阿里云 × CLOUD NATIVE COMPUTING FOUNDATION

云原生技术公开课

第 22 讲

有状态应用编排：StatefulSet

酒祝 阿里云技术专家

关注“阿里巴巴云原生”公众号
获取第一手技术资料

1 “有状态”需求

2 用例解读

3 操作演示

4 架构设计

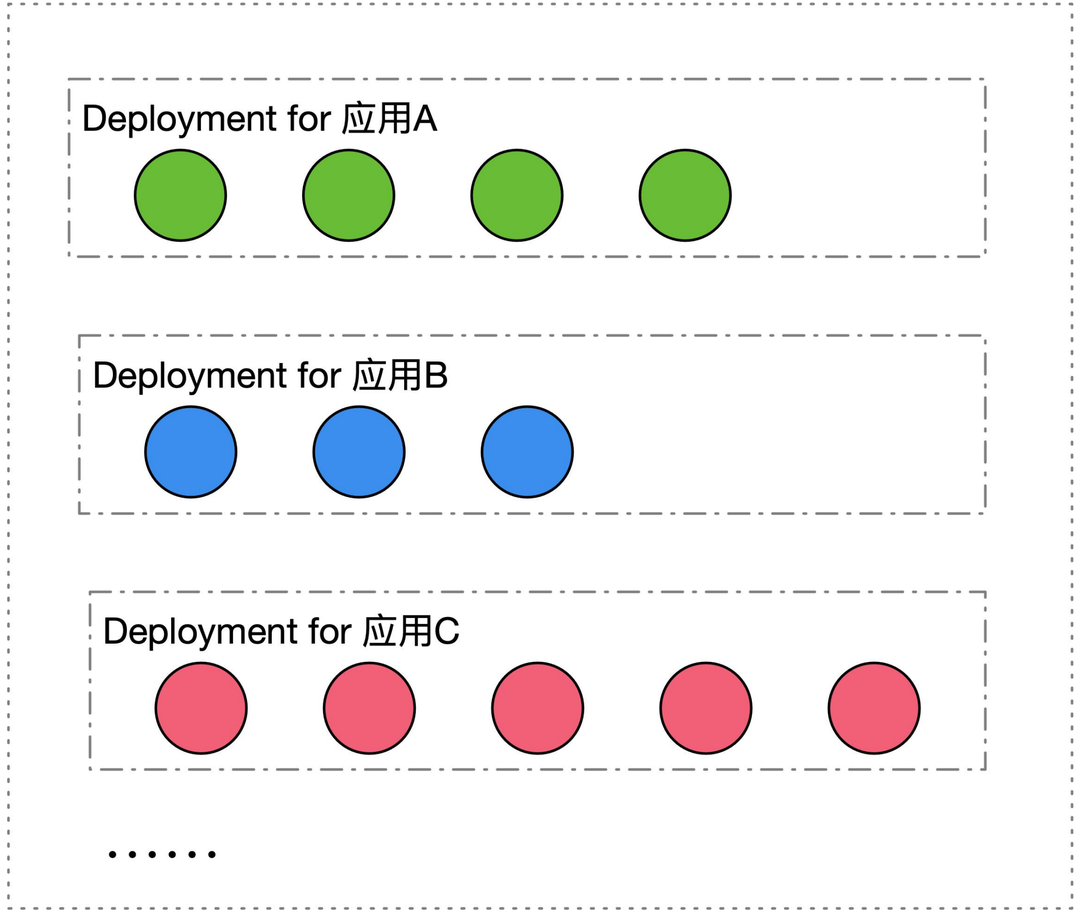
5 课后思考实践

课程回顾

第6讲 “应用编排与管理： Deployment”：

1. 定义一组Pod的期望数量， controller维持Pod数量与期望数量一致
2. 配置Pod发布方式，controller会按照给定策略更新Pod，保证更新过程中不可用的pod数量在限定范围内
3. 如果发布有问题，支持“一键”回滚

Deployment认为：管理的所有同版本的Pod都是一模一样的副本



需求分析

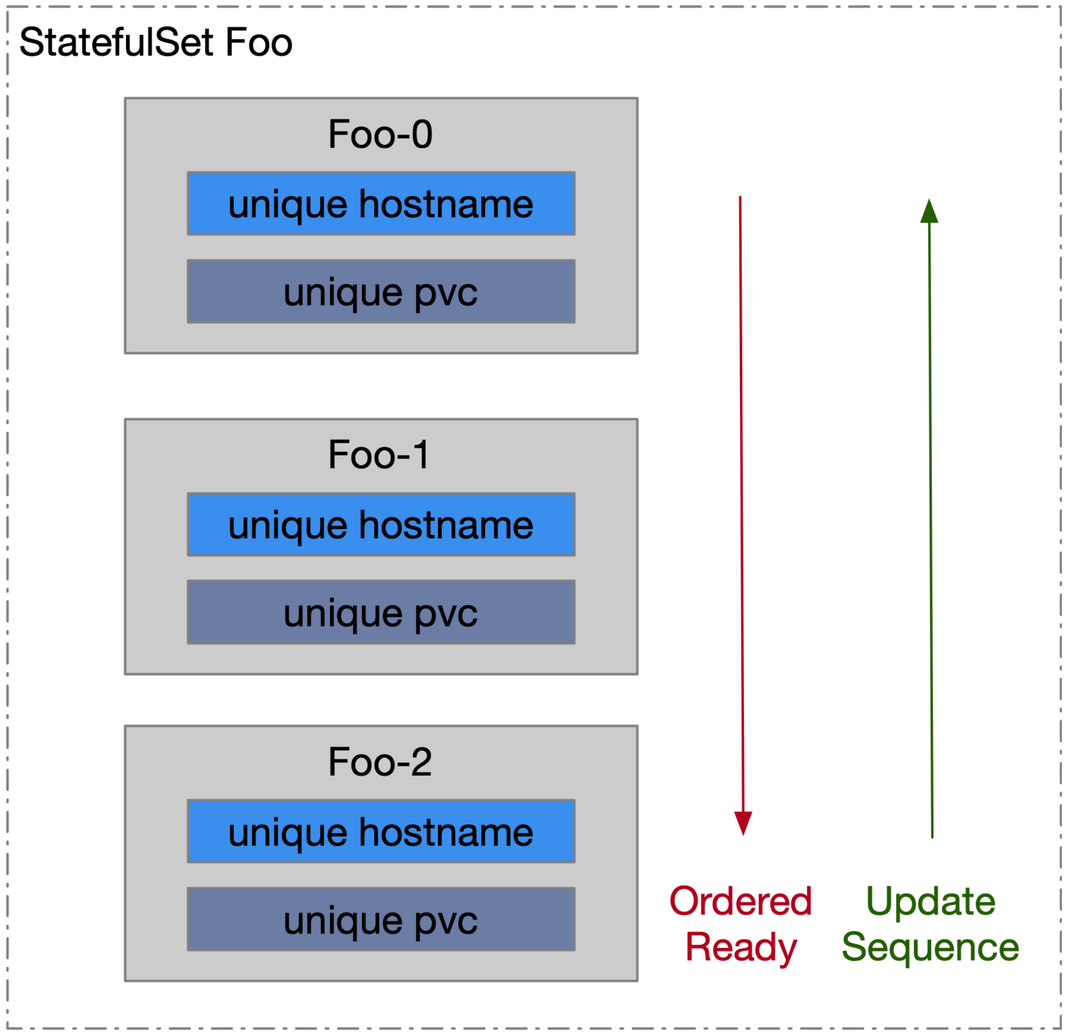
以下需求，来自于某一些有状态应用，思考可以用Deployment满足吗？

1. Pod之间并非相同的副本，每个Pod有一个独立标识
2. Pod独立标识要能够对应到一个固定的网络标识，并在发布升级后继续保持
3. 每个Pod有一块独立的存储盘，并在发布升级后还能继续挂载原有的盘（保留数据）
4. 应用发布时，按照固定顺序升级Pod

StatefulSet: 主要面向有状态应用管理的控制器

StatefulSet能较好地满足一些有状态应用特有的需求：

1. 每个Pod有Order序号，会按序号创建、删除、更新Pod
2. 通过配置headless service，使每个Pod有一个唯一的网络标识(hostname)
3. 通过配置pvc template，每个Pod有一块独享的pv存储盘
4. 支持一定数量的灰度发布



1 “有状态”需求

2 用例解读

3 操作演示

4 架构设计

5 课后思考实践

StatefulSet范例创建

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  ports:
  - port: 80
    name: web
  clusterIP: None
  selector:
    app: nginx
```

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: nginx-web
spec:
  selector:
    matchLabels:
      app: nginx
  serviceName: "nginx"
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
     containers:
      - name: nginx
        image: nginx:alpine
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: www-storage
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
  - metadata:
      name: www-storage
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 20Gi
```

Service、StatefulSet状态

```
$ kubectl get service nginx
NAME    TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
nginx   ClusterIP   None         <none>        80/TCP     99s


$ kubectl get endpoints nginx
NAME    ENDPOINTS                                          AGE
nginx   172.27.0.133:80,172.27.0.5:80,172.27.1.9:80   94s


$ kubectl get sts nginx-web
NAME        READY   AGE
nginx-web   3/3     3m5s
```

Pod、PVC状态

```
$ kubectl get pod -o wide
NAME          READY   STATUS    RESTARTS   AGE   IP              NODEGATES
nginx-web-0   1/1     Running   0          14m   172.27.0.133    cn-hangzhou.192.168.1.85
nginx-web-1   1/1     Running   0          14m   172.27.0.5      cn-hangzhou.192.168.1.83
nginx-web-2   1/1     Running   0          13m   172.27.1.9      cn-hangzhou.192.168.1.84
```

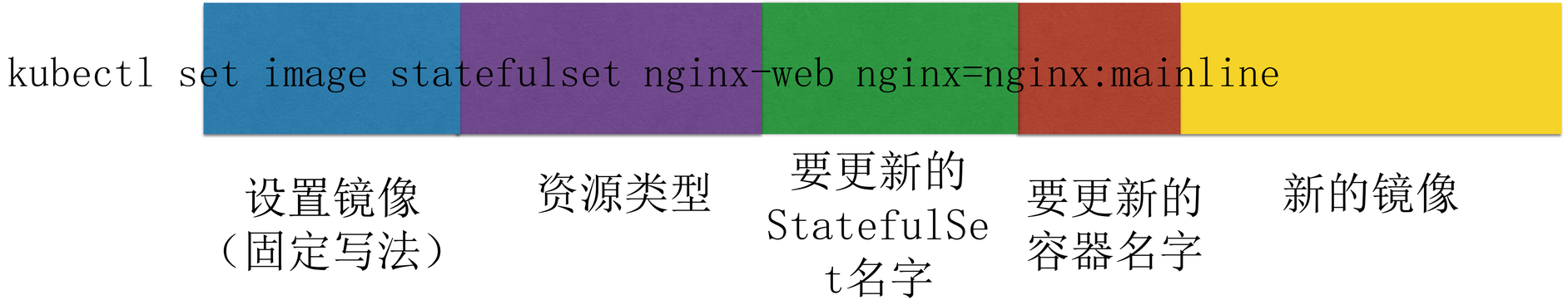
```
$ kubectl get pvc
NAME                      STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STO
www-storage-nginx-web-0   Bound    disk-451c1393-ee56-11e9-a93f-be6d52dd44ce   20Gi       RWO            ali
www-storage-nginx-web-1   Bound    disk-4dd85adf-ee56-11e9-a93f-be6d52dd44ce   20Gi       RWO            ali
www-storage-nginx-web-2   Bound    disk-5bbf8820-ee56-11e9-a93f-be6d52dd44ce   20Gi       RWO            ali
```

Pod版本

不同于Deployment使用ReplicaSet来管理版本和维持副本数，StatefulSet controller直接管理下属的Pod，而Pod中用一个label来标识版本：controller-revision-hash。

```
$ kubectl get pod -L controller-revision-hash
NAME          READY   STATUS    RESTARTS   AGE   CONTROLLER-REVISION-HASH
nginx-web-0   1/1     Running   0          21m   nginx-web-677759c9b8
nginx-web-1   1/1     Running   0          21m   nginx-web-677759c9b8
nginx-web-2   1/1     Running   0          20m   nginx-web-677759c9b8
```

更新镜像




```
apiVersion: apps/v1
kind: StatefulSet
metadata:
 generation: 2
  name: nginx-web
spec:
    # . . .
    spec:
      containers:
      - name: nginx
        image: nginx:mainline
```

查看新版本状态

- Revision hash升级到：nginx-web-7c55499668
- 从2 -> 1 -> 0序号升级
- 复用之前Pod的PVC

```
status:
  collisionCount: 0
  currentReplicas: 3
  currentRevision: nginx-web-7c55499668
  observedGeneration: 2
  readyReplicas: 3
  replicas: 3
  updateRevision: nginx-web-7c55499668
  updatedReplicas: 3
```

```
$ kubectl get pod -L controller-revision-hash
NAME          READY   STATUS    RESTARTS   AGE     CONTROLLER-REVISION-HASH
nginx-web-0   1/1     Running   0          6m50s   nginx-web-7c55499668
nginx-web-1   1/1     Running   0          7m10s   nginx-web-7c55499668
nginx-web-2   1/1     Running   0          7m21s   nginx-web-7c55499668
```

1 “有状态”需求

2 用例解读

3 操作演示

4 架构设计

5 课后思考实践

1 “有状态”需求

2 用例解读

3 操作演示

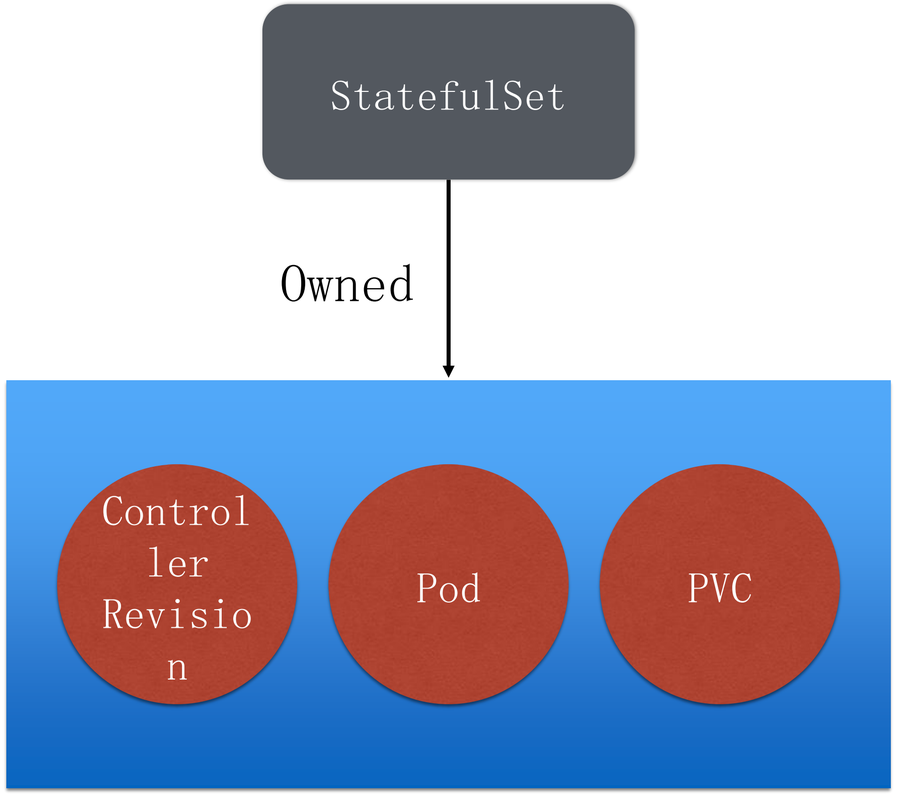
4 架构设计

5 课后思考实践

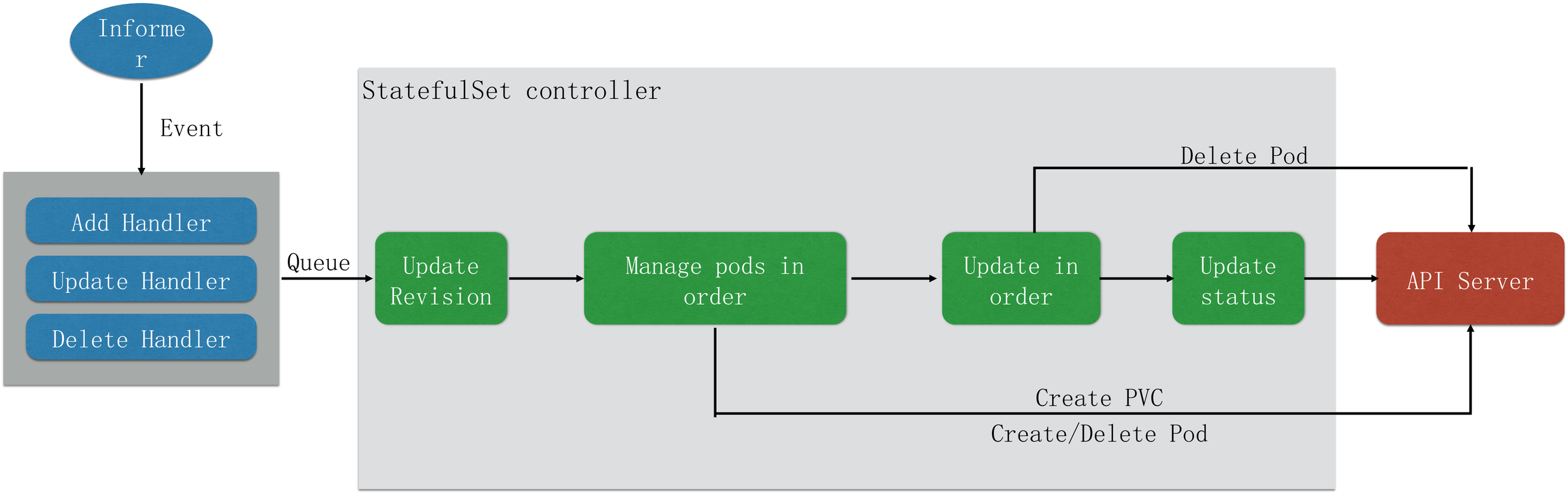
管理模式

StatefulSet会创建管理的资源：

- ControllerRevision：通过这个资源，StatefulSet可以很方便地管理不同版本的template模板
- PVC：如果在StatefulSet中定义了volumeClaimTemplates，StatefulSet会在创建Pod之前，先根据这个模板创建PVC，并把PVC加到Pod volumes中。
- Pod：StatefulSet按照顺序创建、删除、更新Pod，每个Pod有唯一的序号

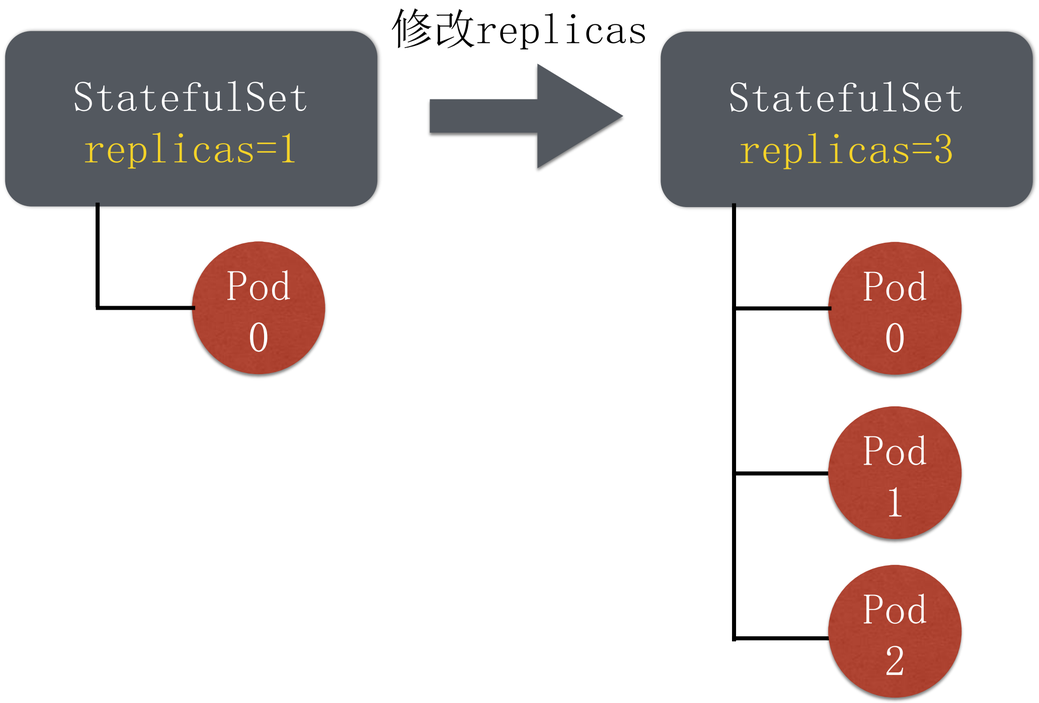


StatefulSet控制器



扩容模拟

StatefulSet下的Pod，从序号0开始创建。
因此，replicas=N的一个StatefulSet，创建出的Pod序号为 [0, N)

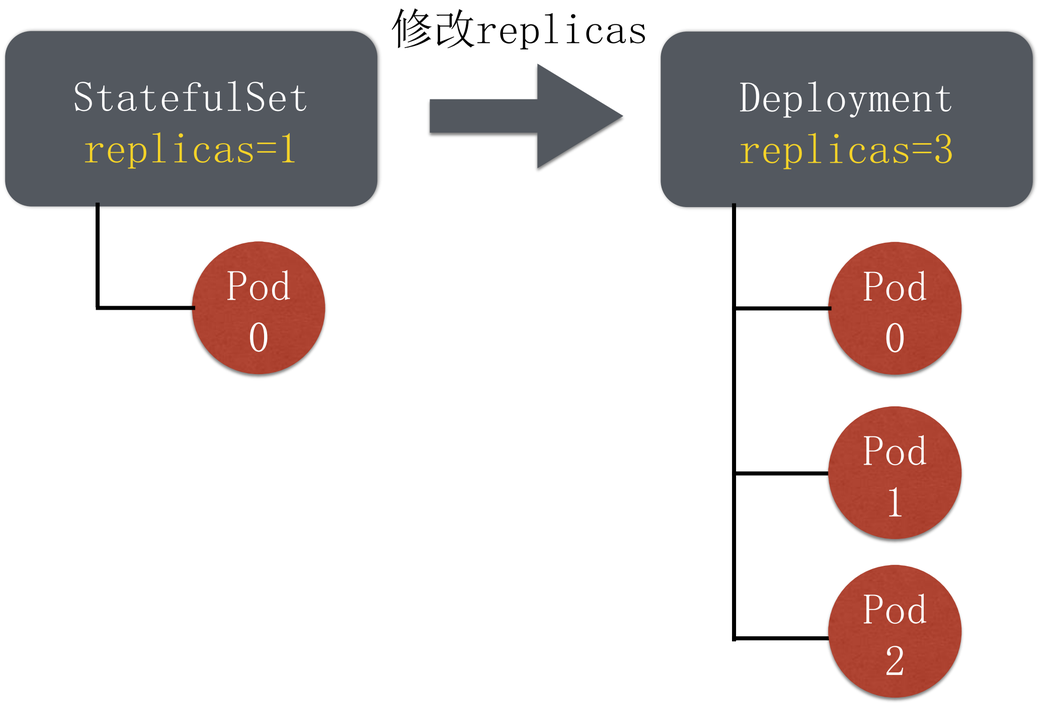


扩缩容管理策略

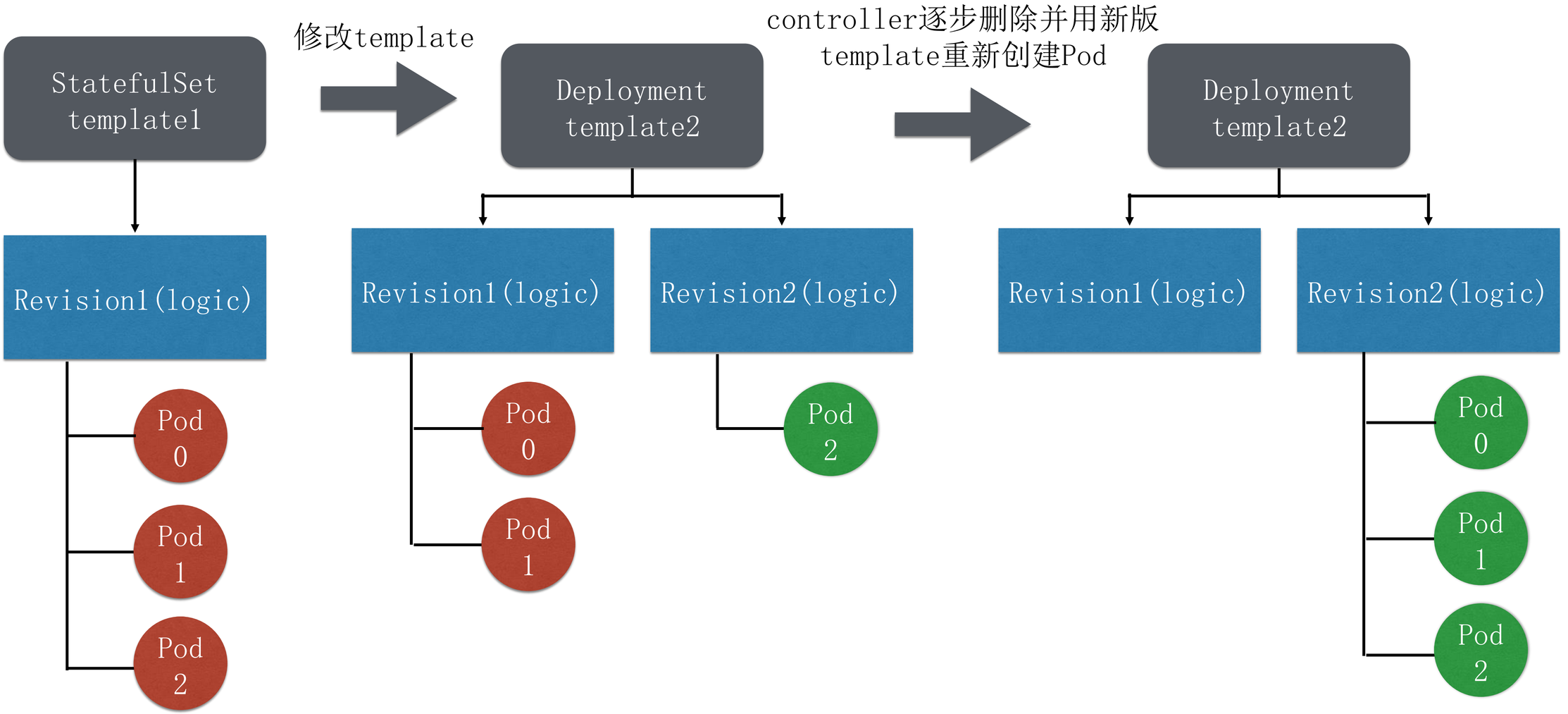
StatefulSet.spec中，有一个字段名为podManagementPolicy，可选策略为OrderedReady和Parallel，默认为前者。

- OrderedReady：扩缩容按照order顺序执行。扩容时，必须前面序号的Pod都ready了，才能扩下一个；缩容时，按照倒序删除。

- Parallel：并行扩缩容，不需要等前面Pod都ready或删除后再处理下一个



发布模拟


StatefulSet
template1
Revision1(logic)
Pod
0
Pod
1
Pod
2
修改template
Deployment
template2
Revision1(logic)
Revision2(logic)
Pod
0
Pod
1
Pod
2
controller逐步删除并用新版
template重新创建Pod
Deployment
template2
Revision1(logic)
Revision2(logic)
Pod
0
Pod
1
Pod
2

spec字段解析1

Replicas：期望数量

Selector：选择器，必须匹配
.spec.template.metadata.labels

Template：Pod模板

VolumeClaimTemplates：PVC模板列表

```go
// A StatefulSetSpec is the specification of a StatefulSet.
type StatefulSetSpec struct {
    // replicas is the desired number of replicas of the given Template.
    // These are replicas in the sense that they are instantiations of the
    // same Template, but individual replicas also have a consistent identity.
    // If unspecified, defaults to 1.
    // TODO: Consider a rename of this field.
    // +optional
    Replicas *int32 `json:"replicas,omitempty" protobuf:"varint,1,opt,name=replicas"`

    // selector is a label query over pods that should match the replica count.
    // It must match the pod template's labels.
    // More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/#label-selectors
    Selector *metav1.LabelSelector `json:"selector" protobuf:"bytes,2,opt,name=selector"`

    // template is the object that describes the pod that will be created if
    // insufficient replicas are detected. Each pod stamped out by the StatefulSet
    // will fulfill this Template, but have a unique identity from the rest
    // of the StatefulSet.
    Template v1.PodTemplateSpec `json:"template" protobuf:"bytes,3,opt,name=template"`

    // volumeClaimTemplates is a list of claims that pods are allowed to reference.
    // The StatefulSet controller is responsible for mapping network identities to
    // claims in a way that maintains the identity of a pod. Every claim in
    // this list must have at least one matching (by name) volumeMount in one
    // container in the template. A claim in this list takes precedence over
    // any volumes in the template, with the same name.
    // TODO: Define the behavior if a claim already exists with the same name.
    // +optional
    VolumeClaimTemplates []v1.PersistentVolumeClaim `json:"volumeClaimTemplates,omitempty" protobuf:"bytes,4,
```

spec字段解析2

ServiceName：对应Headless Service的名字，用于给Pod生成唯一网络标识

PodManagementPolicy：Pod管理策略

UpdateStrategy：Pod升级策略

RevisionHistoryLimit：保留历史ControllerRevision的数量限制（默认为10）

```
// serviceName is the name of the service that governs this StatefulSet.
// This service must exist before the StatefulSet, and is responsible for
// the network identity of the set. Pods get DNS/hostnames that follow the
// pattern: pod-specific-string.serviceName.default.svc.cluster.local
// where "pod-specific-string" is managed by the StatefulSet controller.
ServiceName string `json:"serviceName" protobuf:"bytes,5,opt,name=serviceName"`

// podManagementPolicy controls how pods are created during initial scale up,
// when replacing pods on nodes, or when scaling down. The default policy is
// `OrderedReady`, where pods are created in increasing order (pod-0, then
// pod-1, etc) and the controller will wait until each pod is ready before
// continuing. When scaling down, the pods are removed in the opposite order.
// The alternative policy is `Parallel` which will create pods in parallel
// to match the desired scale without waiting, and on scale down will delete
// all pods at once.
// +optional
PodManagementPolicy PodManagementPolicyType `json:"podManagementPolicy,omitempty" protobuf:"bytes,6,opt,name=podManage

// updateStrategy indicates the StatefulSetUpdateStrategy that will be
// employed to update Pods in the StatefulSet when a revision is made to
// Template.
UpdateStrategy StatefulSetUpdateStrategy `json:"updateStrategy,omitempty" protobuf:"bytes,7,opt,name=updateStrategy"`

// revisionHistoryLimit is the maximum number of revisions that will
// be maintained in the StatefulSet's revision history. The revision history
// consists of all revisions not represented by a currently applied
// StatefulSetSpec version. The default value is 10.
RevisionHistoryLimit *int32 `json:"revisionHistoryLimit,omitempty" protobuf:"varint,8,opt,name=revisionHistoryLimit"`
```

升级策略字段解析

策略类型：
RollingUpdate滚动升级
OnDelete禁止主动升级

Partition：
滚动升级时，保留旧版本Pod的数量
假设replicas=N，partition=M（M<=N），则最终旧版本Pod为[0, M)
新版本Pod为[M, N)

```go
// StatefulSetUpdateStrategy indicates the strategy that the StatefulSet
// controller will use to perform updates. It includes any additional parameters
// necessary to perform the update for the indicated strategy.
type StatefulSetUpdateStrategy struct {
    // Type indicates the type of the StatefulSetUpdateStrategy.
    // Default is RollingUpdate.
    // +optional
    Type StatefulSetUpdateStrategyType `json:"type,omitempty" protobuf:"bytes,1,opt,name=type,casttype=Statefu
    // RollingUpdate is used to communicate parameters when Type is RollingUpdateStatefulSetStrategyType.
    // +optional
    RollingUpdate *RollingUpdateStatefulSetStrategy `json:"rollingUpdate,omitempty" protobuf:"bytes,2,opt,name=
}

// StatefulSetUpdateStrategyType is a string enumeration type that enumerates
// all possible update strategies for the StatefulSet controller.
type StatefulSetUpdateStrategyType string

const (
    // RollingUpdateStatefulSetStrategyType indicates that update will be
    // applied to all Pods in the StatefulSet with respect to the StatefulSet
    // ordering constraints. When a scale operation is performed with this
    // strategy, new Pods will be created from the specification version indicated
    // by the StatefulSet's updateRevision.
    RollingUpdateStatefulSetStrategyType = "RollingUpdate"
    // OnDeleteStatefulSetStrategyType triggers the legacy behavior. Version
    // tracking and ordered rolling restarts are disabled. Pods are recreated
    // from the StatefulSetSpec when they are manually deleted. When a scale
    // operation is performed with this strategy,specification version indicated
    // by the StatefulSet's currentRevision.
    OnDeleteStatefulSetStrategyType = "OnDelete"
)

// RollingUpdateStatefulSetStrategy is used to communicate parameter for RollingUpdateStatefulSetStrategyType.
type RollingUpdateStatefulSetStrategy struct {
    // Partition indicates the ordinal at which the StatefulSet should be
    // partitioned.
    // Default value is 0.
    // +optional
    Partition *int32 `json:"partition,omitempty" protobuf:"varint,1,opt,name=partition"`
}
```

关注“阿里巴巴云原生”公众号
获取第一手技术资料