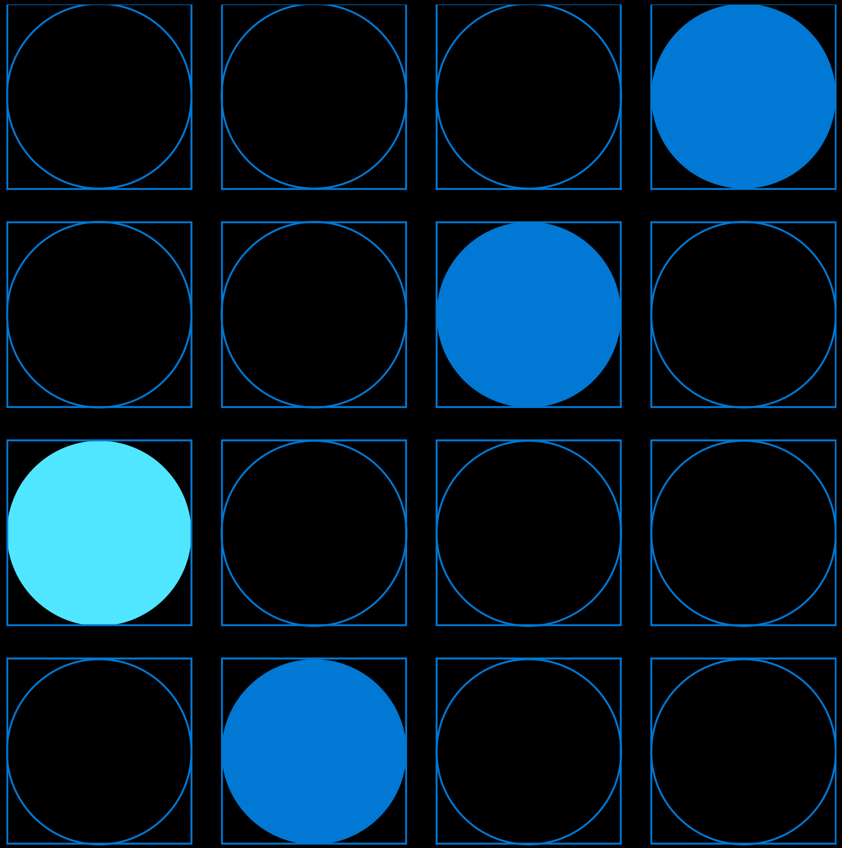
Microsoft Azure

Microsoft Azure DevDays



Microsoft Azure

What are Containers?

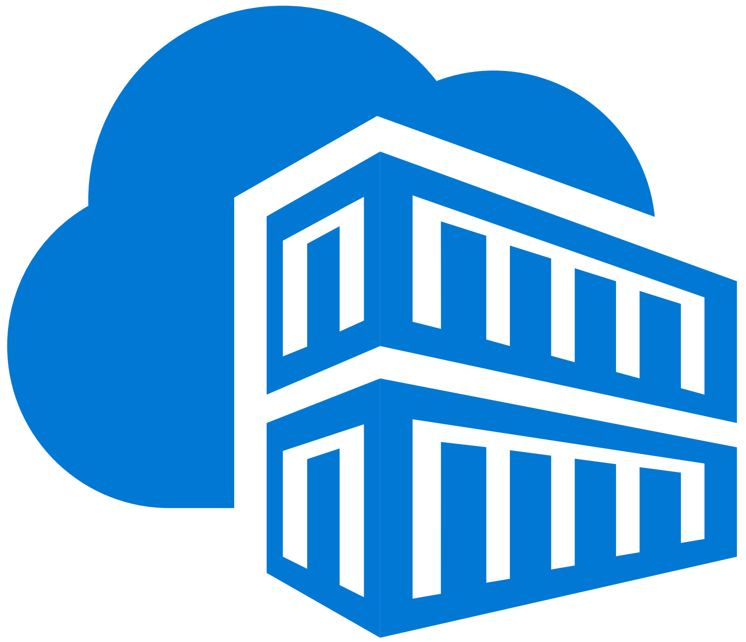
Containers are:

- Containers use resource isolation features of the Linux Kernel, such as **control groups** and **namespaces.**
- All the process run an share the same kernel used by OS.
- Containers are an abstraction at the app layer that package code and dependencies together.
- Containers provide the application and process isolation where one application is completely unaware of the existence of another application.

Wrappers around a UNIX process that directly talks to the kernel to request and use the resources.

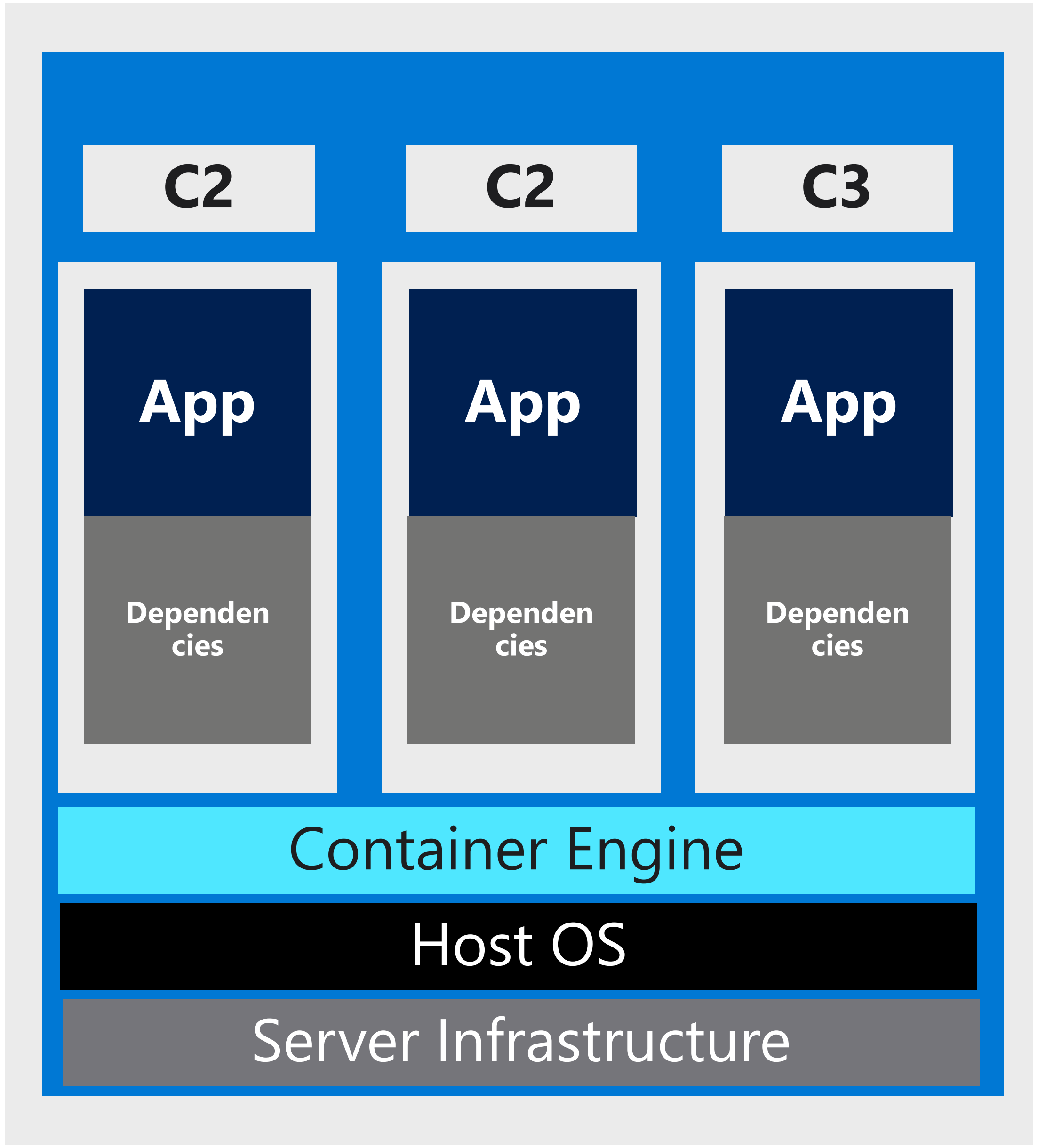
Features of containers

- Efficiency. A virtual machine feels and works like a separate machine. The key advantage is efficient resources usage and isolation from a security standpoint
- Flexibility. Resources can be allocated as needed. CPUs, memory, and the like can all be distributed on an initial requirements basis and when needed.
- Lightweight. Containers leverage and share the host kernel.
- Interchangeable. You can deploy updates and upgrades on-the-fly.
- Portable. You can build locally, deploy to the cloud, and run anywhere

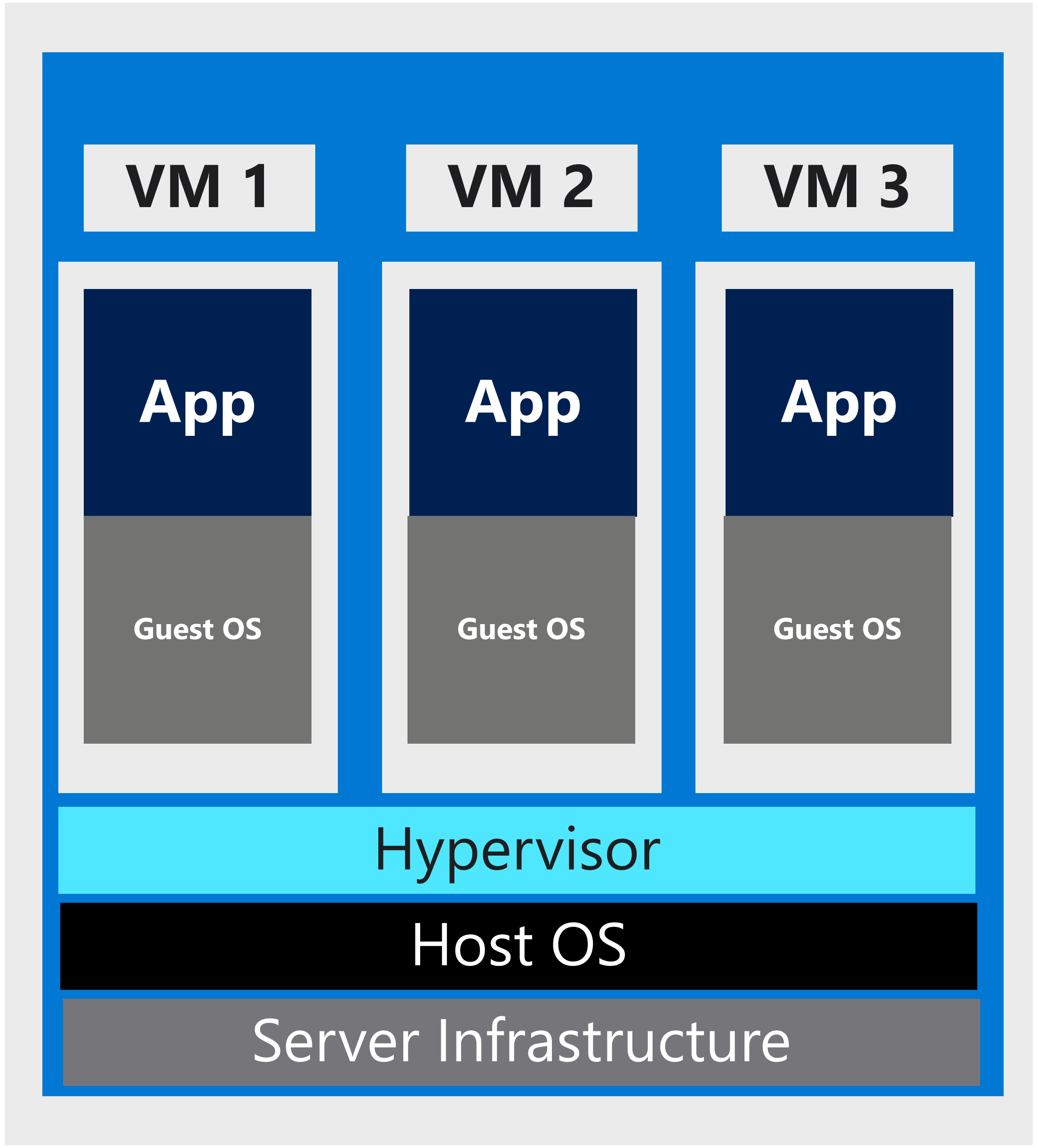


Wrappers around a UNIX process that directly talks to the kernel to request and use the resources.

Container Architecture



Virtual Machine Architecture



Microsoft Azure

Why should you use containers?

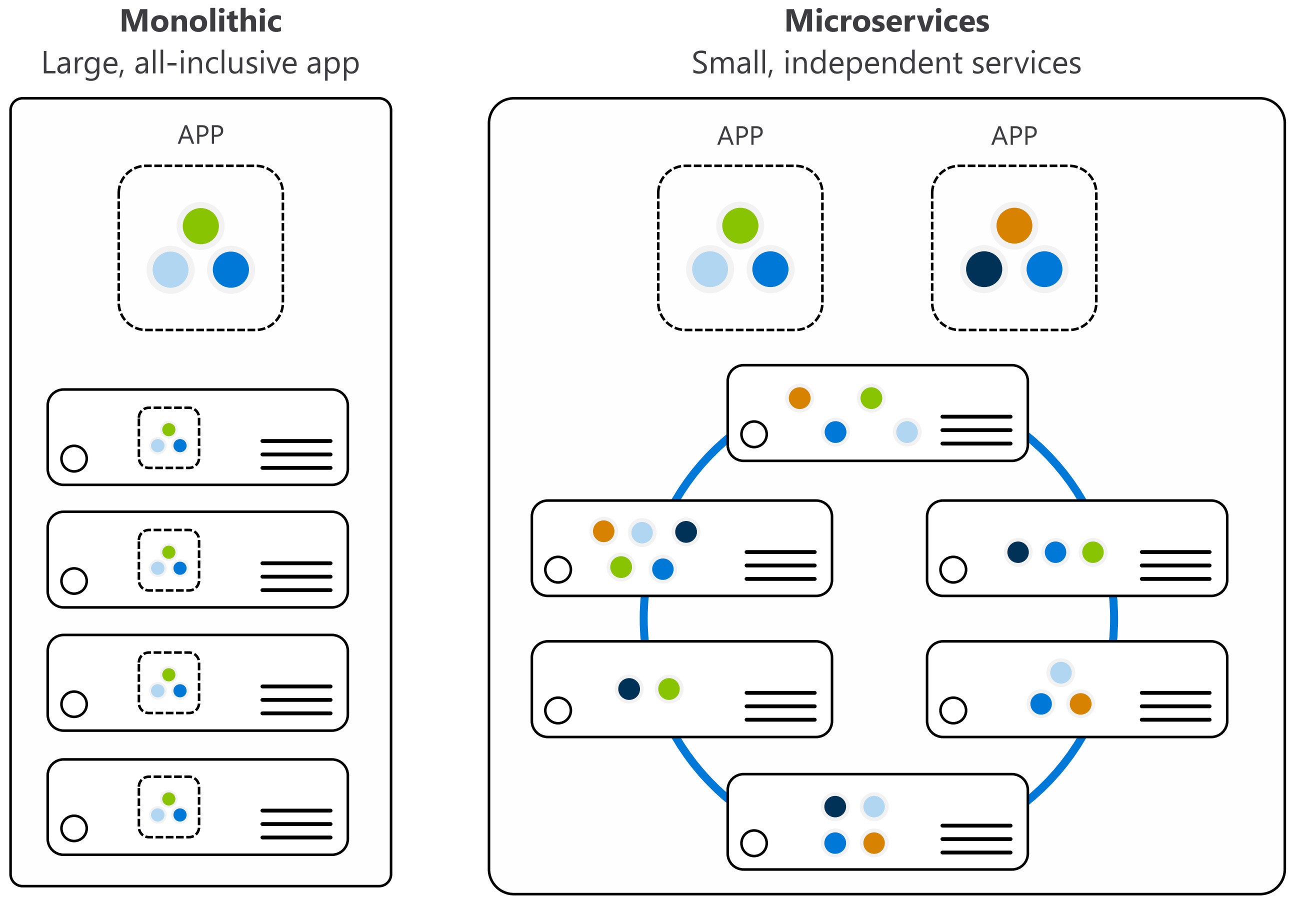
What are microservices?

A Software Architectural Style
Applications are composed of small, independent modules that communicate with each other using well-defined APIs. Not platform specific

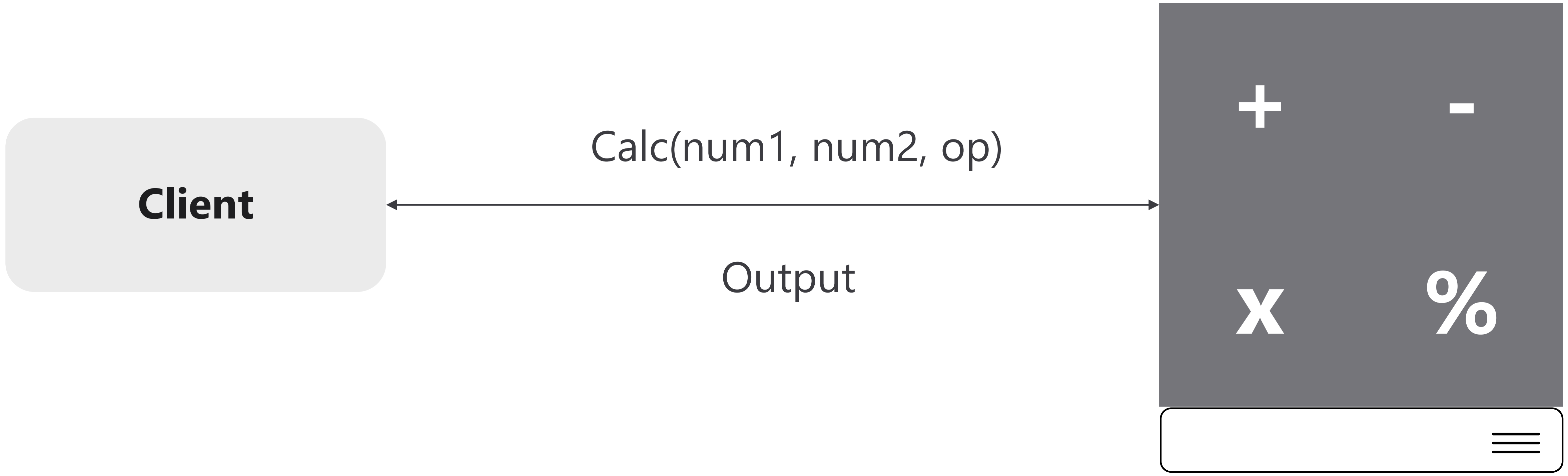
Decoupled
These service modules are highly decoupled building blocks that are small enough to implement a single functionality but together can form larger systems

Independently versioned, deployed, & scaled
With a microservices architecture, developers can create, manage, and improve application services independently, even using different languages

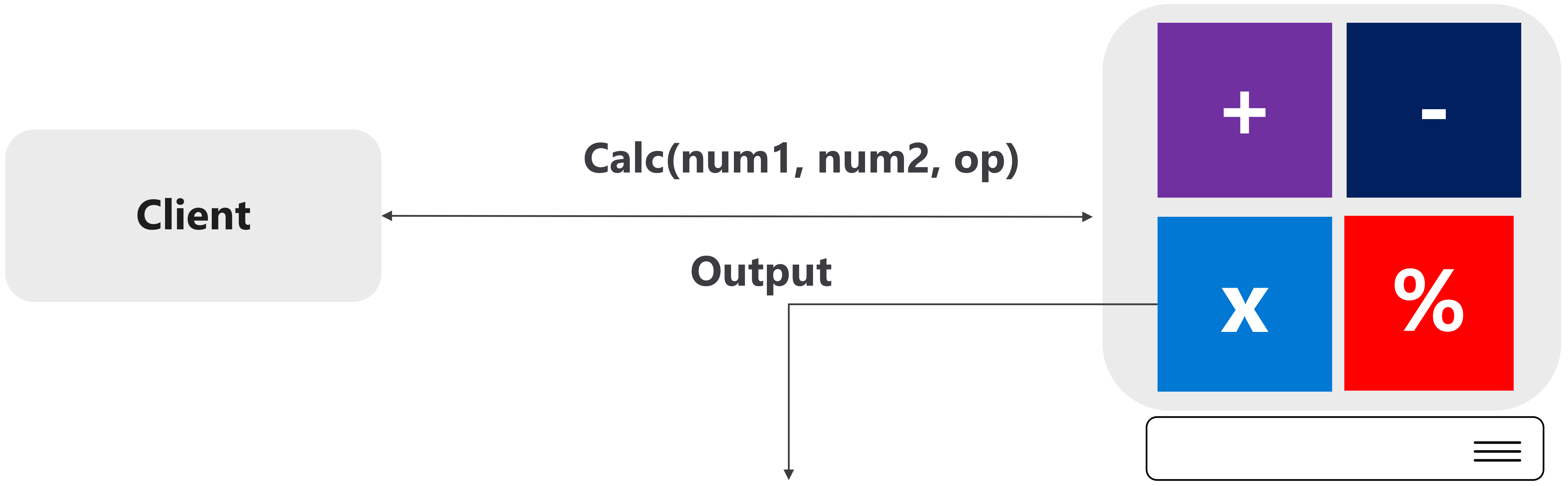
Containers provide the consistent format and isolation desired by microservices



Monolithic Architecture

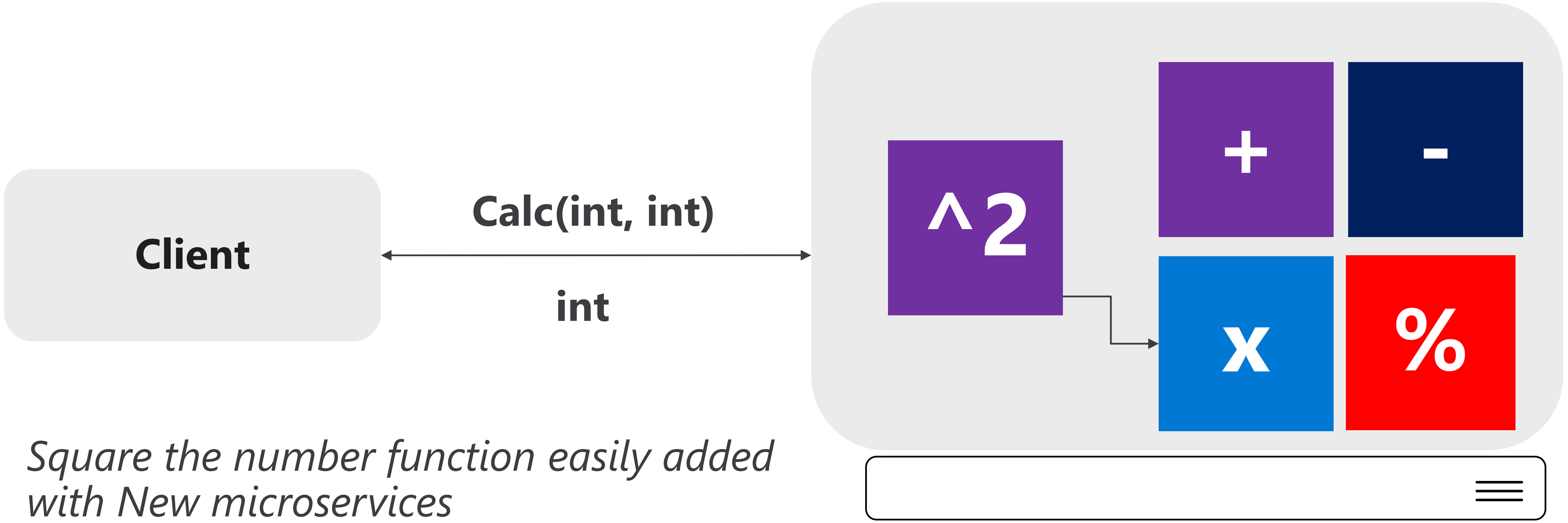


Microservices-based architecture



We have only one microservice to write, compile, and deploy

Microservices-based architecture

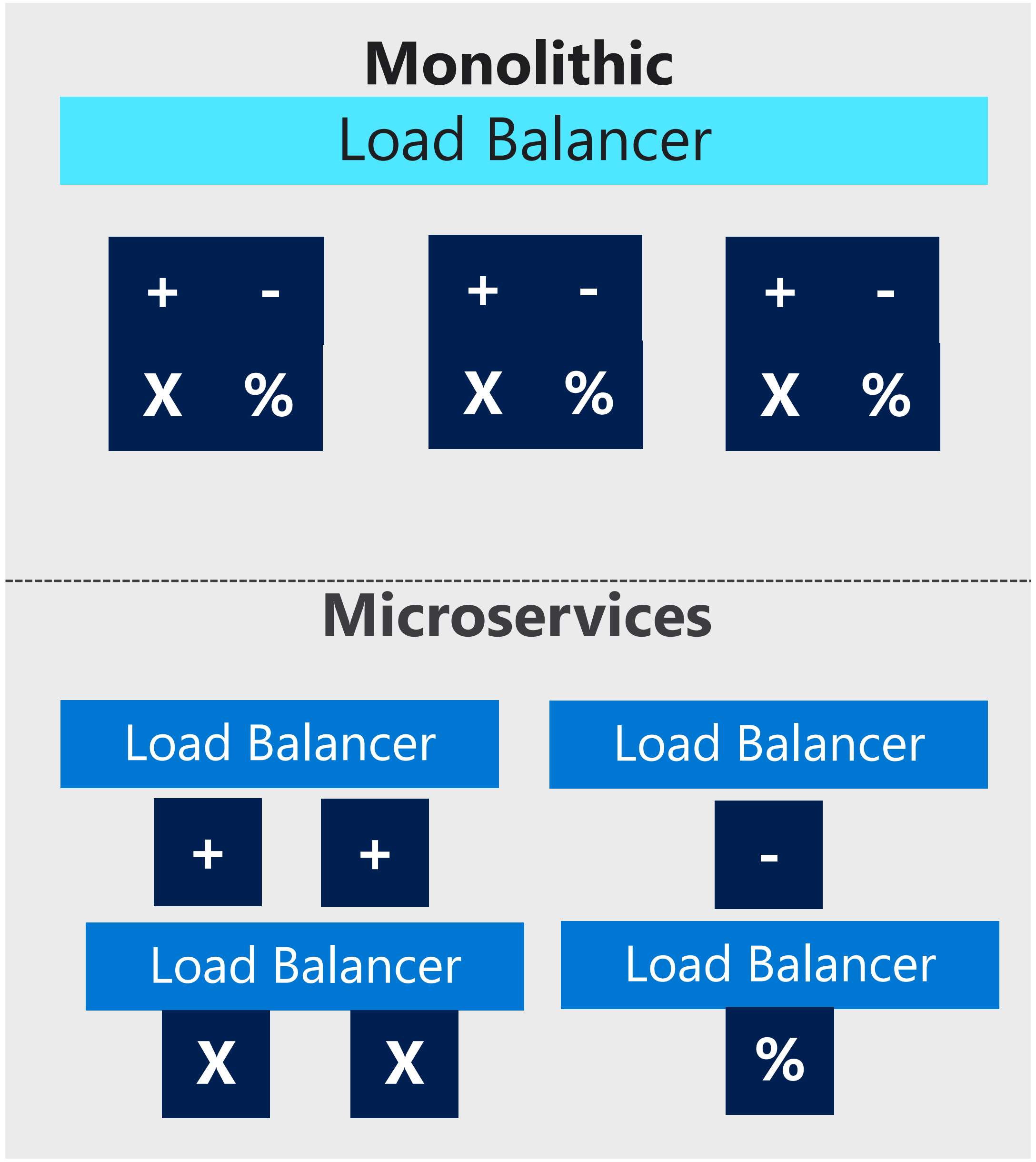


Square the number function easily added with New microservices

Other Advantages of Microservices:

- Simplicity. Each microservices performs only one distinct and well-defined.
- Scalability. To scale a monolithic application, we need to deploy resource-heavy applications on multiple servers behind a load balancer. ***It is not possible to scale just a portion of an application***; *it is all or nothing*. With microservices, we can scale out only only the components that are expected to be highly loaded.
- Continuous delivery.
- More freedom and fewer dependencies.
- Fault isolation

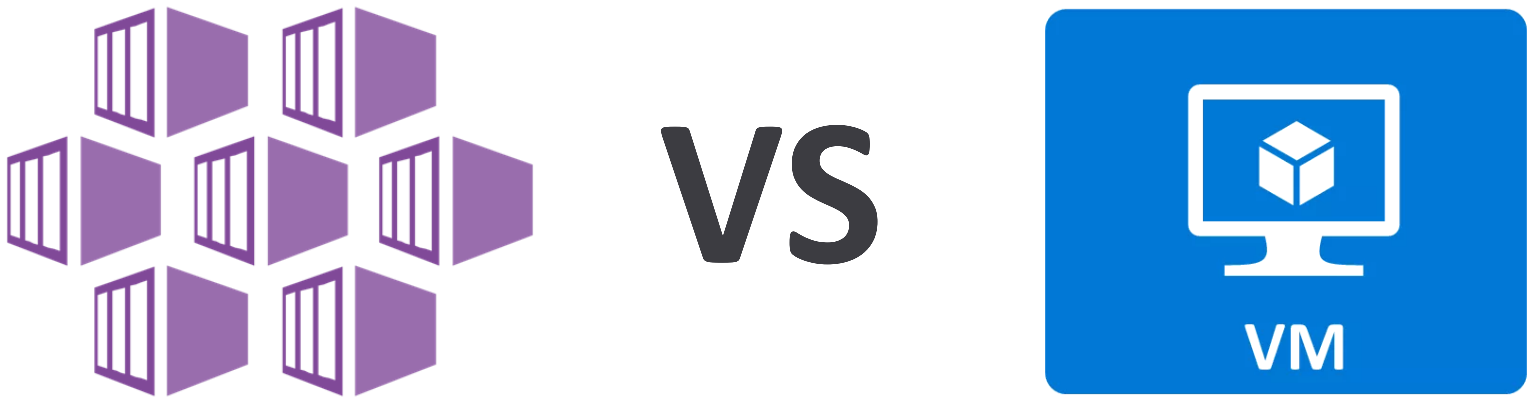
Scalability Comparison



Containers Vs Virtual Machines

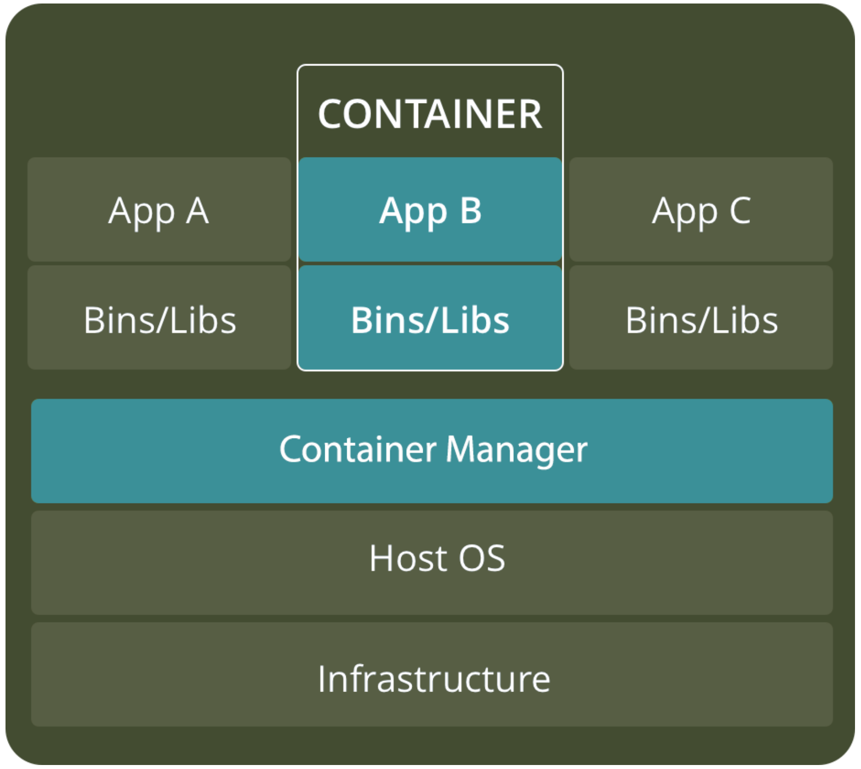
Containers VS Virtual Machine

A container, on the other hand, is nothing more than a single isolated process running in the host OS, consuming only the resources that the app consumes and without the overhead of any additional processes.



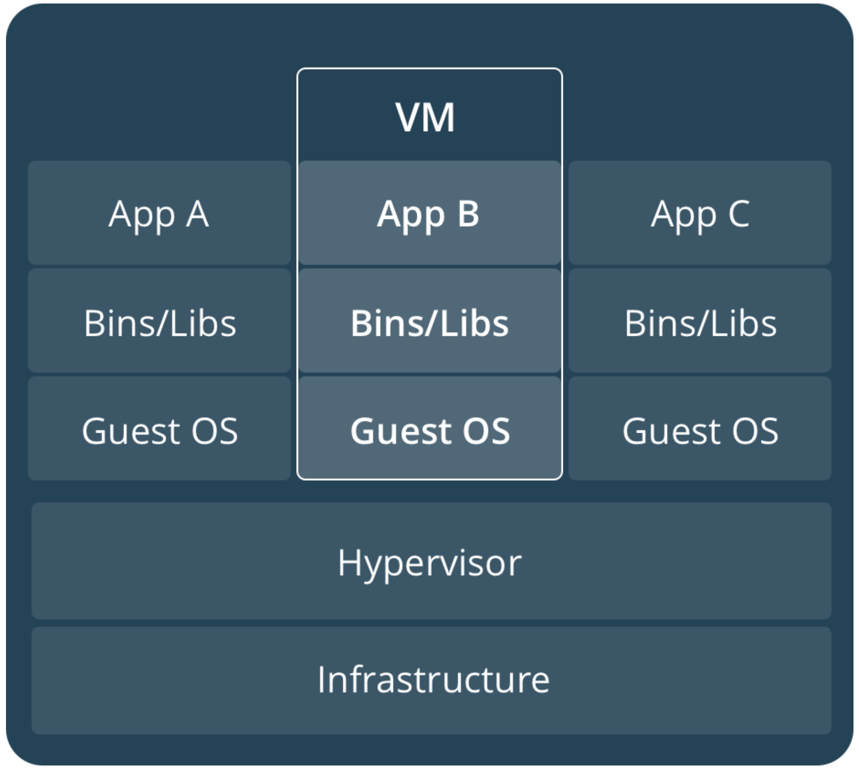
Containers VS Virtual Machine

A **container** runs *natively* on Linux and shares the kernel of the host machine with other containers. It runs a discrete process, taking no more memory than any other executable, making it lightweight.

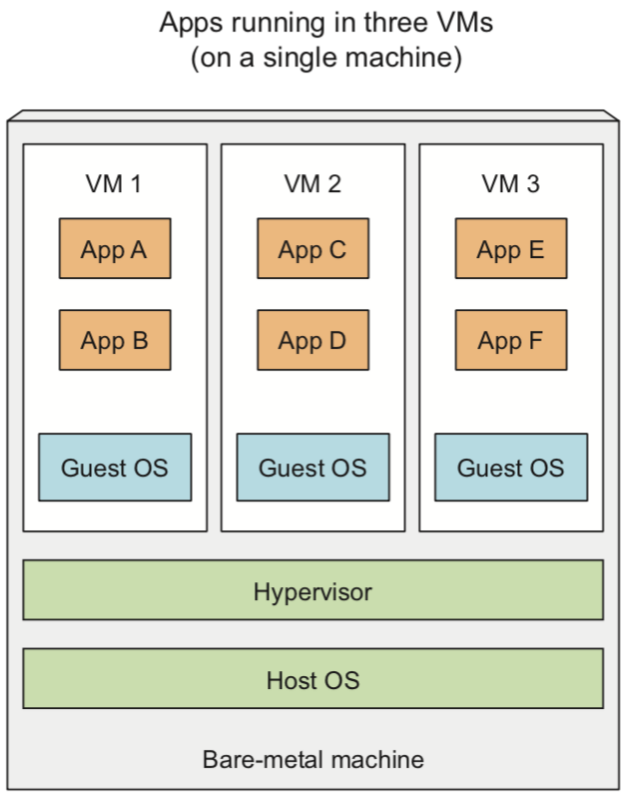


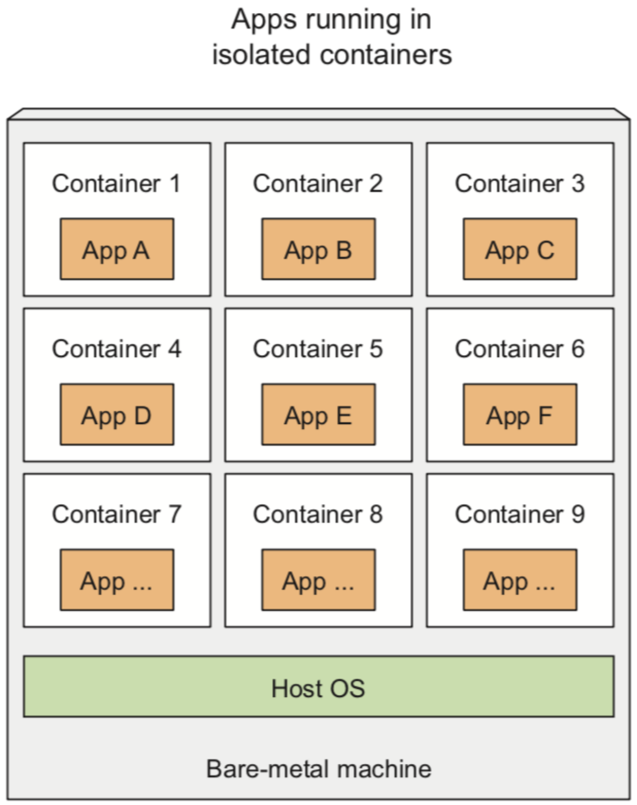
VS

Virtual machines (VMs) are an abstraction of physical hardware turning one server into many servers. The hypervisor allows multiple VMs to run on a single machine. Each VM includes a full copy of an operating system, the application, necessary binaries and libraries - taking up tens of GBs. VMs can also be slow to boot.



Containers VS Virtual Machine





Compared to VMs, containers are much more lightweight, which allows you to run higher numbers of software components on the same hardware, mainly because each VM needs to run its own set of system processes, which requires additional compute resources in addition to those consumed by the component's own process.

A container, on the other hand, is nothing more than a single isolated process running in the host OS, consuming only the resources that the app consumes and without the overhead of any additional processes.

Microsoft Azure

Azure Container Instances (ACI)

Deploy an AKS cluster using Azure CLI

Implementation steps:

1. Create a resource group:

```
az group create --name myAKSCluster --location eastus
```

2. Create an AKS cluster:

```
az aks create --resource-group myAKSCluster --name myAKSCluster --node-count 1 --enable-addons
monitoring --generate-ssh-keys
```

3. Connect to the AKS cluster:

```
az aks get-credentials --resource-group myAKSCluster \
                       --name myAKSCluster
```

4. Create a Kubernetes manifest file (azure-vote.yaml)

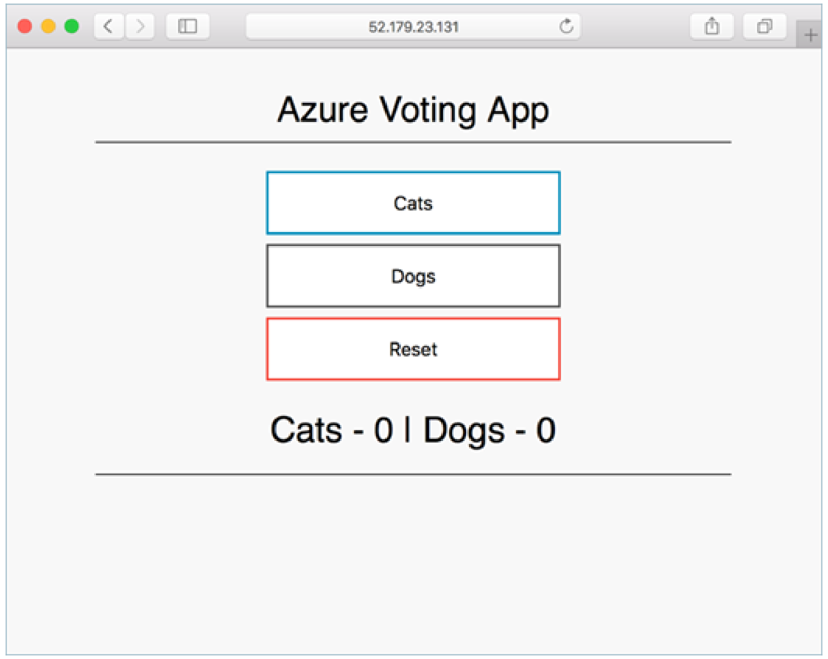
5. Run the containerized application:

```
kubectl apply -f azure-vote.yaml
```

6. Monitor the progress of the deployment:

```
kubectl get service azure-vote-front –watch
```

7. Monitor health and logs (from the Azure portal)



Deploy an AKS cluster using Azure Portal

Implementation steps:

1. Create a resource > Kubernetes Service:

Basics: Project details, Cluster details, Scale
Authentication: A new or existing service principal and RBAC settings
Networking: Http application routing and Network configuration (Basic or Advanced)
Monitoring: A new or existing Log Analytics workspace.

2. Connect to the cluster:

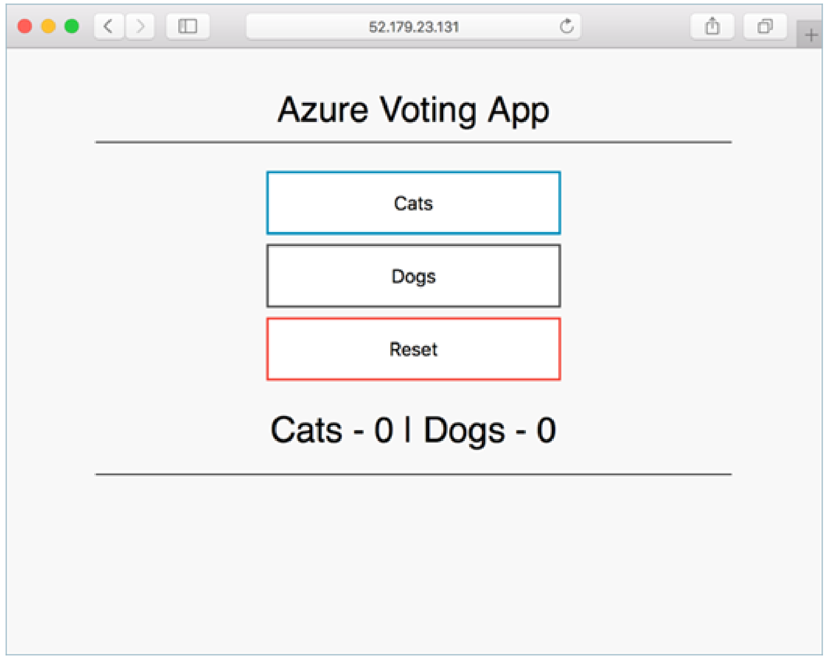
```
az aks get-credentials --resource-group myAKSCluster \
                       --name myAKSCluster
```

3. Create a Kubernetes manifest file (azure-vote.yaml)

4. Run the containerized application:

```
kubectl apply -f azure-vote.yaml
```

5. Monitor health and logs (from the Azure portal)



Azure Container Registry overview

Key concepts:

- **Registry**
- **Repository**
- **Image**
- **Container**

ACR is a managed container registry based on Docker Registry 2.0:

Integrates with Azure Container Registry Build (for building container images)

Is available in three service tiers: Basic, Standard, and Premium

Offers:

- Webhook integration
- Azure AD authentication
- Delete functionality
- Geo-replication (with the Premium tier)

Deploy an image to ACR using Azure CLI

Implementation steps:

1. Create a resource group:

```
az group create --name myResourceGroup --location eastus
```

2. Create a container registry:

```
az acr create --resource-group myResourceGroup --name myContainerRegistry007 --sku Basic
```

3. Log in to ACR:

```
az acr login --name <acrName>
```

4. Push a new image to ACR (or pull and tag an existing image first):

```
docker pull microsoft/aci-helloworld
docker tag microsoft/aci-helloworld <acrLoginServer>/aci-helloworld:v1
docker push <acrLoginServer>/aci-helloworld:v1
```

5. Deploy a container by using an image in ACR:

```
az container create --resource-group myResourceGroup --name acr-quickstart --image
<acrLoginServer>/aci-helloworld:v1 --cpu 1 --memory 1 --registry-username <acrName>
--registry-password <acrPassword> --dns-name-label aci-demo --ports 80
```

Azure Container Instances Overview

A managed container hosting option offering a range of benefits:

- **Fast startup times**
- **Public IP connectivity and DNS name**
- **Hypervisor-level security**
- **Custom sizes**
- **Persistent storage**
- **Linux and Windows containers**
- **Co-scheduled groups**

Implement an application using Virtual Kubelet

Virtual Kubelet provider is an experimental open source project:

- **Allows scheduling AKS-managed containers on Azure Container Instances**
- **Combines functional benefits of AKS with low pricing of ACI**
- **Supports both Windows and Linux containers**

To install the Virtual Kubelet provider:

- **Ensure that the prerequisites are satisfied:**
  - An existing AKS cluster
  - Azure CLI version 2.0.33 and Helm installed
  - A service account and role binding for use with Tiller (for RBAC-enabled AKS clusters)
- **Run the az aks install-connector command:**
  - az aks install-connector --resource-group myAKSCluster --name myAKSCluster --connector-name virtual-kubelet --os-type Both
- **Once the installation completes, you can deploy Windows and Linux containers to ACI:**
  - In the .yaml file, replace kubernetes.io/hostname with the name of the Linux/Windows Virtual Kubelet node

Microsoft Azure

Docker

Features of Docker

Docker is a popular containerization tool used to provide software applications with a filesystem that contains everything they need to run. Using Docker containers ensures that the software will behave the same way, regardless of where it is deployed, because its run-time environment is ruthlessly consistent.

Features of Docker

Docker containers don’t have a separate kernel, as a VM does. Commands run from a Docker container appear in the process table on the host and, in most ways, look much like any other process running on the system.

The difference between an application run in those two environments

File system:

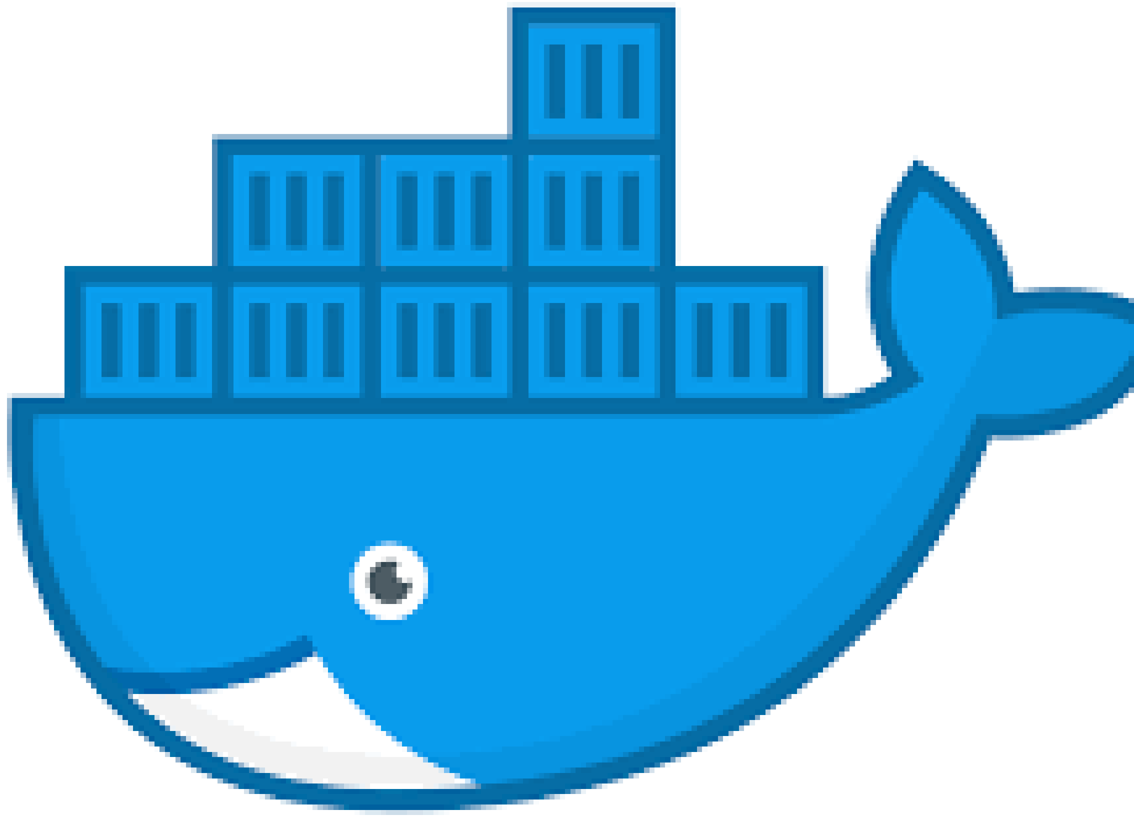
The container has its own file system and cannot see the host system's file system by default. One exception to this rule is that files (such as /etc/hosts and /etc/resolv.conf) may be automatically bind mounted inside the container. Another exception is that you can explicitly mount directories from the host inside the container when you run a container image.

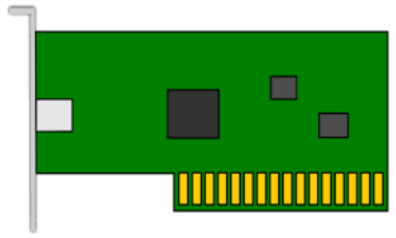
Process table:

Hundreds of processes may be running on a Linux host computer. However, by default, processes inside a container cannot see the host's process table, but instead have their own process table. So the application's process you run when you start up the container is assigned PID 1 within the container. From inside the container, a process cannot see any other processes running on the host that were not launched inside the container.

Network interfaces:

By default, the Docker daemon defines an IP address via DHCP from a set of private IP addresses. Instead of using DHCP, Docker supports other network modes, such as allowing containers to use another container's network interfaces, the host's network interfaces directly, or no network interfaces. If you choose, you can expose a port from inside the container to the same or different port number on the host.





IPC facility:

Processes running inside containers cannot interact directly with the inter-process communications (IPC) facility running on the host system. You can expose the IPC facility on the host to the container, but that is not done by default. Each container has its own IPC facility.

Device:

Processes inside the container cannot directly see devices on the host system. Again, a special privilege option can be set when the container is run to grant that privilege.

Docker Images and Containers:

Containerization is to gather together all the components an application needs to run in a single, contained unit.

Docker unit is referred to as a Docker image.

What is an image?

An image is a static unit that sits in a repository, or the local file system where Docker is installed, and waits to run. When you save a Docker image to a file system, as opposed to storing it in a repository, it is stored as a tarball. That tarball can be transported as you would any other file and then imported later to run as a container on your local system running Docker.

Docker container

Refers to a running instance of a Docker image. Or, more precisely, an instance of an image that has run, since it may be running, paused, or stopped at the moment.

Microsoft

Thank You