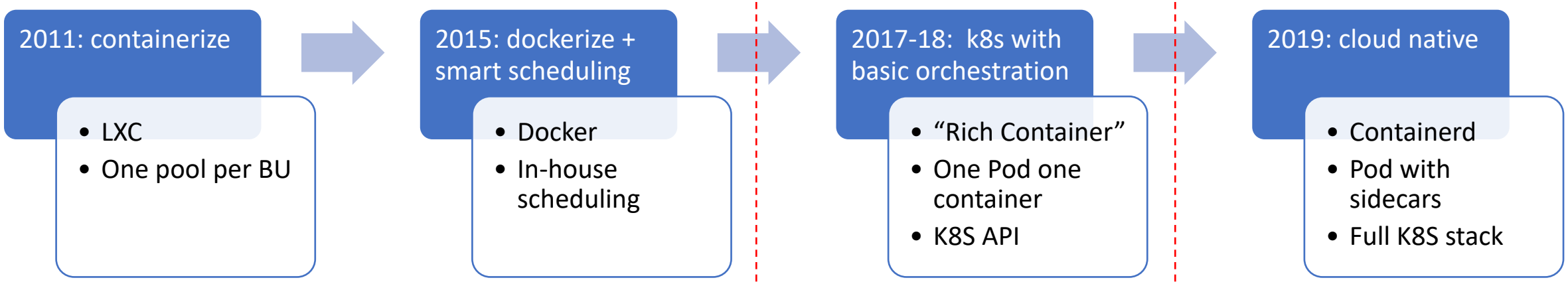
Some Lessons We Learned from Moving E-business Giant to Cloud Native

Lei Zhang, Siyu Wang (花名：酒祝) @Alibaba

Alibaba's Journey to Cloud Native

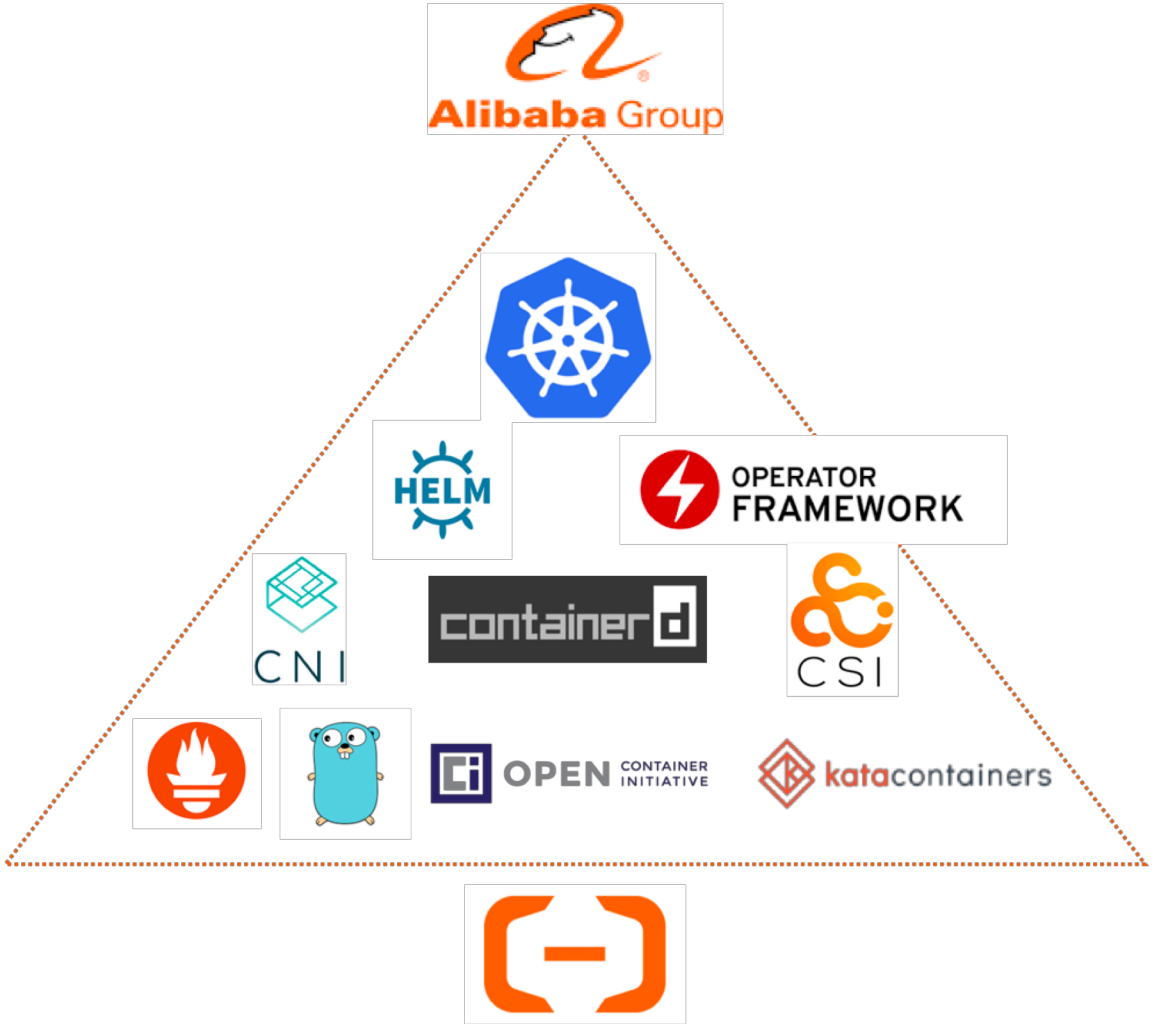


Containerize
(container centric)

Explore
(K8s API centric)

Cloud Native
(Standard + Open)

Current State

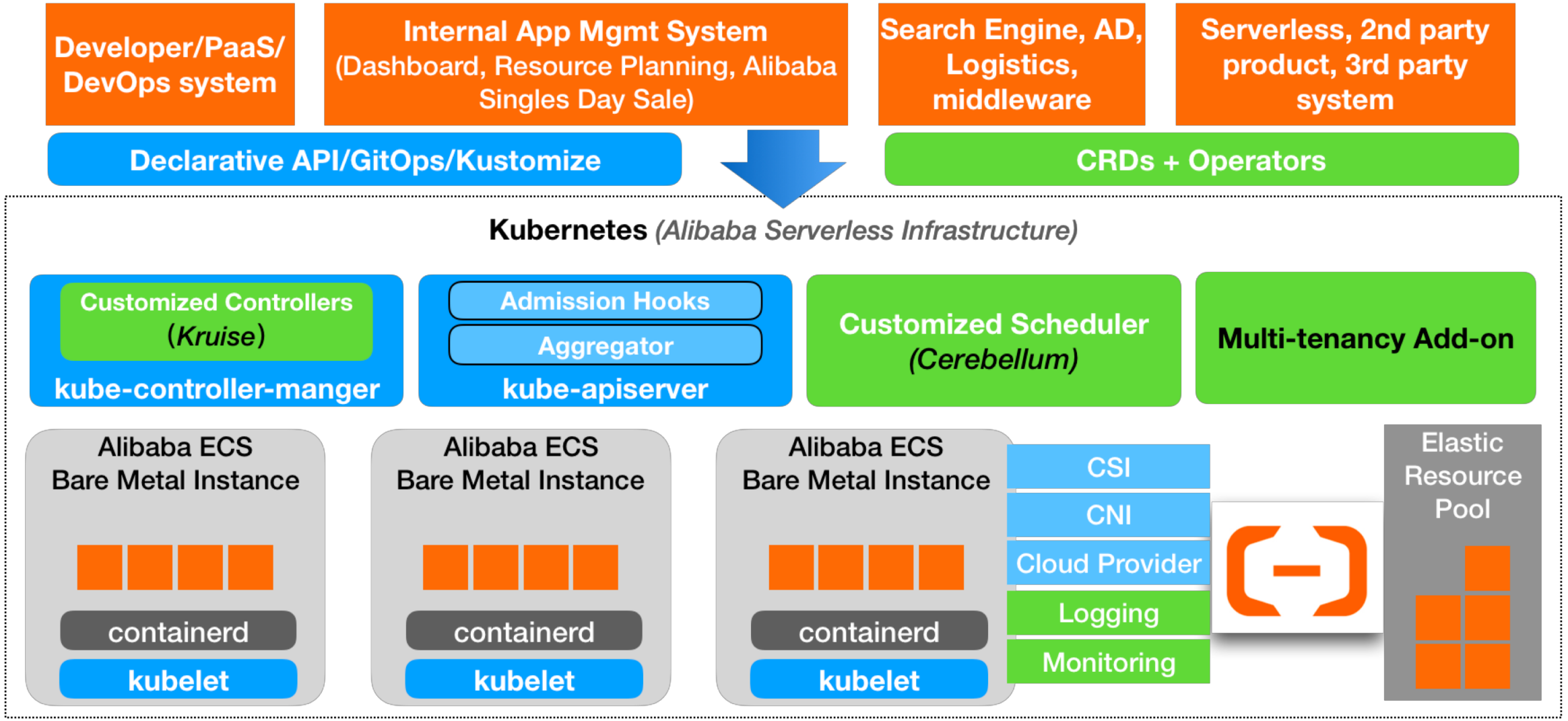


Alibaba began to move its e-business platform to cloud

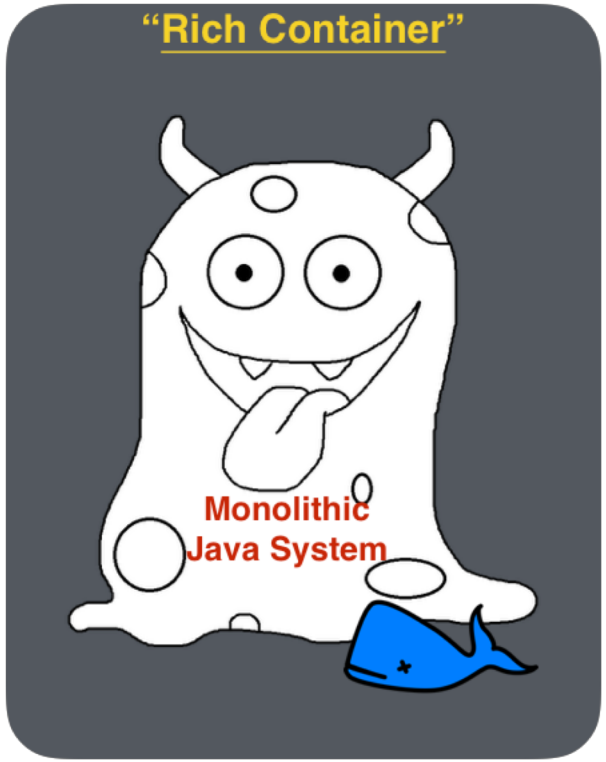
Standing on the shoulder of open source:

- Kubernetes
- Operator Framework
- CNI, CSI, CRI, DevicePlugin …
- Prometheus
- Containerd
  - runC + KataContainers
- DevOps framework from ACK
  - ACK = Alibaba Container Service for Kubernetes
- and much more …

Architecture



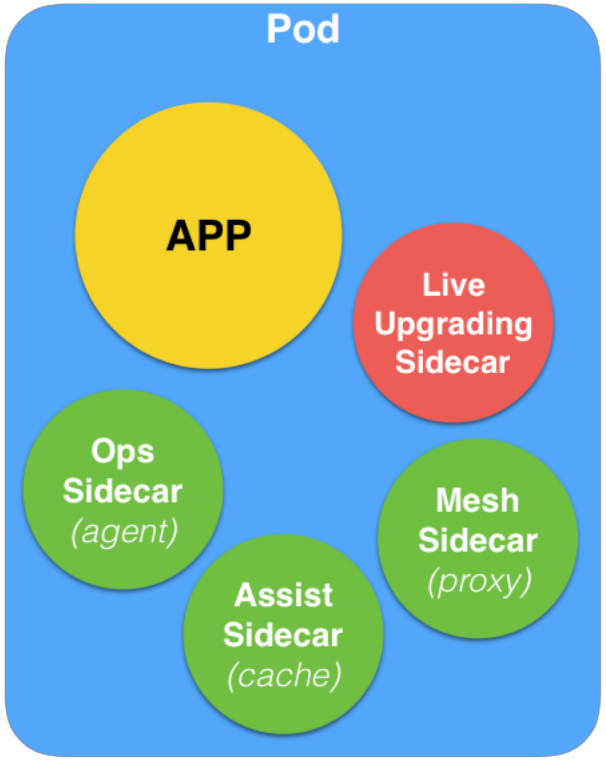
Eliminate “Rich Container”



Before 2018

- Java
- PID 1 process is Systemd
- ALL in ONE container (“Rich Container”), independent upgrade
  - app, sshd, log, monitoring, cache, VIP, DNS, proxy, agent, start/stop scripts …
- Traditional operating workflow
  - Start container -> SSH into container -> Start the app
  - Log files & user data are distributed everywhere in the container
- In-house orchestration & scheduling system

Eliminate “Rich Container”



- Shared volume
- Different resource QoS
- Fine-grained lifecycle control & health check

```
apiVersion: v1
kind: Pod
spec:
  containers:
  - env:
    - name: ali_start_app
      value: “no”
    name: main
    lifecycle:
      postStart:
        exec:
          command:
          - /bin/sh
          - -c
          - for i in $(seq 1 60); do [ -x /home/admin/.start ] && break ; sleep 5
            ; done; sudo -u admin /home/admin/.start>/var/log/kubeapp/start.log 2>&1
            && sudo -u admin /home/admin/health.sh>>/var/log/kubeapp/start.log 2>&1
      preStop:
        exec:
          command:
          - /bin/sh
          - -c
          - sudo -u admin /home/admin/stop.sh>/var/log/kubeapp/stop.log 2>&1
    livenessProbe:
        exec:
          command:
          - /bin/sh
          - -c
          - sudo -u admin /home/admin/health.sh>/var/log/kubeapp/health.log 2>&1
        initialDelaySeconds: 20
        periodSeconds: 60
        timeoutSeconds: 20
```

App start script

App stop script

Health check

The Biggest Benefit is Upgrade of Operating Model

Imperative:

- Create 3 more containers
- Delete container named xxx
- Upgrade this container with new image
- One node has broken, we should delete those containers on it and create them in other places

Declarative:

- I need 5 replicas for my application
- This application should update to a new version

Automation !
Stability!
Efficiency!

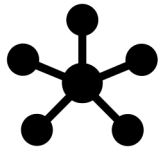
As Result

Resource utilization

Co-location by unified scheduler

App management

Cloud Native App Mgmt



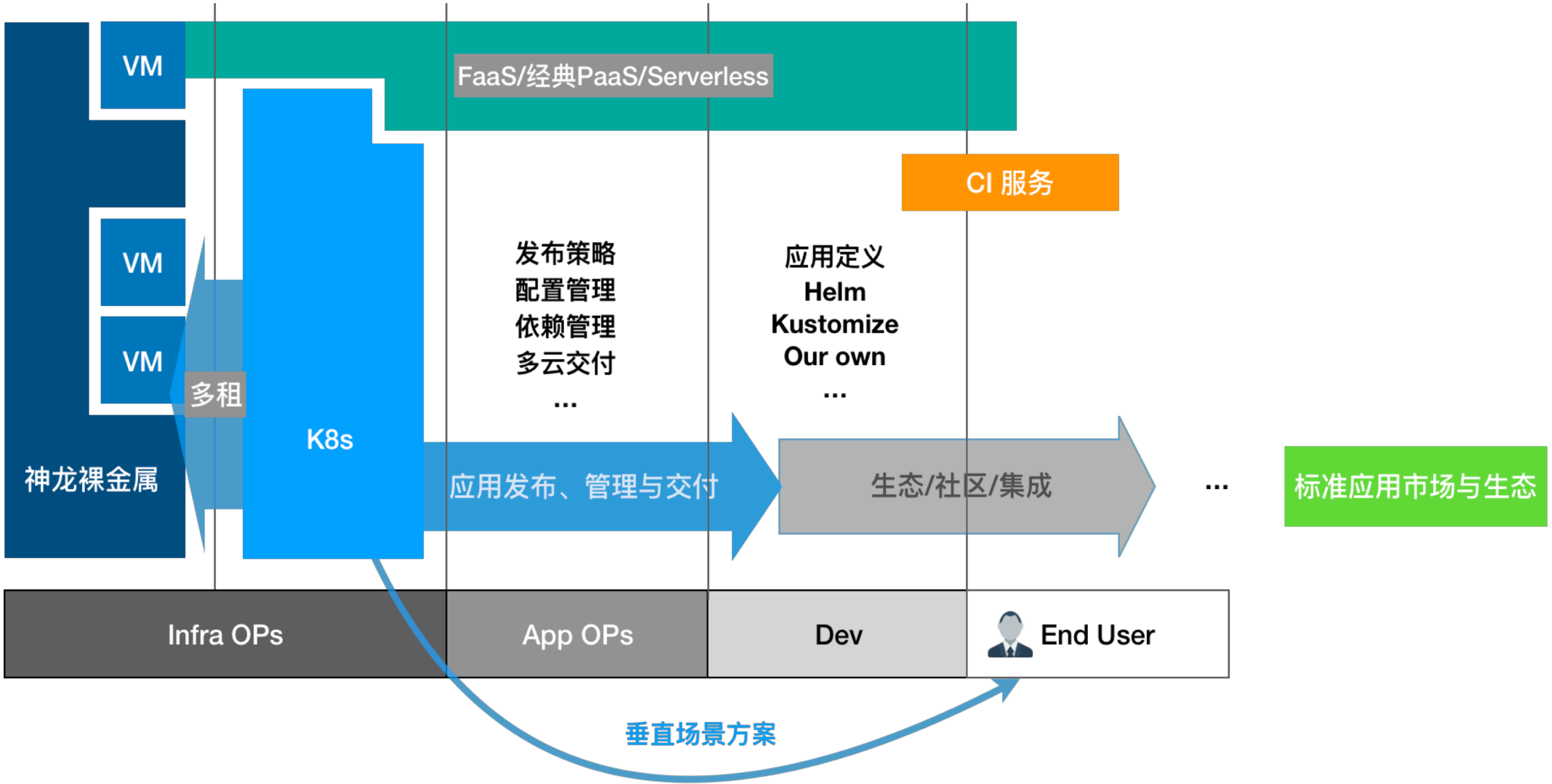
Storage strategy

Data in persistent volume

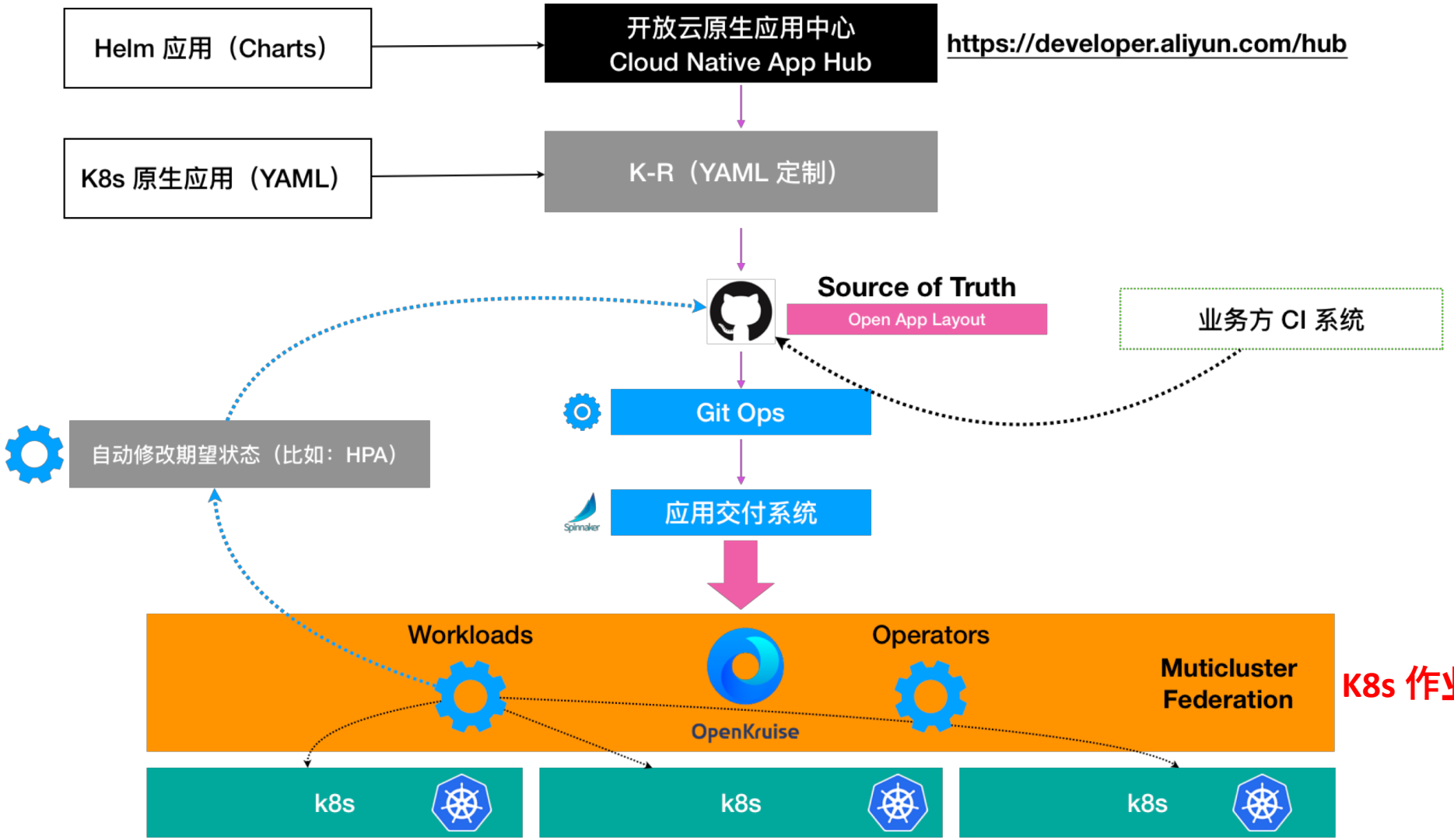
Cooperate with developers all over the world

Standard + Open + Keep updating with upstream
K8s 1.10 -> 1.12 -> 1.14 …

Cloud Native App Mgmt

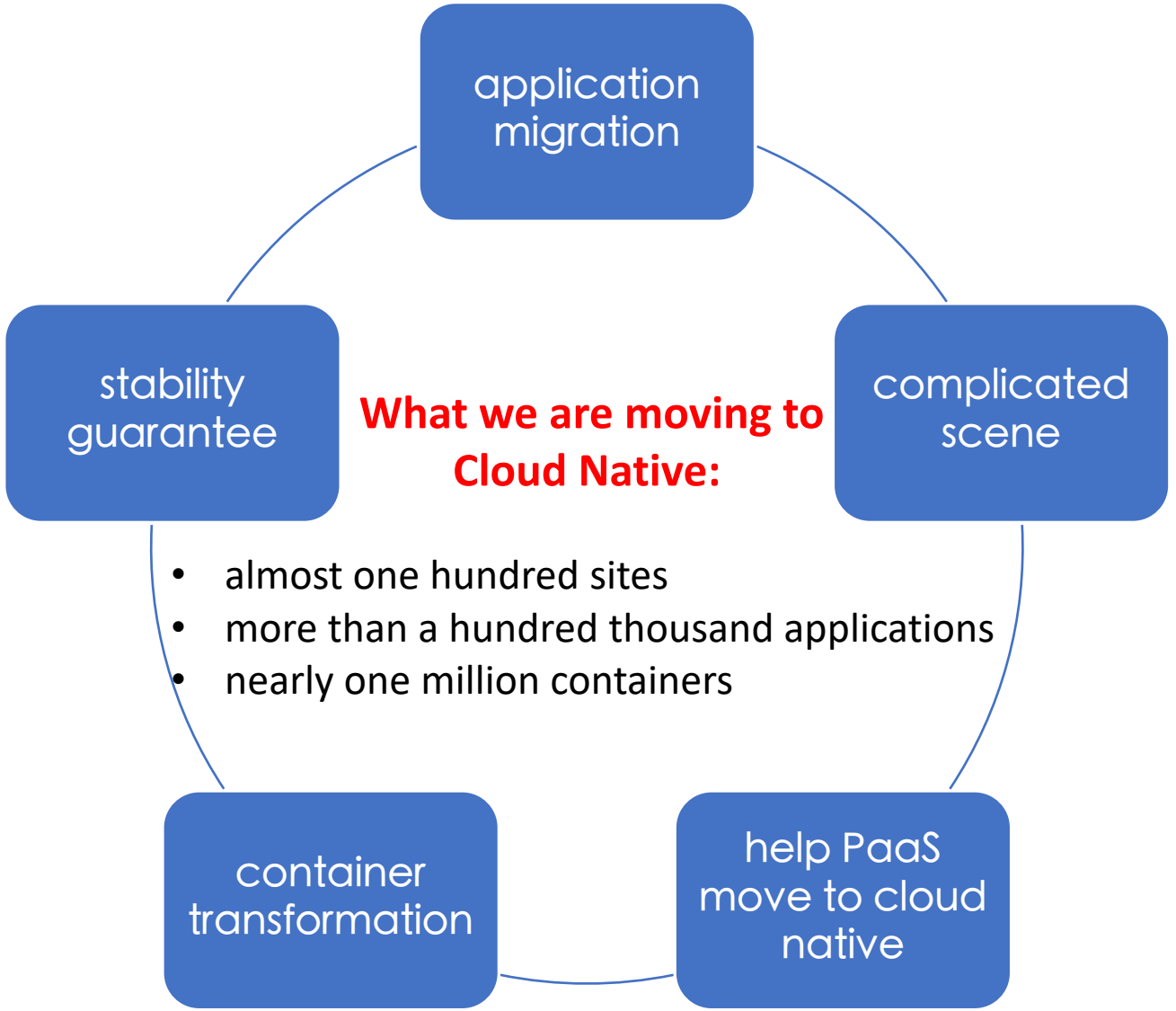


Cloud Native App Mgmt



K8s 作业管理挑战巨大

Workload Mgmt in Web-scale is Challenging



Batch placement

Before Alibaba Singles Day sale, we may:

1. Create more than a hundred thousand of Pods over thousands of applications
2. Nodes of all these Pods will be calculated by a off-line batch scheduler before Pod creation
3. Through this way, we can improve resource utilization, cpu core allocation, affinity/anti-affinity of applications and so on

Batch placement

BatchAllocationPlan

CRD for creating buffer Pods with candidate nodes

```yaml
apiVersion: batch.alibabacloud.com/v1beta1
kind: BatchAllocationPlan
metadata:
  name: sample-allocation-plan
spec:
  template:
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        resources:
          limits:
            cpu: "100m"
            memory: 200Mi
          requests:
            cpu: "100m"
            memory: 100Mi
  replicas: 3
  candidateNodes:
  - byIP: 10.0.0.5
  - byIP: 10.0.0.6
  - byIP: 10.0.0.6
  - bySN: fake-sn
```

BatchAdoption

CRD for helping workload adopt buffer Pods

```yaml
apiVersion: batch.alibabacloud.com/v1beta1
kind: BatchAdoption
metadata:
  name: sample-adoption
spec:
  podSelector:
    matchLabels:
      allocation-name: sample-allocation-plan
      allocation-succeeded: "true"
  workloadName: sample-statefulset
  workloadType: AdvancedStatefulSet
```

Application upgrade

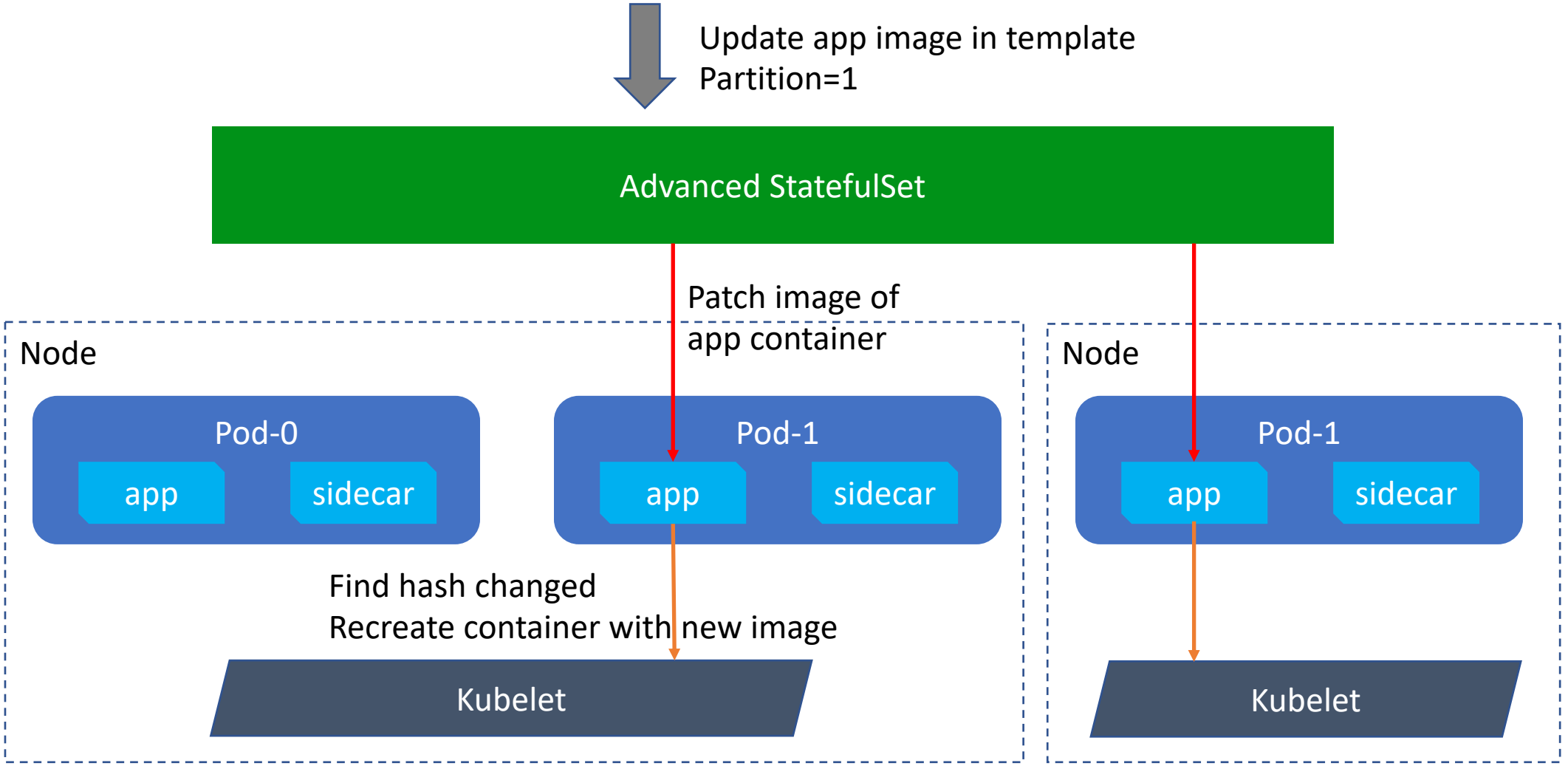
All default workloads recreate Pods during rolling update.
Strategies they offered:

- maxSurge/maxUnavailable/partition/…

What's the problem?

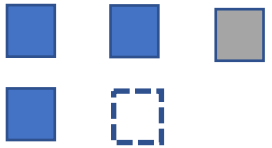
- If we use the default update policy in K8s, the determinacy will be broken
- Topology changes, image re-warm, unexpected overhead, resource allocation churn …

Introduce: in-place update

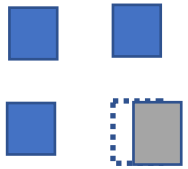


Comparation

rolling update



inplace update



	Recreate update	InPlace update
Cluster determinacy		👍
Efficiency of image downloading		👍
Requirement of resource		👍
Rescheduling and service registration		👍
Recovery automatically	👍	
Support all fields update	👍	

Sidecar management

Defined in app workloads

1. Hard to manage when there are lots of applications and workloads
2. Application developers don't know which sidecar to use
3. How to update sidecar in too many workloads

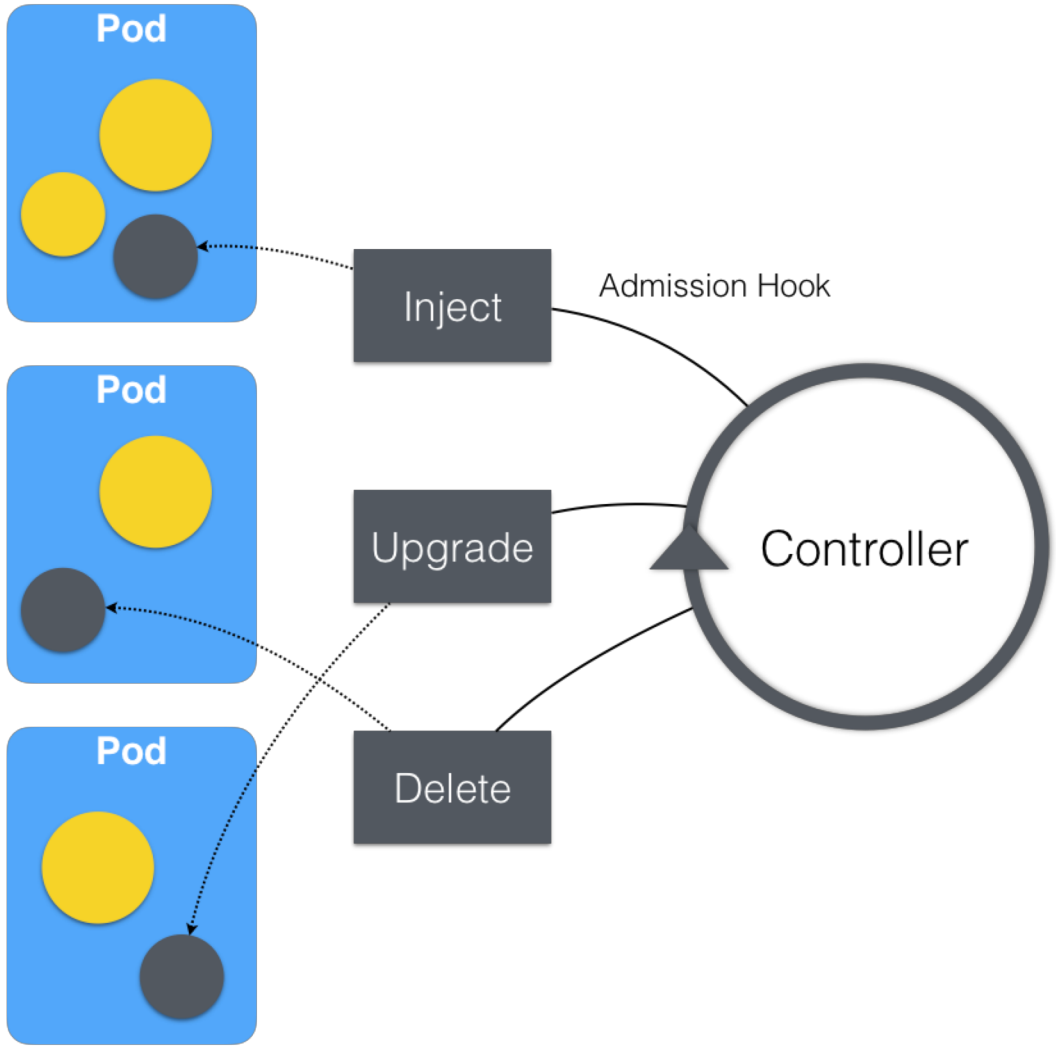
…

Injected by SidecarSet

1. Define sidecars alone, part from application workloads
2. Application developers have not to care about sidecars
3. Update sidecar containers in-place, no effect on applications

…

SidecarSet injection

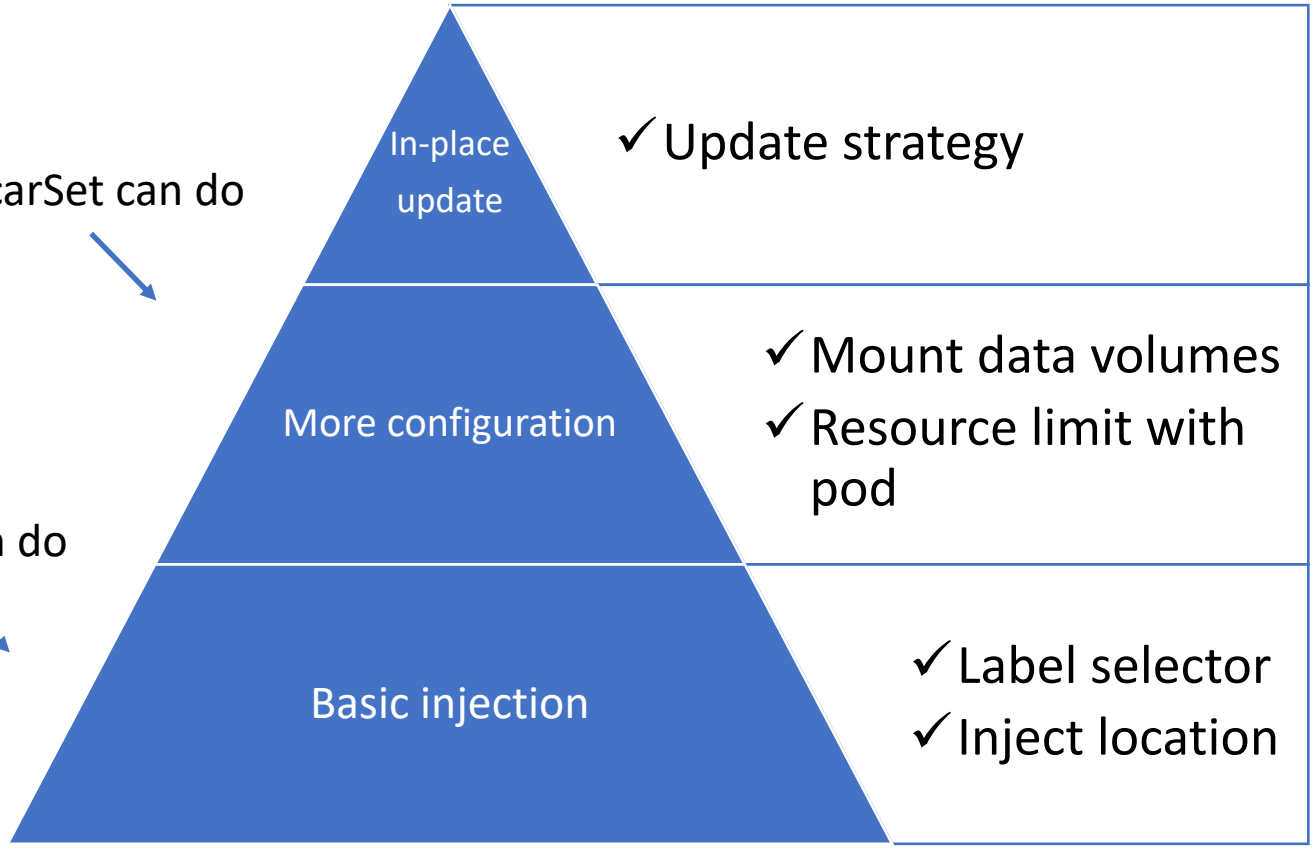


- When we have thousands sidecars, we'll need:
- A SidecarSet CRD:
  - Describe all sidecars need to be operated
- A SidecarOperator:
  1. Inject sidecar containers to selected Pods
  2. Upgrade sidecar containers following rollout policy when SidecarSet is updated
  3. Delete sidecar containers when SidecarSet is deleted

What can SidecarSet do

SidecarSet can do

Istio can do

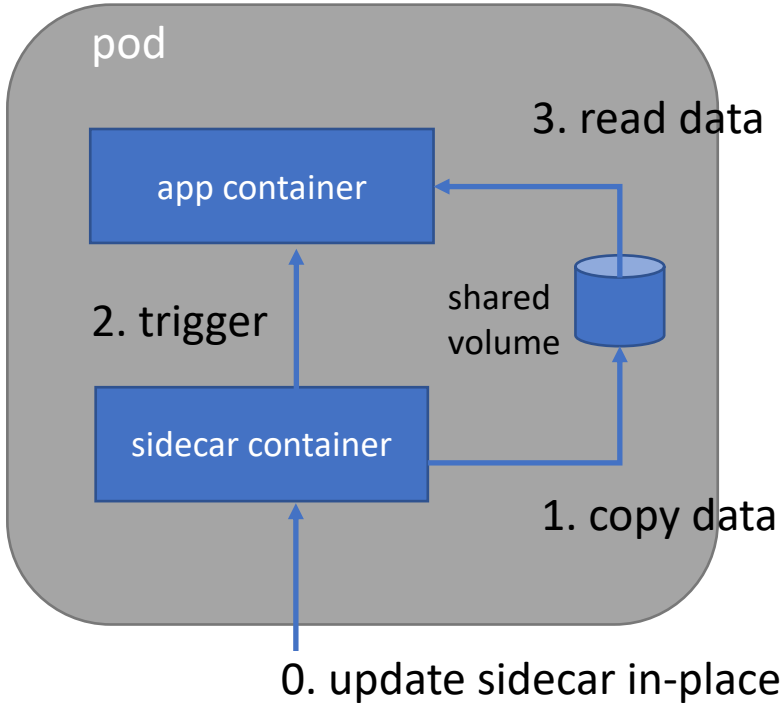


✓Update strategy

✓Mount data volumes
✓Resource limit with pod

✓Label selector
✓Inject location

Hot upgrade using sidecar



OpenKruise

KubeCon CloudNativeCon OPEN SOURCE SUMMIT China 2019

https://github.com/openkruise/kruise

OpenKruise

OpenKruise/Kruise

Kruise is at the core of the OpenKruise project. It is a set of controllers which extends and complements Kubernetes core controllers on application workload management.

Today, Kruise offers three application workload controllers:

- Advanced StatefulSet: An enhanced version of default StatefulSet with extra functionalities such as `inplace-update`, sharding by namespace.
- BroadcastJob: A job that runs pods to completion across all the nodes in the cluster.
- SidecarSet: A controller that injects sidecar container into the pod spec based on selectors.

Please see documents for more technical information.

Several tutorials are provided to demonstrate how to use the workload controllers.

Share what have extremely helped us move to Cloud Native

AdvancedStatefulSet

AdvancedStatefulSet

- MaxUnavailable

Improve speed of updating.
(For a StatefulSet with 500 Pods and update in 10 batches, this may accelerate 50 times.)

- InPlaceUpdate

Update Pods without recreate.
Avoid Pods reschedule and reshuffle
ReadinessGate ensures Pods stay NotReady during updating.

More features coming soon…

```yaml
apiVersion: apps.kruise.io/v1alpha1
kind: StatefulSet
metadata:
  name: guestbook-v1
  labels:
    app: guestbook
    version: "1.0"
spec:
  replicas: 20
  serviceName: guestbook
  selector:
    matchLabels:
      app: guestbook
  template:
    metadata:
      labels:
        app: guestbook
        version: "1.0"
    spec:
      readinessGates:
        # A new condition that ensures the pod remains at NotReady state while the in-place update is happening
        - conditionType: InPlaceUpdateReady
      containers:
      - name: guestbook
        image: openkruise/guestbook:v1
        ports:
        - name: http-server
          containerPort: 3000
  podManagementPolicy: Parallel  # allow parallel updates, works together with maxUnavailable
  updateStrategy:
    type: RollingUpdate
    rollingUpdate:
      # Do in-place update if possible, currently only image update is supported for in-place update
      podUpdatePolicy: InPlaceIfPossible
      # Allow parallel updates with max number of unavailable instances equals to 2
      maxUnavailable: 3
```

SidecarSet and BroadcastJob

SidecarSet

- Inject sidecat container into matched Pods.

Currently, it can just do injection during pod creating, and we will support sidecar update in-place soon

```yaml
apiVersion: apps.kruise.io/v1alpha1
kind: SidecarSet
metadata:
  name: guestbook-sidecar
spec:
  selector: # select the pods to be injected with sidecar containers
    matchLabels:
      app: guestbook
  containers:
    - name: guestbook-sidecar
      image: openkruise/guestbook:sidecar
      imagePullPolicy: Always
      ports:
        - name: sidecar-server
          containerPort: 4000 # different from main guestbook containerPort which is 3000
```

BroadcastJob

- A blend of DaemonSet and Job
- Run pods on all machines exactly once

Use case: software upgrade, node validator, node labeler etc

```yaml
apiVersion: apps.kruise.io/v1alpha1
kind: BroadcastJob
metadata:
  name: download-image
spec:
  template:
    spec:
      containers:
        - name: guestbook
          image: openkruise/guestbook:v2
          command: ["echo",  "started"] # a dummy command to do nothing
      restartPolicy: Never
  completionPolicy:
    type: Always
    ttlSecondsAfterFinished: 60 # the job will be deleted after 60 seconds
```

Scalability Matters in Alibaba

Scalability boundary of upstream K8s (v1.14)

- No more than 5k nodes
- No more than 150k total pods
- No more than 300k total containers
- No more than 100 pods per node

Scalability goal in our web-scale cluster

- More than 10k nodes
- More than 300k pods

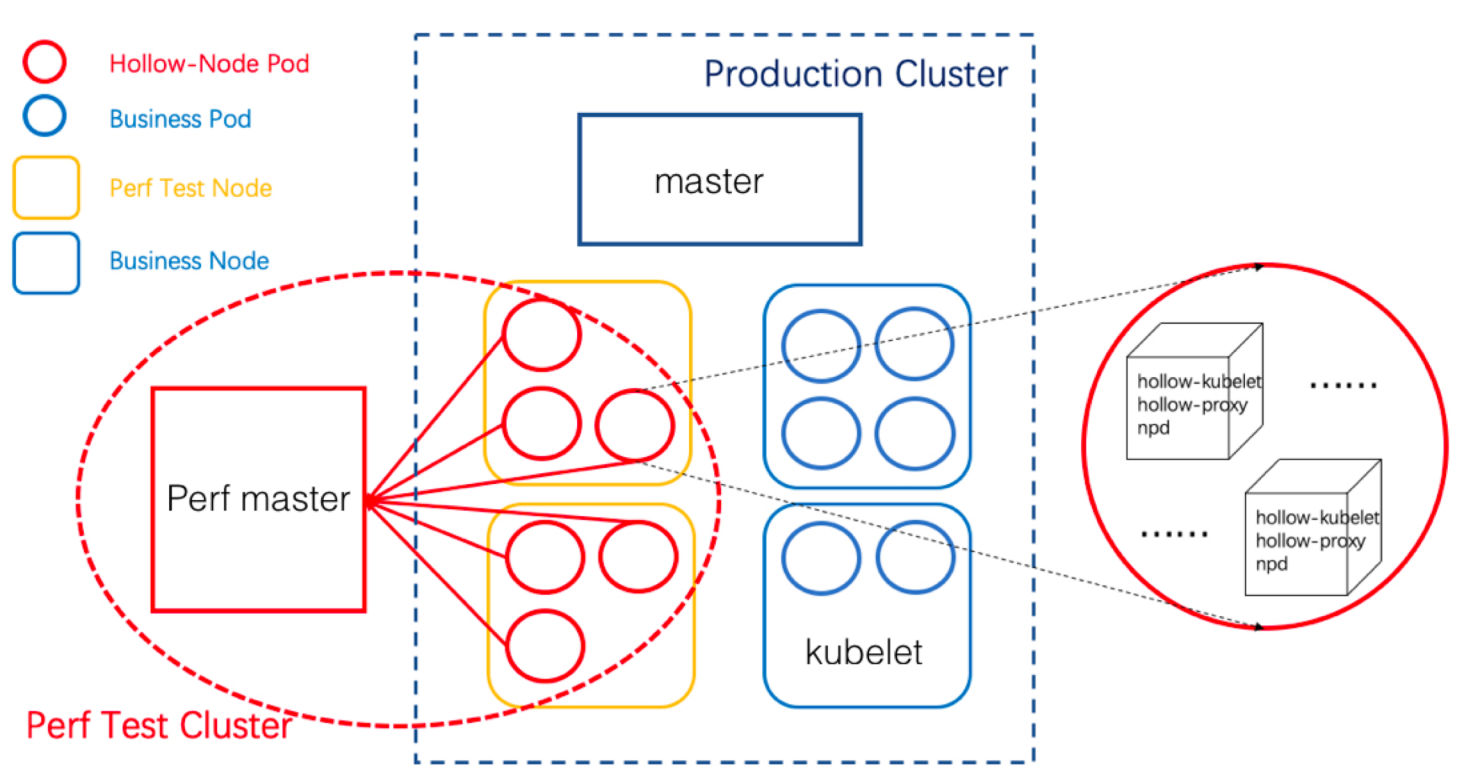
Non-goal:

- Total containers & pods per node

Question:

- **How to discover scalability issue in 10k nodes cluster?**

Performance Benchmark Toolkit



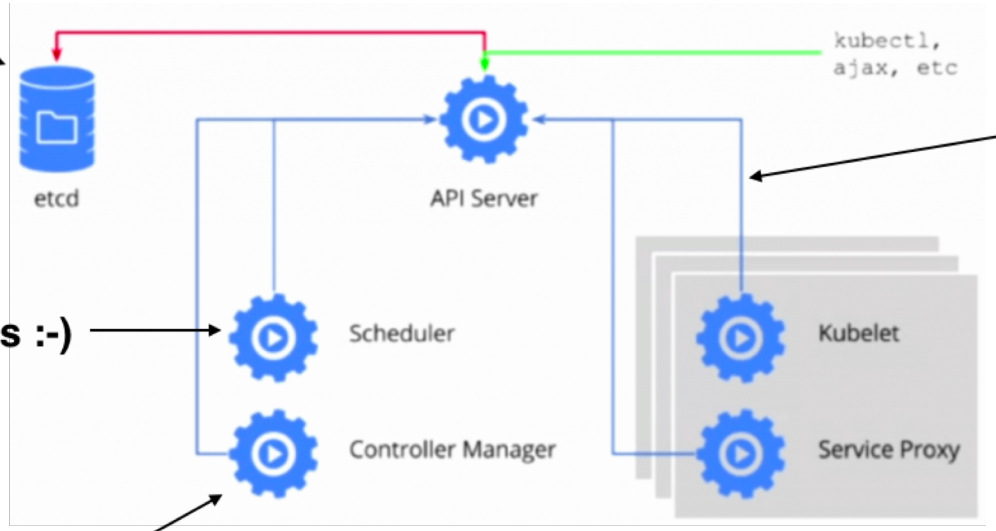
- **kubemark** with HTTP interface
  - **Hollow-Node** Pods
    - cmd/kubemark/hollow-node.go
- Taint and drain nodes for perf test, and run it
- Typical test cases in 10k nodes cluster:
  - Start up time during scaling pods
  - Time of creating and deleting pods
  - Pod listing RT
  - Failure counts

How to run?

```
curl -X POST -H "Content-Type: application/json"  \
"https://k8s-performance-toolkit.alibaba-inc.com/api/kubemark/test" \
-d '{"test_focus":"\\[Feature:Performance\\]","test_skip":"handle","node_count":10000,"pods_per_node":30}'
```

Discover Performance Bottlenecks

Concurrency, locking, data store



Large amount of heartbeat data

Our own implementation, no worries :-)

Large amount of lister & watcher

Fixing Performance Bottlenecks with Upstream

etcd

- #9296
- #9384
- #9511

Lock

- #9418

Perf

- bbolt#141

Algorithm ~24x

- #10283

Configure

APIServer

- Pod List Indexing: ~35x (upstream soon)
- Watch Bookmark: #75474 (*New!*)

Kubelet

- Bug fix：#72709

- Etcd
  - 100 clients, 1 million random key value pairs, 5000 QPS
    - Completion time: ~200s
    - Latency: 99.9% in 97.6ms
- K8s
  - 10k nodes, 100K exiting pods, scale 2000 pod
    - QPS: 133.3 pods/s, **99 %ile 3.474s**
  - On going: metrics data will crash Prometheus

* Cherry pick: Incremental heartbeat #14733

Dance with Upstream

- "fork"
  - Lock down on specific K8s release, never upgrade
  - In-house/modified K8s API, hide/wrap K8s API
  - Bypass K8s core workflow
  - Bypass K8s interface (CSI, CNI, CRI)
  - Replace kubelet with some other agent
  - …

- "non fork"
  - Keep upgrading with 2 releases lag with upstream
  - No API change
    - annotations, aggregator, CRD etc
  - Respect K8s philosophy
    - Declarative API & Controller Pattern
  - Leverage K8s interfaces & extensibility
    - CNI, CSI, admission hook, initializer, extender etc
  - Honor kubelet & CRI

One more thing: set up a small upstream team across your org, it's fun, and rewarding.

Thanks

Q&A

免登陆听课

动手实践课后自测

立即听课

CKA课程内容同步

阿里云原生最佳实践