Book one

Introduction to Kubernetes

Creation date
09/29/2020

Revision date
09/29/2020

Corporate Headquarters
Microsoft Corporation
1 Microsoft Way,
Redmond, WA 98052
www.microsoft.com

Submission/Revision History

Revision	Authors	Released date	Comments
1.0	Mauricio Zaragoza	09/29/2020	Initial release

Technical Review History

Examine	Examiner (s)	Date	Comments
1.0 Within group	Reggie Snead Bhavin Doshi	09/29/2020	Comments
2.0 Outside group		09/29/2020	Comments

Table of contents

Introduction to Kubernetes

Determine the types of business problems that you can solve by using Kubernetes. Describe the benefits of container orchestration with features like deployment management, automatic updates, and self-healing.

Learning Objectives

In this module, you will:

- Describe how Kubernetes supports container orchestration
- Describe the difference between control planes and nodes
- Evaluate whether Kubernetes is an appropriate orchestration platform for a given workload

Prerequisites

- Basic concepts of microservice architectures
- Basic understanding of how Docker containers work
- Basic knowledge of how to install software by using a command-line interface

Definitions

Table 1: Terms and Acronyms used in this document

Term	Acronym	Definition	Comments

Conventions

Table 2: Key Terms and Symbols used in this document

Term	Definition
Note	Notes give you additional information that will help you obey the instructions written in the procedural steps.
▭	Red squares or rectangles shows you where to enter and / or to examine the necessary information (data, IP addresses, etc.).

Bold	Text written in bold in procedural steps indicate data that must be typed and / or menu selections that must selected or clicked on with the mouse pointer.
→	Red arrows indicate where to point and click your mouse pointer. Reference to these in screenshots and other parts of the document.

Document updates

This document contains information about Microsoft Azure and other Microsoft technologies.
Due to the dynamics of the modern world, these technologies are evolving and changing all the time.

Microsoft is adding and deprecating features to Azure every day, increasing quotas, changing limits, releasing new products, changing, and improving architectures, etcetera.

In general, Microsoft is modifying the characteristics of the Azure services to accommodate customers' requirements and needs quite often. For this reason, the details about the features and products contained in this document might be enhanced or changed at some point in time.

Most of the concepts that are outlined in this guide are general concepts that are not expected to change drastically soon.

You need to be aware of the dynamics of Azure and use this document as reference. Please complement the information in this guide with the most current Azure documentation that is available online.

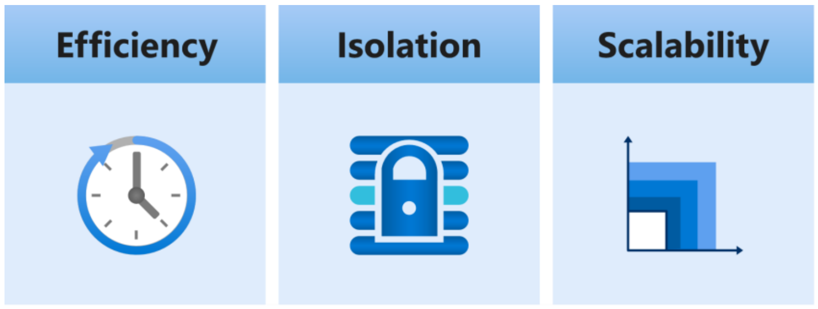
Values, limits, names, capabilities, features, costs, billing models, regions, etc. are valid at the time this document was published.

The document release date is in the **Revision** section, and a comprehensive list of online resources are published in the **References** section. These 2 portions need to be modified accordingly when Microsoft updates any of the functionalities described in this white paper.

Azure

Introduction

Containers are an excellent choice when developing software based on microservice architectures. They make efficient use of hardware, provide security features to run multiple instances simultaneously on the same host without affecting each other, and allow a service to be scaled out by deploying more instances.

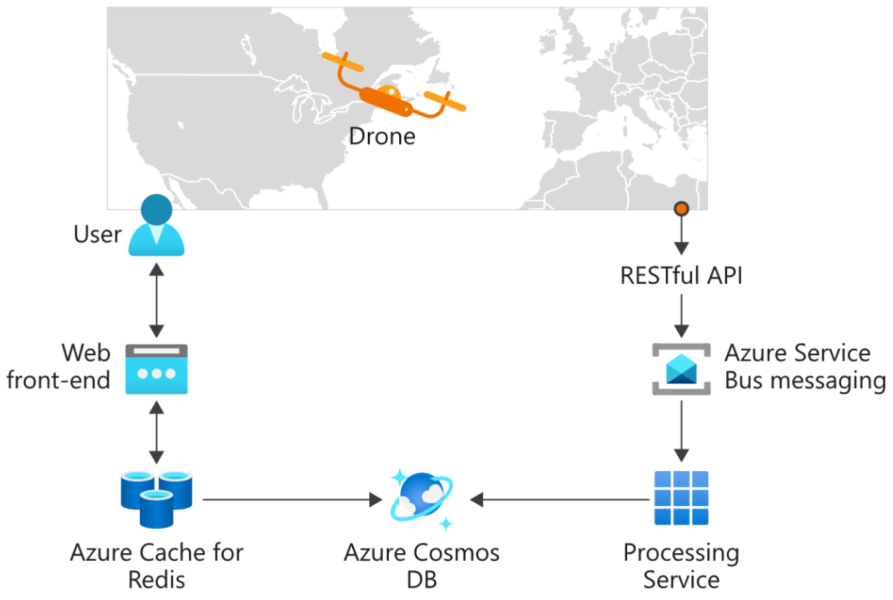


The standard container management runtime focuses on managing individual containers. If you want to scale a complex system with multiple containers working together, it will become challenging. Consider the following aspects you'll have to take care of:

- Configure and maintain load balancing
- Network connectivity
- Orchestration of the deployment process

It's common to use a **container management platform** such as **Kubernetes** to make the management process easier.

Suppose you work at a drone management company. Your company provides a drone tracking solution to customers worldwide. The solution is built and deployed as microservices and includes several major applications:



- Web front end: shows maps and information about tracked drones.
- Cache service: stores frequently requested information displayed on the website.
- RESTful API: used by tracked drones to send data about their status, such as a GPS location and battery charge levels.
- Queue: holds unprocessed data collected by the RESTful API.
- Data processing service: fetches and processes data from the queue.
- NoSQL database: stores processed tracking data and user information captured from the website and the data processing service.

You're using containerized instances to quickly deploy into new customer regions and scale resources as needed to meet customer demands. To simplify the development, deployment, and management of these complex containerized applications you want to use a container orchestration platform.

Learning objectives

In this module, you will:

- Describe how Kubernetes supports container orchestration
- Describe the difference between control planes and nodes
- Evaluate whether Kubernetes is an appropriate orchestration platform for a given workload

Prerequisites

- Basic concepts of microservice architectures
- Basic understanding of how Docker containers work
- Basic knowledge of how to install software by using a command-line interface

Part 1

What is Kubernetes?

What is Kubernetes?

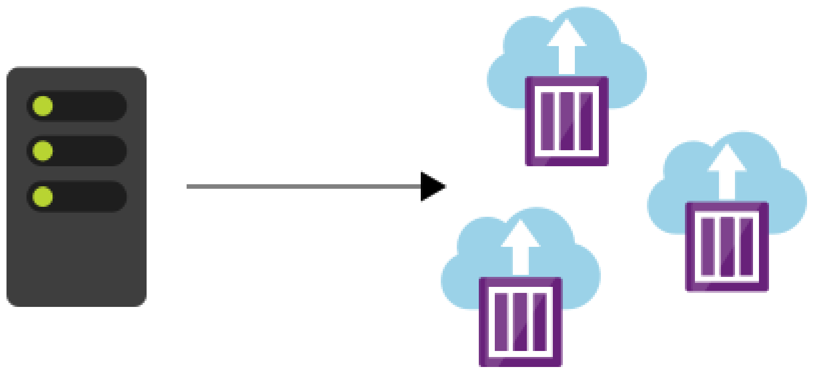
The decoupled design of microservices combined with the atomicity of containers make it possible to scale out applications and respond to increased demand by deploying more container instances and to scale back in if demand is decreasing. In complex solutions like the drone tracking app, the process of deploying, updating, monitoring, and removing containers introduces challenges.

Before we look at what Kubernetes is, let's summarize a few concepts that are key to containerized workloads.

What is container management?

Container management is the process of organizing, adding, removing, or updating a significant number of containers.

The drone tracking application consists of multiple microservices, responsible for tasks like caching, queuing, or data processing. Each of these services is hosted in a container and can be deployed, updated, and scaled independently from the other.



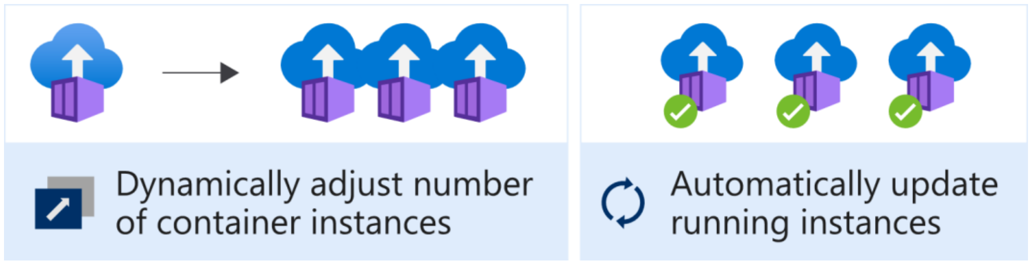
For example, with the drone tracking app's website, you find that at specific times during the day, you need more instances of the site's caching service to keep performance up, so you add more caching service container instances.

Now assume, that you've increased the number of caching instances and need to roll out a new version of the microservice. You'll have to make sure to update **all** the active containers.

Container management helps you with these otherwise manual tasks.

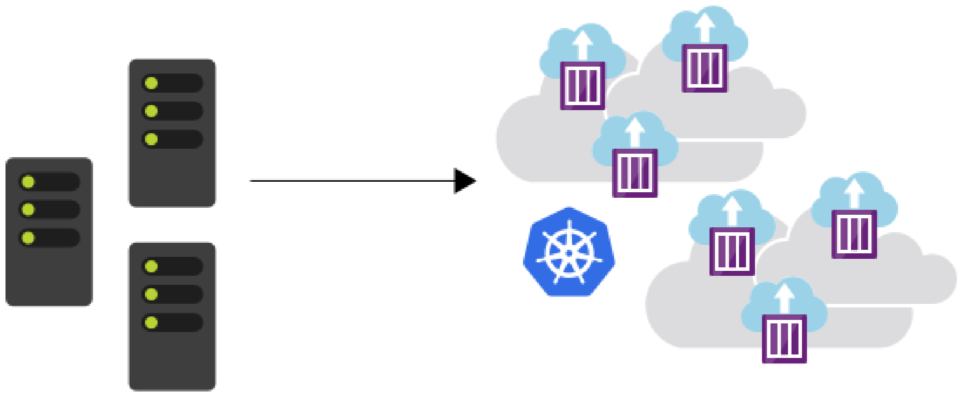
What is container orchestration?

A container orchestrator is a system that automatically deploys and manages containerized applications. For example, the orchestrator can dynamically respond to changes in the environment to increase or decrease the deployed instances of the managed application. Or, it can ensure all deployed container instances get updated if a new version of a service is released.


Dynamically adjust number of container instances
Automatically update running instances

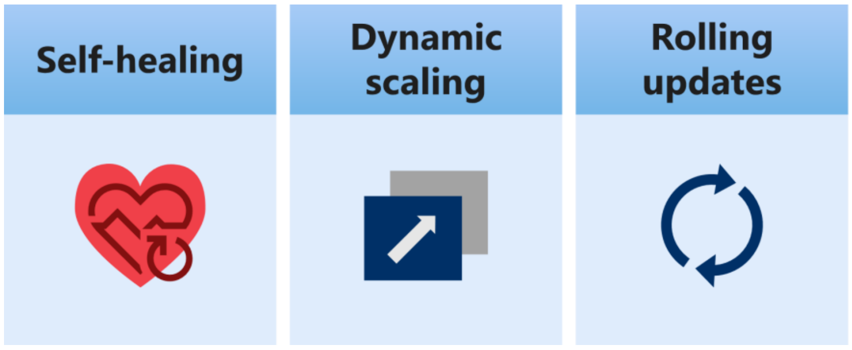
Define Kubernetes

Kubernetes is a portable, extensible open-source platform for managing and orchestrating containerized workloads. Kubernetes abstracts away complex container management tasks and provides you with declarative configuration to orchestrate containers in different computing environments. This orchestration platform gives you the same ease of use and flexibility you may already know from platform as a service (PaaS) or infrastructure as a service (IaaS) offerings.



Kubernetes benefits

The benefits of using Kubernetes are based on the abstraction of tasks.



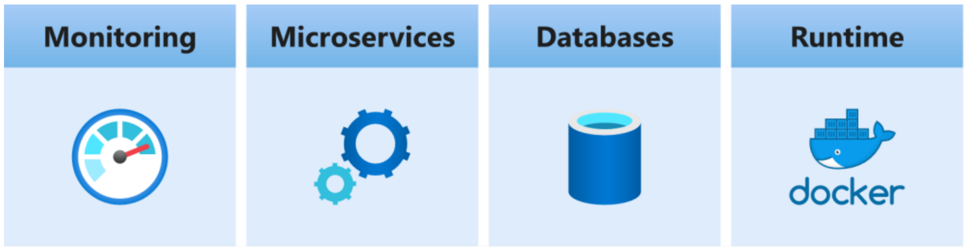
These tasks include:

- Self-healing of containers. Example would be restarting containers that fail or replacing containers.
- Scaling deployed container count up or down dynamically based on demand.
- Automation of rolling updates and rollbacks of containers.
- Management of storage.
- Management of network traffic.
- Storage and management of sensitive information such as usernames and passwords.

> ⓘ Important
>
> Keep in mind that all of the preceding aspects of Kubernetes require configuration and a good understanding of the underlying technologies. For example, you need to understand concepts such as virtual networks, load balancers, and reverse proxies to configure Kubernetes networking.

Kubernetes considerations

With Kubernetes, you can view your datacenter as one large compute resource. You don't worry about how and where you deploy your containers, only about deploying and scaling your applications as needed.



However, it's important to understand that Kubernetes isn't a single installed application that comes with all possible components needed to manage and orchestrate a containerized solution.

- Aspects such as deployment, scaling, load balancing, logging, and monitoring are all optional. You're responsible for finding the best solution that fits your needs to address these aspects.
- Kubernetes doesn't limit the types of applications that can run on the platform. If your application can run in a container, it can run on Kubernetes. To make optimal use of containerized solutions, your developers need to understand concepts such as microservices architecture.
- Kubernetes doesn't provide middleware, data-processing frameworks, databases, caches, or cluster storage systems. All these items are run as containers or as part of another service offering.
- For Kubernetes to run containers, it needs a container runtime like Docker. The container runtime is the object that's responsible for managing containers. For example, the container runtime starts, stops, and reports on the container's status.
- You're responsible for maintaining your Kubernetes environment. For example, you need to manage OS upgrades and the Kubernetes installation and upgrades. You also manage the hardware configuration of the host machines, such as networking, memory, and storage.

Cloud services, such Azure Kubernetes Service (AKS), reduce these challenges by providing a hosted Kubernetes environment. These services also simplify the deployment and management of containerized applications in Azure. With AKS, you get the benefits of open-source Kubernetes without the complexity or operational overhead of running your own custom Kubernetes cluster.

> Note
>
> Kubernetes is sometimes abbreviated to K8s. The 8 represents the eight characters between the K and the s of the word K[ubernete]s.

Part 2

How Kubernetes works

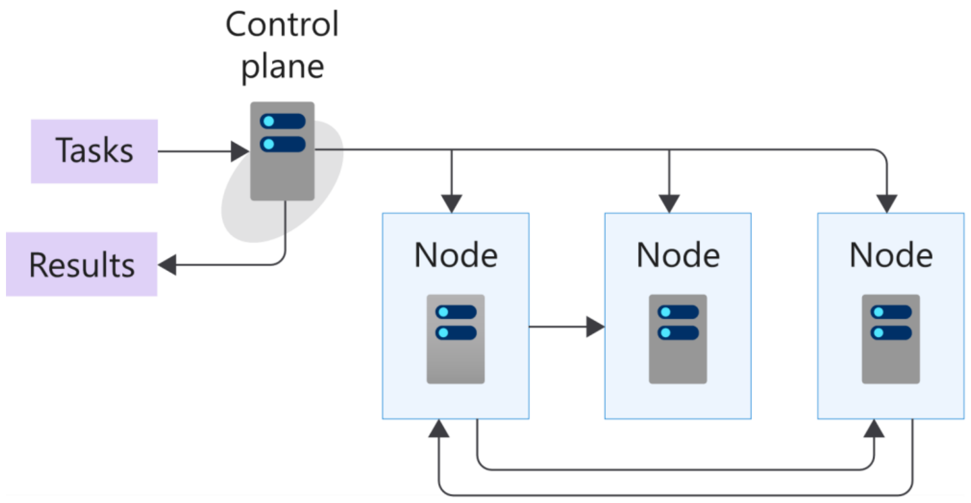
How Kubernetes works

A successfully configured Kubernetes installation depends on a good understanding of the Kubernetes system architecture. Here you'll look at all the components that make up a Kubernetes installation.

What is a computer cluster?

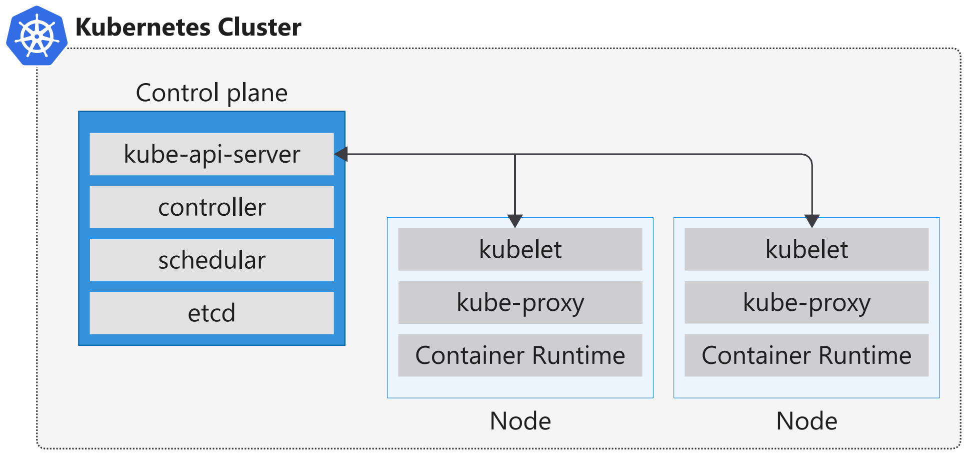
A cluster is a set of computers that you configure to work together and view as a single system. The computers configured in the cluster will typically do the same kinds of tasks. For example, they'll all host websites, APIs, or run compute-intensive work.

A cluster uses centralized software that's responsible for scheduling and controlling these tasks. The **computers in a cluster that run the tasks are called nodes**, and the **computers that run the scheduling software are called control planes**.



Kubernetes architecture

Recall from earlier that an orchestrator is a system that deploys and manages applications. You also learned a cluster is a set of computers that work together and are viewed as a single system. You use Kubernetes as the orchestration and cluster software to deploy your apps and respond to changes in compute resource needs.



A Kubernetes cluster contains at least one main node and one or more nodes. Both the control planes and node instances can be physical devices, virtual machines, or instances in the cloud. The default host operating system in Kubernetes is Linux, with default support for Linux-based workloads.

It's also possible to run Microsoft workloads by using Windows Server 2019 or later on cluster nodes. For example, assume that the data processing service in the drone tracking application is written as a .NET 4.5 application that uses specific Windows OS API calls. This service can run only on nodes that run a Windows Server OS.

Let's look at both the control planes and worker nodes, and the software that runs on each, in more detail. Understanding the role of each component and where each component runs in the cluster helps you when it comes to installing Kubernetes.

Kubernetes control plane

The Kubernetes control plane in a Kubernetes cluster runs a collection of services that manage the orchestration functionality in Kubernetes.

From a learning perspective, it makes sense to use a single control plane in your test environment as you explore Kubernetes functionality. However, in production and cloud deployments such as Azure Kubernetes Service (AKS), you'll find that the preferred configuration is a high-availability deployment with three to five replicated control planes.

ⓘ Note

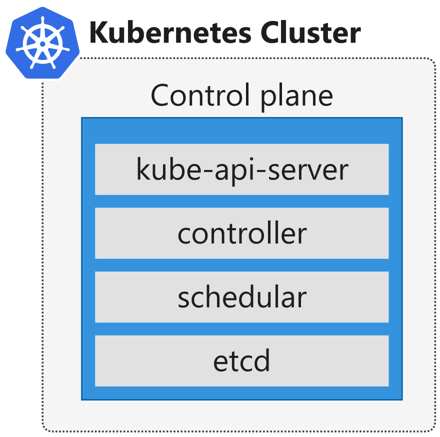
The fact that a control plane runs specific software to maintain the state of the cluster doesn't exclude it from running other compute workloads. However, you usually want to exclude the control plane from running noncritical and user application workloads.

Kubernetes node

A node in a Kubernetes cluster is where your compute workloads run. Each node communicates with the control plane via the API server to inform it about state changes on the node.

Services that run on a control plane

Kubernetes relies on several administrative services running on the control plane. These services manage aspects such as cluster component communication, workload scheduling, and cluster state persistence.



The following services make up the control plane for a Kubernetes cluster:

- API server
- Backing store
- Scheduler
- Controller manager
- Cloud controller manager

What is the API server?

You can think of the API server as the front end to the control plane in your Kubernetes cluster. All the communication between the components in Kubernetes is done through this API.

For example, as a user, you use a command-line application called `kubectl` that allows you to run commands against your Kubernetes cluster's API server. The component that provides this API is called `kube-apiserver`, and you can deploy several instances of this component to support scaling in your cluster.

This API exposes a RESTful API that you can use to post commands or YAML-based configuration files. YAML is a human-readable data serialization standard for programming languages. You use YAML files to define the intended state of all the objects within a Kubernetes cluster.

For example, assume that you want to increase the number of instances of your application in the cluster. You'll define the new state by using a YAML-based file and submit this file to the API server. The API server will validate the configuration, save it to the cluster, and finally enact the configured increase in application deployments.

What is the backing store?

The backing store is a persistence store that your Kubernetes cluster uses to save the complete configuration of a Kubernetes cluster. Kubernetes uses a high-availability, distributed, and reliable key-value store called `etcd`. This key-value store stores the current state and the desired state of all objects within your cluster.

In a production Kubernetes cluster, the official Kubernetes guidance is to have three to five replicated instances of the `etcd` database for high availability.

> Note
>
> `etcd` isn't responsible for data backup. It's your responsibility to ensure that an effective backup plan is in place to back up the `etcd` data.

What is the scheduler?

The scheduler is the component that's responsible for the assignment of workloads across all nodes. The scheduler monitors the cluster for newly created containers and assigns them to nodes.

What is the controller manager?

The controller manager is responsible for launching and monitoring the controllers configured for a cluster through the API server.

Kubernetes uses controllers to track the state of objects in the cluster. Each controller runs in a non-terminating loop while watching and responding to events in the cluster. For example, there are controllers to monitor nodes, containers, and endpoints.

The controller communicates with the API server to determine the state of the object. If the current state is different from the wanted state of the object, then the controller will take action to ensure the wanted state.

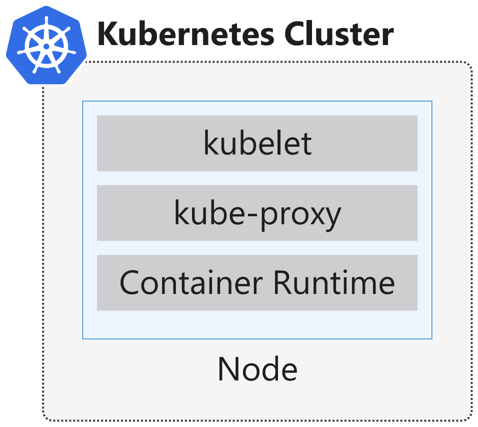
Let's assume that one of three containers running in your cluster stops responding and has died. In this case, a controller decides whether you need to launch new containers to ensure that your applications are always available. If the wanted state is to run three containers at any time, then a new container is scheduled to run.

What is the cloud controller manager?

The cloud controller manager integrates with the underlying cloud technologies in your cluster when the cluster is running in a cloud environment. These services can be load balancers, queues, and storage, for example.

Services that run on a node

The are several services that run on a Kubernetes node to control how workloads run.



The following services run on the Kubernetes node:

- Kubelet
- Kube-proxy
- Container runtime

What is the kubelet?

The kubelet is the agent that runs on each node in the cluster and monitors work requests from the API server. It makes sure that the requested unit of work is running and healthy.

The kubelet monitors the nodes and makes sure that the containers scheduled on each node run as expected. The kubelet manages only containers created by Kubernetes. It isn't responsible for rescheduling work to run on other nodes if the current node can't run the work.

What is kube-proxy?

The kube-proxy component is responsible for local cluster networking and runs on each node. It makes sure that each node has a unique IP address. It also implements rules to handle routing and load balancing of traffic by using iptables and IPVS. This proxy doesn't provide DNS services by itself. A DNS cluster add-on based on CoreDNS is recommended and installed by default.

What is the container runtime?

The container runtime is the underlying software that runs containers on a Kubernetes cluster. The runtime is responsible for fetching, starting, and stopping container images. Kubernetes supports several container runtimes, including but not limited to Docker, rkt, CRI-O, containerd, and frakti. The support for many container runtime types is based on the Container Runtime Interface (CRI). The CRI is a plug-in design that allows the kubelet to communicate with the available container runtime. The default container runtime in Azure Kubernetes Service is Docker. However, you can also use Kata Containers and containerd.

Interact with a Kubernetes cluster

Kubernetes provides a command-line tool called `kubectl` that you use to manage your cluster. You use `kubectl` to send commands to the cluster's control plane or fetch information about all Kubernetes objects via the API server.

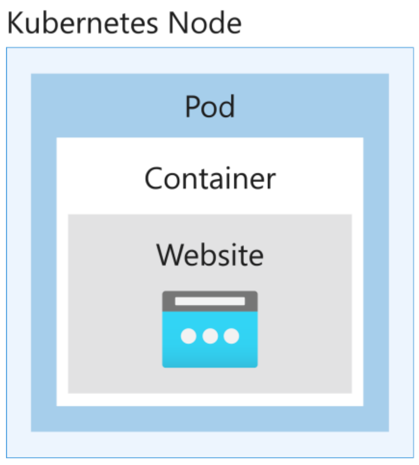
`kubectl` uses a configuration file that includes configuration information:

- Use the **Cluster** configuration to specify a cluster name, certificate information, and the service API endpoint associated with the cluster. This definition enables you to connect from a single workstation to multiple clusters.
- Use the **User** configuration to specify the users and their permission levels when they're accessing the configured clusters.
- Use the **Context** configuration to group clusters and users by using a friendly name. For example, you might have a "dev-cluster" and a "prod-cluster" to identify your development and production clusters.

You can configure `kubectl` to connect to multiple clusters by providing the correct context as part of the command-line syntax.

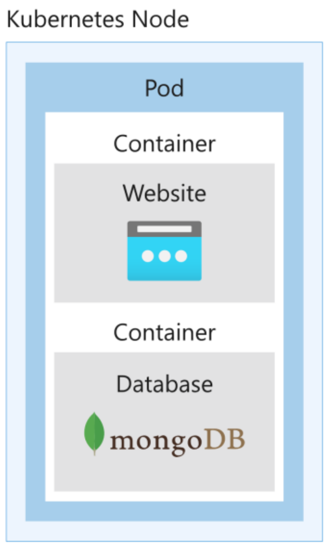
Kubernetes pods

A pod represents a single instance of an application running in Kubernetes. The workloads that you run on Kubernetes are containerized applications. Unlike in a Docker environment, you can't run containers directly on Kubernetes. You package the container into a Kubernetes object called a pod. A pod is the smallest object that you can create in Kubernetes.



A single pod can hold a group of one or more containers. However, a pod typically doesn't contain multiples of the same application.

A pod includes information about the shared storage and network configuration, and a specification on how to run its packaged containers. You use pod templates to define the information about the pods that run in your cluster. Pod templates are YAML coded files that you reuse and include in other objects to manage pod deployments.



For example, assume that you want to deploy a website to a Kubernetes cluster. You create the pod definition file that specifies the application's container images and configuration. Then you deploy the pod definition file to Kubernetes.

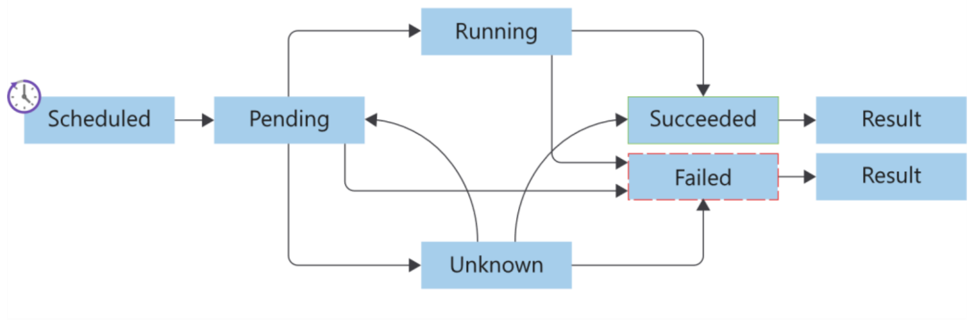
It's unlikely that a web application has a website as the only component in the solution. A web application typically has some kind of datastore and other supporting elements. Kubernetes pods can also contain more than one container.

Assume that your site uses a database. The website is packaged in the main container, and the database is packaged in the supporting container. For these two containers to function and communicate with each other, you expect them to run in an environment that provides a host OS, a network stack, kernel namespaces, shared memory, and volumes to persist data. The pod is the sandbox environment that provides all of these services to your application. The pod also allows the containers to share its assigned IP address.

Because you can potentially create many pods that are running on many nodes, it can be hard to identify them. You recognize and group pods by using string labels that you specify when you define a pod.

Life cycle of a Kubernetes pod

Kubernetes pods have a distinct life cycle that affects the way you deploy, run, and update pods. You start by submitting the pod YAML manifest to the cluster. After the manifest file is submitted and persisted to the cluster, it defines the wanted state of the pod. The scheduler schedules the pod to a healthy node that has enough resources to run the pod.



Let's look at the phases in a pod's life cycle.

Pending	After the pod run is scheduled, the container runtime downloads container images and starts all containers for the pod.
Running	The pod transitions to a running state after all of the resources within the pod are ready.
Succeeded	The pod transitions to a succeeded state after the pod completes its intended task and runs successfully.
Failed	Pods can fail for various reasons. A container in the pod might have failed, leading to the termination of all other containe Or maybe an image wasn't found during preparation of the pod containers. In these types of cases, the pod can go to a failed state. Pods can transition to a failed state from either a pending state or a running state. A specific failure can also place a pod back in the pending state.
Unknown	If the state of the pod can't be determined, the pod is an unknown state.

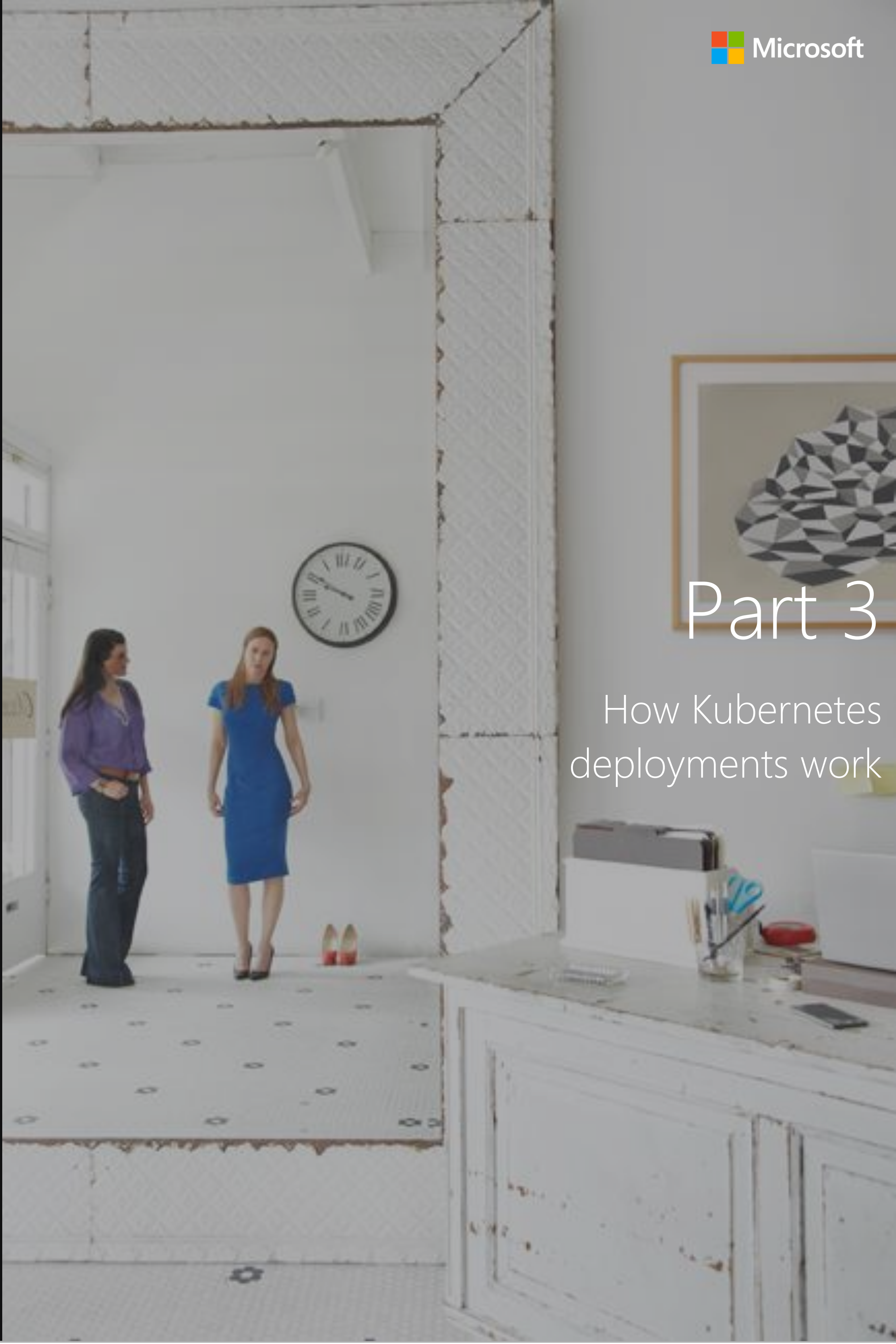
Pods are kept on a cluster until a controller, the control plane, or a user explicitly removes them. When a pod is deleted and is replaced by a new pod, the new pod is an entirely new instance of the pod based on the pod manifest.

The cluster doesn't save the pod's state or dynamically assigned configuration. For example, it doesn't save the pod's ID or IP address. This aspect affects how you deploy pods and how you design your apps. For example, you can't rely on preassigned IP addresses for your pods.

Container states

Keep in mind that the phases are a summary of where the pod is in its life cycle. When you inspect a pod, the cluster uses three states to track your containers inside the pod.

Waiting	This is the default state of a container and the state that the container is in when it's not running or terminated.
Running	The container is running as expected without any problems.
Terminated	The container is no longer running. The reason is that either all tasks finished or the container failed for some reason. A reason and exit code are available for debugging both cases.

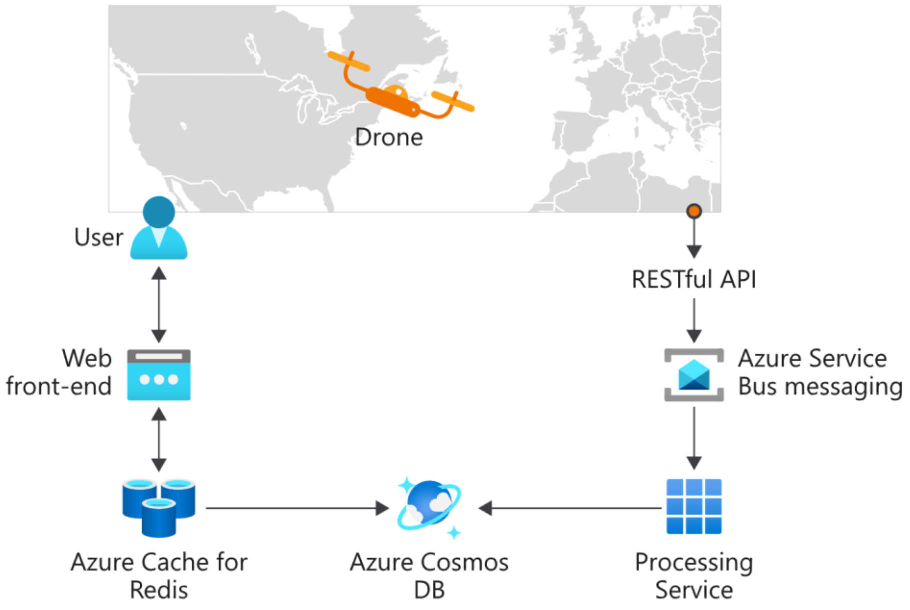


Part 3

How Kubernetes deployments work

How Kubernetes deployments work

The drone tracking application has several components that are deployed separately from each other. It's your job to configure deployments for these components on the cluster. Here you'll look at some of the deployment options available to you to deploy these components.



Pod deployment options

There are several options to manage the deployment of pods in a Kubernetes cluster when you're using `kubectl`. The options are:

- Pod templates
- Replication controllers
- Replica sets
- Deployments

You can use any of these four Kubernetes object type definitions to deploy a pod or pods. These files make use of YAML to describe the intended state of the pod or pods that will be deployed.

What is a pod template?

A pod template allows you to define the configuration of the pod you wish to deploy. The template contains information such as the name of container image, which container registry to use to fetch the images. The template may also include runtime configuration information such as ports to use. Templates are defined by using YAML in the same way as when you create Docker files.

You can use templates to deploy pods manually. However, a manually deployed pod isn't relaunched after it fails, is deleted, or is terminated. To manage the lifecycle of a pod, you need to create a higher-level Kubernetes object.

What is a replication controller?

A replication controller uses pod templates and defines a specified number of pods that must run. The controller helps you run multiple instances of the same pod and ensures pods are always running on one or more nodes in the cluster. The controller replaces running pods in this way with new pods if they fail, are deleted, or are terminated.

For example, assume you deploy the drone tracking front-end website and users start accessing the website. If all the pods fail for any reason, the website is unavailable to your users unless you launch new pods. A replication controller helps you make sure your website is always available.

What is a replica set?

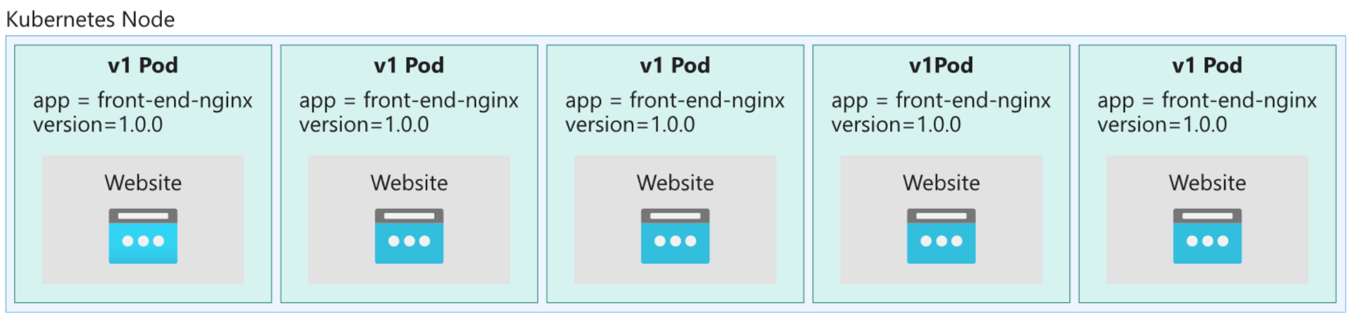
A replica set replaces the replication controller as the preferred way to deploy replicas. A replica set includes the same functionality as a replication controller. However, it has an extra configuration option to include a selector value.

A selector allows the replica set to identify all the pods running underneath it. This feature allows you to manage pods labeled with the same value as the selector value, but not created with the replicated set.

What is a deployment?

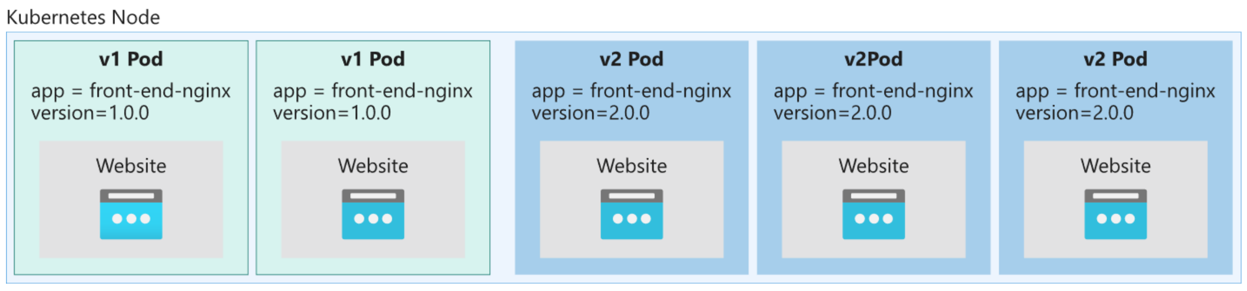
A deployment creates a management object one level higher than a replica set and allows you to deploy and manage updates for pods in a cluster.

Assume that you have five instances of your application deployed in your cluster. There are five pods running version 1.0.0 of your application.



If you decide to update your application manually, you can remove all pods and then launch new pods running version 2.0.0 of your application. With this strategy, your application will experience downtime.

What you instead want to do is execute a rolling update, where you launch pods with the new version of your application before you remove the older app versioned pods. Rolling updates will launch one pod at a time instead of taking down all the older pods at once. Deployments honor the number of replicas configured in the section that describes information about replica sets. It will maintain the number of pods specified in the replica set as it replaces old pods with new pods.



Deployments, by default, provide a rolling update strategy for updating pods. You can also use a re-create strategy. This strategy will terminate pods before launching new pods.

Deployments also provide you with a rollback strategy, which you can execute by using `kubectl`.

Deployments make use of YAML-based definition files and make it easy to manage deployments. Keep in mind that deployments allow you to apply any changes to your cluster. For example, you can deploy new versions of an app, update labels, and run other replicas of your pods.

`kubectl` has convenient syntax to create a deployment automatically when you're using the `kubectl run` command to deploy a pod. This command creates a deployment with the required replica set and pods. However, the command doesn't create a definition file. A best practice is to manage all deployments with deployment definition files and track changes by using a version control system.

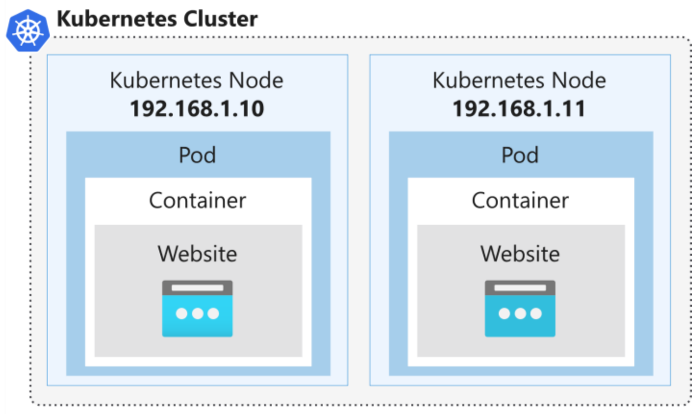
Deployment considerations

Kubernetes has specific requirements on how you configure networking and storage for a cluster. How you configure these two aspects affects your decisions on how to expose your applications on the cluster network and store data.

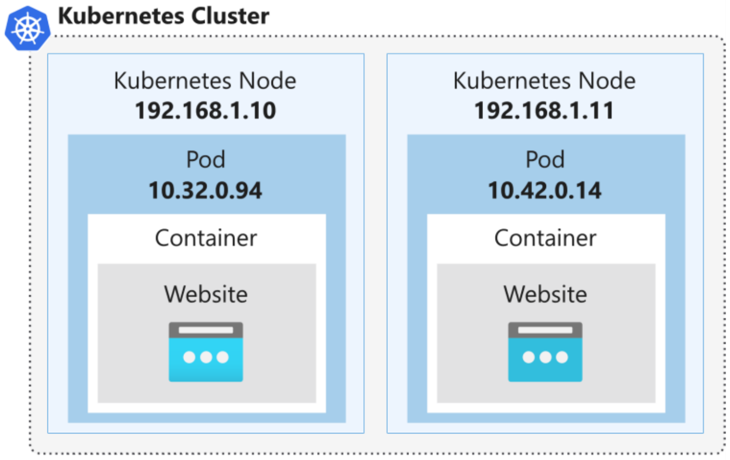
For example, each of the services in the drone tracking application has specific requirements for user access, inter-process network access, and data storage. Let's have a look at these aspects of a Kubernetes cluster and how they affect the deployment of applications.

Kubernetes networking

Assume you have a cluster with one control planes and two nodes. When you add nodes to Kubernetes, an IP address is automatically assigned to each node from an internal private network range. For example, assume that your local network range is 192.168.1.0/24.



Each pod that you deploy gets assigned an IP from a pool of IP addresses. For example, assume that your configuration uses the 10.32.0.0/12 network range.



By default, the pods and nodes can't communicate with each other by using different IP address ranges.

To further complicate matters, recall that pods are transient. The pod's IP address is temporary and can't be used to reconnect to a newly created pod. This configuration affects how your application communicates with its internal components and how you and services interact with it externally.

To simplify communication, Kubernetes expects you to configure networking in such a way that:

- Pods can communicate with one another across nodes without Network Address Translation (NAT).
- Nodes can communicate with all pods, and the other way around, without NAT.
- Agents on a node can communicate with all nodes and pods.

Kubernetes offers several networking options that you can install to configure networking. Examples include Antrea, Cisco Application Centric Infrastructure (ACI), Cilium, Flannel, Kubenet, VMware NSX-T, and Weave Net.

Cloud providers also provide their networking solutions. For example, Azure Kubernetes Service (AKS) supports the Azure Virtual Network container network interface (CNI), Kubenet, Flannel, Cilium, and Antrea.

Kubernetes services

A Kubernetes service is a Kubernetes object that provides stable networking for pods. A Kubernetes service enables communication between nodes, pods, and users of your application, both internal and external, to the cluster.

Kubernetes assigns a service an IP address on creation, just like a node or pod. These addresses get assigned from a service cluster's IP range. An example is 10.96.0.0/12. A service is also assigned a DNS name based on the service name, and an IP port.

In the drone tracking application, network communication is as follows:

- The website and RESTful API are accessible to users outside the cluster.
- The in-memory cache and message queue services are accessible to the front end and the RESTful API, respectively, but not to external users.
- The message queue needs access to the data processing service, but not to external users.
- The NoSQL database is accessible to the in-memory cache and data processing service, but not to external users.

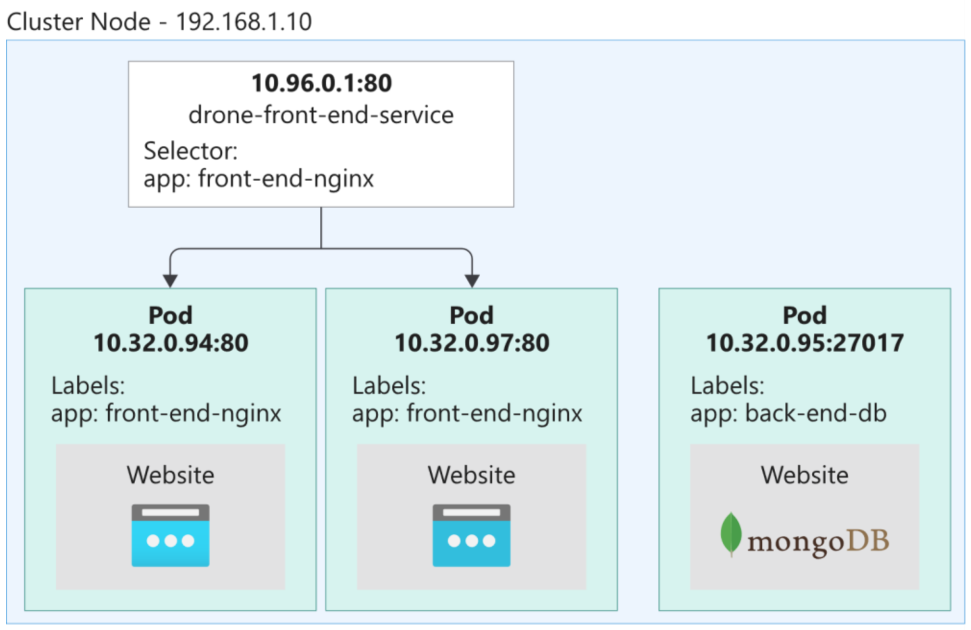
To support these scenarios, you can configure three types of services to expose your app's components.

ClusterIP	The address assigned to a service that makes the service available to a set of services inside the cluster. For example, c back-end components of your application.
NodePort	The node port, between 30000 and 32767, that the Kubernetes control plane assigns to the service. An example is 192.169.1.11 on clusters01. You then configure the service with a target port on the pod that you want to expose. For example, configure port 80 on the pod running one of the front ends. You can now access the front end through a node IP and port address.
LoadBalancer	The load balancer that allows for the distribution of load between nodes running your application and exposing the pod to public network access. You typically configure load balancers when you use cloud providers. In this case, traffic from the external load balancer is directed to the pods running your application.

In the drone tracking application, you might decide to expose the tracking website and the RESTful API by using a LoadBalancer and the data processing service by using a ClusterIP.

How to group pods

Managing pods by IP address isn't practical. Pod IP addresses change as controllers re-create them, and you might have any number of pods running.

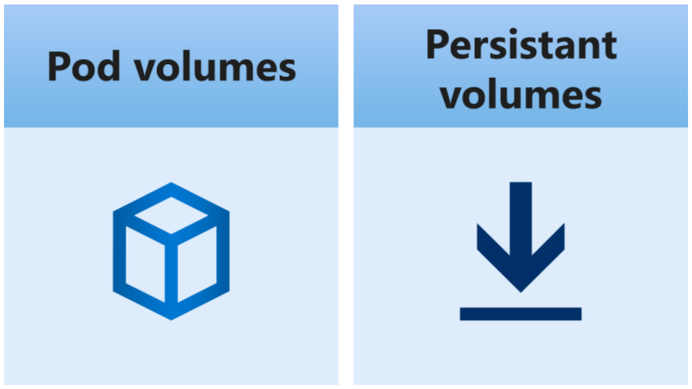


A service object allows you to target and manage specific pods in your cluster by using selector labels. You set the selector label in a service definition to match the pod label defined in the pod's definition file.

For example, assume that you have many running pods. Only a few of these pods are on the front end, and you want to set a LoadBalancer service that targets only the front-end pods. You can apply your service to expose these pods by referencing the pod label as a selector value in the service's definition file. The service will now group only the pods that match the label. If a pod is removed and re-created, the new pod is automatically added to the service group through its matching label.

Kubernetes storage

Kubernetes uses the same storage volume concept that you find when using Docker. Docker volumes are less managed than the Kubernetes volumes because Docker volume lifetimes aren't managed. The Kubernetes volume's lifetime is an explicit lifetime that matches the pod's lifetime. This lifetime match means a volume outlives the containers that run in the pod. However, if the pod is removed, so is the volume.



Kubernetes provides options to provision persistent storage with the use of *PersistentVolumes*. You can also request specific storage for pods by using *PersistentVolumeClaims*.

Keep both of these options in mind when you're deploying application components that require persisted storage like message queues and databases.

Cloud integration considerations

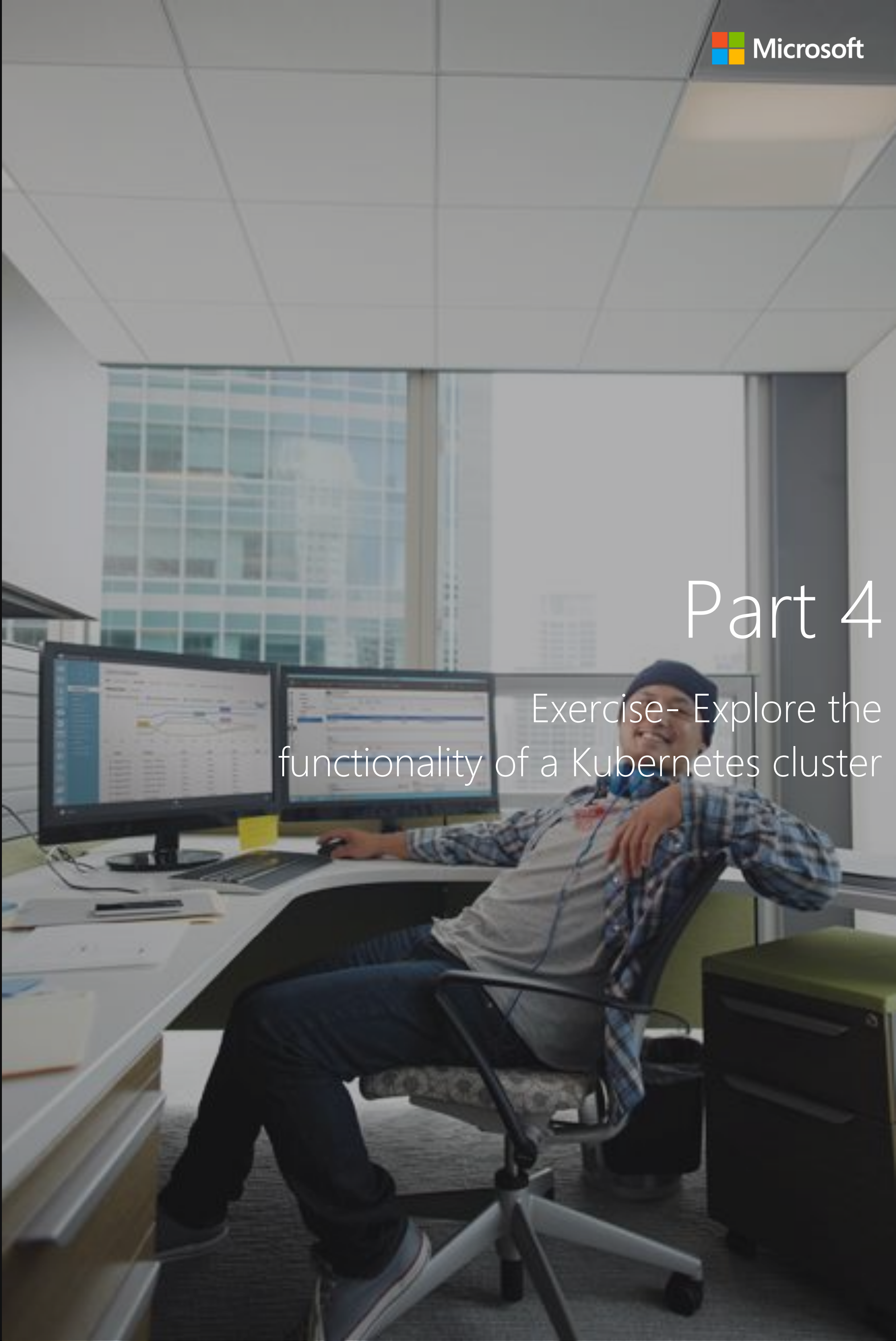
Kubernetes doesn't dictate the technology stack you use in your cloud-native application. In a cloud environment such as Azure, you can use several services outside the Kubernetes cluster.

Recall from earlier that Kubernetes doesn't provide any of the following services:

- Middleware
- Data-processing frameworks
- Databases
- Caches
- Cluster storage systems

In the example drone tracking solution, there are three services that provide middleware functionality. A NoSQL database, an in-memory cache service, and a message queue. You might choose to use MongoDB Atlas for the NoSQL solution, Redis to manage in-memory cache and RabbitMQ, or Kafka, depending on your message queue needs.

When you're using a cloud environment such as Azure, it's a best practice to use services outside the Kubernetes cluster. This decision can simplify the cluster's configuration and management. For example, you can use *Azure Cache for Redis* for the in-memory caching services, *Azure Service Bus messaging* for the message queue, and *Azure Cosmos DB* for the NoSQL database.



Part 4

Exercise- Explore the functionality of a Kubernetes cluster

Exercise - Explore the functionality of a Kubernetes cluster

Several options are available when you're running Kubernetes locally. You can install Kubernetes on physical machines or VMs, or use a cloud-based solution such as Azure Kubernetes Service (AKS).

Your goal in this exercise is to explore a Kubernetes installation with a single-node cluster. You're going to configure a *MicroK8s* environment that's easy to set up and tear down. Then you'll deploy an NGINX website and scale it out to multiple instances. Finally, you'll go through the steps to delete the running pods and clean up the cluster.

> Note
>
> This exercise is optional and includes steps that show how to delete and uninstall the software and resources you'll use in the exercise.

What is MicroK8s?

MicroK8s is an option for deploying a single-node Kubernetes cluster as a single package to target workstations and Internet of Things (IoT) devices. Canonical, the creator of Ubuntu Linux, originally developed and maintains MicroK8s.

You can install MicroK8s on Linux, Windows, and macOS. However, installation instructions are slightly different for each operating system. Choose the option that best fits your environment.

Install MicroK8s on Windows

You use Multipass to run MicroK8s on Windows. Multipass is a lightweight VM manager for Linux, Windows, and macOS.

1. Download and install the latest release of Multipass for Windows from GitHub .
2. In a command console, run the Multipass launch command to configure and run the microk8s-vm image. This step might take a few minutes to complete, depending on the speed of your internet connection and desktop.

   cmd
   ```
   multipass launch --name microk8s-vm --mem 4G --disk 40G
   ```

3. After you receive the launch confirmation for microk8s-vm, you can access the VM instance by using the `multipass shell microk8s-vm` command.

cmd

```
multipass shell microk8s-vm
```

At this point, you can access the Ubuntu VM that will host your cluster and install MicroK8s.

4. Install the MicroK8s snap application. This step might take a few minutes to complete, depending on the speed of your internet connection and desktop.

Bash

```bash
sudo snap install microk8s --classic
```

A successful installation shows the following message.

Output

```
2020-03-16T12:50:59+02:00 INFO Waiting for restart...
microk8s v1.17.3 from Canonical✓ installed
```

You're now ready to install add-ons on the cluster.

Prepare the cluster

You can use the status command in MicroK8s to view the status of the installed add-ons on your cluster. These add-ons provide several services, some of which you covered previously. One example is DNS functionality.

1. To check the status of the installation, run the `microk8s.status --wait-ready` command.

Bash

```bash
sudo microk8s.status --wait-ready
```

Notice that you can enable several add-ons on your cluster. Don't worry about the add-ons that you don't recognize. You'll enable only three of these add-ons in your cluster.

Output

```
microk8s is running
addons:
cilium: disabled
dashboard: disabled
dns: disabled
```

```
fluentd: disabled
gpu: disabled
helm3: disabled
helm: disabled
ingress: disabled
istio: disabled
jaeger: disabled
juju: disabled
knative: disabled
kubeflow: disabled
linkerd: disabled
metallb: disabled
metrics-server: disabled
prometheus: disabled
rbac: disabled
registry: disabled
storage: disabled
```

2. Next, you'll enable the DNS, Dashboard, and Registry add-ons. Here is the purpose of each add-on.

DNS	Deploys the `coreDNS` service.
Dashboard	Deploys the `kubernetes-dashboard` service and several other services that support its functionality. It's a general-purpose, web-based UI for Kubernetes clusters.
Registry	Deploys a private registry and several services that support its functionality. You can use this registry to store private containers.

3. Install the add-ons by running the following command.

Bash

```bash
sudo microk8s.enable dns dashboard registry
```

You're now ready to access your cluster by using `kubectl`.

Explore the Kubernetes cluster

MicroK8s provides a version of `kubectl` that you can use to interact with your new Kubernetes cluster. This copy of `kubectl` allows you to have a parallel installation of another system-wide `kubectl` instance without affecting its functionality.

1. Run the `snap alias` command to alias `microk8s.kubectl` to `kubectl`. This step simplifies usage.

   Bash

   ```bash
   sudo snap alias microk8s.kubectl kubectl
   ```

   You'll see the following output when the command finishes successfully.

   Output

   ```
   Added:
     - microk8s.kubectl as kubectl
   ```

Display cluster node information

Recall from earlier that a Kubernetes cluster exists out of control planes and worker nodes. Let's explore the new cluster to see what's installed.

1. Check the nodes that are running in your cluster.

   You know that MicroK8s is a single-node cluster installation, so you expect to see only one node. Keep in mind, though, that this node is both the control plane and a worker node in the cluster. Confirm this configuration by running the `kubectl get nodes` command. You can use the `kubectl get` command to retrieve information about all the resources in your cluster.

   Bash

   ```bash
   sudo kubectl get nodes
   ```

   The result will be similar to the following example, which shows you that there's only one node in the cluster with the name microk8s-vm. Notice that the node is in a ready state. The ready state indicates that the control plane might schedule workloads on this node.

   Output

   ```
   NAME          STATUS   ROLES    AGE   VERSION
   microk8s-vm   Ready    <none>   35m   v1.17.3
   ```

You can get more information for the specific resource that's requested. For example, let's assume that you need to find the IP address of the node. You use the `-o wide` parameter to fetch extra information from the API server.

Bash

```
sudo kubectl get nodes -o wide
```

The result will be similar to the following example. Notice that you now can see the internal IP address of the node, the OS running on the node, the kernel version, and the container runtime.

Output

```
NAME          STATUS   ROLES    AGE   VERSION   INTERNAL-IP      EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION      CONTAINER-RUNTIME
microk8s-vm   Ready    <none>   36m   v1.17.3   192.168.56.132   <none>        Ubuntu 18.04.4 LTS   4.15.0-88-generic   containerd://1.2.5
```

2. The next step is to explore the services running on your cluster. As with nodes, you can use the `kubectl get` command to find information about the services running on the cluster.

Bash

```
sudo kubectl get services -o wide
```

The result will be similar to the following example. But notice that only one service is listed. You installed add-ons on the cluster earlier, and you expect to see these services as well.

Output

```
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE   SELECTOR
kubernetes   ClusterIP   10.152.183.1   <none>        443/TCP   37m   <none>
```

The reason for the single service listing is that Kubernetes uses a concept called namespaces. You can use namespaces to logically divide a cluster into multiple virtual clusters.

Use the `--all-namespaces` parameter to fetch all services in all namespaces.

Bash

```
sudo kubectl get services -o wide --all-namespaces
```

The result will be similar to the following example. Notice that you have three namespaces in your cluster. They're the default, `container-registry`, and `kube-system` namespaces. Here you can see the `registry`, `kube-dns`, and `kubernetes-`

`dashboard` instances that you installed. You'll also see the supporting services that were installed alongside some of the add-ons.

Output

```
NAMESPACE          NAME                        TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)                  AGE   SELECTOR
container-registry registry                    NodePort   10.152.183.36   <none>        5000:32000/TCP           28m   app=registry
default            kubernetes                  ClusterIP  10.152.183.1    <none>        443/TCP                  37m   <none>
kube-system        dashboard-metrics-scraper   ClusterIP  10.152.183.130  <none>        8000/TCP                 28m   k8s-app=dashboard-metrics-
scraper
kube-system        heapster                    ClusterIP  10.152.183.115  <none>        80/TCP                   28m   k8s-app=heapster
kube-system        kube-dns                    ClusterIP  10.152.183.10   <none>        53/UDP,53/TCP,9153/TCP   28m   k8s-app=kube-dns
kube-system        kubernetes-dashboard        ClusterIP  10.152.183.132  <none>        443/TCP                  28m   k8s-app=kubernetes-
dashboard
kube-system        monitoring-grafana          ClusterIP  10.152.183.88   <none>        80/TCP                   28m   k8s-app=influxGrafana

kube-system        monitoring-influxdb         ClusterIP  10.152.183.232  <none>        8083/TCP,8086/TCP        28m   k8s-app=influxGrafana
```

Now that you can see the services running on the cluster, you can schedule a workload on the worker node.

Install a web server on a cluster

You want to schedule a web server on the cluster to serve a website to your customers. You can choose from several options. For this example, you'll use NGINX.

Recall from earlier that you can use pod manifest files to describe your pods, replica sets, and deployments to define workloads. Because you haven't covered these files in detail, you'll use `kubectl` to directly pass the information to the API server.

Even though the use of `kubectl` is handy, using manifest files is a best practice. Manifest files allow you to roll forward or roll back deployments with ease in your cluster. These files also help document the configuration of a cluster.

1. Use the `kubectl create deployment` command to create your NGINX deployment. Specify the name of the deployment and the container image to create a single instance of the pod.

   Bash

   ```bash
   sudo kubectl create deployment nginx --image=nginx
   ```

   The result will be similar to the following example.

   Output

   ```
   deployment.apps/nginx created
   ```

2. Use `kubectl get deployments` to fetch the information about your deployment.

Bash

```bash
sudo kubectl get deployments
```

The result will be similar to the following example. Notice that the name of the deployment matches the name you gave it, and that one deployment with this name is in a ready state and available.

Output

```
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
nginx   1/1     1            1           18s
```

3. The deployment created a pod. Use the `kubectl get pods` command to fetch info about your cluster's pods.

Bash

```bash
sudo kubectl get pods
```

The result will be similar to the following example. Notice that the name of the pod is a generated value prefixed with the name of the deployment, and the pod has a status of running.

Output

```
NAME                     READY   STATUS    RESTARTS   AGE
nginx-86c57db685-dj6lz   1/1     Running   0          33s
ubuntu@microk8s-vm:~$
```

Test the website installation

Test the NGINX installation by connecting to the web server through the pod's IP address.

1. Use the `-o wide` parameter to find the address of the pod.

Bash

```bash
sudo kubectl get pods -o wide
```

The result will be similar to the following example. Notice that the command returns both the IP address of the node and the node name on which the workload is scheduled.

Output

```
NAME                      READY   STATUS    RESTARTS   AGE    IP           NODE          NOMINATED NODE   READINESS GATES
nginx-86c57db685-dj6lz    1/1     Running   0          4m17s  10.1.83.10   microk8s-vm   <none>           <none>
```

2. Use `wget` to access the website.

Bash

```
wget 10.1.83.10
```

The result will be similar to the following example.

Output

```
--2020-03-16 13:34:17--  http://10.1.83.10/
Connecting to 10.1.83.10:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 612 [text/html]
Saving to: 'index.html'

index.html
100%[=============================================================================================>]
612  --.-KB/s    in 0s

2020-03-16 13:34:17 (150 MB/s) - 'index.html' saved [612/612]
```

Scale a web server deployment on a cluster

Assume that you suddenly see an increase in users who access your website, and the website starts failing because of the load. You can deploy more instances of the site in your cluster and split the load across the instances.

You can use the `kubectl scale` command to scale the number of replicas in your deployment. You specify the number of replicas you need and the name of the deployment.

1. Run the `kubectl scale` command to scale the total of NGINX pods to three.

Bash

```
sudo kubectl scale --replicas=3 deployments/nginx
```

The result will be similar to the following example.

Output

```
sudo kubectl scale --replicas=3 deployments/nginx
deployment.apps/nginx scaled
```

The scale command allows you to scale the instance count up or down.

2. Check the number of running pods by using the `kubectl get` command, and again pass the `-o wide` parameter.

Bash

```
sudo kubectl get pods -o wide
```

The result will be similar to the following example. Notice that you now see three running pods, each with a unique IP address.

Output

```
NAME                      READY   STATUS    RESTARTS   AGE     IP           NODE          NOMINATED NODE
READINESS GATES
nginx-86c57db685-dj6lz    1/1     Running   0          7m57s   10.1.83.10   microk8s-vm   <none>           <none>
nginx-86c57db685-lzrwp    1/1     Running   0          9s      10.1.83.12   microk8s-vm   <none>           <none>
nginx-86c57db685-m7vdd    1/1     Running   0          9s      10.1.83.11   microk8s-vm   <none>           <none>
ubuntu@microk8s-vm:~$
```

You would need to apply several additional configurations to the cluster to effectively expose your website as a public-facing website. Examples include installing a load balancer and mapping node IP addresses. This type of configuration forms part of advanced aspects that you'll explore in the future.

Uninstall MicroK8s

You can remove everything you've deployed so far, and even the VM, to recover space on your development machine. Keep in mind that this procedure is optional.

1. Remove the add-ons from the cluster by running the `microk8s.disable` command and specifying the add-ons to remove.

Bash

```bash
sudo microk8s.disable dashboard dns registry
```

2. Remove MicroK8s from the VM by running the `snap remove` command.

Bash

```bash
sudo snap remove microk8s
```

There are a few additional steps to take on Windows and macOS if you want to remove the Multipass VM manager from your machine.

1. Exit the VM by running the `exit` command.

Bash

```bash
exit
```

2. Stop the VM by running the `multipass stop` command and specifying the VM's name.

Bash

```bash
multipass stop microk8s-vm
```

3. Delete and purge the VM instance by running `multipass delete` and then `multipass purge`.

cmd

```cmd
multipass delete microk8s-vm
multipass purge
```

Microsoft

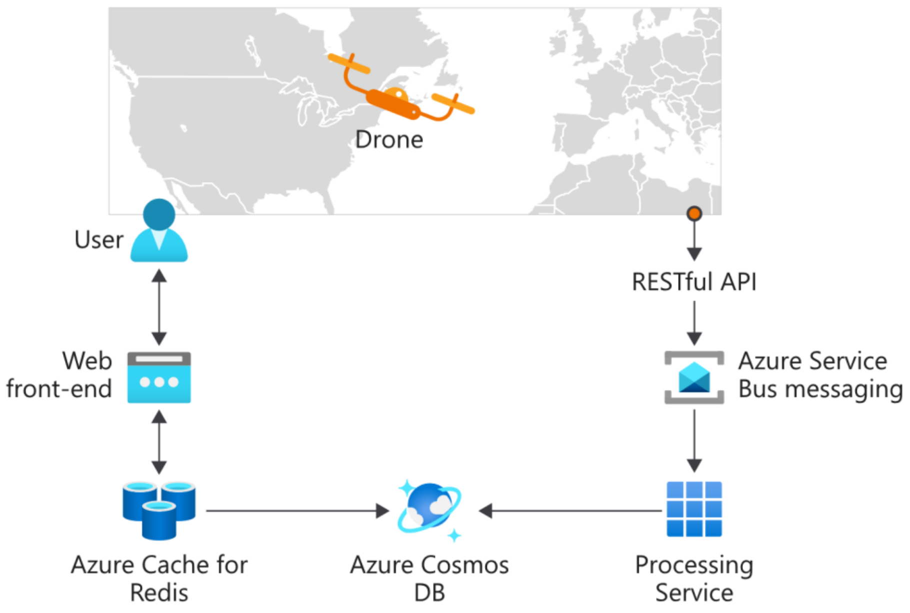
Part 5

When to use Kubernetes

When to use Kubernetes

The decision to use a container orchestration platform like Kubernetes depends on business and development requirements. Let's review the high-level architecture of the drone tracking solution.

The solution is built as microservices that are designed as loosely coupled, collaborative services. You're deploying these services separately from each other to simplify the solution's design and maintenance. Here is the current configuration of your solution.



- Web front end: shows maps and information about tracked drones.
- Cache service: stores frequently requested information displayed on the website.
- RESTful API: used by tracked drones to send data about their status, such as a GPS location and battery charge levels.
- Queue: holds unprocessed data collected by the RESTful API.
- Data processing service: fetches and processes data from the queue.
- NoSQL database: stores processed tracking data and user information captured from the website and the data processing service.

Separate teams in your company develop and own these services. Each team uses containers to build and deploy its service. This new strategy allows the development teams to keep up with the requirements of modern software development for automation, testing, and overall stability and quality.

The change in developer thinking has resulted in several process and business benefits for the company. Examples include better use of hosted compute resources, new features that have improved time to market, and improved customer reach.

However, several challenges with container management led your company to investigate container orchestration solutions. Your teams found that scaling the tracking application to a handful of deployments was relatively easy, but scaling and managing many instances was hard.

There are several other aspects to consider. Examples include dealing with failed containers, storage allocation, network configuration, and management of application secrets.

As you've seen earlier, Kubernetes provides support for all of these challenges as an orchestration platform.

You want to use Kubernetes when your company:

- Develops applications as microservices.
- Develops applications as cloud-native applications.
- Deploys microservices by using containers.
- Updates containers at scale.
- Requires centralized container networking and storage management.

When not to use Kubernetes

Not all applications need to run in Kubernetes. As a result, Kubernetes might not be a good fit for your company.

For example, the effort in containerization and deployment of a monolithic application might be more than the benefits of running the application in Kubernetes. A monolithic architecture can't easily use features such as individual component scaling or updates.

Kubernetes can introduce many business benefits for software development, deployment, management, and streamlining of processes. However, Kubernetes has a steep learning curve. The modular design of Kubernetes introduces potentially new concepts that will affect teams across your company.

Your development teams will have to embrace modern design concepts when developing and designing applications. These concepts include the use of microservices and the containerization of these services. Teams also needs to experiment with container and orchestration environments to make the best use of all the available options.

If your company isn't ready to adopt this change, then Kubernetes might not be a good fit for your company.

Microsoft

Summary

Module summary

Summary

Our goal was to help you evaluate whether Kubernetes would be a good choice as a container orchestration platform for your business. We looked at several features that enhance the Azure Kubernetes Service (AKS) offering. We saw how these features can help you decide if Kubernetes is a good fit for new projects, or if you should move to Kubernetes to orchestrate current container deployments.

You saw how Kubernetes provides for:

- Deployment of containers.
- Self-healing of containers.
- Dynamically scaling container count up or down.
- Automated rolling updates and rollbacks of containers.
- Management of storage.
- Management of network traffic.
- Storage and management of sensitive information such as usernames and passwords.

You were looking for a container orchestration platform to deploy and manage your drone tracking solution into new customer regions. You now understand how Kubernetes can help you develop, deploy, and manage applications in your container environment.

Learn more

To learn more about Kubernetes, running Kubernetes on Azure, and related tools, visit the following sites and articles:

- Kubernetes
- Azure Kubernetes Service
- MicroK8s
- Multipass GitHub repository

References

✓ **Introduction to Kubernetes**
https://docs.microsoft.com/en-us/learn/modules/intro-to-kubernetes/

Microsoft

Cloud Solution Architects

Customer Success

NorthEast Region

August 2020