Microsoft

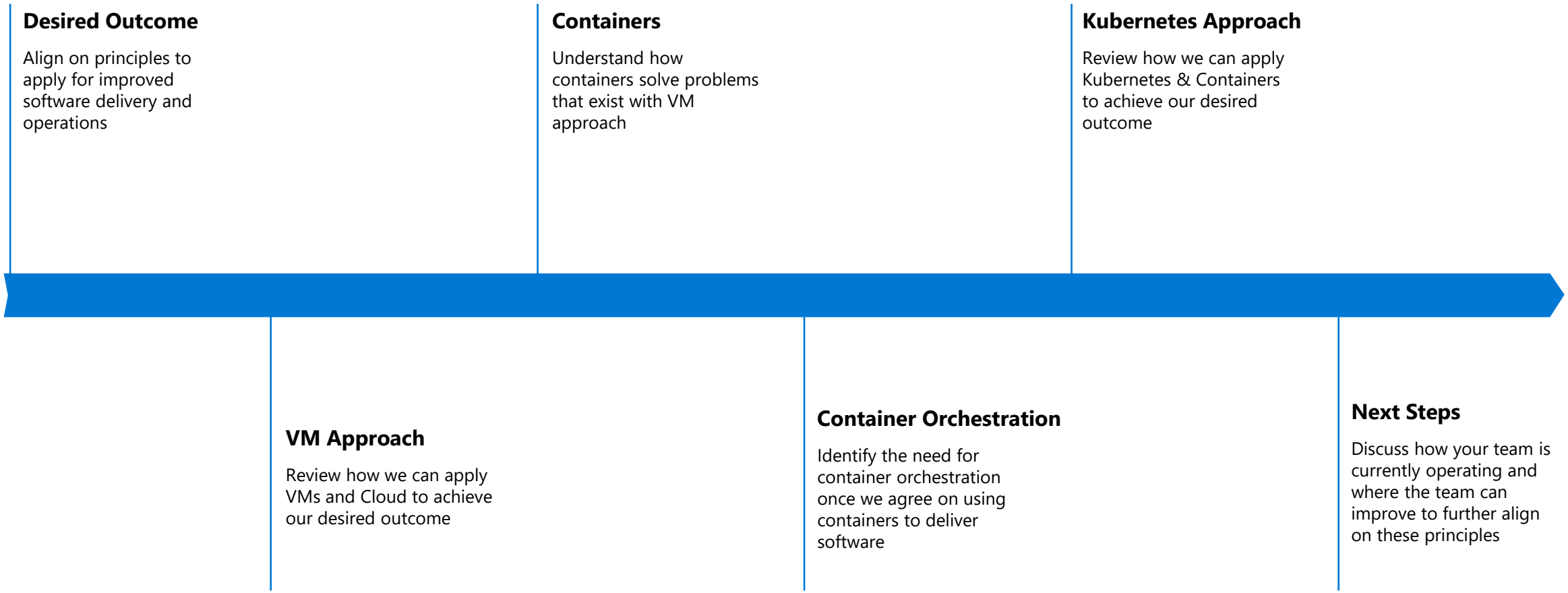
Container Adoption Journey Whitepaper

Haitham Shahin
Cloud Solutions Architect
Microsoft Federal

Goal

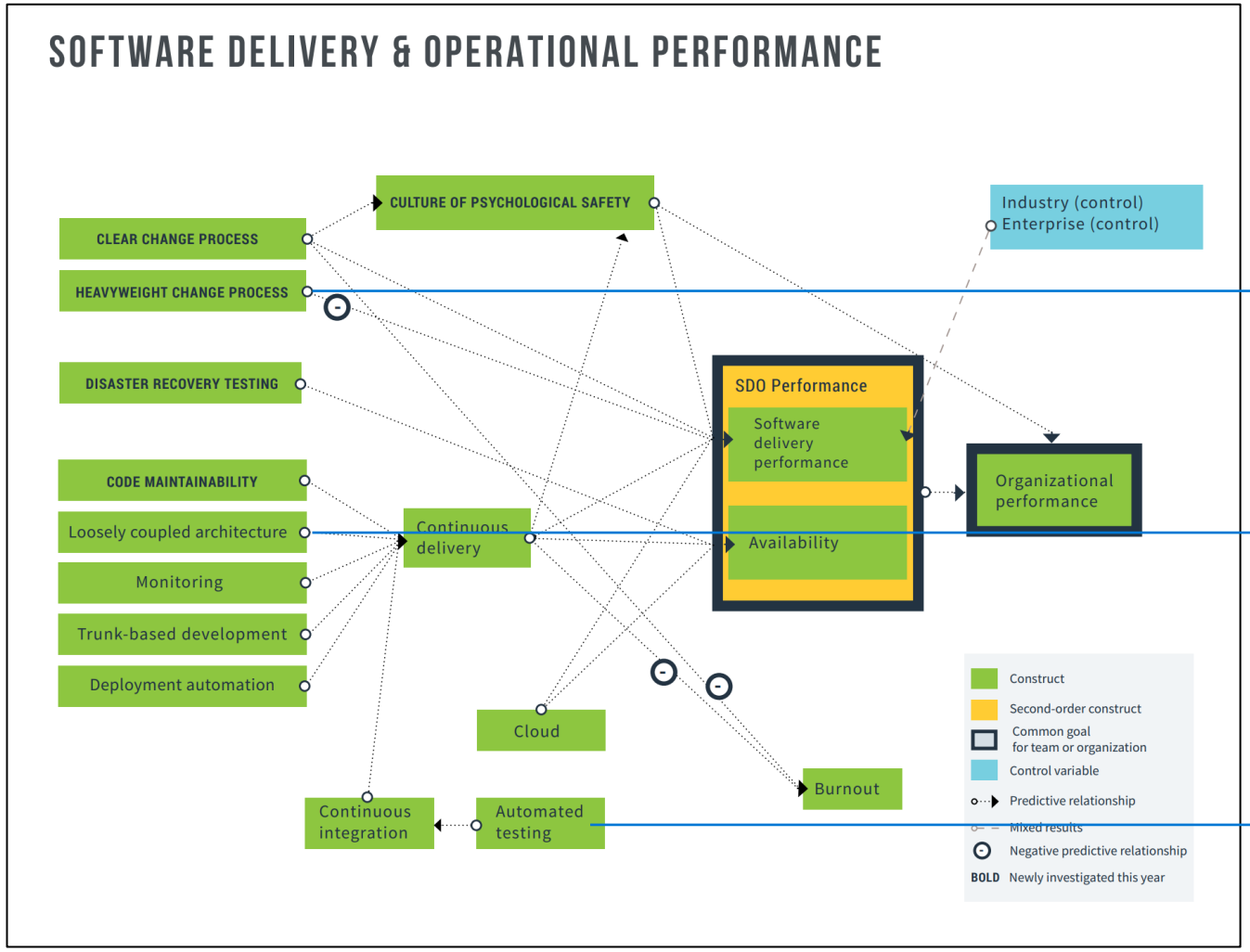
Align on a ***Desired Outcome for Software Delivery*** and how ***Containers & Kubernetes*** help achieve that mission

Container Adoption Journey



What Principles form the Foundation for Good Software Delivery?

Accelerate State of DevOps Report, 2019



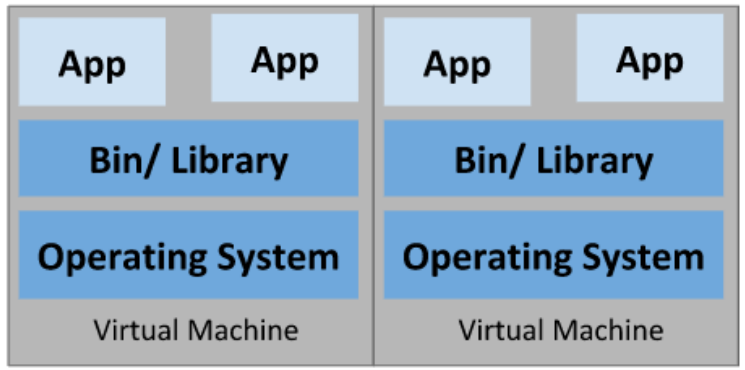
Ability to Change drives **Innovation**

Loosely Coupled Architecture + Automation drives **Agility**

Automated Testing & CI drives **Confidence**

How Can we Achieve These Principles with VMs?

Looking at a Virtual Machine



Virtualized Deployment

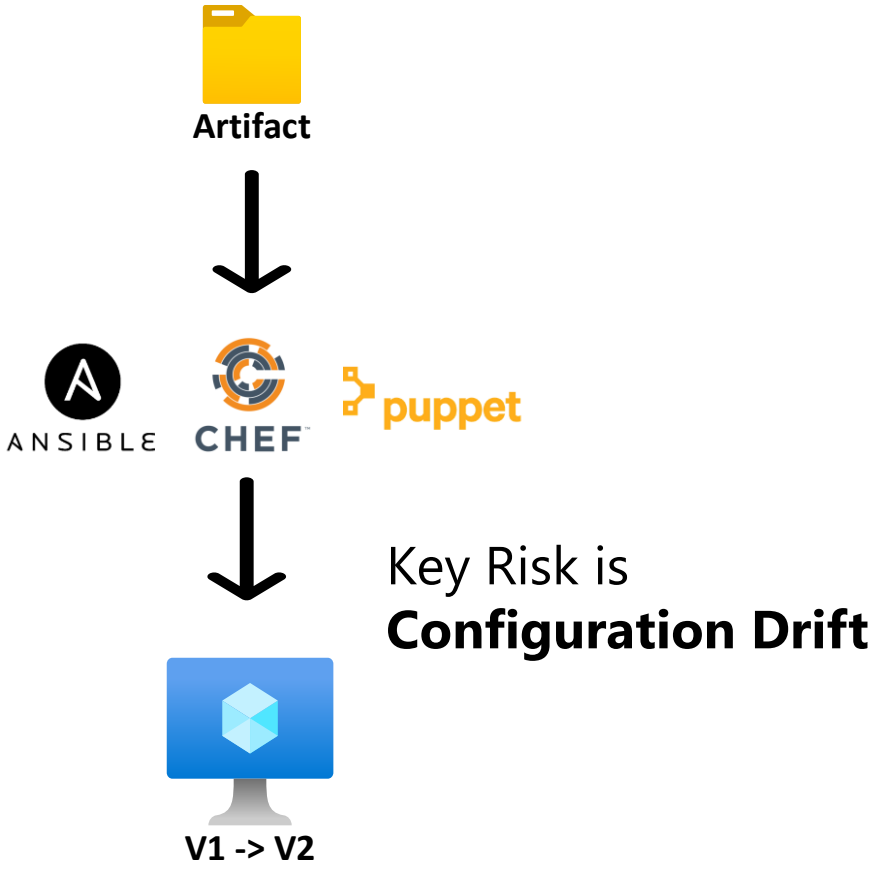
Shipping Kernel + User Space

Host Kernel and Hypervisor Layer as VM Runtime

[What is Kubernetes? | Kubernetes]

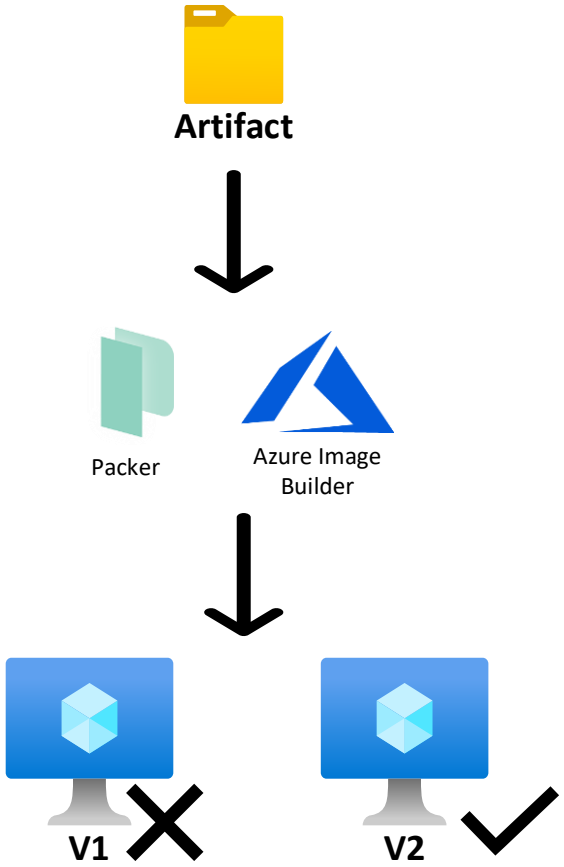
How Should we Approach Deployment Automation

Mutable



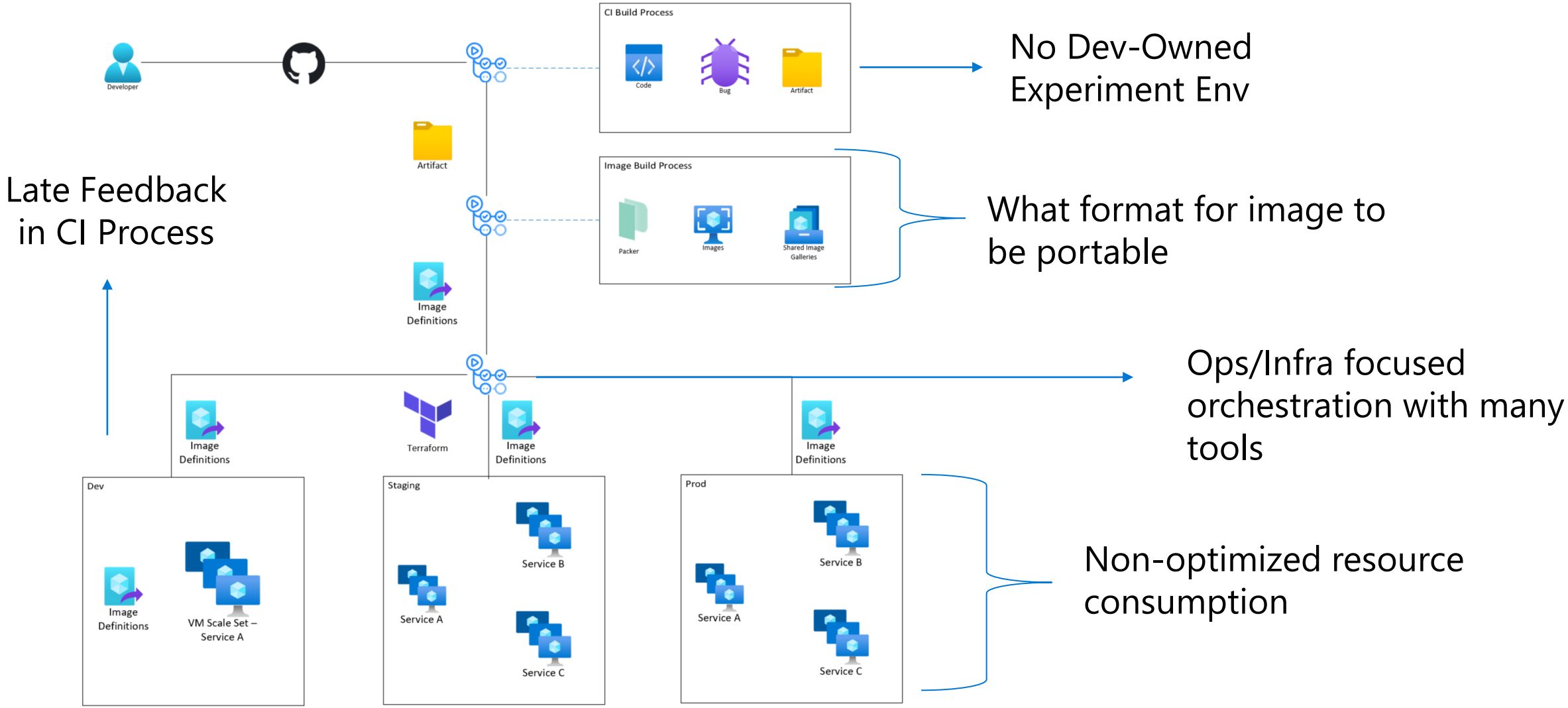
Key Risk is
Configuration Drift

Immutable



More orchestration
required to deploy

Example Architecture for DevOps Automation (Immutable Approach)



Azure Examples for VMs and Immutable Infrastructure

[Immutable infrastructure CI/CD using Jenkins and Terraform on Azure - Azure Solution Ideas | Microsoft Docs]

[Learn about Azure Image Builder - Azure Virtual Machines | Microsoft Docs]

[Build custom virtual machine images with GitHub Actions and Azure | Microsoft Docs]

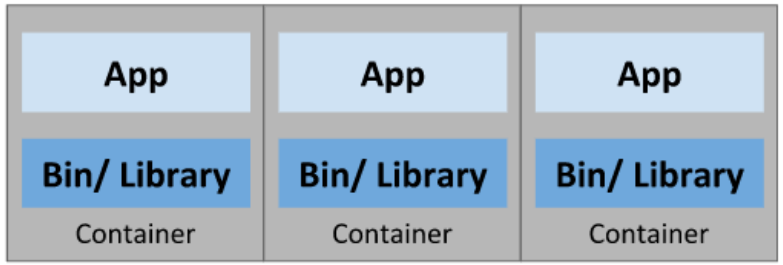
Tutorials on Azure using Azure Image Builder and GitHub Actions to deploy Immutable VMs

Takeaway: VMs Are Not Optimized for Delivery of Loosely-Coupled Architecture and Developer Agility

1. **Lack of Portability** through Coupling to Cloud/Platform Specifics
2. **Underutilized Resource Consumption** across Services
3. **Custom Orchestration and Automation** Required by Ops/Infra

How do Containers Address VM Gaps?

Looking at a Container



User Space Only

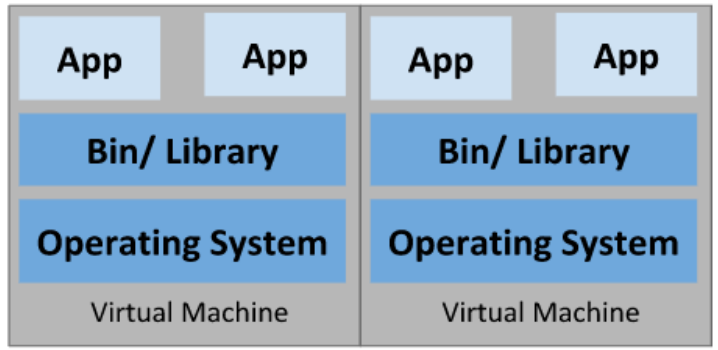
OCI Compliant Format & Runtime

Shared Kernel on Host

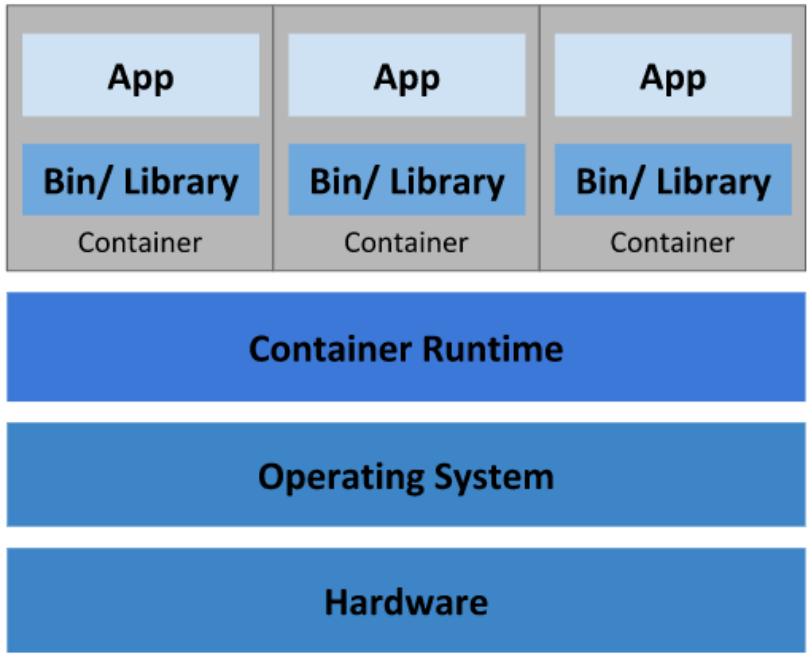
Container Deployment

[What is Kubernetes? | Kubernetes]

Virtual Machine vs Container



Virtualized Deployment



Container Deployment

Containers are run as isolated processes on a **Shared Kernel**

OCI standardizes image and runtime specs for container runtimes on host OS

[What is Kubernetes? | Kubernetes]

Quick Demo for Deeper Understanding: Container

```
hshahin@DESKTOP-631K2UD:~/repos/my-repos/Federal-App-Innovation-Community$ docker run -d nginx
0e5364b1a65dcf1d5a6e94650148e1bca957c56f99656825c551f2de526a0b60
hshahin@DESKTOP-631K2UD:~/repos/my-repos/Federal-App-Innovation-Community$ docker run -d ubuntu sleep 3600
cbec2c504765764df6796a4f5014b241a7c9a224b115754a68a4e270be277a3a
hshahin@DESKTOP-631K2UD:~/repos/my-repos/Federal-App-Innovation-Community$ docker run -d debian sleep 4500
83cc23e805b282f9adc9908e8d27f6232a18b9446054c766ab88836a6fa792a0
hshahin@DESKTOP-631K2UD:~/repos/my-repos/Federal-App-Innovation-Community$ ps axf
```

All 3 Containers running as processes on host

```
  820 pts/0    Sl+    0:03      |                   \_ /home/hshahin/.vscode-server/bin/dfd34e8260c270da74b5c2d86d61aee4b6d56977/node /home/hshahin/.vscode-server/extensions/redhat.vscode-yaml-1.6.0/dist/lan
 1305 ?        Sl     0:16      \_ /usr/bin/dockerd -p /var/run/docker.pid
 1319 ?        Ssl    0:10      |   \_ containerd --config /var/run/docker/containerd/containerd.toml --log-level info
 4450 ?        Sl     0:00      \_ /usr/bin/containerd-shim-runc-v2 -namespace moby -id 0d79cb0061ea54160d902b3ffd4c5668ed128b6af49acbe3780eede6ee0186b4 -address /var/run/docker/containerd/containerd.sock
 4470 ?        Ss     0:00      |   \_ sleep 3500
 6775 ?        Sl     0:00      \_ /usr/bin/containerd-shim-runc-v2 -namespace moby -id 0e5364b1a65dcf1d5a6e94650148e1bca957c56f99656825c551f2de526a0b60 -address /var/run/docker/containerd/containerd.sock
 6795 ?        Ss     0:00      |   \_ nginx: master process nginx -g daemon off;
 6853 ?        S      0:00      |       \_ nginx: worker process
 6854 ?        S      0:00      |       \_ nginx: worker process
 6855 ?        S      0:00      |       \_ nginx: worker process
 6856 ?        S      0:00      |       \_ nginx: worker process
 6857 ?        S      0:00      |       \_ nginx: worker process
 6858 ?        S      0:00      |       \_ nginx: worker process
 6859 ?        S      0:00      |       \_ nginx: worker process
 6860 ?        S      0:00      |       \_ nginx: worker process
 6926 ?        Sl     0:00      \_ /usr/bin/containerd-shim-runc-v2 -namespace moby -id cbec2c504765764df6796a4f5014b241a7c9a224b115754a68a4e270be277a3a -address /var/run/docker/containerd/containerd.sock
 6947 ?        Ss     0:00      |   \_ sleep 3600
 7012 ?        Sl     0:00      \_ /usr/bin/containerd-shim-runc-v2 -namespace moby -id 83cc23e805b282f9adc9908e8d27f6232a18b9446054c766ab88836a6fa792a0 -address /var/run/docker/containerd/containerd.sock
 7032 ?        Ss     0:00      |   \_ sleep 4500
   44 ?        Ss     0:00 /init
```

Quick Demo for Deeper Understanding: Container

```
hshahin@DESKTOP-631K2UD:~$ docker run -it ubuntu /bin/bash
root@7fbafac6509d:/#
```

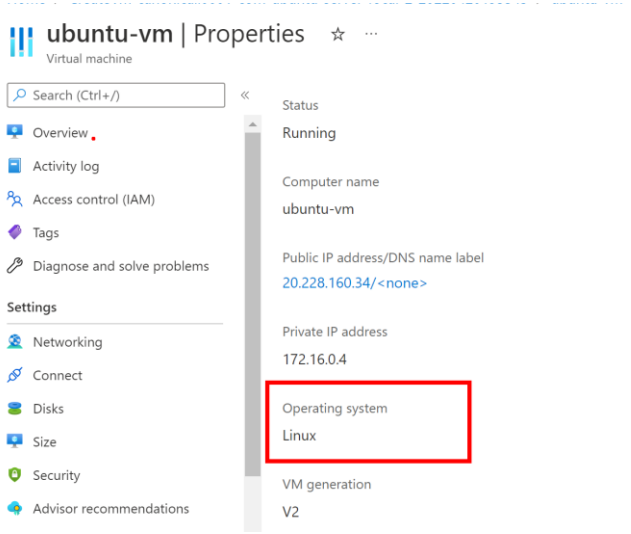
```
root@9133e1f01815:/# tree -L 1 /boot/
/boot/

0 directories, 0 files
root@9133e1f01815:/#
```

No Kernel in /boot inside container image

```
hshahin@DESKTOP-631K2UD:~$ docker run -it ubuntu /bin/bash
root@7fbafac6509d:/# uname -r
5.10.102.1-microsoft-standard-WSL2
root@7fbafac6509d:/# exit
exit
hshahin@DESKTOP-631K2UD:~$ uname -r
5.10.102.1-microsoft-standard-WSL2
hshahin@DESKTOP-631K2UD:~$
```

uname command returns host kernel version

Quick Demo for Deeper Understanding: VM



Deploy Ubuntu VM in Azure and ssh into the VM to run commands below

```
azureuser@ubuntu-vm:~$ tree -L 1 /boot/
/boot/
├── System.map-5.13.0-1022-azure
├── config-5.13.0-1022-azure
├── efi
├── grub
├── initrd.img -> initrd.img-5.13.0-1022-azure
├── initrd.img-5.13.0-1022-azure
├── initrd.img.old -> initrd.img-5.13.0-1022-azure
├── vmlinuz -> vmlinuz-5.13.0-1022-azure
├── vmlinuz-5.13.0-1022-azure
└── vmlinuz.old -> vmlinuz-5.13.0-1022-azure

2 directories, 8 files
azureuser@ubuntu-vm:~$ uname -r
5.13.0-1022-azure
azureuser@ubuntu-vm:~$
```

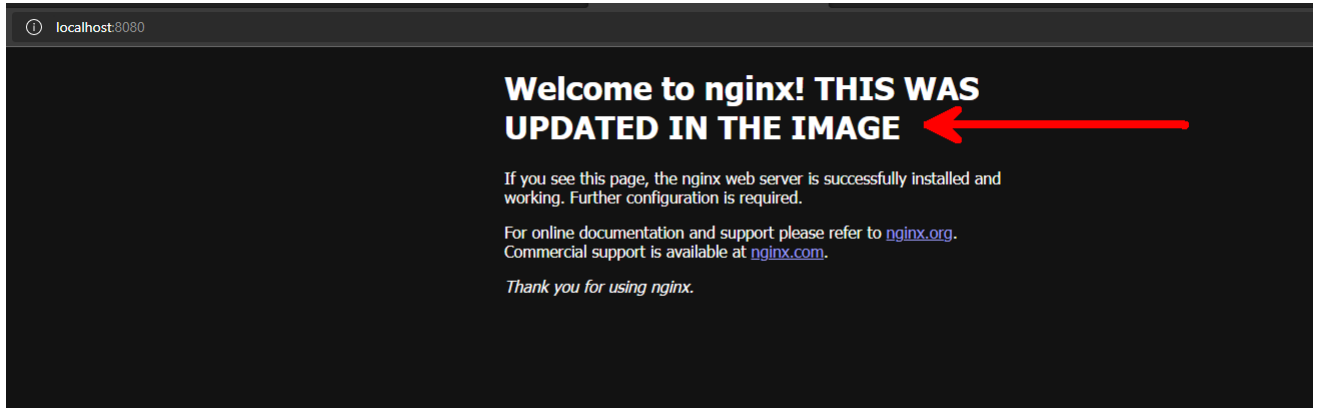
/boot directory contains Kernel

Quick Demo for Deeper Understanding: Immutability

```
hshahin@DESKTOP-631K2UD:~/repos/my-repos/Federal-App-Innovation-Community$ docker run -d -p 8080:80 nginx
03ac13c99d8e99bad5b2c9e6fd18b577b044d93c595a57d9963bf4bf626f17a5
hshahin@DESKTOP-631K2UD:~/repos/my-repos/Federal-App-Innovation-Community$ docker exec -it 03ac13c99d8e99bad5b2c9e6fd18b577b044d93c595a57d9963bf4bf626f17a5 /bin/bash
```

```
root@03ac13c99d8e:/# nano /usr/share/nginx/html/index.html
root@03ac13c99d8e:/#
```

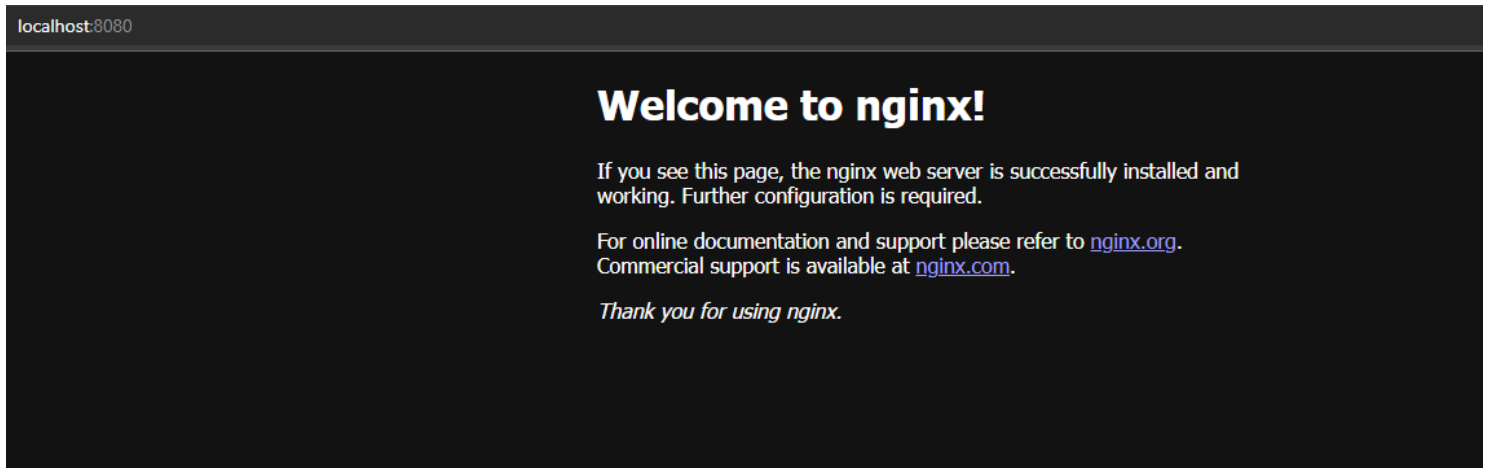
```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx! THIS IS UPDATED IN THE IMAGE</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to ngin! THIS WAS UPDATED IN THE IMAGE</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
```



In-Place Update of index.html file to simulate “SSH”

Quick Demo for Deeper Understanding: Immutable

```
hshahin@DESKTOP-631K2UD:~/repos/my-repos/Federal-App-Innovation-Community$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS                                   NAMES
03ac13c99d8e   nginx     "/docker-entrypoint.…"   6 minutes ago   Up 22 seconds   0.0.0.0:8080->80/tcp, :::8080->80/tcp   happy_mayer
hshahin@DESKTOP-631K2UD:~/repos/my-repos/Federal-App-Innovation-Community$ docker stop 03ac13c99d8e
03ac13c99d8e
hshahin@DESKTOP-631K2UD:~/repos/my-repos/Federal-App-Innovation-Community$ docker run -d -p 8080:80 nginx
5ef1bc5f85433b216228187525bcd028d6b86015bfb313126cebb572252d842f
hshahin@DESKTOP-631K2UD:~/repos/my-repos/Federal-App-Innovation-Community$
```



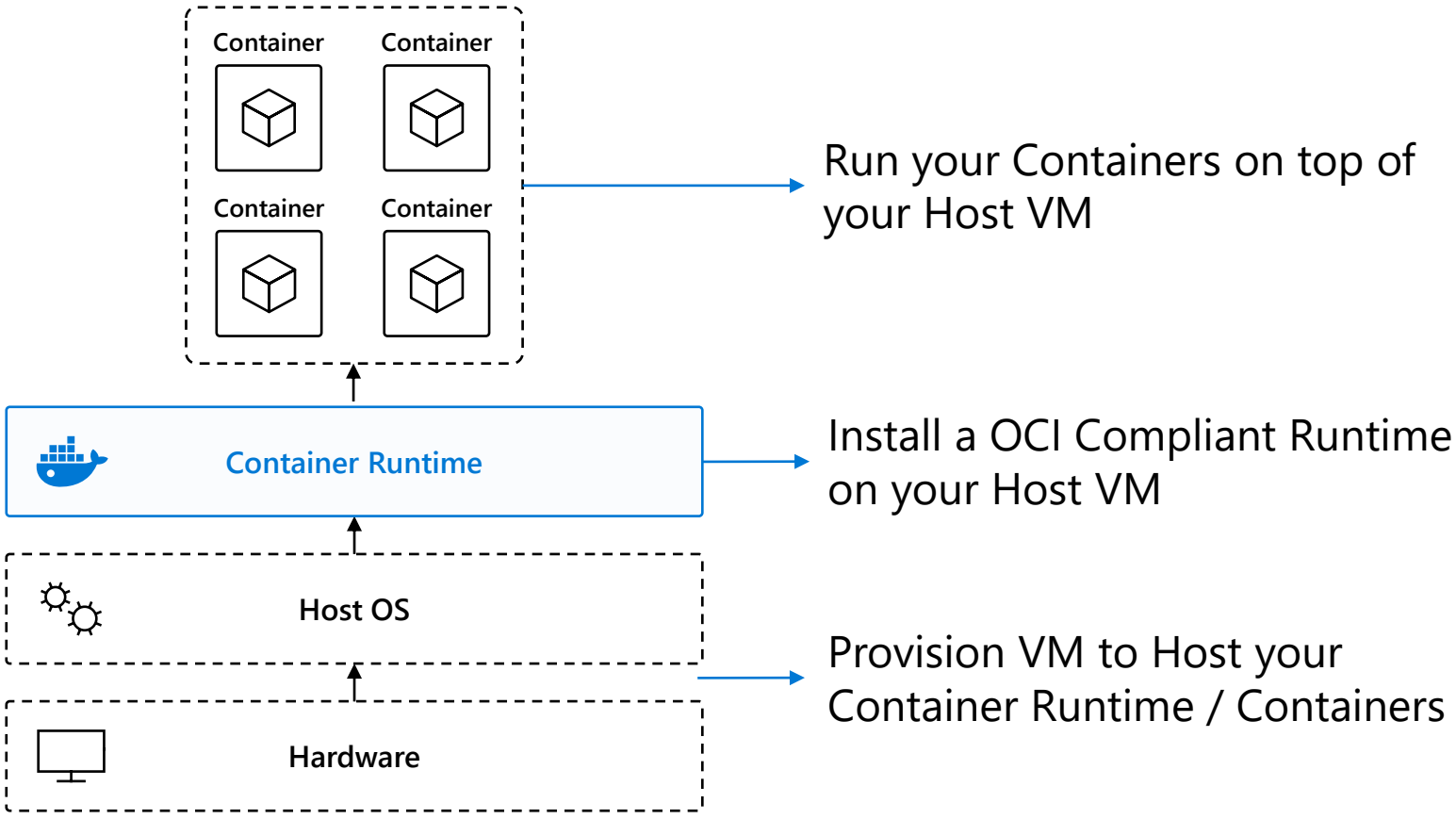
A stop/start of same image does not maintain the prior update

Takeaway: Containers Provide a Standardized Application Package to Run On Shared Host

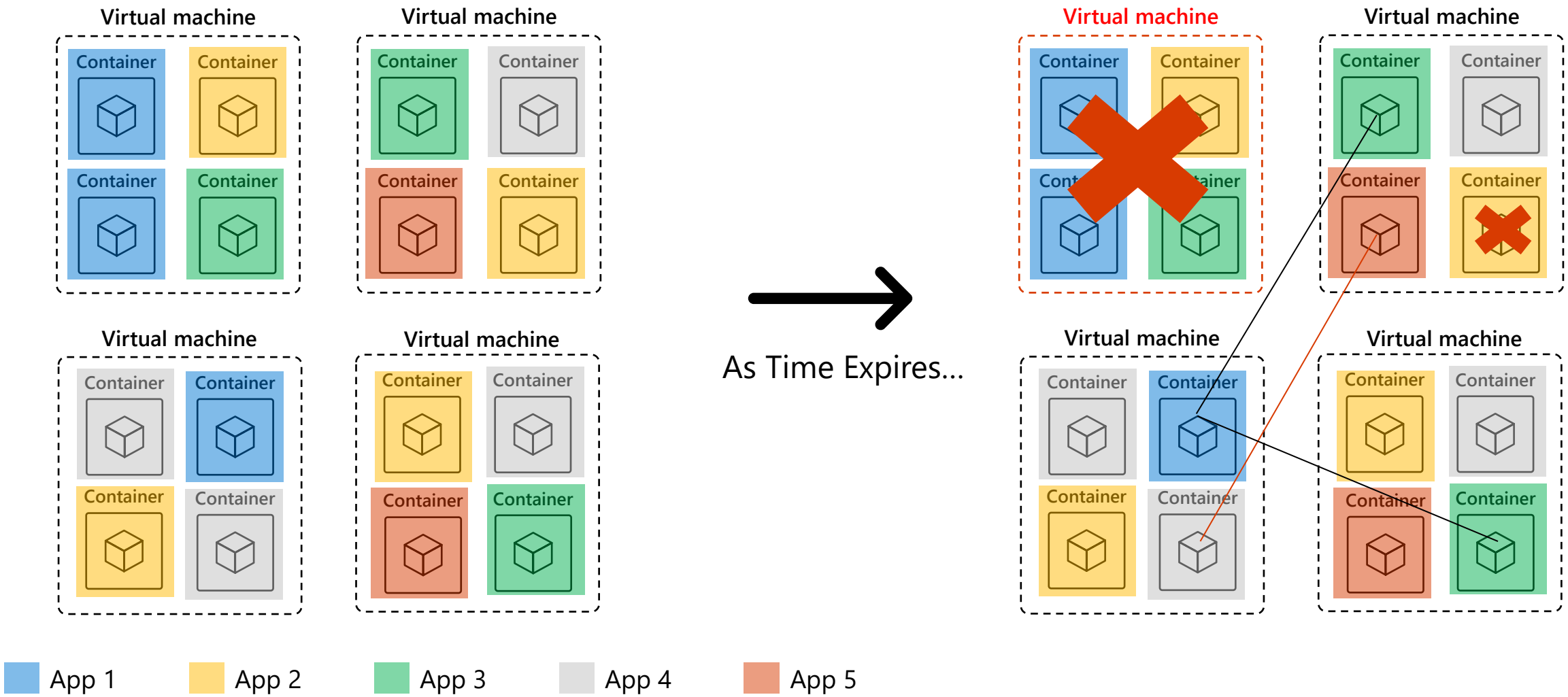
1. **Portability** “Write Once, Run Anywhere” by OCI Standardization
2. **Resource Utilization Improved** by Shared Kernel from Host
3. **Immutable by Default** as a Set of Files on the Host

How do I Take Advantage of the Container Benefits and Run at Scale?

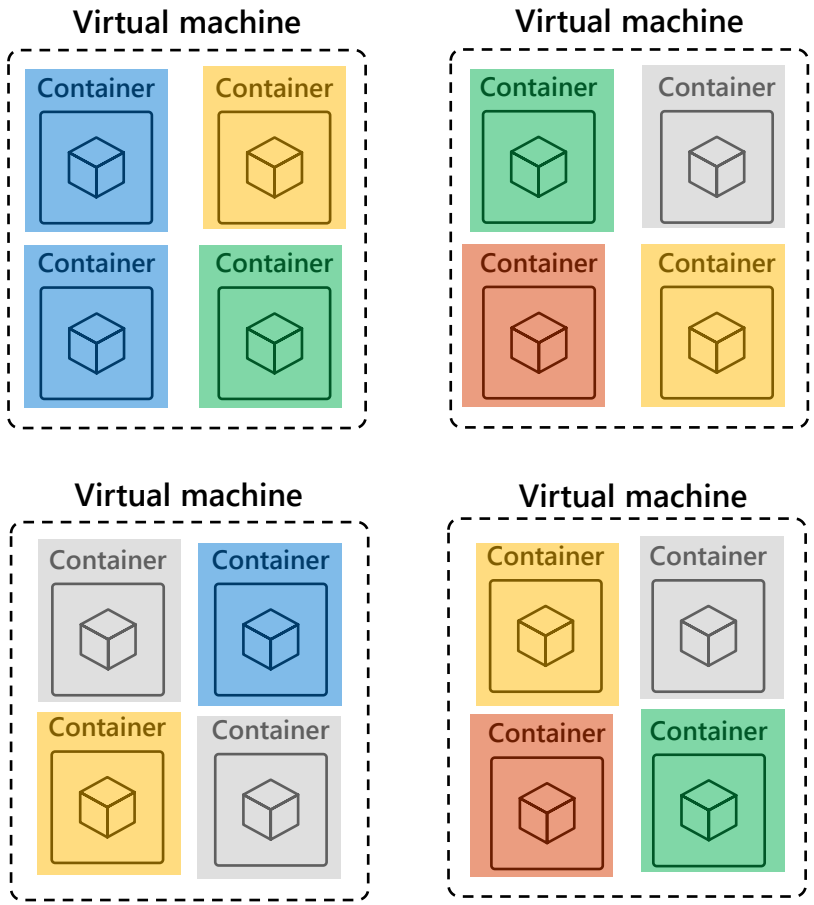
Running Containers Natively on Host VM

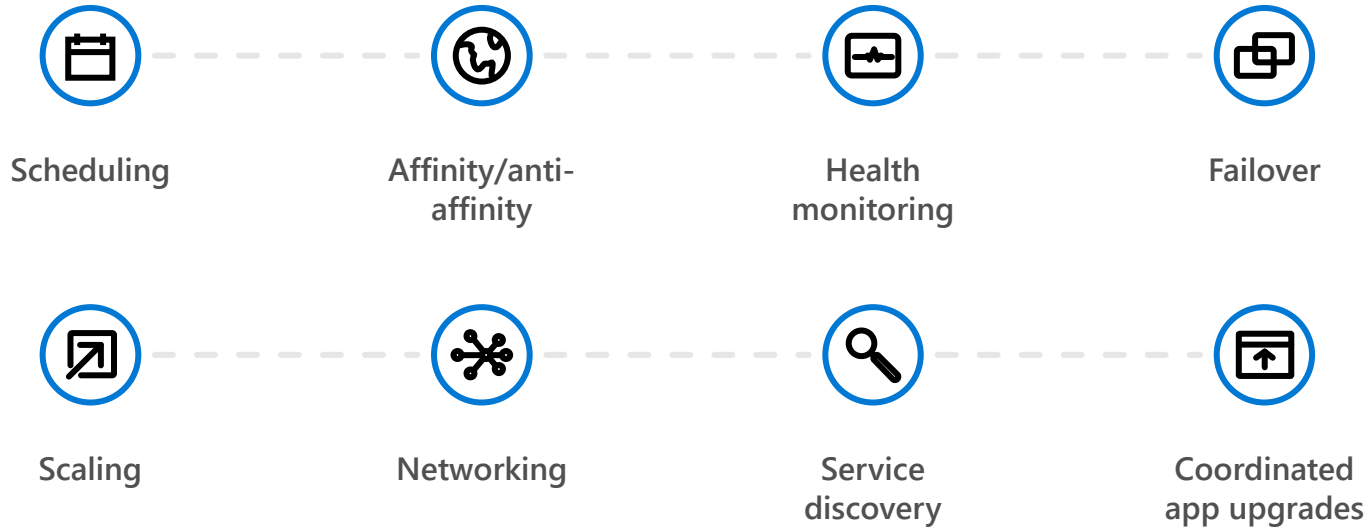


Running Services as Containers Across VMs



What do I Need to Actually Run Containers on VMs

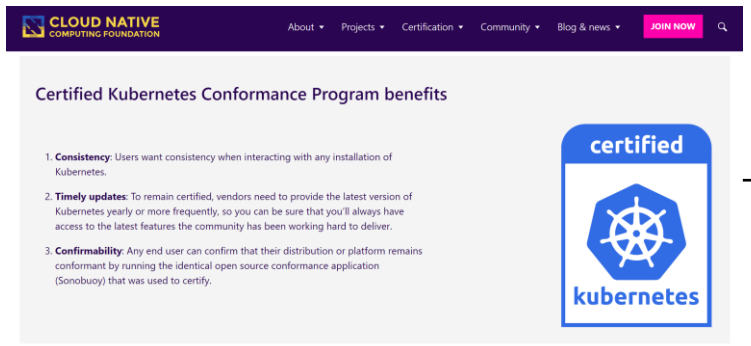




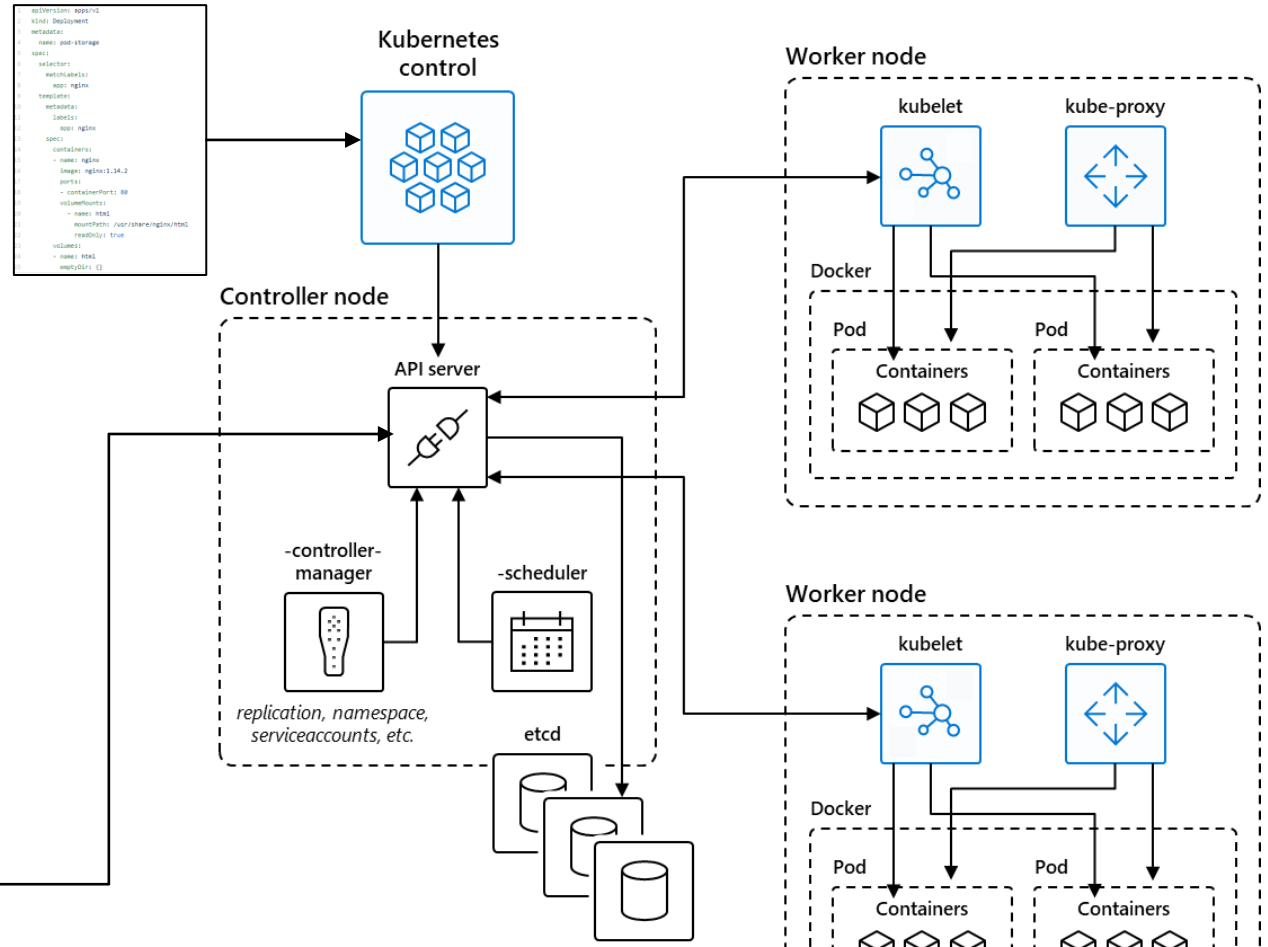
App 1 App 2 App 3 App 4 App 5

Kubernetes for Container Orchestration

1. Kubernetes users communicate with **API server and apply declarative state**
2. Controller nodes actively **enforce desired state on workers**
3. Worker nodes support **communication between containers**
4. Worker nodes **ensure container is running and healthy**



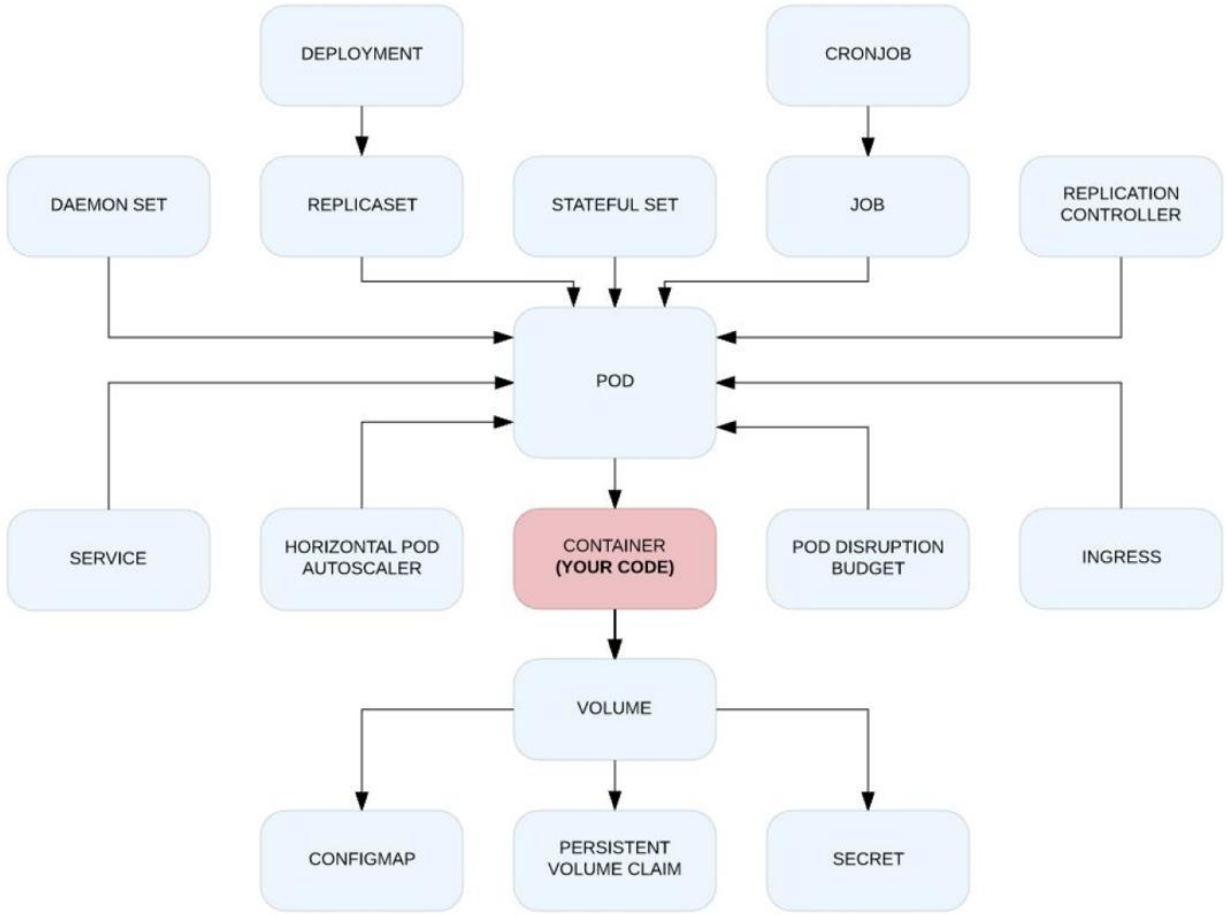
Standardized API for Portability



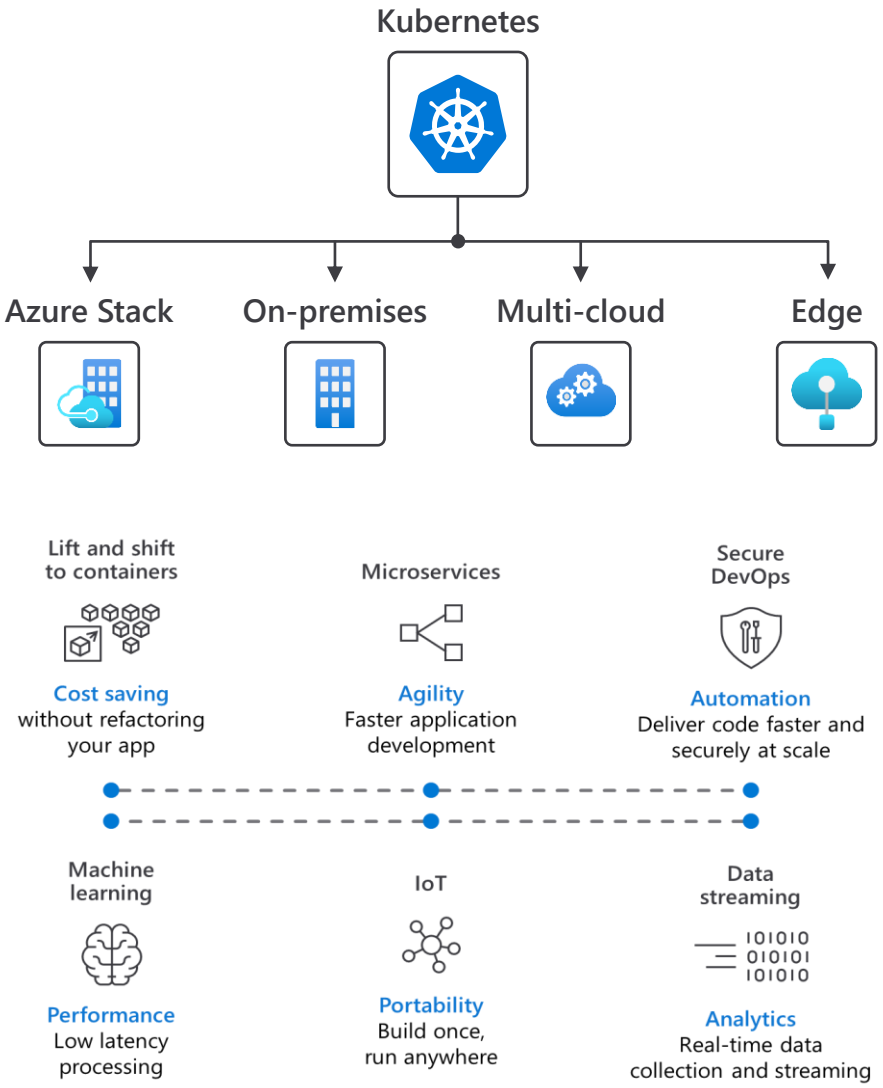
kubernetes

Diving Deeper into Kubernetes API

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: file-storage-claim
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: azurefile-premium
  resources:
    requests:
      storage: 100Gi
---
apiVersion: v1
kind: Service
metadata:
  name: volumes-lb
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: nginx
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: dynamic-shared-storage
spec:
  selector:
    matchLabels:
      app: nginx
  template:
```



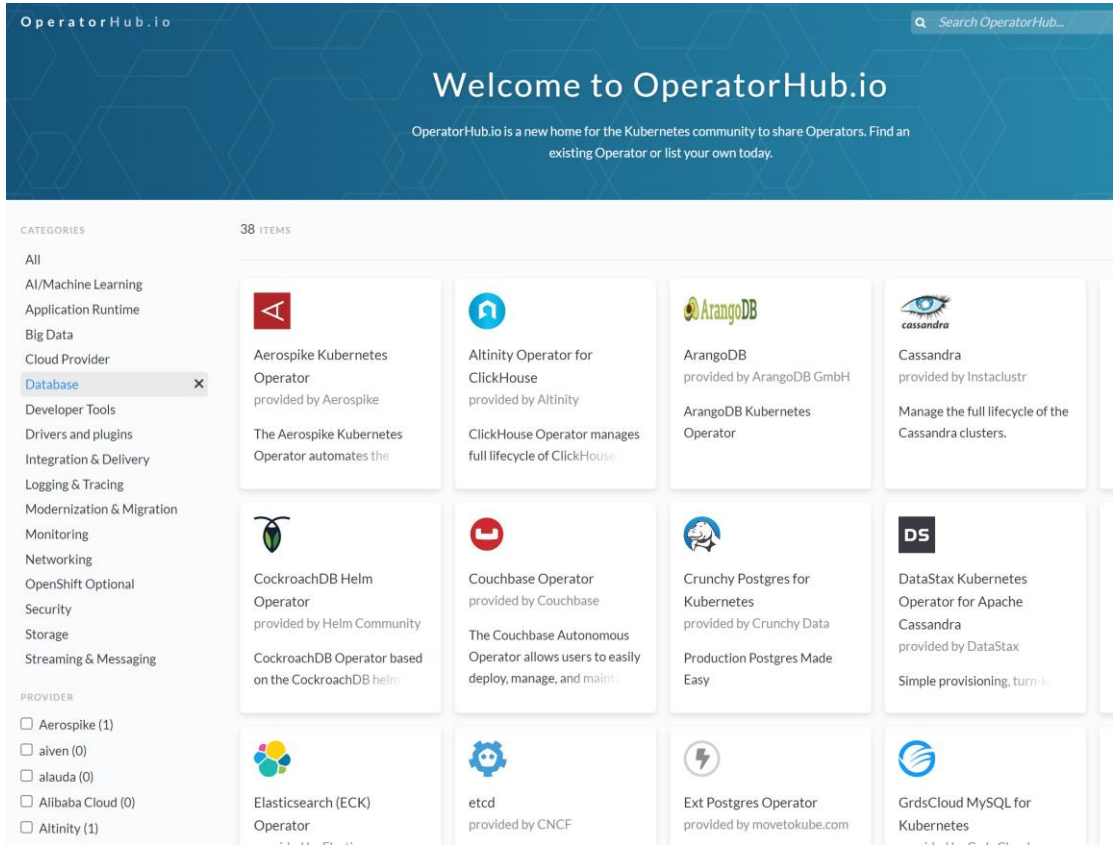
Standardization Across Environments and Use Cases

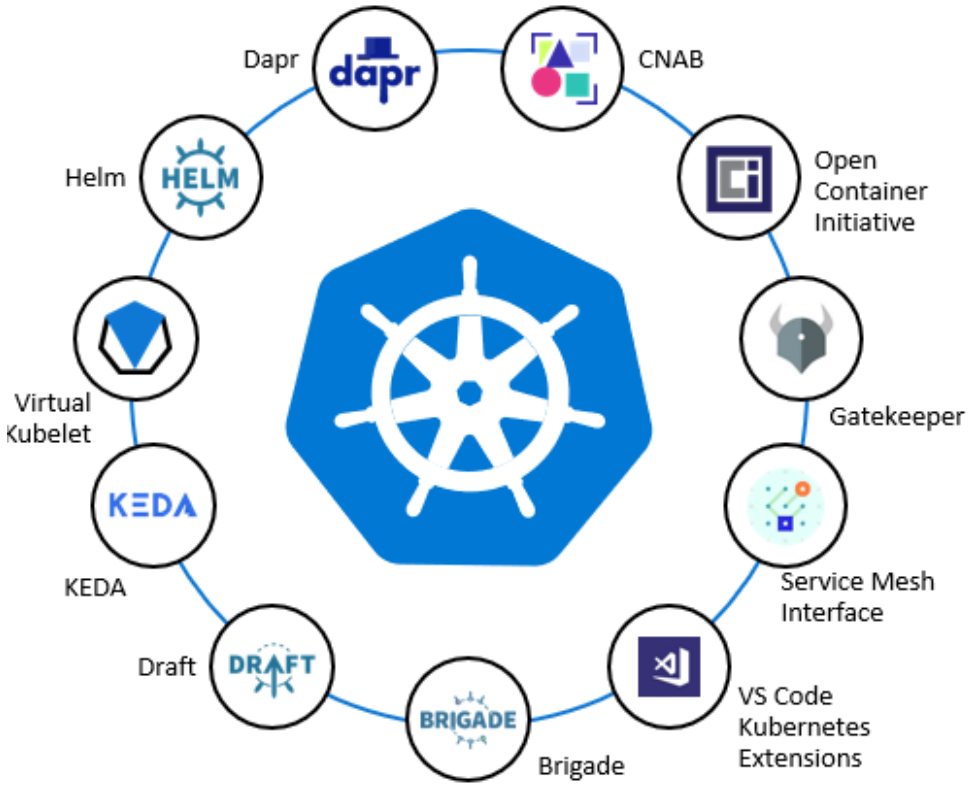


Standardized API enables consistent operations and deployment automation

Apply **Consistent Operation/Governance** Principles Across workloads

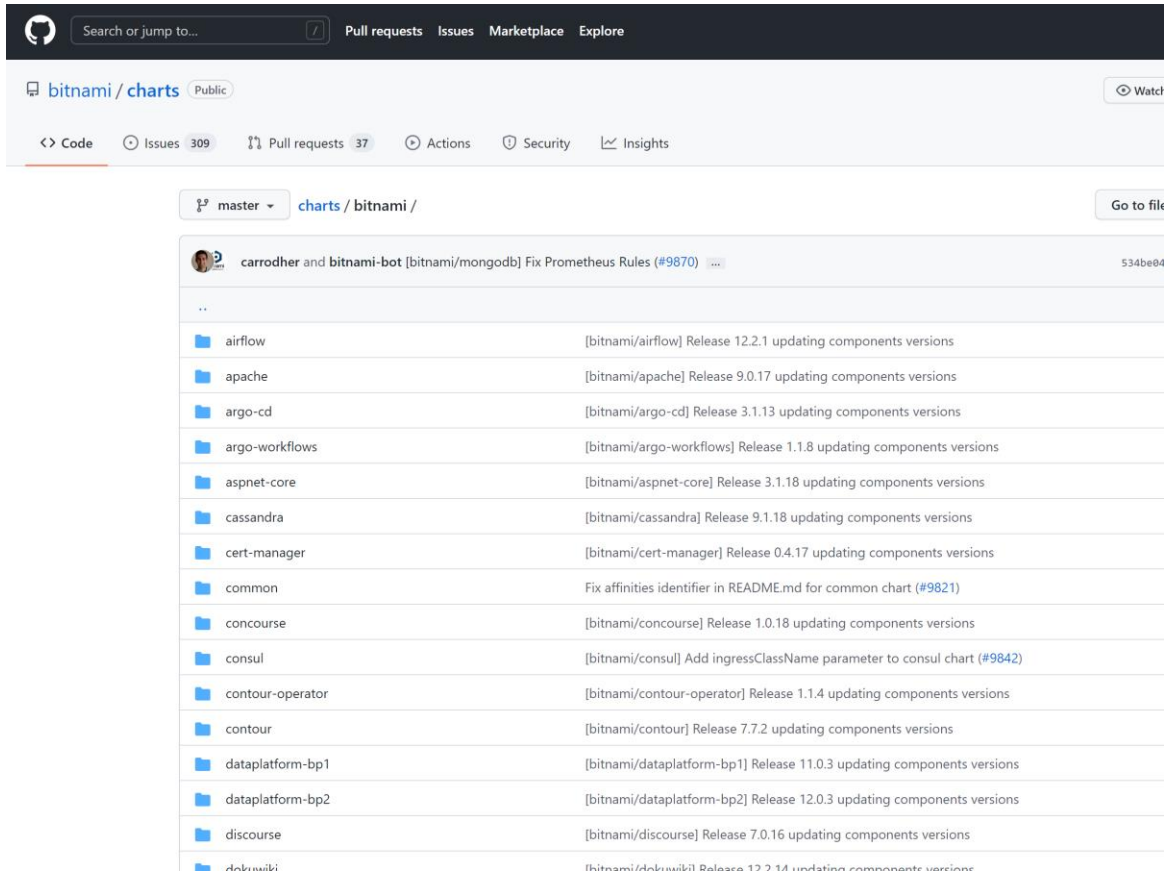
Extending the Kubernetes API and Accelerating Innovation through Community





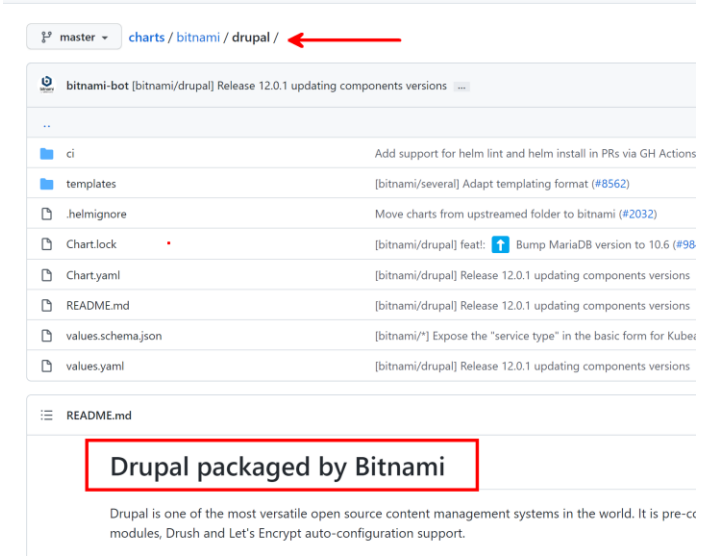
Operators allow you to extend the Kubernetes API with **Custom Resource Definitions and Controllers** that monitor those resources like a traditional Kubernetes Resource

Example Showing the Power of Kubernetes API



Community-Driven Helm Charts provide ways to deploy common applications all codified in Kubernetes-Native Resources

Example Showing the Power of Kubernetes API



Install Helm CLI locally and run the commands below against your Kubernetes cluster to deploy Drupal

TL;DR

```
$ helm repo add bitnami https://charts.bitnami.com/bitnami
$ helm install my-release bitnami/drupal
```

Example Showing the Power of Kubernetes API

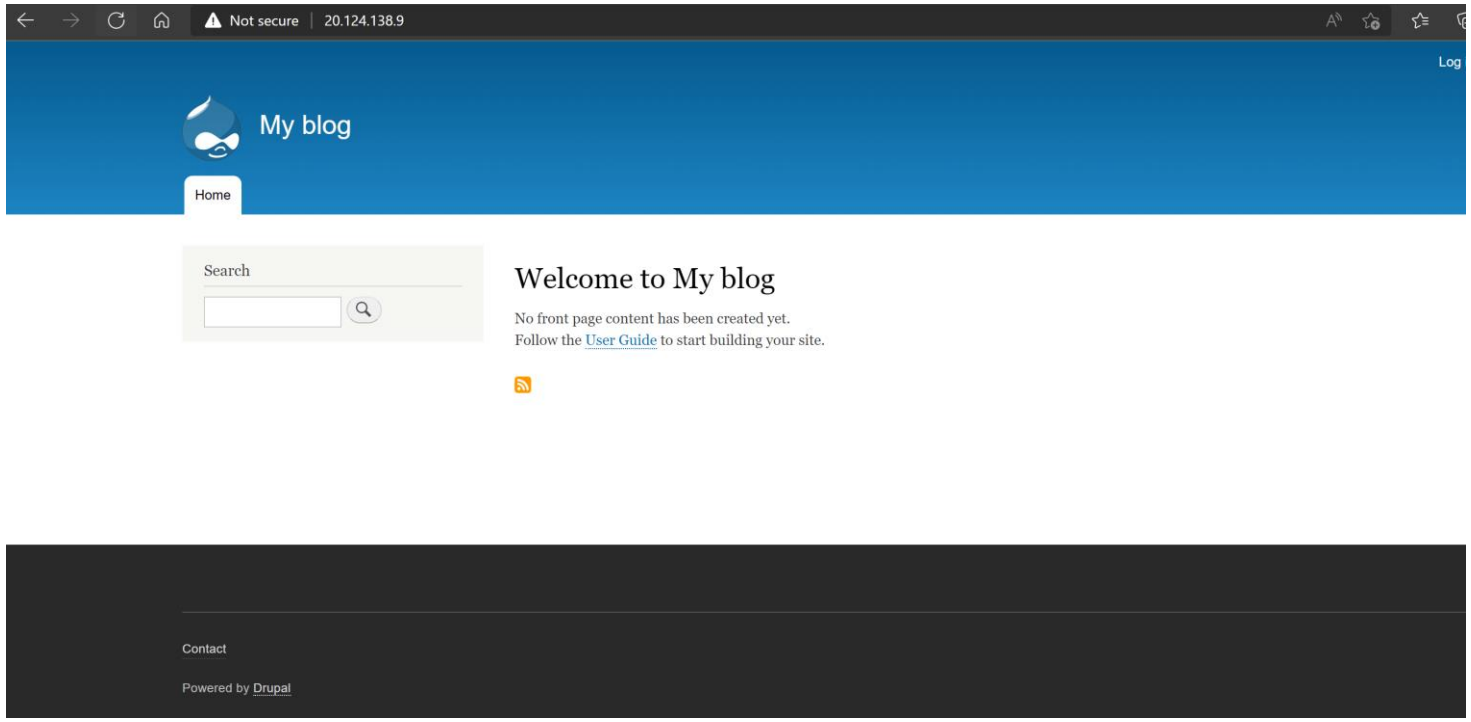
```
hshahin@DESKTOP-631K2UD:~/repos/my-repos/Federal-App-Innovation-Community/topics/kubernetes/whitepapers/container-adoption-journey$ helm repo add bitnami https://charts.bitnami.com/bitnami
"bitnami" already exists with the same configuration, skipping
hshahin@DESKTOP-631K2UD:~/repos/my-repos/Federal-App-Innovation-Community/topics/kubernetes/whitepapers/container-adoption-journey$ helm install my-release bitnami/drupal
NAME: my-release
LAST DEPLOYED: Fri Apr 22 11:01:46 2022
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
CHART NAME: drupal
CHART VERSION: 11.0.29
APP VERSION: 9.3.9** Please be patient while the chart is being deployed **

1. Get the Drupal URL:

  NOTE: It may take a few minutes for the LoadBalancer IP to be available.
        Watch the status with: 'kubectl get svc --namespace default -w my-release-drupal'

  export SERVICE_IP=$(kubectl get svc --namespace default my-release-drupal --template "{{ range (index .status.loadBalancer.ingress 0) }}{{ . }}{{ end }}")
  echo "Drupal URL: http://$SERVICE_IP/"

2. Get your Drupal login credentials by running:

  echo Username: user
  echo Password: $(kubectl get secret --namespace default my-release-drupal -o jsonpath="{.data.drupal-password}" | base64 --decode)
```

```
hshahin@DESKTOP-631K2UD:~/repos/my-repos/Federal-App-Innovation-Community/topics/kubernetes/whitepapers/container-adoption-journey$ kubectl get pods
NAME                                READY   STATUS              RESTARTS   AGE
my-release-drupal-6689879c85-gk89d   0/1     ContainerCreating   0          37s
my-release-mariadb-0                0/1     Running             0          37s
hshahin@DESKTOP-631K2UD:~/repos/my-repos/Federal-App-Innovation-Community/topics/kubernetes/whitepapers/container-adoption-journey$
```

View the Pods being deployed after running the helm install

Example Showing the Power of Kubernetes API

```
hshahin@DESKTOP-631K2UD:~/repos/my-repos/Federal-App-Innovation-Community/topics/kubernetes/whitepapers/container-adoption-journey$ kubectl get pods
NAME                                  READY   STATUS              RESTARTS   AGE
my-release-drupal-6689879c85-gk89d    0/1     ContainerCreating   0          37s
my-release-mariadb-0                  0/1     Running             0          37s
hshahin@DESKTOP-631K2UD:~/repos/my-repos/Federal-App-Innovation-Community/topics/kubernetes/whitepapers/container-adoption-journey$ kubectl get svc
NAME                 TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)                      AGE
kubernetes           ClusterIP      10.0.0.1       <none>          443/TCP                      34h
my-release-drupal    LoadBalancer   10.0.5.106     20.124.138.9    80:30024/TCP,443:30853/TCP   2m28s
my-release-mariadb   ClusterIP      10.0.114.149   <none>          3306/TCP                     2m28s
```



By default a *LoadBalancer Service* is deployed with a Public IP

Example Showing the Power of Kubernetes API

```
etes/whitepapers/container-adoption-journey$ helm template drupal bitnami/drupal > template.yaml
etes/whitepapers/container-adoption-journey$
```

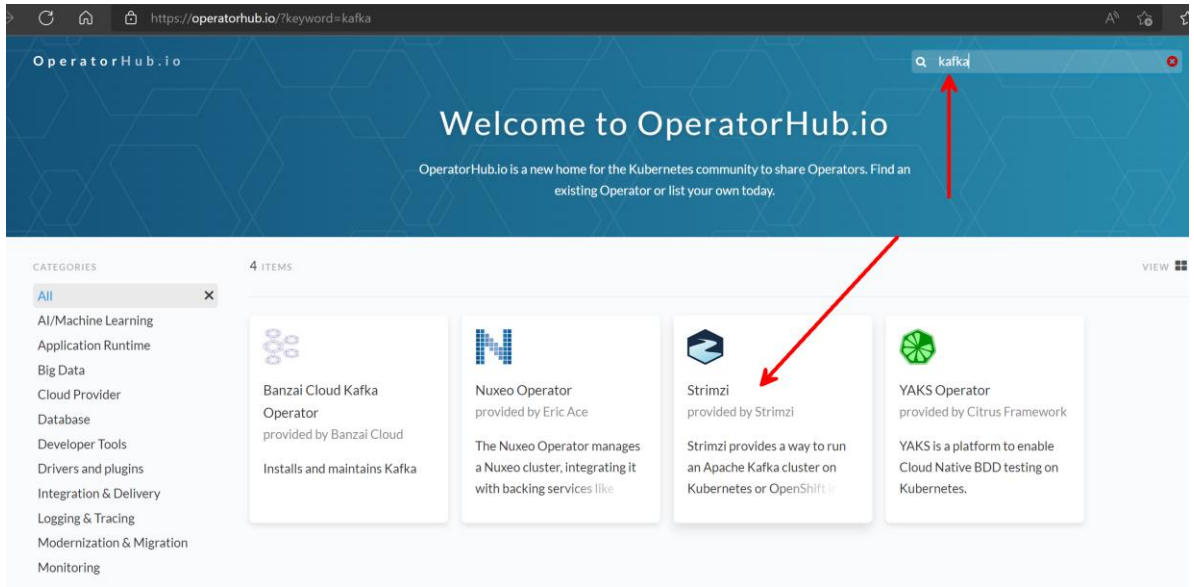
topics > kubernetes > whitepapers > container-adoption-journey > ! template.yaml > ...

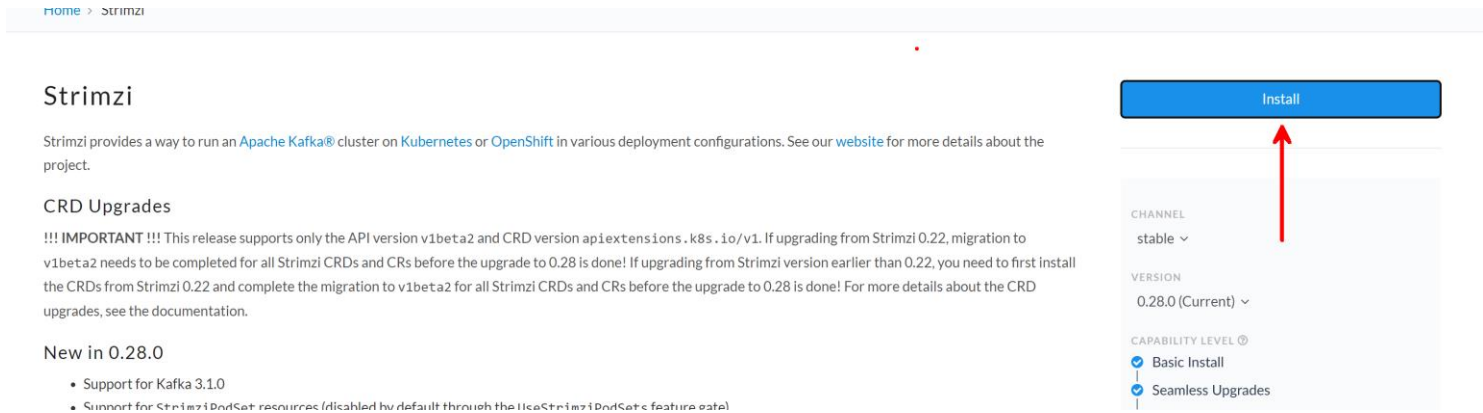
```yaml
---
# Source: drupal/charts/mariadb/templates/serviceaccount.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: drupal-mariadb
  namespace: "default"
  labels:
    app.kubernetes.io/name: mariadb
    helm.sh/chart: mariadb-10.5.0
    app.kubernetes.io/instance: drupal
    app.kubernetes.io/managed-by: Helm
  annotations:
automountServiceAccountToken: false
---
# Source: drupal/charts/mariadb/templates/secrets.yaml
apiVersion: v1
kind: Secret
metadata:
  name: drupal-mariadb
  namespace: "default"
  labels:
    app.kubernetes.io/name: mariadb
    helm.sh/chart: mariadb-10.5.0
    app.kubernetes.io/instance: drupal
```

You can view the YAML that was generated through Helm and deployed to Kubernetes

These are all the native Kubernetes Resources deployed to run Drupal

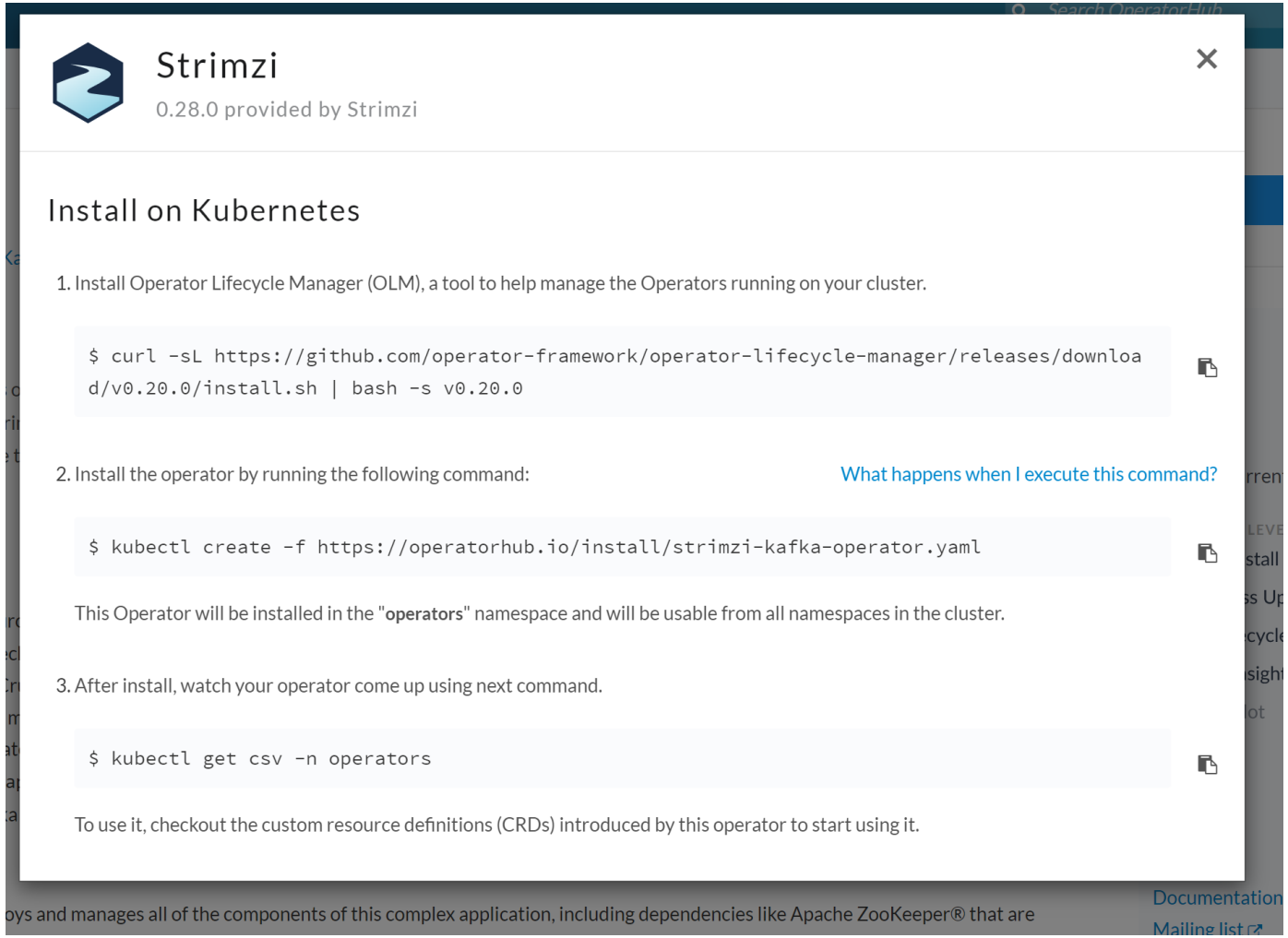
Example Showing the Power of Kubernetes API: Operators





Let's Deploy the Strimzi Operator which allows us to deploy Kafka to Kubernetes

Example Showing the Power of Kubernetes API: Operators



Step 1 will deploy the Operator Lifecycle Manager to your cluster to manage the install and upgrading of your Operator

Step 2 will deploy the Strimzi Operator to the cluster

Step 3 is a way to ensure that the Operator is up and running

Example Showing the Power of Kubernetes API: Operators

```
hshahin@DESKTOP-631K2UD:~/repos/my-repos/Federal-App-Innovation-Community/topics/kubernetes/whitepapers/container-adoption-journey$ kubectl get ns
NAME                STATUS   AGE
cluster-config      Active   34h
default             Active   34h
dev                 Active   34h
flux-system         Active   34h
gatekeeper-system   Active   34h
kube-node-lease     Active   34h
kube-public         Active   34h
kube-system         Active   34h
operators           Active   7m6s
hshahin@DESKTOP-631K2UD:~/repos/my-repos/Federal-App-Innovation-Community/topics/kubernetes/whitepapers/container-adoption-journey$ kubectl get pods -n operators
NAME                                        READY   STATUS    RESTARTS   AGE
strimzi-cluster-operator-v0.28.0-6f7ff86449-6svb5   1/1     Running   0          6m17s
hshahin@DESKTOP-631K2UD:~/repos/my-repos/Federal-App-Innovation-Community/topics/kubernetes/whitepapers/container-adoption-journey$
```

After deploying the Operator, you can see that the Strimzi Operator Pod is running in the Operators namespace

```
hshahin@DESKTOP-631K2UD:~/repos/my-repos/Federal-App-Innovation-Community/topics/kubernetes/whitepapers/container-adoption-journey$ kubectl get crds | grep kafka
kafkabridges.kafka.strimzi.io                         2022-04-22T15:13:18Z
kafkaconnectors.kafka.strimzi.io                      2022-04-22T15:13:17Z
kafkaconnects.kafka.strimzi.io                        2022-04-22T15:13:17Z
kafkamirrormaker2s.kafka.strimzi.io                   2022-04-22T15:13:18Z
kafkamirrormakers.kafka.strimzi.io                    2022-04-22T15:13:18Z
kafkarebalances.kafka.strimzi.io                      2022-04-22T15:13:18Z
kafkas.kafka.strimzi.io                               2022-04-22T15:13:18Z
kafkatopics.kafka.strimzi.io                          2022-04-22T15:13:18Z
kafkausers.kafka.strimzi.io                           2022-04-22T15:13:18Z
hshahin@DESKTOP-631K2UD:~/repos/my-repos/Federal-App-Innovation-Community/topics/kubernetes/whitepapers/container-adoption-journey$
```

You also created the Custom Resource Definitions which have created Native Kubernetes Resources for deploying Kafka – the Strimzi Operator monitors for these resources

Example Showing the Power of Kubernetes API: Operators

On the [Strimzi Quick Starts Page](), you can find the commands to deploy a Kafka Cluster – follow steps here

```
kubectl create namespace kafka
```

Provision the Apache Kafka cluster

After that we feed Strimzi with a simple **Custom Resource**, which will then give you a small persistent Apache Kafka Cluster with one node each for Apache Zookeeper and Apache Kafka:

```
# Apply the `Kafka` Cluster CR file
kubectl apply -f https://strimzi.io/examples/latest/kafka/kafka-persistent-single.yaml -n kafka
```

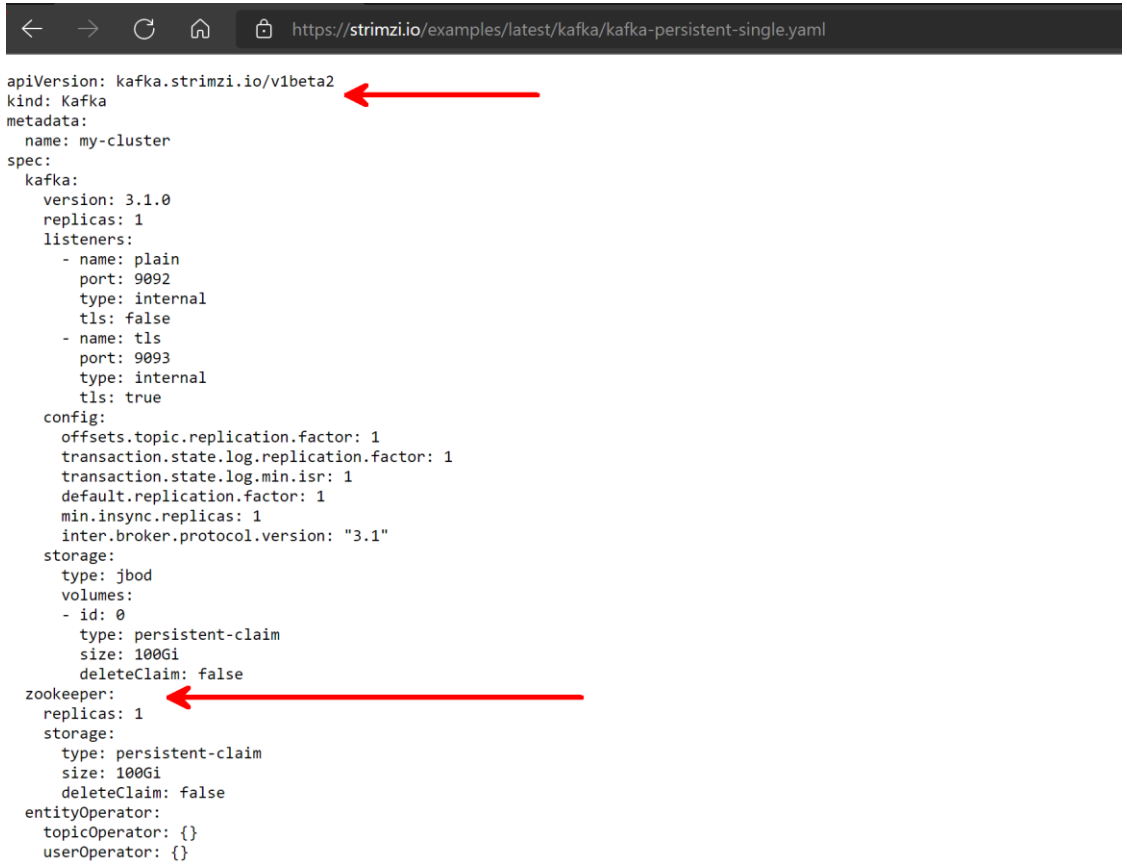
Example Showing the Power of Kubernetes API: Operators

```
hshahin@DESKTOP-631K2UD:~/repos/my-repos/Federal-App-Innovation-Community/topics/kubernetes/whitepapers/container-adoption-journey$ kubectl create ns kafka
namespace/kafka created
hshahin@DESKTOP-631K2UD:~/repos/my-repos/Federal-App-Innovation-Community/topics/kubernetes/whitepapers/container-adoption-journey$ kubectl apply -f https://strimzi.io/examples/latest/ka
.yaml -n kafka
kafka.kafka.strimzi.io/my-cluster created
hshahin@DESKTOP-631K2UD:~/repos/my-repos/Federal-App-Innovation-Community/topics/kubernetes/whitepapers/container-adoption-journey$ kubectl get kafka -n kafka
NAME         DESIRED KAFKA REPLICAS   DESIRED ZK REPLICAS   READY   WARNINGS
my-cluster   1                        1
hshahin@DESKTOP-631K2UD:~/repos/my-repos/Federal-App-Innovation-Community/topics/kubernetes/whitepapers/container-adoption-journey$
```

```
hshahin@DESKTOP-631K2UD:~/repos/my-repos/Federal-App-Innovation-Community/topics/kubernetes/whitepapers/container-adoption-journey$ kubectl get pods -n kafka
\NAME                     READY   STATUS    RESTARTS   AGE
my-cluster-kafka-0        0/1     Pending   0          1s
my-cluster-zookeeper-0    1/1     Running   0          47s
hshahin@DESKTOP-631K2UD:~/repos/my-repos/Federal-App-Innovation-Community/topics/kubernetes/whitepapers/container-adoption-journey$ \
```

You have deployed an instance of Kafka through the Operator without having to code the native Kubernetes Resources yourself

Example Showing the Power of Kubernetes API: Operators



If you navigate to the Strimzi Example, notice how the YAML is using the Custom Kafka Resource

The Value of the Operator is that it translates this Kafka-Native resource to the Kubernetes-Native resources like Pods and Services to make this work

Takeaway: Kubernetes Provides the Orchestration and API Required to Run Containers at Scale

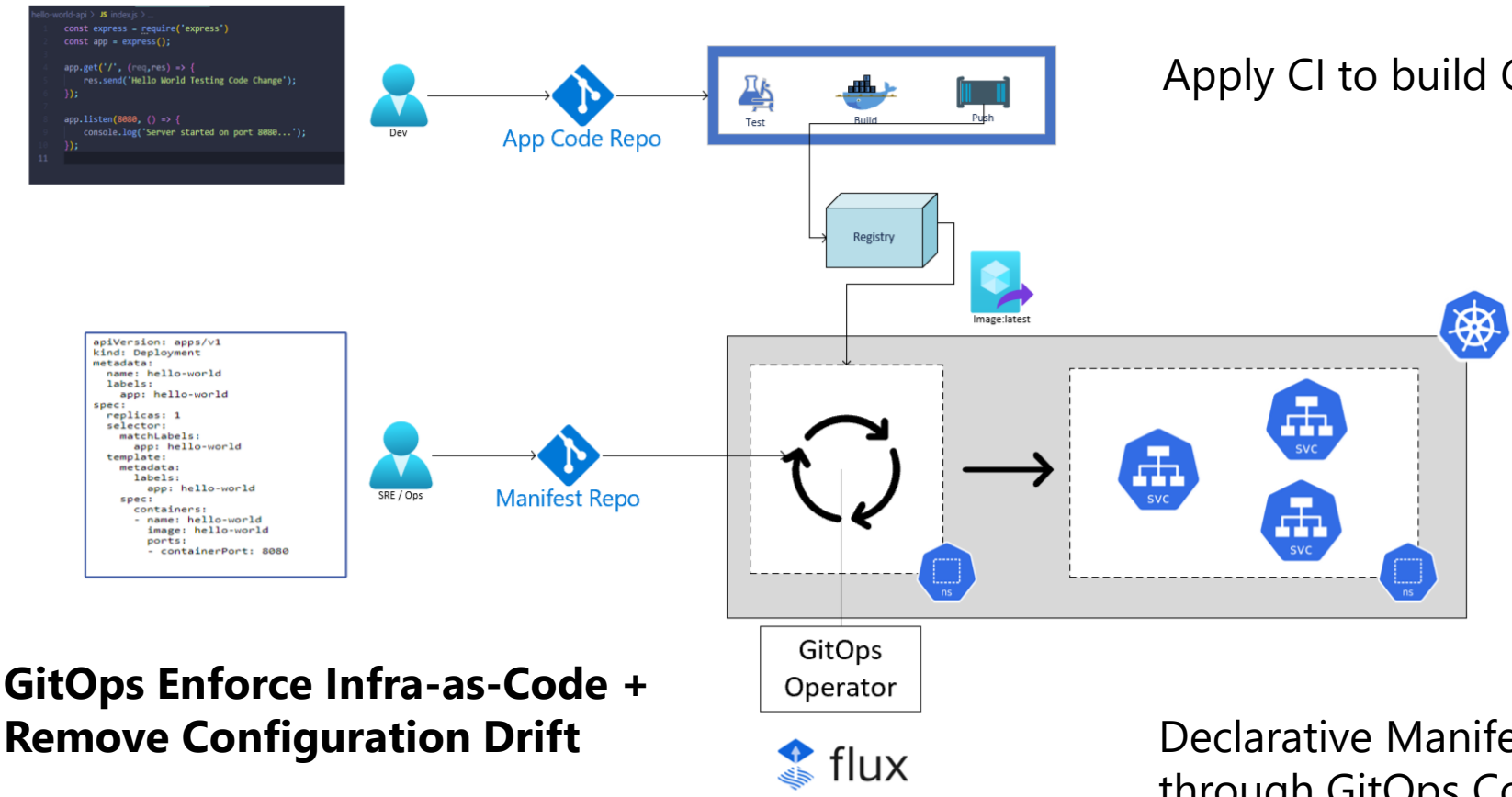
1. **Standard, Declarative, Portable, Extensible API** provides Orchestration Needs and Consistent Automation

2. **Dynamic Scaling for Optimized Resource Utilization** through cluster autoscaling + pod autoscaling

3. **Community Adoption** Accelerates Innovation and Creates Design Patterns

How do I use Kubernetes to Achieve the Desired Outcomes for Elite Software Delivery?

Automated CI/CD + GitOps

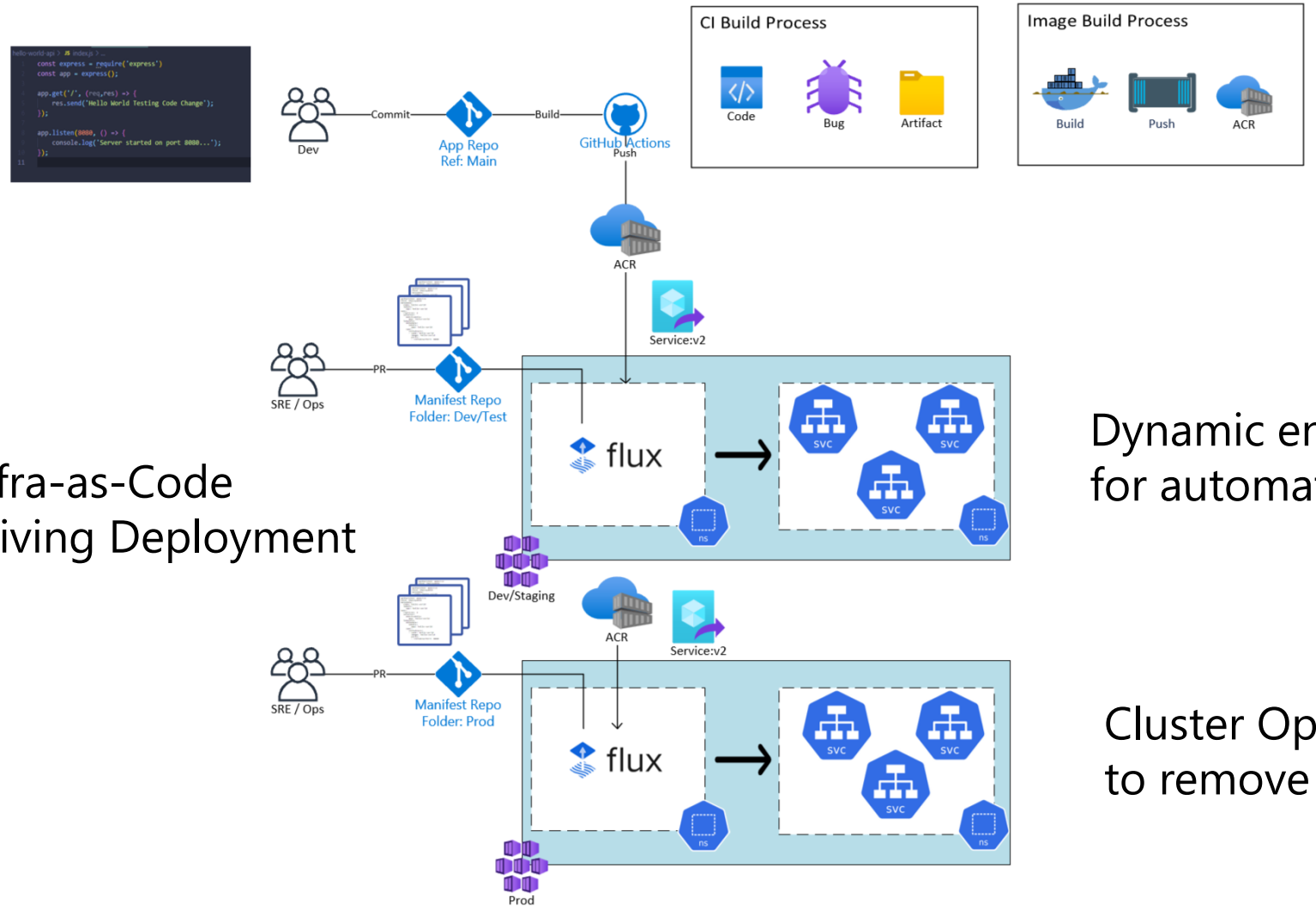


Apply CI to build Container Image

GitOps Enforce Infra-as-Code + Remove Configuration Drift

Declarative Manifests deployed to Cluster through GitOps Controller monitoring repo

Example: Applying GitOps across Environments



Developer Portability with Containers

Infra-as-Code driving Deployment

Dynamic environments for automated Testing

Cluster Operator Pulling Code to remove Config Drift

Demo: Deploy CI/CD with GitOps

Tutorial: Implement CI/CD with GitOps (Flux v2) - Azure Arc | Microsoft Docs

The following tutorial has both Azure DevOps and GitHub implementations for CI/CD + Flux for GitOps

You can ignore the “Azure Arc” component here – this tutorial can be run right on AKS

Takeaway: CI/CD + GitOps Enables Reliable & Secure Deployments and Cluster Operations

1. **Standard, Declarative, Portable, Extensible API** provides Orchestration Needs and Consistent Automation

2. **Dynamic Scaling for Optimized Resource Utilization** through cluster autoscaling + pod autoscaling

3. **Community Adoption** Accelerates Innovation and Creates Design Patterns

What are the Next Steps to Applying Kubernetes to Achieve the Desired Outcomes?

Next Steps for Driving Towards Desired Outcomes

1. **Identify Current Gaps** in Operations/Software Delivery and formulate metrics to quantity current performance

2. **Evaluate Containers and Kubernetes** concepts as ways to improve upon gaps and metrics

3. **Begin with a POC** to evaluate and document how container and Kubernetes enhances performance and delivery outcomes