Decision Tree

* [Horizontal Pod Autoscaler](https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale)
* Why
  + - * HPA changes the shape of your Kubernetes workload by automatically increasing or decreasing the number of Pods in response to the workload's CPU or memory consumption, or in response to custom metrics reported from within Kubernetes or external metrics from sources outside of your cluster
    - Alternatives
* Manually or programmatically monitor and alert on pod CPU utilization or [custom metrics](https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/#support-for-custom-metrics) and adjusting the [ReplicaSet](https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/" \t "_blank) via the Kubernetes API
* Limitations :
* Do not use HPA together with Vertical Pod Autoscaling (VPA) on CPU or memory. However, you can use HPA with VPA if HPA evaluates metrics other than CPU or memory
* If you have a Deployment, don't configure HPA on the ReplicaSet or Replication Controller backing it. When you perform a rolling update on the Deployment or Replication Controller, it is effectively replaced by a new Replication Controller. Instead configure HPA on the Deployment itself
* HPA cannot be used for workloads that cannot be scaled, such as DaemonSets.
* Cluster Autoscaler
* Why
  + - * The Kubernetes Cluster Autoscaler automatically adjusts the number of nodes in your cluster when the pods fail to launch due to lack of resources or when nodes in the cluster are underutilized and their pods can be rescheduled onto other nodes in the cluster. When demand is high, cluster autoscaler adds nodes to the node pool. When demand is low, cluster autoscaler scales back down to a minimum size that you designate. This can increase the availability of your workloads when you need it, while controlling costs
    - **Limitations**
      * Local PersistentVolumes.
      * Scaling up a node group of size 0, for Pods requesting resources beyond CPU, memory and GPU (ex. ephemeral-storage)
      * Cluster autoscaler supports up to 5000 nodes running 30 Pods each. For more details on scalability guarantees, refer to [Scalability report](https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/proposals/scalability_tests.md)
      * When scaling down, cluster autoscaler honors a graceful termination period of 10 minutes for rescheduling the node's Pods onto a different node before forcibly terminating the node
      * Occasionally, cluster autoscaler cannot scale down completely and an extra node exists after scaling down. This can occur when required system Pods are scheduled onto different nodes, because there is no trigger for any of those Pods to be moved to a different node. See [I have a couple of nodes with low utilization, but they are not scaled down. Why?](https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md#i-have-a-couple-of-nodes-with-low-utilization-but-they-are-not-scaled-down-why). To work around this limitation, you can configure a [Pod disruption budget](https://kubernetes.io/docs/concepts/workloads/pods/disruptions/)
      * Custom scheduling with altered Filters is not supported
* Considerations
* Enable [logging of the Cluster Autoscaler](https://docs.microsoft.com/en-us/azure/aks/cluster-autoscaler#retrieve-cluster-autoscaler-logs-and-status) so you can understand why it is autoscaling…or not.
* Alternatives
* Manually or programmatically monitor and alert on Kubernetes scheduler failures or node CPU/RAM utilization and [adjusting the node count](https://docs.microsoft.com/en-us/azure/aks/scale-cluster) via the Azure CLI or Azure Portal
* CNI (vs kubenet)
  + - Why
      * Kubenet is a very basic, simple network plugin, on Linux only. It does not, of itself, implement more advanced features like cross-node networking or network policy. It is typically used together with a cloud provider that sets up routing rules for communication between nodes, or in single-node environments.
      * With kubenet, nodes get an IP address from the Azure Virtual Network subnet. Pods receive an IP address from a logically different address space (POD CIDR - POD Classless Inter-Domain Routing) to the Azure Virtual Network Subnet of the nodes. Network address translation (NAT) is then configured so that the pods can reach resources on the Azure Virtual Network. The source IP address of the traffic is NAT’d to the node’s primary IP address. NAT is under Layer 3 operations. This approach greatly reduces the number of IP addresses that you need to reserve in your network space for pods to use.
    - [Alternatives](https://docs.microsoft.com/en-us/azure/aks/concepts-network#compare-network-models)
* Use kubenet when:
  + - * + You have limited IP address space.
        + Most of the pod communication is within the cluster.
        + You don't need advanced AKS features such as virtual nodes or Azure Network Policy. Use Calico network policies.
* Use Azure CNI when:
  + - * + You have available IP address space.
        + Most of the pod communication is to resources outside of the cluster.
        + You don't want to manage the UDRs.
        + You need AKS advanced features such as virtual nodes or Azure Network Policy. Use Calico network policies.
        + You need a nodepool for Windows Containers.
* [Azure Monitor for Containers](https://docs.microsoft.com/en-us/azure/azure-monitor/insights/container-insights-overview)
* Why
* Azure Monitor for containers collects lots of data to effectively monitor Kubernetes clusters. Full observability into your applications, infrastructure, and network. It provides sophisticated tools for collecting and analyzing telemetry that allow you to maximize the performance and availability of your cloud and on-premises resources and applications
* Store and analyze operational telemetry, advanced analytic engine ,interactive query language, allows you to configure charts, dashboards, alerting and [triggering](https://docs.microsoft.com/en-us/azure/azure-monitor/platform/action-groups#create-an-action-group-by-using-the-azure-portal) of Automation Runbook, Azure Function, Email Azure Resource Manager Role, Email/SMS/Push/Voice, ITSM, Logic App, Secure Webhook, Webhook
* Alternatives
* There are a wide range of other tools to consider:
* [Loki](https://grafana.com/oss/loki/) - a horizontally-scalable, highly-available, multi-tenant log aggregation system inspired by Prometheus.
* [Prometheus](https://prometheus.io/docs/introduction/overview/) - is an open-source systems monitoring and alerting toolkit
* [Grafana](https://grafana.com/grafana/) - allows you to query, visualize, alert on and understand your metrics no matter where they are stored. Create, explore, and share dashboards with your team and foster a data driven culture.
* Splunk
* Journald
* Logging considerations
* Why
* Diagnostic logs should be enabled when needed - as excessive logging can hurt performance.
* However, we recommend enabling logging for the following components:
  + - * + [Cluster Autoscaler](https://docs.microsoft.com/en-us/azure/aks/cluster-autoscaler#retrieve-cluster-autoscaler-logs-and-status) - understand why it is autoscaling…or not
        + [KubeControllerManager](https://kubernetes.io/docs/tasks/debug-application-cluster/debug-cluster/#master) - visibility into the replication controller and any impact of using Azure Policy