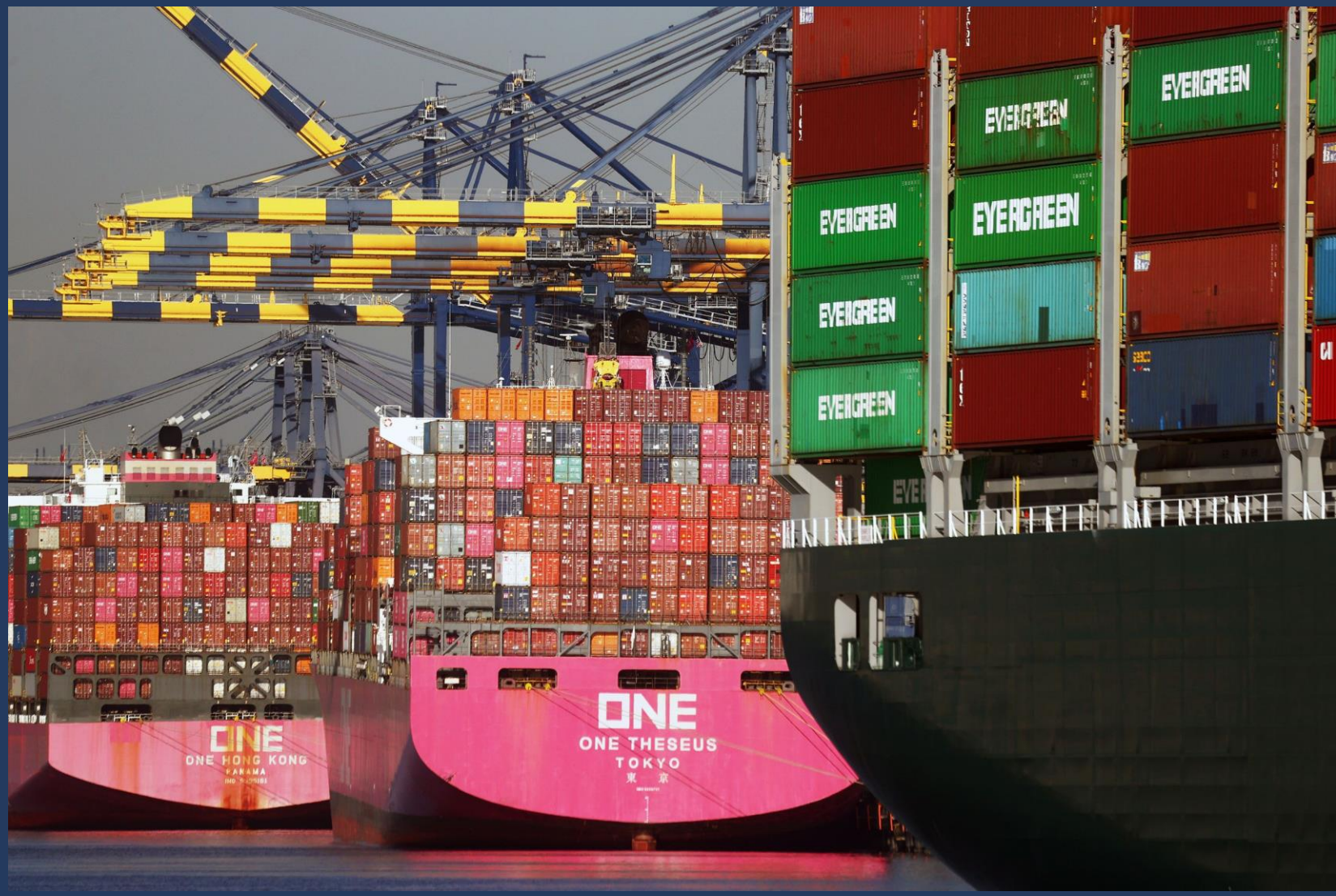
Microsoft

Kubernetes Technical Briefing

Module 7:
Advanced Kubernetes Topics
Part 2



Agenda

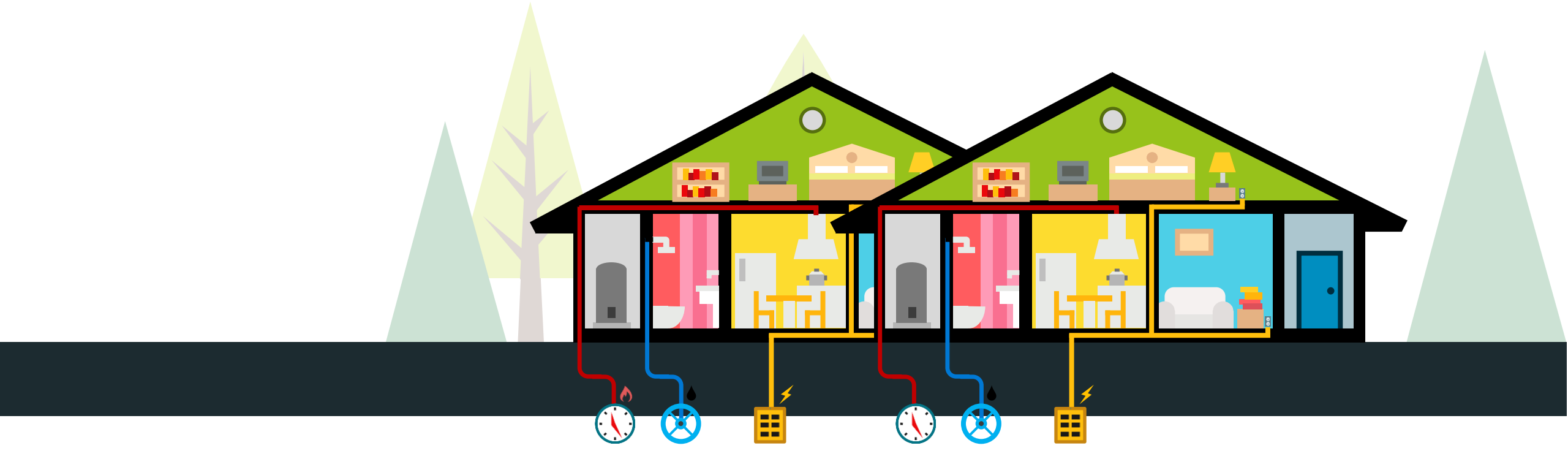
- Requests and Limits
- Limit Ranges and Resource Quotas
- Vertical Pod Autoscaler
- Horizontal Pod Autoscaler
- Kubernetes Event Driven Autoscaling (KEDA)
- AKS Patching and Upgrading
- Pod Disruption Budgets

Requests and Limits

1

Resources: Virtual Machines

- VMs are single, isolated entities residing on the same host
- Each VM has its own set of dedicated resources
- Think of single houses on a block



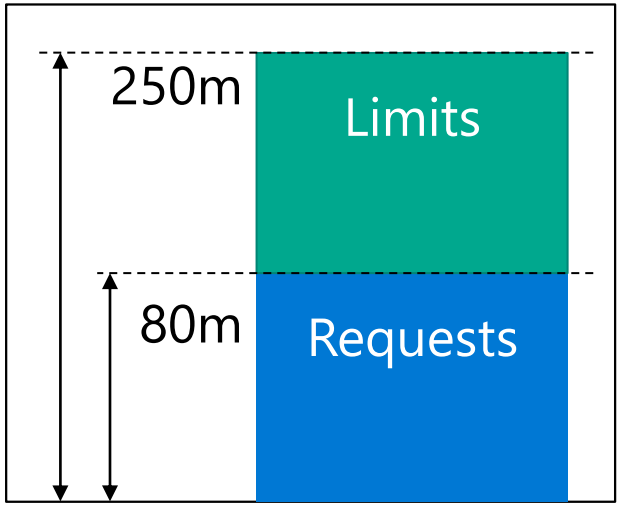
Resources: Containers

- Containers share the resources of the VM (Node) they're running on.
- Just like apartments in a building, Containers must contend with others competing for the same set of limited resources.
- A good apartment building should be able to limit the amount of resources each tenant can use so that others won't run out.

Resource Request and Limits

- A Container can ***request*** *resources*, usually for CPU and memory, to be set aside for its use.
- A ***request*** *as the* amount of a resource that the Kubernetes will guarantee to the container.
- A ***limit*** is the maximum amount of a resource that Kubernetes will allow the container to use.
- The scheduler **uses the requested values to decide which Node to schedule** the Pod on.
- If the node where a Pod is running has enough of a resource available, it's possible (and allowed) for a container to use more resource than its ***request*** for that resource specifies.
- However, a container is not allowed to use more than its resource ***limit***.

```yaml
resources:
  requests:
    cpu: 80m
    memory: 128Mi
  limits:
    cpu: 250m
    memory: 256Mi
```



Resource Request and Limits

- CPU and memory resources requests and limits are specified as follows:
  - CPU resources can be specified in terms of milli-cores (1/1000th of a CPU).
    - For example, **cpu: 100m** is asking for 1/10th of a CPU core.
  - Memory requests can be specified in T, G, M, K. **You can also use the power-of-two equivalents**: Ti, Gi, Mi, Ki.
    - Example: 256K (kilobytes) = 250Ki (kibibytes)

```yaml
resources:
  requests:
    cpu: 80m
    memory: 128Mi
  limits:
    cpu: 250m
    memory: 256Mi
```

- If a Container specifies a CPU/memory ***limit***, but does not specify a CPU/memory ***request***, Kubernetes automatically assigns a memory **request that matches the limit**.
- If your Pod's containers have **no requests or limits**, then, by definition, they are using more than they requested, so are prime candidates for termination.

Resource Guarantees

- ***Compressible Resource Guarantees***:
  - Kubernetes only supports **CPU** for now.
  - Pods will be *throttled* if they exceed their limit. If limit is unspecified, then the pods can use excess CPU when available.
- ***Incompressible Resource Guarantees***:
  - Kubernetes only supports **memory** for now.
  - Pods will get the amount of memory they request, if they exceed their memory request, they *could be killed* (if some other Pod needs memory), but if pods consume less memory than requested, they will not be killed.
  - When Pods use more memory than their limit, the process that is using the most amount of memory, inside one of the pod's containers, will be killed by the kernel.

Limit Ranges and Resource Quotas

2

Default Requests and Limits

- In a large system, it may not be practical to define resource requests/limits for all the existing containers.
- The **LimitRange** object allows you to specify a default set of requests (***defaultRequests***) and limits (***defaults***) for any Container created in a *Namespace*.
- When requests/limits are not specified by the Container, the defaults in the **LimitRange** will be used.
- When requests/limits are specified, those values override the defaults in the **LimitRange**.
- The ***min*** (requests) and ***max*** (limits) can also be specified to ensure values specified for a Container stay within set bounds.

```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: mem-limit-range
spec:
  limits:
    # Default limit if not specified
    - default:
        cpu: 200m
        memory: 256Mi
      # Default request if not specified
      defaultRequest:
        cpu: 100m
        memory: 128Mi
      # Max limit if specified
      max:
        cpu: 1
        memory: 1Gi
      # Min request if specified
      min:
        cpu: 50m
        memory: 100Mi
      type: Container
```

Namespace Resource Quota

- A **ResourceQuota** object, defines constraints that limit aggregate resource (total CPU and memory) consumption per namespace.
- A Resource Quota can also limit the quantity of objects that can be created in a namespace by type (max PVCs for example).
- Resource Quotas only work if requests and limits specified.
- It's **highly recommended** to create a **LimitRange** resource in the same namespace as the **ResourceQuota**.
- Limits should be large enough to accommodate upgrades.

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: mem-cpu-rq
spec:
  hard:
    requests.cpu: "1200m"
    limits.cpu: "2400m"
    requests.memory: 1.5Gi
    limits.memory: 3Gi
```

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: resources-rq
spec:
  hard:
    pods: "10"
    configmaps: "10"
    secrets: "10"
    persistentvolumeclaims: "4"
    services: "10"
    services.loadbalancers: "2"
```

Resource Requests, Limits, Ranges and Quotas

Managing Resource Requests and Limits

Vertical Pod Autoscalar

3

Quality of Service (QoS) Classes

- Kubernetes assigns different levels of **Quality of Service (QoS)** to Pods depending on their **request** and **limits** values.
- Kubernetes uses QoS classes to make decisions about scheduling and evicting Pods.
- ***Best-Effort*** (QoS)
  - Pods will be treated as **lowest priority**. Processes in these pods are the first to get killed if the system runs out of memory. Default setting if no requests are specified by any Containers.
- ***Burstable*** (QoS)
  - Pods have some form of minimal resource guarantee but can use more resources when available. Under system memory pressure, these containers are more likely to be killed once they exceed their requests.
  - At least one Container in the Pod has a memory or CPU requests specified.
- ***Guaranteed*** (QoS)
  - Pods are considered **top-priority** and are guaranteed to not be killed until they exceed their limits.
  - Requests and limits values MUST BE EQUAL!

What Values Requests and Limits Should Be Used?

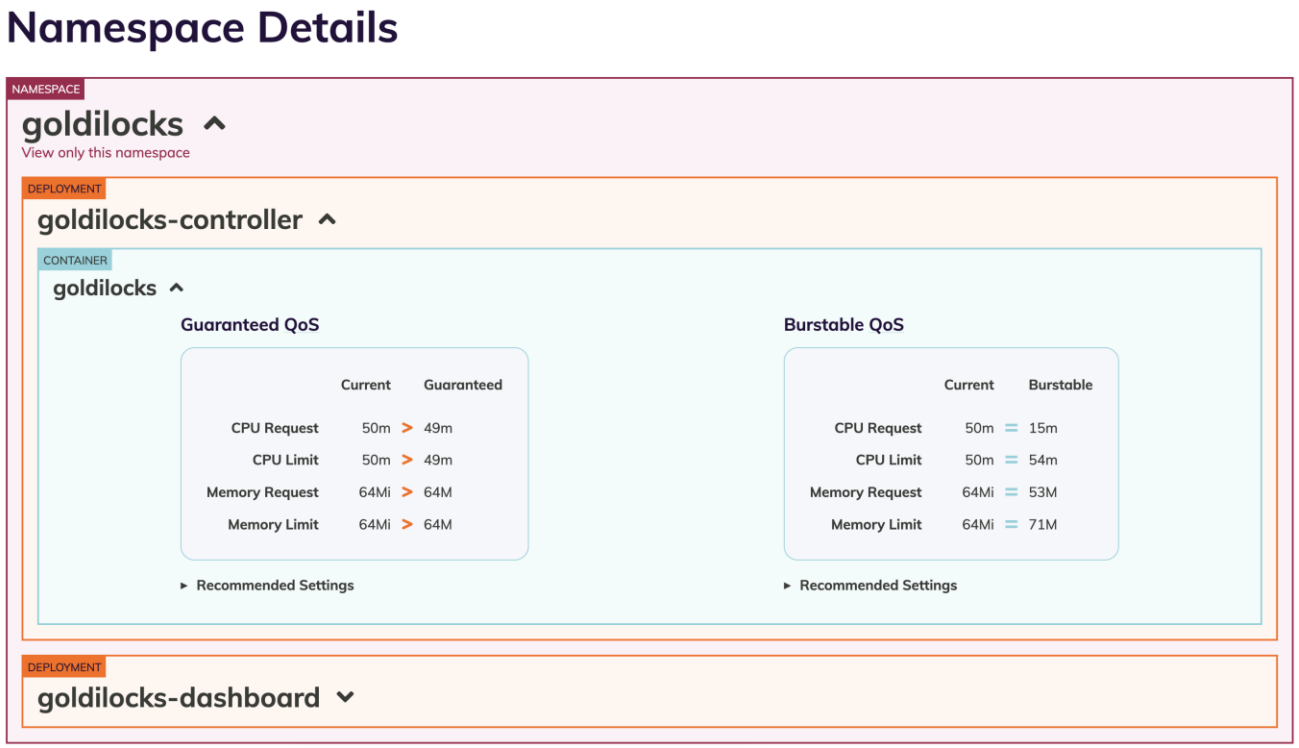
- Once you realize the benefits of setting requests and limits values for your containers, it's time to determine which values to use.
- Of course, you can make an educated guess. Based on a study done by DataDog*, about 80% of deployments in production specify requests and limits that are significantly smaller or larger than the resources their containers actually use.
- Setting request values too high could lead to Node underutilization.
- Setting them too low could lead a Node to be over provisioned and would result in evictions and CPU throttling.

Vertical Pod Autoscaler – Add-on

- The **Vertical Pod Autoscaler (VPA)** (available as a **CustomResourceDefinition**), can be used to recommend the best values to use for requests and limits, based on the current load on the containers.
- These values can then be used in manifest files to ensure these settings are optimal.
- But if resource demands changes over time? Should you keep checking the VPA recommendations periodically and update your manifest files accordingly?
- The VPA has an ***Auto*** mode, which can ***automatically update*** the requests and limits settings of Deployments as workload needs change over time.
- These updates respect the *min/max* ranges set in the **LimitRange** object, if any.

Fairwinds Goldilocks

- Fairwinds Goldilocks utility provides an implementation of ***VPA*** (in manual mode).
- Includes a dashboard to monitor usage trends to help determine current resource settings.



Vertical Pod Autoscaler Demo

Determining Optimal Resource Requests and Limits Settings

Horizontal Pod Autoscalar

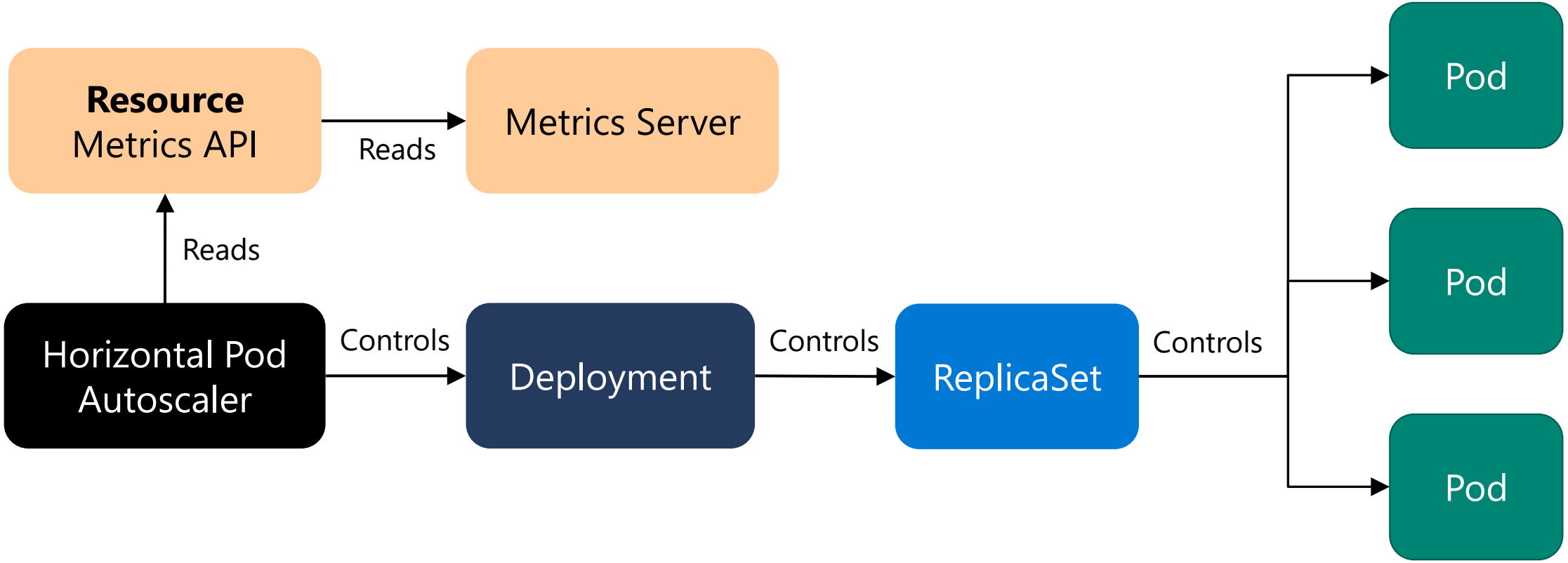
4

Horizontal Pod Autoscaler – Built-in

- Horizontal scaling is the process of increasing/decreasing the number of Pods (replicas) in response to the load on a workload
- This is different from vertical scaling, which for Kubernetes would mean assigning more resources (for example: memory or CPU) to the Pods that are already running for the workload. See VPA.
- A **HorizontalPodAutoscaler** automatically updates a workload resource (such as a *Deployment* or *StatefulSet*), with the aim of automatically scaling the workload to match demand.
- Horizontal Pod autoscaling does not apply to objects that can't be scaled (for example: a *DaemonSet*.)

Horizontal Pod Autoscaler – Built-in

- The **HorizontalPodAutoscaler** is implemented as a Kubernetes API resource and a controller.



Horizontal Pod Autoscaler Demo

Automatically scales replicas based on CPU or Memory thresholds

Kubernetes Event-driven Autoscaling (KEDA)

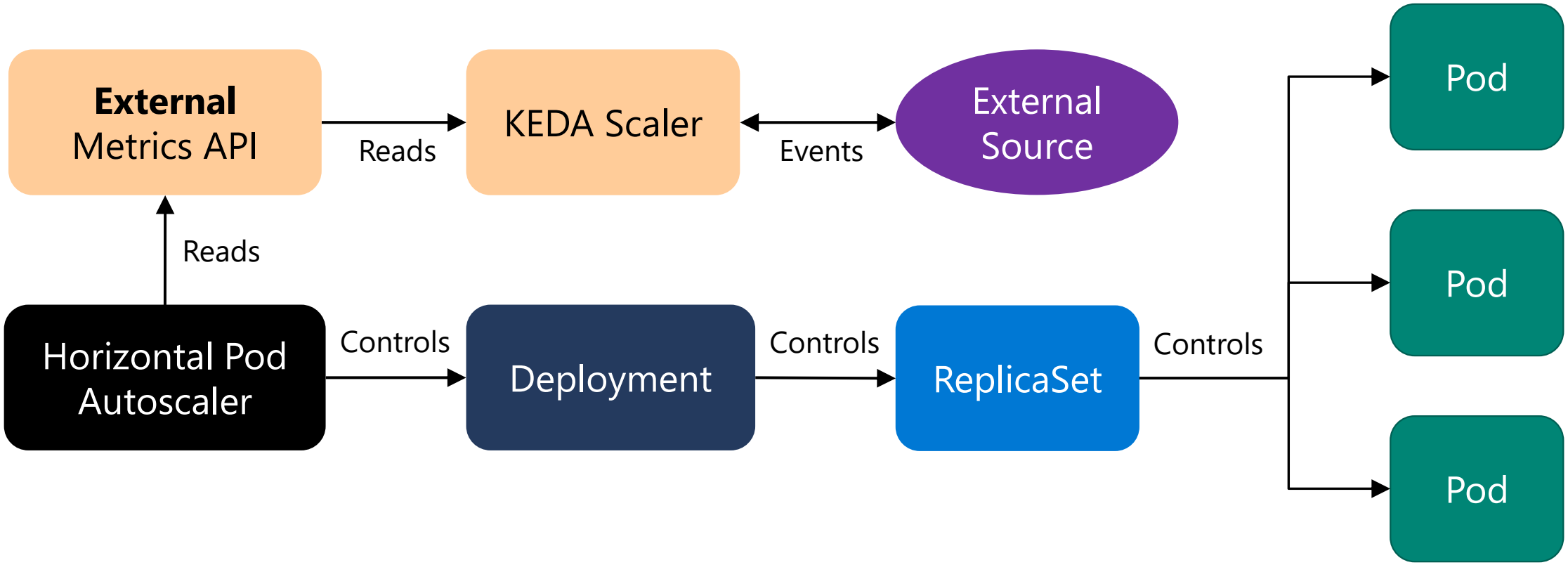
5

Kubernetes Event Driven Autoscaling (KEDA)

- **KEDA** is a single-purpose and lightweight component that can be added into any Kubernetes cluster.
- **KEDA** works alongside standard Kubernetes components like the *Horizontal Pod Autoscaler* and can extend functionality without overwriting or duplication.
- With **KEDA** you can explicitly map the apps you want to use event-driven scale, with other apps continuing to function.
- This makes **KEDA** a flexible and safe option to run alongside any number of any other Kubernetes applications or frameworks.

Horizontal Pod Autoscaler – Built-in

- With **KEDA**, you can drive the scaling of any container in Kubernetes based on the number of events needing to be processed.



KEDA Scalers

- Here's a partial list of KEDA scalers (external event sources) currently supported, with additional ones added with almost every update.

- ActiveMQ
- ActiveMQ Artemis
- Apache Kafka
- AWS CloudWatch
- AWS Kinesis Stream
- AWS SQS Queue
- Azure App Insights
- Azure Blob Storage
- Azure Event Hubs
- Azure Log Analytics
- **Azure Monitor**
- Azure Pipelines
- **Azure Service Bus**
- Azure Storage Queue
- Cassandra
- CPU
- **Cron**
- Datadog
- Elasticsearch
- External
- External Push
- Graphite
- Huawei Cloudeye
- IBM MQ
- InfluxDB
- Liiklus Topic
- Memory
- Metrics API
- MongoDB
- **MSSQL**
- MySQL
- NATS Streaming
- New Relic
- OpenStack Metric
- OpenStack Swift
- PostgreSQL
- Prometheus
- **RabbitMQ Queue**
- Redis Lists
- Redis Streams

KEDA Demo

Automatically scaling replicas based on External metrics like RabbitMQ

AKS Patching and Upgrading

6

AKS Upgrade Options

- Azure offers 3 primary upgrade modes for AKS, which can be configured to be performed automatically if desired.
- **Node image updates** – Azure issues OS image security updates every 1-2 weeks. It's important that the nodes in an AKS cluster be upgraded frequently to the latest OS images.
- **Kubernetes Patch upgrades** – Kubernetes patches (ex: 1.25.5 -> 1.25.9) are available on a regular basis.
- **Kubernetes Minor version upgrades** – Kubernetes releases minor versions (1.25 -> 1.26) every 3-4 months.
  - Review the Deprecated API Migration Guide before performing a minor Kubernetes upgrade.

Node Image Upgrades

- Microsoft provides patches and new images for image nodes weekly but does not automatically patch them by default.
- Updated node images contain up-to-date OS security patches, kernel updates, Kubernetes security updates, newer versions of binaries like *kubelet*, and component version updates.
- Consider checking and applying node image upgrades bi-weekly or automating the node image upgrade process.
- Upgrade all node pools to latest OS image:

```
az aks upgrade -g k8slabs-rg --cluster-name kizaks --node-image-only
```

- Create an auto-upgrade channel to automatically update OS images weekly:

```
az aks update -g k8slabs-rg --cluster-name kizaks `
        --auto-upgrade-channel node-image
```

Kubernetes Patch/Minor Version Upgrades

- Minor version releases include new features and improvements.
- Patch releases are more frequent (sometimes weekly) and are intended for critical bug fixes within a minor version. Patch releases include fixes for security vulnerabilities or major bugs.
- Get a list of available versions in region:

```
az aks get-versions --location eastus
```

```
KubernetesVersion    Upgrades
-------------------  ----------------------
1.26.3               None available
1.26.0               1.26.3
1.25.6               1.26.0, 1.26.3
1.25.5               1.25.6, 1.26.0, 1.26.3
1.24.10              1.25.5, 1.25.6
1.24.9               1.24.10, 1.25.5, 1.25.6
```

- Get a list of available upgrades for your cluster:

```
az aks get-upgrades -g azure-kiz-rg --cluster-name myAKSName
```

```
Name     ResourceGroup    MasterVersion    Upgrades
-------  ---------------  ---------------  ----------------------
default  azure-kiz-rg     1.24.9           1.24.10, 1.25.5, 1.25.6
```

Kubernetes Minor Version Upgrades

- When you upgrade a supported AKS cluster, Kubernetes **minor versions can't be skipped**. You must perform all upgrades sequentially by major version number. For example, upgrades between 1.19.x -> 1.20.x or 1.20.x -> 1.21.x are allowed, however 1.19.x -> 1.21.x is not allowed.
- Skipping multiple versions can only be done when upgrading from an unsupported version to a supported version. For example: unsupported 1.19.x -> a supported 1.24.x.
- When performing an upgrade from an unsupported version that skips two or more minor versions, the upgrade is performed **without any guarantee of functionality and is excluded from the service-level agreements.**
- If your version is significantly out of date, **we recommend you recreate your cluster**.
- **Example:** Upgrade cluster to a patch/minor version

```
az aks upgrade --kubernetes-version 1.23.1 -g myRG --name myAKS
```

Planned Maintenance Window Schedule

- By default, automatic updates can happen at any time.
- Planned Maintenance allows you to schedule weekly maintenance windows to perform updates and minimize workload impact.
- Once scheduled, upgrades occur only during the window you selected.
- **aksManagedAutoUpgradeSchedule** – Controls when cluster upgrades scheduled by your designated auto-upgrade channel are performed.
- **aksManagedNodeOSUpgradeSchedule** - Controls when node operating system upgrades scheduled by your node OS auto-upgrade channel are performed.
- **Example:** Schedule maintenance to run every third Friday between 12:00 AM and 8:00 AM in the UTC+5:30 timezone

```
az aks maintenanceconfiguration add -n aksManagedAutoUpgradeSchedule
    --schedule-type Weekly --day-of-week Friday
    --interval-weeks 3 --start-time 00:00 --duration 8
```

Pod Disruption Budget (PDB)

7

Pod Disruptions

- There are two types of disruptions that can remove a running Pod:
- **Involuntary Disruptions** – Possible causes:
  - A hardware failure of the physical machine backing the node
  - Cluster administrator deletes VM (instance) by mistake
  - Node disappears from the cluster due to cluster network partition
  - Eviction of a pod due to the node being out-of-resources.
- **Voluntary Disruptions** – Possible causes:
  - Deleting the deployment or other controller that manages the Pod
  - Updating a deployment's pod template causing a restart
  - Draining a node for repair or upgrade.
  - Draining a node from a cluster to scale the cluster down (Cluster Autoscaling).
  - Removing a pod from a node to permit something else to fit on that node.

Pod Disruption Budget (PDB)

- Kubernetes offers features to help you run highly available applications even when you introduce frequent *voluntary disruptions*.
- You can create **PodDisruptionBudgets (PDB)** to protected applications from voluntary disruptions.
- PDBs limits the number of Pods of a replicated application that are down simultaneously when node are intentionally taken offline.
- PDBs **DO NOT** protect pods from *involuntary disruptions*!

Pod Disruption Budget (PDB) Configurations

- PDBs have only 2 settings:
  - **Selector** – Specifies the labels of the Pods to limit.
  - **Disruption Budget**:
    - **maxUnavailable** – Maximum number (or percentage) of Pods that can be unavailable at any time.
    - **minAvailable** – Minimum number (or percentage) of Pods that must be always available.
  - One budget is required, both are not allowed.
- ***maxUnavailable*** in a *Deployment* dictates Pod behavior when a Pod rollout occurs.
- ***maxUnavailable*** in a *PodDisruptionBudget* dictates Pod behavior when a Node rollout occurs.

```yaml
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: main-pdb
spec:
  # Mutually exclusive
  minAvailable: 2
  maxUnavailable: "20%"
  selector:
    matchLabels:
      app: main
```

Node Pool Upgrade Demo with PDBs

Update Node Pool Images with PDBs in place

Microsoft

Thank you