Autoscaling in Kubernetes

Abdullah Al Hasib

abdullah.alhasib@kratosdefense.com

09-02-2022

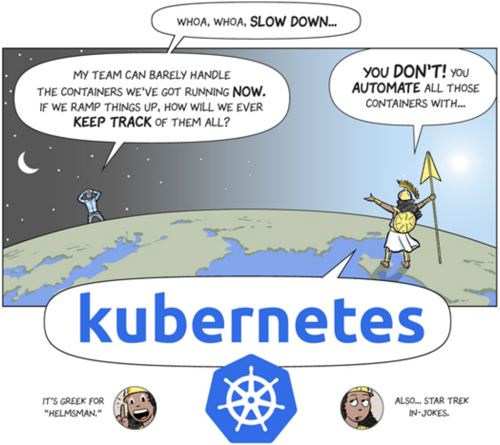
KRATOS®
DEFENSE & SECURITY SOLUTIONS

1. Kubernetes Recap
2. Autoscaling
3. HPA
4. VPA
5. CA
6. Conclusion

1. Kubernetes Recap
2. Autoscaling
3. HPA
4. VPA
5. CA
6. Conclusion

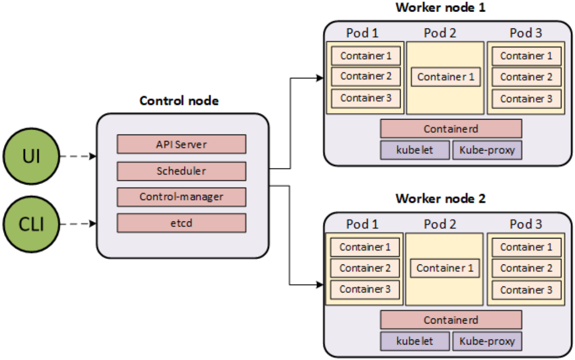
A Quick Recap

What is kubernetes:


WHOA, WHOA, SLOW DOWN...
MY TEAM CAN BARELY HANDLE THE CONTAINERS WE'VE GOT RUNNING NOW. IF WE RAMP THINGS UP, HOW WILL WE EVER KEEP TRACK OF THEM ALL?
YOU DON'T! YOU AUTOMATE ALL THOSE CONTAINERS WITH...
kubernetes
IT'S GREEK FOR "HELMSMAN."
ALSO... STAR TREK IN-JOKES.

A Quick Recap

Kubernetes architecture:



1. Kubernetes Recap
2. Autoscaling
3. HPA
4. VPA
5. CA
6. Conclusion

Autoscaling

Autoscaling:

- automatically adjusts computational resource usage according to the load.

Why do we need it:

- to cope with the demand
- to reduce cost
- to reduce power consumption

Autoscaling Types

The Kubernetes autoscaling mechanism uses two layers:

- Pod-based scaling - supported by
  - Horizontal Pod Autoscaler (HPA)
  - Vertical Pod Autoscaler (VPA)
- Node-based scaling - supported by
  - Cluster Autoscaler (CA)

	Pods	**Nodes**
Horizontal	# of pods	# of nodes
Vertical	resources of a pod	resources of a node

1 Kubernetes Recap

2 Autoscaling

3 HPA

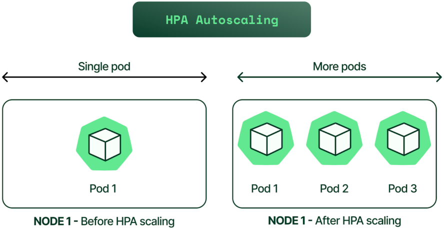
4 VPA

5 CA

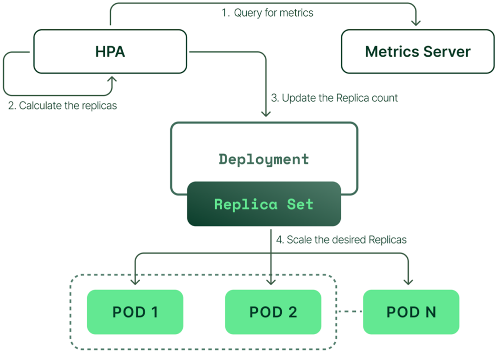
6 Conclusion

Horizontal Pod Autoscaling

HPA increases or decreases the number of pods in a replication controller, deployment, replica set etc. based on CPU utilization



How Does HPA Work?


1. Query for metrics
HPA
Metrics Server
2. Calculate the replicas
3. Update the Replica count
Deployment
Replica Set
4. Scale the desired Replicas
POD 1
POD 2
POD N

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

Metrics Types

Types of metrics APIs:

1. Resource Metrics (`metrics.k8s.io`)
   - predefined resource usage metrics of Pods and Nodes
   - can be expressed as raw value of percentage
   - example: `CPU`, `Memory`

Metrics Types

Types of metrics APIs:

1. Resource Metrics (`metrics.k8s.io`)
   - predefined resource usage metrics of Pods and Nodes
   - can be expressed as raw value of percentage
   - example: `CPU`, `Memory`
2. Custom Metrics (`custom.metrics.k8s.io`)
   - custom metrics associated with a Kubernetes object
   - example: rate_of_client_requests

Metrics Types

Types of metrics APIs:

1. Resource Metrics (`metrics.k8s.io`)
   - predefined resource usage metrics of Pods and Nodes
   - can be expressed as raw value of percentage
   - example: `CPU`, `Memory`
2. Custom Metrics (`custom.metrics.k8s.io`)
   - custom metrics associated with a Kubernetes object
   - example: rate_of_client_requests
3. External Metrics (`external.metrics.k8s.io`)
   - custom metrics not associated with a Kubernetes object

Customizing Scaling Behavior

1. `scaleup`: control scaling behavior while scaling up
   - `stabilizationWindowSeconds`: (default:0)
   - `selectPolicy`: can be `Min`, `Max`, `Disabled` (default:Max)
   - `policies`
     - `type`: `Pods` or `Percent`
     - `periodSeconds`: (default: 60)
     - `value`

Customizing Scaling Behavior

1. scaleup: control scaling behavior while scaling up
   - stabilizationWindowSeconds: (default:0)
   - selectPolicy: can be Min, Max, Disabled (default:Max)
   - policies
     - type: Pods or Percent
     - periodSeconds: (default: 60)
     - value
2. scaledown: control scaling behavior while scaling down
   - stabilizationWindowSeconds: (default: 300)
   - selectPolicy: can be Min, Max, Disabled (default:Max)
   - policies
     - type: Pods or Percent
     - periodSeconds: (default: 60)
     - value

User Stories

1. Scale up as fast as possible, scale down as usual

```
behavior:
  scaleUp:
    policies:
    - type: Percent
      value: 900%
```

User Stories

1. Scale up as fast as possible, scale down as usual

```
behavior:
  scaleUp:
    policies:
    - type: Percent
      value: 900%
```

2. Scale up as usual, do not scale down

```
behavior:
  scaleDown:
    selectPolicy: Disabled
```

User Stories

4. Scale Up As Fast As Possible, Scale Down Very Gradually

```yaml
behavior:
  scaleUp:
    policies:
    - type: Percent
      value: 900%
  scaleDown:
    policies:
    - type: Pods
      value: 1
      periodSeconds: 600
```

User Stories (Cont'd)

❺ Stabilization before scaling down

```yaml
behavior:
  scaleDown:
    stabilizationWindowSeconds: 600
    policies:
    - type: Pods
      value: 5
```

HPA Demo

HPA Demo

1. Kubernetes Recap

2. Autoscaling

3. HPA

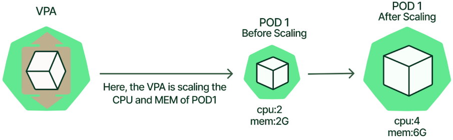
4. VPA

5. CA

6. Conclusion

Vertical Pod Autoscaling

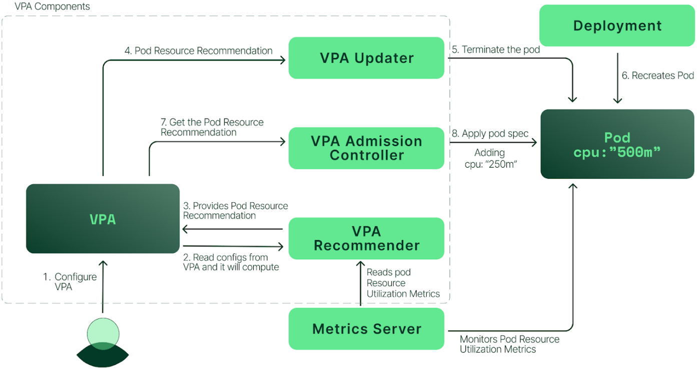
VPA adjusts the resource requests and limits of containers in the cluster.



Resource configuration types:

- requests: define the minimum amount of resources that containers need
- limits: define the maximum amount of resources that a given container can consume

How Does VPA Work?


VPA Components
4. Pod Resource Recommendation
VPA Updater
5. Terminate the pod
Deployment
6. Recreates Pod
7. Get the Pod Resource Recommendation
VPA Admission Controller
8. Apply pod spec
Adding cpu: "250m"
Pod cpu:"500m"
VPA
3. Provides Pod Resource Recommendation
VPA Recommender
2. Read configs from VPA and it will compute
1. Configure VPA
Reads pod Resource Utilization Metrics
Metrics Server
Monitors Pod Resource Utilization Metrics

VPA Concepts

Types Operation Modes:

- Off: provides recommendations only
- Initial: only assigns resource requests on pod creation
- Recreate: assigns resource requests on pod creation and updates them on existing pods by evicting them. [should be used rarely]
- Auto: currently does the same as Recreate. [may cause service downtime]

VPA Demo

VPA Demo

1 Kubernetes Recap

2 Autoscaling

3 HPA

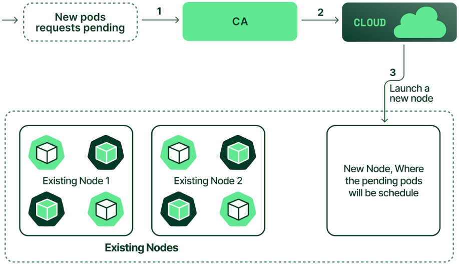
4 VPA

5 CA

6 Conclusion

Cluster Autoscaling

CA adjusts the number of nodes in the cluster when pods fail to schedule or when nodes are underutilized


New pods requests pending
1
CA
2
CLOUD
3
Launch a new node
Existing Node 1
Existing Node 2
New Node, Where the pending pods will be schedule
Existing Nodes

1. Kubernetes Recap
2. Autoscaling
3. HPA
4. VPA
5. CA
6. Conclusion

Conclusion

- Install a metric server
- Do not mix HPA with VPA
- Define pod requests and limits
- Resource requests should be close to the average usage of the pods
- Increase CPU limits for slow starting applications

[1] Minikube Handbook
[2] Horizontal Pod Autoscaling
[3] Vertical Pod Autoscaling
[4] Scaling Kubernetes Clusters
[5] KEDA: Kubernetes Event-Driven Autoscaling

Thanks!