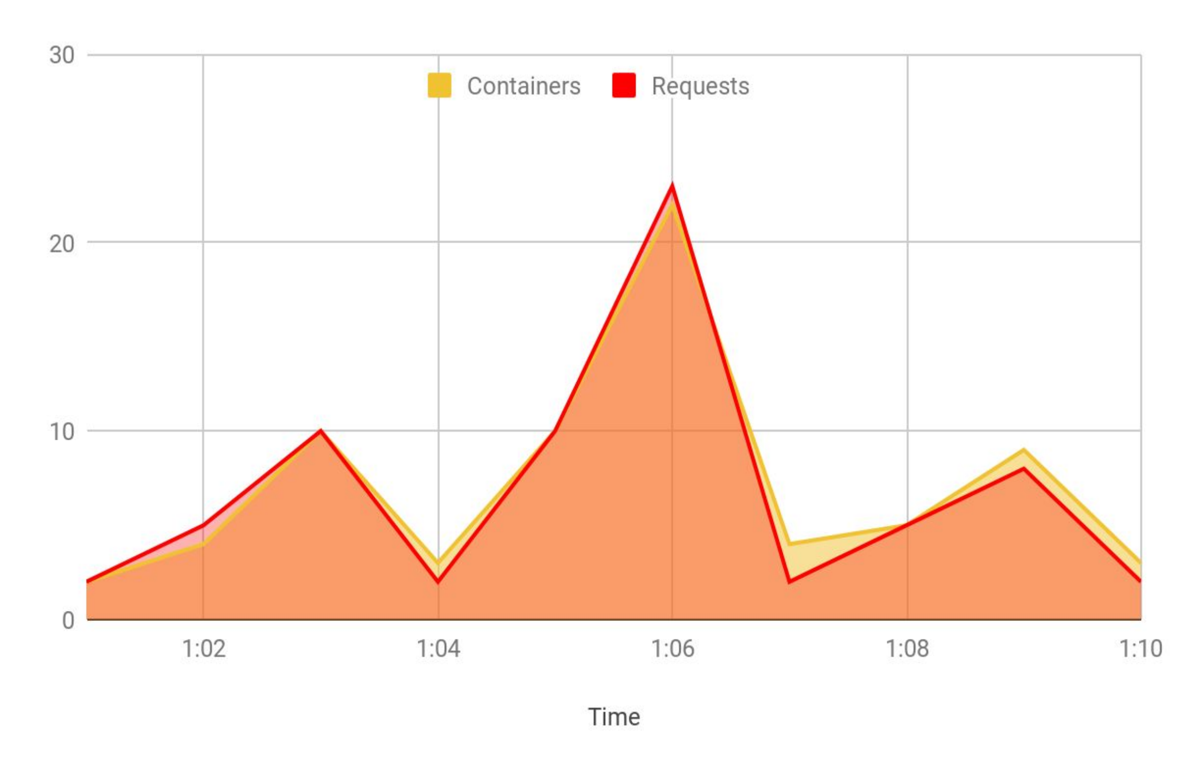
- Workloads have different size and resources requirements
  - **Under provisioning** – not enough resources might cause outages, latency and overall bad UX
  - **Over provisioning** – too many resources, expensive, wasteful

- It’s hard to specify resource and pod allocation, as resources requirements change across the time –> we need elasticity
  - Applications autoscaling
  - Cluster autoscaling

Why “Elastic Kubernetes”?

Resources defined statically vs. Enabled autoscaling





Resource requests & limits

- It’s recommended to **always set memory resource requests and limits**
  - and requests == limits
- It’s recommended to **always set cpu resource requests**
  - and no limits
- Every workload has a different set of requirements
- Capacity planning is important aspect
- If not sure about correct values -> collect metrics over time and modify gradually

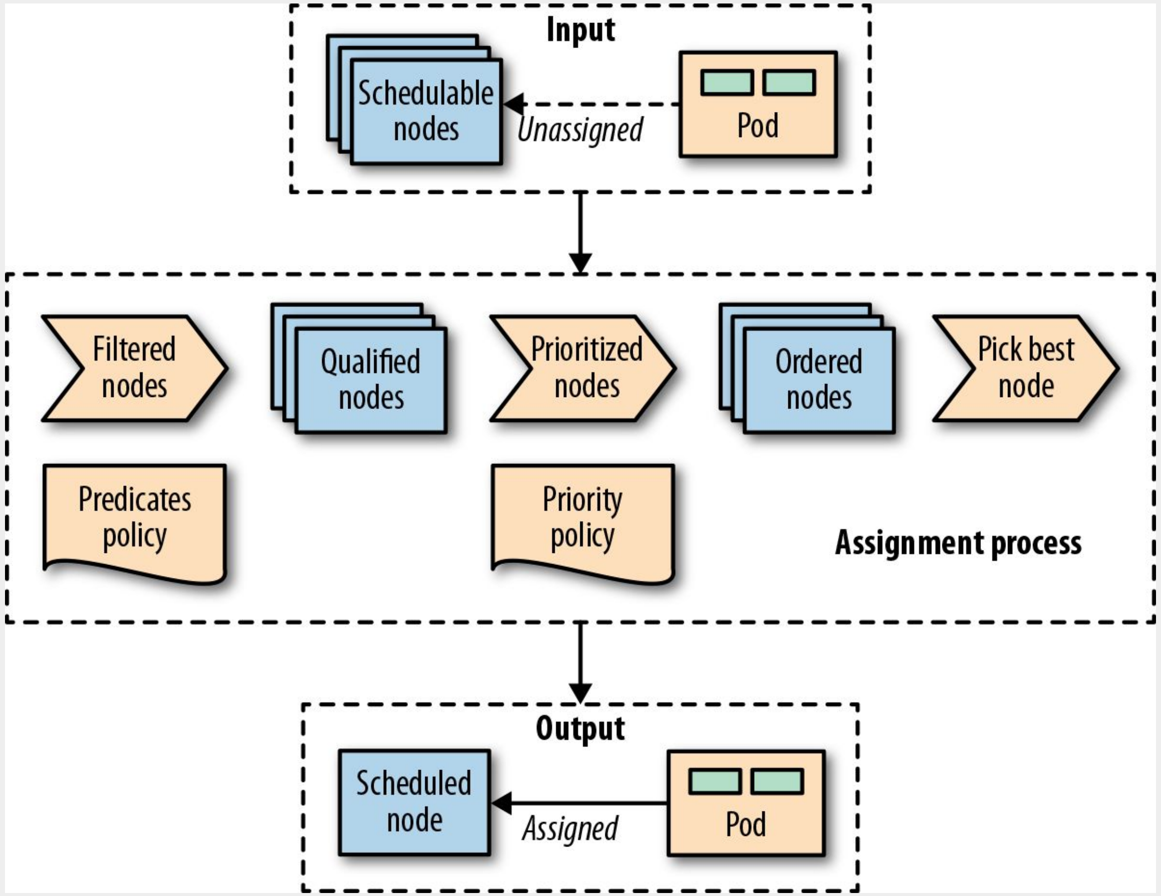
Pods:

- Resource **Requests** - the minimal amount of resources needed for the Pod
- Resource **Limits** - the maximal amount of resources that can be used by Pod

Kubernetes scheduler

- Responsible for assigning Pods to nodes
- **Predicates**: Filter of suitable nodes
- **Priorities**: Ordering of nodes according to preference

- Predicates + Priorities = Scheduler Policy

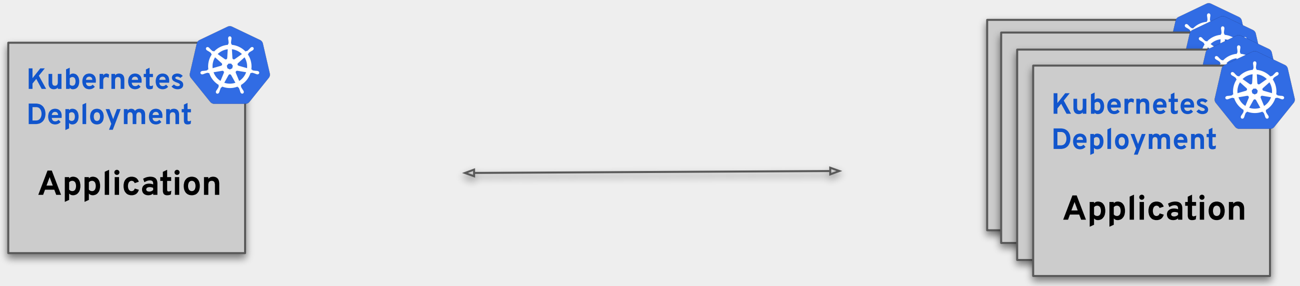


Red Hat

Horizontal Pod Scaling

Horizontal Pod Scaling

- **Scale out/in** operation
- Increasing/decreasing the number of replicas (Pods)
- Application deployment or resource requests & limits don't change



Red Hat

Manual Horizontal Pod Scaling

- Specifying the number of replicas (Pods) manually
- Can be declared:
  - Imperatively (kubectl)
  - Declaratively (yaml)

```
$ kubectl scale my-app --replicas=5
```

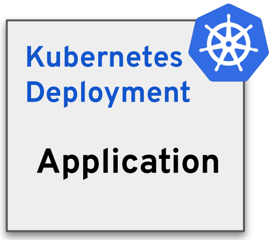
Horizontal Pod Autoscaling

- **Horizontal Pod Autoscaler** (HPA)
    - Built-in Kubernetes component
- https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/

Example:

Application performing a resource expensive task

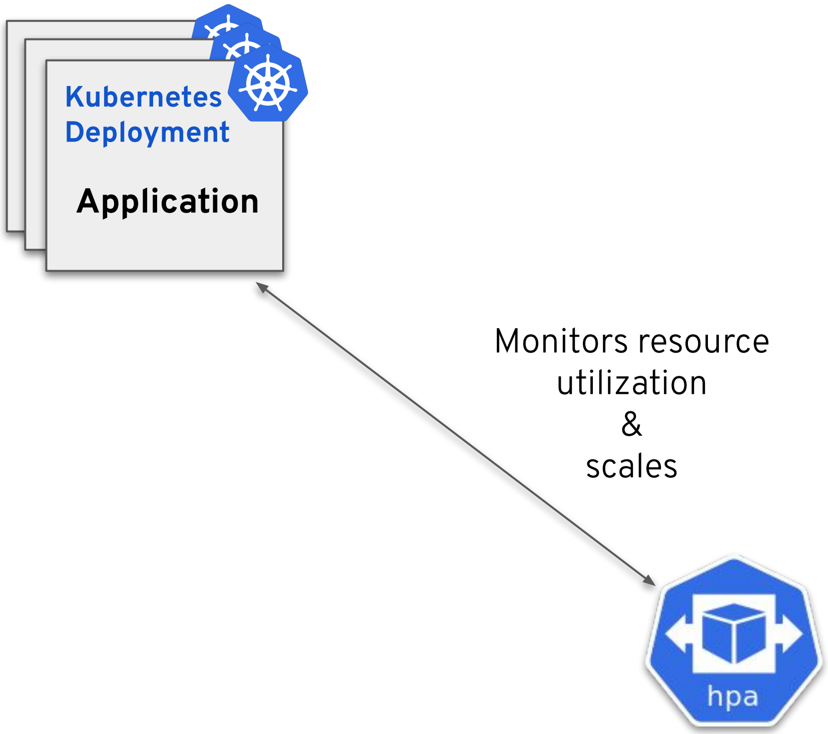
- Application is deployed as standard Kubernetes Deployment



Example:

Application redesigned to utilize Horizontal Pod Autoscaler

- Application remains the same and is being deployed the same way
- Autoscaling via HPA: based on CPU & Memory consumption



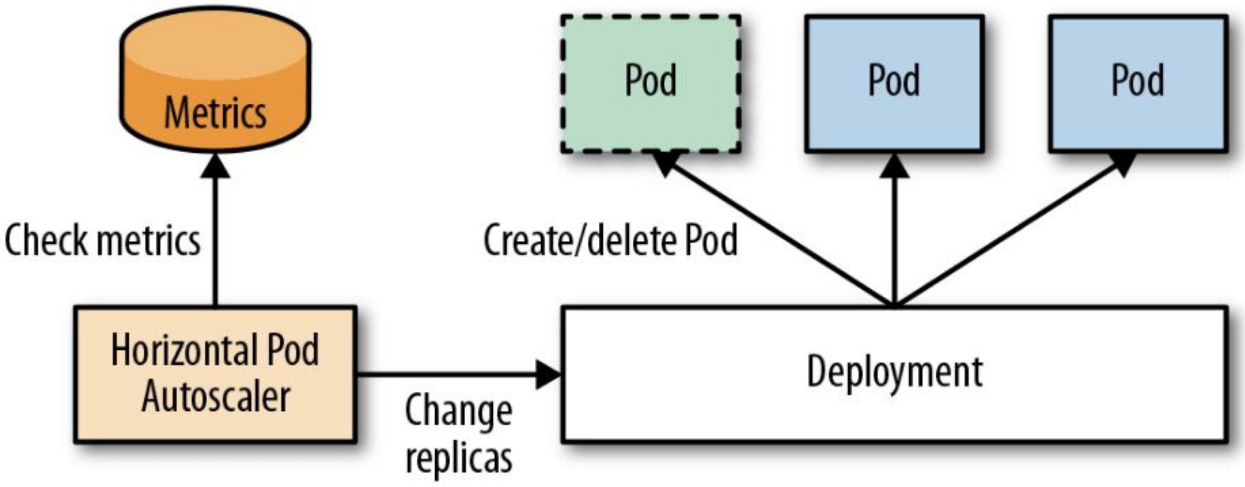
Horizontal Pod Autoscaler

- Needs Kubernetes Metrics Server enabled
  - it is a cluster wide aggregator of resource usage data
- Scales Deployments, StatefulSets and Custom Resources that enable the `/scale` subresource
- Target workload needs to specify **resource limits**
- Metric Types:
  - **Resource metrics** - cpu/memory utilization (built-in)
  - **Custom / External Metrics** - metrics about custom resources
- Can **not scale to 0** (for custom metrics currently in development)

Horizontal Pod Autoscaler

- HPA operates on the ratio between desired metric value and current metric value:

```
desiredReplicas = ceil(
                         currentReplicas * (currentMetricValue / desiredMetricValue)
                       )
```



Horizontal Pod Autoscaling

HPA

- Can be declared:
  - Imperatively (kubectl)
  - Declaratively (yaml)

- Multiple metrics can be defined

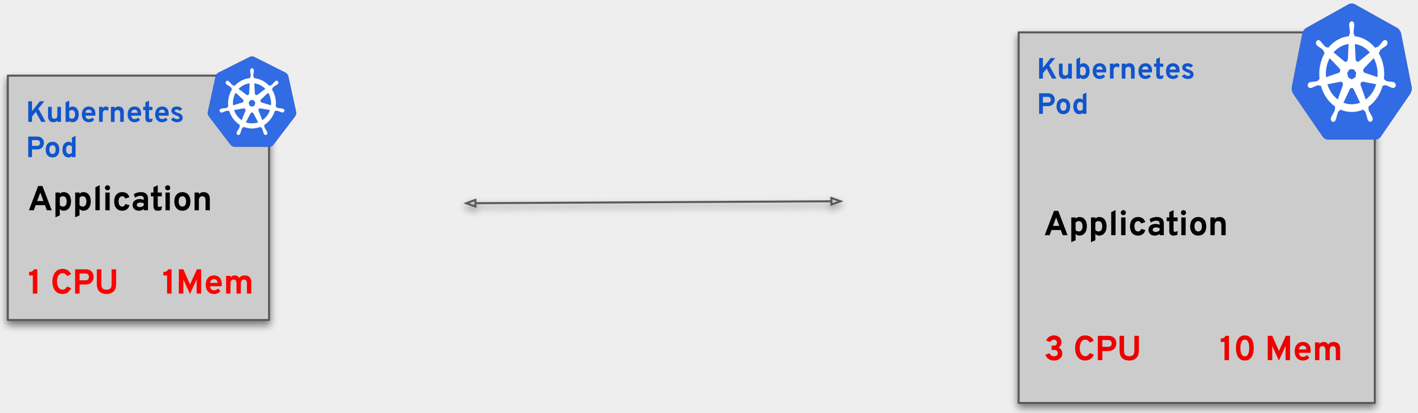
```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
    name: example-hpa
spec:
 scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: my-app
 minReplicas: 1
 maxReplicas: 10
 metrics:
 - type: Resource
   resource:
      name: cpu
      target:
         type: Utilization
         averageUtilization: 50
```

```
$ kubectl autoscale deployment my-app --cpu-percent=50 --min=1 --max=10
```

Vertical Pod Scaling

Vertical Pod Scaling

- **Scale up/down** operation
- Increasing/decreasing the amount of resources assigned to the Pod
- Number of replicas doesn't change



Vertical Pod Autoscaling

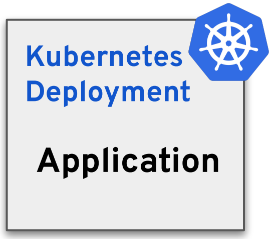
- **Vertical Pod Autoscaler** (VPA)
  - Add-on that needs to be installed on cluster
- https://github.com/kubernetes/autoscaler/tree/master/vertical-pod-autoscaler

Example:

Application performing a resource expensive task

- Application is deployed as standard

  Kubernetes Deployment

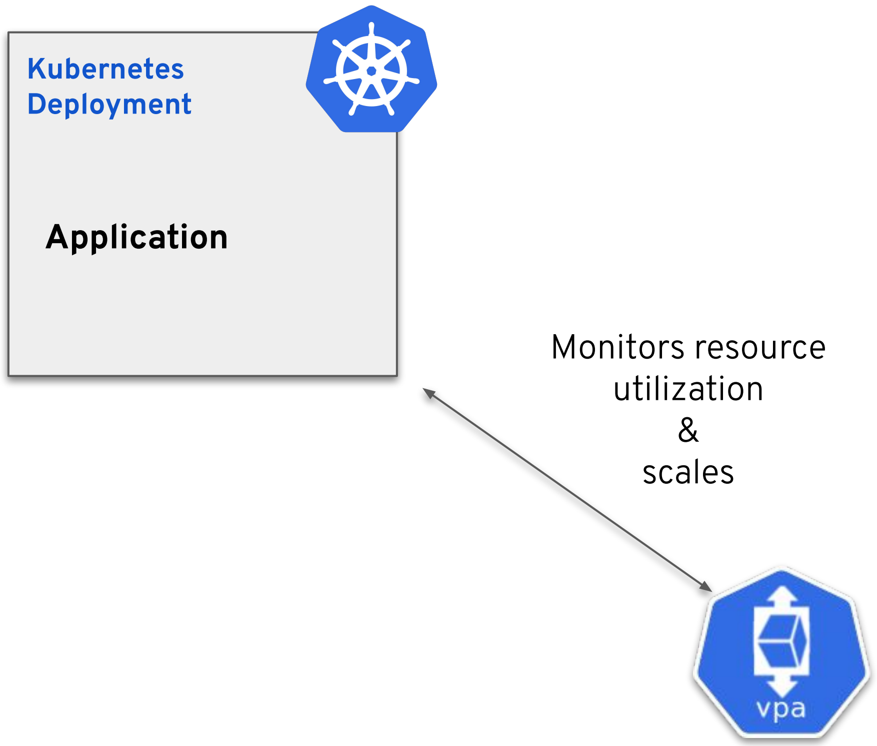


Example:

Application redesigned to utilize Vertical Pod Autoscaler (VPA)

- Resources assigned to the application has been controlled by VPA based on resource utilization

- Autoscaling via VPA: based on CPU & Memory consumption

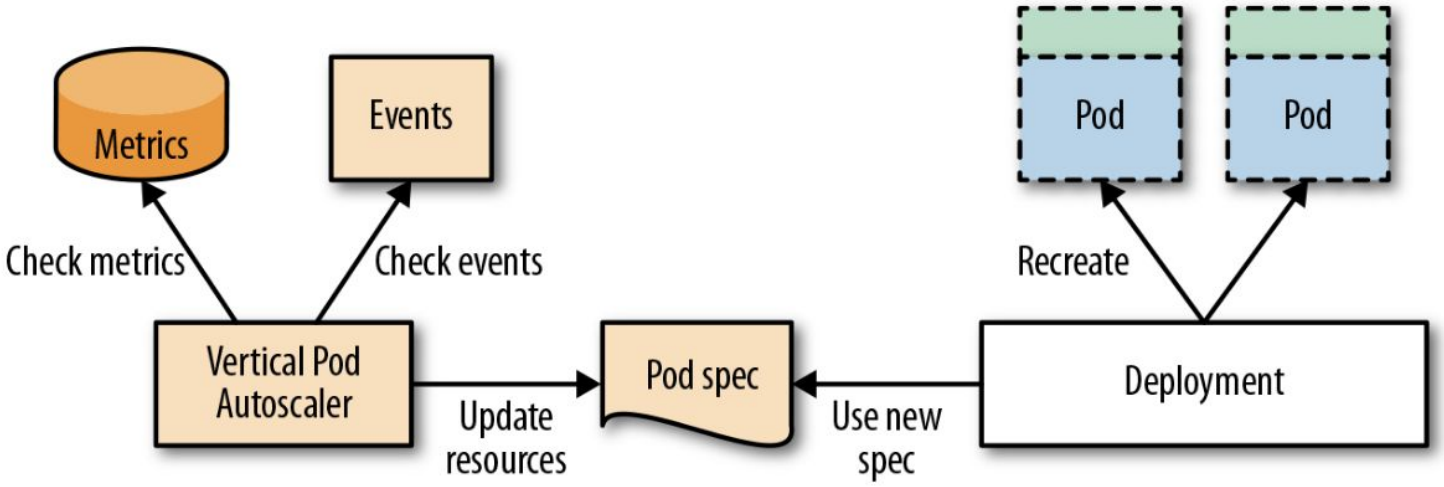


Vertical Pod Autoscaler

- **Determines** resource limits/requests based on historic and current metrics
- Containers might be **rescheduled** on a different node based on VPA limit recommendations
- Can **not be used** together with **HPA**
- Four modes:
    - **Auto/Recreate** – automatically apply the VPA resource recommendations
    - **Initial** – apply the VPA recommendations only at pod creation
    - **Off** – only provides the VPA recommendations in the status section

Vertical Pod Autoscaler

- **Recommender** – monitors resource utilization and computes target values
- **Updater** – evicts those pods that need the new resource limits
- **Admission Plugin** – sets the correct resource requests on new pods



VPA

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
    name: example-vpa
spec:
  targetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: my-app
  updatePolicy:
      updateMode: “Off”
```

```
$ kubectl describe vpa example-vpa
```

```
...
status:
  recommendation:
    containerRecommendations:
    - containerName: my-container
      lowerBound:
        cpu: 25m
        memory: 262144k
      target:
        cpu: 25m
        memory: 262144k
      uncappedTarget:
        cpu: 25m
        memory: 262144k
      upperBound:
        cpu: 262m
        memory: "274357142"
```

Demand based Autoscaling

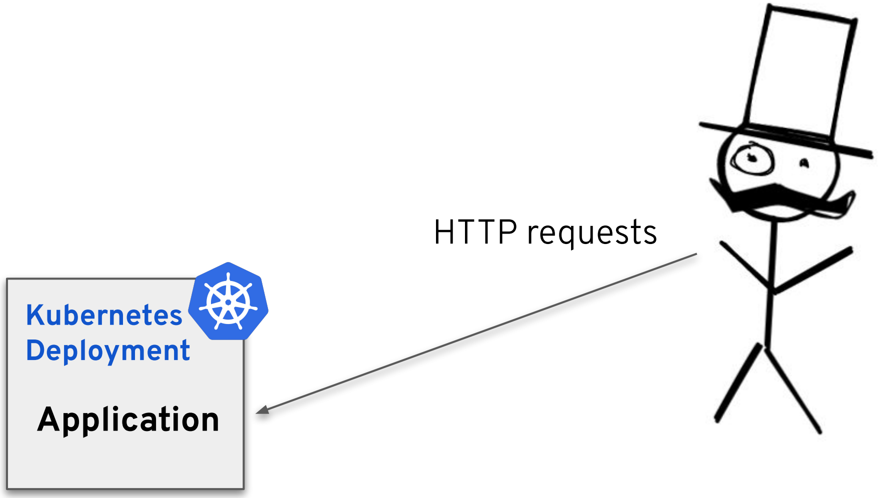
Demand Based Autoscaling

- **Knative Autoscaler**
  - Core component of Knative Serving
  - CNCF project
- https://knative.dev/docs/serving/autoscaling/autoscaler-types/#knative-pod-autoscaler-kpa

Example:

Application serving HTTP requests

- Application is deployed as standard Kubernetes Deployment
- Can be autoscaled only via standard k8s HPA: CPU & Memory
- No demand based autoscaling

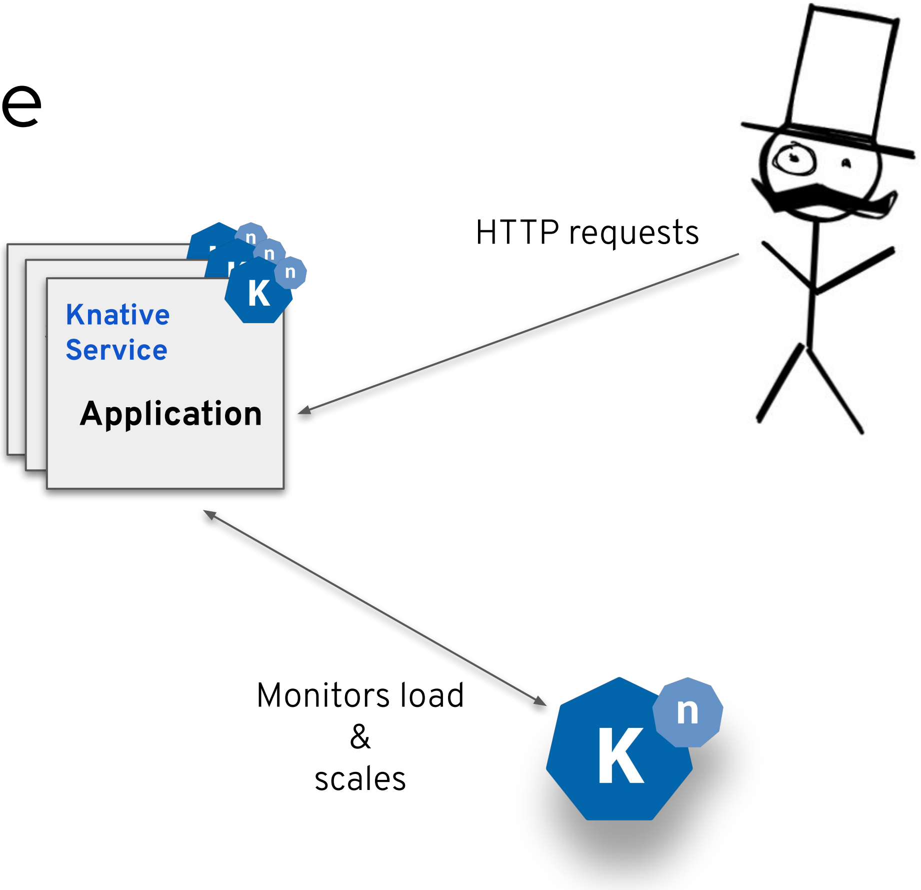


Example:

Application redesigned to utilize Knative

- Application is deployed as Knative Service

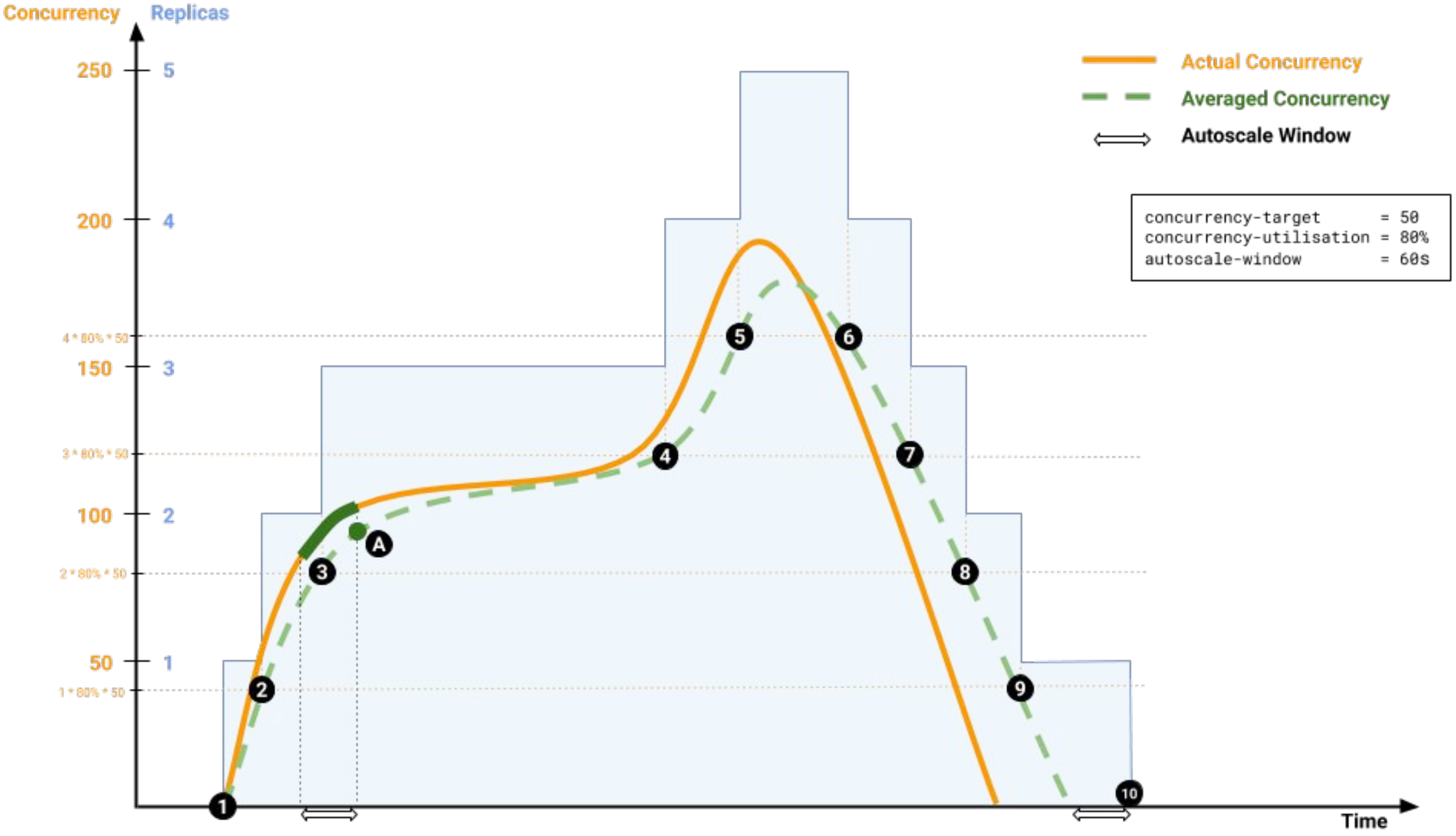
- Knative Autoscaler monitors load and scales the application based on demand



Knative Autoscaler Concepts

- Knative Autoscaler scales **Knative Service**, a CR representing the workload, it manages needed Kubernetes resources (Deployment, Service, Ingress,...)
- **Activator** component enables scale to 0
  - Incoming requests are being hold until the app is scaled to 1 replica
- Autoscaler itself has 3 components:
  - **PodAutoscaler Reconciler** - ensures that all components are up to date
  - **Collector** - collect metrics from various sources
  - **Decider** - based on metrics decides how the app should be scaled
    - `want = concurrencyInSystem/targetConcurrencyPerInstance`

Knative Autoscaler



```
concurrency-target      = 50
concurrency-utilisation = 80%
autoscale-window        = 60s
```

❶ Scale up from 0 to 1 replica on first request.

❷ Scale from 1 to 2 replicas if the utilisation 80% of the concurrency target 50 is reached for the **averaged concurrency**.

❸ ... ❾ Up- and downscaling events when **averaged concurrency** crosses the utilisation threshold counted across the current number of replicas. (2 * 80% * 50 = 80, 3 * 80 % * 50= 120, ...)

❿ Scale down to 0 when **averaged concurrency** is going down to 0 for the length of the autoscale window.

Ⓐ The **averaged concurrency** is calculated every 2 seconds by averaging **concurrent requests** for the past auto-scale window length (default: 60s)

Knative Service

- Application deployed as Knative Service

- Can be declared:
  - Imperatively (kn cli)
  - Declaratively (yaml)

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
    name: example
spec:
 template:
   spec:
     containers:
       - image: johndoe/my-image
         ports:
           - containerPort: 8080
```

```
$ kn service create example --image johndoe/my-image --port 8080
```

Event Driven Autoscaling

Event Driven Autoscaling

KEDA

- **K**ubernetes **E**vent **D**riven **A**utoscaling dead simple
  - CNCF project
- https://keda.sh

Red Hat

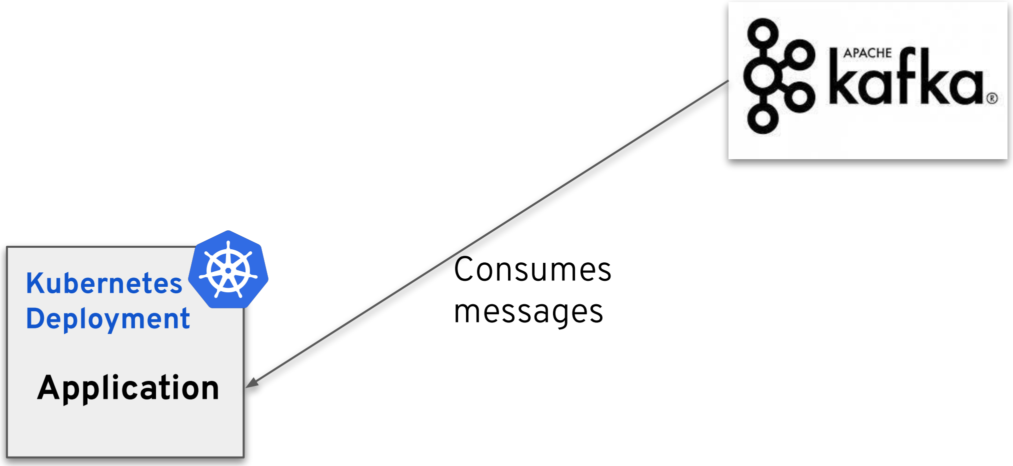
Example:

Application consuming messages from Kafka topic

- Application is deployed as standard Kubernetes Deployment

- Can be autoscaled only via standard k8s HPA: CPU & Memory

- No event-driven autoscaling

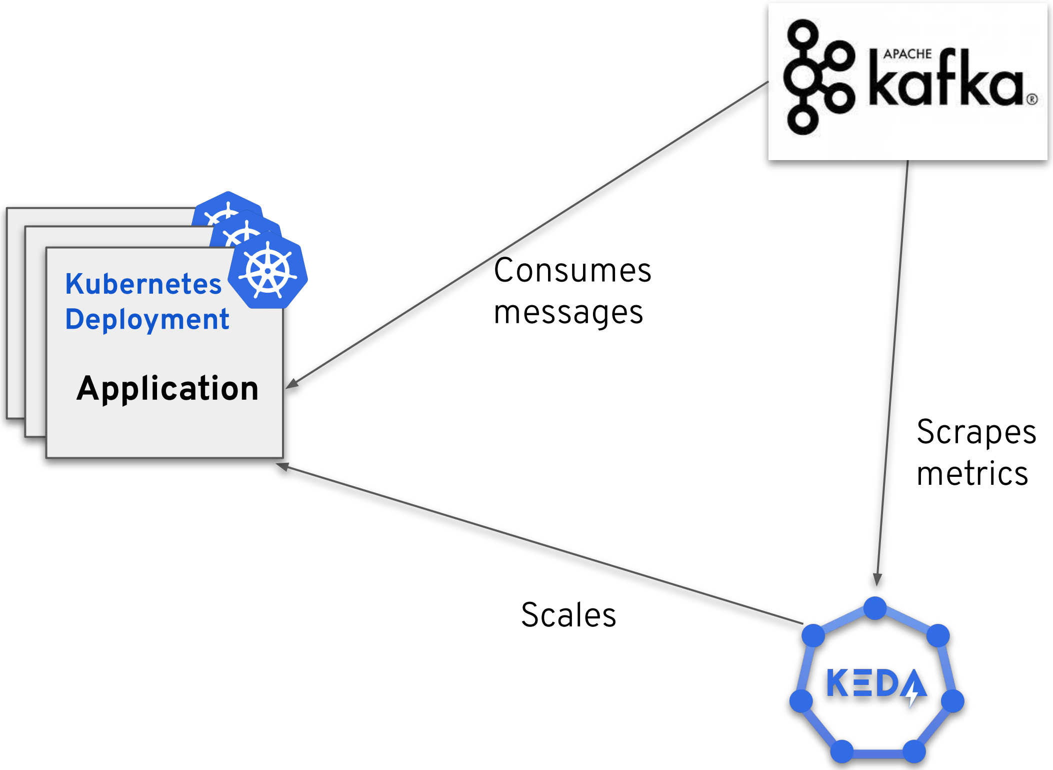


Example:

Application redesigned to utilize **KEDA**

- Application remains the same and is being deployed the same way

- Event driven autoscaling enabled through KEDA

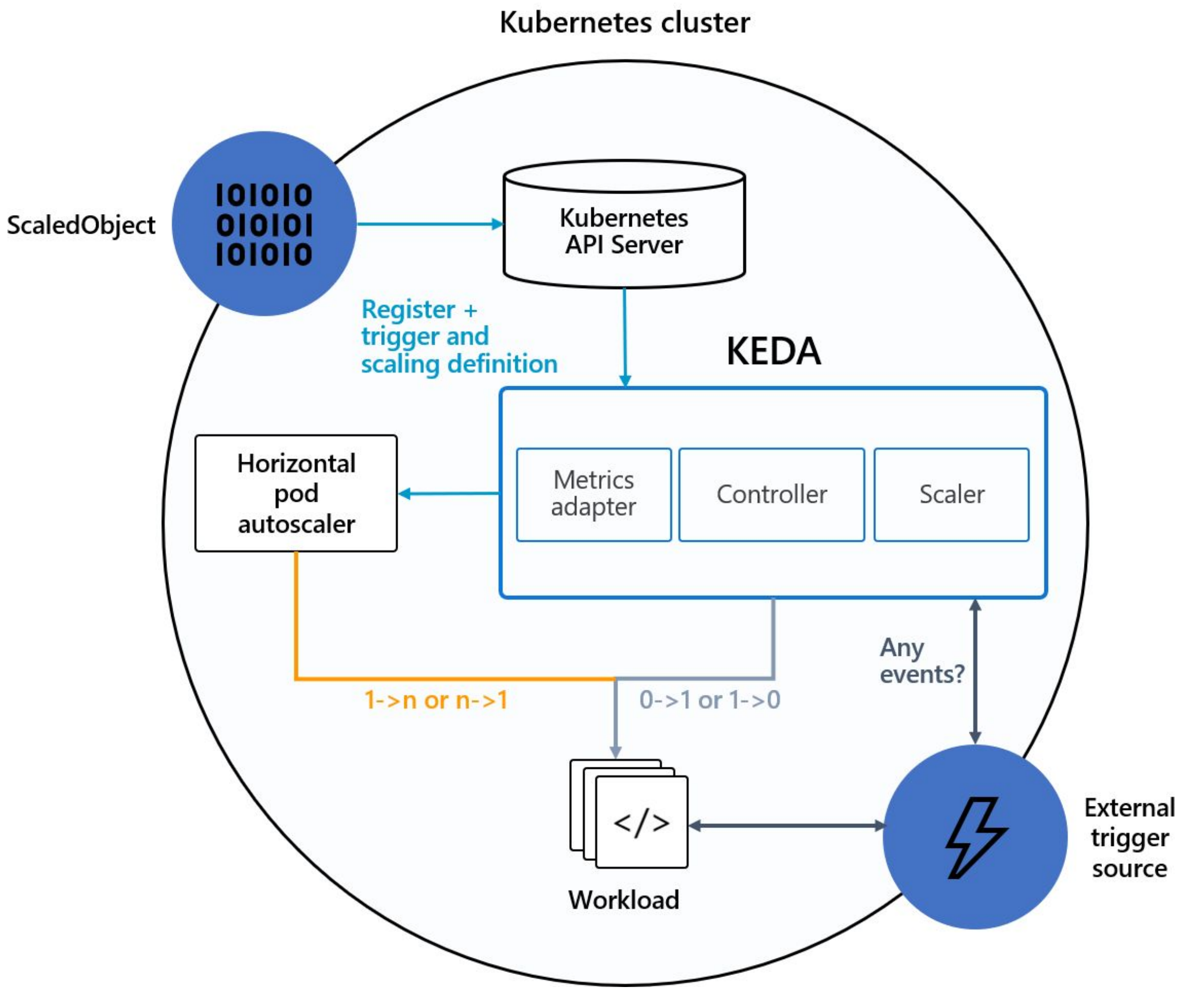


KEDA Concepts

- Automatically scale Kubernetes Deployments, Jobs & Custom Resources
- Provides **50+** built-in scalers, but users can build own external scalers
  - Kafka, Prometheus, RabbitMQ, AWS services, Azure Services,...
- Scale resources **based on events** in the target scalers, eg. messages in Kafka topic
- KEDA does not scale on HTTP requests
- Save resources by **scale to 0**
- KEDA **does not** manipulate the data, just scales the workload

KEDA Architecture

- KEDA is built on top of Kubernetes
- Use **ScaledObject/ScaledJob** to define scaling metadata
- Manages workloads to scale to 0
- Registers itself as Kubernetes Metric Adapter
- Provides external metrics for Horizontal Pod Autoscaler (HPA)



ScaledObject

- Can target Deployment, StatefulSet or Custom Resource with /scale
- Multiple scalers can be defined as triggers for the target workload
- User can specify HPA related settings to tweak the scaling behavior

```yaml
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
 name: example-so
spec:
 scaleTargetRef:
       name: example-deployment
 minReplicaCount: 0
 maxReplicaCount: 100
 triggers:
 - type: kafka
   metadata:
       bootstrapServers: kafka.svc:9092
       consumerGroup: my-group
       topic: test-topic
       lagThreshold: '5'
```

Demo

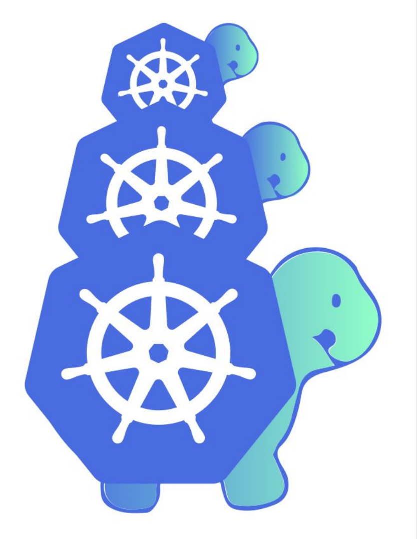
Cluster Autoscaling

Cluster Autoscaling

- When autoscaling of applications is not enough
- Autoscaling of applications (more Pods and Containers) makes pressure on the infrastructure as well
- We should scale Kubernetes Nodes together with applications to really achieve “Elastic Kubernetes”

Cluster API & Cluster Autoscaler

- Standardized around **Cluster API**:
  - To manage the lifecycle of Kubernetes conformant clusters using a declarative API.
  - To work in different environments, both on-premis cloud.



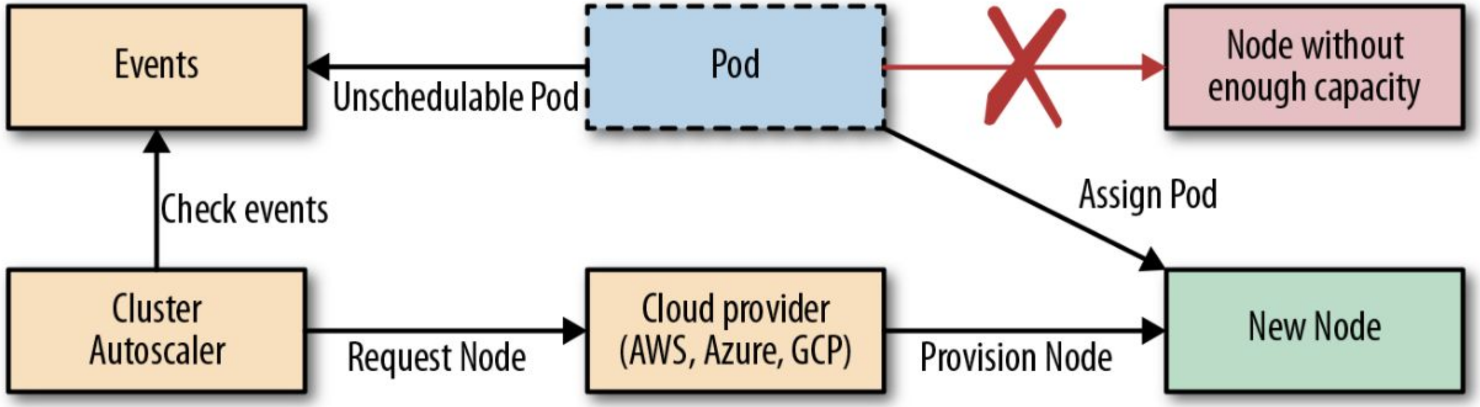
- There are cloud vendor specific implementations of cloud autoscalers
- https://cluster-api.sigs.k8s.io/

Cluster Autoscaler

- Adjusts the number of Nodes in the cluster when Pods fail to schedule or when nodes are underutilized
- Makes decision based on requests and limits for CPU and memory resources, **not the actual load**!
- Best practices:
  - Set correct resources requests & limits
  - Use PodDisruptionBudgets to prevent pods from being deleted too abruptly
  - Check if your cloud provider's quota is big enough

Cluster Autoscaler

- Unschedulable pods makes pressure on Cluster Autoscaler -> new Node is provisioned
- In case Node is not needed -> Cluster Autoscaler terminates the underlying instance in a cloud-provider-dependent manner



Cluster Autoscaler

- Technique: Keep an empty spare node in the cluster to reduce provisioning time
- Shrinking the cluster is hard is it require rebalancing the cluster.
- Works on a different time-scale than application auto-scaling
  - it takes much more time to spin up a new node than a new pod

Summary

Summary

- **Why Elastic Kubernetes**
- **Application Autoscaling**
  - HPA
  - VPA
  - Knative Autoscaler
  - KEDA
- **Cluster Autoscaling**

Thank you

@ro14nd@hachyderm.io

@zroubalik

Picture Credits

https://kubernetes.io/docs/concepts/scheduling-eviction/scheduling-framework/

https://www.pexels.com/photo/boat-island-ocean-sea-218999/

https://pixabay.com/photos/hamburg-speicherstadt-channel-2976711/

https://pixabay.com/photos/beer-machine-alcohol-brewery-1513436/

https://pixabay.com/photos/audience-band-concert-crowd-1853662/

https://pixabay.com/photos/metal-tube-chair-stack-stacked-1756245/

https://pixabay.com/photos/above-summary-the-activities-of-the-1956156/