

# Cloud Platform Operations

---

Azure Container, K8s

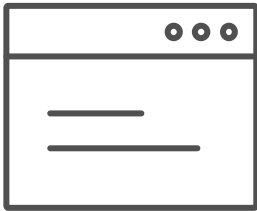
David Yoon | CEO  
david@2miles.co.kr



강의자료 -

<https://bit.ly/3aRFnSI>

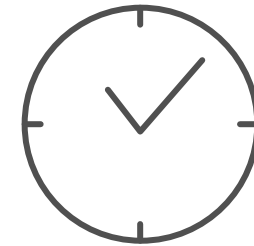
# What we hear from **developers**



I need to create applications  
at a competitive rate without  
worrying about IT



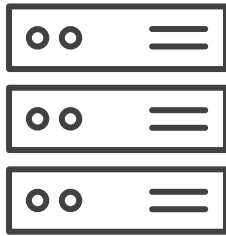
New applications run smoothly  
on my machine but malfunction  
on traditional IT servers



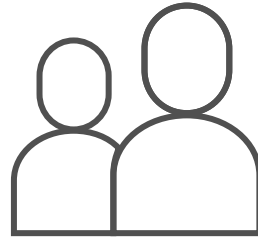
My productivity and application  
innovation become suspended  
when I have to wait on IT



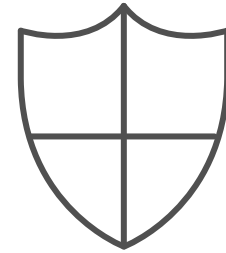
# What we hear from IT



I need to manage servers  
and maintain compliance  
with little disruption



I'm unsure of how to integrate  
unfamiliar applications, and I  
require help from developers

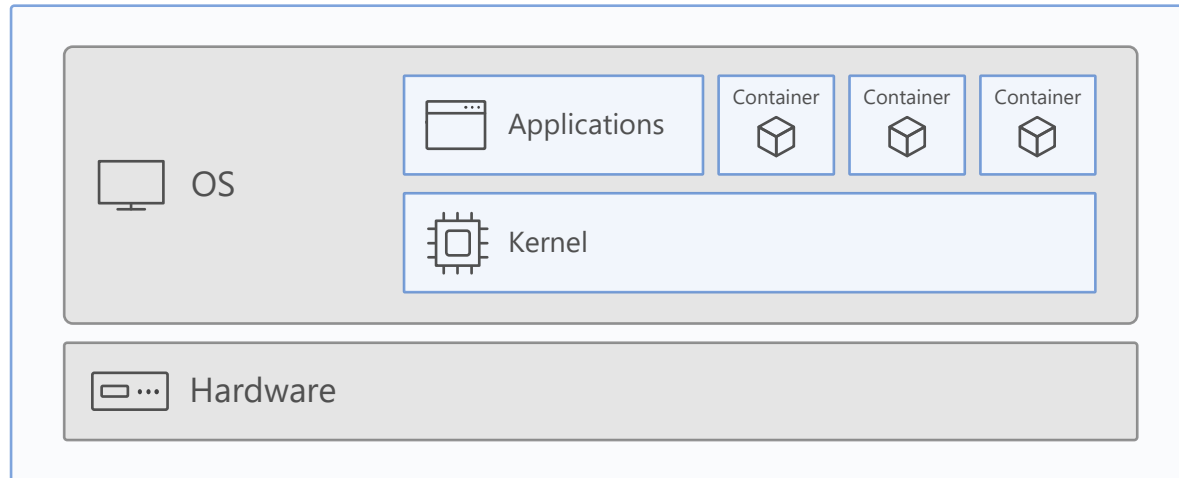


I'm unable to focus on both  
server protection and  
application compliance

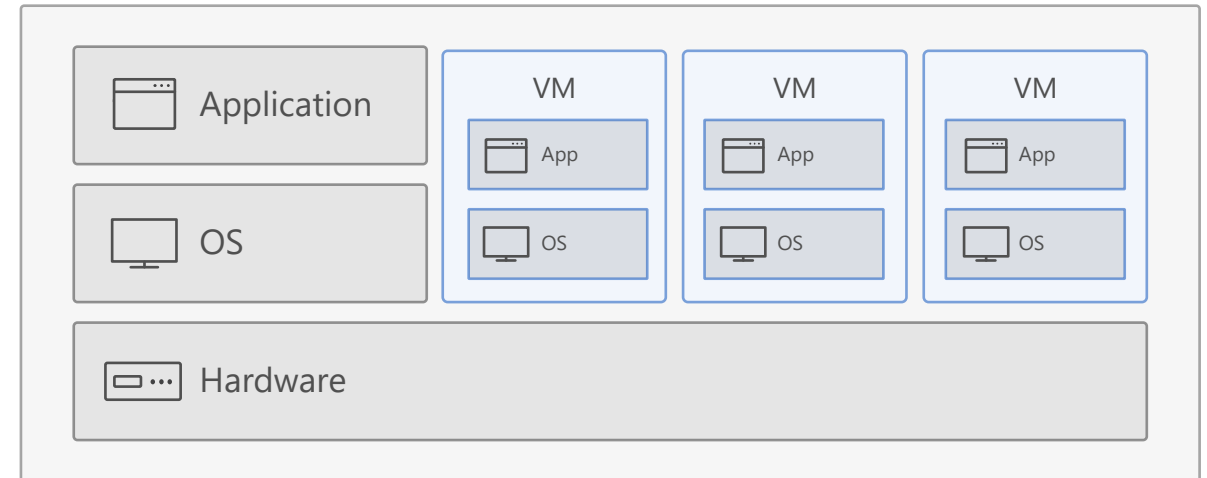


# What is a **container**?

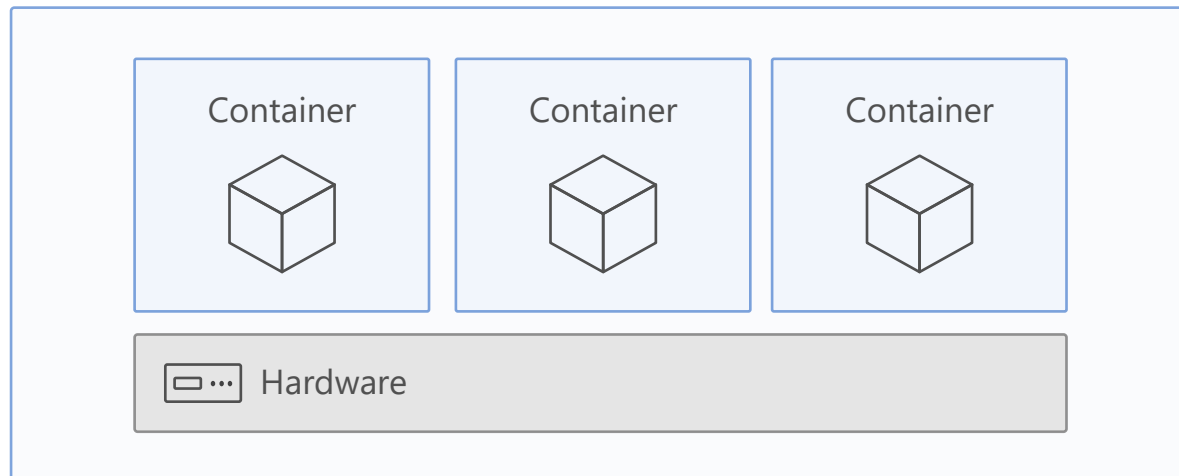
**Containers** = operating system virtualization



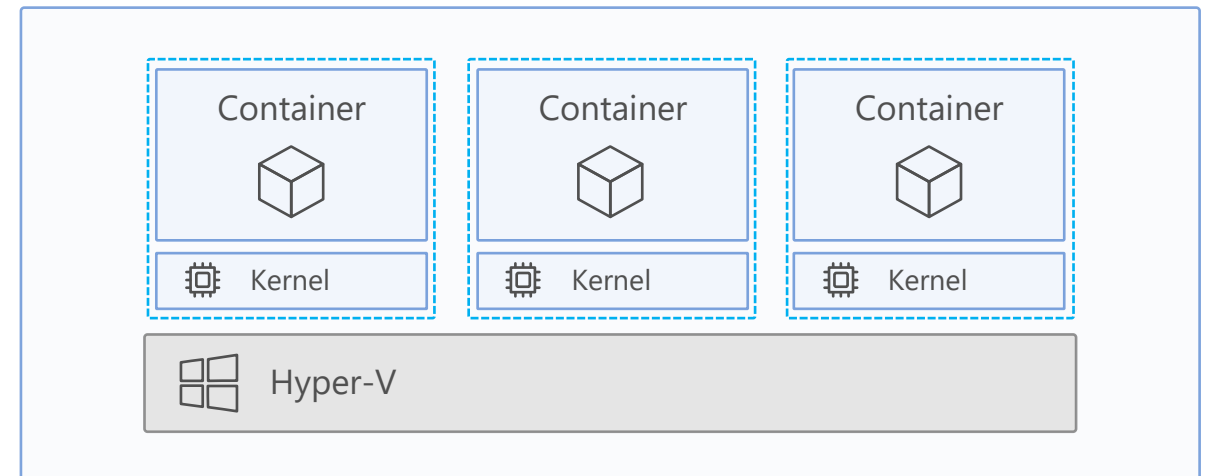
Traditional virtual machines = hardware virtualization



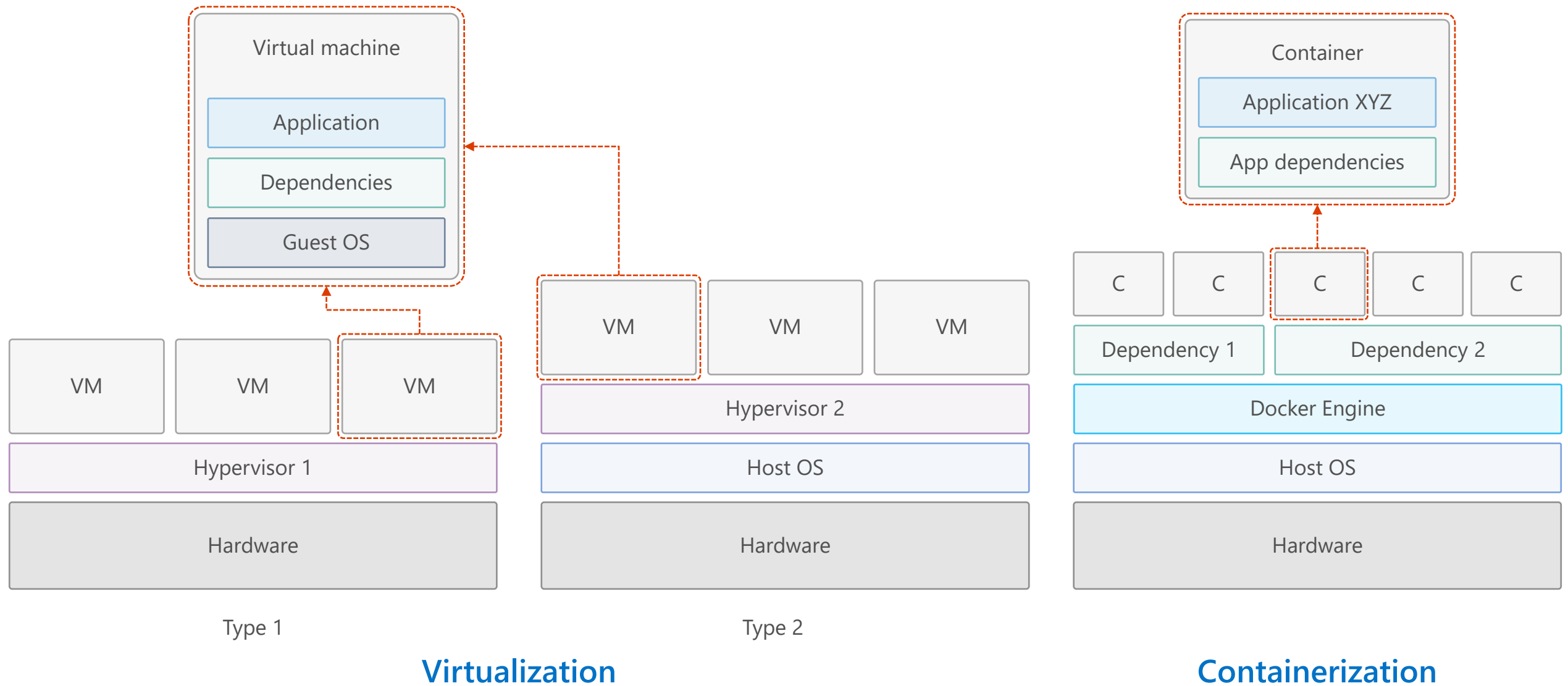
**Windows Server containers:** maximum speed and density



**Hyper-V containers:** isolation plus performance



# Virtualization versus **containerization**

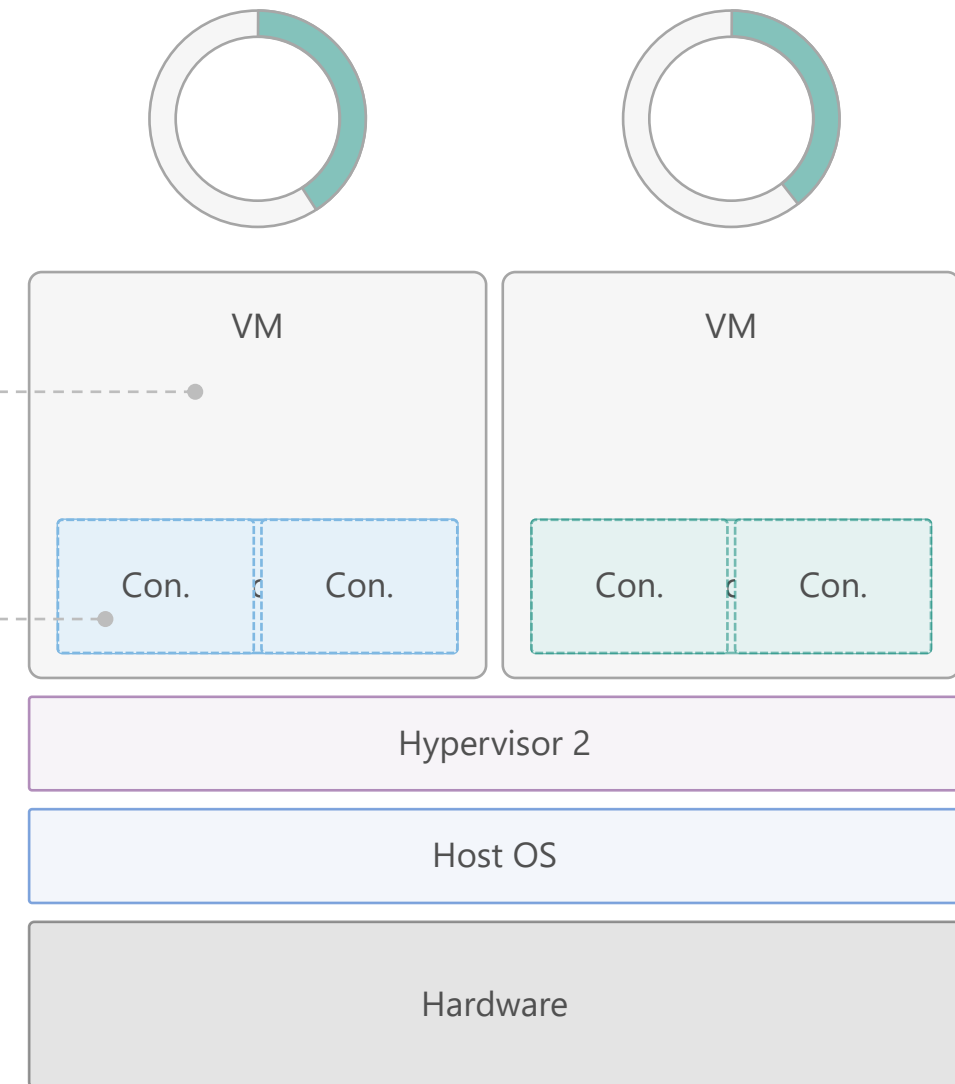


# The container **advantage**

## Traditional virtualized environment

Low utilization of container resources

Containerization of applications and their dependencies

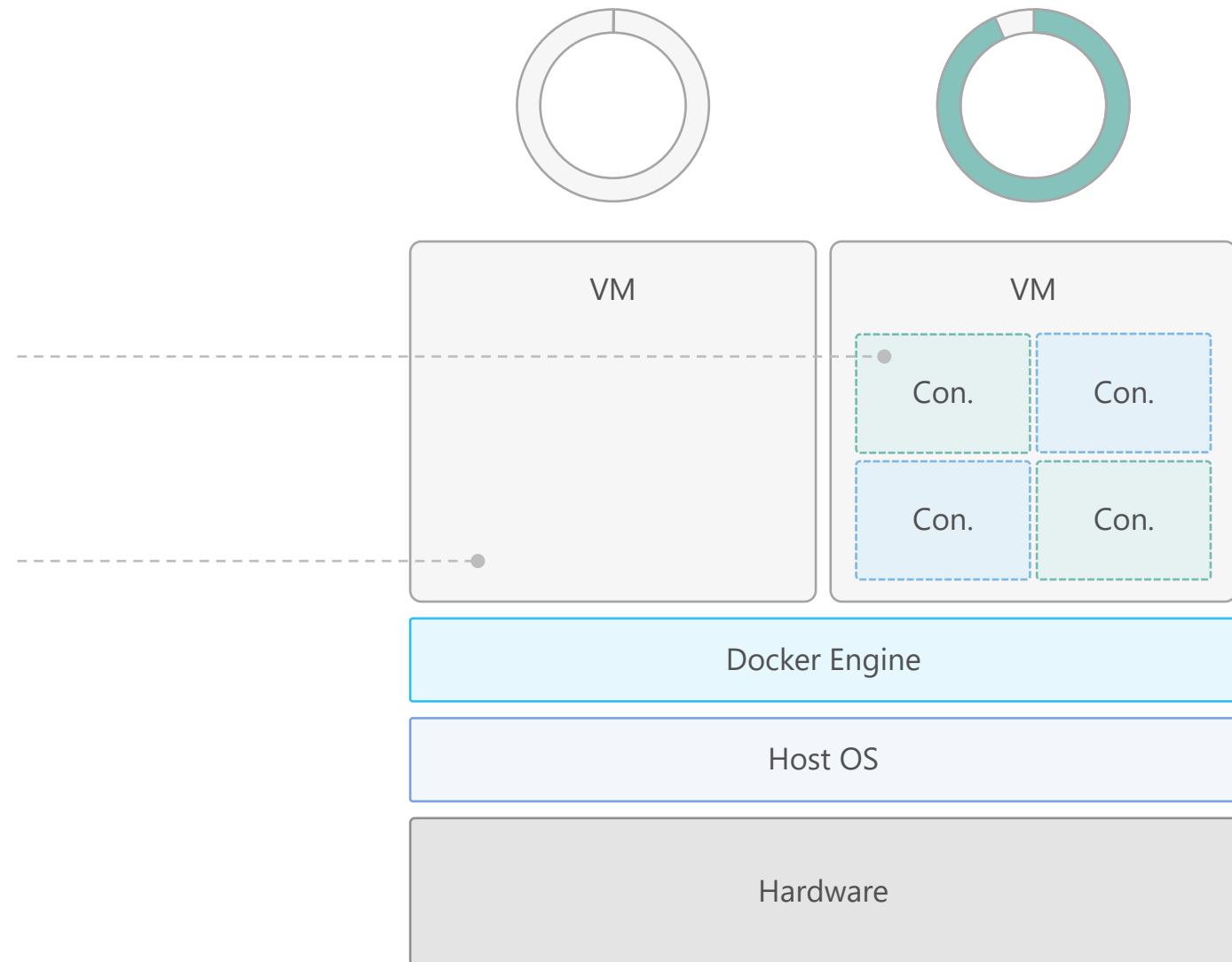


# The container **advantage**

## Containerized environment

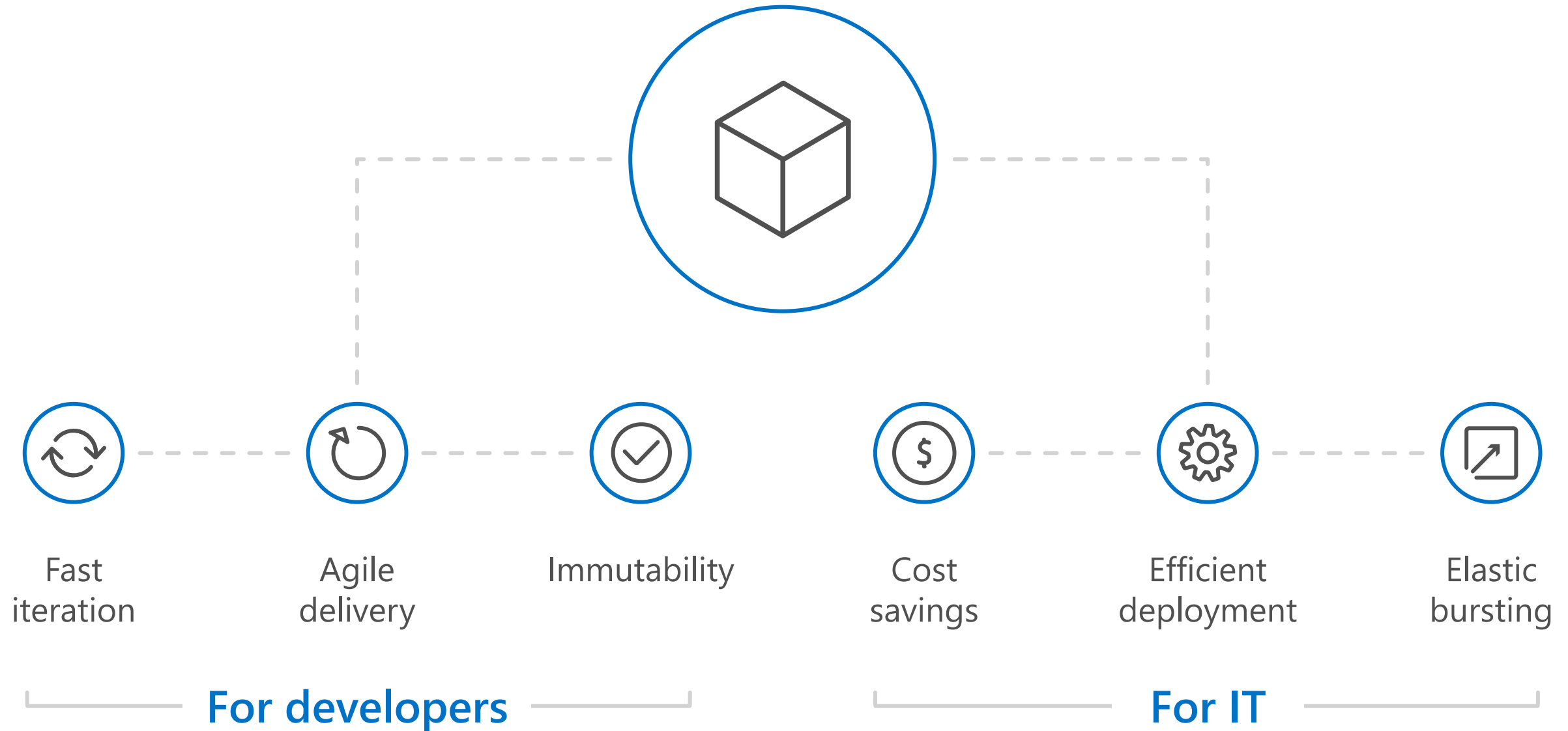
Migrate containers and their dependencies to underutilized VMs for improved density and isolation

Decommission unused resources for efficiency gains and cost savings



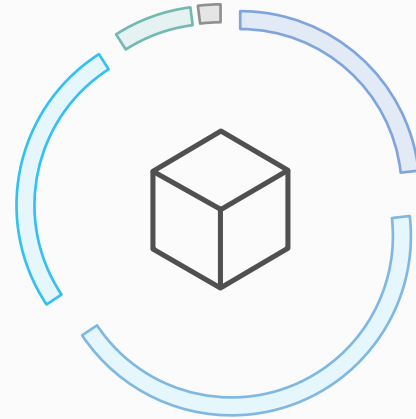
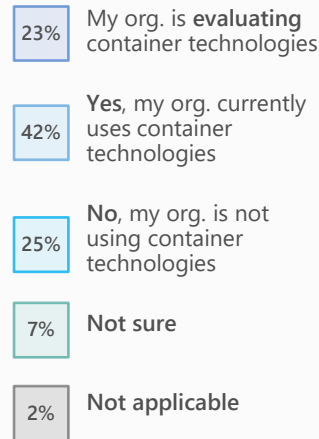


# The container **advantage**



# Containers are gaining **momentum**

Does your organization currently use container technologies?<sup>1</sup>

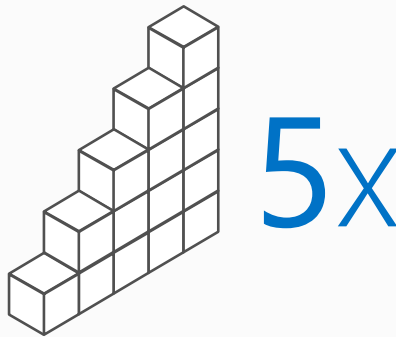


Larger companies are leading adoption.<sup>2</sup>

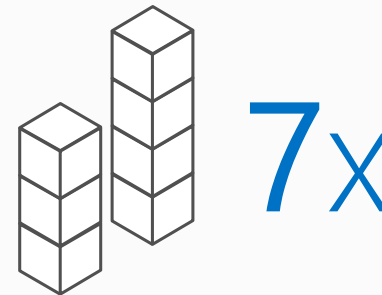
Nearly 60% percent of organizations running 500 or more hosts are classified as **container dabblers** or adopters.



The average company **QUINTUPLES** its container usage within 9 months.<sup>1</sup>



Container hosts often run **SEVEN** containers at a time.<sup>1</sup>



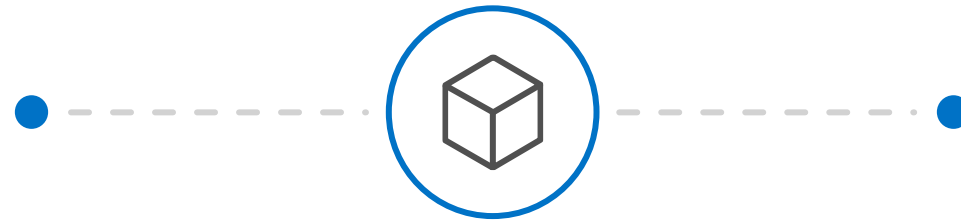
Containers churn 9 times **FASTER** than VMs.<sup>1</sup>



Source:

1: Datadog: 8 Surprising Facts About Real Docker Adoption; 2: DZone: The DZone Guide to Deploying and Orchestrating Containers

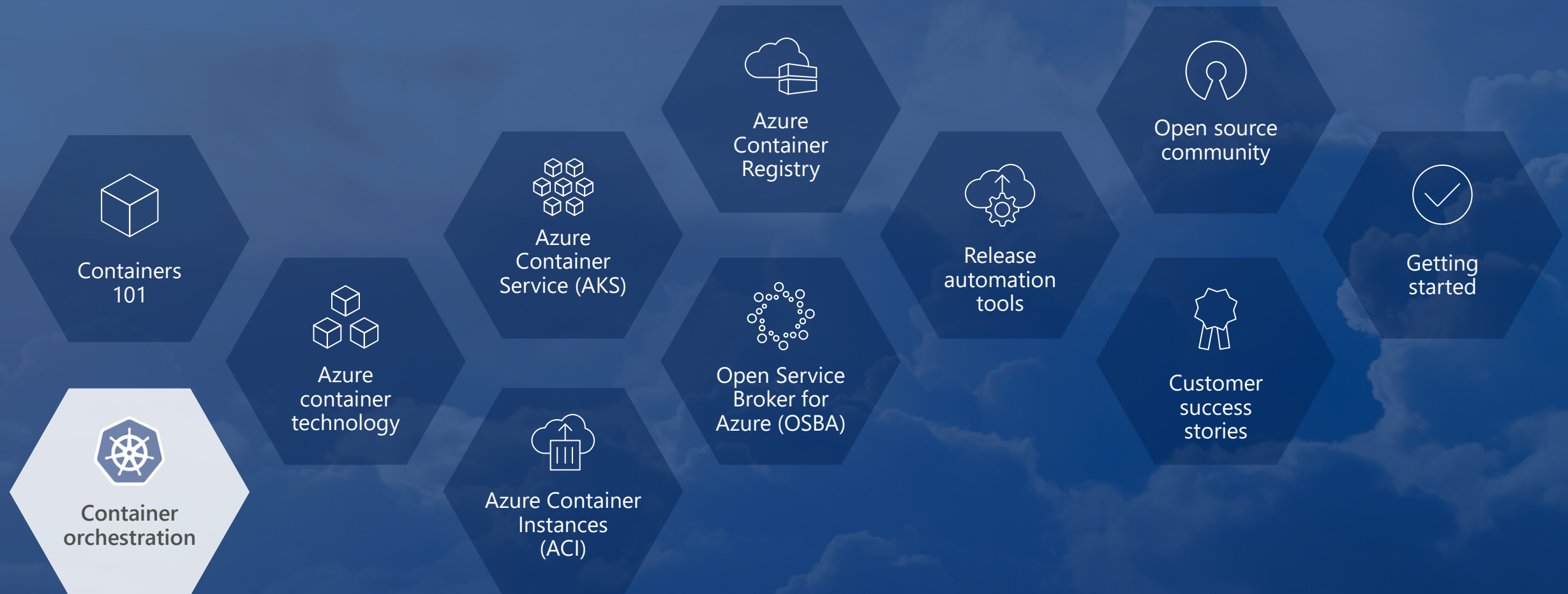
# Industry analysts **agree**



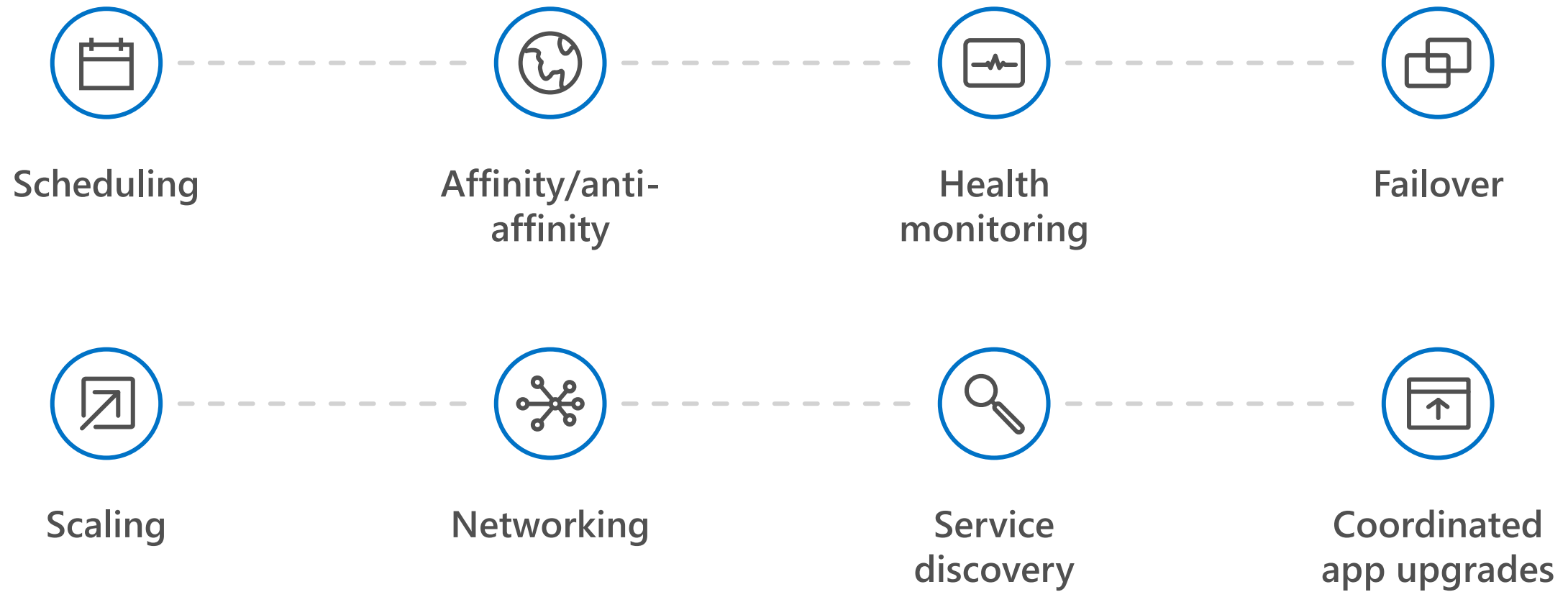
“By 2020, more than 50% of enterprises will run mission-critical, containerized cloud-native applications in production, up from less than 5% today.”

**Gartner**<sup>®</sup>

# Container orchestration



# The elements of **orchestration**



# Kubernetes: the de-facto orchestrator



## Portable

Public, private, hybrid,  
multi-cloud

## Extensible

Modular, pluggable,  
hookable, composable

## Self-healing

Auto-placement, auto-restart,  
auto-replication, auto-scaling

# Kubernetes: empowering you to do more



Deploy your  
applications quickly  
and predictably



Scale your  
applications on  
the fly

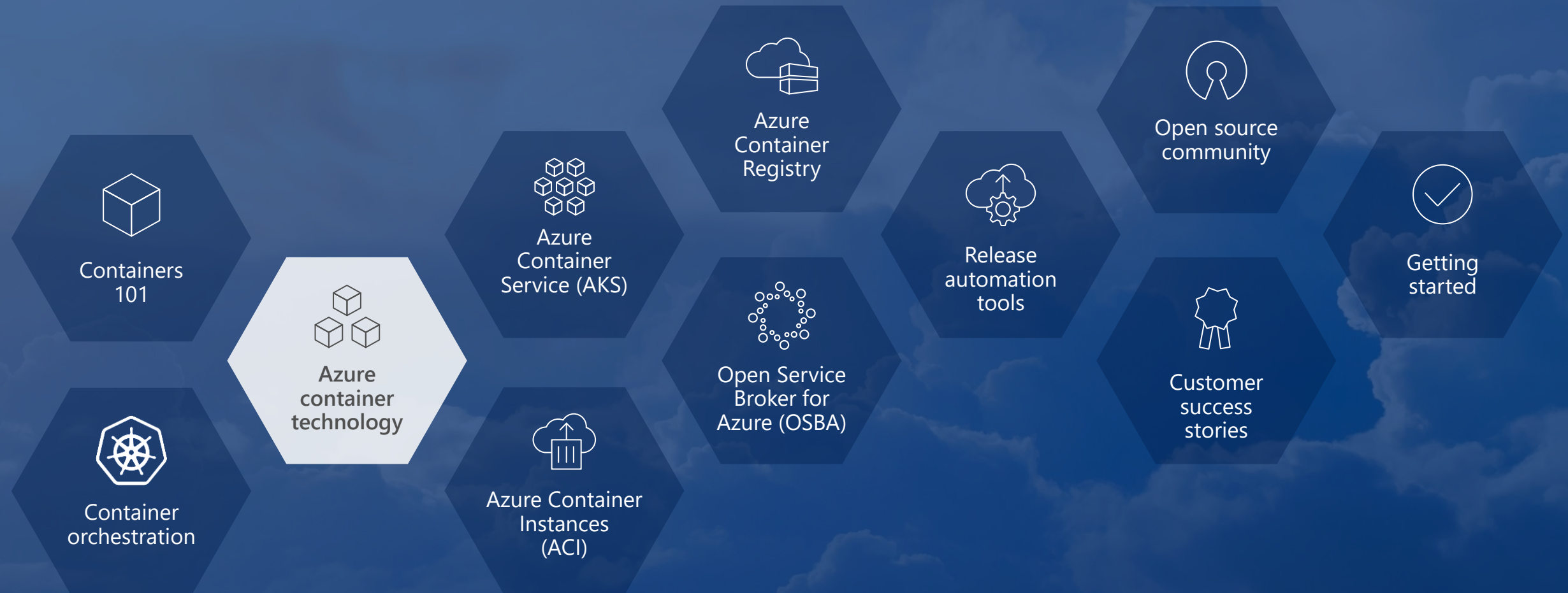


Roll out  
new features  
seamlessly



Limit hardware  
usage to required  
resources only

# Azure container technology

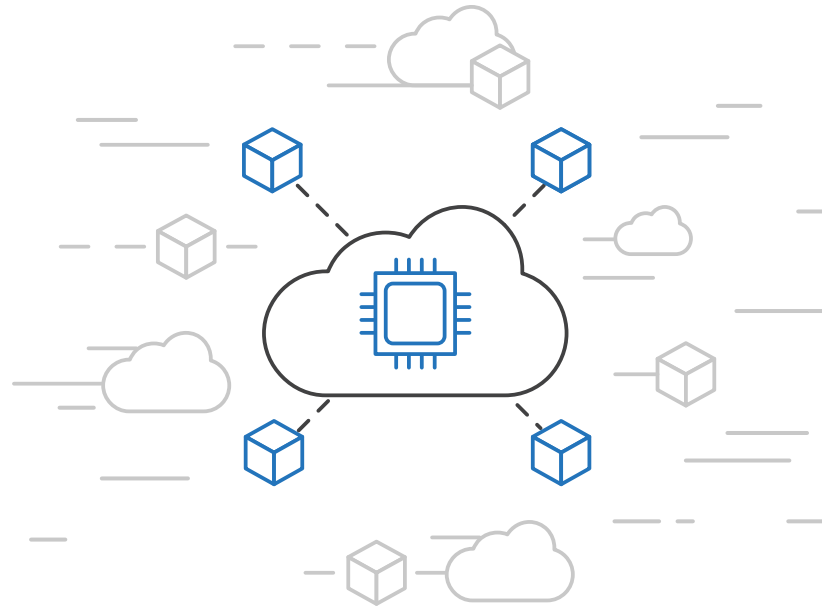




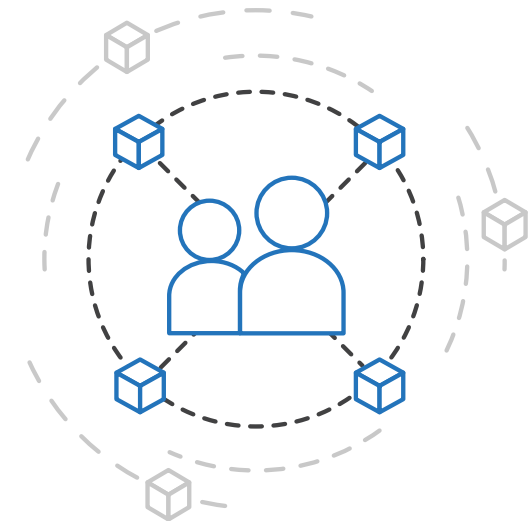
# Azure container **strategy**



Embrace containers  
as ubiquitous

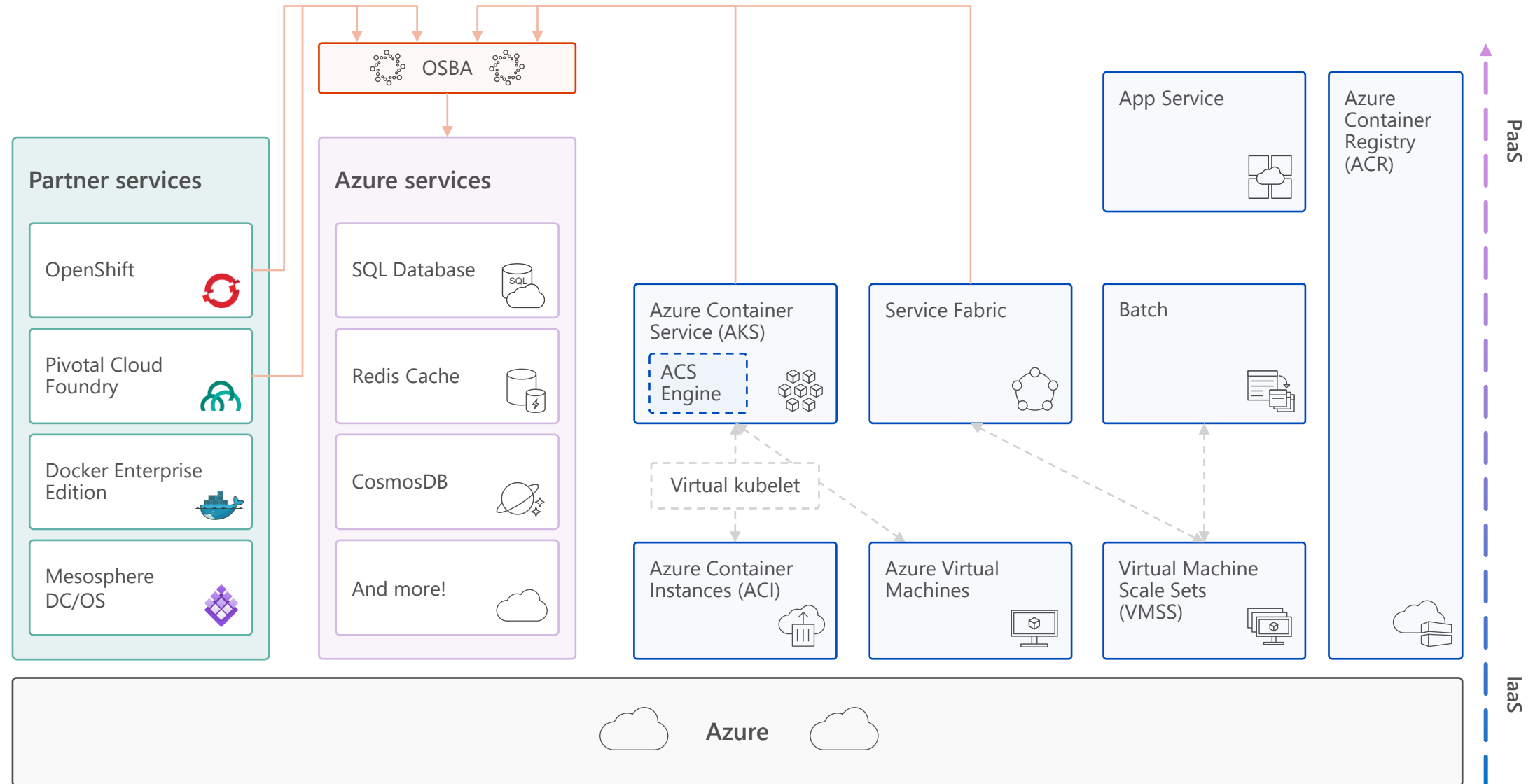


Support containers  
across the compute  
portfolio

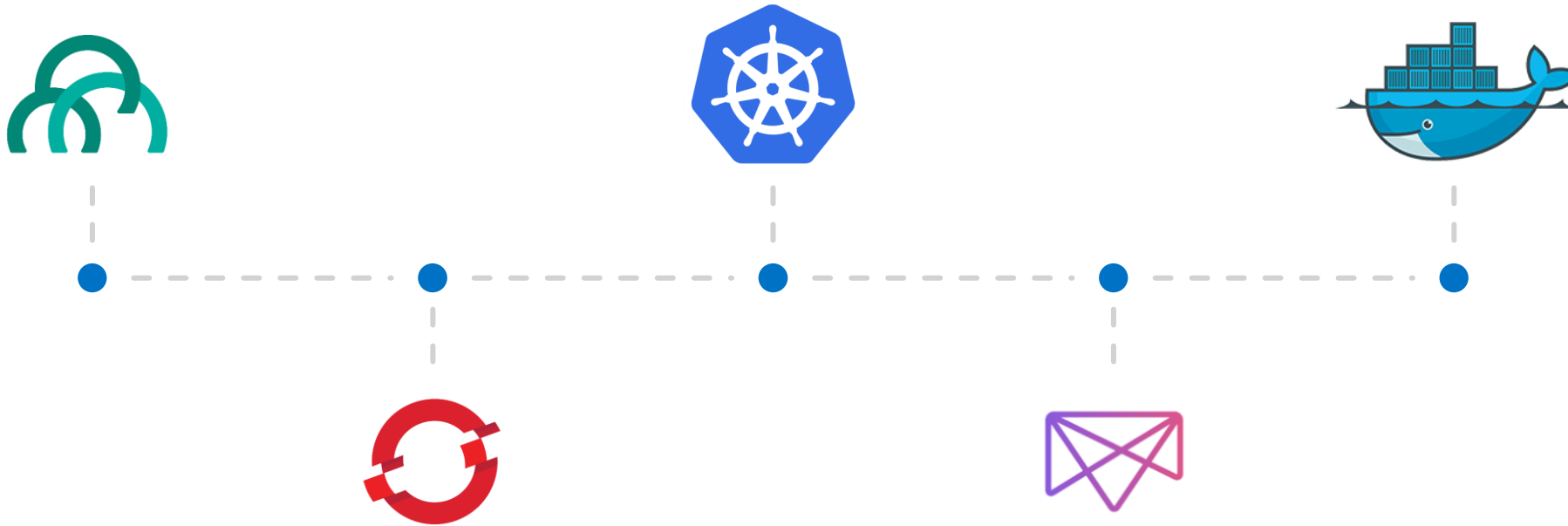


Democratize  
container technology

# Azure container ecosystem

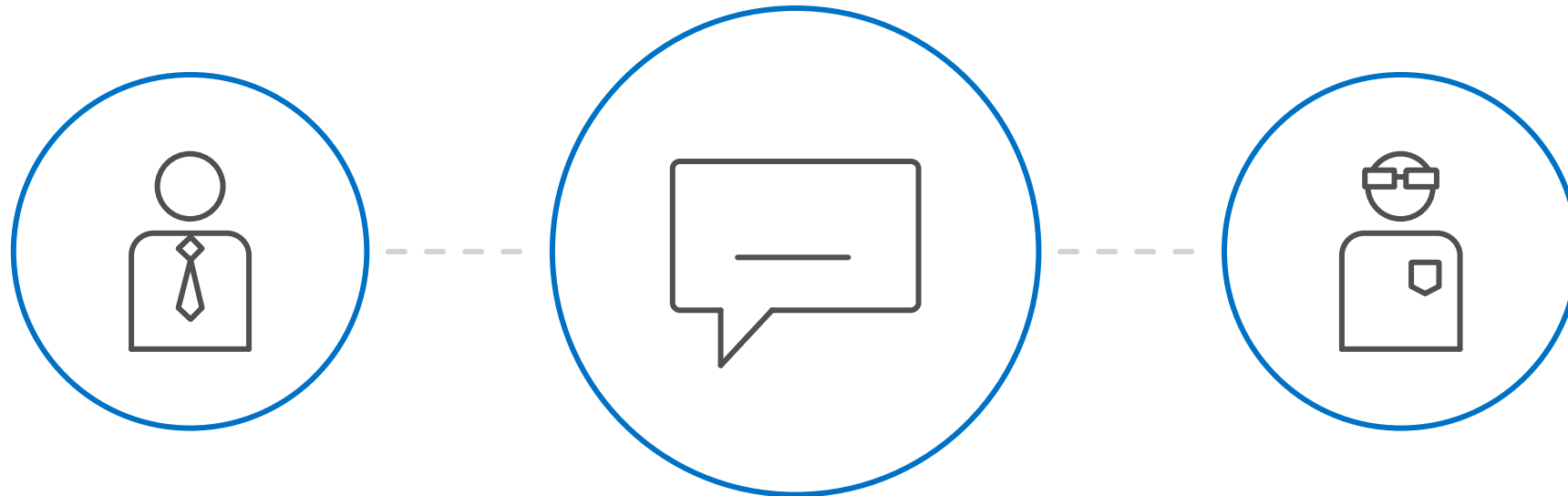


If you have a preferred container platform  
**Pivotal Cloud Foundry • Kubernetes • Docker Enterprise Edition**  
**Red Hat OpenShift • Mesosphere DC/OS**



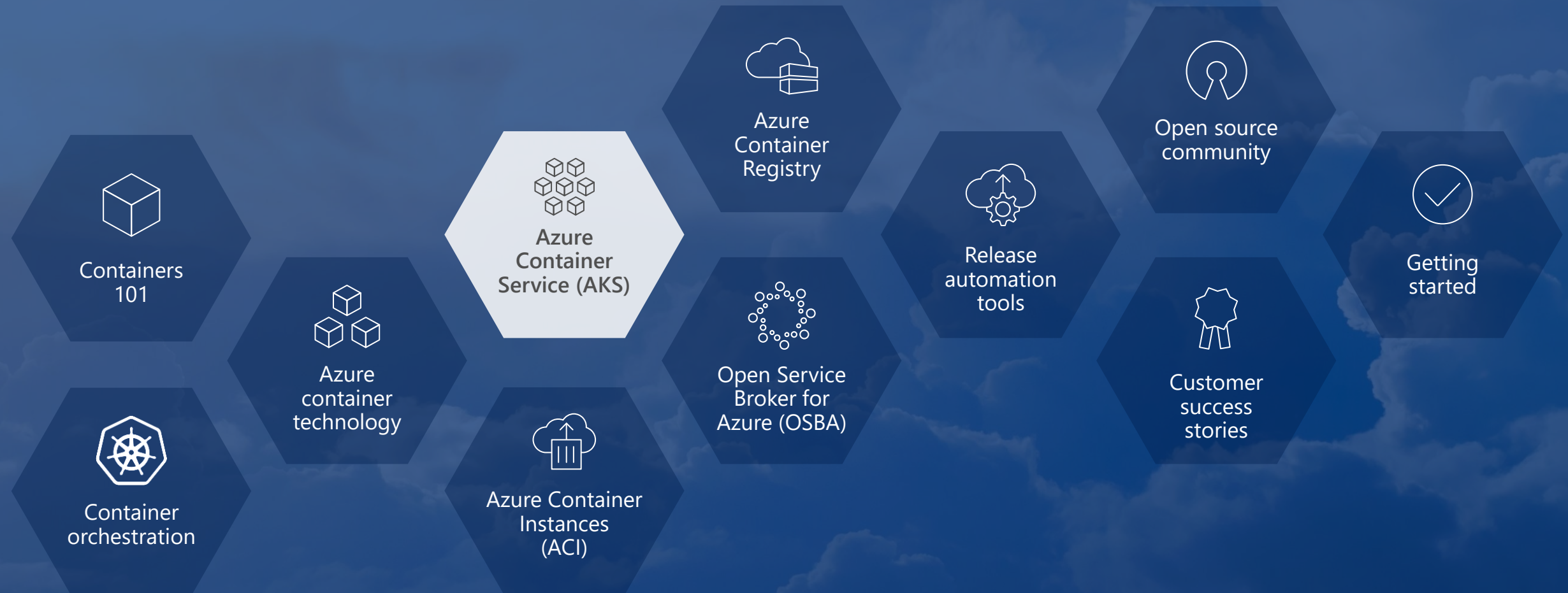
**Lets help you bring that platform to Azure**

If you are without a preferred container platform...



**Lets profile your needs and help you select the right  
option**

# Azure Container Service (AKS)





Azure Container Service (AKS)



Azure Container Instances (ACI)



Azure Container Registry

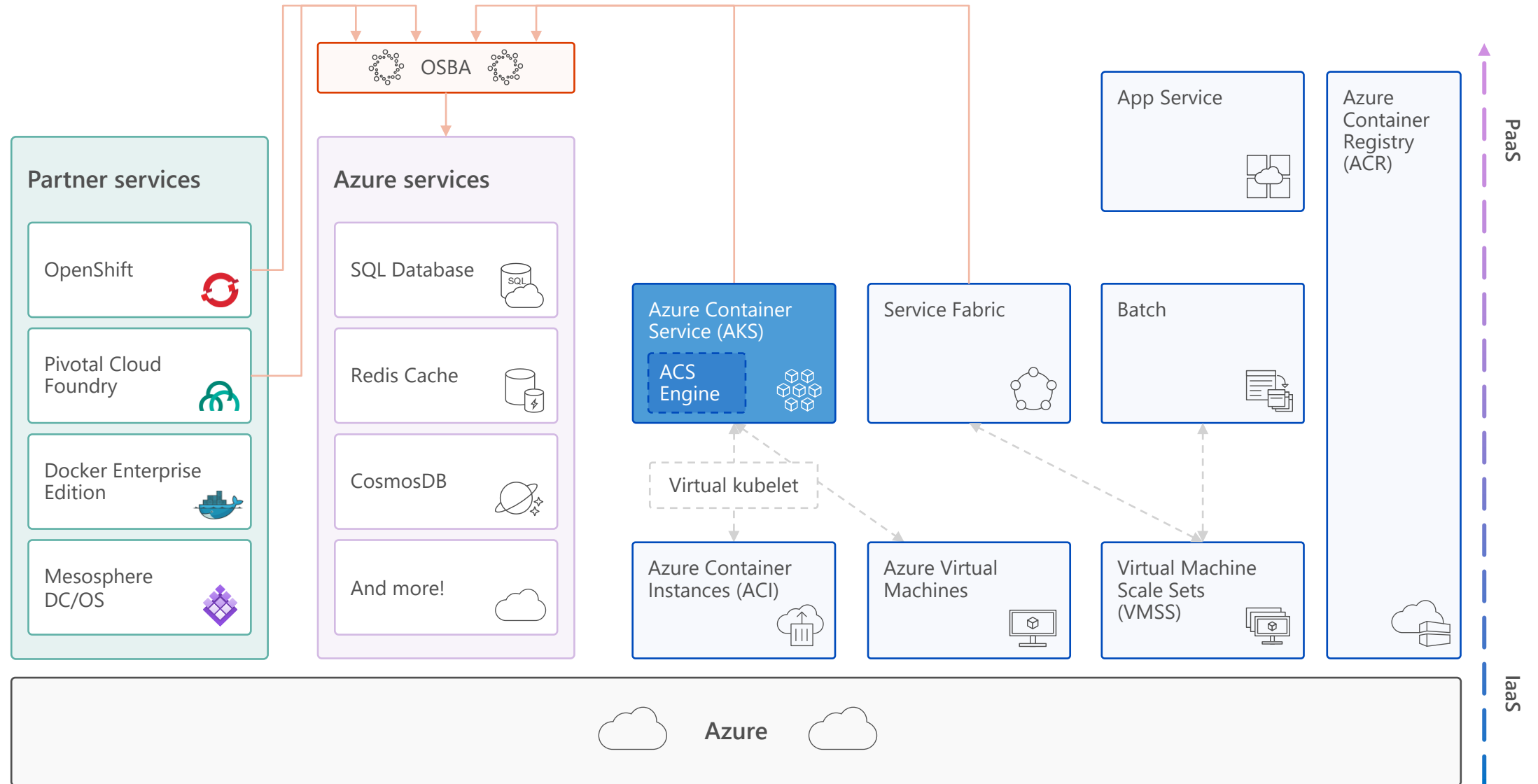


Open Service Broker API (OSBA)



Release Automation Tools

# Azure Container Service (AKS)





Azure Container  
Service (AKS)



Azure Container  
Instances (ACI)



Azure Container  
Registry



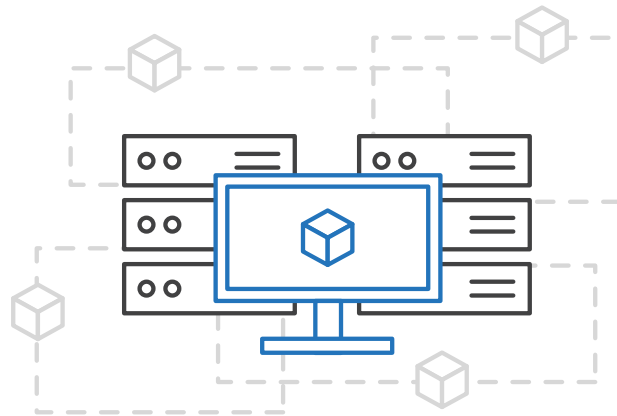
Open Service  
Broker API (OSBA)



Release  
Automation Tools

# Azure Container Service (AKS)

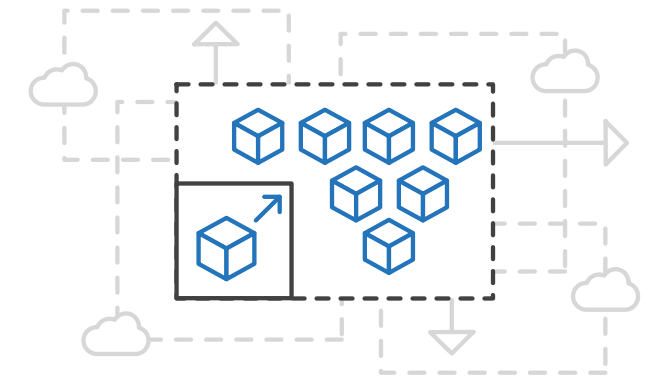
Simplify the deployment, management, and  
operations of Kubernetes



Focus on your  
containers not the  
infrastructure



Work how you  
want with open-  
source APIs



Scale and run  
applications with  
confidence





Azure Container  
Service (AKS)



Azure Container  
Instances (ACI)



Azure Container  
Registry



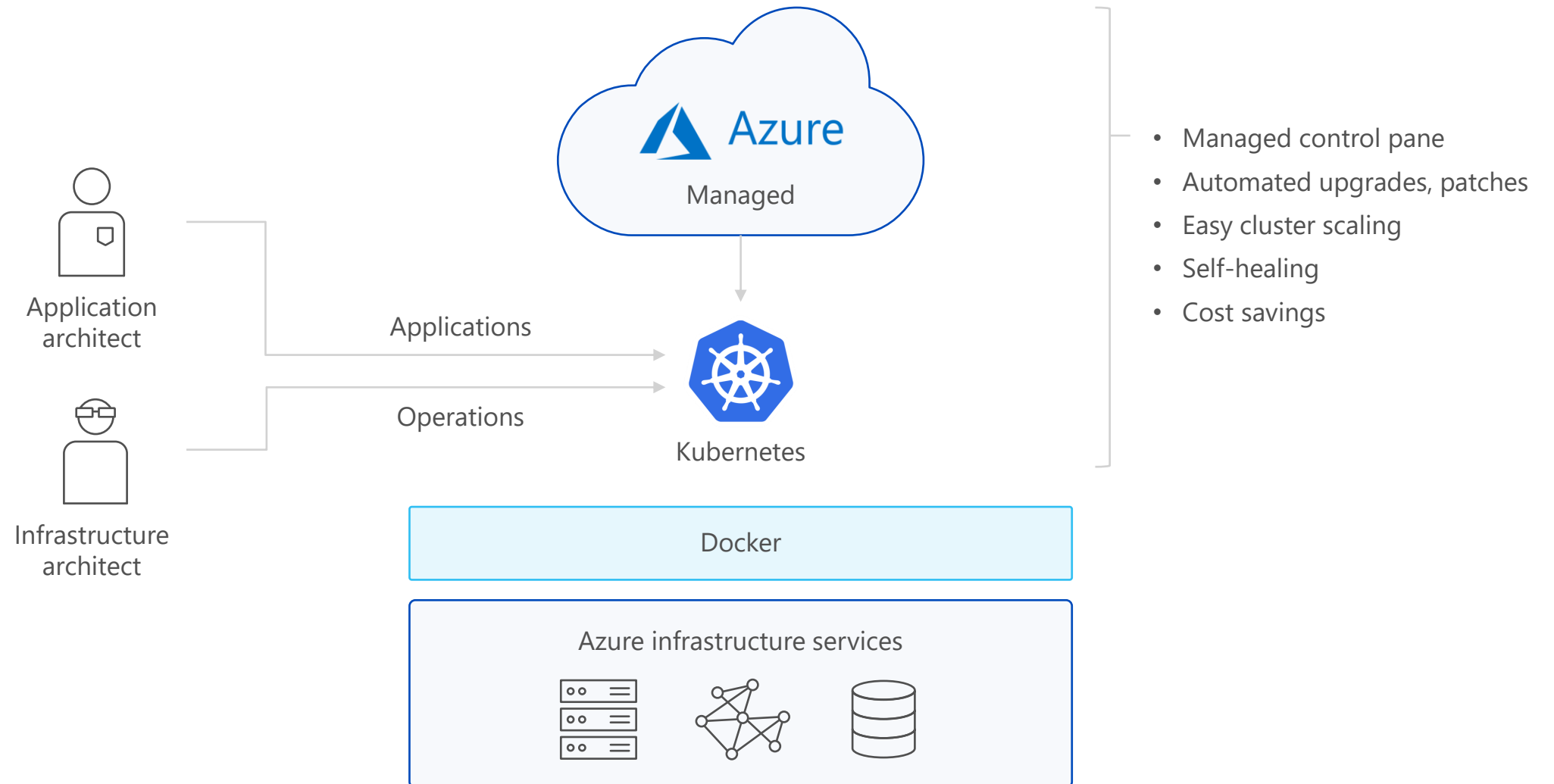
Open Service  
Broker API (OSBA)



Release  
Automation Tools

# Azure Container Service (AKS)

A fully managed Kubernetes cluster







Azure Container  
Service (AKS)



Azure Container  
Instances (ACI)



Azure Container  
Registry



Open Service  
Broker API (OSBA)



Release  
Automation Tools

# Azure Container Service (AKS)

## Get started easily

```
$ az aks create -g myResourceGroup -n user**Cluster --generate-ssh-keys  
\ Running ..
```

```
$ az aks install-cli (Azure Cloudshell에서는 불필요)  
Downloading client to /usr/local/bin/kubectl ..
```

```
$ az aks get-credentials -g myResourceGroup -n user**Cluster  
Merged "myCluster" as current context ..
```

```
$ kubectl get nodes
```

NAME	STATUS	AGE	VERSION
aks-mycluster-36851231-0	Ready	4m	v1.18.1
aks-mycluster-36851231-1	Ready	4m	v1.18.1
aks-mycluster-36851231-2	Ready	4m	v1.18.1



Azure Container  
Service (AKS)



Azure Container  
Instances (ACI)



Azure Container  
Registry



Open Service  
Broker API (OSBA)



Release  
Automation Tools

# Azure Container Service (AKS)

## Manage an AKS cluster

```
$ az aks list -o table ( 혹은 az aks list --resource-group myResourceGroup -o table)
```

Name	Location	ResourceGroup	KubernetesRelease	ProvisioningState
myCluster	westus2	myResourceGroup	1.18.1	Succeeded

```
$ az aks get-upgrades --resource-group myResourceGroup --name myCluster -o table
```

(해당 Region에서 지원되는 K8s버전 확인)

```
$ az aks upgrade -g myResourceGroup -n user**Cluster --kubernetes-version 1.19.1
```

\ Running .. (대략 20분 소요)

```
$ kubectl get nodes
```

NAME	STATUS	AGE	VERSION
aks-mycluster-36851231-0	Ready	12m	v1.19.1
aks-mycluster-36851231-1	Ready	8m	v1.19.1
aks-mycluster-36851231-2	Ready	3m	v1.19.1

```
$ az aks scale -g myResourceGroup -n user**Cluster --node-count 4
```

\ Running ..



Azure Container  
Service (AKS)



Azure Container  
Instances (ACI)



Azure Container  
Registry



Open Service  
Broker API (OSBA)



Release  
Automation Tools

# Azure Container Service (AKS)

Create an AKS cluster via the Azure portal

Microsoft Azure

Report a bug

Search resources, services and documentation

Home > New

Create a resource

All services

FAVORITES

- Dashboard
- All resources
- Resource groups
- App Services
- SQL databases
- Azure Cosmos DB

Azure Marketplace [See all](#)

Featured [See all](#)

- Get started
- Recently created
- Compute
- Networking
- Storage
- Web + Mobile
- Containers**
- Databases
- Data + Analytics

**Azure Container Service - AKS (preview)**  
[Learn more](#)

**Azure Container Service**  
[Learn more](#)

**Azure Container Instances (preview)**  
[Learn more](#)

**Azure Container Registry**  
[Learn more](#)



Azure Container  
Service (AKS)



Azure Container  
Instances (ACI)



Azure Container  
Registry



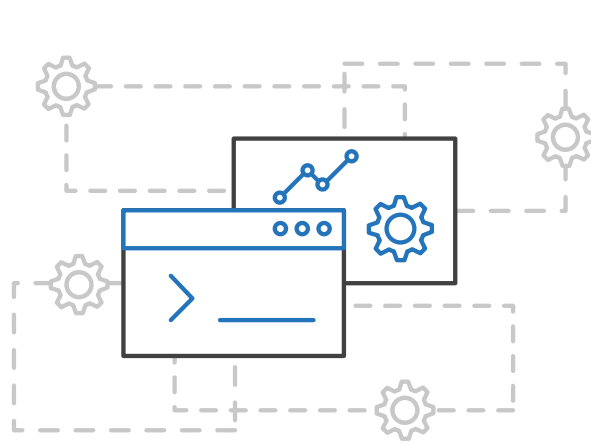
Open Service  
Broker API (OSBA)



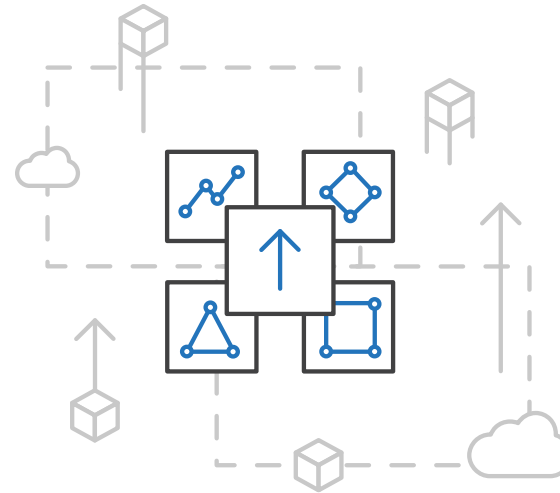
Release  
Automation Tools

# Azure Container Service (AKS)

## Azure Container Service Engine



A proving ground  
for new features



Enables custom  
deployments



Available  
on GitHub





Azure Container  
Service (AKS)



Azure Container  
Instances (ACI)



Azure Container  
Registry



Open Service  
Broker API (OSBA)



Release  
Automation Tools

# Azure Container Service (AKS)

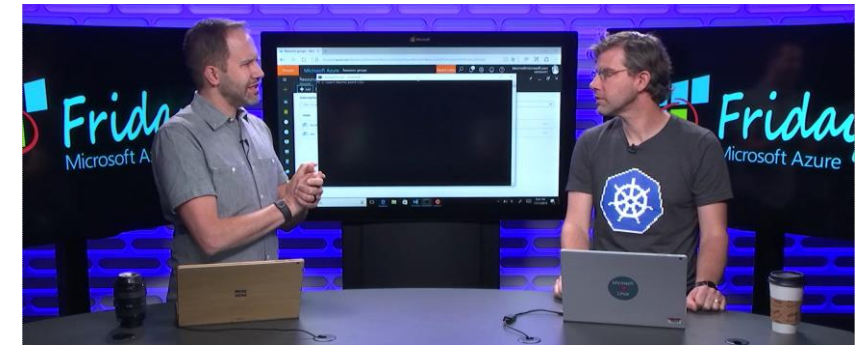
## Resources

- [Azure Container Service \(AKS\) webpage](#)
- [AKS videos](#)
- [AKS technical documentation](#)
- [AKS pricing details](#)
- [AKS roadmap](#)
- [Azure Container Service Engine: Github](#)

### Container Orchestration Simplified with AKS

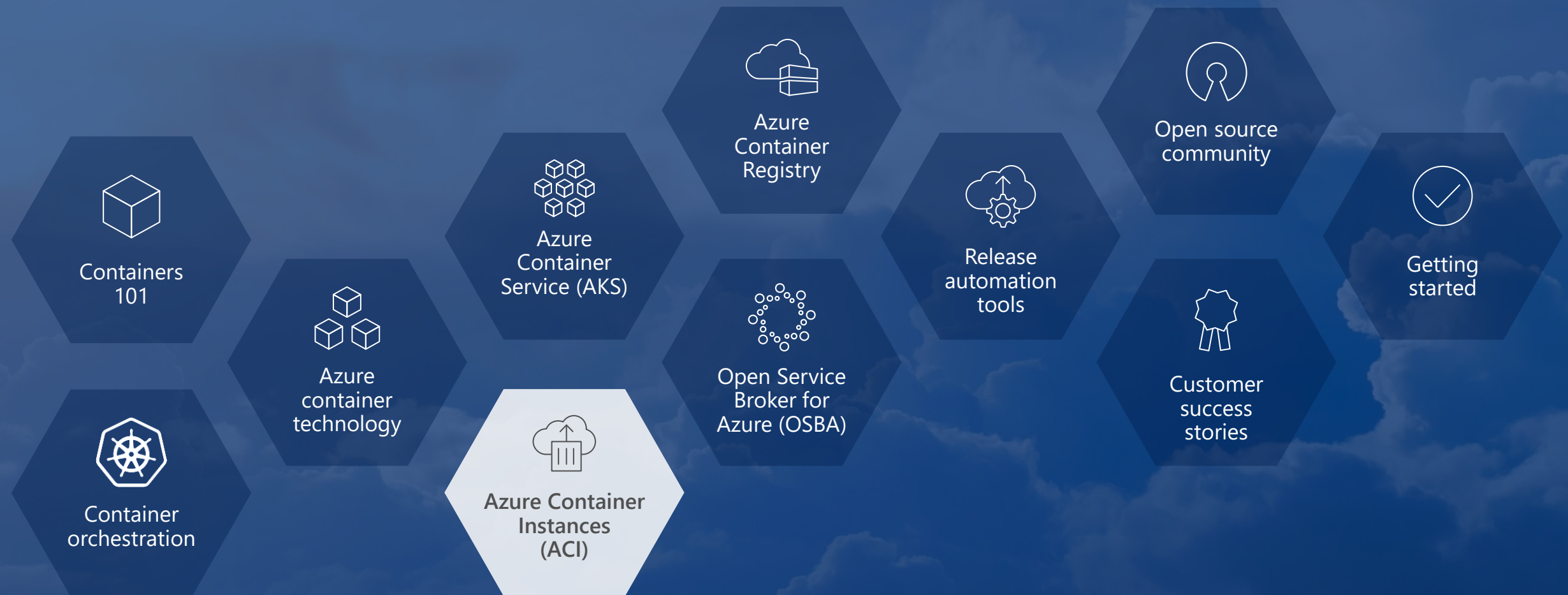


### Kubernetes Support in Azure Container Services





# Azure Container Instances (ACI)





Azure Container Service (AKS)



Azure Container Instances (ACI)



Azure Container Registry

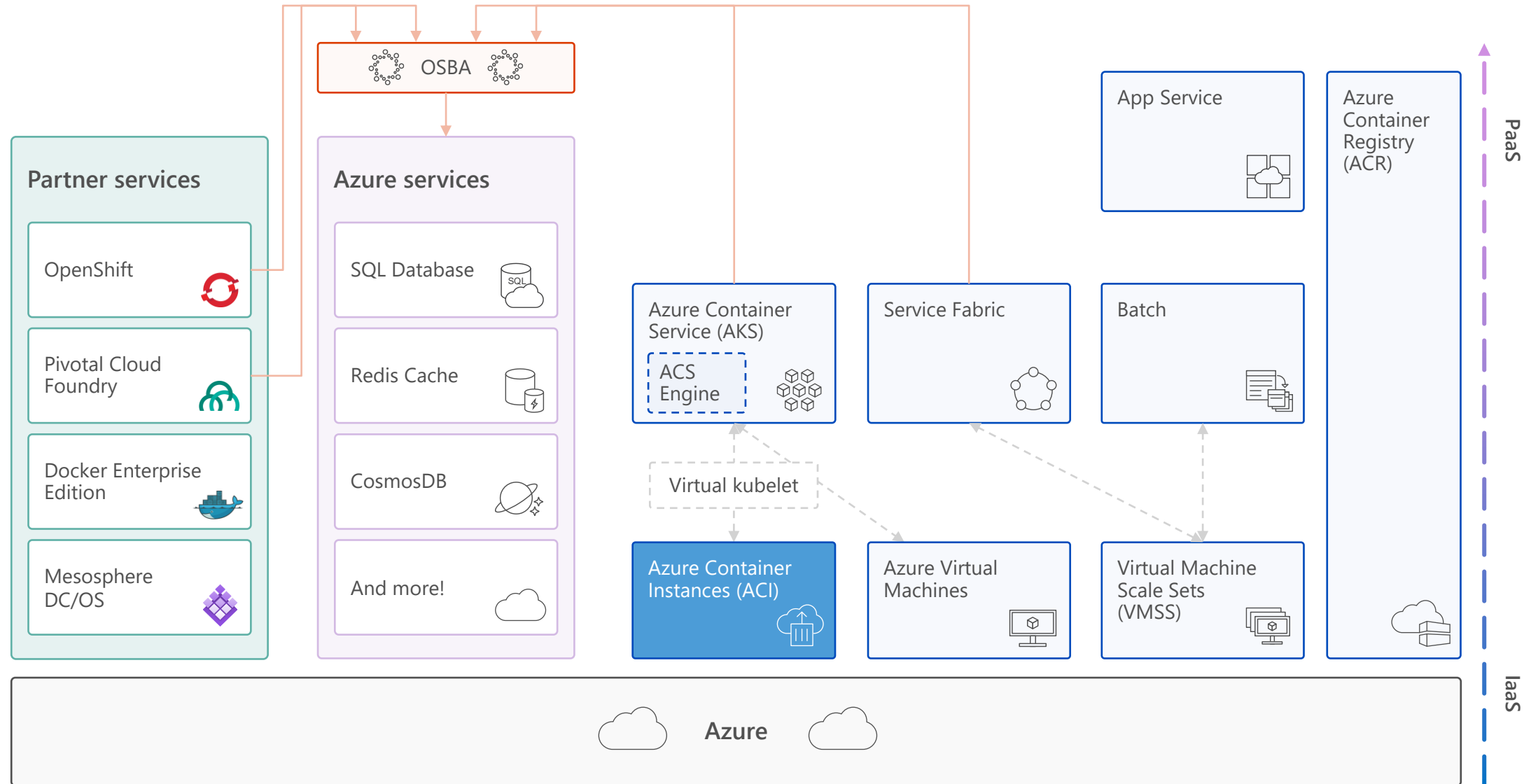


Open Service Broker API (OSBA)



Release Automation Tools

# Azure Container Instances (ACI) **PREVIEW**





Azure Container  
Service (AKS)



Azure Container  
Instances (ACI)



Azure Container  
Registry



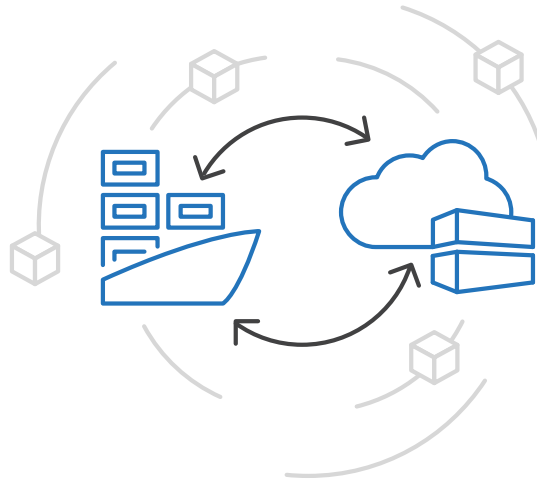
Open Service  
Broker API (OSBA)



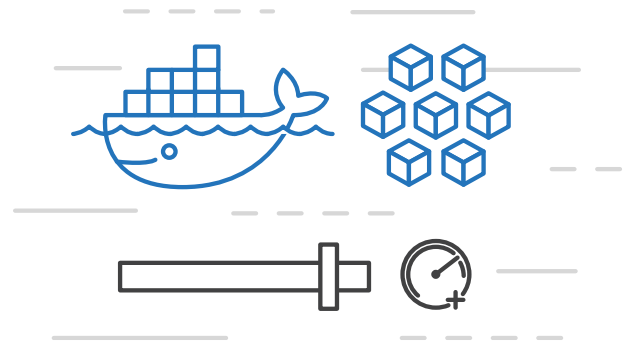
Release  
Automation Tools

# Azure Container Instances (ACI) <sup>PREVIEW</sup>

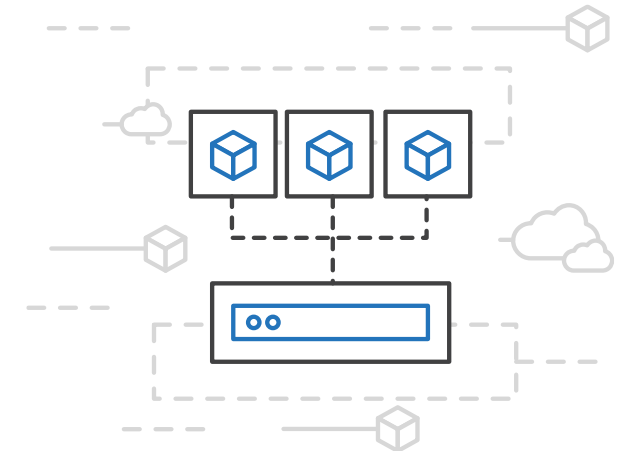
Easily run containers on Azure with a single command



Start using  
containers right away



Cloud-scale  
container capacity



Hyper-visor  
isolation







Azure Container  
Service (AKS)



Azure Container  
Instances (ACI)



Azure Container  
Registry



Open Service  
Broker API (OSBA)



Release  
Automation Tools

# Azure Container Instances (ACI) **PREVIEW**

Get started easily

```
$az container create --name mycontainer --image microsoft/aci-helloworld --  
resource-group myResourceGroup --ip-address public
```

```
  "ipAddress": {  
    "ip": "52.168.86.133",  
    "ports": [...]  
  },  
  "location": "eastus",  
  "name": "mycontainer",  
  "osType": "Linux",  
  "provisioningState": "Succeeded",
```

```
$ curl 52.168.86.133
```

```
<html>  
<head>  
  <title>Welcome to Azure Container Instances!</title>  
</head>
```



Azure Container  
Service (AKS)



Azure Container  
Instances (ACI)



Azure Container  
Registry



Open Service  
Broker API (OSBA)



Release  
Automation Tools

# Azure Container Instances (ACI) **PREVIEW**

Create an Azure Container Instance quickly

Microsoft Azure

Report a bug Search resources, services and more

Home > New

Create a resource

All services

FAVORITES

- Dashboard
- All resources
- Resource groups
- App Services
- SQL databases
- Azure Cosmos DB

Azure Marketplace See all

Featured See all

- Get started
- Recently created
- Compute
- Networking
- Storage
- Web + Mobile
- Containers**
- Databases
- Data + Analytics

Azure Container Service - AKS (preview) Learn more

Azure Container Service Learn more

**Azure Container Instances (preview) Learn more**

Azure Container Registry Learn more



Azure Container Service (AKS)



Azure Container Instances (ACI)



Azure Container Registry



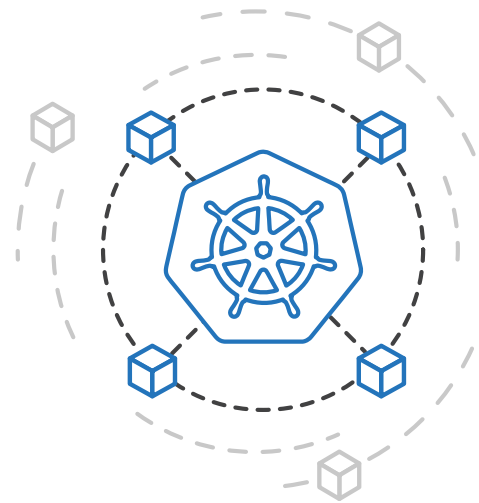
Open Service Broker API (OSBA)



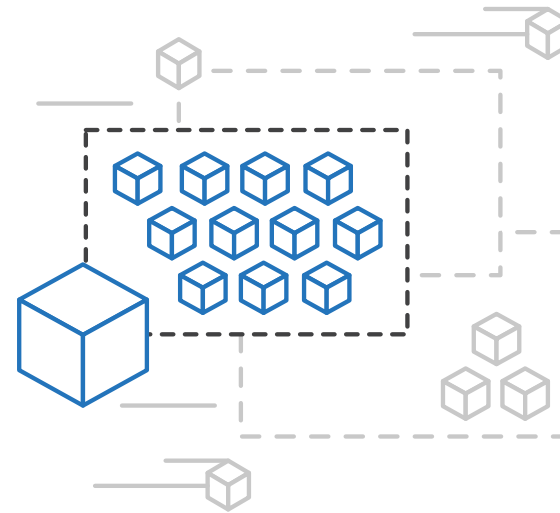
Release Automation Tools

# Azure Container Instances (ACI) <sup>PREVIEW</sup>

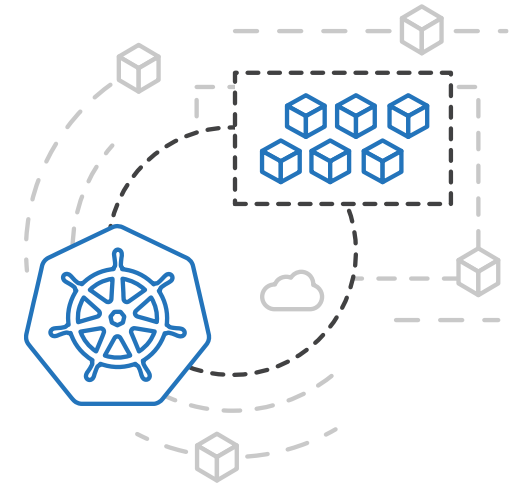
## ACI Connector for Kubernetes



Kubernetes provides rich orchestration capabilities



ACI provides infinite container-based scale



The ACI Connector for K8s brings them together





Azure Container Service (AKS)



Azure Container Instances (ACI)



Azure Container Registry



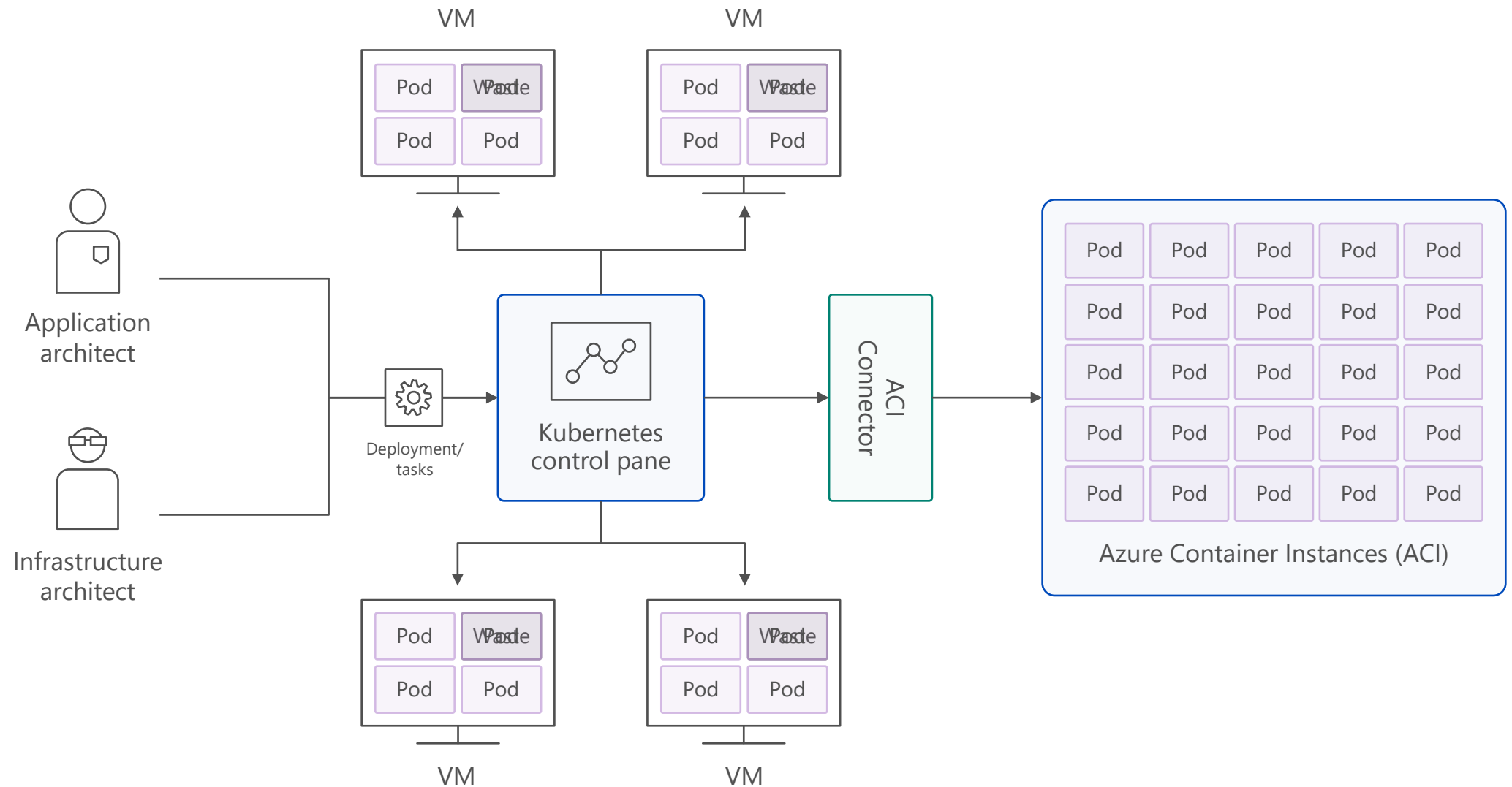
Open Service Broker API (OSBA)



Release Automation Tools

# Azure Container Instances (ACI) **PREVIEW**

## Bursting with the ACI Connector





Azure Container  
Service (AKS)



Azure Container  
Instances (ACI)



Azure Container  
Registry



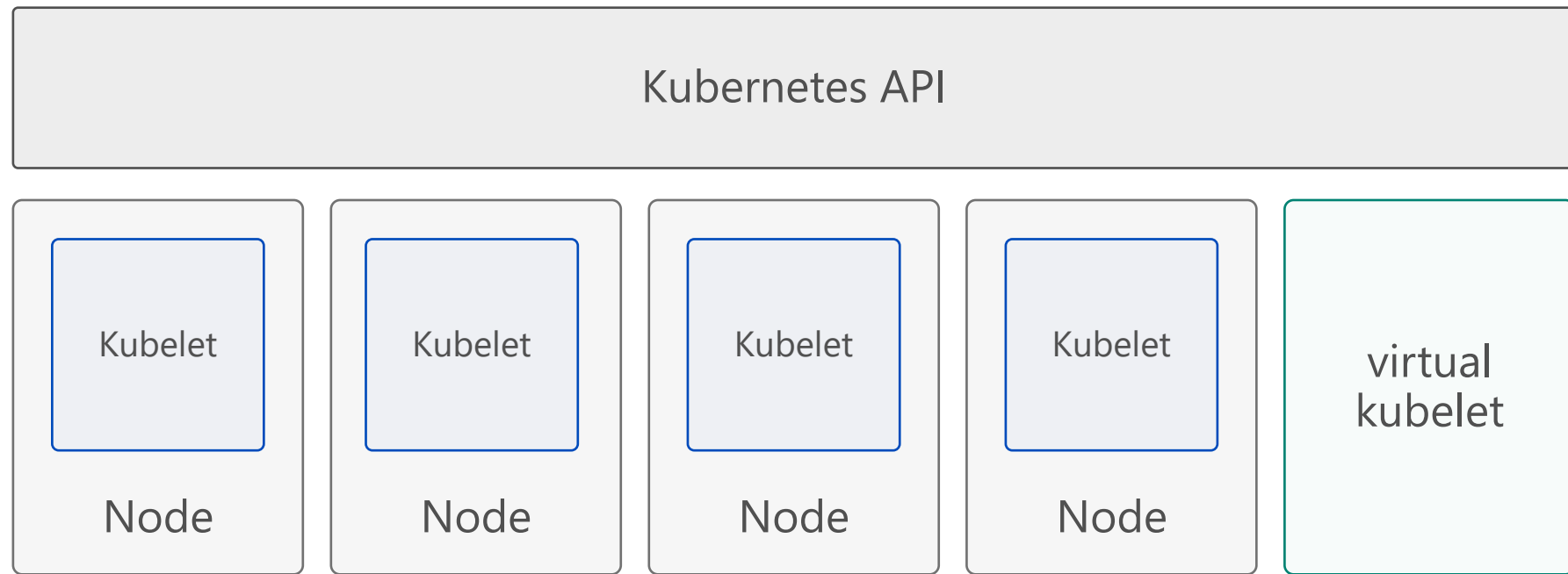
Open Service  
Broker API (OSBA)



Release  
Automation Tools

# Azure Container Instances (ACI) <sup>PREVIEW</sup>

## Virtual Kubelet



Typical kubelets implement the pod and container operations for each node as usual.

Virtual kubelet registers itself as a “node” and allows developers to program their own behaviors for operations on pods and containers.



Azure Container  
Service (AKS)



Azure Container  
Instances (ACI)



Azure Container  
Registry



Open Service  
Broker API (OSBA)



Release  
Automation Tools

# Azure Container Instances (ACI) PREVIEW

## Resources

- [Azure Container Instances \(ACI\) webpage](#)
- [ACI videos](#)
- [ACI technical documentation](#)
- [ACI pricing details](#)
- [ACI roadmap](#)

### Azure Container Instances

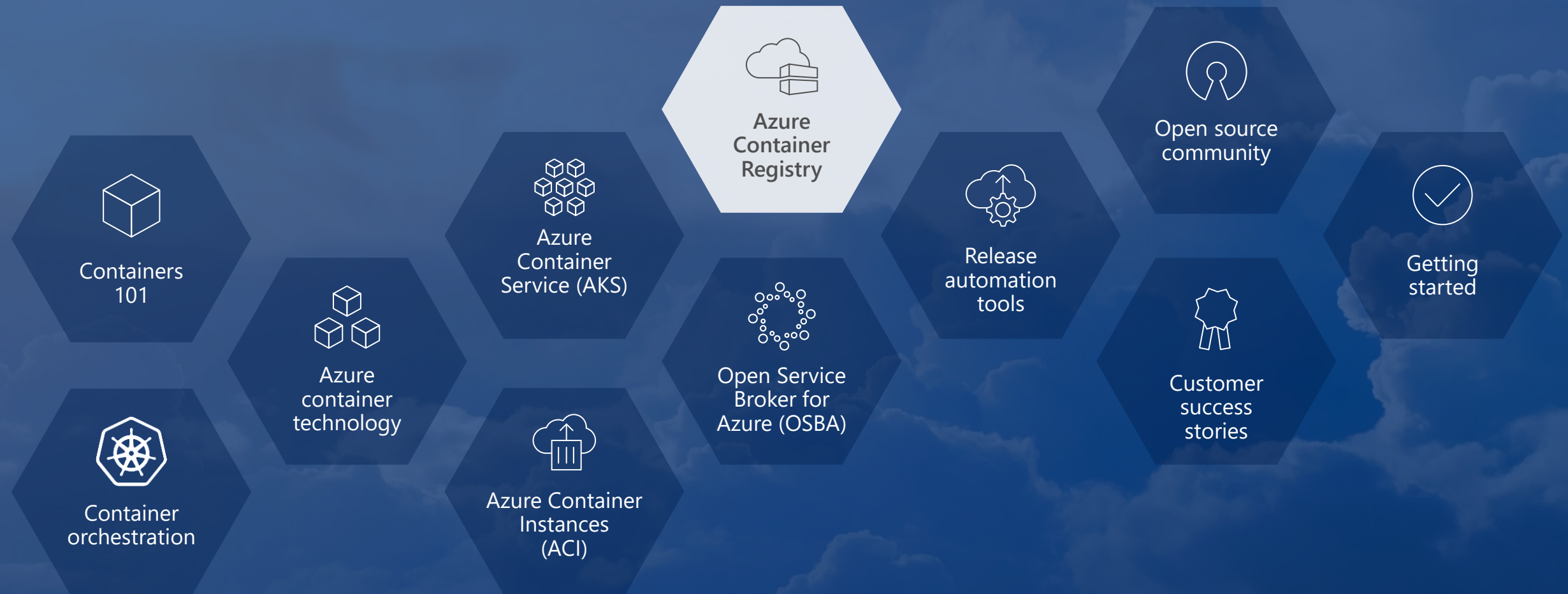


### Using Kubernetes with Azure Container Instances





# Azure Container Registry





Azure Container Service (AKS)



Azure Container Instances (ACI)



Azure Container Registry

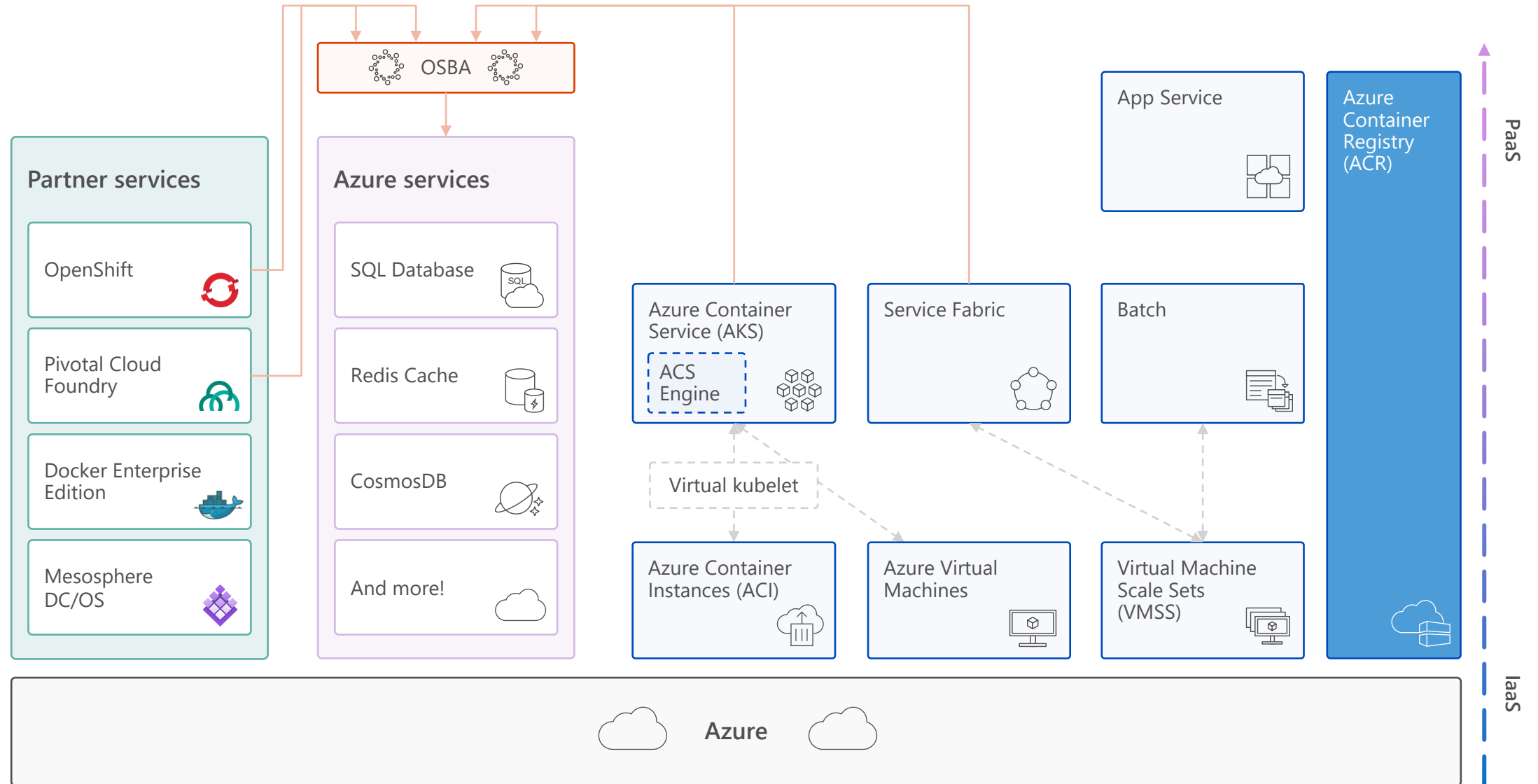


Open Service Broker API (OSBA)



Release Automation Tools

# Azure Container Registry







Azure Container Service (AKS)



Azure Container Instances (ACI)



Azure Container Registry



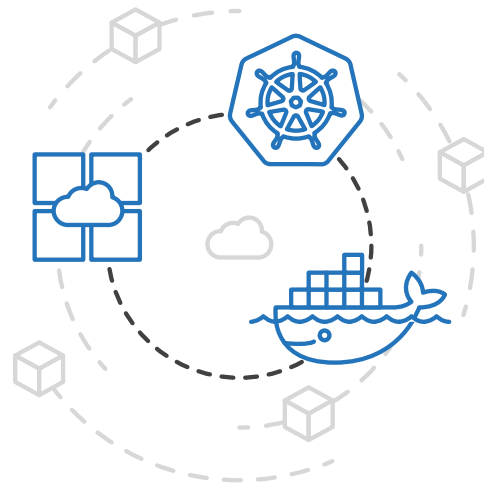
Open Service Broker API (OSBA)



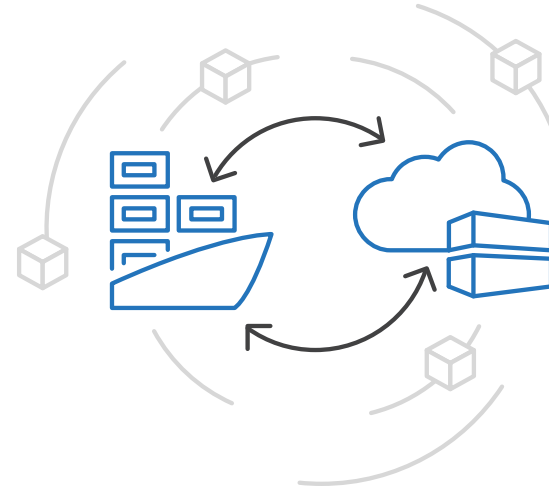
Release Automation Tools

# Azure Container Registry

Manage a Docker private registry as a first-class Azure resource



Manage images for all types of containers



Use familiar, open-source Docker CLI tools



Azure Container Registry geo-replication





Azure Container Service (AKS)



Azure Container Instances (ACI)



Azure Container Registry



Open Service Broker API (OSBA)



Release Automation Tools

# Azure Container Registry

Create a container in the Registry quickly

The screenshot shows the Microsoft Azure portal interface. On the left is a navigation sidebar with options like 'Create a resource', 'All services', 'FAVORITES', 'Dashboard', 'All resources', 'Resource groups', 'App Services', 'SQL databases', and 'Azure Cosmos DB'. The main area is titled 'New' and contains a list of categories: 'Get started', 'Recently created', 'Compute', 'Networking', 'Storage', 'Web + Mobile', 'Containers' (highlighted with a dashed red box), 'Databases', and 'Data + Analytics'. To the right of these categories are 'Featured' items, including 'Azure Container Service - AKS (preview)', 'Azure Container Service', 'Azure Container Instances (preview)', and 'Azure Container Registry' (highlighted with a dashed red box). The top of the page has the 'Microsoft Azure' logo, a 'Report a bug' button, and a search bar.



Azure Container  
Service (AKS)



Azure Container  
Instances (ACI)



Azure Container  
Registry



Open Service  
Broker API (OSBA)



Release  
Automation Tools

# Azure Container Registry

## Resources

- [Azure Container Registry webpage](#)
- [Registry technical documentation](#)
- [Registry pricing details](#)
- [Registry roadmap](#)

### Creating, configuring the Azure Container Registry


Why: Azure Container Registry

**Keep Your Images Private**  
Stored in Azure with your resources

**Network-Close**  
Deployed to your targets within the same data center  
No ingress/egress fees or latency

**Azure Active Directory Integration**  
Manage registry access using AAD

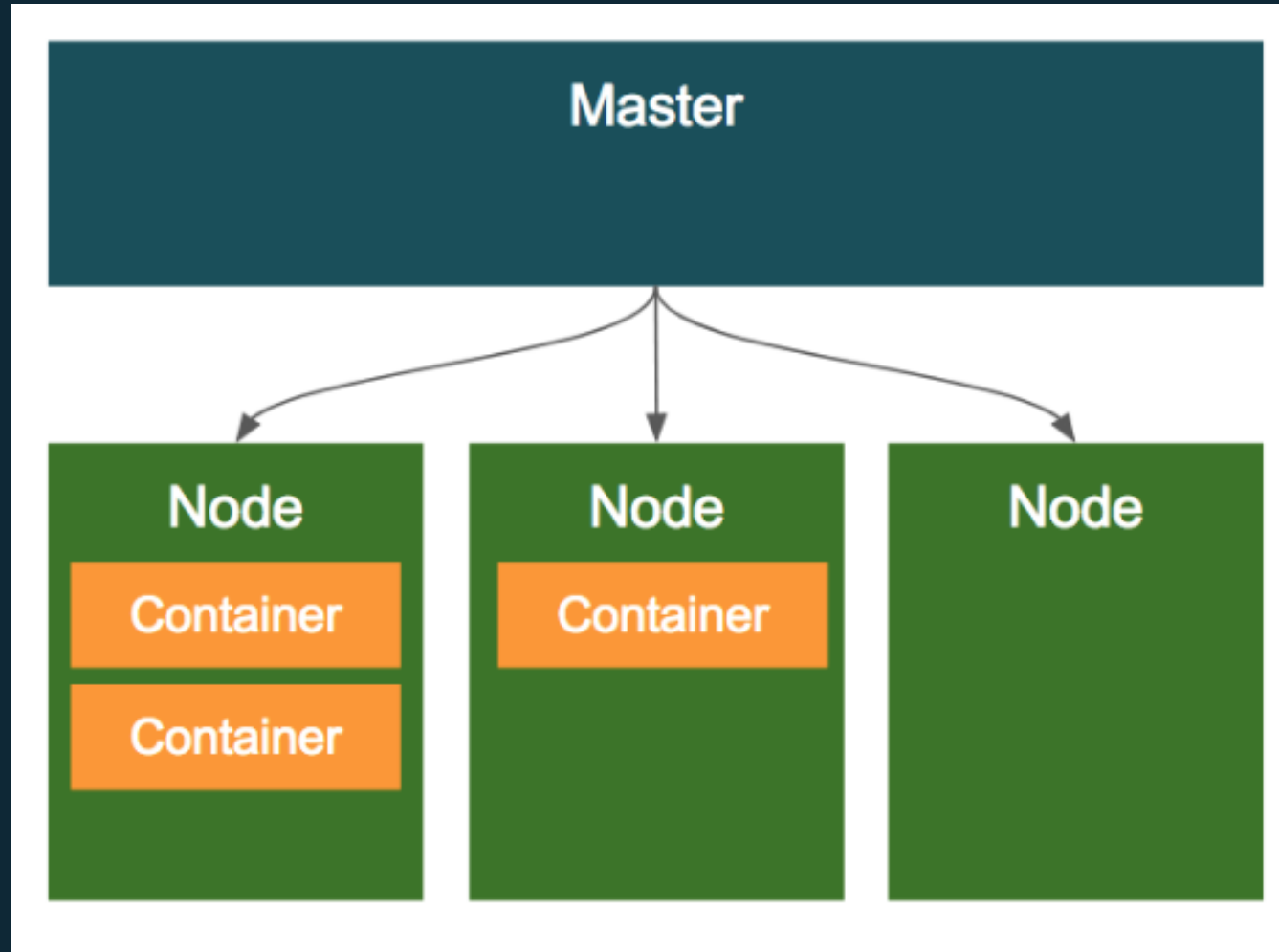
**Familiar Open Source CLIs**  
docker login, pull, push



# 참고 자료

# 마스터와 노드

클러스터 전체를 관리하는 컨트롤러로써 마스터가 존재하고, 컨테이너가 배포되는 머신 (가상머신이나 물리적인 서버머신)인 노드가 존재



# 오브젝트

쿠버네티스를 이해하기 위해서 가장 중요한 부분이 오브젝트이다. 가장 기본적인 구성단위가 되는 기본 오브젝트(Basic object)와, 이 기본 오브젝트(Basic object)를 생성하고 관리하는 추가적인 기능을 가진 컨트롤러(Controller)로 이루어진다. 그리고 이러한 오브젝트의 스펙(설정)이외에 추가정보인 메타 정보들로 구성이 된다고 보면 된다.

## 오브젝트 스펙 (Object Spec)

오브젝트들은 모두 오브젝트의 특성 (설정정보)을 기술한 오브젝트 스펙 (Object Spec)으로 정의가 되고, 커맨드 라인을 통해서 오브젝트 생성시 인자로 전달하여 정의를 하거나 또는 yaml이나 json 파일로 스펙을 정의할 수 있다.

## 기본 오브젝트 (Basic Object)

쿠버네티스에 의해서 배포 및 관리되는 가장 기본적인 오브젝트는 컨테이너화되어 배포되는 애플리케이션의 워크로드를 기술하는 오브젝트로 Pod,Service,Volume,Namespace 4가지가 있다.

간단하게 설명 하자면 Pod는 컨테이너화된 애플리케이션, Volume은 디스크, Service는 로드밸런서 그리고 Namespace는 패키지명 정도로 생각하면 된다. 그러면 각각을 자세하게 살펴보도록 하자.

# Pod

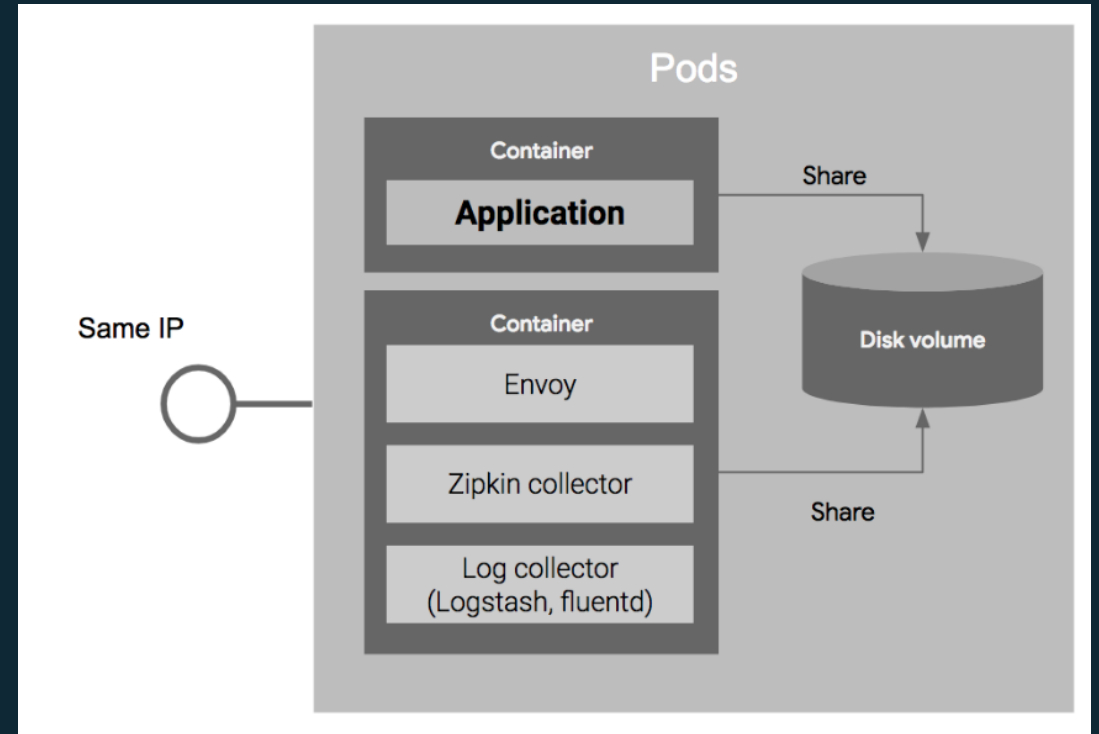
Pod 는 쿠버네티스에서 가장 기본적인 배포 단위로, 컨테이너를 포함하는 단위이다.

쿠버네티스의 특징중의 하나는 컨테이너를 개별적으로 하나씩 배포하는 것이 아니라 Pod 라는 단위로 배포하는데, Pod는 하나 이상의 컨테이너를 포함한다.

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.7.9
    ports:
    - containerPort: 8090
```

- apiVersion은 이 스크립트를 실행하기 위한 쿠버네티스 API 버전이다 보통 v1을 사용한다.
- kind 에는 리소스의 종류를 정의하는데, Pod를 정의하려고 하기 때문에, Pod라고 넣는다.
- metadata에는 이 리소스의 각종 메타 데이터를 넣는데, 라벨(뒤에서 설명할)이나 리소스의 이름등 각종 메타데이터를 넣는다
- spec 부분에 리소스에 대한 상세한 스펙을 정의한다.
- Pod는 컨테이너를 가지고 있기 때문에, container 를 정의한다. 이름은 nginx로 하고 도커 이미지 nginx:1.7.9 를 사용하고, 컨테이너 포트 8090을 오픈한다.

- Pod 내의 컨테이너는 IP와 Port를 공유한다.  
두 개의 컨테이너가 하나의 Pod를 통해서 배포되었을때, localhost를 통해서 통신이 가능하다.  
예를 들어 컨테이너 A가 8080, 컨테이너 B가 7001로 배포가 되었을 때, B에서 A를 호출할때는 localhost:8080 으로 호출하면 되고, 반대로 A에서 B를 호출할때엔 localhost:7001로 호출하면 된다.
- Pod 내에 배포된 컨테이너간에는 디스크 볼륨을 공유 가능
- 근래 애플리케이션들은 실행할때 애플리케이션만 올라가는것이 아니라 Reverse proxy, 로그 수집기등 다양한 주변 솔루션이 같이 배포 되는 경우가 많고, 특히 로그 수집기의 경우에는 애플리케이션 로그 파일을 읽어서 수집한다. 애플리케이션 (Tomcat, node.js)와 로그 수집기를 다른 컨테이너로 배포할 경우, 일반적인 경우에는 컨테이너에 의해서 파일 시스템이 분리되기 때문에, 로그 수집기가 애플리케이션이 배포된 컨테이너의 로그파일을 읽는 것이 불가능하지만, 쿠버네티스의 경우 하나의 Pod 내에서는 컨테이너들끼리 볼륨을 공유할 수 있기 때문에 다른 컨테이너의 파일을 읽어올 수 있다.

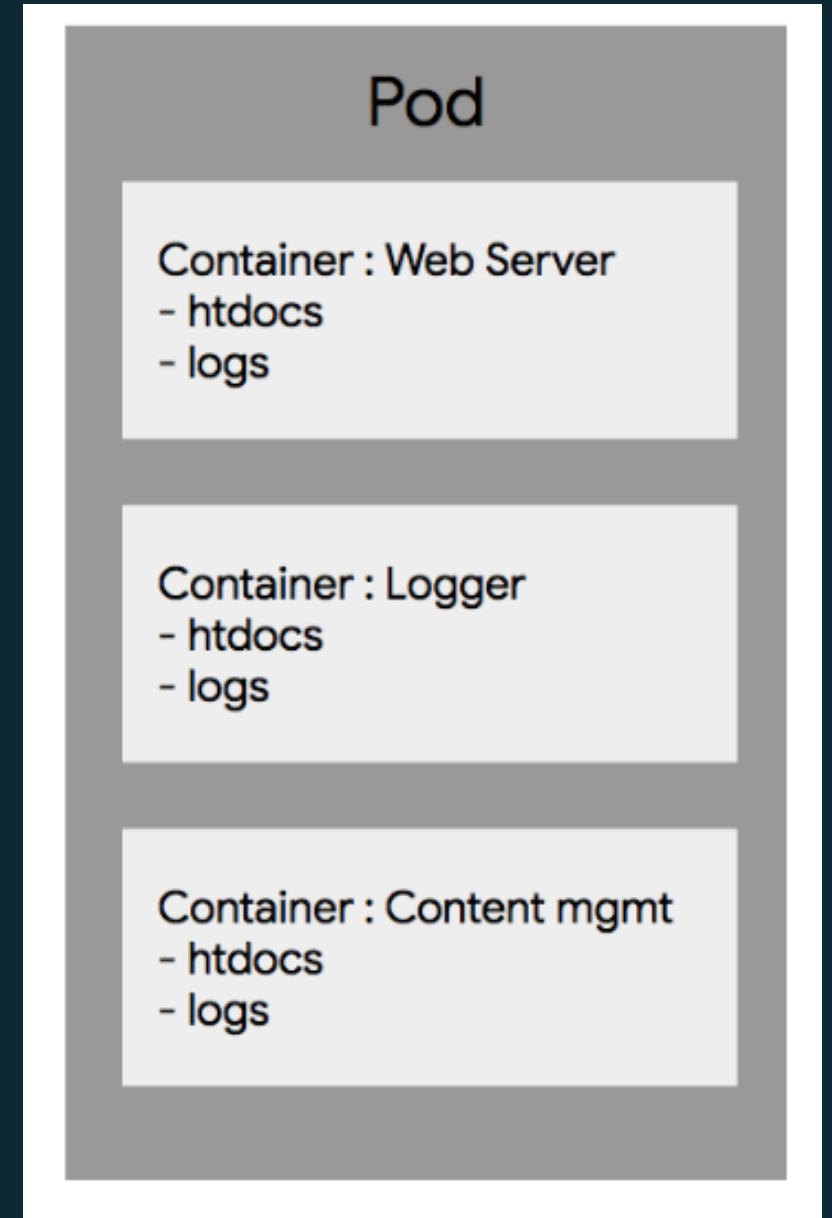


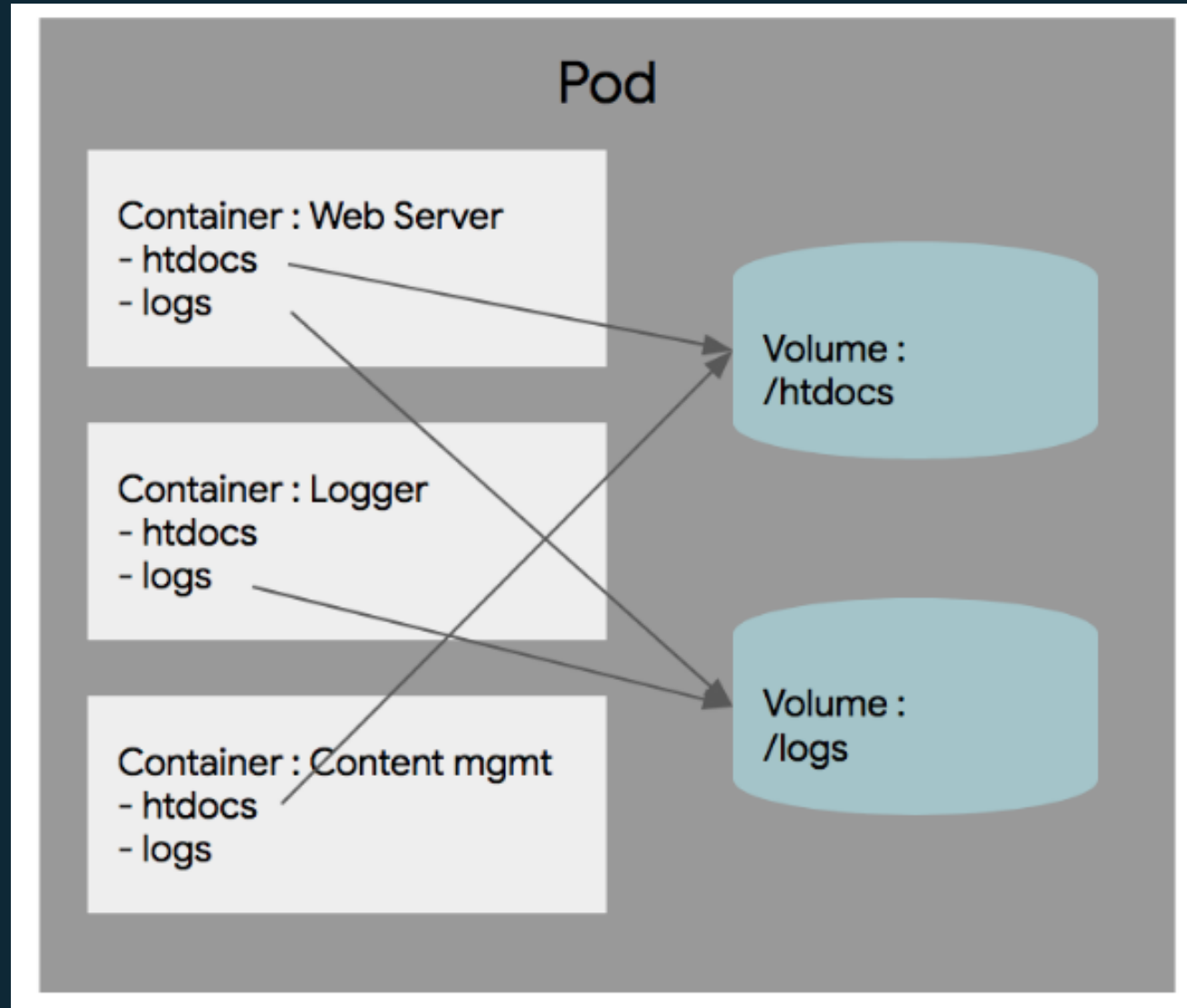


## Volume

Pod가 기동할때 디폴트로, 컨테이너마다 로컬 디스크를 생성해서 기동되는데, 이 로컬 디스크의 경우에는 영구적이지 못하다. 즉 컨테이너가 리스타트 되거나 새로 배포될때 마다 로컬 디스크는 Pod 설정에 따라서 새롭게 정의되서 배포되기 때문에, 디스크에 기록된 내용이 유실된다.

데이터 베이스와 같이 영구적으로 파일을 저장해야 하는 경우에는 컨테이너 리스타트에 상관 없이 파일을 영속적으로 저장해야 하는데, 이러한 형태의 스토리지를 볼륨이라고 한다. 볼륨은 컨테이너의 외장 디스크로 생각하면 된다. Pod가 기동할때 컨테이너에 마운트해서 사용한다.





htdocs와 logs 볼륨을 각각 생성한 후에, htdocs는 WebServer와, Contents management 컨테이너에 마운트 해서 공유하고, logs볼륨은 Logger와 WebServer 컨테이너에서 공유하도록 하면된다.

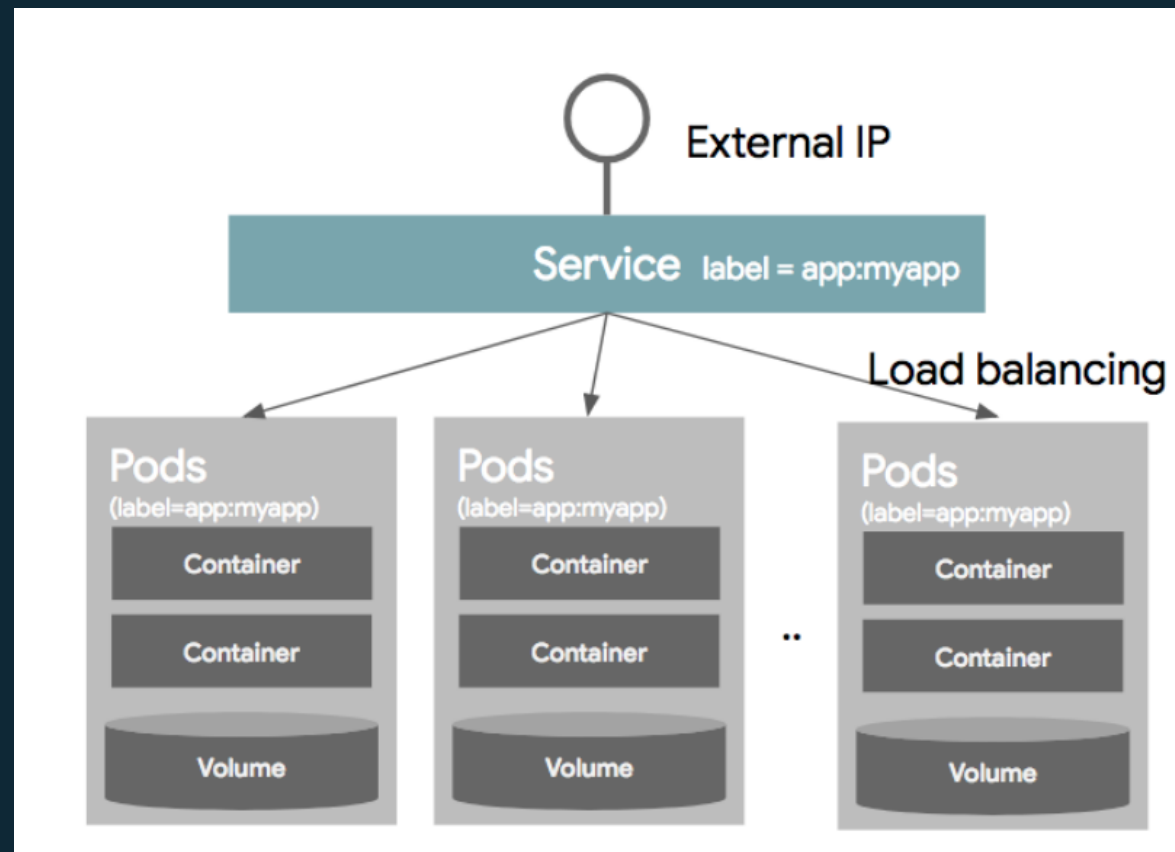
쿠버네티스는 다양한 외장 디스크를 추상화된 형태로 제공한다. iSCSI나 NFS와 같은 온프레임 기반의 일반적인 외장 스토리지 이외에도, 클라우드의 외장 스토리지인 AWS EBS, Google PD,에서 부터 github, glusterfs와 같은 다양한 오픈소스 기반의 외장 스토리지나 스토리지 서비스를 지원하여, 스토리지 아키텍처 설계에 다양한 옵션을 제공

## Service

Pod와 볼륨을 이용하여, 컨테이너들을 정의한 후에, Pod 를 서비스로 제공할때, 일반적인 분산환경에서는 하나의 Pod로 서비스 하는 경우는 드물고, 여러개의 Pod를 서비스하면서, 이를 로드밸런서를 이용해서 하나의 IP와 포트로 묶어서 서비스를 제공한다.

Pod의 경우에는 동적으로 생성이 되고, 장애가 생기면 자동으로 리스타트 되면서 그 IP가 바뀌기 때문에, 로드밸런서에서 Pod의 목록을 지정할 때는 IP주소를 이용하는 것은 어렵다. 또한 오토 스케일링으로 인하여 Pod 가 동적으로 추가 또는 삭제되기 때문에, 이렇게 추가/삭제된 Pod 목록을 로드밸런서가 유연하게 선택해 줘야 한다.  
그래서 사용하는 것이 라벨(label)과 라벨 셀렉터(label selector) 라는 개념이다.

서비스를 정의할때, 어떤 Pod를 서비스로 묶을 것인지를 정의하는데, 이를 라벨 셀렉터라고 한다. 각 Pod를 생성할때 메타데이터 정보 부분에 라벨을 정의할 수 있다. 서비스는 라벨 셀렉터에서 특정 라벨을 가지고 있는 Pod만 선택하여 서비스에 묶게 된다. 아래 그림은 서비스가 라벨이 "myapp"인 서비스만 골라내서 서비스에 넣고, 그 Pod간에만 로드밸런싱을 통하여 외부로 서비스를 제공하는 형태이다.



```
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
  selector:
    app: myapp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

리소스 종류가 Service 이기 때문에, kind는 Service로 지정하고, 스크립트를 실행할 api 버전은 v1으로 apiVersion에 정의했다.

메타데이터에 서비스의 이름을 my-service로 지정하고 spec 부분에 서비스에 대한 스펙을 정의한다.

selector에서 라벨이 app:myapp인 Pod 만을 선택해서 서비스에서 서비스를 제공하게 하고

포트는 TCP를 이용하되, 서비스는 80 포트로 서비스를 하되, 서비스의 80 포트의 요청을 컨테이너의 9376 포트로 연결해서 서비스를 제공한다.

# Name space

네임스페이스는 한 쿠버네티스 클러스터내의 논리적인 분리단위라고 보면 된다.

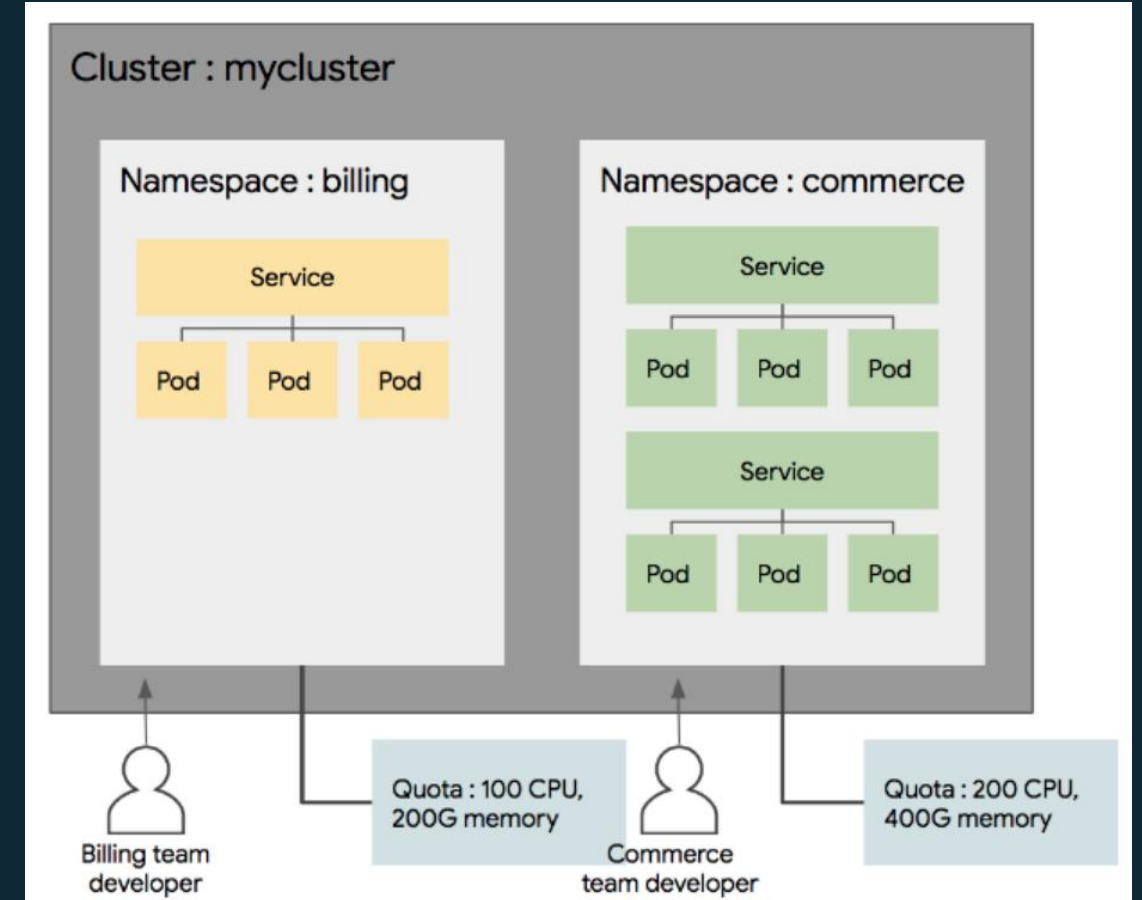
Pod, Service 등은 네임 스페이스 별로 생성이나 관리가 될 수 있고, 사용자의 권한 역시 이 네임 스페이스 별로 나눠서 부여할 수 있다. 즉 하나의 클러스터 내에, 개발/운영/테스트 환경이 있을때, 클러스터를 개발/운영/테스트 3개의 네임 스페이스로 나눠서 운영할 수 있다.

네임스페이스로 할 수 있는 것은

- 사용자별로 네임스페이스별 접근 권한을 다르게 운영할 수 있다.
- 네임스페이스별로 리소스의 쿼타 (할당량)을 지정할 수 있다. 개발계에는 CPU 100, 운영계에는 CPU 400과 GPU 100개 식으로, 사용 가능한 리소스의 수를 지정할 수 있다.
- 네임 스페이스별로 리소스를 나눠서 관리할 수 있다. (Pod, Service 등)

주의할점은 네임 스페이스는 논리적인 분리 단위이지 물리적이거나 기타 장치를 통해서 환경을 분리(Isolation)한것이 아니다. 다른 네임 스페이스간의 pod 라도 통신은 가능하다.

물론 네트워크 정책을 이용하여, 네임 스페이스간의 통신을 막을 수 있지만 높은 수준의 분리 정책을 원하는 경우에는 쿠버네티스 클러스터 자체를 분리하는 것을 권장한다.



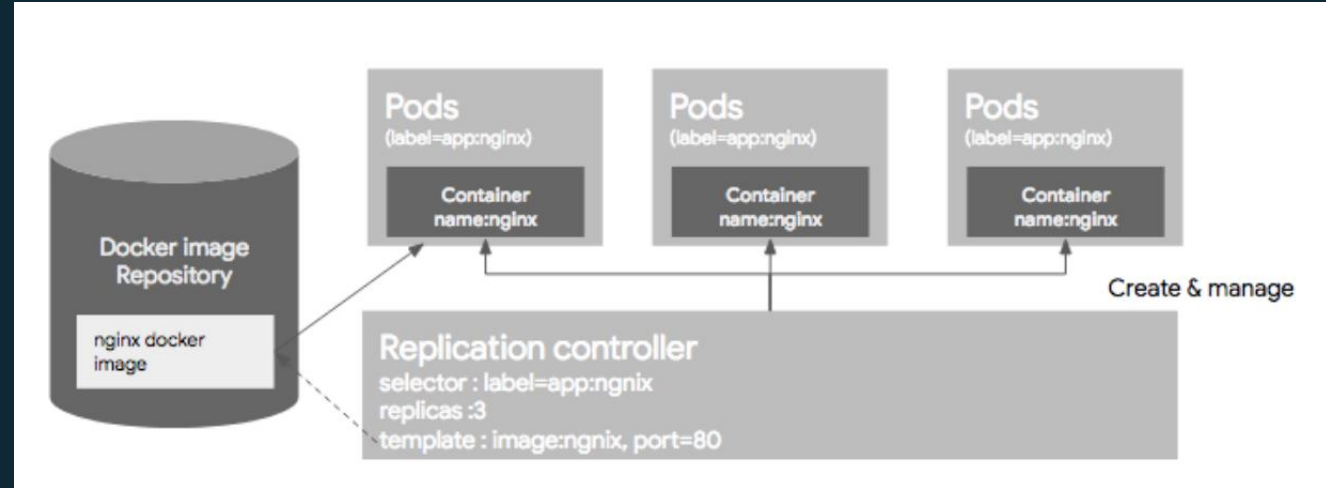
# 컨트롤러

애플리케이션을 조금 더 편리하게 관리하기 위해서 쿠버네티스는 컨트롤러라는 개념을 사용한다. 컨트롤러는 기본 오브젝트들을 생성하고 이를 관리하는 역할을 해준다. 컨트롤러는 Replication Controller (aka RC), Replication Set, DaemonSet, Job, StatefulSet, Deployment 등이 있다. 각자의 개념에 대해서 살펴보도록 하자.

## Replication Controller

Replication Controller는 Pod를 관리해주는 역할을 하는데, 지정된 숫자로 Pod를 기동 시키고, 관리하는 역할을 한다. Replication Controller (이하 RC)는 크게 3가지 파트로 구성되는데, Replica의 수, Pod Selector, Pod Template 3가지로 구성된다.

- Selector : 먼저 Pod selector는 라벨을 기반으로 하여, RC가 관리한 Pod를 가지고 오는데 사용한다.
- Replica 수 : RC에 의해서 관리되는 Pod의 수인데, 그 숫자만큼 Pod의 수를 유지하도록 한다. 예를 들어 replica 수가 3이면, 3개의 Pod만 띄우도록 하고, 이보다 Pod가 모자르면 새로운 Pod를 띄우고, 이보다 숫자가 많으면 남은 Pod를 삭제한다.
- Pod를 추가로 기동할 때 그러면 어떻게 Pod를 만들지 Pod에 대한 정보 (도커 이미지, 포트, 라벨 등)에 대한 정보가 필요한데, 이는 Pod template이라는 부분에 정의 한다.



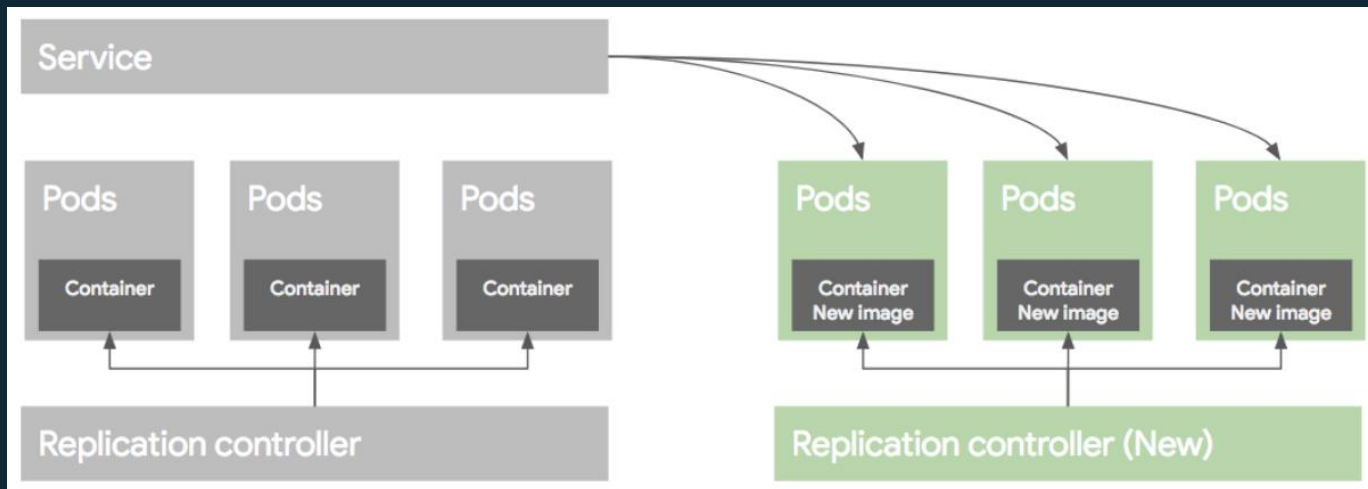
# ReplicaSet

ReplicaSet은 Replication Controller 의 새버전으로 생각하면 된다.

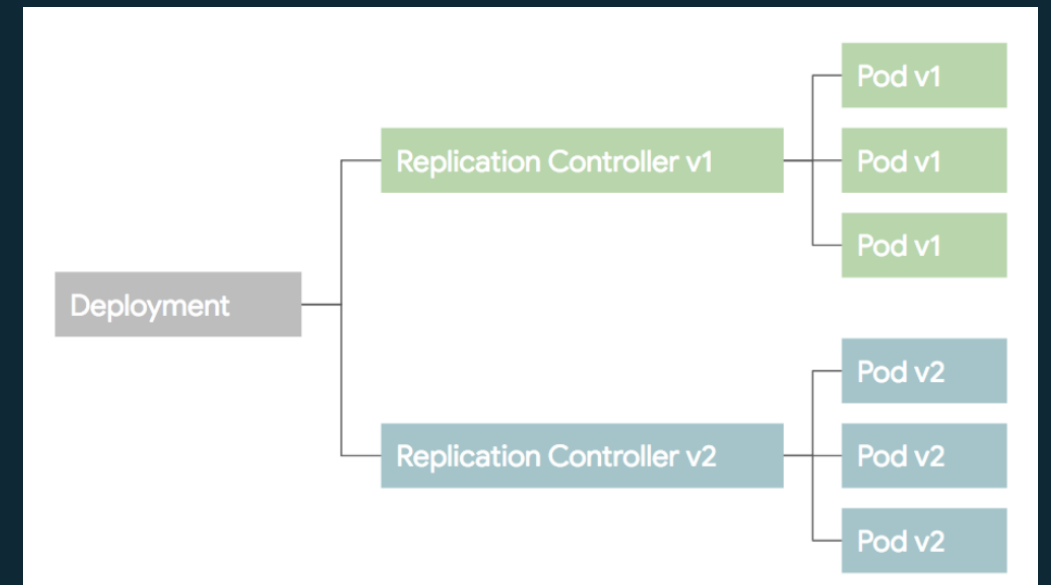
큰 차이는 없고 Replication Controller 는 Equality 기반 Selector를 이용하는데 반해, Replica Set은 Set 기반의 Selector를 이용한다.

# Deployment

Deployment (이하 디플로이먼트) Replication controller와 Replica Set의 좀더 상위 추상화 개념이다. 실제 운영에서는 ReplicaSet 이나 Replication Controller를 바로 사용하는 것보다, 좀더 추상화된 Deployment를 사용하게 된다.

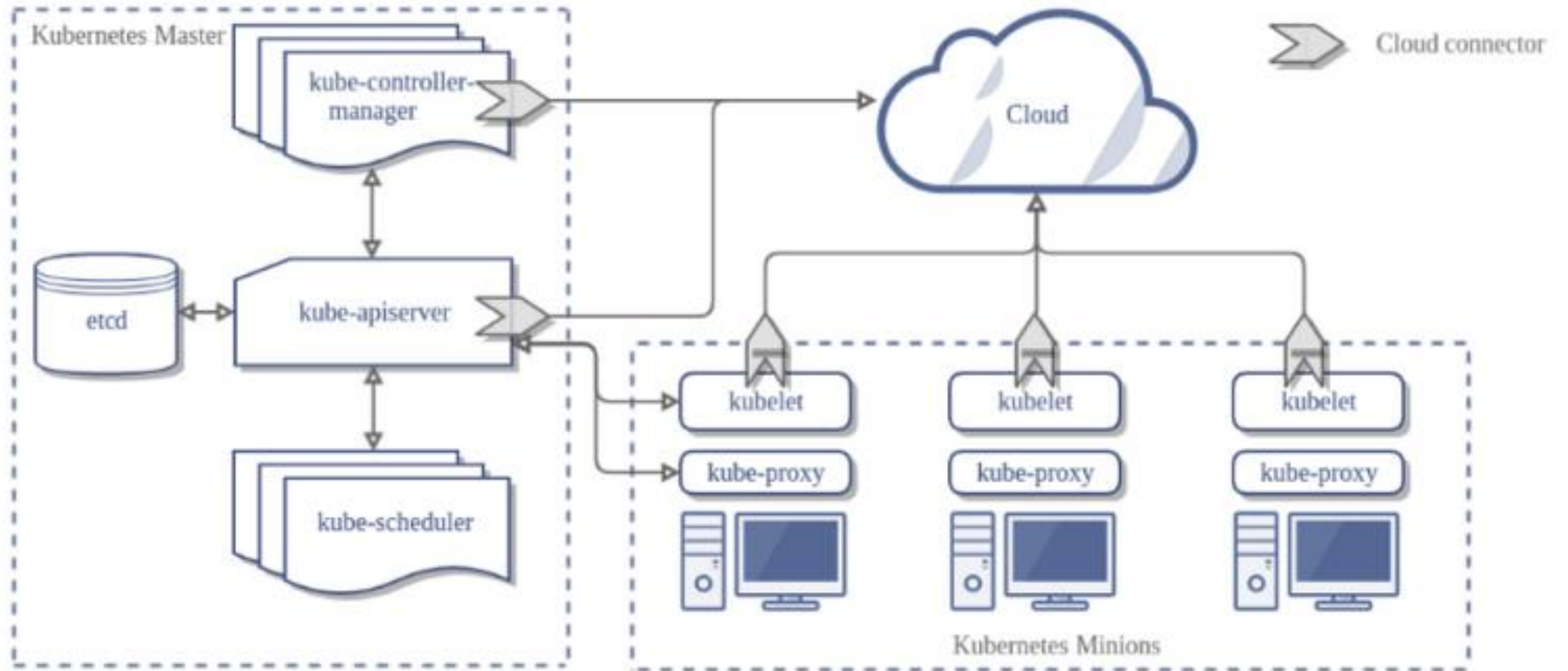


블루/그린 배포





# 쿠버네티스 구조



# 마스터와 노드

쿠버네티스는 크게 마스터(Master)와 노드(Node) 두 개의 컴포넌트로 분리된다. 마스터는 쿠버네티스의 설정 환경을 저장하고 전체 클러스터를 관리하는 역할을 맡고있고, 노드는 파드나 컨테이너 처럼 쿠버네티스 위에서 동작하는 워크로드를 호스팅하는 역할을 한다.

## 마스터

쿠버네티스 클러스터 전체를 컨트롤 하는 시스템으로, 크게 다음과 API 서버, 스케줄러, 컨트롤러 매니저, etcd 로 구성되어 있다.

### API 서버

쿠버네티스는 모든 명령과 통신을 API를 통해서 하는데, 그 중심이 되는 서버가 API서버이다. 쿠버네티스의 모든 기능들을 REST API로 제공하고 그에 대한 명령을 처리한다.

### Etcd

API 서버가 명령을 주고 받는 서버라면, 쿠버네티스 클러스터의 데이터 베이스 역할이 되는 서버로 설정값이나 클러스터의 상태를 저장하는 서버이다. etcd라는 분산형 키/밸류 스토어 오픈소스 (<https://github.com/coreos/etcd>) 로 쿠버네티스 클러스터의 상태나 설정 정보를 저장한다.

## 컨트롤러 매니저

컨트롤러 매니저는 컨트롤러(Replica controller, Service controller, Volume Controller, Node controller 등)를 생성하고 이를 각 노드에 배포하며 이를 관리하는 역할을 한다.

## 스케줄러

스케줄러는 Pod, 서비스 등 각 리소스들을 적절한 노드에 할당하는 역할을 한다.

### 컨트롤러 매니저

컨트롤러 매니저는 컨트롤러(Replica controller, Service controller, Volume Controller, Node controller 등)를 생성하고 이를 각 노드에 배포하며 이를 관리하는 역할을 한다.

## DNS

그림에는 빠져있는데, 쿠버네티스는 리소스의 엔드포인트(Endpoint)를 DNS로 맵핑하고 관리한다. Pod나 서비스 등은 IP를 배정받는데, 동적으로 생성되는 리소스이기 때문에 그 IP 주소가 그때마다 변경이 되기 때문에, 그 리소스에 대한 위치 정보가 필요한데, 이러한 패턴을 Service discovery 패턴이라고 하는데, 쿠버네티스에서는 이를 내부 DNS서버를 두는 방식으로 해결하였다.

새로운 리소스가 생기면, 그 리소스에 대한 IP와 DNS 이름을 등록하여, DNS 이름을 기반으로 리소스에 접근할 수 있도록 한다.

# 노드

노드는 마스터에 의해 명령을 받고 실제 워크로드를 생성하여 서비스 하는 컴포넌트이다. 노드에는 Kubelet, Kube-proxy, cAdvisor 그리고 컨테이너 런타임이 배포된다.

## Kubelet

노드에 배포되는 에이전트로, 마스터의 API서버와 통신을 하면서, 노드가 수행해야 할 명령을 받아서 수행하고, 반대로 노드의 상태등을 마스터로 전달하는 역할을 한다.

## Kube-proxy

노드로 들어오거나 네트워크 트래픽을 적절한 컨테이너로 라우팅하고, 로드밸런싱등 노드로 들어오고 나가는 네트워크 트래픽을 프록시하고, 노드와 마스터간의 네트워크 통신을 관리한다.

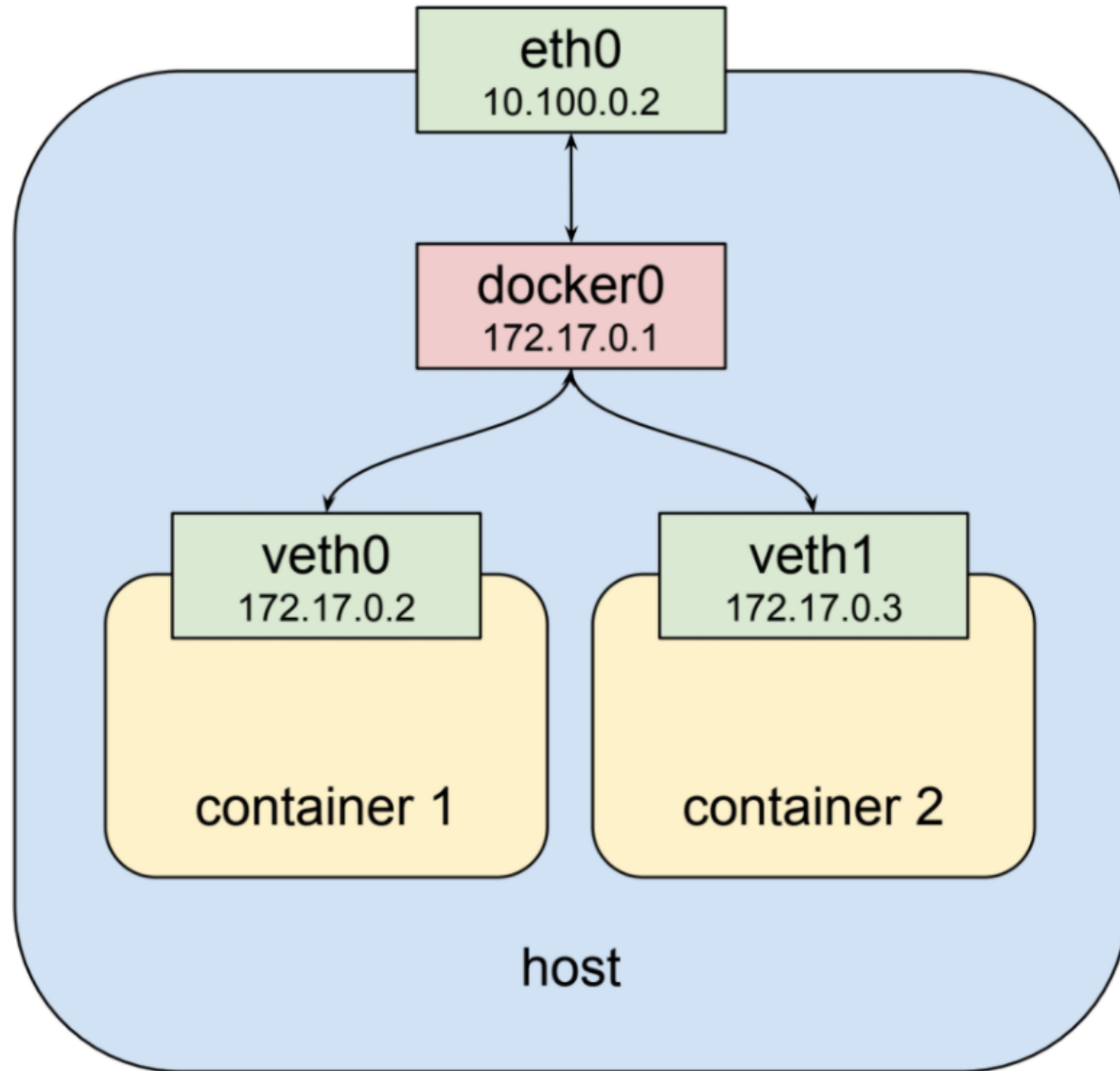
## Container runtime (컨테이너 런타임)

Pod를 통해서 배포된 컨테이너를 실행하는 컨테이너 런타임이다. 컨테이너 런타임은 보통 도커 컨테이너를 생각하기 쉬운데, 도커 이외에도 rkt (보안이 강화된 컨테이너), Hyper container 등 다양한 런타임이 있다.

## cAdvisor

cAdvisor는 각 노드에서 기동되는 모니터링 에이전트로, 노드내에서 가동되는 컨테이너들의 상태와 성능등의 정보를 수집하여, 마스터 서버의 API 서버로 전달한다. 이 데이터들은 주로 모니터링을 위해서 사용된다.

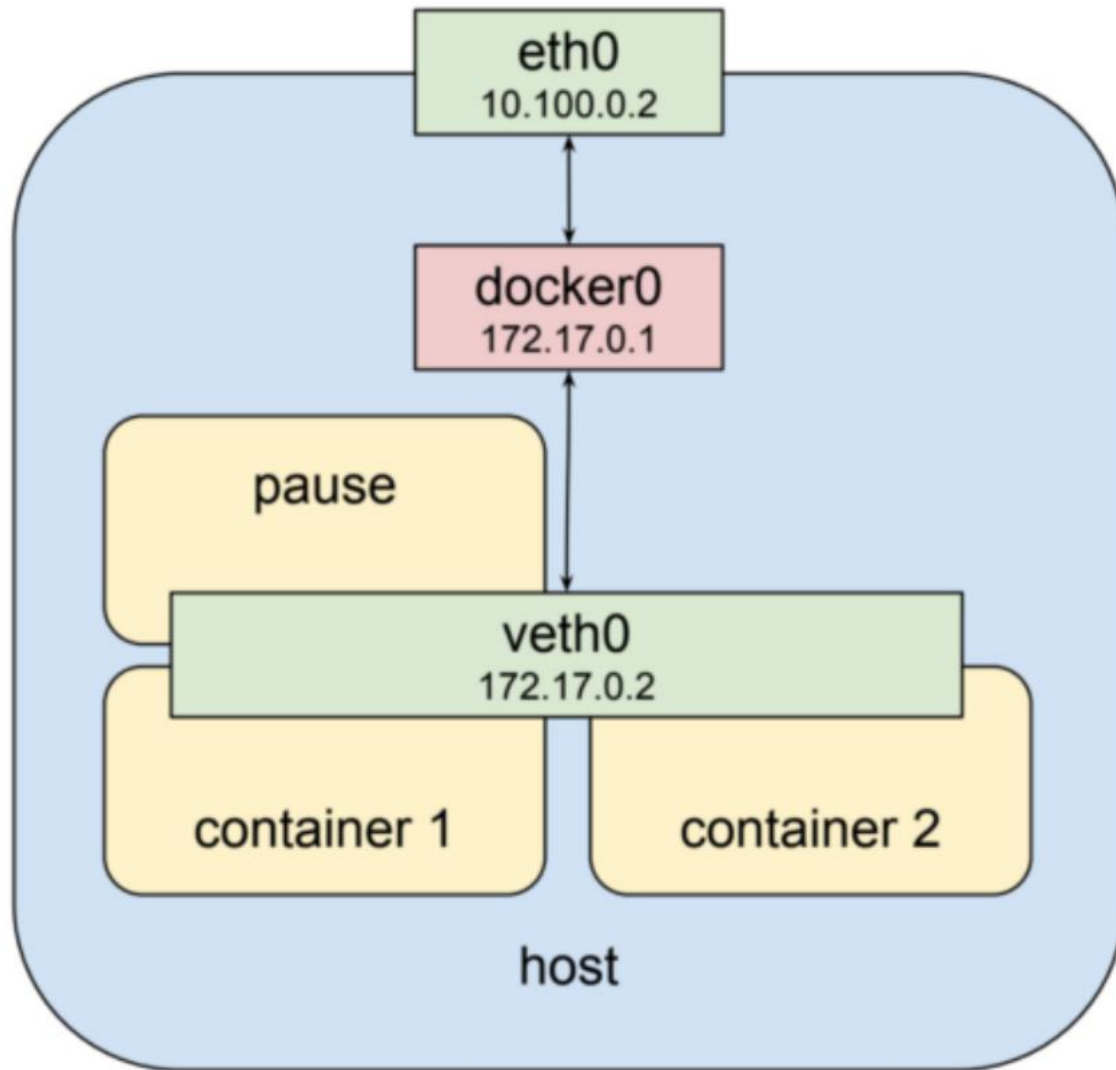
- 서로 결합된 컨테이너와 컨테이너 간 통신
- Pod와 Pod 간의 통신
- Pod와 Service간의 통신
- 외부와 Service간의 통신



Docker에서는 기본적으로 같은 노드(host) 내의 컨테이너 끼리의 통신은 위 그림과 같이 docker0라는 가상 네트워크 인터페이스 (172.17.0.0/24)를 통해 가능

각 컨테이너는 veth라는 가상 네트워크 인터페이스를 고유하게 가지며 따라서 각각의 veth IP 주소 값으로 통신

```
david@user24linux:~$ sudo su -
root@user24linux:~# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 00:22:48:5f:51:87 brd ff:ff:ff:ff:ff:ff
    inet 10.0.1.4/24 brd 10.0.1.255 scope global eth0
        valid_lft forever preferred_lft forever
    inet6 fe80::222:48ff:fe5f:5187/64 scope link
        valid_lft forever preferred_lft forever
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:61:5c:27:11 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
        valid_lft forever preferred_lft forever
    inet6 fe80::42:61ff:fe5c:2711/64 scope link
        valid_lft forever preferred_lft forever
77: veth852ac61@if76: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP group default
    link/ether ae:6a:f7:fb:e8:6a brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::ac6a:f7ff:fefb:e86a/64 scope link
        valid_lft forever preferred_lft forever
```



그림에서는 veth0 가상 네트워크 인터페이스에 두 개의 컨테이너가 동시에 할당

veth0 안에서 각 컨테이너는 고유한 port 번호로 서로를 구분

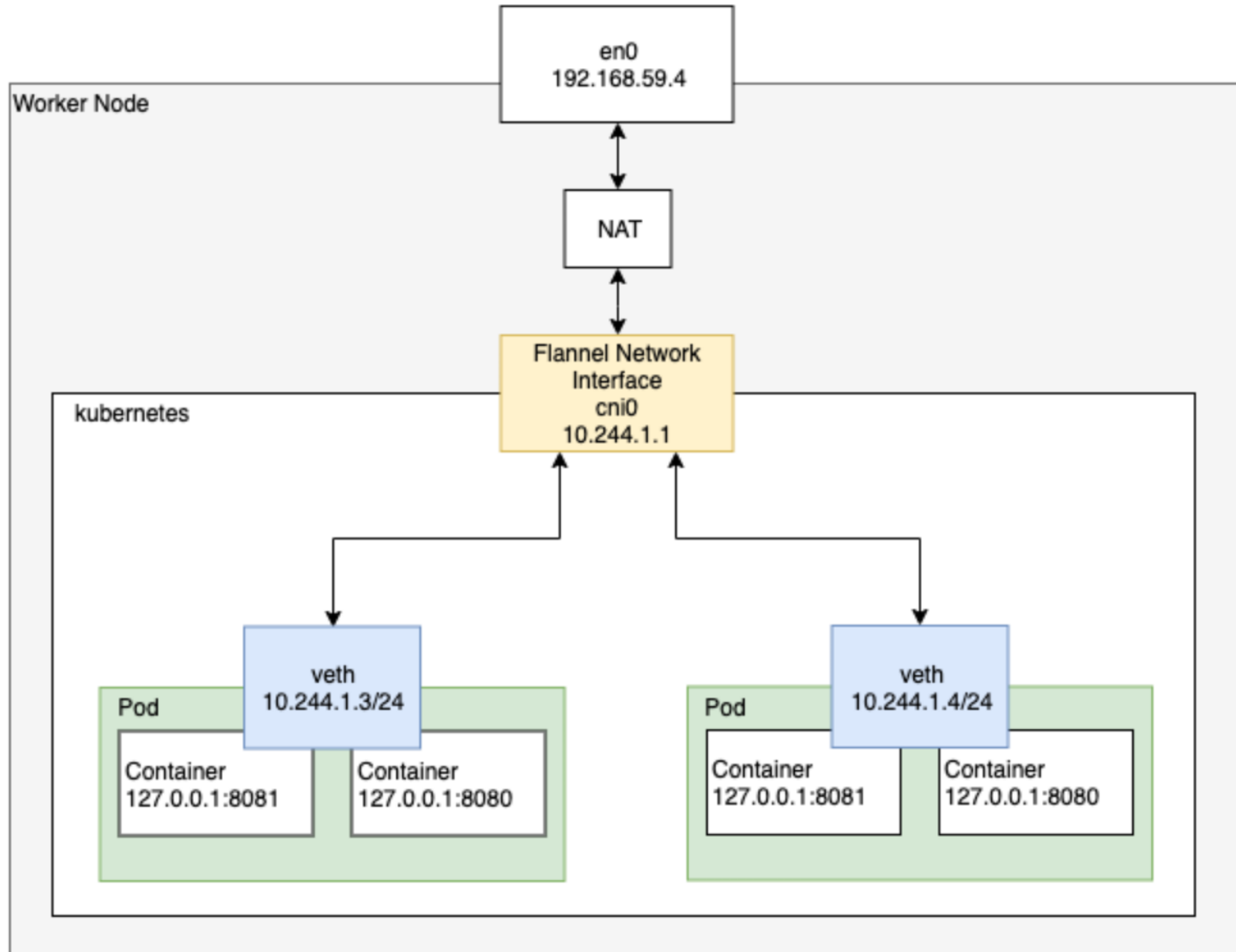
Pod 내에서 컨테이너는 각자 고유한 port 번호를 사용해야함

kubernetes Pod가 실행되고 있는 워커 노드에 들어가서 docker ps 명령어를 입력하면 적어도 한 개 이상의 pause 라는 명령으로 실행된 컨테이너를 볼 수 있음

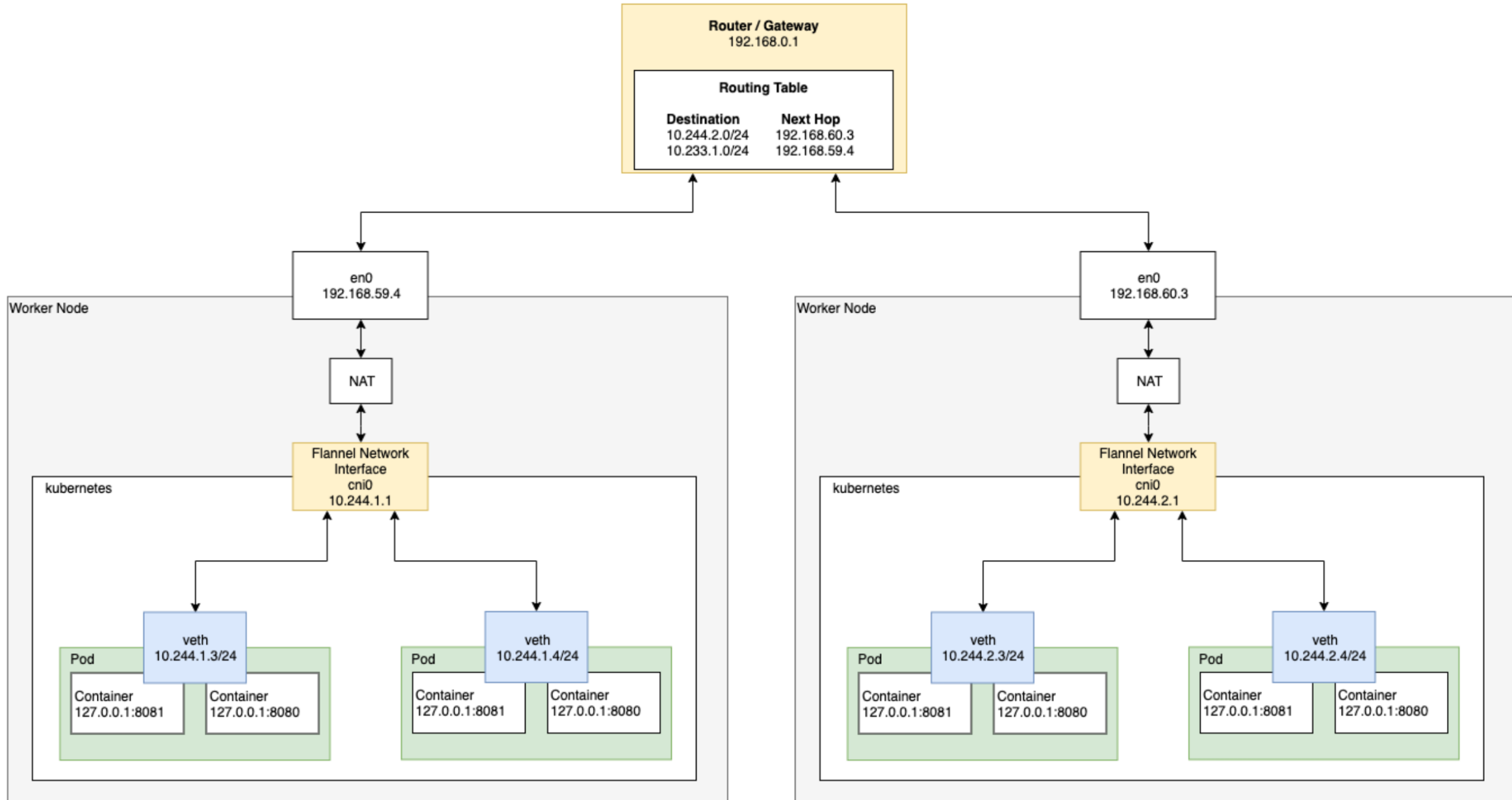
특별한 컨테이너들은 각 Pod마다 존재하며 다른 컨테이너들에게 네트워크 인터페이스를 제공하는 역할만 담당



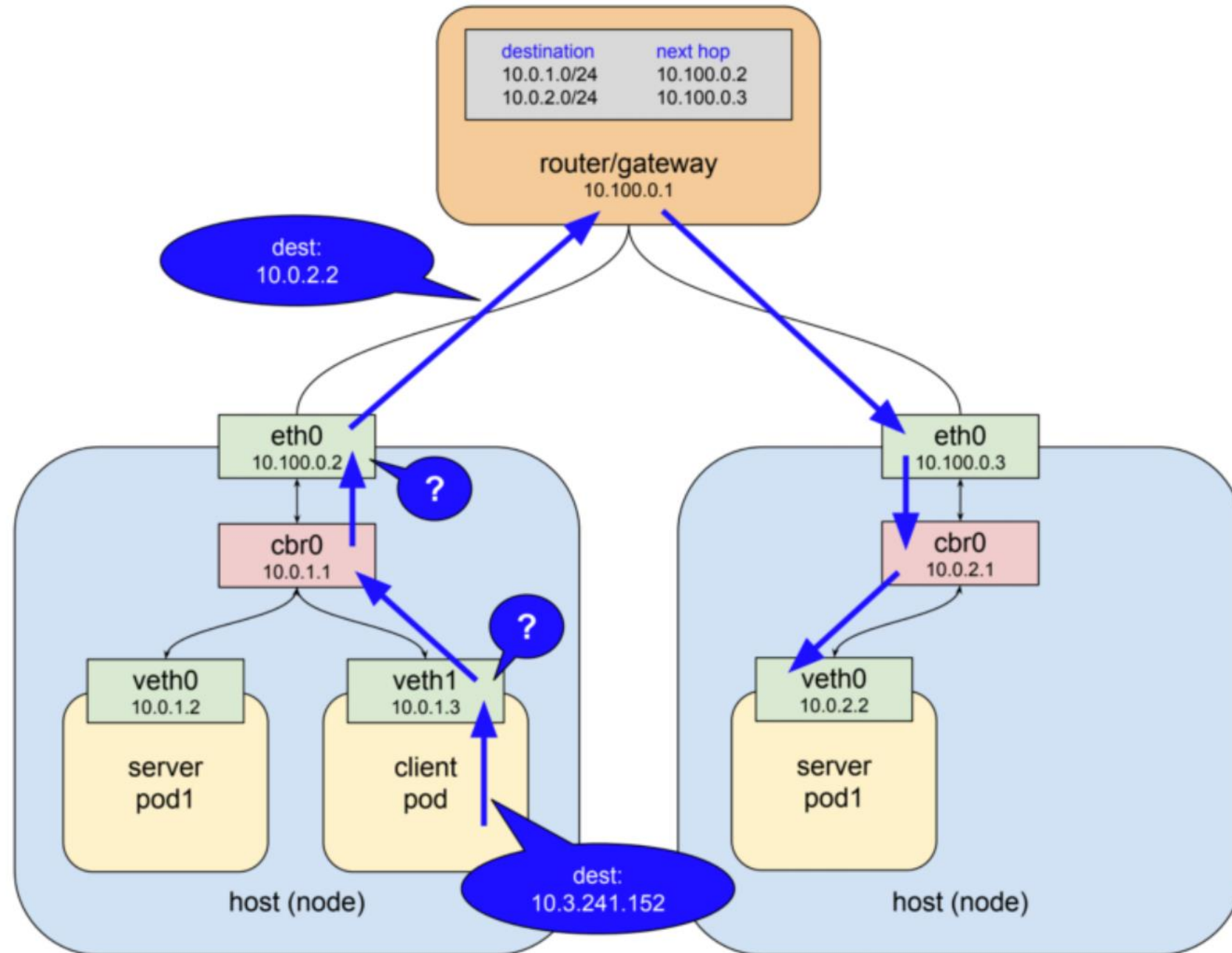
# 싱글 노드 Pod 네트워크



각각 다른 노드에 존재하는 Pod가 서로 통신하려면 라우터를 거쳐서 통신



# Pod와 서비스간 네트워크



# 외부와 서비스간 네트워크 – LB, NodePort

