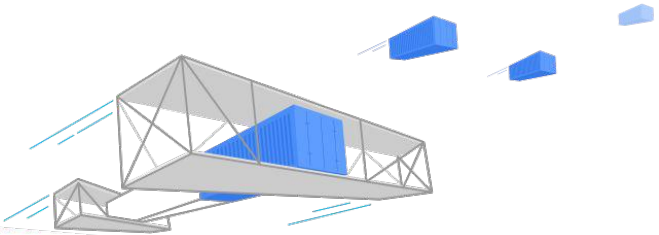
hyscale

Simplifying App Migration to Kubernetes

with an App-centric Abstraction



Member of

CLOUD NATIVE COMPUTING FOUNDATION

Agenda

Microservices

Migration to k8s

Simplified migration

Self-service deployment

Abstractions for k8s

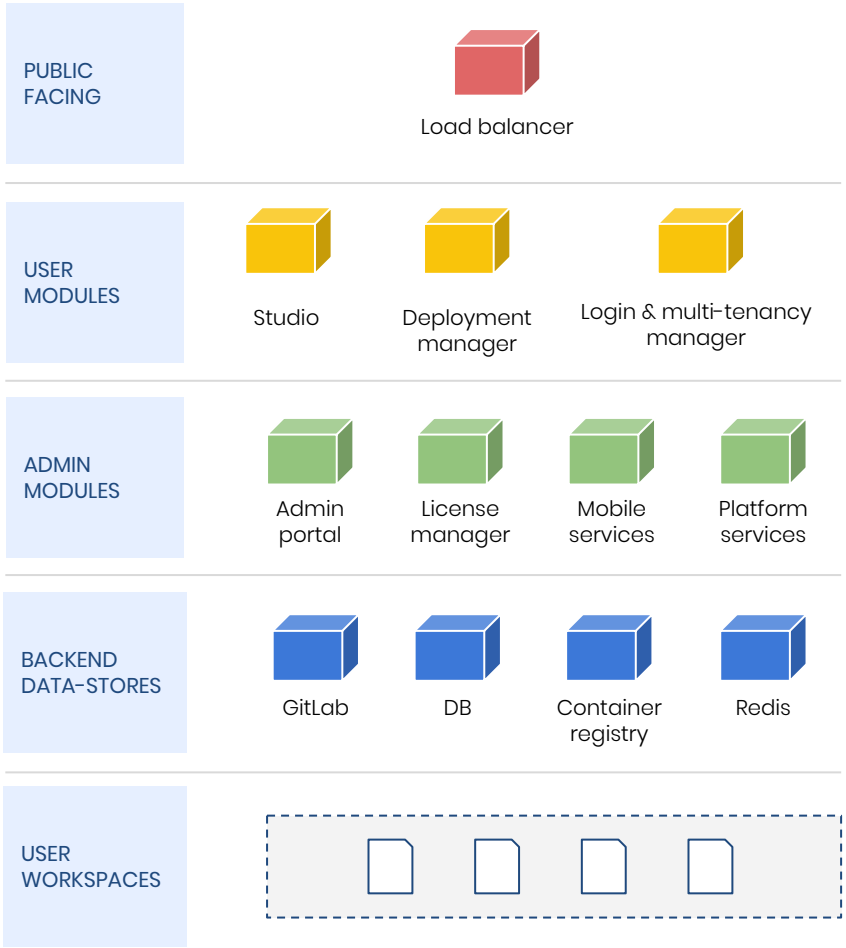
---

1. Getting ready - The migration goals, considerations and challenges
2. Roadblocks - Technical barriers & learnings
3. Way forward - Simplifying migration & building abstraction

The App's architecture

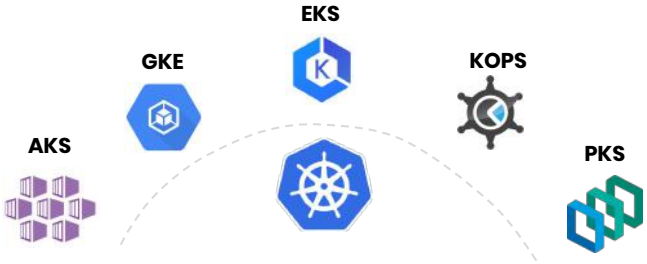
Enterprise Platform for
low-code app development

- Web-app
- Microservices architecture
- Multi-tenant
- Scalable
- Production level



Why the platform wanted K8s?

1. **Customers wanting setups on various clouds/on-prem**
   (Cloud-agnostic deployments)

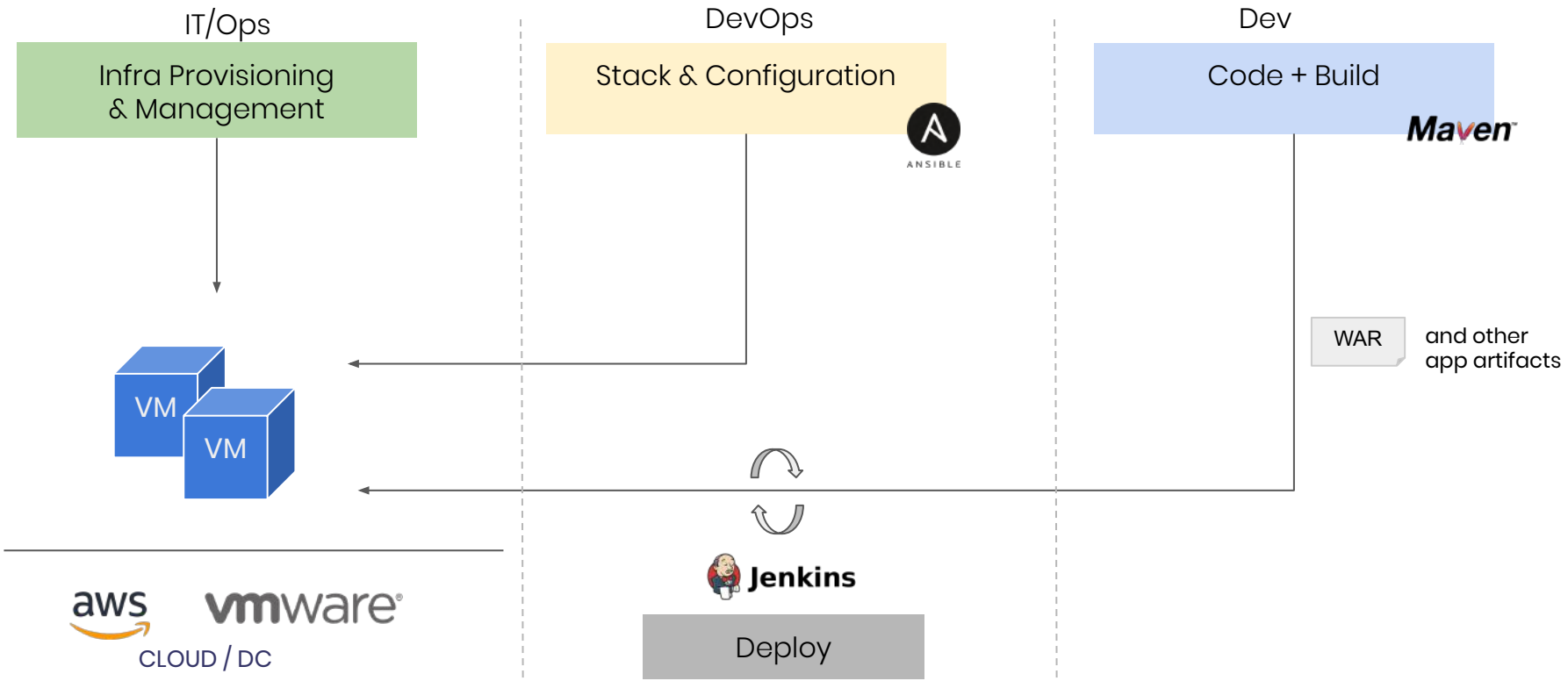


2. Scalability to **support volatile demand**

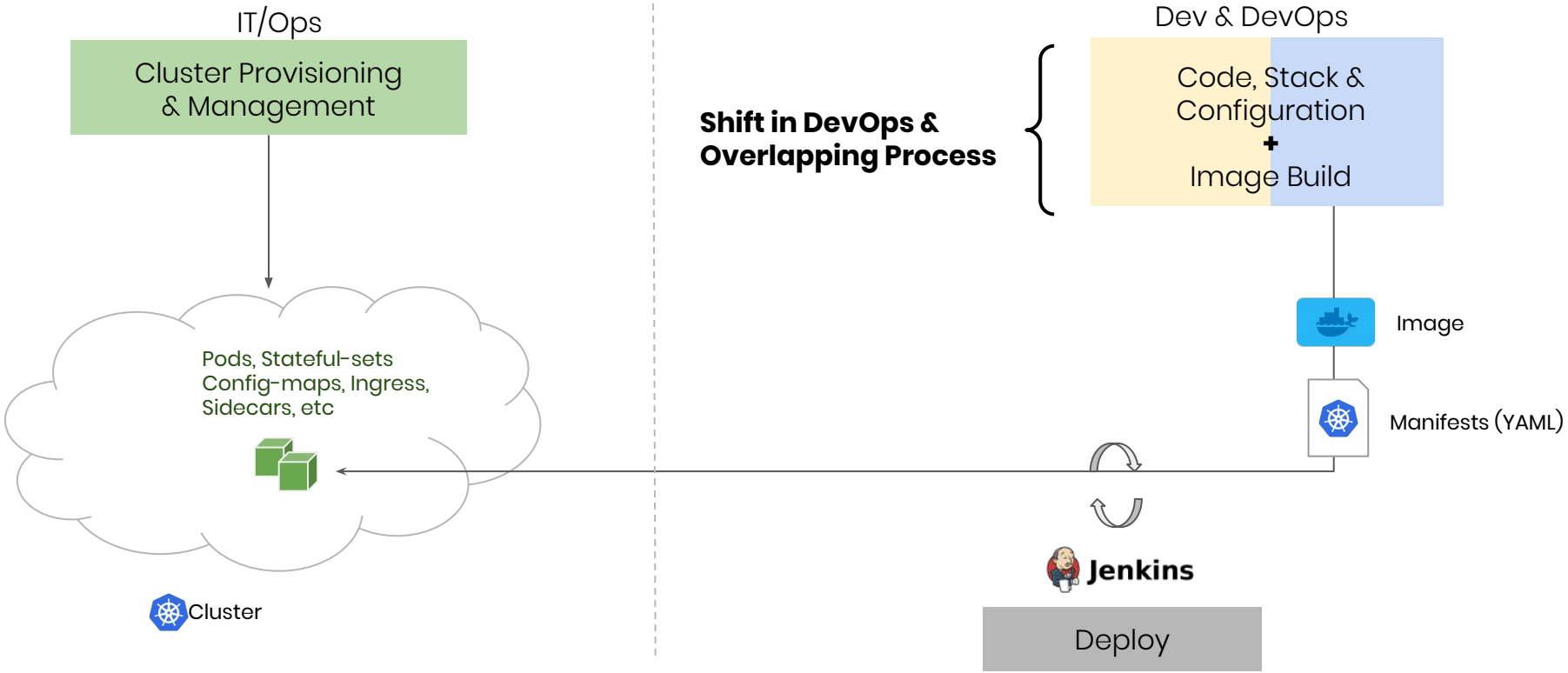
3. Ability to **add more tenants cost-effectively**

4. **More reliable delivery & upgrades** with declarative approach

Pre-K8s Scenario



With K8s



Considerations for migration

Shift in DevOps with containers & K8s

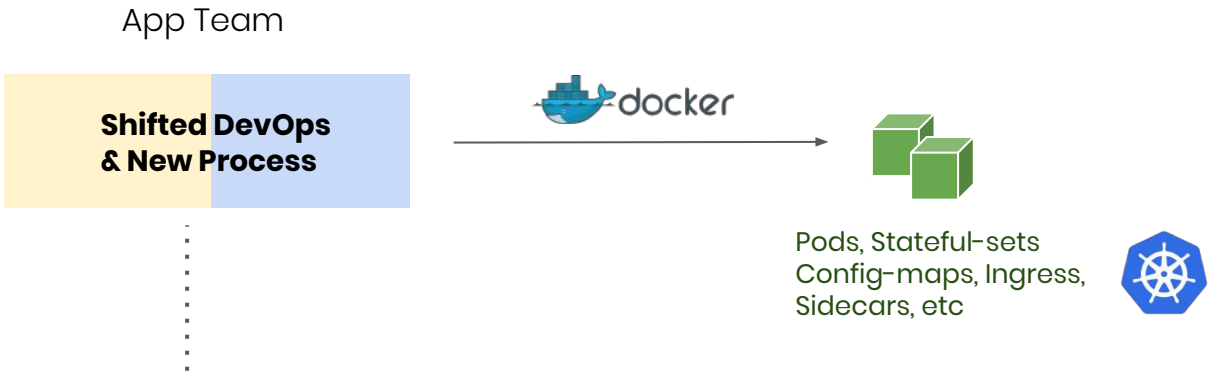
Immutability
Always replaced, never updated or modified

Frequency of Change
Most containers live less than a week

Change in Troubleshooting
For fix → rebuild & redeploy

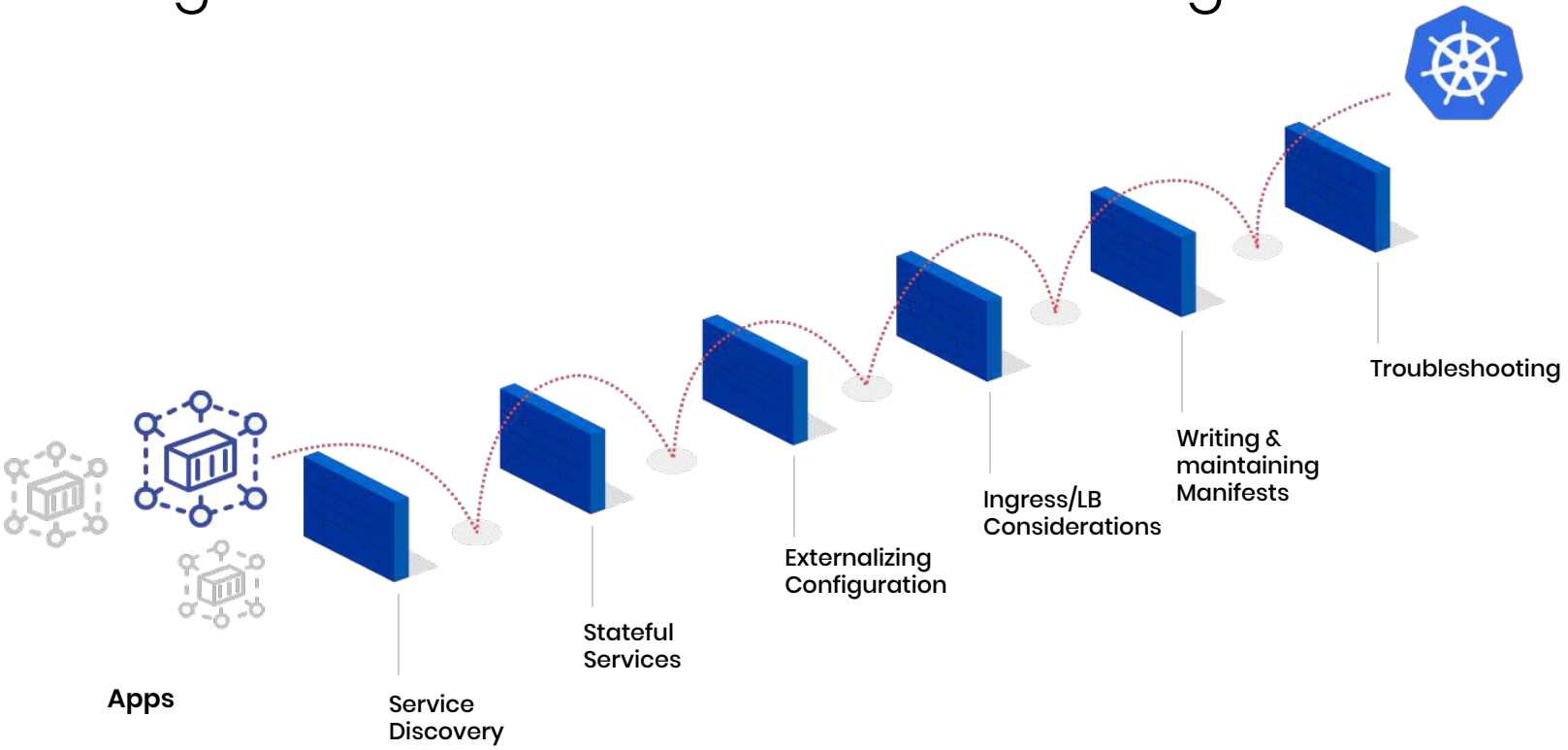
Role shifts
Complete change in process & mindset

New challenges



1. Change in deployment workflow
2. New terminologies & concepts in k8s
3. Differences in troubleshooting and ops

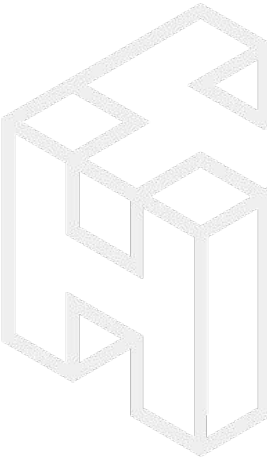
Getting to K8s - some barriers & learnings



1

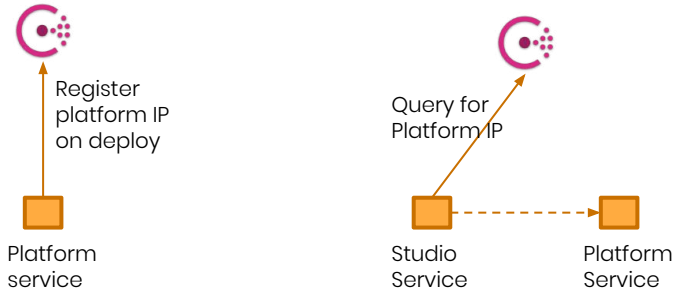
Service discovery

- Moving from traditional to K8s-native discovery
- Moving with and without code changes
- Difference between relying on service IP & pod IPs



To K8s Native Service Discovery

Pre-K8s

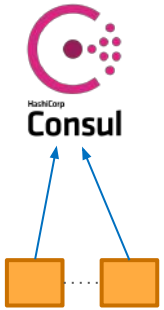


K8s Native Service Discovery

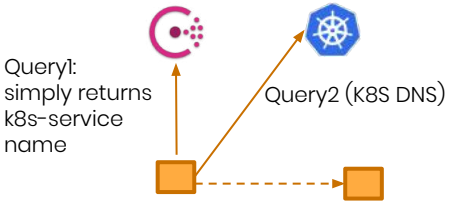
- Use K8s Service Discovery instead of Consul.
- In K8s, service IP registration is automatic!
- Service names return IPs automatically via DNS natively
- In case of replica pods, the K8s service IP automatically load balances across the pods (pod-IPs)

Service discovery journey

PRE-K8s

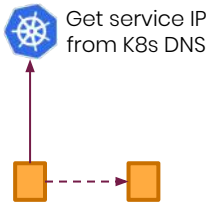


K8s attempt 1
avoid code-change



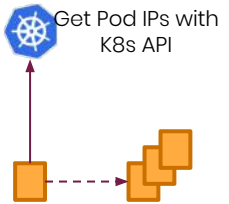
Two hops to get to the service!

K8s attempt 2
avoid 2 hops



- resolves even if no pods are up
- connection pooling issues
- limited to round-robin

K8s attempt 3
only get healthy pods



- need to cache pod IPs list
- Refresh periodically & invalidate

Newer Options for Service Discovery

1. Use a library like spring-cloud ribbon in code and query for pod IPs.

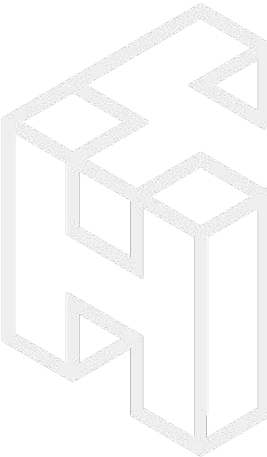
   OR

2. Use something like consul's k8s sync

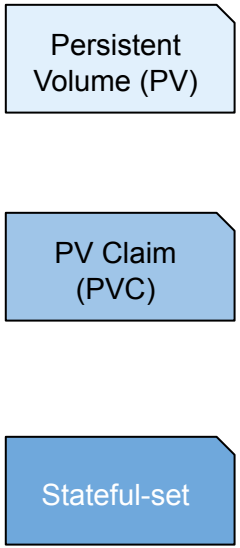
2

Stateful services & persistent volumes

- Different ways of provisioning volumes
- Proper wiring
- Resizing



To K8s Persistent Volumes



Represents a physical volume in K8s (eg. EBS volume)

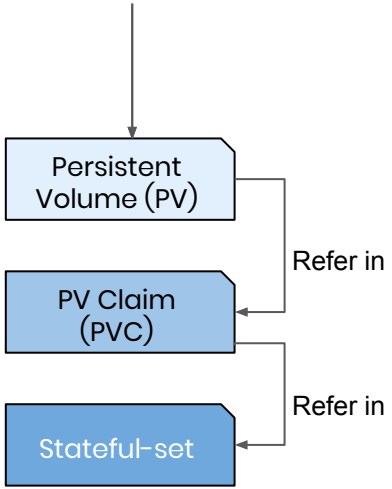
Specifies which PV to attach into the pod

Deploy services as a “Stateful-Set” so that volume attachments are properly persisted

Provisioning Volumes

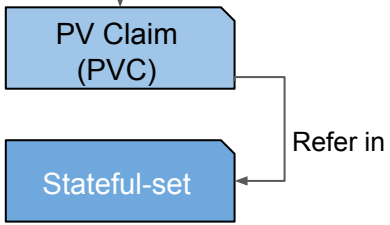
STATIC PROVISIONING

EBS Vol with Data **OR** EBS Snapshot



DYNAMIC PROVISIONING

Physical volumes are automatically provisioned by PVCs when deployed



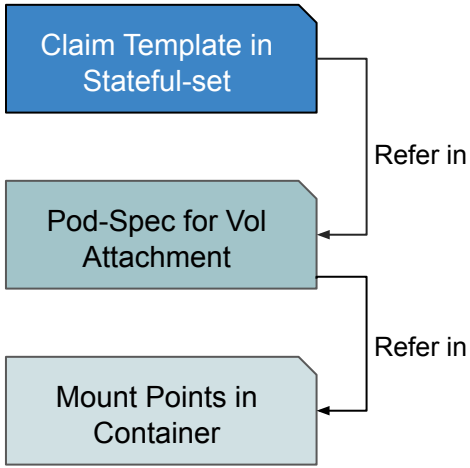
Handles volume provisioning for replicas

Everything in a single YAML

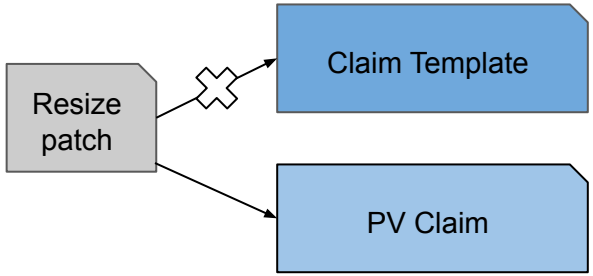
Stateful-set with Claim Template

More considerations for Volumes

- Ensure proper references that don't break on changes



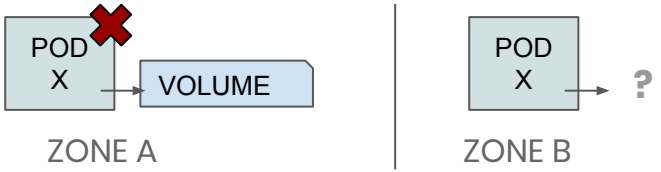
- Volume resizing considerations



Cannot apply K8s patch for resizing to templates

Query for underlying PVC & then patch as needed

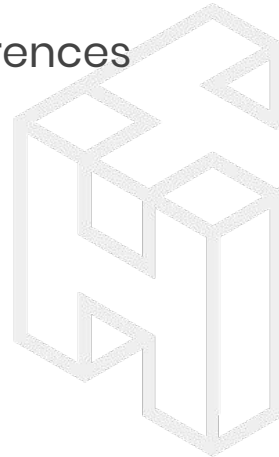
- Multi-zone challenges



3

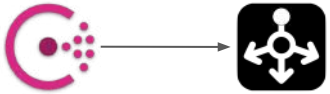
Load balancers & Ingress

- Moving to Ingress Controller, Ingress and Ingress Rules
- Configuring SSL, headers, rules, etc.
- K8s abstraction of LB vs Ingress provider differences



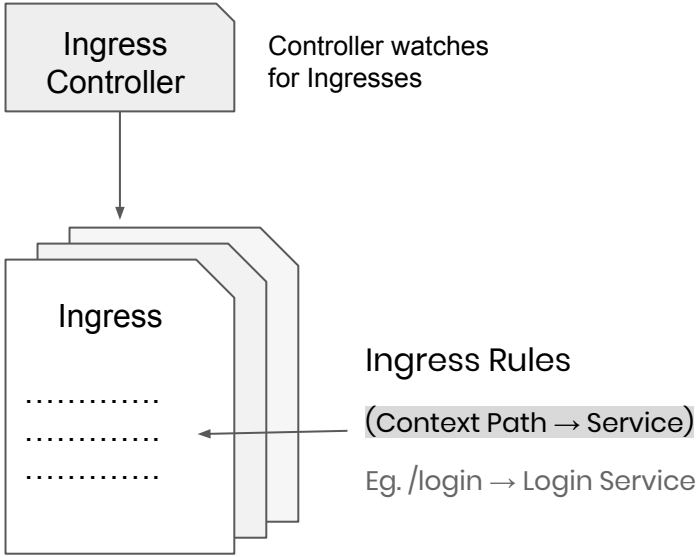
Moving from a Load balancer to K8s Ingress

PRE-K8s



Watch consul and register backend-nodes into LB

IN K8s



- Ingress control is not native, choose a provider
- DON'T: Aggregate context-path rules in a single ingress
- Cannot individually specify headers, etc.
- DO: Create a separate ingress per service
- SSL config & rotation is via K8s secrets referred within ingress rules

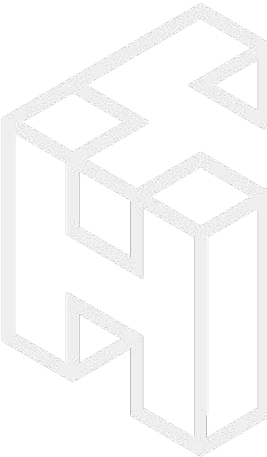
Ingress Considerations

1. Only context-path routing abstracted by K8s
2. Several LB configs (size-limits, time-outs, etc) are provider specific
3. Ingress regex different from provider supported regex
4. Restrict ingress controller to watch only relevant namespaces

4

Config profiles & templates

- Moving to config maps
- Static vs Dynamic Config considerations
- Use of templates and pros/cons

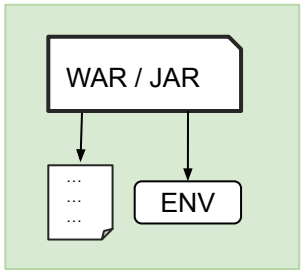


Config map considerations

PRE-K8s

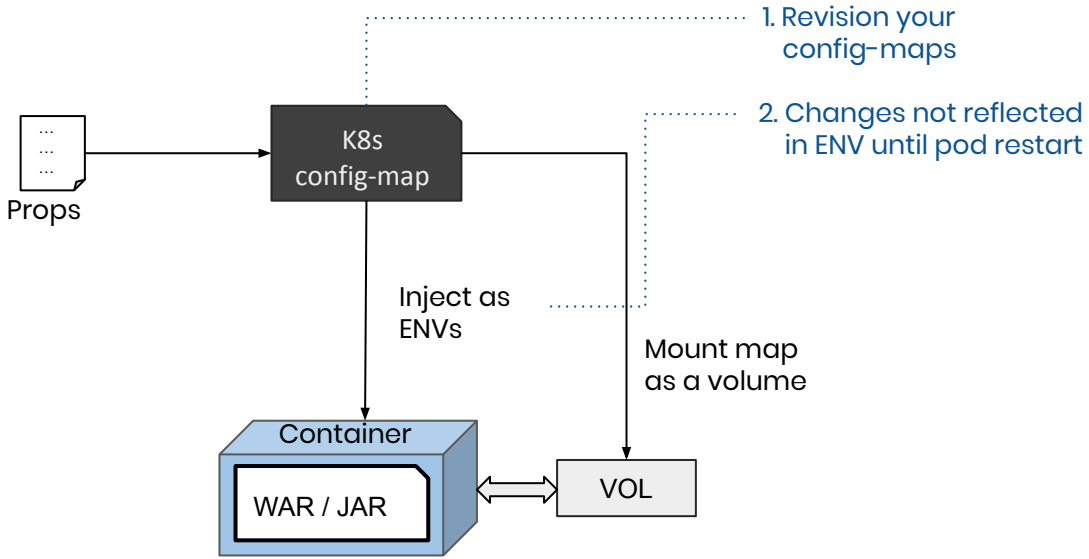
1.

App code / binary deployed together with the config props

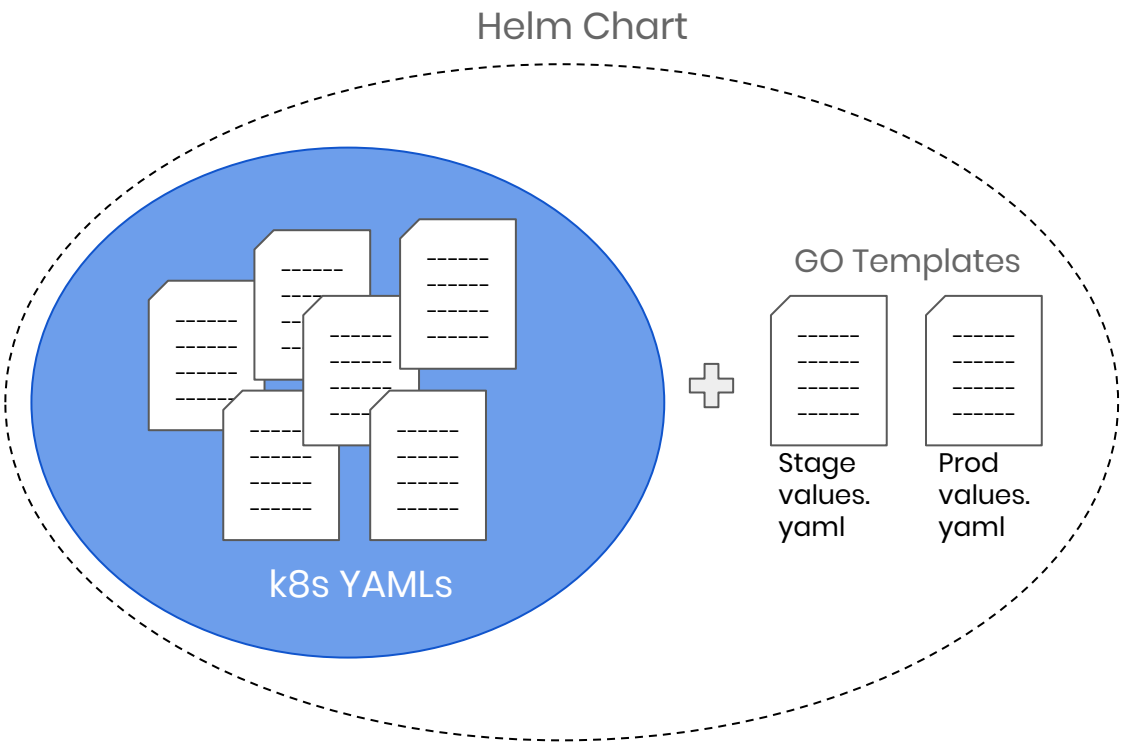


2. App reads props from local file or from ENVs

IN K8s



Templating & profiles



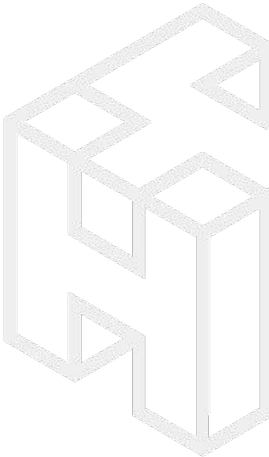
LEARNINGS

- Must understand k8s yamls as well as go-template language
- Choose appropriately
  - Debugging can be a challenge
  - Security considerations
- Resource grouping is lost after deploy
- More useful for OTS services, less for bespoke

5

Writing and managing manifests

- What goes in Image? What goes in YAMLs?
- Which configs/resources go into which “Kinds”
- Know K8s language, *everything* is in YAML
- Ensure proper resource binding



Challenges with troubleshooting

- serviceX→ getLogs()

1. Get deployment name of ServiceX using labels
2. Get pods of deployment
3. List containers in pod & identify container name
4. Get logs of each container in the list
5. Repeat for replicas of the pod

Consider sidecar agents & log-aggregation as a *must*

- Get pod name, container name & exec into the pod

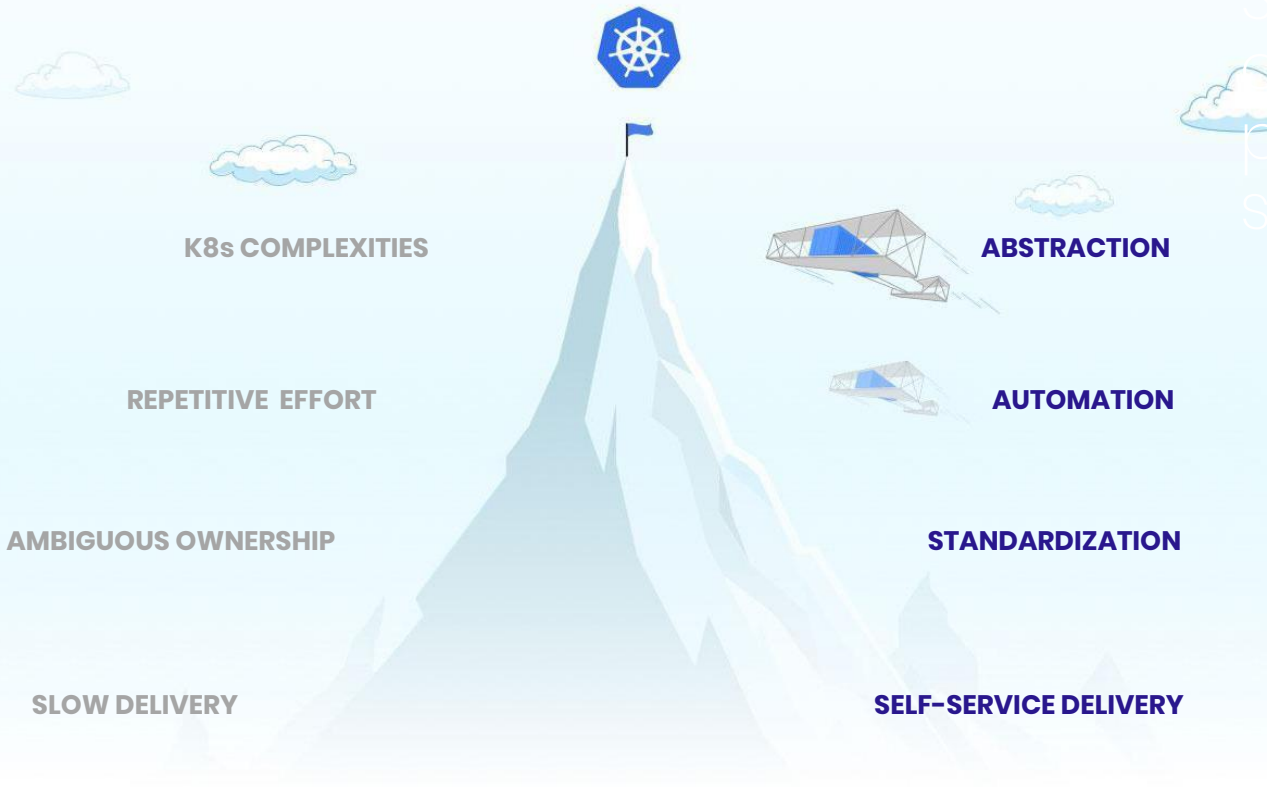
APPLY CAUTION! Consider debugging agents & app observability

- Understand K8s error messages

CrashLoopBackOff, ImagePullBackOff, RunContainerErr, OOMKilled, etc.

Taking a Step Back:

Mapping some of the difficulties faced



Finding Solutions

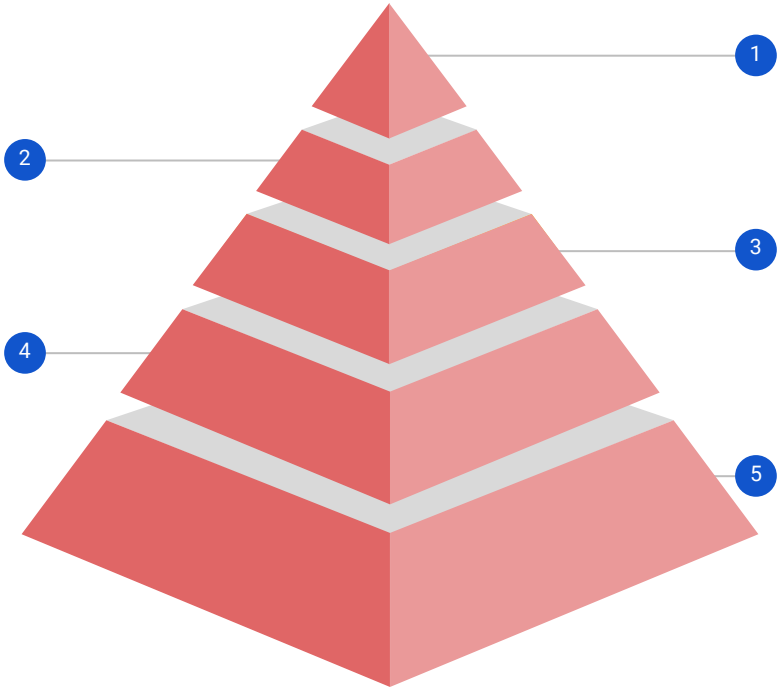
App-centric Abstraction

App	Profile	ConfigProps	Agents
Service	Volumes	Secrets	Jobs
Dependency	HealthChecks	Memory/CPU	LoadBalancer
Environment	Ports	Replica	Initialization / Finalization

Kubernetes Complexity (concepts)

pods	hostAlias	accessModes	targetPort
statefulSet	hostname	dataSource	sessionAffinity
replicaSet	imagePullSecrets	storageClassName	healthCheckNodePort
deployments	initContainers	CronJob	Ingress
daemonSet	nodeSelector	concurrencyPolicy	IngressController
sidecar	restartPolicy	failedJobsHistoryLimit	annotations
configMap	preemptionPolicy	successfulJobsHistoryLimit	labels
imagePullPolicy	serviceAccount	schedule	endpoints
lifecycleHooks	tolerations	suspend	endpointSlices
livenessProbe	terminationGracePeriod	jobTemplate	targetRef
readinessProbe	replicas	backoffLimit	topology
startupProbe	revisionHistoryLimit	completions	persistentVolumeClaim
resources	minReadySeconds	parallelism	horizontalPodAutoscaler
resourceTypes	progressDeadlineSeconds	clusterIP	metricSpec
limits	selector	loadBalancerIP	scaleTargetRef
securityContext	rollingUpdate	externalIPs	networkPolicy
capabilities	podManagementPolicy	externalName	egress
volumeMounts	updateStrategy	externalTrafficPolicy	ingress
affinity	volumeClaimTemplates	loadBalancerSourceRanges	podSelector
dnsConfig	volumeMode	nodePort	policyTypes

How did we abstract K8s



1. **App-centric Keywords**
   Keywords & concepts already understood by an app team
2. **Validations & Inferences**
   Validate inputs to catch issues early. Infer the required k8s kinds
3. **Translation**
   Translate app requirements into k8s yamls
4. **Bindings**
   Bind different yamls & snippets together with the right labels
5. **Troubleshooting**
   Execute a troubleshooting flowchart to provide easier debugging

App-centric abstraction (“hspec”)

```
volumes:
        - name: tomcat-logs
          path: /usr/local/tomcat/logs
          size: 1Gi
```

```
environment: production
overrides: my-service

replicas:
        min: 1
        max: 4
        cpuThreshold: 30%
```

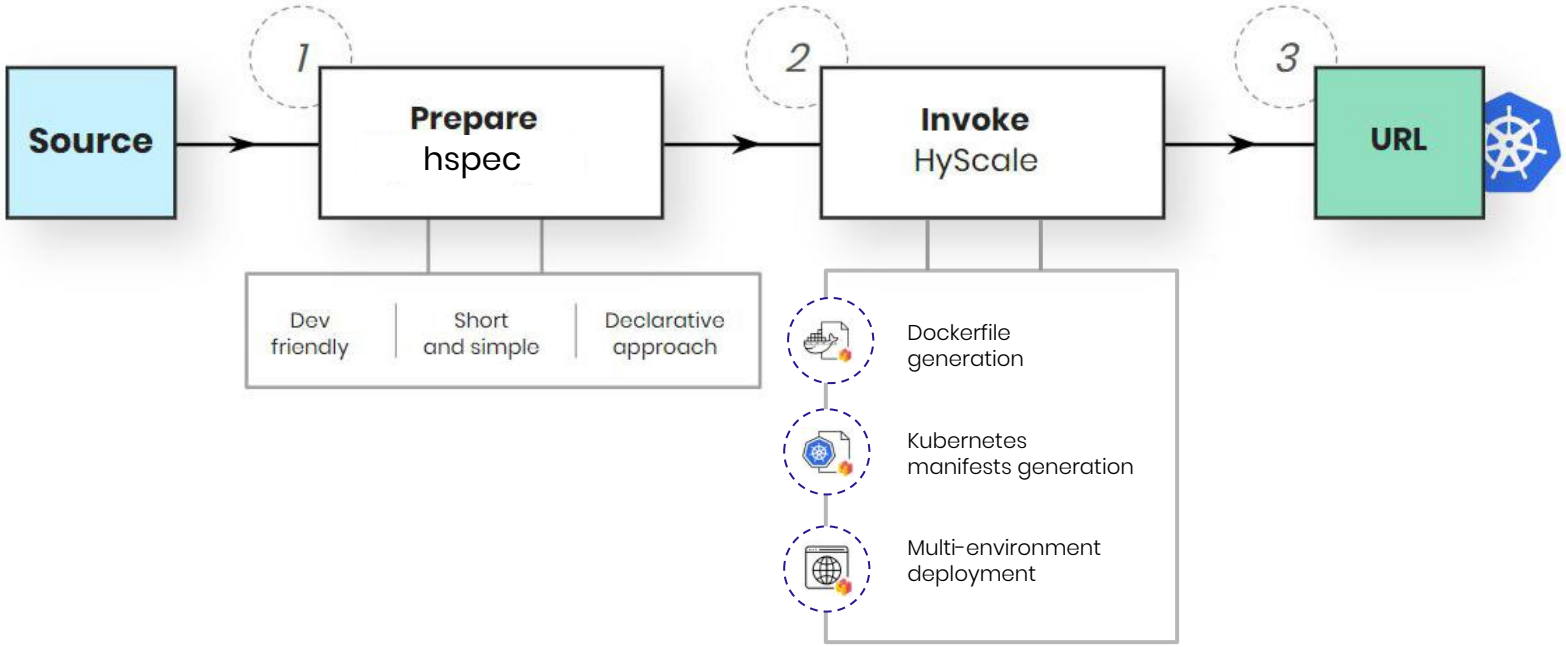
```
props:
      JAVA_HOME: /usr/local/java
      TOMCAT_HOME: /usr/local/tomcat
secrets:
     - MYSQL_PASSWORD
```

```
ports:
      - port: 8080/tcp
        healthCheck:
            httpPath: /docs/images/tomcat.gif
```

github.com/hyscale/**hspec**

#app2k8s

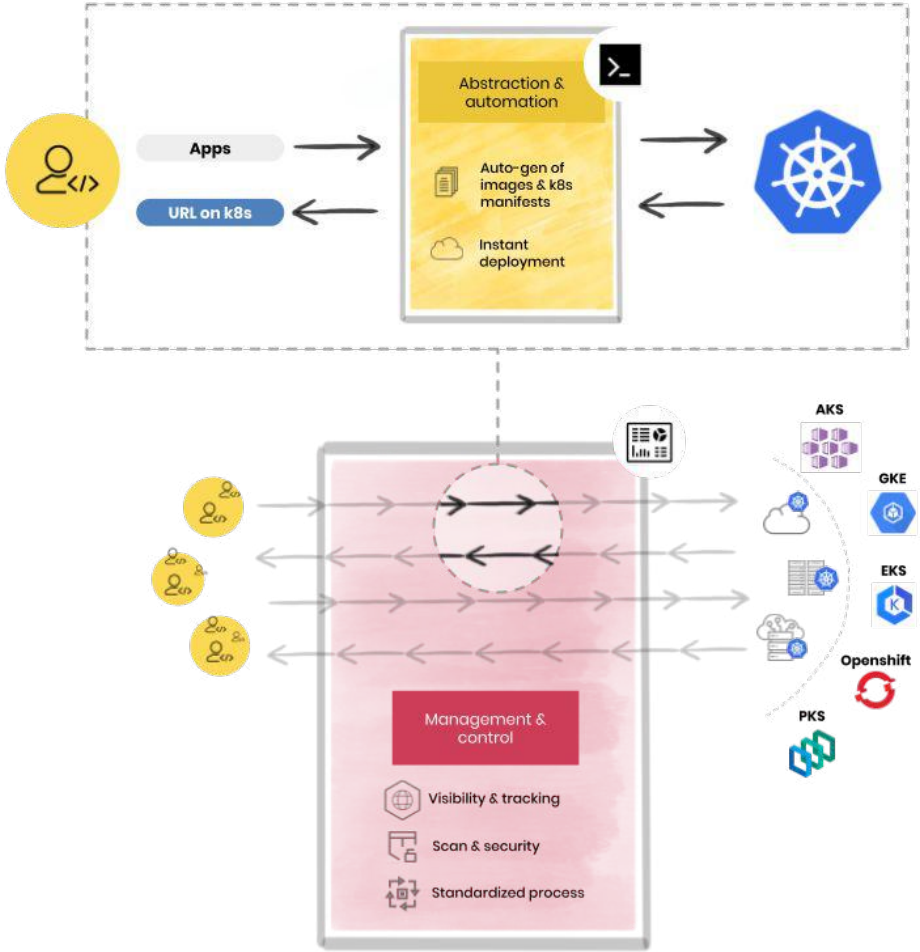
github.com/hyscale/**hyscale**



Troubleshooting Abstraction

k8s error message	More meaningful message
CrashLoopBackOff	Application found crashing , Refer to logs at <path>
	Error in the start Commands
	Health check failing or incorrect health check specified
ImagePullBackOff	Incorrect docker registry credentials
	Incorrect image name
	Incorrect image tag
Pending	Not enough space in cluster
	Unable to bind volume to the service. Contact k8s admin
Running 0/1	Fix healthcheck, service should listen on 0.0.0.0
	Application found crashing , Refer to logs at <path>

Building in Layers



The Outcome

APP DEV TEAMS

Considerations	Manual approach	Our findings
Ease of delivery for app-teams	Low	High
Container and Kubernetes learning curve	High	Low

IT / DEVOPS

Considerations	Manual approach	Our findings
Dependency on custom scripts and tools	High	Low
Delivery friction and delays	High	Low

90%
Less time for new environment setup

Hours →Mins
Upgrade time

6X
Reduction in repetitive effort

60%
Reduction in infra required

Try HyScale

Get Involved

Star Us on github!

---

https://github.com/hyscale/hyscale

www.hyscale.io

https://twitter.com/hyscaleio

connect@hyscale.io