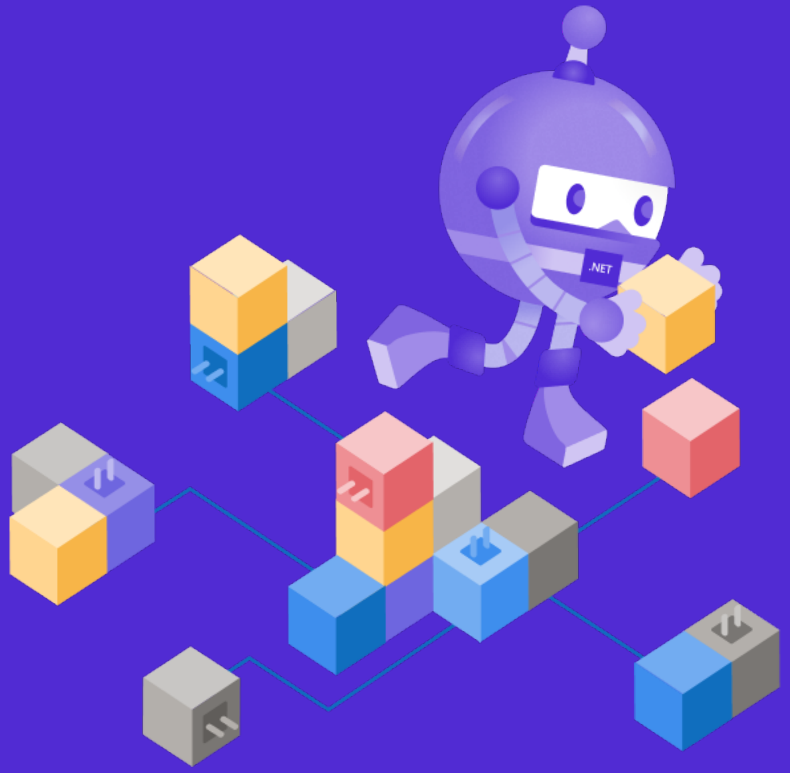
Kubernetes Core Concepts

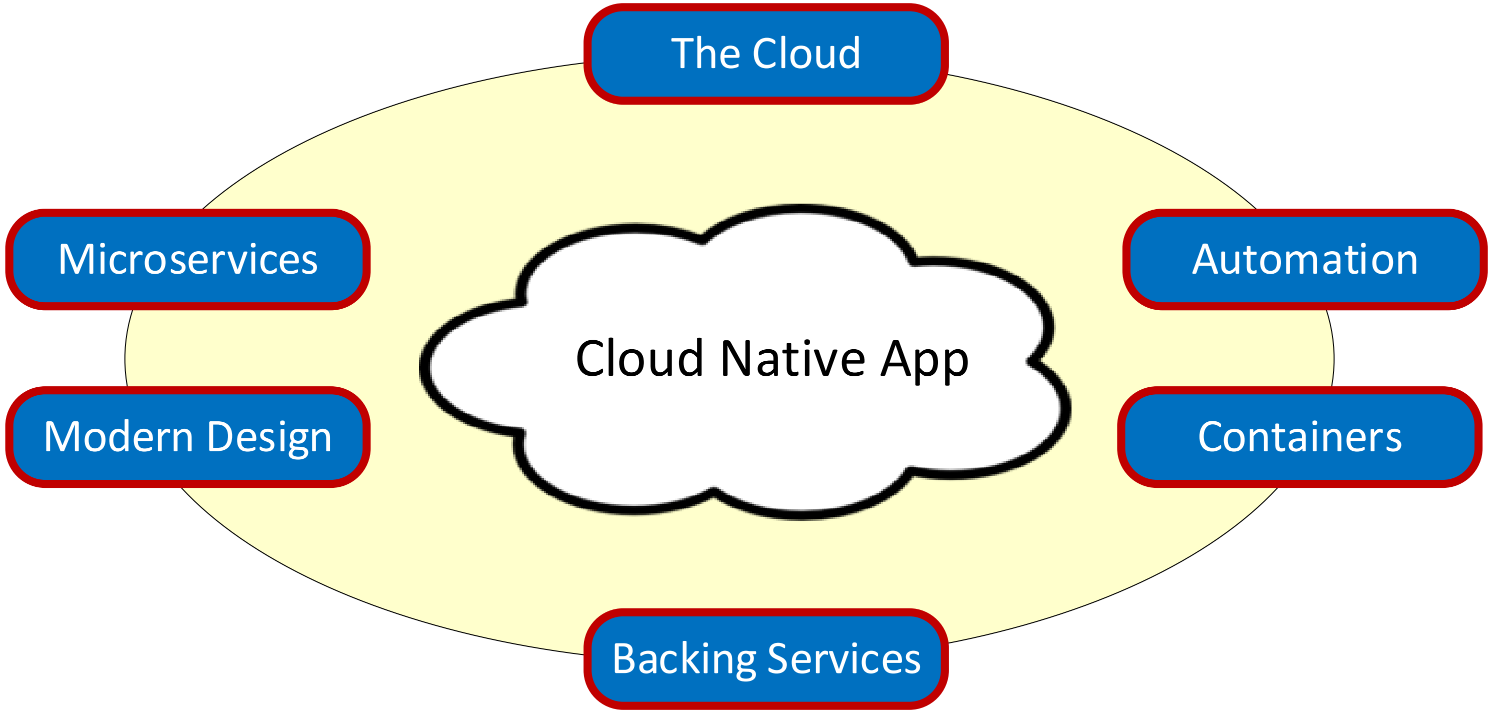
Monu Bambroo

Rob Vettor

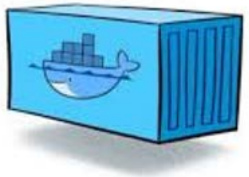


Containers and Cloud Native?

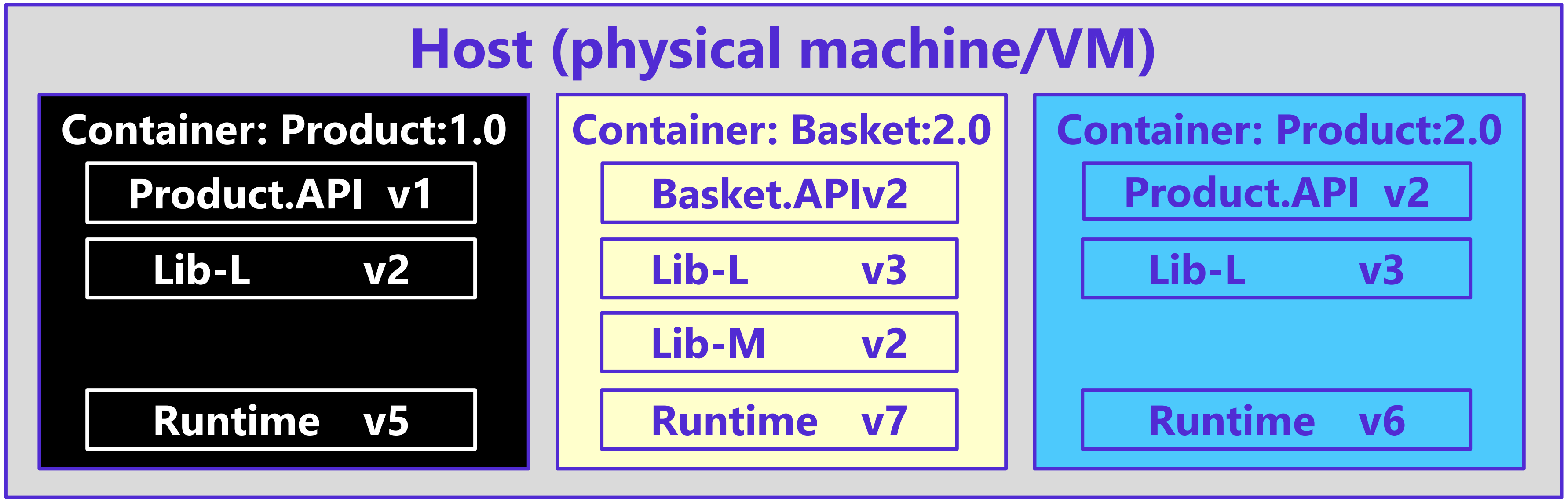
- Cloud Native is a modern approach for building cloud-based systems
- Built for the cloud, in the cloud and deployed to cloud
- Cloud Natives systems embrace six key characteristics…



What is a Container?



- Portable unit of deployment
- Packs application code and dependencies together into single unit
- Virtualization without the need of a full virtual machine
  - Slice up the OS to run multiple apps on a single host OS
  - Each container runs in isolated memory, but shares the kernel of underlying host
- Typically run one service per container (container and app share lifecycle)



Host (physical machine/VM)

Container: Product:1.0

Product.API v1

Lib-L v2

Runtime v5

Container: Basket:2.0

Basket.APIv2

Lib-L v3

Lib-M v2

Runtime v7

Container: Product:2.0

Product.API v2

Lib-L v3

Runtime v6

What is Docker?

- Docker is the company driving the software container movement
  - In a short time, it has become the de facto standard for packaging, deploying and running distributed and cloud native platforms
- Docker is a technology stack…
  - An open platform that enables you to “build, ship, and run any app, anywhere”
  - A container format
  - A set of tools for creating and running application in containers
  - Includes open source and (for-purchase) enterprise offerings
- Docker has become a standard for solving one of the costliest aspects of software: deployment

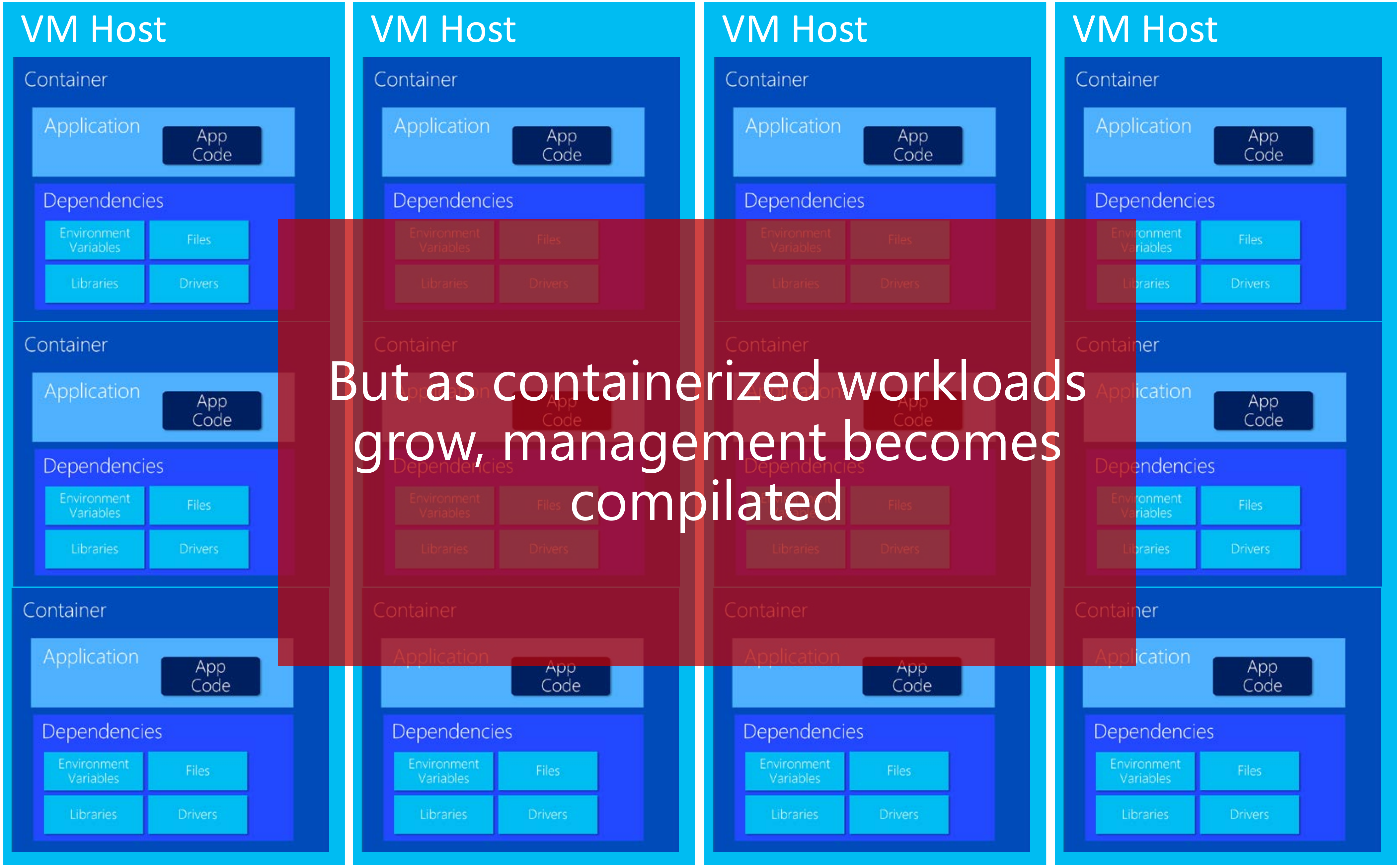
What Problems Do Containers Solve?

- Guarantees consistency across dev, test and prod environments – everything is self-contained
  - Provides portability – works across all environment
- Increases Productivity
  - Less time setting up environments/debugging environment-specific issues
- Smaller footprint than VMs
  - Increased density per host
- Isolation
  - Each container has separate slice of OS, CPU and memory isolated from other containers
- Performant and quick start-up

Container Management

Containers need to…

- Discover and talk to each other
- Sometimes manage state
- Upgrade with zero downtime
- Report health & resource usage
- Be placed appropriately across a cluster
- Scale in/out on demand
- Have resiliency to HW and SW faults



But as containerized workloads grow, management becomes compilated

VM Host

Container

Application

App Code

Dependencies

Environment Variables

Files

Libraries

Drivers

Container

Application

App Code

Dependencies

Environment Variables

Files

Libraries

Drivers

Container

Application

App Code

Dependencies

Environment Variables

Files

Libraries

Drivers

VM Host

Container

Application

App Code

Dependencies

Environment Variables

Files

Libraries

Drivers

Container

Application

App Code

Dependencies

Environment Variables

Files

Libraries

Drivers

Container

Application

App Code

Dependencies

Environment Variables

Files

Libraries

Drivers

VM Host

Container

Application

App Code

Dependencies

Environment Variables

Files

Libraries

Drivers

Container

Application

App Code

Dependencies

Environment Variables

Files

Libraries

Drivers

Container

Application

App Code

Dependencies

Environment Variables

Files

Libraries

Drivers

VM Host

Container

Application

App Code

Dependencies

Environment Variables

Files

Libraries

Drivers

Container

Application

App Code

Dependencies

Environment Variables

Files

Libraries

Drivers

Container

Application

App Code

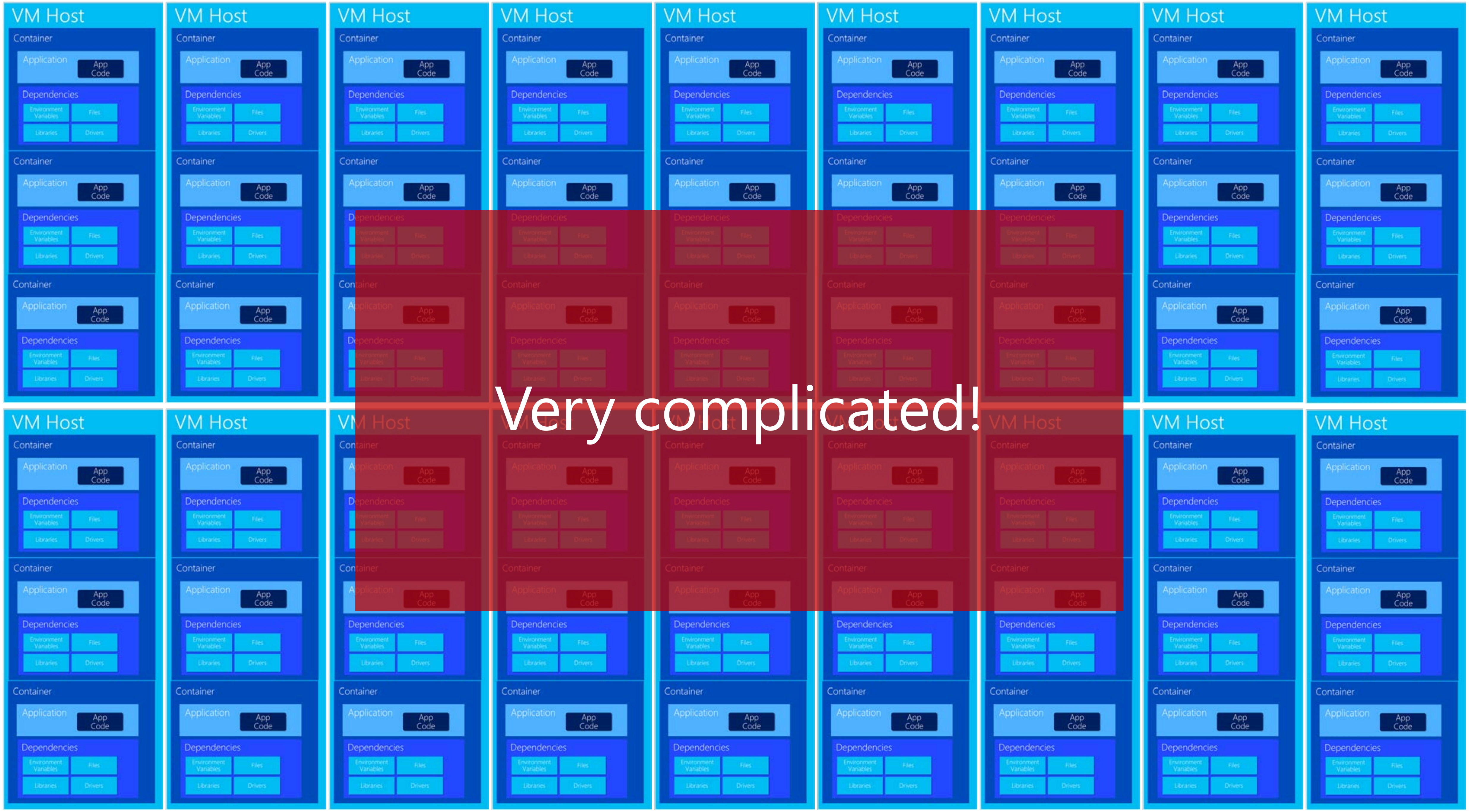
Dependencies

Environment Variables

Files

Libraries

Drivers



Very complicated!

Orchestration

Large containerized workloads require automated management, or *orchestration*…

Scheduling

Provision container instances

Affinity/anti-affinity

Provision nearby or far apart: Facilitating availability/performance

Health monitoring

Detect and fix failures

Failover

Re-provision instances to healthy machines

Scaling

Add/remove instances to meet demand

Networking

Networking overlay for container communication

Service discovery

Enable containers to locate each other

Coordinated app upgrades

Avoid downtime and automatically rollback

Kubernetes (K8s)

What exactly is Kubernetes?

- An open-source platform for managing containerized workloads
- Has become the de facto container-orchestration system
- Many call it the operating system for the cloud native world
- Automates deployment, scaling, and operational concerns of containerized workloads across clusters of VMs
- Originated from Google and donated to Cloud Native Computing Foundation
- Microsoft’s Brendan Burns was a co-creator

However, provisioning and managing it yourself is (highly) complex

Azure Kubernetes Service (AKS)

Fully-managed Kubernetes platform hosted in Azure as a PaaS service

Abstracts the complexity and operational overhead of managing Kubernetes

- You’re a tenant: Microsoft manages it, you use it
- You see Kubernetes as a managed service in the portal
- Azure sees the 1700 line configuration file

Deeply integrated with Azure dev tools and services

At no charge…

- Automated upgrades, patches
- High reliability, availability
- Automatic scaling
- Self-healing
- Monitoring

Demo: Provision AKS Cluster

How Do You Manage AKS?

- Kubectl is the official command line utility required to interact the Kubernetes cluster and the resources running upon it

```
kubectl <command> <kubernetes object type> <object name> <flags>
```

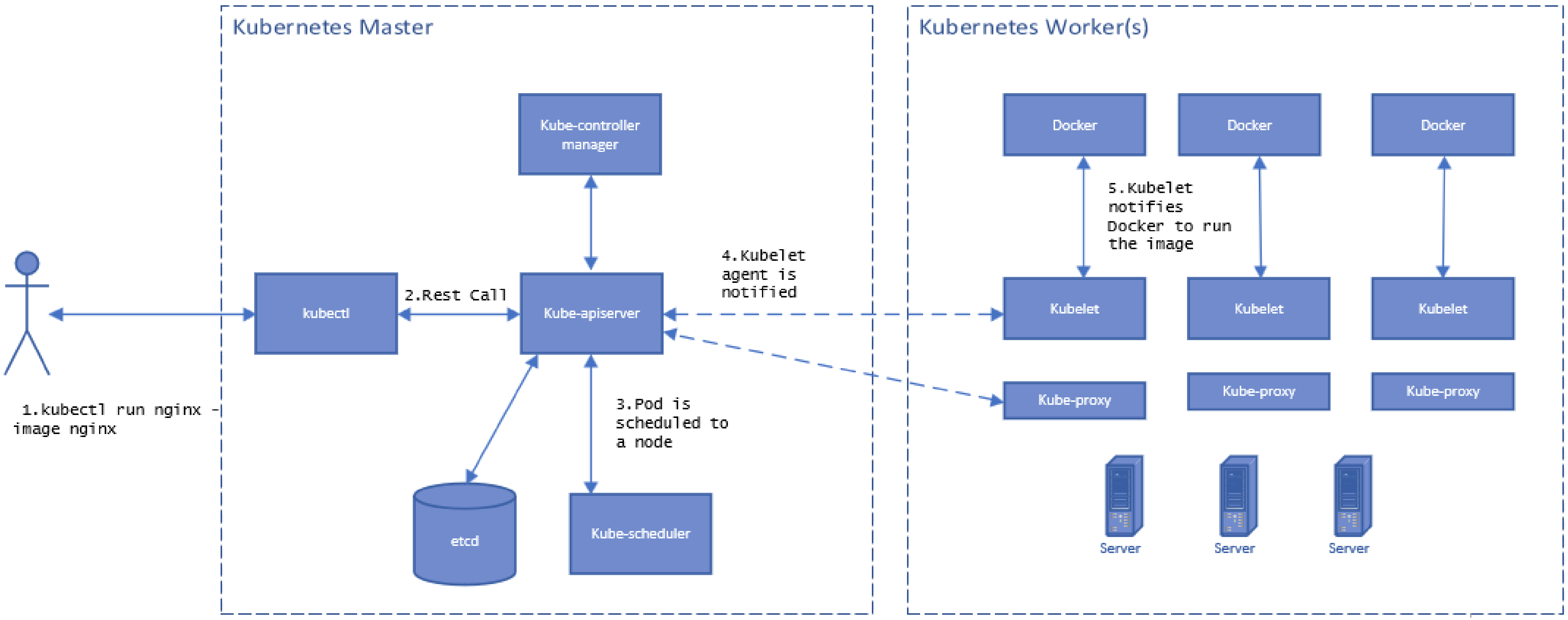
- Under the hood, Kubectl connects to the cluster through an underlying API
- Type of commands include…
  - Generic commands – manage Kubernetes objects
  - Cluster management commands
  - Troubleshooting commands
  - Deployment commands – deployment and scaling
  - Setting commands – configuration values

Demo: Kubernetes @ 10,000 feet

Kubernetes Building Blocks

- Nodes (Master + Worker)
- Docker
- Kubectl
- Pods
- Deployment
- Services
- Volumes
- Namespaces
- Labels and Selectors

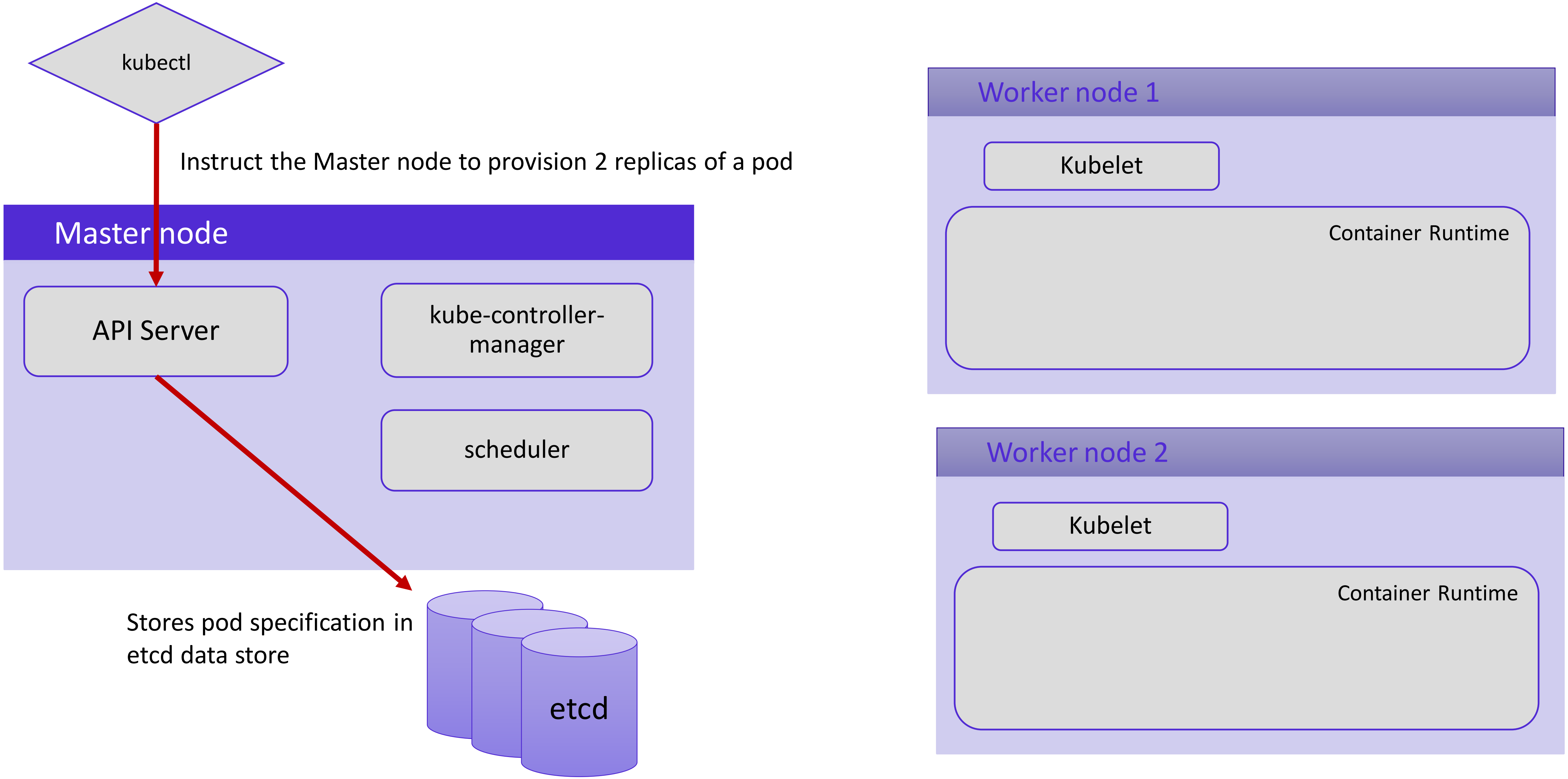
Overview of Cluster Components Communication



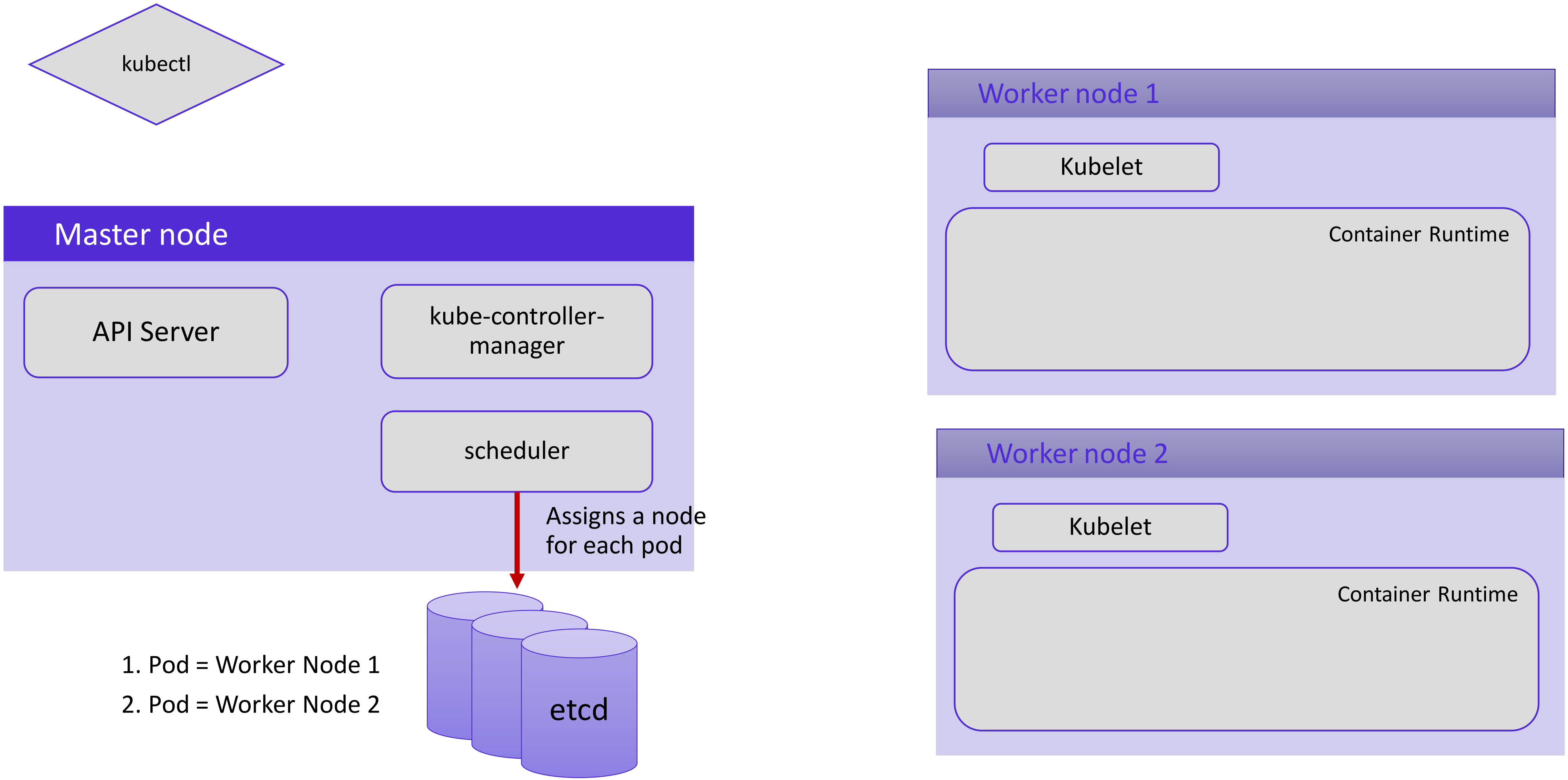
What are Pods?

- The POD is smallest building block in Kubernetes…
  - A collection of co-located containers and volumes
  - Running in the same execution environment
  - Managed as a single atomic unit
- You never directly run a container, instead you run a POD
- Apps running in a POD share the same IP, port and communicate using native interprocess communication channels
- Pods are immutable - if a change is made to a pod definition, a new pod is created, and the old pod is deleted
- PODs are identified using labels (key-value pairs)…
  - app=voting-app; tier= frontend

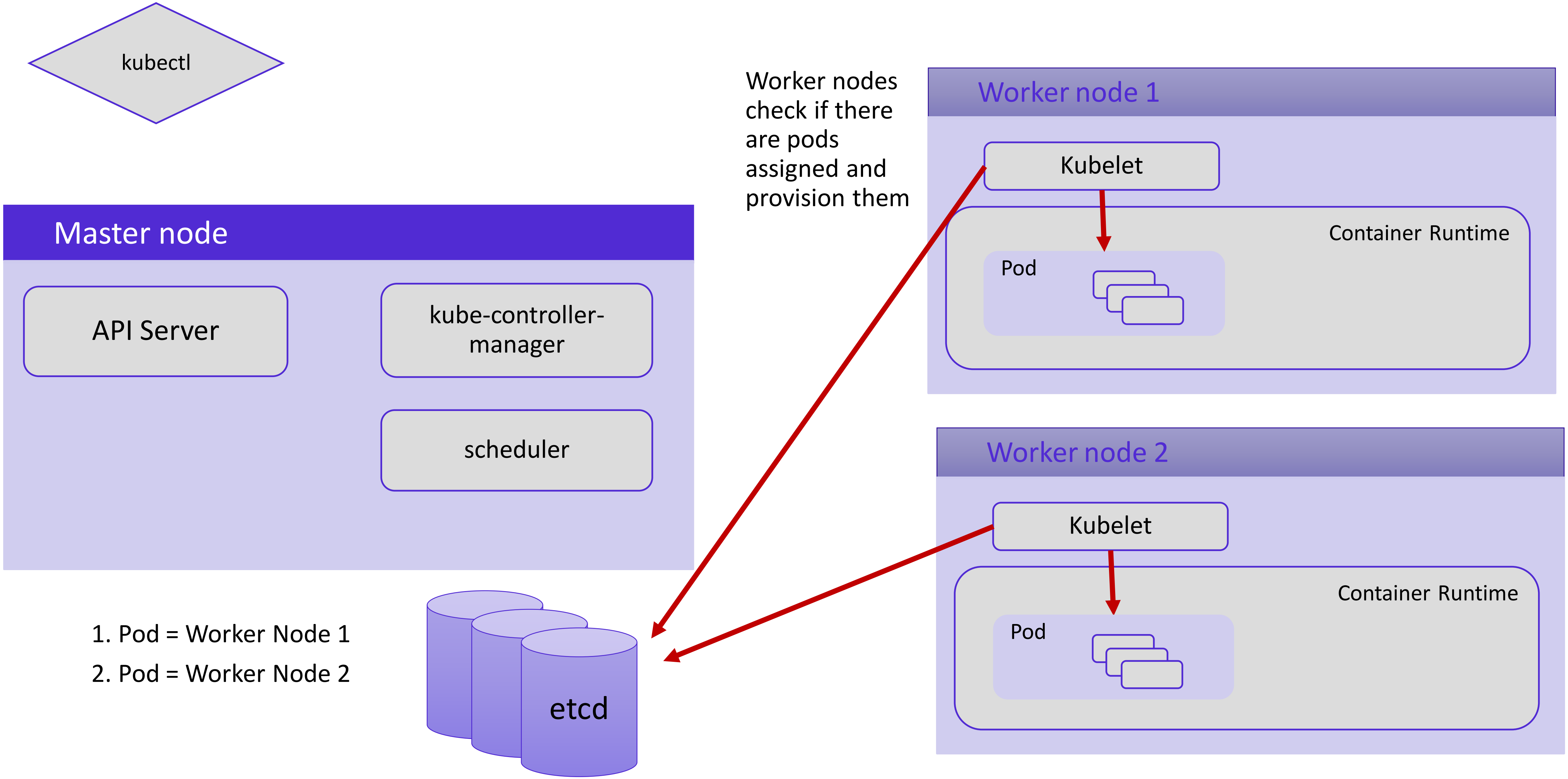
How Pods Work



How Pods Work



How Pods Work



kubectl

Master node

API Server

kube-controller-manager

scheduler

etcd

1. Pod = Worker Node 1
2. Pod = Worker Node 2

Worker nodes check if there are pods assigned and provision them

Worker node 1

Kubelet

Container Runtime

Pod

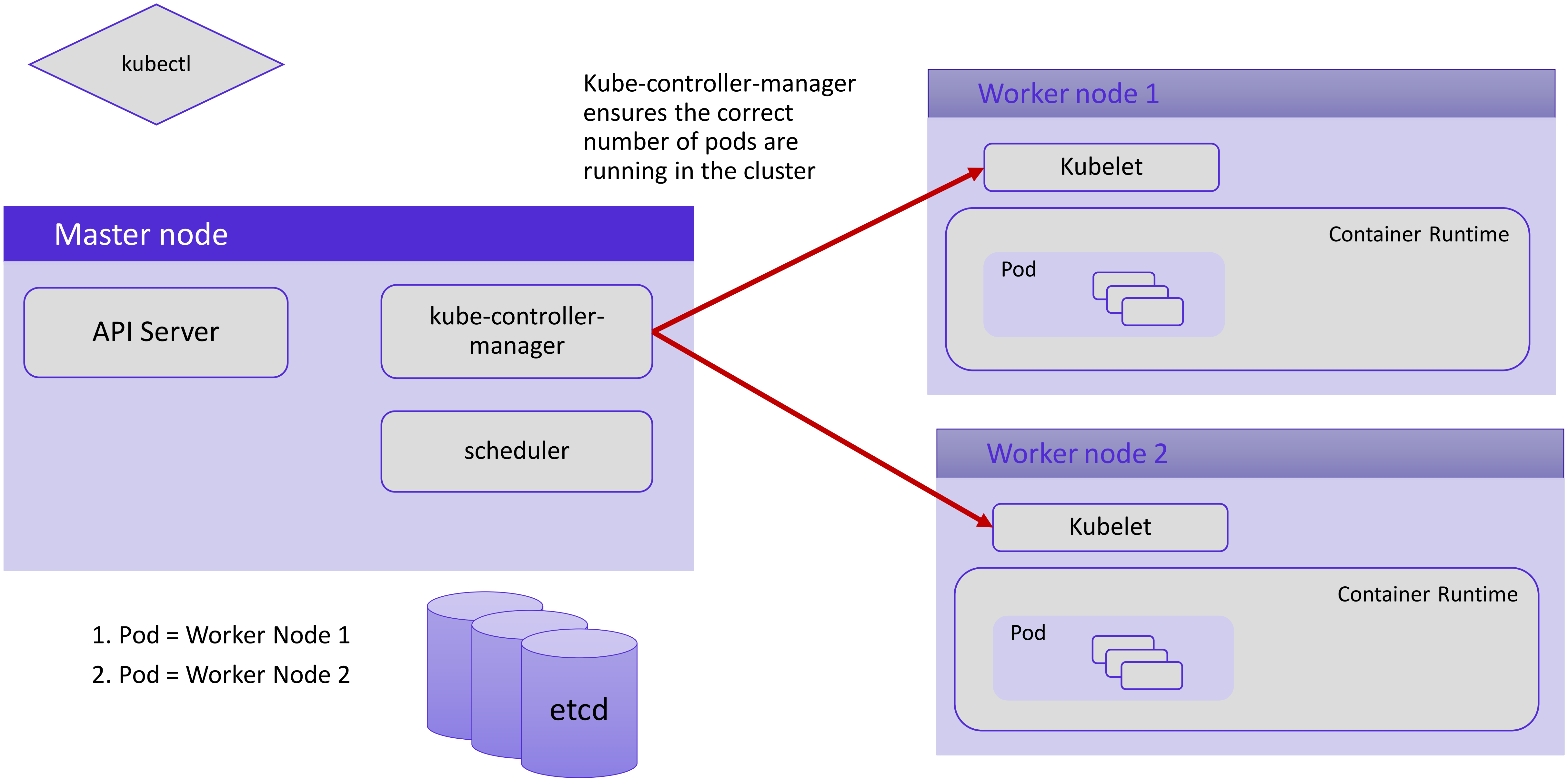
Worker node 2

Kubelet

Container Runtime

Pod

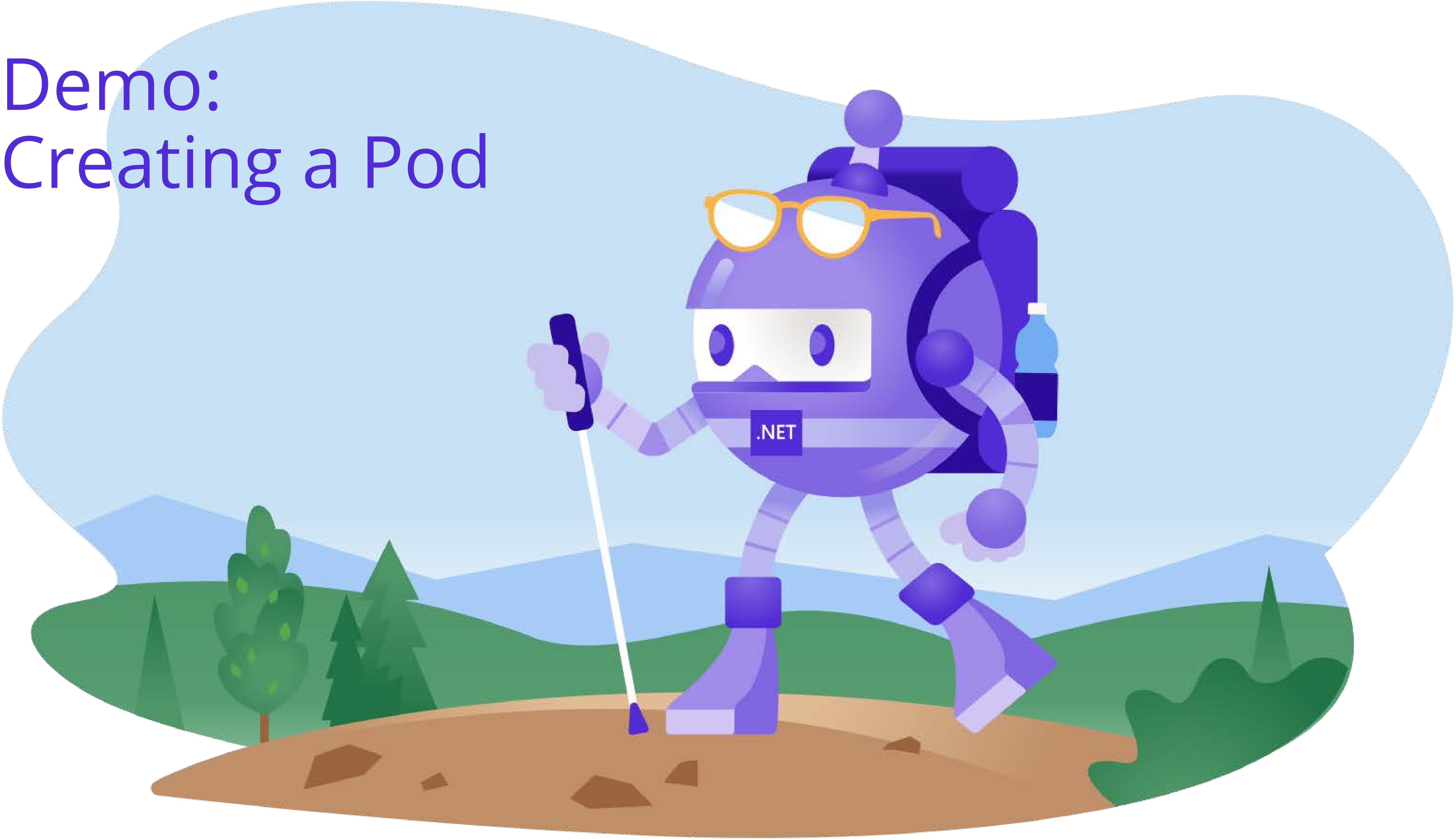
How Pods Work



Kube-controller-manager ensures the correct number of pods are running in the cluster

1. Pod = Worker Node 1
2. Pod = Worker Node 2

Demo:
Creating a Pod



Labels & Annotations

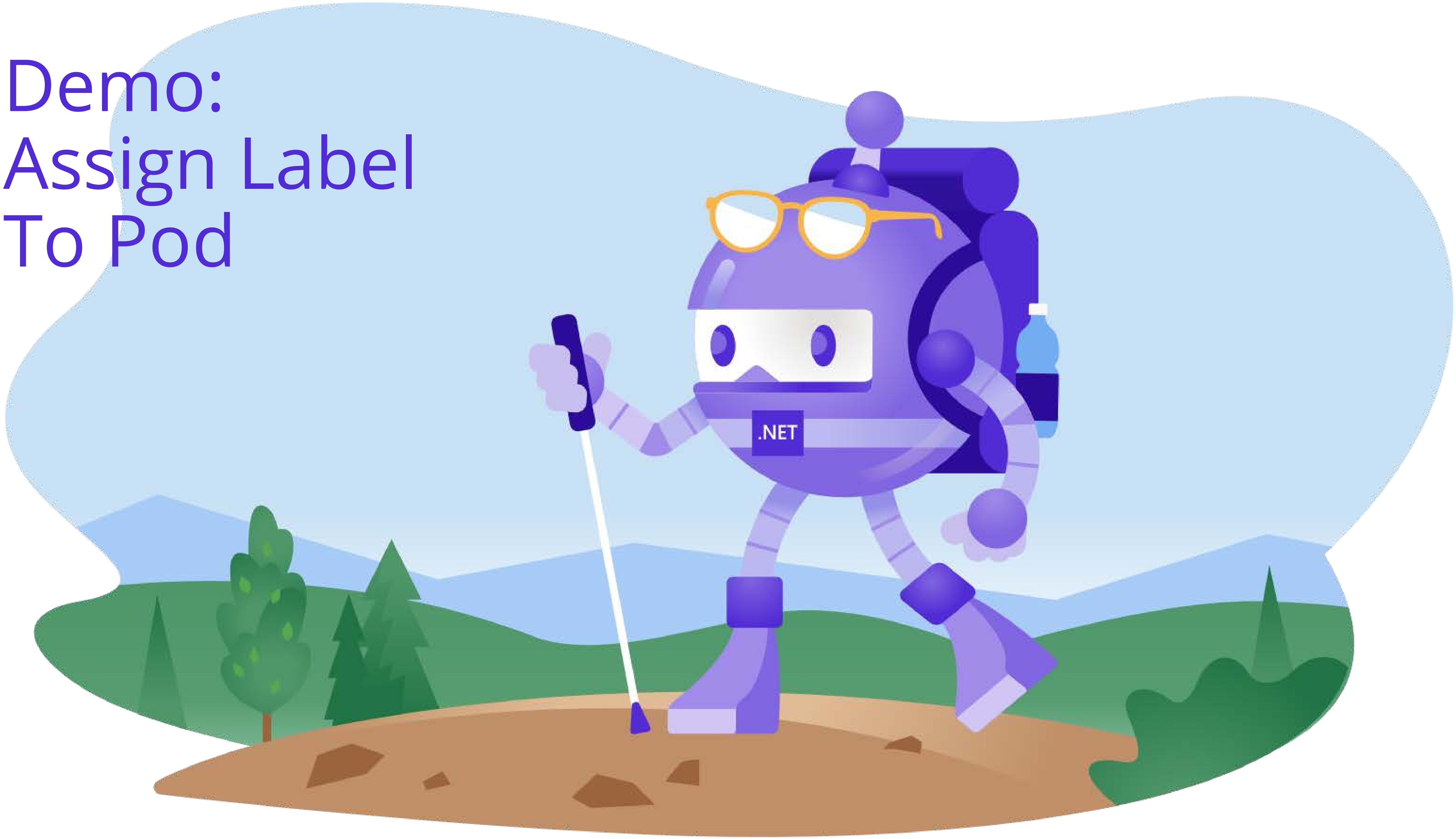
- A declarative way for which to identify Kubernetes objects
- Label – simple key-value pair
  - Mechanism to attach arbitrary but meaningful metadata to an object
  - Foundation for grouping Kubernetes objects: PODs, Deployments, Services
  - A Kubernetes object can have zero to many labels

Tier : Staging

- Annotations- similar to labels, also key-value pairs
  - Mechanism to share information with external tools and libraries
  - Other programs that drive Kubernetes can store information with an object
  - Often used to track roll-out and roll-back status

service.beta.kubernetes.io/azure-load-balancer-internal: "true“

Demo:
Assign Label
To Pod



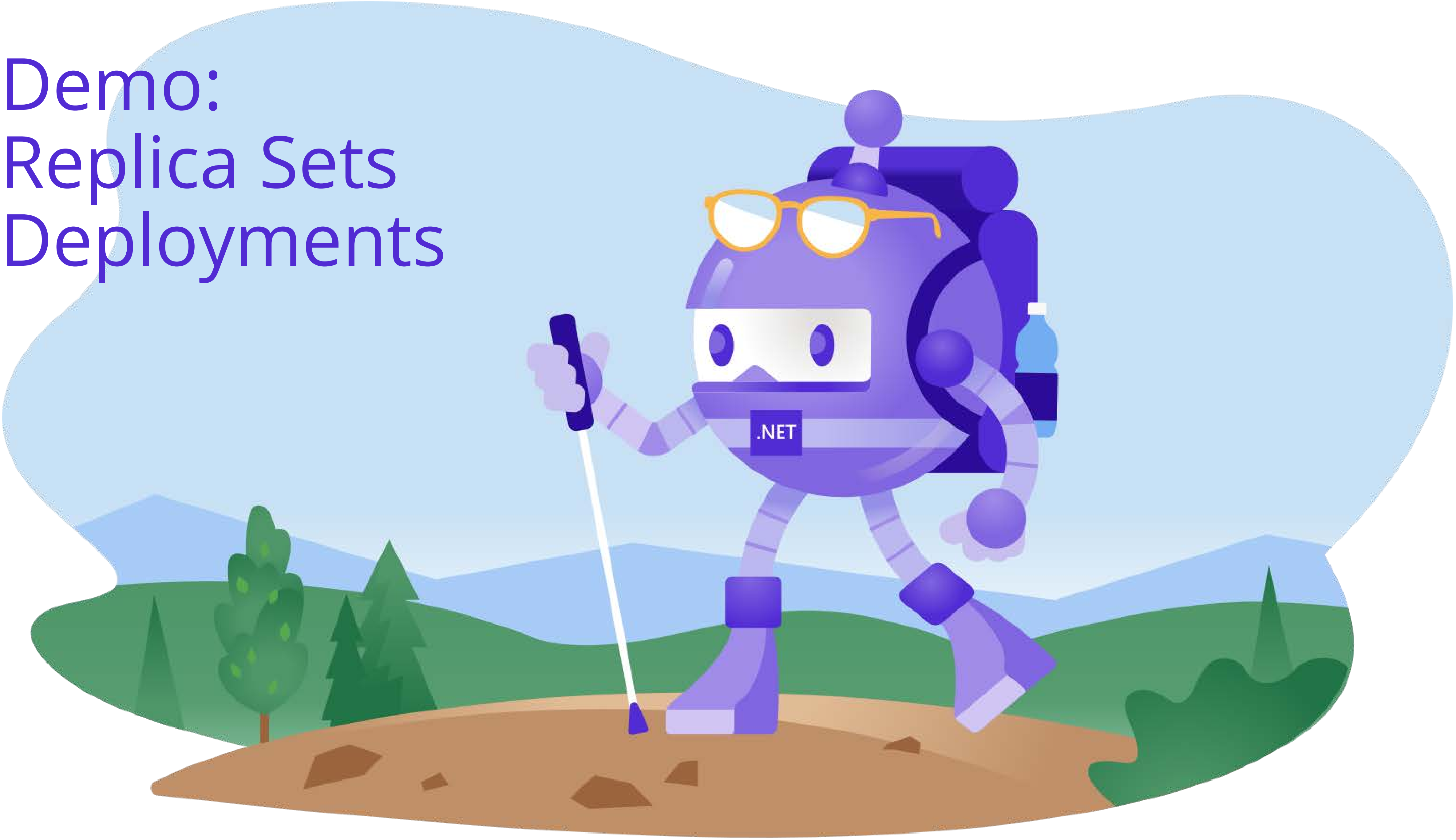
Replica Sets

- A POD is essentially a one-off singleton instance
- Typically, you'll need to expose multiple replicas of a container…
  - Redundancy – multiple instances allow for failure
  - Scale – multiple instances allow for more requests to be processed
- *Replica Sets* are a Kubernetes object that manage PODs
- They monitor the cluster and ensure the desired number of PODs are correctly running
  - If no PODs are provisioned, the Replica Set Controller will schedule them
  - If actual count drops below the desired, the controller will schedule replacements
  - If you exceed the desired count, the controller would destroy them
- Replica sets are automatically created by Kubernetes Deployment objects

What are Deployments Objects?

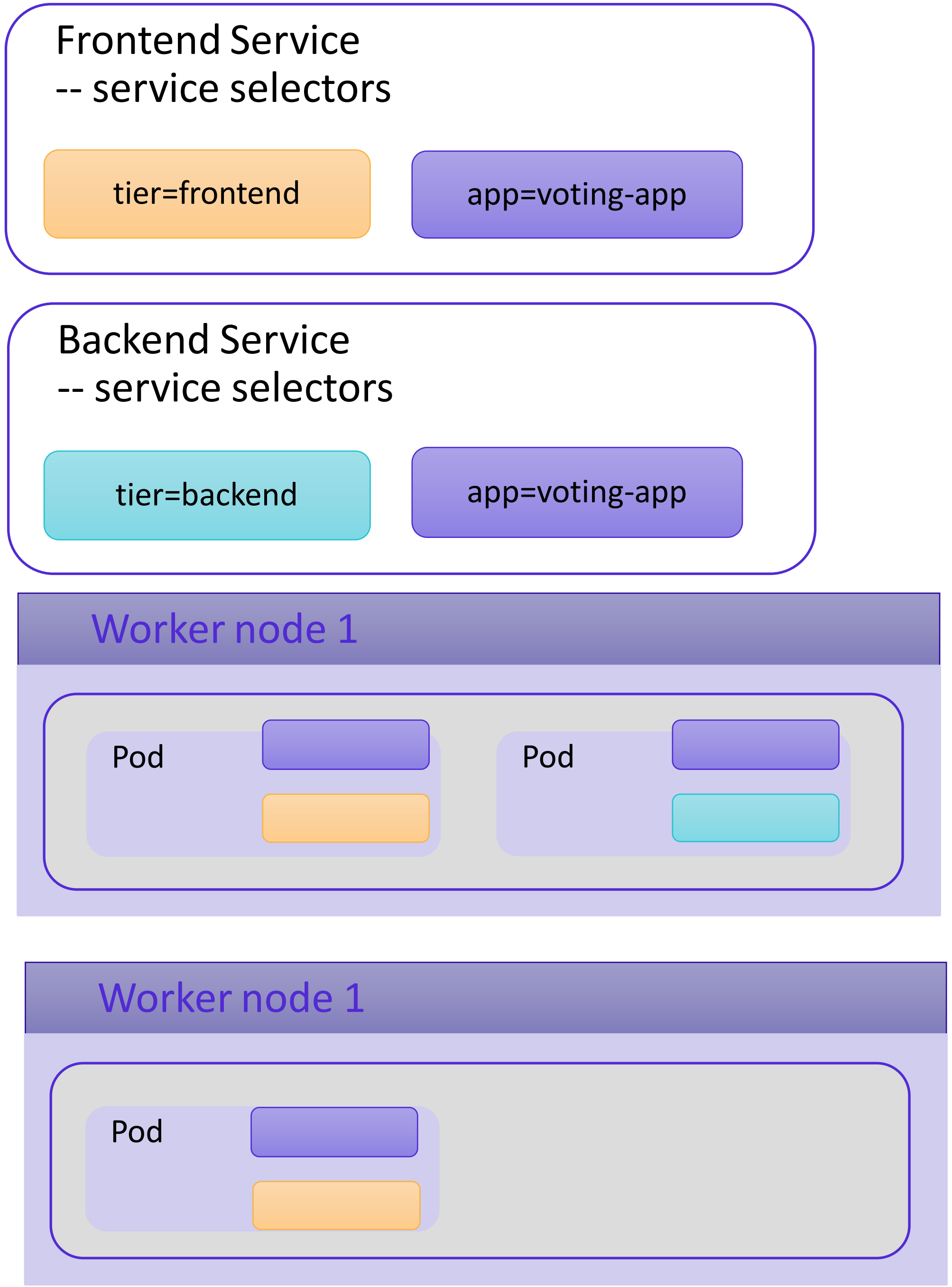
- A Kubernetes object that manages releases, updates and scaling…
  - Controls PODs and replica set objects
  - Instructs the Kubernetes Deployment controller how to create and update instances of your services
  - Provide fine-grained control over how and when a new pod version is rolled out as well as rolled back to a previous state
- The Kubernetes Deployment Controller reads the deployment and schedules the service instances onto individual nodes in the cluster
- Deployment objects abstract provisioning operations…
  - To delete a set of pods, simply delete the deployment that controls them
  - To update a set of pods, edit the deployment definition

Demo:
Replica Sets
Deployments



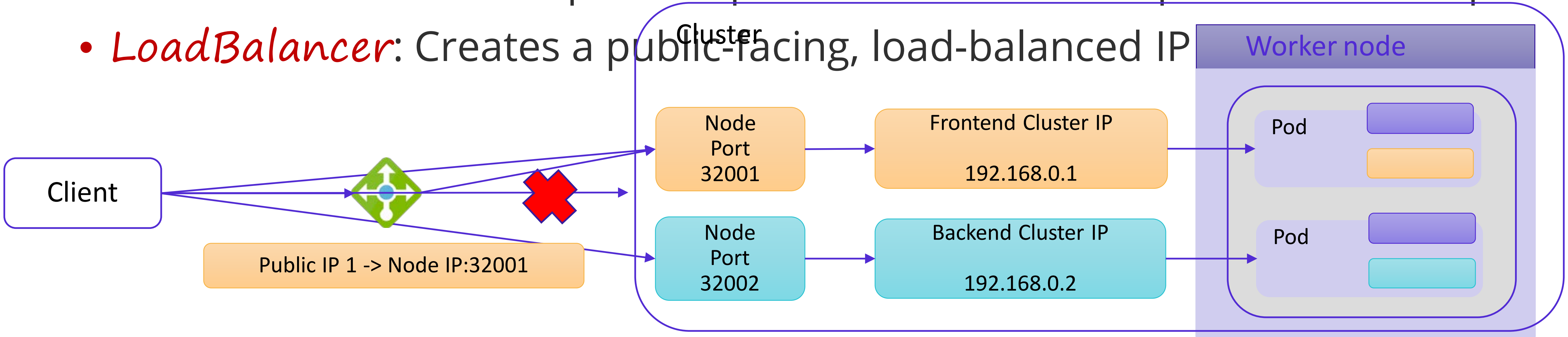
What is a Service?

- An abstraction that defines a logical set of loosely-coupled PODs and a policy by which to access them
  - A Services is defined with a YAML markup file
  - They use “selectors” to define which pods to represent
- Used to load balance traffic to your PODs
  - e.g. I want to expose the frontend of my voting app, so my service will expose all pods with the labels app=voting-app and tier=frontend



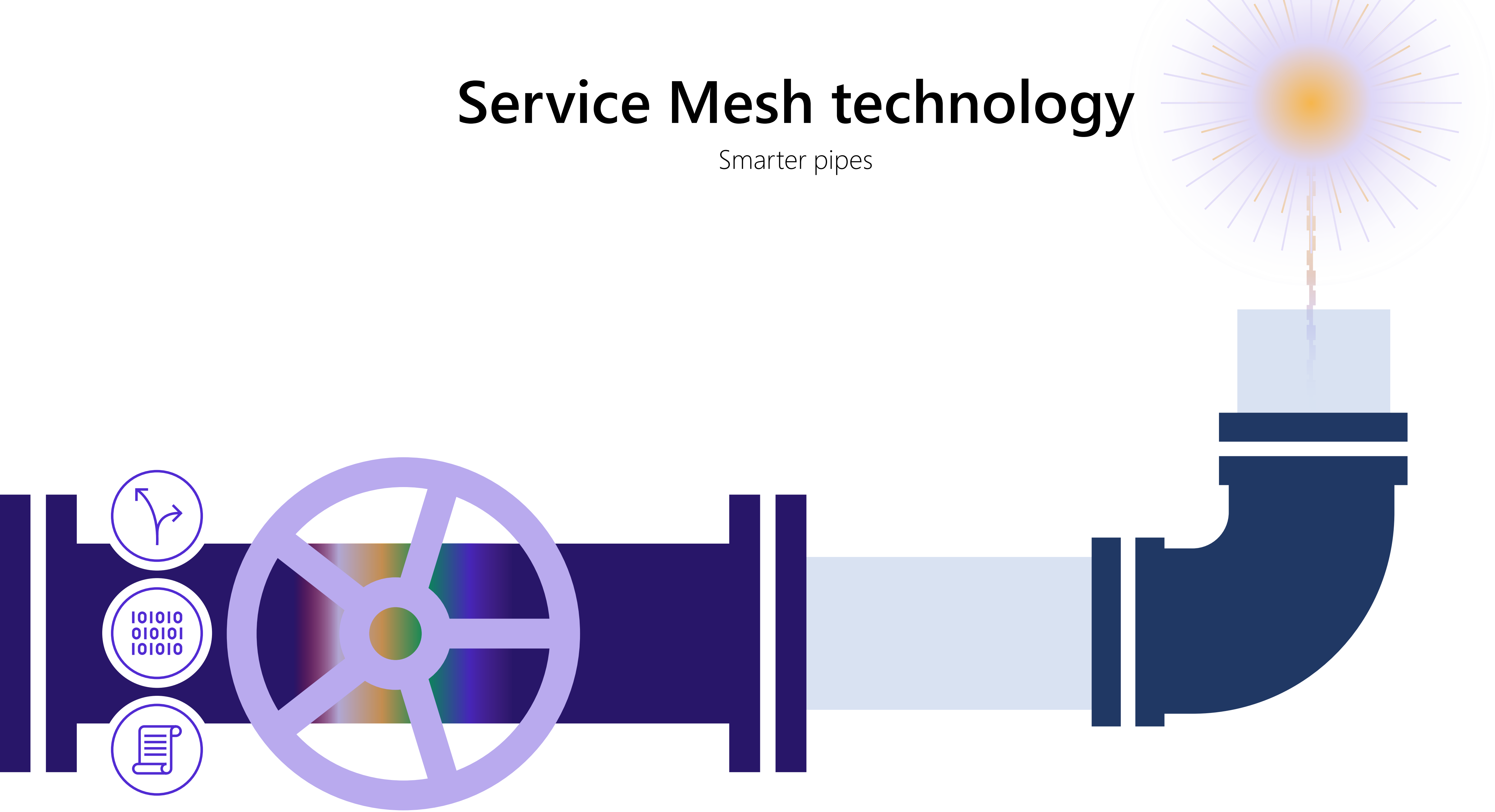
Service Object Types

- PODs are not exposed outside of the immediate cluster without a Service object – they allow PODs to receive traffic
- There are different types of services that expose your pod in different ways
  - *ClusterIP*: Provides a single IP internal to the cluster to represent a set of pods
  - *NodePort*: Reserves a specified port on the node to represent a set of pods
  - *LoadBalancer*: Creates a public-facing, load-balanced IP


Cluster
Worker node
Client
Node Port 32001
Frontend Cluster IP 192.168.0.1
Pod
Node Port 32002
Backend Cluster IP 192.168.0.2
Pod
Public IP 1 -> Node IP:32001

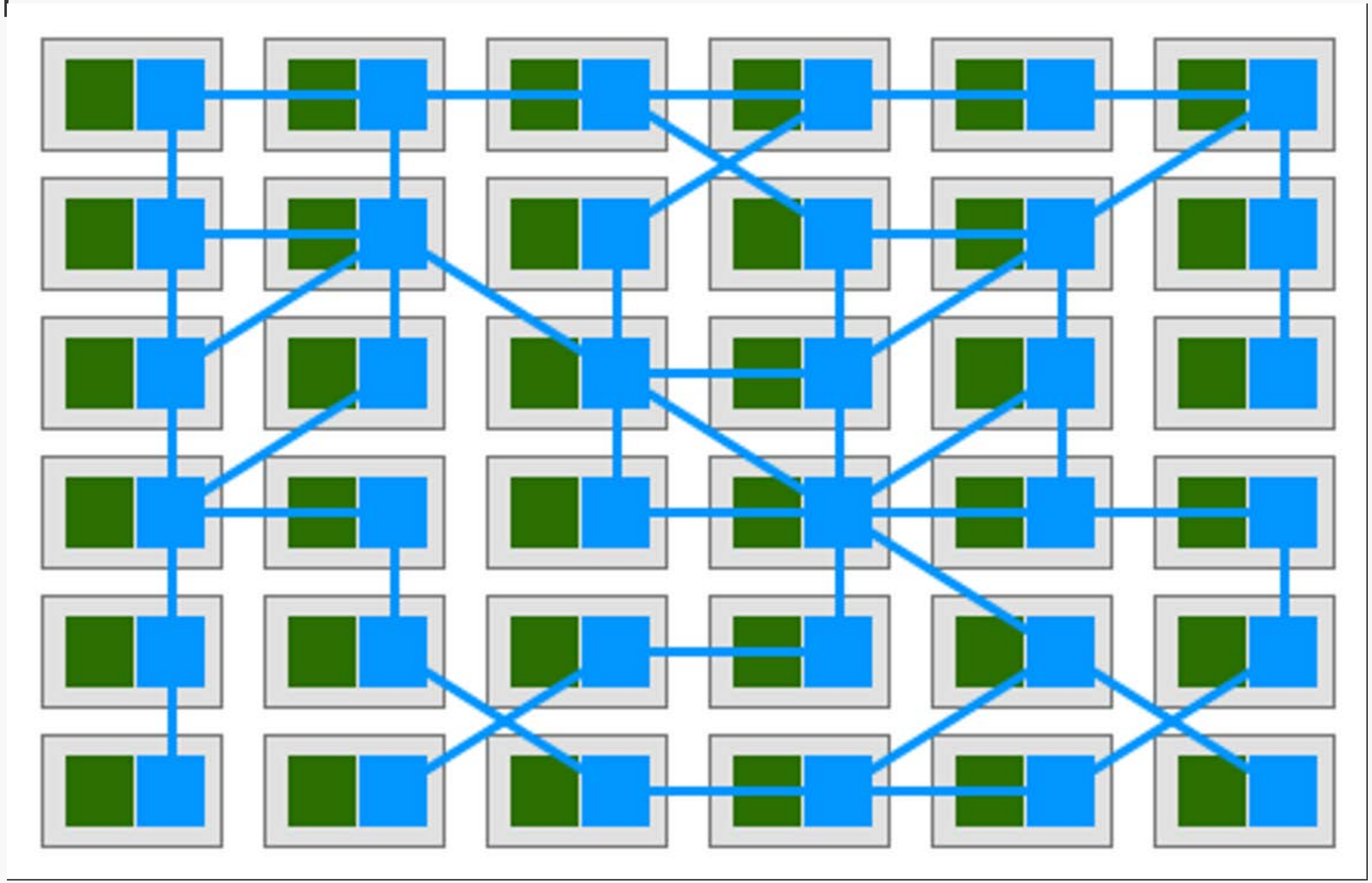
Service Mesh technology

Smarter pipes



Service Mesh Topology

- Proxies (blue) are isolated per service (green), but together form an interconnected (mesh) network through which traffic is routed and services communicate among each other



Service Mesh Components

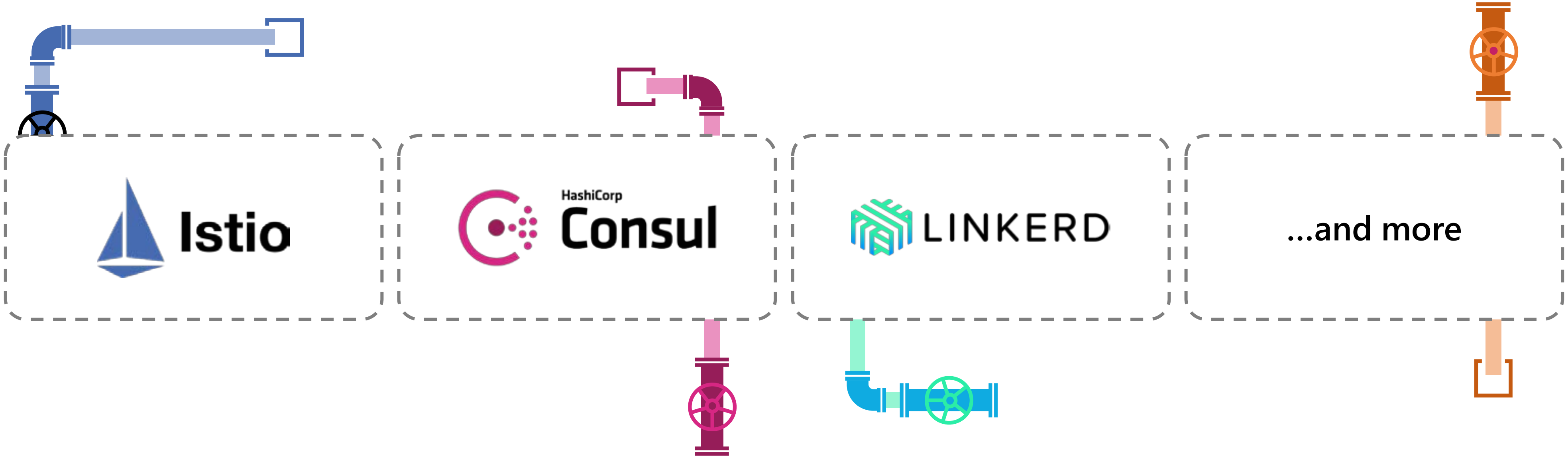
- A Service Mesh is composed of two disparate pieces
- The Data Plane
  - Routes network requests to service instances
  - Provides service discovery
  - Manages traffic, including resiliency such as Circuit breakers, retries, timeouts etc.
  - Load balancing to determine the specific service instance to which the request will be sent
  - Security, establishing a secure communication channel, managing authentication, authorization and encryption
- The Control Plane
  - Provides management and monitoring concerns
  - Provisions new instances
  - Probes and terminates unhealthy instances
  - Scales in and out, when required
  - Applies application-wide policies

What Can a Mesh Do?

- A mesh can apply dynamic routing rules to determine which service is being requested: Production, staging, on-prem, cloud?
- A mesh can retrieve a corresponding pool of instances from a service discovery endpoint
- A mesh will send the request to a specific instance, recording the latency and response type of the result
- A mesh can choose the instance most likely to return a fast response based on a variety of factors, including its observed latency for recent requests
- If an instance is unresponsive or fails, a mesh can retry the request on another instance
- If an instance consistently returns errors, a mesh can evict it from the load balancing pool, to be periodically retried later (for example, an instance may be undergoing a transient failure)
- If a request times-out, a mesh can fail and then retry the request
- A mesh captures the above behavior in the form of metrics and distributed tracing, which are emitted to a centralized metrics system

Service Mesh Platforms

- Linkerd (“linker-dee”)
  - First widely available system created by Buoyant
  - Based on early work from Twitter
  - Feature-rich and cross-platform
  - Written in Scala and runs on the JVM
- Envoy
  - From the engineering team at Lyft
  - Written in C++
  - Open source
- Istio
  - Open-source collaboration among Lyft, IBM, Google
  - Written in Go to be platform agnostic
  - Targets Kubernetes deployments



Istio

HashiCorp Consul

LINKERD

...and more

Service Mesh Interface (SMI) for Kubernetes

In partnership with

Microsoft

LINKERD

HashiCorp

solo.io

Red Hat

RANCHER

docker

weaveworks

ASPEN MESH

Pivotal

vmware

kinvolk

CANONICAL

kubecost

Demo: Kubernetes Services

What are ConfigMaps & Secrets

- ConfigMaps & Secrets are objects that allow you to expose information to your pods in the form of environment variables
- They contain a set of key value pairs that represent configuration and connection definitions for the application
- Secrets contain sensitive information that is encoded
  - e.g. Connection Strings, Passwords
- ConfigMaps contain non-sensitive information
  - e.g. the name of a backend component

Demo:
Config Maps
And Secrets

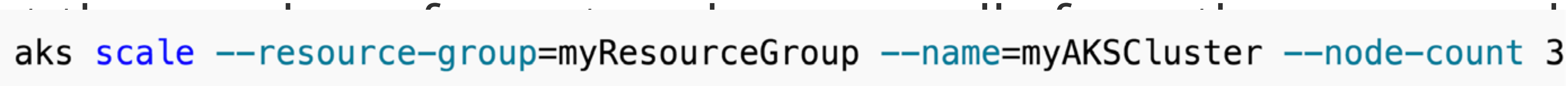
What are Volumes

- Volumes in Kubernetes are the same as in the traditional concept of volumes- they expose some sort of directory to the pod
- There are different types of volumes:
  - Volume- mounts a volume on the underlying node
  - Persistent Volume- defines a static, external volume (e.g. Azure file, Azure disk)
  - Persistent Volume Claim (PVC)- reserves a set of memory from a Persistent Volume

Demo: Volumes

AKS - Scaling Cluster

- **How do you add/remove nodes to the cluster to manage demand?**
- **Scale manually…**
  - Adju shell

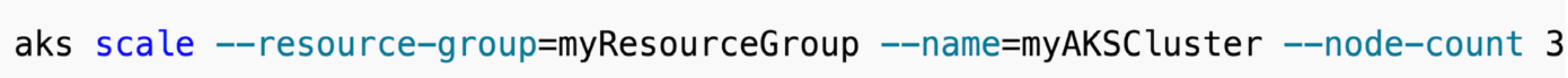


```
aks scale --resource-group=myResourceGroup --name=myAKSCluster --node-count 3
```

- **Autoscale (Preview)…**
  - Enables cluster to grow to meet that demand based on constraints you set
  - Scans the cluster periodically to check for pending pods or empty nodes and changes size if needed
  - Removes a node if it's unneeded for more than 10 minutes.

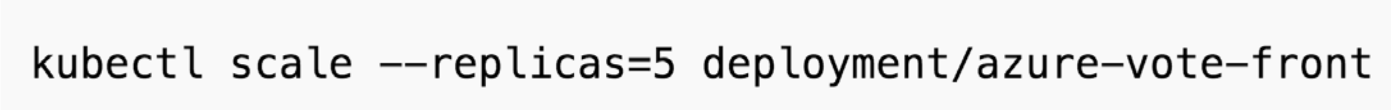
AKS - Scaling Pods

- **Manually scale pods…**
  - Change the replicas count to increase or decrease the number of pods



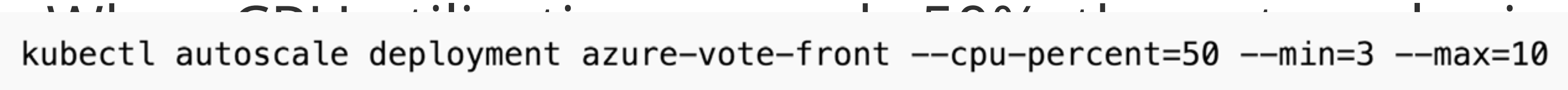
```
aks scale --resource-group=myResourceGroup --name=myAKSCluster --node-count 3
```

- **Autoscale pods…**
  - Kubernetes supports horizontal pod autoscaling to adjust the number of pods in a deployment depending on CPU utilization or other select metrics



```
kubectl scale --replicas=5 deployment/azure-vote-front
```

  - Kubectl autoscale command takes parameters to increase/decrease the pods
  - Example: When CPU utilization exceeds 50% the autoscaler increases the pods up to a maximum of 10 instances



```
kubectl autoscale deployment azure-vote-front --cpu-percent=50 --min=3 --max=10
```

Demo: Updating The Cluster

Module Summary

- Containers are a component of a Pod
- Pods are the atomic unit of a Kubernetes Cluster
- Objects like Deployments control the configuration and replication of pods
- Objects like services, secrets, configmaps, and volumes expose or connect the pod to other resources

Microsoft