



Application Migration Patterns

Re-host and Re-platform with
Tanzu

Yogi Rampuria
Advisory Platform Architect, MAPBU
July 2021

Agenda

Executive Summary

Scope and Reasons for Migration

Common Application Scenarios

Target Architecture

Migration Patterns

Q&A

Application Migration Patterns - Executive Summary

1. Application migration is key for customer's success
2. Every migration project has “Lift and Shift” strategy
3. Key challenges
 - File System
 - In memory state
 - IP addresses-based Service Discovery
 - Pre-Launch and Post-Launch tasks
4. Common techniques to address #3 and go cloud-native

Assumption

Backend Modules: We are talking about backend application/modules that run on server and are accessed over network. No Desktop or Mobile modules.

Minimum Portfolio Size: Application portfolio is of reasonable size. At least 100+ instances running (1 app - 100 instance or 10 apps with 10 instance each)

Business Application: Application is supporting a business. This is important as the migration has its own cost and unless there is an inherent business value associated with application, this would not fly

No/Low Code Change: We keep code change to a minimal. No code change is preferred as that triggers long QA cycles.

Kubernetes/Container: Our target platform is Kubernetes. Moving from VM to container

Why are we talking about this?

Every enterprise is thinking and talking about it

Organizations want to innovate faster and get the operational benefits of containers

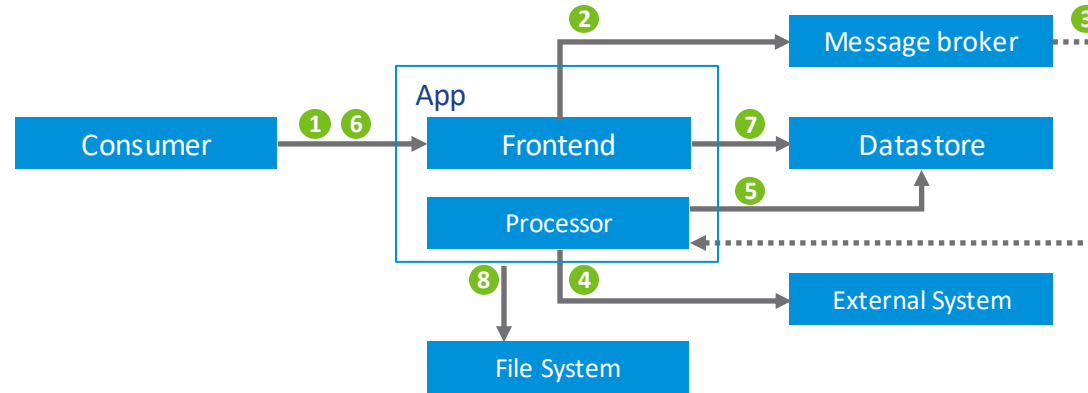
Keeping risk to minimal is a given

Reduce operational risk (complexity, cost, error, reliability, etc.)

About 40% of the workload is still in the Lift and Shift space.

An Enterprise Application

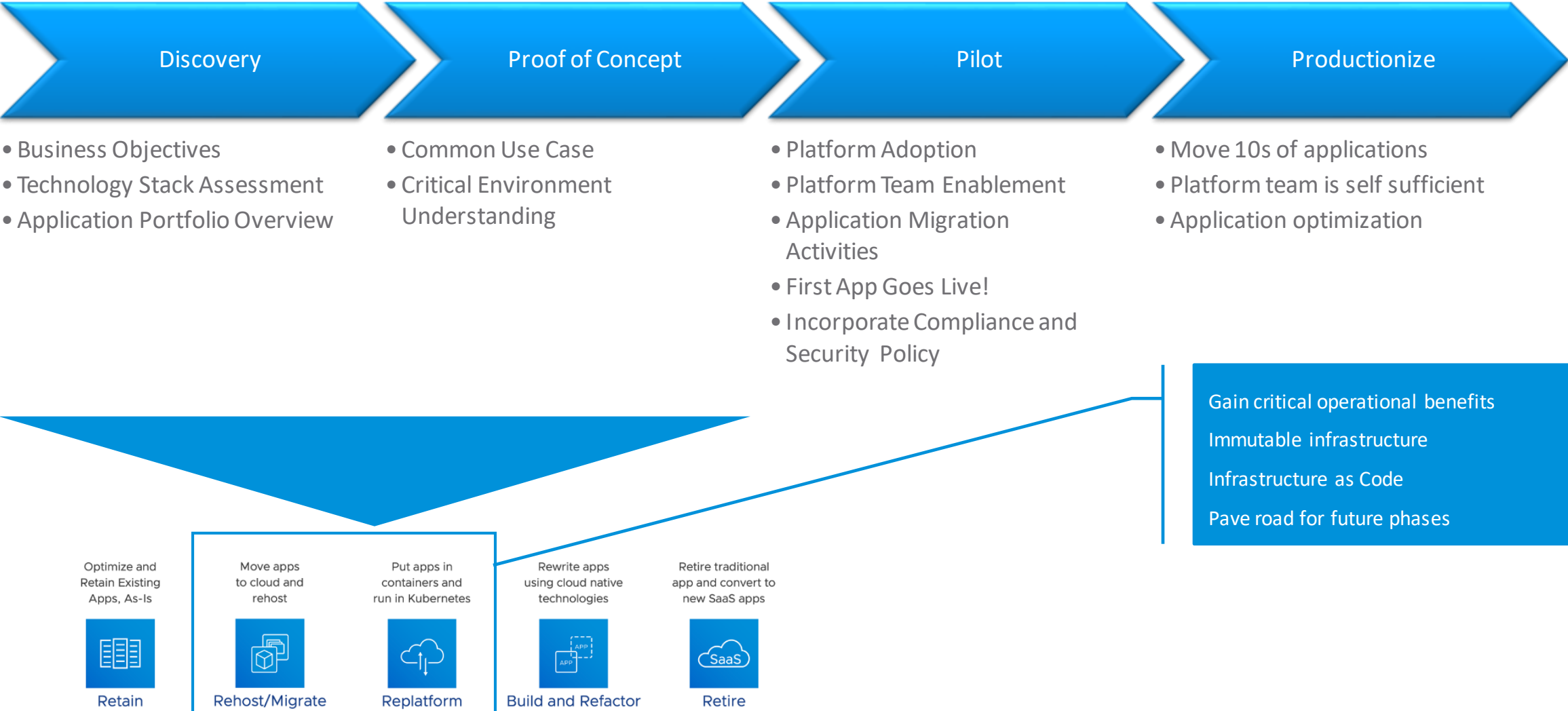
Component interaction



1. Consumer (User/System) submits a request
2. Frontend module queues request via a messaging system, and sends queuing response back
3. Processor module receives request asynchronously
4. Processor module connects to an external system and processes request
5. Processor stores processed information (response) in database
6. Consumer make a new request to check status
7. Frontend check database and responds to consumer appropriately
8. App uses filesystem for config, data and logging

Modernization Journey

High Level View of an Organization's Modernization Journey



Most Common Scenarios

Java / Java EE / J2EE Application

Runs on Linux Happily


Common in Enterprises

EJBs are the most complex

Spring Boot apps are a breeze

Web Apps, API, SOAP service,
Integrations, etc.

WARs can run on Tomcat

EARs need containerizing App
Server 

.Net Application

Often are windows based

Should be able to run with “dotnet
core” on Linux

Web Apps, API, SOAP service,
Extensions to Microsoft tools, etc.

Native Application

C/C++

Not first movers

Rarely used for Web applications

Batch jobs (NAV Calculations)

Cloud Native Principals

12 Factor Apps

Codebase: One codebase tracked in revision control, many deploys

Dependencies: Explicitly declare and isolate dependencies

Configuration: Store configuration in the environment

Backing Services: Treat backing services as attached resources

Build, release, run: Strictly separate build and run stages

Processes: Execute the app as one or more stateless processes

Port binding: Export services via port binding

Concurrency: Scale out via the process model

Disposability: Maximize robustness with fast start-up and graceful shutdown

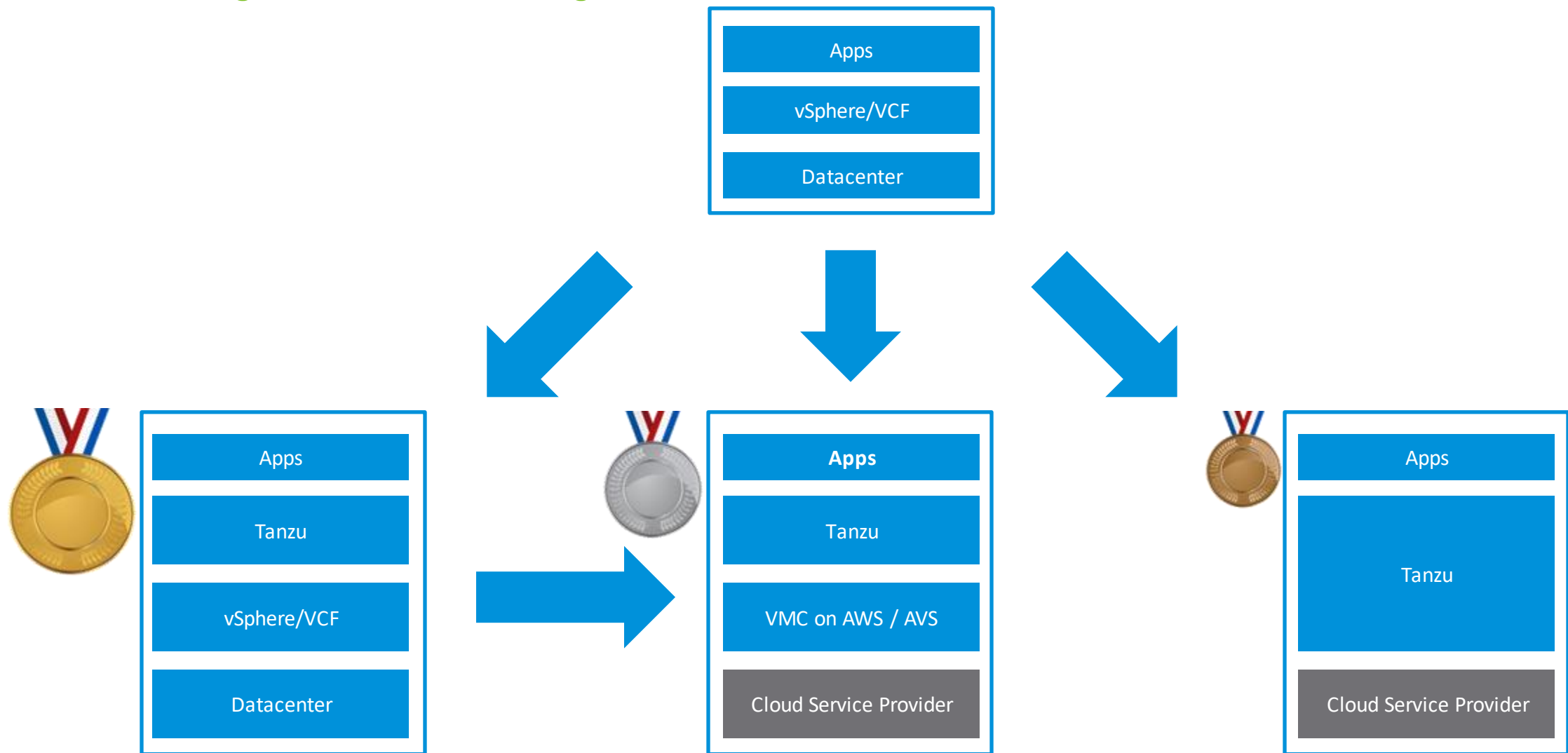
Dev/prod parity: Keep development, staging, and production as similar as 1. possible

Logs: Treat logs as event streams

Admin processes: Run admin/management tasks as one-off processes

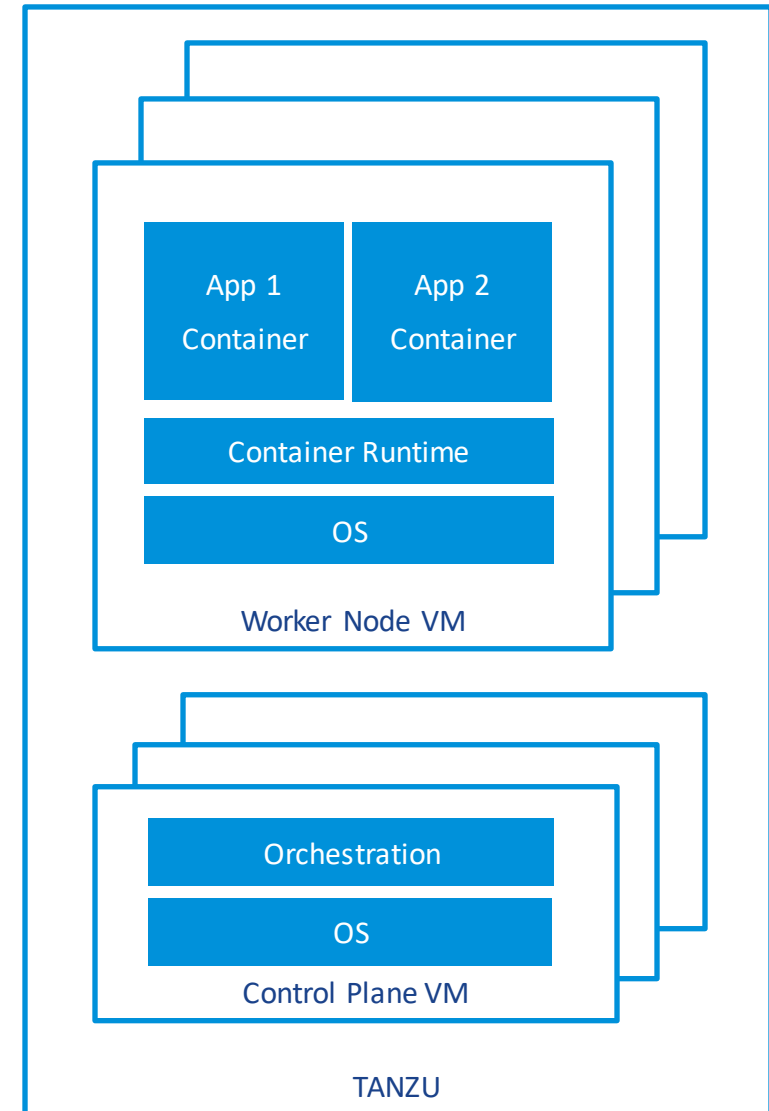
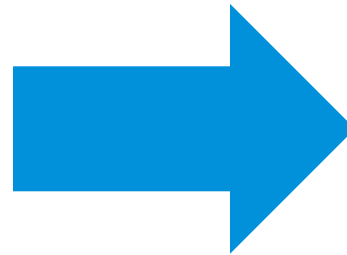
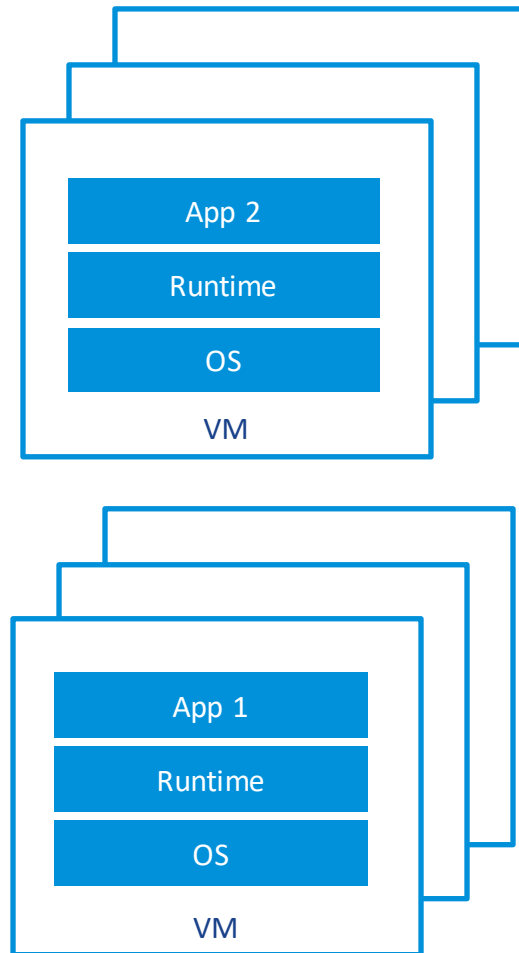
Migration Paths and Options

In the order of “degree/amount” of change



Target Runtime Environment

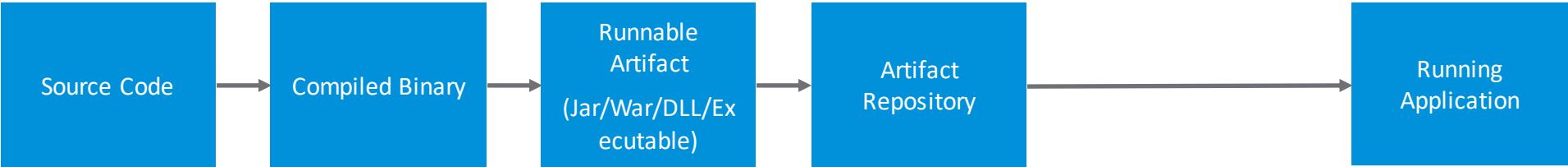
How would your application deployments look like?



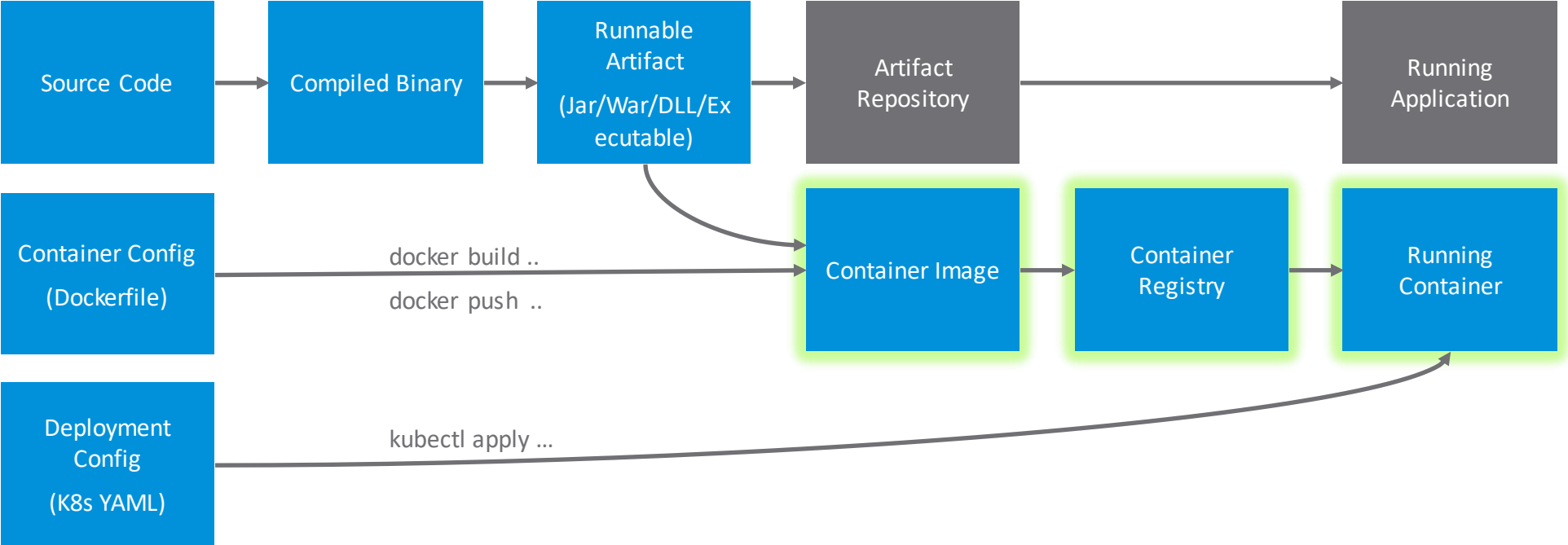
Delivery Flow

Changes in the delivery flow to accommodate containerization and Kubernetes

Current State



Target State



Container Configuration - Dockerfile

Create only runtime container or build in container

```
1 FROM tomcat:9
2 ENV JAVA_OPTS "-Denvironment=production"
3 COPY myapp.war /usr/local/bin/webapps
```

```
1 FROM maven:3-openjdk-11 as build
2 RUN mkdir -p /tmp/app
3 WORKDIR /tmp/app
4 COPY . .
5 RUN mvn package
6
7 FROM tomcat:9 as run
8 ENV JAVA_OPTS "-Denvironment=production"
9 COPY --from=build myapp.war /usr/local/bin/webapps
```

`docker build -t registry.core.cna-demo.ga/samples/myapp:v1.0 .`

`docker push registry.core.cna-demo.ga/samples/myapp:v1.0`

Deployment Config – Kubernetes Yaml

Deployment, Service and Ingress Configurations

```
1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   labels:
5     app: myapp
6   name: myapp-deployment
7 spec:
8   replicas: 1
9   selector:
10    matchLabels:
11      app: myapp
12   strategy: {}
13   template:
14     metadata:
15       labels:
16         app: myapp
17     spec:
18       containers:
19       - image: registry.core.cna-demo.ga/samples/myapp:v1.0
20         name: myapp
21         resources:
22           requests:
23             memory: "64Mi"
24             cpu: "250m"
25           limits:
26             memory: "128Mi"
27             cpu: "500m"
28         ports:
29         - containerPort: 8080
30           name: http
```

```
1 ---
2 kind: Service
3 apiVersion: v1
4 metadata:
5   name: myapp-service
6   labels:
7     app: myapp
8 spec:
9   selector:
10     app: myapp
11   type: ClusterIP
12   ports:
13   - name: http
14     port: 80
15     targetPort: http
```

```
1 ---
2 apiVersion: networking.k8s.io/v1
3 kind: Ingress
4 metadata:
5   name: myapp-ingress
6   labels:
7     name: myapp
8 spec:
9   rules:
10   - host: myapp.apps.cna-demo.ga
11     http:
12       paths:
13       - pathType: Prefix
14         path: "/"
15         backend:
16           service:
17             name: myapp
18             port:
19             name: http
```

kubectl apply -f deployment.yaml -f service.yaml -f ingress.yaml

Migration Patterns

File System Dependency - Configuration

Static and Dynamic (Generated) Config

ConfigMap and Secrets

```
apiVersion: v1
data:
  special.how: very
kind: ConfigMap
metadata:
  creationTimestamp: null
  name: special-config
---
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
  - name: test-container
    image: k8s.gcr.io/busybox
    command: [ "/bin/sh", "-c", "env" ]
    env:
      # Define the environment variable
      - name: SPECIAL_LEVEL_KEY
        valueFrom:
          configMapKeyRef:
            # The ConfigMap containing the value you want to assign to SPECIAL_LEVEL_KEY
            name: special-config
            # Specify the key associated with the value
            key: special.how
  restartPolicy: Never
```

Init Container

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox:1.28
    command: ['sh', '-c', 'echo The app is running! && sleep 3600']
  initContainers:
  - name: init-myservice
    image: busybox:1.28
    command:
      - sh
      - -c
      - until nslookup myservice.default.svc.cluster.local; do echo waiting for myservice; sleep 2; done
  - name: init-mysql
    image: busybox:1.28
    command:
      - sh
      - -c
      - until nslookup mydb.default.svc.cluster.local; do echo waiting for mydb; sleep 2; done
```


File System Dependency - Logs

Sidecar Pod and Streaming Logs

Sidecar Example

```
apiVersion: v1
kind: Pod
metadata:
  name: two-containers
spec:
  restartPolicy: Never

  volumes:
  - name: shared-data
    emptyDir: {}

  containers:
  - name: nginx-container
    image: nginx
    volumeMounts:
    - name: shared-data
      mountPath: /usr/share/nginx/html

  - name: debian-container
    image: debian
    volumeMounts:
    - name: shared-data
      mountPath: /pod-data
    command: ["/bin/sh"]
    args: ["-c", "echo Hello from the debian container > /pod-data/index.html"]
```

Stream to Standard Streams

```
apiVersion: v1
kind: Pod
metadata:
  name: two-containers
spec:
  restartPolicy: Never
  containers:
  - name: nginx-container
    image: nginx
    volumeMounts:
    - name: log-data
      mountPath: /var/log/nginx

  - name: debian-container
    image: debian
    volumeMounts:
    - name: log-data
      mountPath: /log-data
    command: ["/bin/sh"]
    args: ["tail", "-f", "/log-data/server.log"]
  volumes:
  - name: log-data
    emptyDir: {}
```

File System Dependency - Data

Shared Filesystem

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: vsan-file-sc
provisioner: csi.vsphere.vmware.com
parameters:
  storagepolicyname: "RAID1"
  csi.storage.k8s.io/fstype: nfs4
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: file-pvc
spec:
  storageClassName: vsan-file-sc
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 2Gi
```

Dedicated Storage per Instance

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: vsan-block-sc
provisioner: csi.vsphere.vmware.com
parameters:
  storagepolicyname: "RAID1"
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: block-pvc
spec:
  storageClassName: vsan-block-sc
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
```

In Memory State - Session

StatefulSet + Sticky Sessions

```
apiVersion: projectcontour.io/v1
kind: HTTPProxy
metadata:
  name: httpbin
  namespace: default
spec:
  virtualhost:
    fqdn: httpbin.davecheney.com
  routes:
  - services:
    - name: httpbin
      port: 8080
    loadBalancerPolicy:
      strategy: Cookie
```

Datastore backed Session Replication

This need some changes in the in each app.

- In web.xml add <distributable/>
- All session objects should be serializable

Example: <https://tomcat.apache.org/tomcat-8.0-doc/cluster-howto.html>

Service Discovery

Kubernetes DNS Service Discovery

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox:1.28
    command: ['sh', '-c', 'echo The app is running! && sleep 3600']
  initContainers:
  - name: init-myservice
    image: busybox:1.28
    command:
    - sh
    - -c
    - until nslookup myservice.default.svc.cluster.local; do echo waiting for myservice; sleep 2; done
  - name: init-mydb
    image: busybox:1.28
    command:
    - sh
    - -c
    - until nslookup mydb.default.svc.cluster.local; do echo waiting for mydb; sleep 2; done
```

ExternalName for Proxy

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
  namespace: prod
spec:
  type: ExternalName
  externalName: my.database.example.com
```

But...

Watchout for these...

Gotchas

“Hard”ware coupling

Licensing

- Not all vendor have container friendly license.
- Vendors support only their own Kubernetes distribution.
 - JBoss Middleware

Latency

- Small amount of latency (< 5ms) could be introduced based on network layers.
- Could be more if the target is a cloud environment. Think RTT from your DC to CSP DC.

Security and Compliance

- May become a make-or-break case
- Engage security team early
- Need educating the security team
- Need to propose risk mitigating controls for any gaps



Thank You