universität innsbruck

Advanced Distributed Systems

Lecture 08-Autoscaling

Dr. Zahra Najafabadi Samani

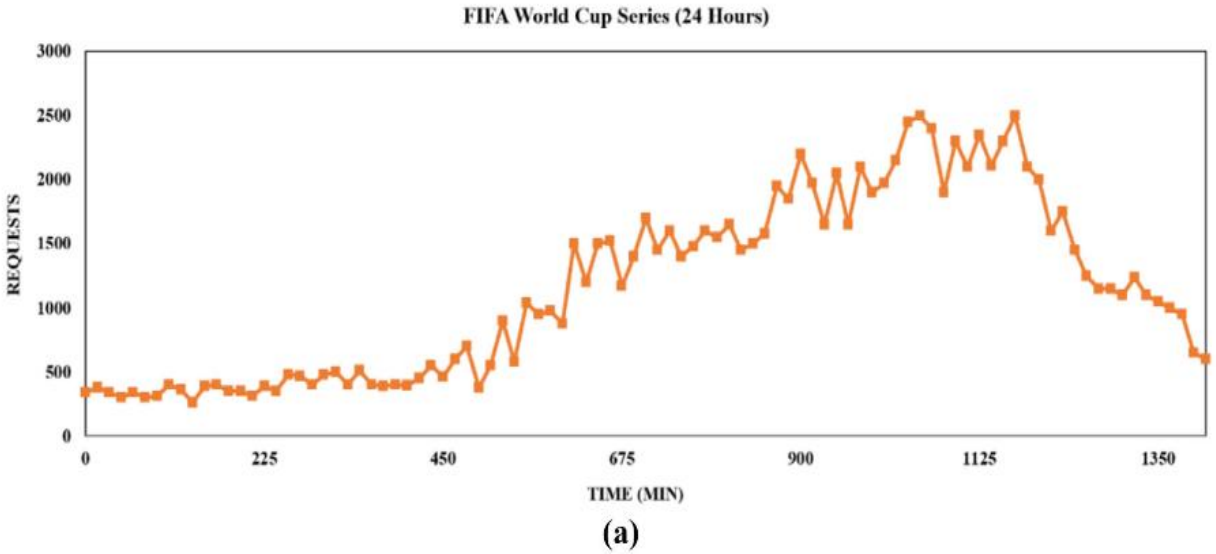
Agenda

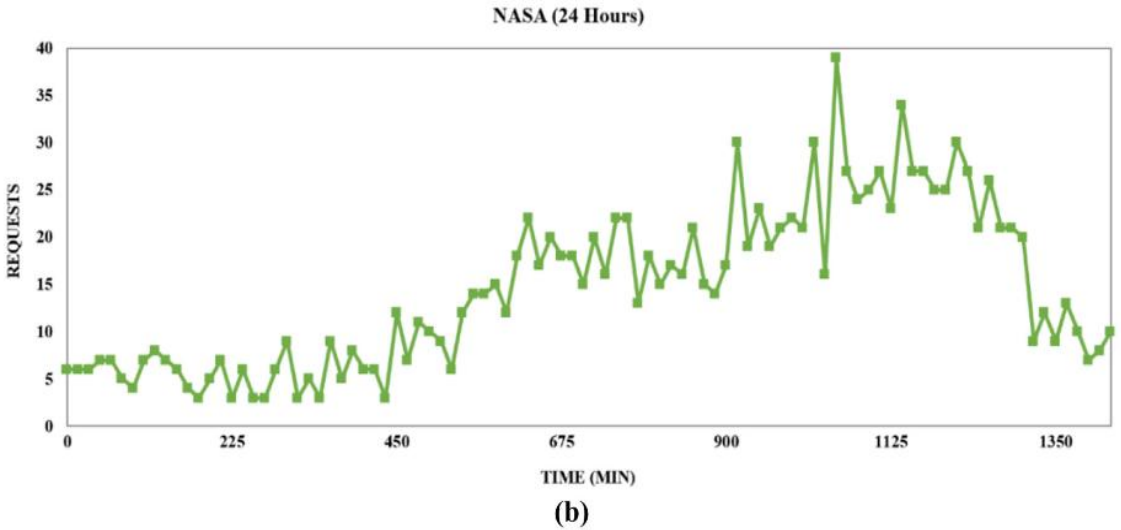
- Motivation
- Autoscaling
- Different type of autoscaling
- Autoscaling in Kubernetes

Motivation

Why autoscaling?

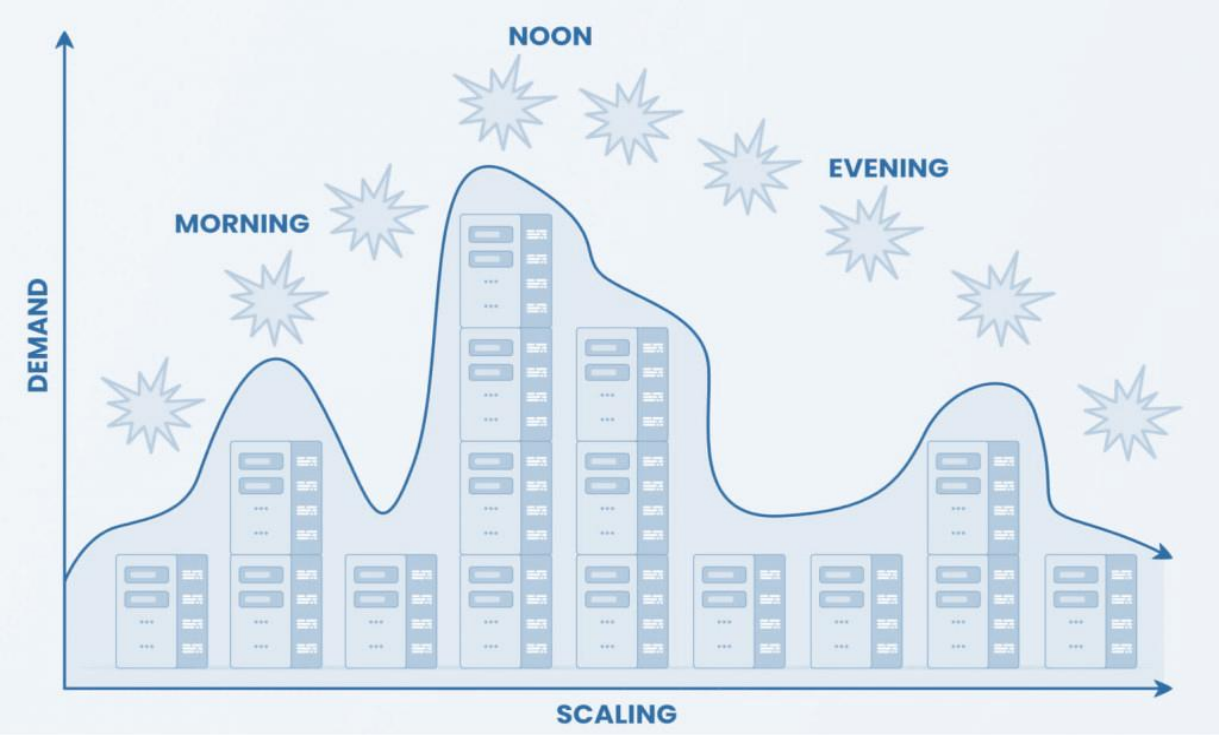
Workloads in various applications often exhibit significant fluctuations over time.





Why autoscaling?

Workloads in various applications often exhibit significant fluctuations over time.

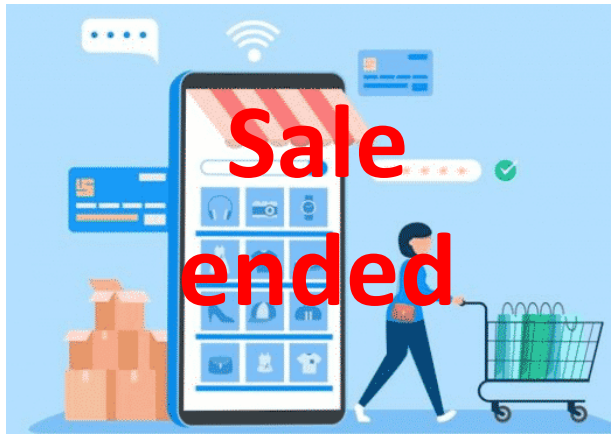


Why autoscaling?

Scale up
Increase the resources

Stay responsiveness
Prevent down time
Increase the performance

Why autoscaling?



Scale down
decrease the resources

Save cost
Reduce resource wastage

Why autoscaling?

Workloads in various applications often exhibit significant fluctuations over time leading to several critical challenges:

- **Resource inefficiency:** During periods of low demand system provides higher capacity than necessary, resulting in resource inefficiency such as escalated costs and unnecessary resource wastage.
- **QoS violation:** The system's incapacity to dynamically adjust to varying workloads can lead to performance issues such as increased response times and the risk of service failure, particularly during peak usage periods.
- **Increased management complexity:** Manually managing resources to meet fluctuating demand can be complex and time-consuming and making it challenging to efficiently allocate resources and respond quickly.

Autoscaling

What is autoscaling?

- Automatically adjusts computing resources based on real-time demand.
- Relies on automated processes and orchestration to reduce the need for manual intervention.
- Quickly scales resources in response to fluctuations in demand for consistent performance.
- Ensures optimal performance by scaling resources up or down to match current workloads.
- Controls costs by provisioning and utilizing resources only when necessary.
- Allows administrators to define policies based on metrics like CPU utilization, or other metrics.
- Integrated with load balancing to evenly distribute traffic and optimize resource utilization.

Advantage of autoscaling

Efficient resource utilization: Automatically adjusts resources based on demand, minimizing underutilization or overprovisioning.

Cost savings: Ensures cost efficiency by scaling resources up during peak demand and down during periods of low activity, avoiding unnecessary expenses.

Improved performance: Maintains optimal performance levels by dynamically adapting to changes in workload, preventing performance degradation during high-demand periods.

Enhanced scalability: Facilitates the seamless expansion or contraction of infrastructure, allowing systems to handle increased loads without manual intervention.

Reduced downtime: Enhances system reliability by distributing workloads and adapting to changes, reducing the risk of service outages and downtime.

Advantage of autoscaling

- **Automatic response to demand fluctuations:** Provides a quick and automated response to changes in demand, ensuring that resources align with the current needs of the system.
- **Increased flexibility:** Offers flexibility in managing varying workloads without requiring constant manual adjustments, allowing for a more adaptive and responsive infrastructure.

Different type of autoscaling

Different types of autoscaling

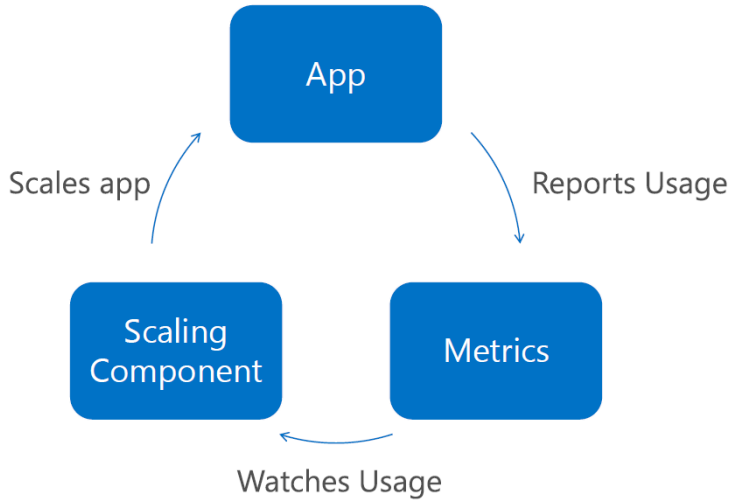
Fundamentally there are three types of autoscaling:

- Reactive
- Proactive
- Hybrid

Reactive autoscaling

Reactive autoscaling involves dynamically adjusting resources based on immediate changes in demand, responding to predefined triggers or conditions rather than proactively anticipating future requirements.

- **Key features:**
  - **Monitoring:** Continuously monitoring key performance indicators (KPIs) such as CPU usage, memory, and network traffic in real-time.
  - **Scaling polices:** defining rules for resource allocation adjustments and reactively response to specific conditions.
  - **Common Metrics:** CPU utilization, network traffic, or response times are often used as triggers for reactive scaling.
  - **Triggered by Events:** triggering by predefined events or conditions, such as increased traffic or high resource utilization.



Advantages of Reactive autoscaling

- **Simplicity:** It is often simpler to implement, especially with rule-based systems.
- **Cost Control:** In some scenarios, reactive autoscaling may help control costs by only scaling resources when predefined triggers are met. This can prevent unnecessary scaling during periods of low demand.
- **Predictability:** With well-defined triggers and policies, reactive autoscaling can provide a predictable response to specific events, making it easier to understand and manage.
- **Continuous Improvement:** Regularly review and refine the reactive autoscaling setup based on performance analysis and feedback. Adjust trigger conditions, scaling policies, and other parameters to enhance efficiency and responsiveness.

Reactive autoscaling limitation

- **Latency:** One of the primary drawbacks is the inherent latency between the occurrence of a triggering event and the initiation of scaling actions. This delay can impact performance during sudden spikes in demand.
- **Overprovisioning/uderprovisioning:** Reactive autoscaling may lead to overprovisioning, where resources are added too late or in excess. This can result in unnecessary costs and underutilized resources during periods of decreased demand.
- **Limited Adaptability:** Rule-based reactive systems may lack adaptability to dynamic and unpredictable workloads. These systems may struggle to cope with scenarios not covered by predefined rules.
- **Complexity in Rule Tuning:** Fine-tuning the rules for reactive autoscaling can be challenging. Constant adjustments may be needed to avoid both false positives and negatives, making it an ongoing maintenance task.

Proactive autoscaling

Proactive autoscaling employs predictive analytics and machine learning to forecast upcoming changes in demand. It dynamically adjusts resources ahead of actual spikes, optimizing performance and resource utilization.

- **Key features:**
  - **Monitoring and Analytics:**
    - Continuous monitoring of key performance indicators (KPIs).
  - **Predictive Algorithms:**
    - Utilization of historical data for trend analysis.
    - Machine learning models to forecast future demand.
    - Identify patterns and anomalies.
  - **Automated Scaling:**
    - Trigger scaling actions before reaching critical points.
    - Preemptive adjustments based on predictions.

Advantage of proactive autoscaling

- **Improved Performance:** Anticipating future demand allows for timely resource allocation, ensuring optimal system performance and responsiveness during peak periods.
- **Cost Optimization:** Proactive autoscaling helps in efficient resource utilization by scaling up or down based on predicted demand. This can lead to cost savings by avoiding unnecessary overprovisioning.
- **Scalability:** Proactive autoscaling is well-suited for applications with rapidly changing workloads, enabling systems to scale seamlessly in response to evolving usage patterns.
- **Predictive Analytics:** Leveraging machine learning and predictive analytics allows for more accurate forecasting, enabling organizations to make informed decisions about resource allocation.
- **Reduced Operational Burden:** With the ability to automatically adjust resources based on predictions, proactive autoscaling reduces the need for manual intervention and constant monitoring, easing the operational burden on administrators.

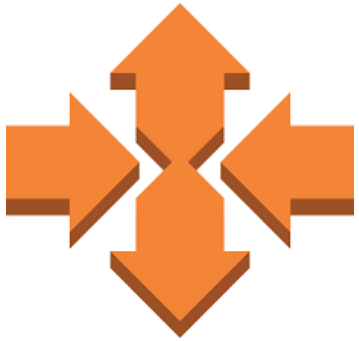
Proactive autoscaling limitation

- **Complex Implementation:** Implementing predictive analytics and machine learning models can be complex and may require expertise in data science.
- **Potential for Inaccuracies:** Predictive models may not always accurately forecast future demand, leading to overestimation or underestimation of resource needs.
- **Resource Costs for Predictive Analytics:** Developing and maintaining predictive analytics models may incur additional resource costs.
- **Adaptability to Rapid Changes:** Proactive autoscaling may face challenges in adapting to rapid and unforeseen changes in demand that fall outside the scope of historical patterns.
- **Continuous Model Refinement:** Predictive models require ongoing refinement and adjustment to accommodate changes in application behavior, making it a continuous process that demands attention and resources.

Hybrid autoscaling

Hybrid autoscaling combines both reactive and proactive scaling strategies to create a dynamic and flexible approach to managing resources in response to changing demands.

Hybrid autoscaling approaches leverages the strengths of both reactive and proactive scaling mechanisms to achieve optimal performance, resource utilization, and cost efficiency.



Different types of autoscaling

Auto-scaling encompasses a variety of strategies tailored to meet the dynamic demands of applications. Three primary types include:

- Horizontal autoscaling
- Vertical autoscaling
- Hybrid autoscaling

Horizontal autoscaling

Horizontal auto-scaling Involves adding or removing identical instances of resources, such as servers, to accommodate changing workloads. It is especially important for companies that need **high availability** services with a requirement for minimal downtime.

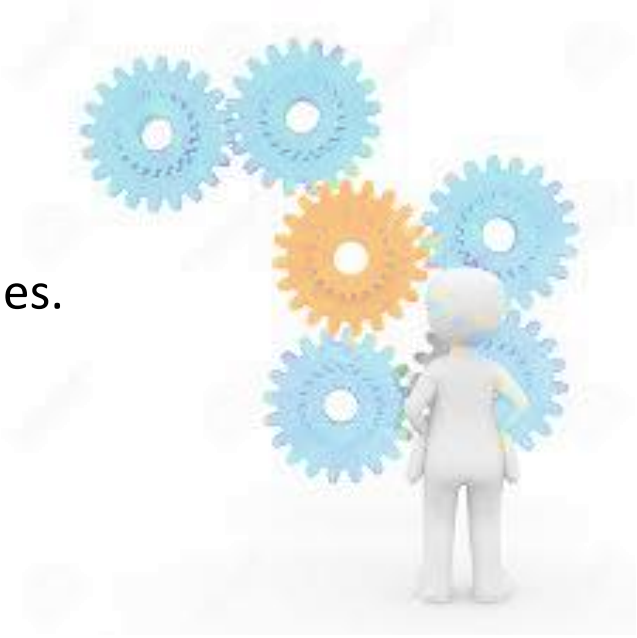
- **Key Points:**
  - **Scalability:** Achieved by adding more instances to the system.
  - **Load Balancing:** Traffic is distributed across multiple instances to ensure even resource usage.

Advantages of horizontal autoscaling

- **Increased High Availability:**
  - Spreading infrastructure across multiple areas enhances availability.
  - If one machine fails, others can seamlessly handle the load.
- **Reduced Downtime:**
  - Adding machines allows scaling without the need to switch off the old machine.
  - Minimal or no downtime required for effective scaling.
- **Easy Resizing:**
  - Scalability allows effortless adjustment according to current needs.
- **Cost Efficiency:**
  - Cost can be linked to usage and you don’t always have to pay for peak demand.

Horizontal autoscaling limitation

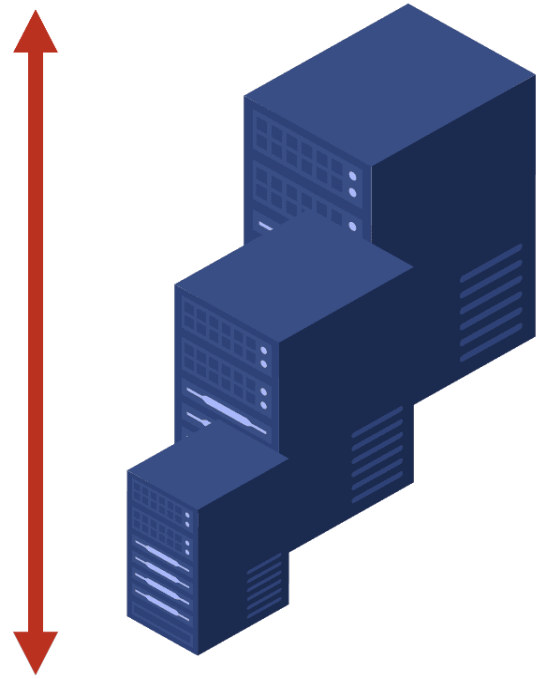
- **Use case limitation:**
  - Applications must be designed to scale horizontally.
- **Increasing complexity:**
  - Shared state and session management can be challenging.
- **Increasing Initial costs:**
  - Adding new servers is far more expensive than upgrading old ones.



Vertical autoscaling

Vertical auto-scaling Involves adjusting the capacity of a single resource, such as increasing the CPU, RAM, or storage of a server. The expansion has an upper limit based on the capacity of the server or machine being expanded.

- **Key Points:**
  - **Scaling Up:** Adding more power to existing instances by increasing resources.
  - **Scaling Down:** Reducing resources to optimize performance and cost.

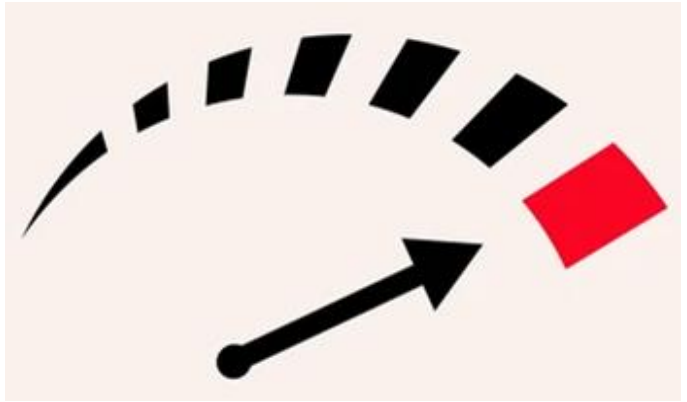


Advantages of vertical autoscaling

- **Cost-effective:**
  - Upgrading a pre-existing server costs less than purchasing a new one. It is less likely to add new backup and virtualization software when scaling vertically. Maintenance costs may potentially remain the same too.
- **Less complex process communication:**
  - When a single node handles all the layers of your services, it will not have to synchronize and communicate with other machines to work. This may result in faster responses.
- **Less complicated maintenance:**
  - Not only the maintenance is cheaper but it is less complex because of the number of nodes you will need to manage.
- **Less need for software changes:**
  - It is less likely to change how the software on a server works or how it is implemented.

Vertical autoscaling limitation

- **Downtime Considerations:**
  - Without a backup server capable of handling operations, downtime is inevitable during machine upgrades.
- **Risk of Single Point of Failure:**
  - Potential loss of all data in the event of a hardware or software failure.
- **Upgrade Constraints:**
  - Limited by the maximum capacity of a single resource.



Horizontal vs Vertical autoscaling

	Horizontal scaling	Vertical scaling
Description	Increase or decrease the number of nodes in a cluster or system to handle an increase or decrease in workload	Increase or decrease the power of a system to handle increased or reduced workload
Example	Add or reduce the number of virtual machines (VM) in a cluster of VMs	Add or reduce the CPU or memory capacity of the existing VM
Execution	Scale in/out	Scale up/down
Workload distribution	Workload is distributed across multiple nodes.	A single node handles the entire workload.
Required architecture	Distributed	Any

Horizontal vs Vertical autoscaling

	Horizontal scaling	Vertical scaling
Complexity and maintenance	Higher	Lower
Implementation	Takes more time, expertise, and effort	Takes less time, expertise, and effort
Configuration	This requires modifying a sequential piece of logic in order to run workloads concurrently on multiple machines	No need to change the logic. The same code can run on a higher-spec device
Costs	High costs initially; optimal over time	Low-cost initially; less cost-effective over time
Networking	Slower machine-to-machine communication	inter-machine communication
Performance	Higher	Lower

Horizontal vs Vertical autoscaling

	Horizontal scaling	Vertical scaling
Limitation	Add as many machines as you can	Limited to the resource capacity the single machine can handle

Autoscaling in Kubernetes

Autoscaling in Kubernetes

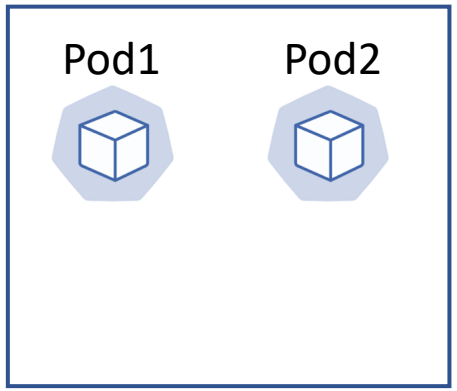
Kubernetes provides three types of autoscaling:

Autoscaling in Kubernetes

Kubernetes provides three types of autoscaling:

- Horizontal pod autoscaling (HPA):

  Automatically adjusts the number of pod replicas in a deployment or replica set based on observed metrics, such as CPU utilization or custom metrics.

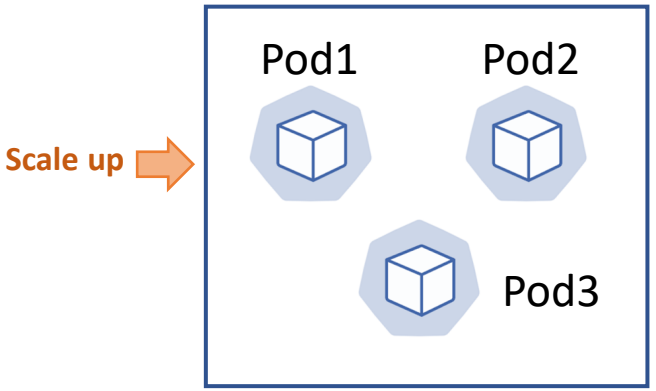


Autoscaling in Kubernetes

Kubernetes provides three types of autoscaling:

- Horizontal pod autoscaling (HPA):

  Automatically adjusts the number of pod replicas in a deployment or replica set based on observed metrics, such as CPU utilization or custom metrics.

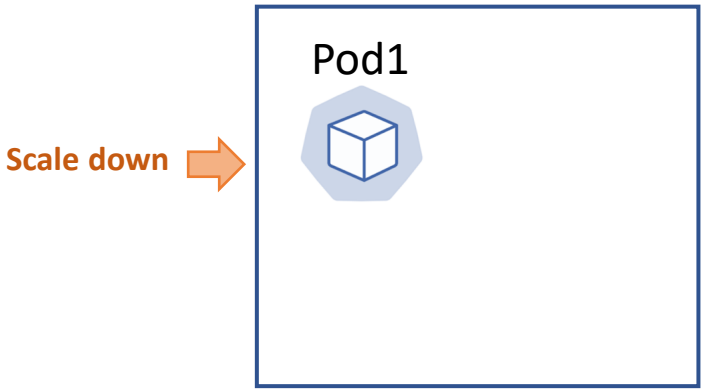


Autoscaling in Kubernetes

Kubernetes provides three types of autoscaling:

- Horizontal pod autoscaling (HPA):

  Automatically adjusts the number of pod replicas in a deployment or replica set based on observed metrics, such as CPU utilization or custom metrics.



Autoscaling in Kubernetes

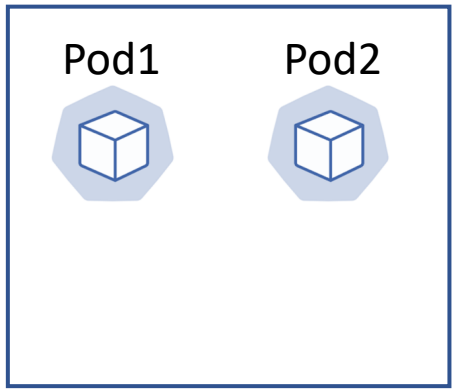
Kubernetes provides three types of autoscaling:

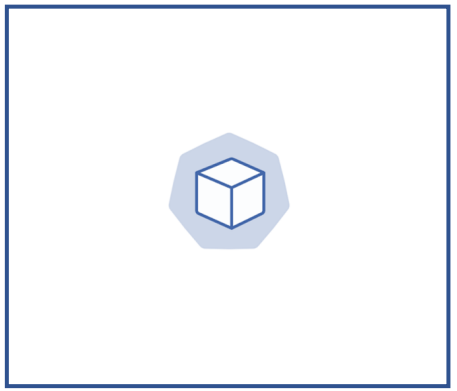
- Horizontal pod autoscaling (HPA):

  Automatically adjusts the number of pod replicas in a deployment or replica set based on observed metrics, such as CPU utilization or custom metrics.

- Vertical pod autoscaling (VPA)

  Automatically adjusts the resource requests (CPU and memory) of individual containers within a pod based on observed usage.





Autoscaling in Kubernetes

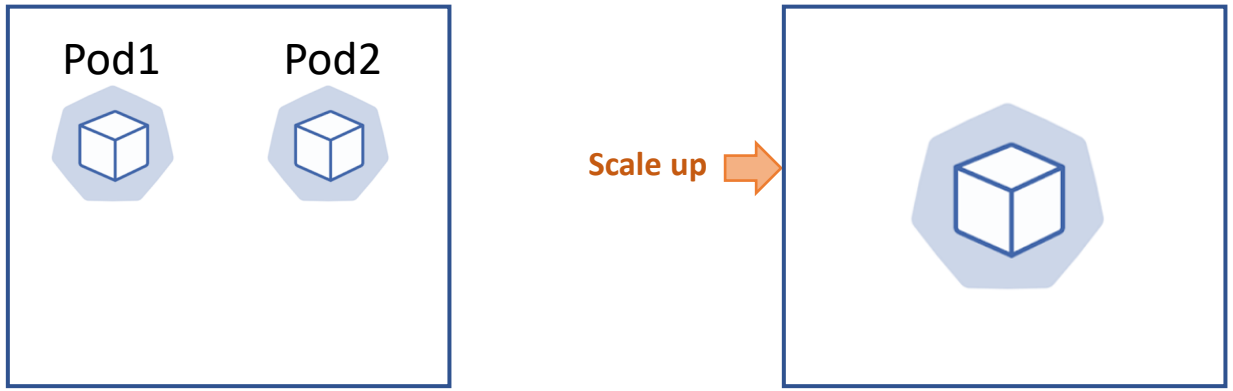
Kubernetes provides three types of autoscaling:

- Horizontal pod autoscaling (HPA):

  Automatically adjusts the number of pod replicas in a deployment or replica set based on observed metrics, such as CPU utilization or custom metrics.

- Vertical pod autoscaling (VPA)

  Automatically adjusts the resource requests (CPU and memory) of individual containers within a pod based on observed usage.



Autoscaling in Kubernetes

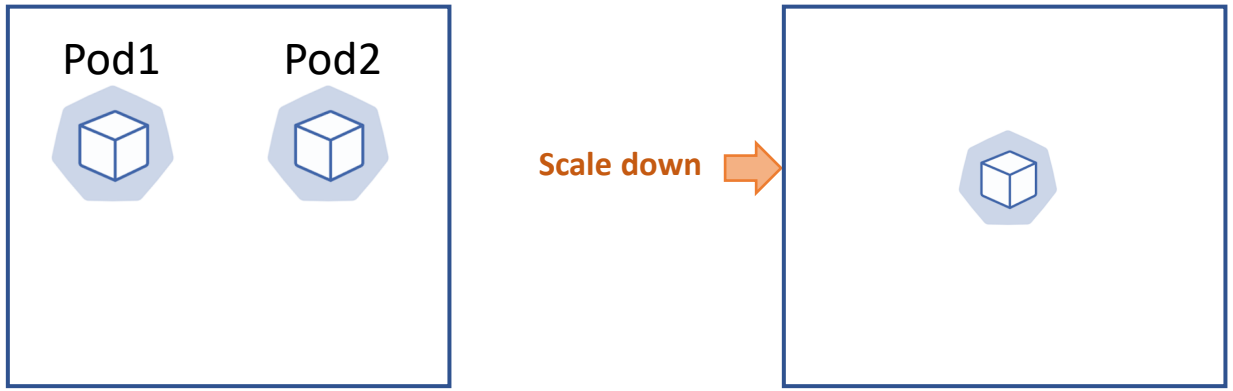
Kubernetes provides three types of autoscaling:

- Horizontal pod autoscaling (HPA):

  Automatically adjusts the number of pod replicas in a deployment or replica set based on observed metrics, such as CPU utilization or custom metrics.

- Vertical pod autoscaling (VPA)

  Automatically adjusts the resource requests (CPU and memory) of individual containers within a pod based on observed usage.

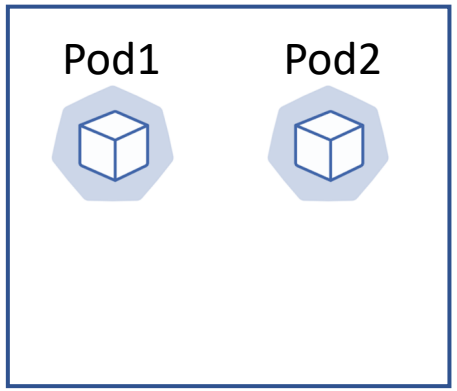


Autoscaling in Kubernetes

Kubernetes provides three types of autoscaling:

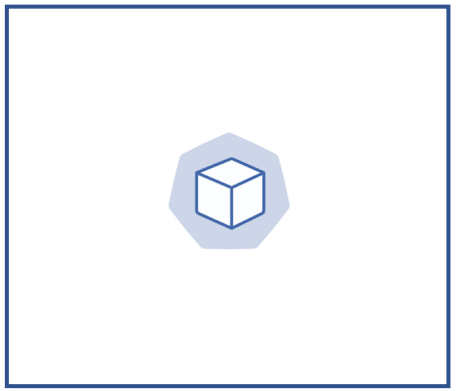
- Horizontal pod autoscaling (HPA):

  Automatically adjusts the number of pod replicas in a deployment or replica set based on observed metrics, such as CPU utilization or custom metrics.



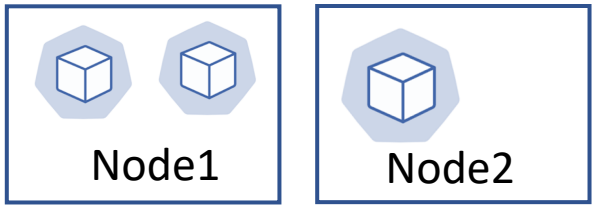
- Vertical pod autoscaling (VPA)

  Automatically adjusts the resource requests (CPU and memory) of individual containers within a pod based on observed usage.



- Cluster autoscaling (CA):

  Automatically adjust the number of nodes based on the resource requirements and constraints of the workloads running in the cluster.

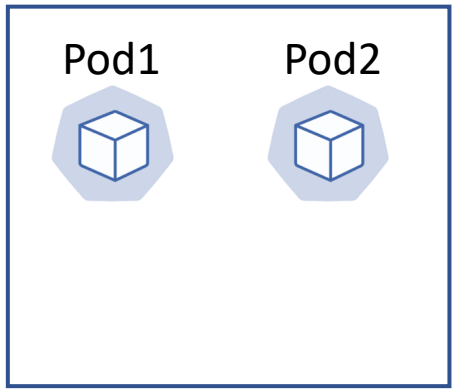


Autoscaling in Kubernetes

Kubernetes provides three types of autoscaling:

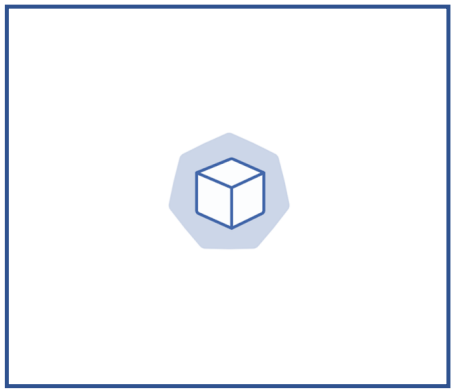
- Horizontal pod autoscaling (HPA):

  Automatically adjusts the number of pod replicas in a deployment or replica set based on observed metrics, such as CPU utilization or custom metrics.



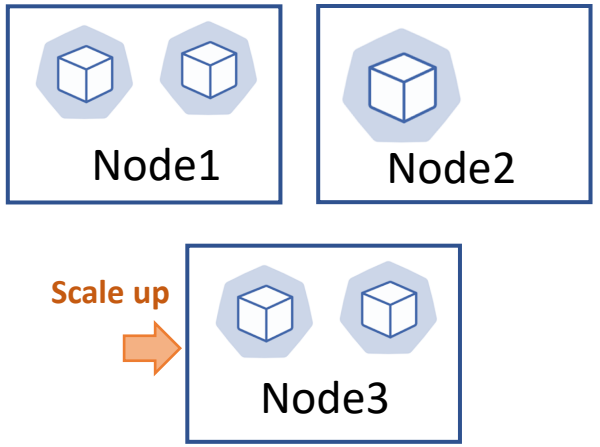
- Vertical pod autoscaling (VPA)

  Automatically adjusts the resource requests (CPU and memory) of individual containers within a pod based on observed usage.



- Cluster autoscaling (CA):

  Automatically adjust the number of nodes based on the resource requirements and constraints of the workloads running in the cluster.

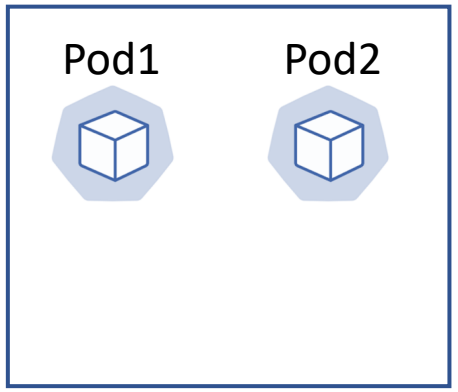


Autoscaling in Kubernetes

Kubernetes provides three types of autoscaling:

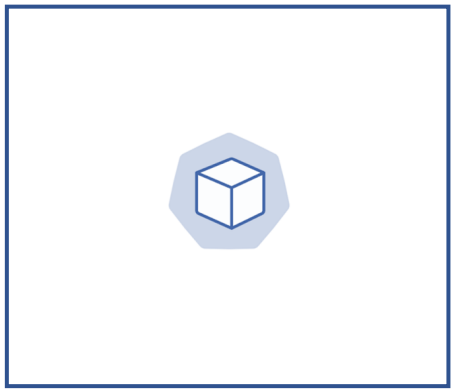
- Horizontal pod autoscaling (HPA):

  Automatically adjusts the number of pod replicas in a deployment or replica set based on observed metrics, such as CPU utilization or custom metrics.



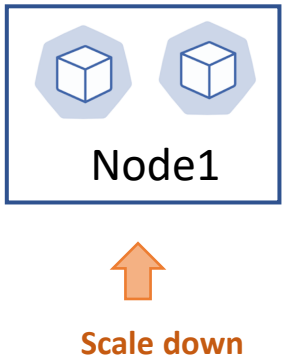
- Vertical pod autoscaling (VPA)

  Automatically adjusts the resource requests (CPU and memory) of individual containers within a pod based on observed usage.



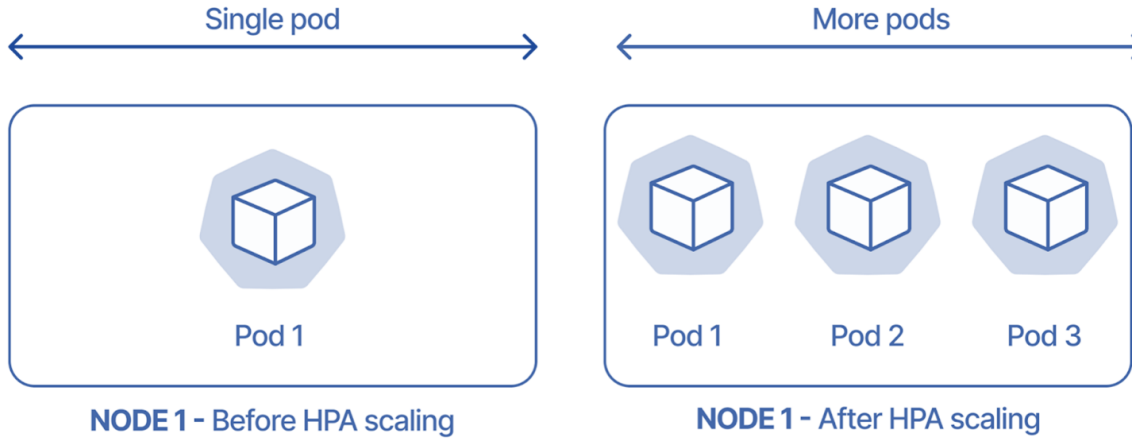
- Cluster autoscaling (CA):

  Automatically adjust the number of nodes based on the resource requirements and constraints of the workloads running in the cluster.



Horizontal Pod Autoscaling

- Kubernetes monitors the specified metrics for each pod in the target deployment.
- If the observed metrics go beyond or fall below the defined thresholds, the HPA controller triggers scaling actions.
- Scaling actions involve adjusting the number of pod replicas to meet the desired metric targets.



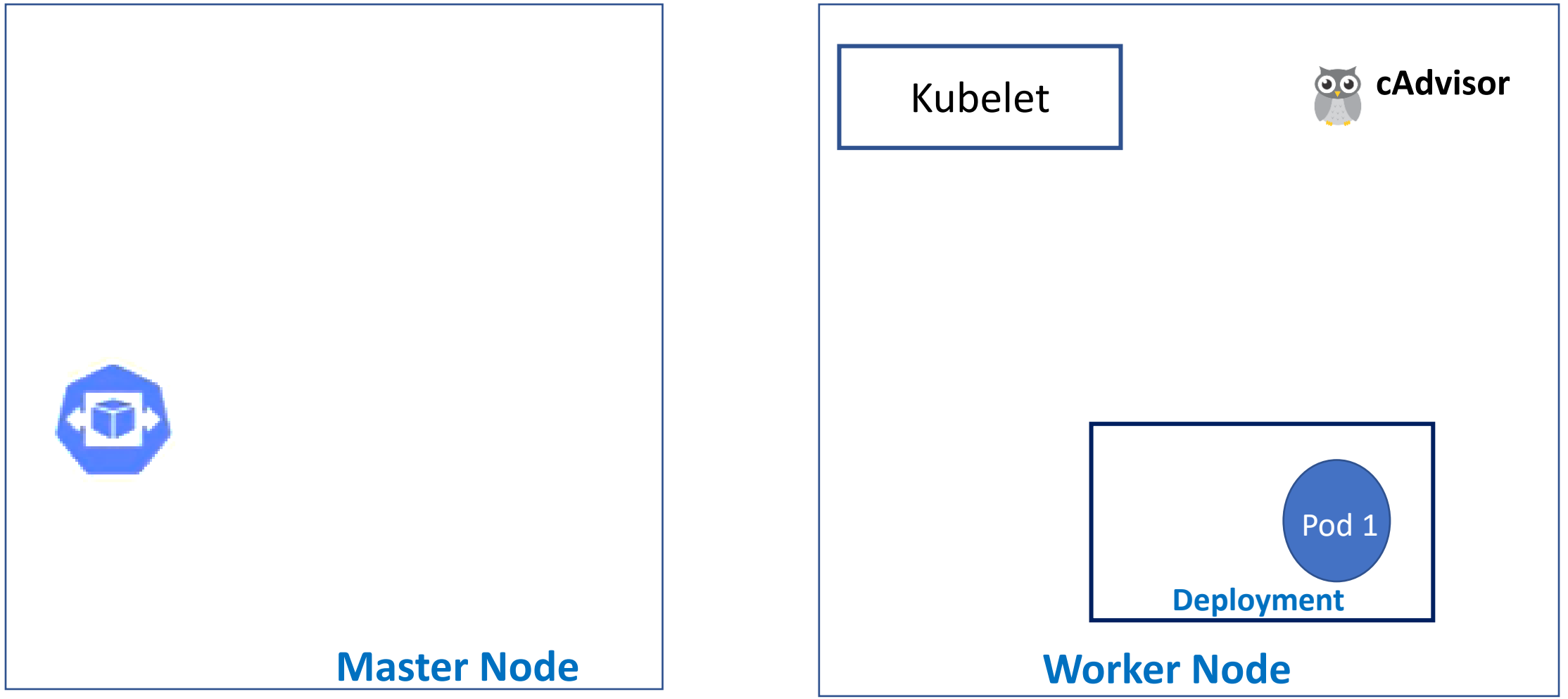
Horizontal Pod Autoscaling

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: utility-api
spec:
  minReplicas: 1
  maxReplicas: 5
  metrics:
    - resource:
        name: cpu
        target:
          averageUtilization: 70
          type: Utilization
      type: Resource
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: utility-api
```

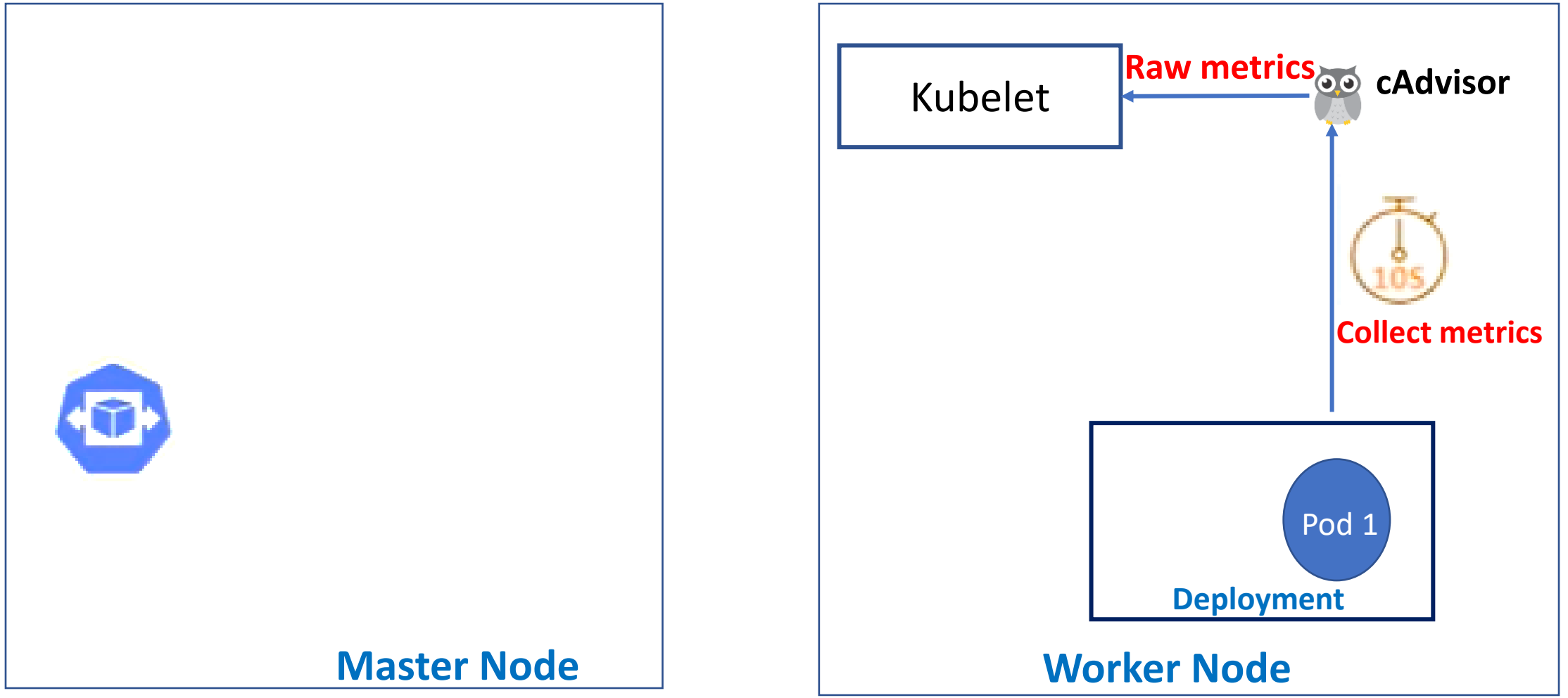
Master Node

Worker Node

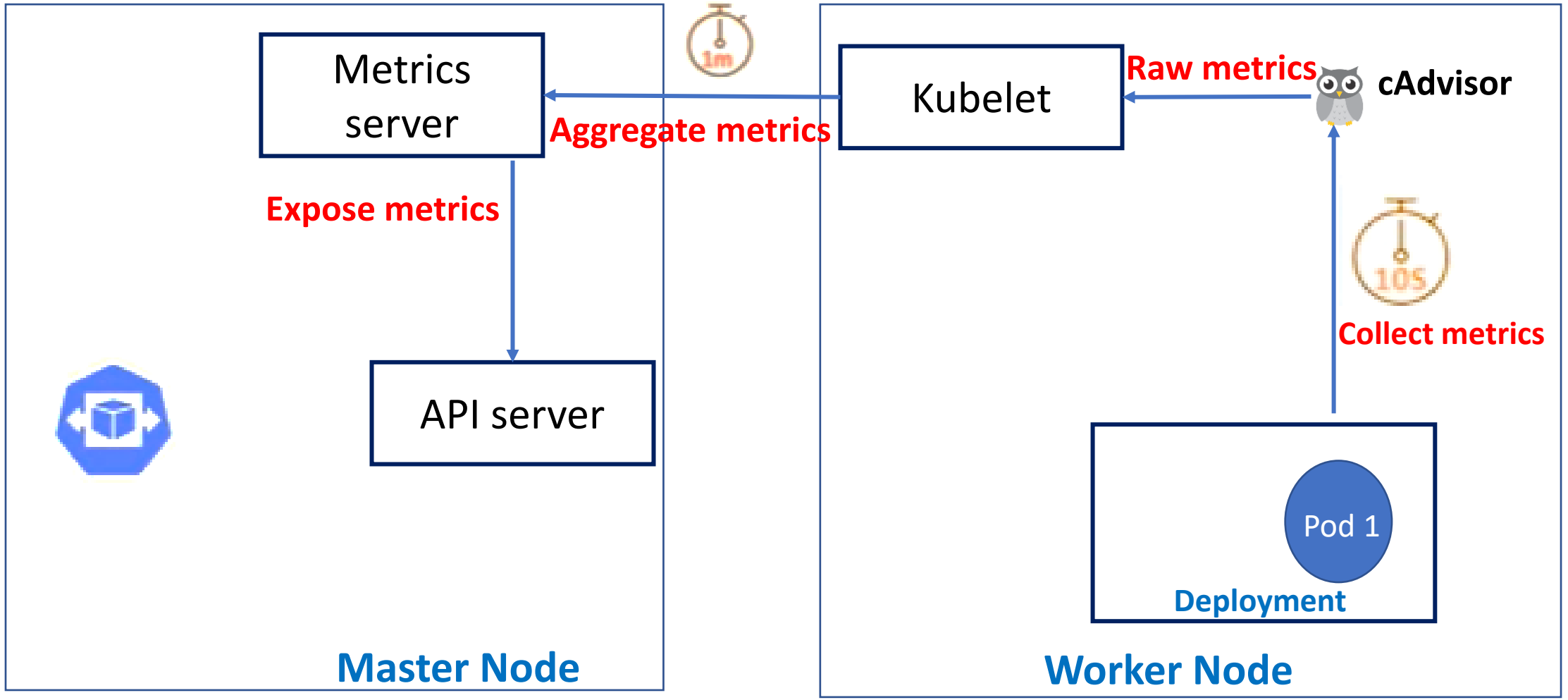
Horizontal Pod Autoscaling



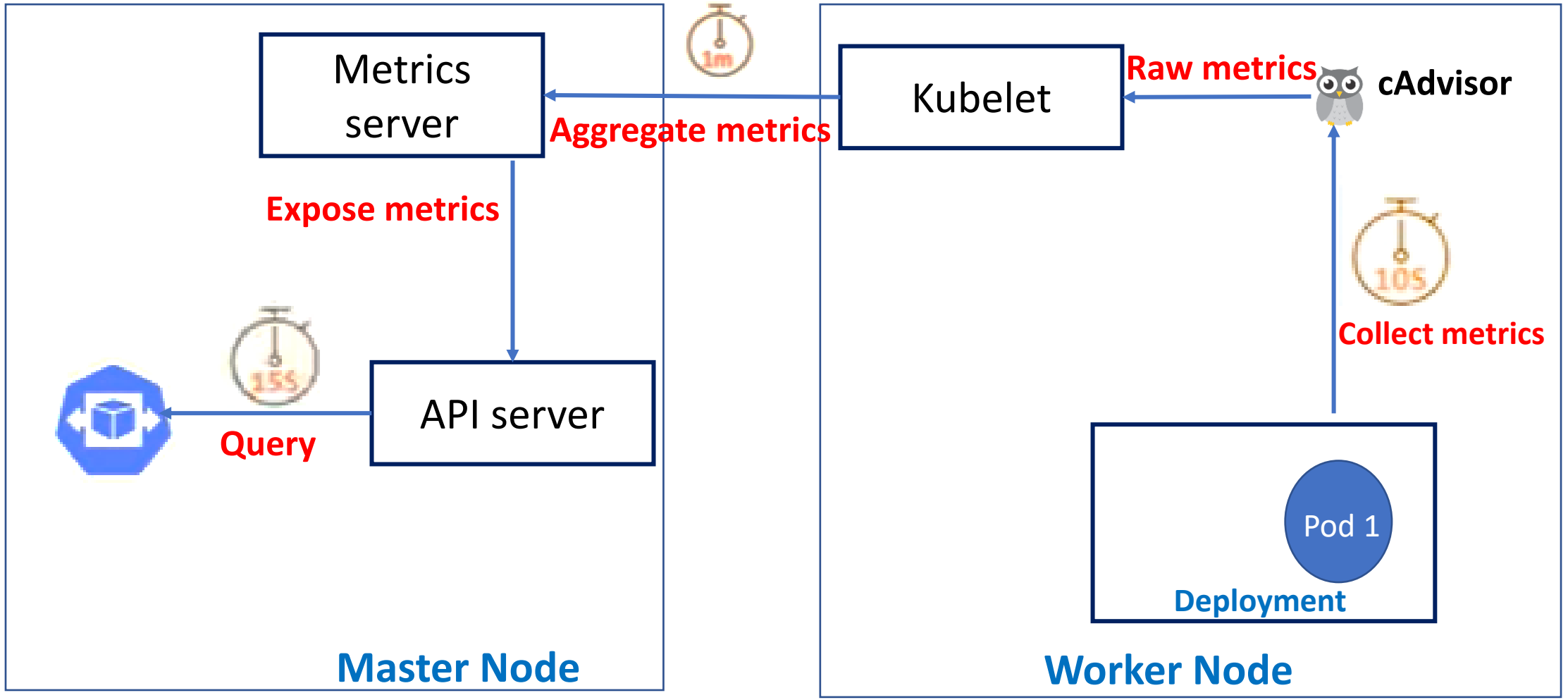
Horizontal Pod Autoscaling



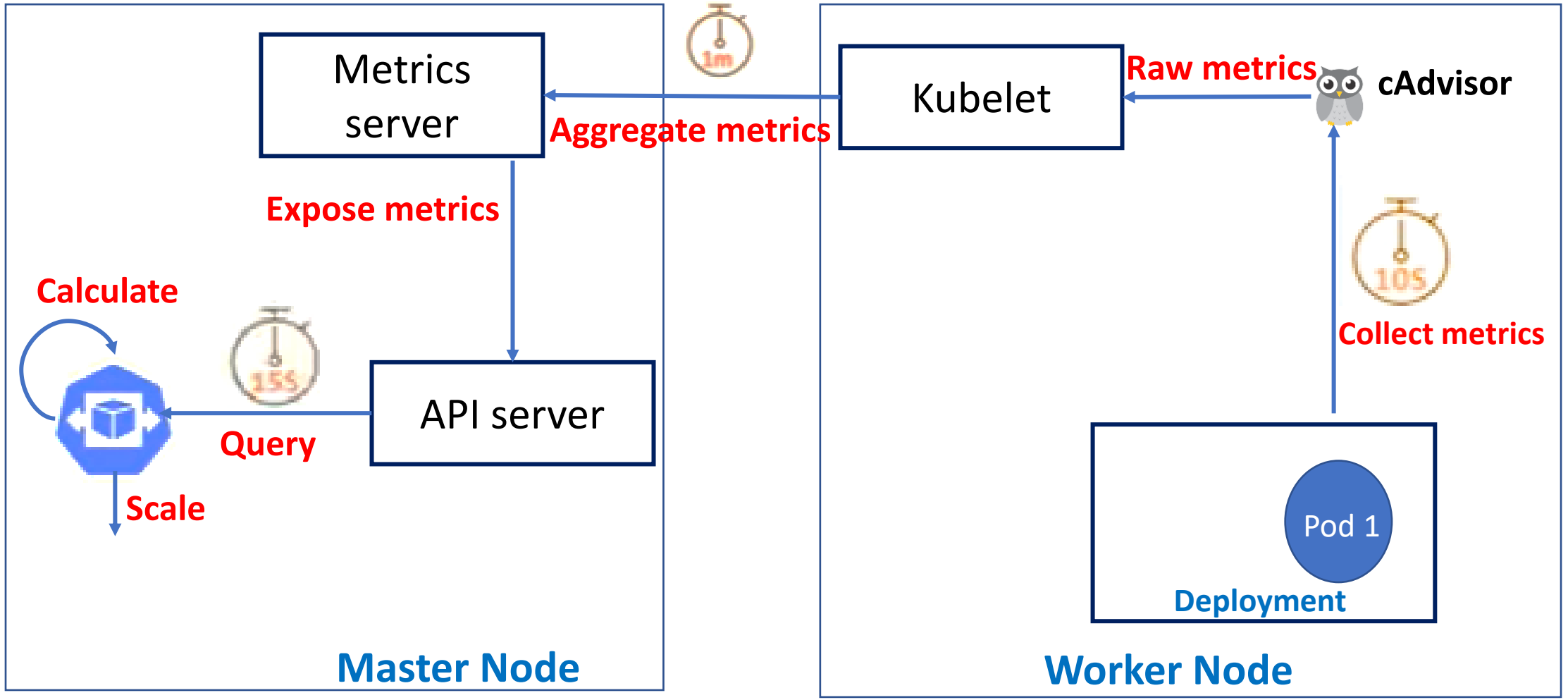
Horizontal Pod Autoscaling



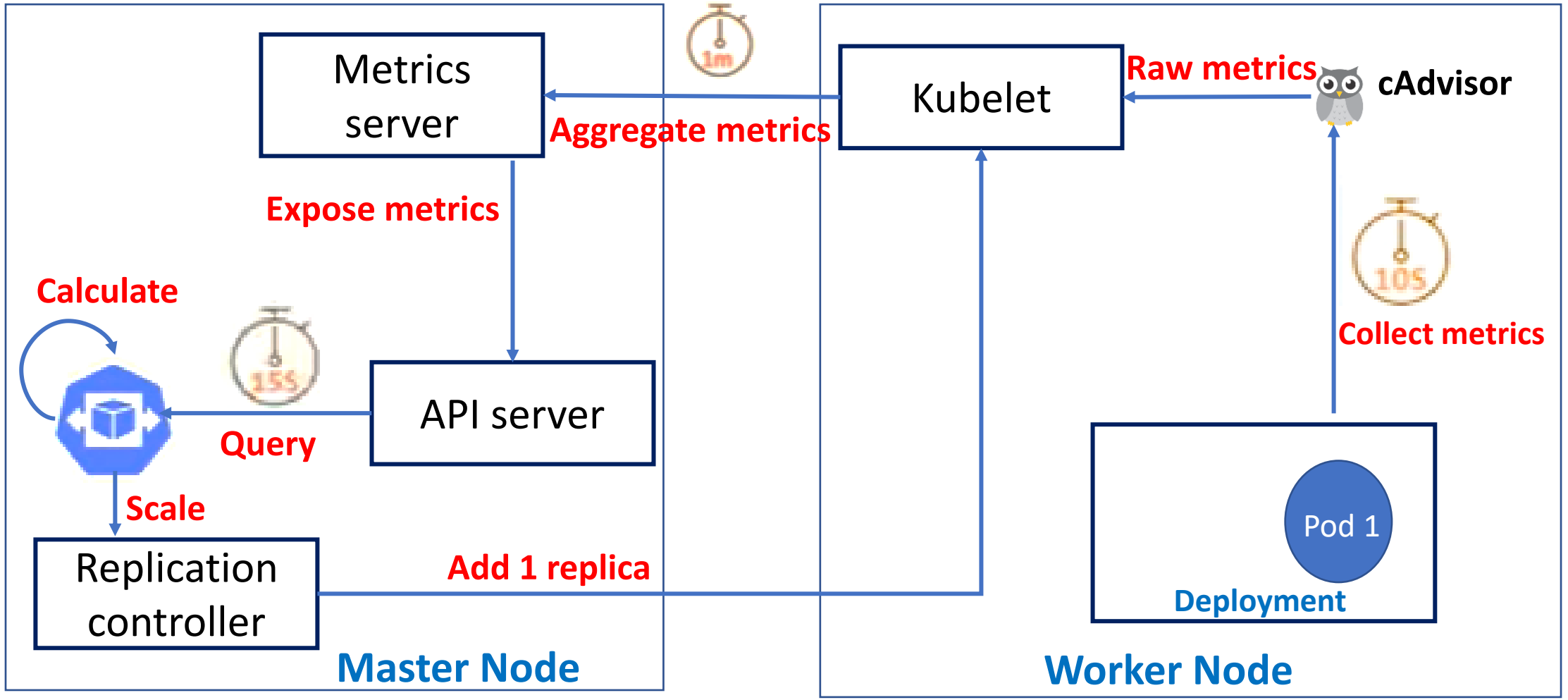
Horizontal Pod Autoscaling



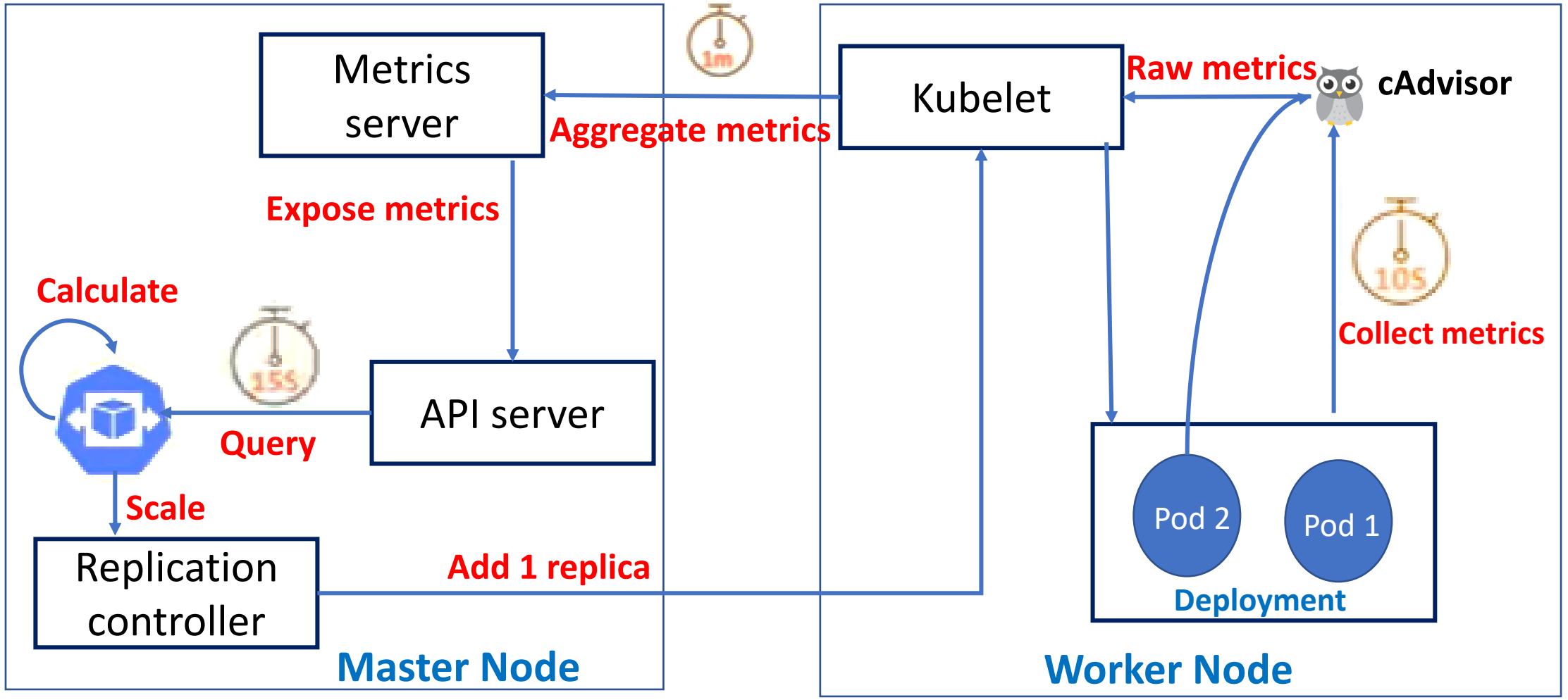
Horizontal Pod Autoscaling



Horizontal Pod Autoscaling



Horizontal Pod Autoscaling



Horizontal Pod Autoscaling

Calculating the desired number of replicas:

$$\text{desiredReplicas} = \left\lceil \text{currentReplicas} \cdot \frac{\text{currentMetricValue}}{\text{desiredMetricValue}} \right\rceil$$

Horizontal Pod Autoscaling

Calculating the desired number of replicas:

$$\text{desiredReplicas} = \left\lceil \text{currentReplicas} \cdot \frac{\text{currentMetricValue}}{\text{desiredMetricValue}} \right\rceil$$

Example:

CPU: $\text{desiredReplicas} = \left\lceil 2 \cdot \frac{90}{70} \right\rceil = 3 \text{ Replica}$

Memory: $\text{desiredReplicas} = \left\lceil 2 \cdot \frac{1000}{500} \right\rceil = 4 \text{ Replica}$

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: utility-api
spec:
  minReplicas: 1
  maxReplicas: 5
  metrics:
    - resource:
        name: cpu
        target:
          averageUtilization: 70
          type: Utilization
      type: Resource
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: utility-api
```

Horizontal Pod Autoscaling

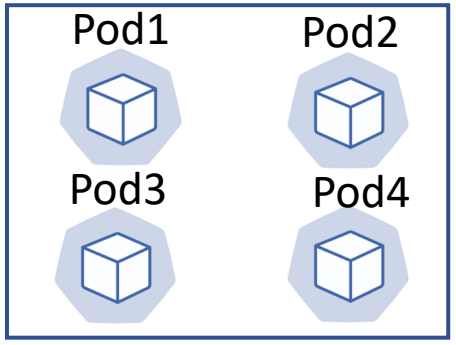
Calculating the desired number of replicas:

$$\text{desiredReplicas} = \left\lceil \text{currentReplicas} \cdot \frac{\text{currentMetricValue}}{\text{desiredMetricValue}} \right\rceil$$

Example:

CPU: $\text{desiredReplicas} = \left\lceil 2 \cdot \frac{90}{70} \right\rceil = 3 \text{ Replica}$

Memory: $\text{desiredReplicas} = \left\lceil 2 \cdot \frac{1000}{500} \right\rceil = 4 \text{ Replica}$




```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: utility-api
spec:
  minReplicas: 1
  maxReplicas: 5
  metrics:
    - resource:
        name: cpu
        target:
          averageUtilization: 70
          type: Utilization
      type: Resource
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: utility-api
```

Horizontal Pod Autoscaling

Creating a deployment:

deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: utility-api
spec:
  replicas: 2
  selector:
    matchLabels:
      app: utility-api
  template:
    metadata:
      name: utility-api-pod
      labels:
        app: utility-api
    spec:
      containers:
        - name: utility-api
          image: pavanelthepu/utility-api
          ports:
            - containerPort: 8080
          resources:
            requests:
              memory: 20Mi
```

Creating a service to access the deployment:

deployment.yaml | service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: utility-api-service
spec:
  selector:
    app: utility-api
  ports:
    - port: 8080
      targetPort: 8080
```

Horizontal Pod Autoscaling

Generating HPA for the deployment:

Horizontal Pod Autoscaling

Automatically scaling down the number of pod replicas:

```
avan.elthepu@rrcs-172-254-255-63 autoscaling % kubectl get pods
AME                                READY   STATUS    RESTARTS   AGE
tility-api-6cd64bbb76-4vbxr   1/1     Running   0          3m16s
tility-api-6cd64bbb76-8vmj7   1/1     Running   0          3m16s
avan.elthepu@rrcs-172-254-255-63 autoscaling % kubectl top pods
AME                                CPU(cores)   MEMORY(bytes)
tility-api-6cd64bbb76-4vbxr   2m           89Mi
avan.elthepu@rrcs-172-254-255-63 autoscaling % kubectl get pods
AME                                READY   STATUS    RESTARTS   AGE
tility-api-6cd64bbb76-4vbxr   1/1     Running   0          7m28s
avan.elthepu@rrcs-172-254-255-63 autoscaling % kubectl get hpa
AME           REFERENCE                     TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
tility-api   Deployment/utility-api   1%/70%    1         5         1          9m25s
avan.elthepu@rrcs-172-254-255-63 autoscaling %
```

Number of replicas before autoscaling

Horizontal Pod Autoscaling

Automatically scaling down the number of pod replicas:

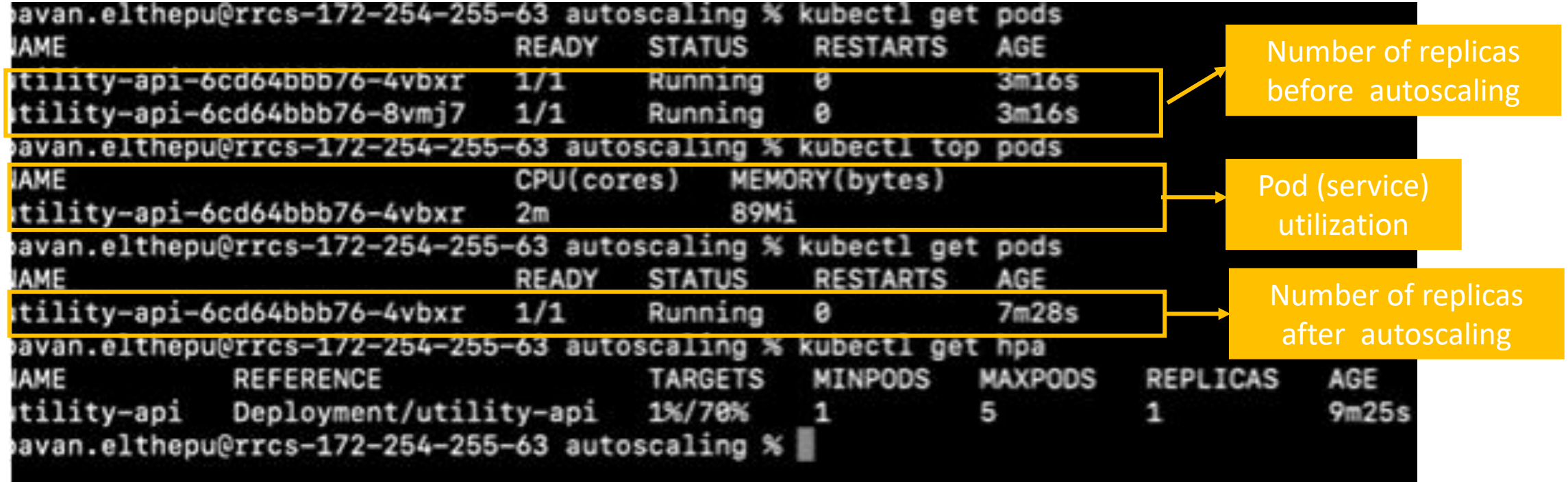
```
pavan.elthepu@rrcs-172-254-255-63 autoscaling % kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
utility-api-6cd64bbb76-4vbxr        1/1     Running   0          3m16s
utility-api-6cd64bbb76-8vmj7        1/1     Running   0          3m16s
pavan.elthepu@rrcs-172-254-255-63 autoscaling % kubectl top pods
NAME                                CPU(cores)   MEMORY(bytes)
utility-api-6cd64bbb76-4vbxr        2m           89Mi
pavan.elthepu@rrcs-172-254-255-63 autoscaling % kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
utility-api-6cd64bbb76-4vbxr        1/1     Running   0          7m28s
pavan.elthepu@rrcs-172-254-255-63 autoscaling % kubectl get hpa
NAME          REFERENCE                      TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
utility-api   Deployment/utility-api         1%/70%    1         5         1          9m25s
pavan.elthepu@rrcs-172-254-255-63 autoscaling %
```

Number of replicas before autoscaling

Pod (service) utilization

Horizontal Pod Autoscaling

Automatically scaling down the number of pod replicas:



Horizontal Pod Autoscaling

```
avan.elthepu@rrcs-172-254-255-63 autoscaling % kubectl describe hpa utility-api
arning: autoscaling/v2beta2 HorizontalPodAutoscaler is deprecated in v1.23+, unavailable in v1.26+; use autoscaling/v2 Horizonta
caler
ame:                                                  utility-api
amespace:                                             default
abels:                                                <none>
nnotations:                                           <none>
reationTimestamp:                                     Sat, 18 Feb 2023 19:47:12 +0530
eference:                                             Deployment/utility-api
etrics:                                               ( current / target )
 resource cpu on pods  (as a percentage of request):  0% (2m) / 70%
in replicas:                                          1
ax replicas:                                          5
eployment pods:                                       1 current / 1 desired
onditions:
 Type            Status  Reason              Message
 ----            ------  ------              -------
 AbleToScale     True    ScaleDownStabilized  recent recommendations were higher than current one, applying the highest recent r
tion
 ScalingActive   True    ValidMetricFound     the HPA was able to successfully calculate a replica count from cpu resource utili
ercentage of request)
 ScalingLimited  False   DesiredWithinRange   the desired count is within the acceptable range
vents:
 Type     Reason                        Age                    From                       Message
 ----     ------                        ----                   ----                       -------
 Warning  FailedComputeMetricsReplicas  7m12s (x12 over 9m57s)  horizontal-pod-autoscaler  invalid metrics (1 invalid out of 1),
ror is: failed to get cpu resource metric value: failed to get cpu utilization: unable to get metrics for resource cpu: unable t
etrics from resource metrics API: the server could not find the requested resource (get pods.metrics.k8s.io)
 Warning  FailedGetResourceMetric       6m57s (x13 over 9m57s)  horizontal-pod-autoscaler  failed to get cpu utilization: unable
etrics for resource cpu: unable to fetch metrics from resource metrics API: the server could not find the requested resource (ge
trics.k8s.io)
 Normal   SuccessfulRescale             4m57s                  horizontal-pod-autoscaler  New size: 1; reason: All metrics belo
avan.elthepu@rrcs-172-254-255-63 autoscaling %
```

Horizontal Pod Autoscaling

Increasing the resource utilization of pod(service):

1. Generating workload stress:

```java
@RestController
@RequestMapping("/api")
@CrossOrigin
public class UtilityController {

    private static final Logger logger = LoggerFactory.getLogger(clazz: UtilityContr

    @GetMapping("/stress")
    public int generateStress() {
        logger.info(msg: "Generating stress..");
        int counter = 0;
        for(int i = 0; i <= 5000000; i++) {
            counter += i;
        }
        return counter;
    }
}
```

Create load over the CPU

Horizontal Pod Autoscaling

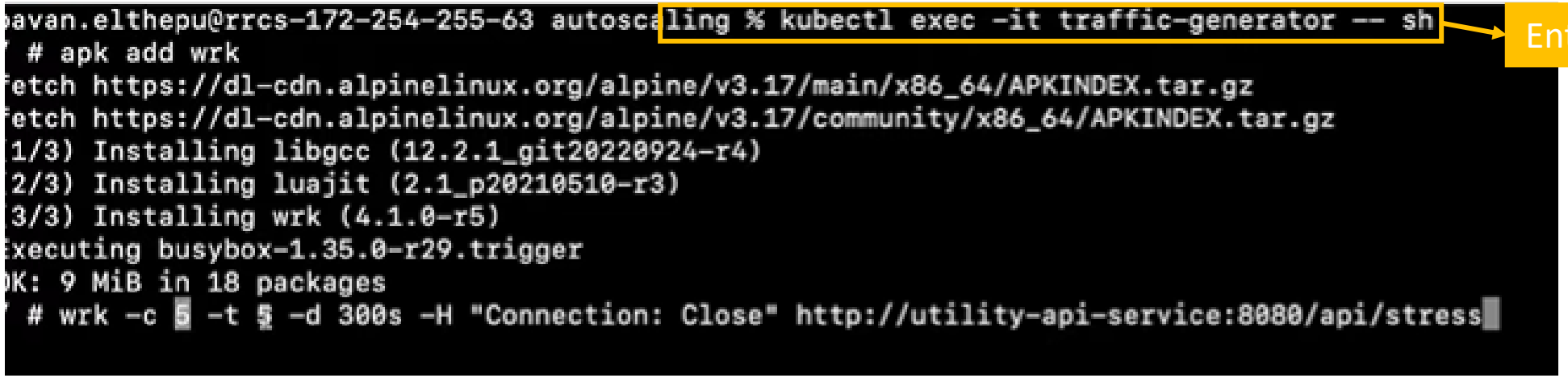
Increasing the resource utilization of pod(service):

2. Creating alpine pod to communicate with utility api:

Horizontal Pod Autoscaling

Increasing the resource utilization of pod(service):

3. Generate load on utility/ api pod so that the CPU utilization goes beyond 70% and we can see how pod scale up automatically. For that call stress api continuously so that the CPU usage goes up with using a http benchmarking tool called wrk*.



*https://github.com/wg/wrk

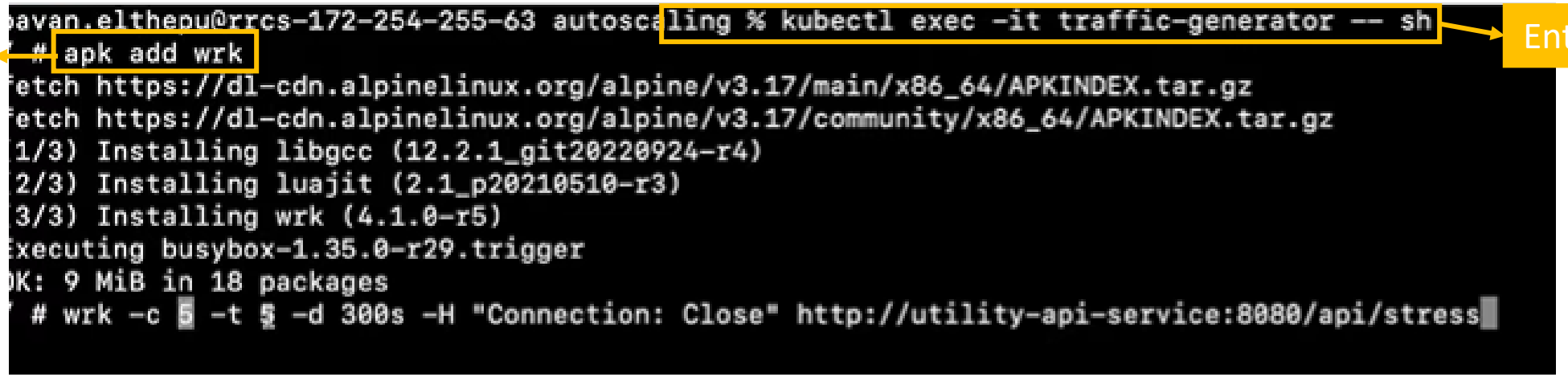
Horizontal Pod Autoscaling

Increasing the resource utilization of pod(service):

3. Generate load on utility/ api pod so that the CPU utilization goes beyond 70% and we can see how pod scale up automatically. For that call stress api continuously so that the CPU usage goes up with using a http benchmarking tool called wrk*.

Install wrk

Enter to the pod

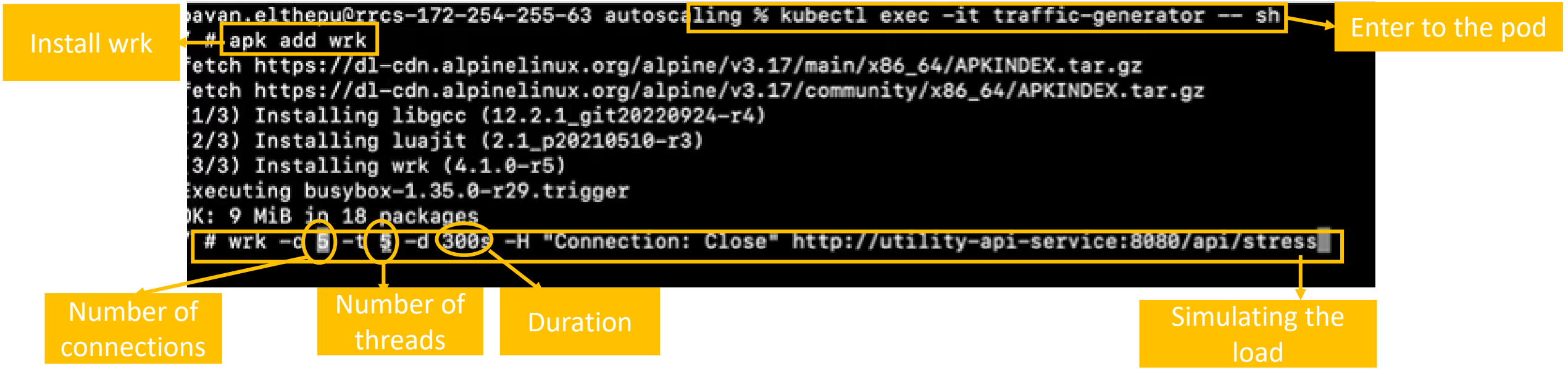


*https://github.com/wg/wrk

Horizontal Pod Autoscaling

Increasing the resource utilization of pod(service):

3. Generate load on utility/ api pod so that the CPU utilization goes beyond 70% and we can see how pod scale up automatically. For that call stress api continuously so that the CPU usage goes up with using a http benchmarking tool called wrk*.



*https://github.com/wg/wrk

Horizontal Pod Autoscaling

Automatically scaling up the number of pod replicas:

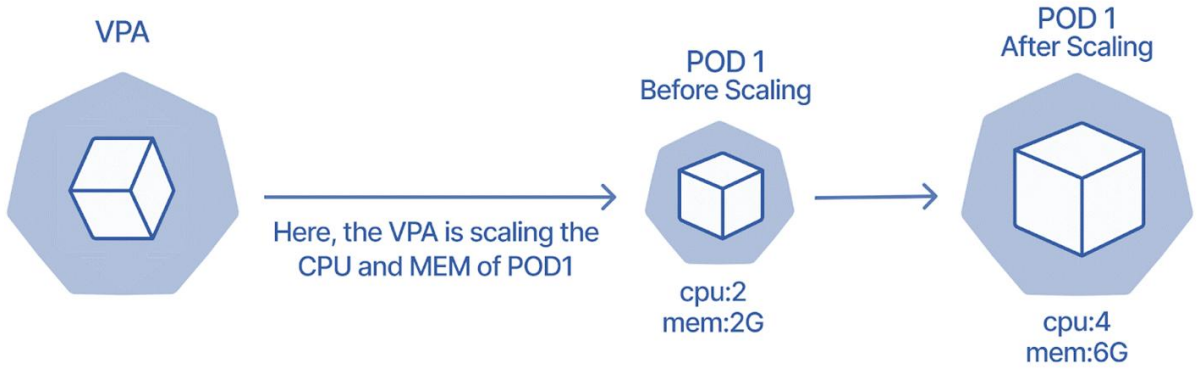
```
pavan.elthepu@Pavan-Elthepu autoscaling % kubectl top pods
NAME                              CPU(cores)   MEMORY(bytes)
traffic-generator                 1m           6Mi
utility-api-6cd64bbb76-4vbxr      43m          102Mi
pavan.elthepu@Pavan-Elthepu autoscaling % kubectl top pods
NAME                              CPU(cores)   MEMORY(bytes)
traffic-generator                 57m          7Mi
utility-api-6cd64bbb76-4vbxr      301m         122Mi
utility-api-6cd64bbb76-snxfj      388m         106Mi
utility-api-6cd64bbb76-tk8v8      342m         104Mi
utility-api-6cd64bbb76-z9ct6      339m         104Mi
utility-api-6cd64bbb76-zmw6g      469m         108Mi
pavan.elthepu@Pavan-Elthepu autoscaling %
```

Horizontal Pod Autoscaling

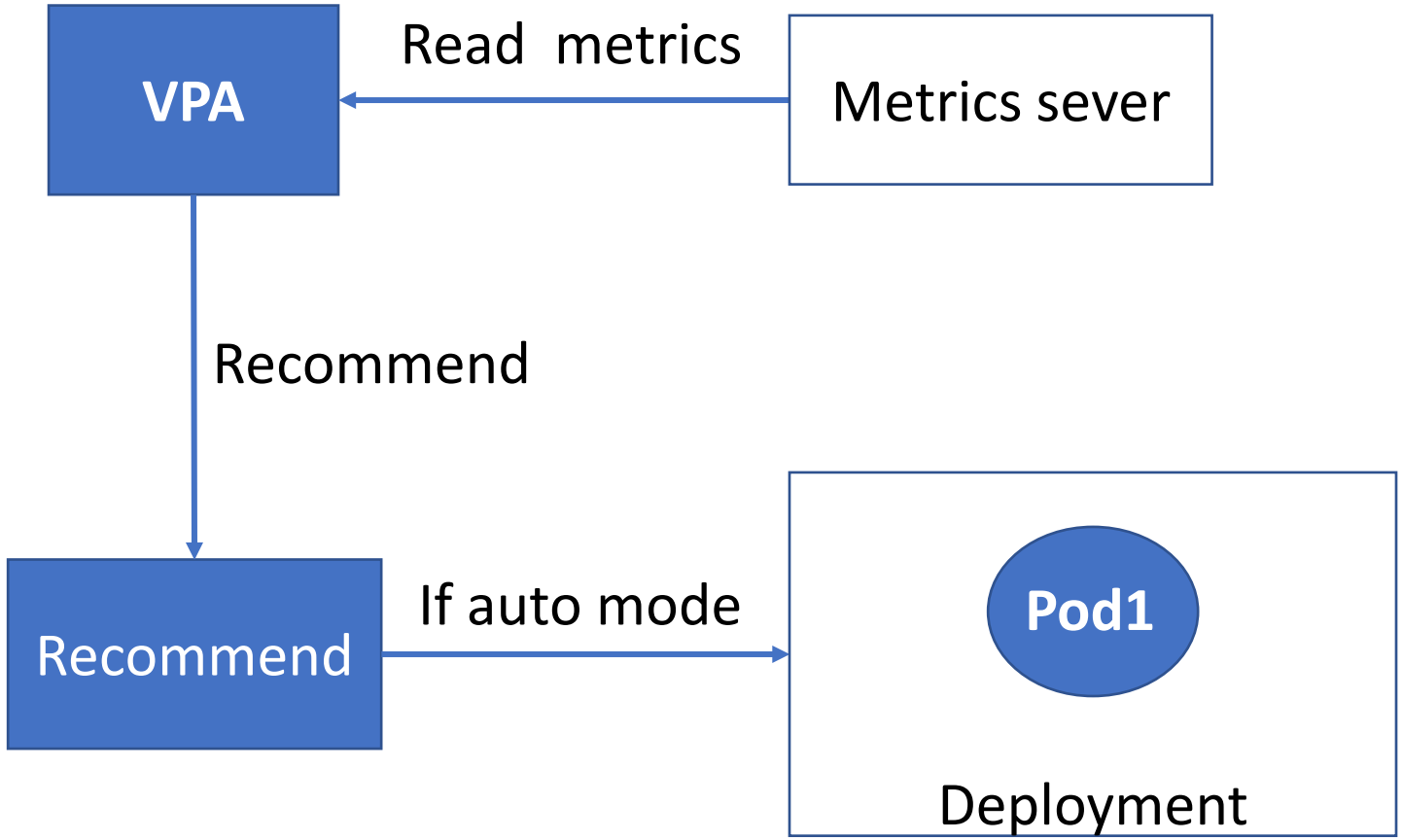
```
pavan.elthepu@Pavan-Elthepu autoscaling % kubectl describe hpa utility-api
Warning: autoscaling/v2beta2 HorizontalPodAutoscaler is deprecated in v1.23+, unavailable in v1.26+; use autoscaling/v2 HorizontalPo
scaler
Name:                                                  utility-api
Namespace:                                             default
Labels:                                                <none>
Annotations:                                           <none>
CreationTimestamp:                                     Sat, 18 Feb 2023 19:47:12 +0530
Reference:                                             Deployment/utility-api
Metrics:                                               ( current / target )
  resource cpu on pods  (as a percentage of request):  117% (294m) / 70%
Min replicas:                                          1
Max replicas:                                          5
Deployment pods:                                       5 current / 5 desired
Conditions:
  Type            Status  Reason            Message
  ----            ------  ------            -------
  AbleToScale     True    ReadyForNewScale  recommended size matches current size
  ScalingActive   True    ValidMetricFound  the HPA was able to successfully calculate a replica count from cpu resource utilizatio
centage of request)
  ScalingLimited  True    TooManyReplicas   the desired replica count is more than the maximum replica count
Events:
  Type     Reason                        Age                From                       Message
  ----     ------                        ----               ----                       -------
  Warning  FailedComputeMetricsReplicas  20m (x12 over 23m)  horizontal-pod-autoscaler  invalid metrics (1 invalid out of 1), first
is: failed to get cpu resource metric value: failed to get cpu utilization: unable to get metrics for resource cpu: unable to fetch
ics from resource metrics API: the server could not find the requested resource (get pods.metrics.k8s.io)
  Warning  FailedGetResourceMetric       20m (x13 over 23m)  horizontal-pod-autoscaler  failed to get cpu utilization: unable to get
ics for resource cpu: unable to fetch metrics from resource metrics API: the server could not find the requested resource (get pods.
s.k8s.io)
  Normal   SuccessfulRescale             18m                horizontal-pod-autoscaler  New size: 1; reason: All metrics below targe
  Normal   SuccessfulRescale             3m23s              horizontal-pod-autoscaler  New size: 4; reason: cpu resource utilizatio
centage of request) above target
  Normal   SuccessfulRescale             3m8s               horizontal-pod-autoscaler  New size: 5; reason: cpu resource utilizatio
```

Vertical Pod Autoscaling

- The VPA consists of three components:
  - **Admission controller:** Triggers when new pods are created and checks if it is covered by a VPA.
  - **Recommender:** Provides advice based on historical and live resource usage metrics.
  - **Updater**: Decides which pods should be restarted based on resources allocation recommendation calculated by the Recommender.



Vertical Pod Autoscaling



Horizontal Pod Autoscaling

autoscaling > ! vpa.yaml > {} spec > {} targetRef

```yaml
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: utility-api
spec:
  targetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: utility-api
  updatePolicy:
    updateMode: "Off"
```

Auto
Off
Initial

Auto: It applies the recommendations that suggested by vertical autoscaler directly by updating the pods.

Off: It just gives you the recommendations but it does not update the replicas

Initial: It applies the recommended values only to the new created pod

Update mode: “off” is more recommended in productions because when vertical pod autoscaler applies the changes the pod will be restarted which might cause the workload disruption.

Vertical Pod Autoscaling

```
Spec:
  Target Ref:
    API Version:  apps/v1
    Kind:         Deployment
    Name:         utility-api
  Update Policy:
    Update Mode:  Off
Status:
  Conditions:
    Last Transition Time:  2023-02-18T14:59:03Z
    Status:                True
    Type:                  RecommendationProvided
  Recommendation:
    Container Recommendations:
      Container Name:  utility-api
      Lower Bound:
        Cpu:     25m
        Memory:  262144k
      Target:
        Cpu:     25m
        Memory:  262144k
      Uncapped Target:
        Cpu:     25m
        Memory:  262144k
      Upper Bound:
        Cpu:     100G
        Memory:  100T
```

Current usage

Usage limit

Vertical Pod Autoscaling

- Install the vertical pod autoscaling

```
git clone http://github.com/Kubernetes/autoscaler.git
cd autoscaler
./vertical-pod-autoscaler/hack/vpa-up.sh
```

- Update the vertical pod autoscaler

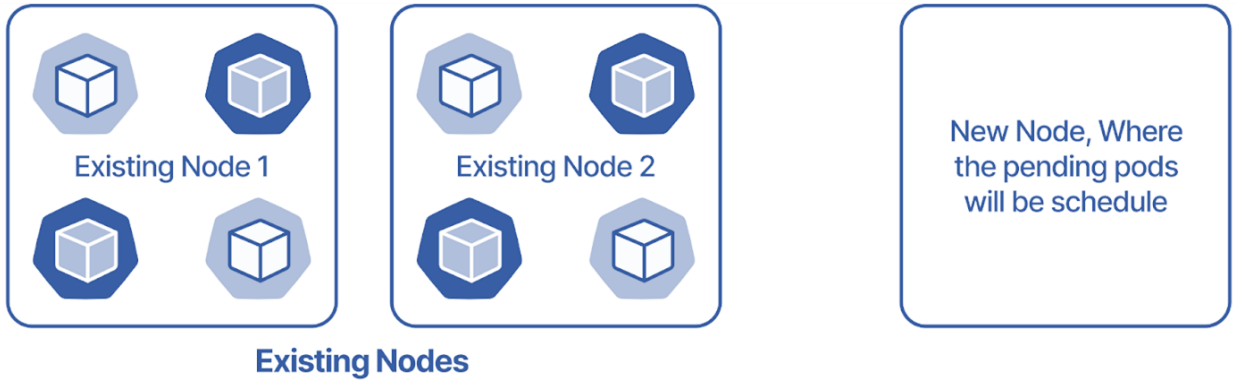
```
Kubectl apply -f vpa.yaml
```

- Verify vertical pod autoscaler

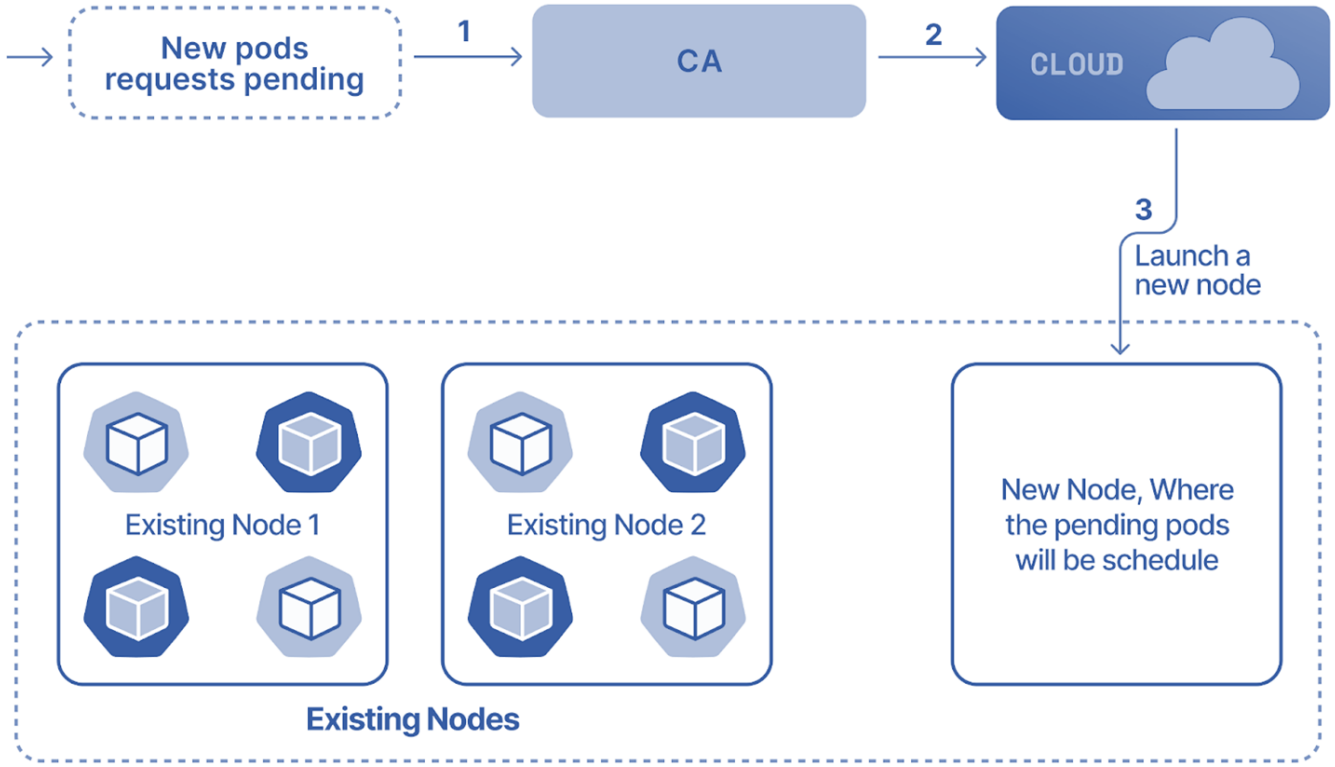
```
pavan.elthepu@Pavan-Elthepu autoscaling % kubectl get vpa utility-api
NAME          MODE   CPU   MEM       PROVIDED   AGE
utility-api   Off    25m   262144k   True       74s
pavan.elthepu@Pavan-Elthepu autoscaling % kubectl describe vpa utility-api
```

Cluster Autoscaling

- The Cluster Autoscaler automatically adds or removes nodes in a cluster based on resource requests from pods.
- The Cluster Autoscaler doesn’t directly measure CPU and memory usage values to make a scaling decision
- The Cluster Autoscaler checks every 10 seconds to detect any pods in a pending state, suggesting that the scheduler could not assign them to a node due to insufficient cluster capacity.



Cluster Autoscaling



Cluster Autoscaling limitation

- CA does not base scaling decisions on CPU or memory usage but solely on a pod's specified requests and limits for these resources.
- Unused computing resources requested by users are not identified by CA, leading to potential cluster waste and suboptimal utilization efficiency.
- When there's a need to scale up the cluster, CA initiates a scale-up request to the cloud provider within 30–60 seconds.
- The actual time taken by the cloud provider to create a new node can extend to several minutes or more.

universität innsbruck

Thank you for your attention☺

Dr. Zahra Najafabadi Samani

Email: Zahra.Najafabadi-Samani@uibk.ac.at