STATEFUL SETS

About Stateful sets

In Application level there are two types, state less and state full.
State-less means any activity cannot be stored, but stateful means activity been stored, like DB.
Stateful is like deployment, in extra from API stateful will be added. Headless service is very important to create state-full, since to communicate internal components, it will help to fix the domain name for internal communication till the stateful set present. Pods will be created one after one and while deleting in reverse order one by one. In case any pods created under state-full, if gets crashed means then it will comeback with same name only, where as in deployment it will not come up with same name.

Actual Readme:-

Manages the deployment and scaling of a set of Pods and provides guarantees about the ordering and uniqueness of these Pods. Like a Deployment , a Stateful Set manages Pods that are based on an identical container spec. Unlike a Deployment, a Stateful Set maintains a sticky identity for each of their Pods. These pods are created from the same spec but are not interchangeable: each has a persistent identifier that it maintains across any rescheduling. A StatefulSet operates under the same pattern as any other Controller. You define your desired state in a StatefulSet object, and the StatefulSet controller makes any necessary updates to get there from the current state.

Using StatefulSets

StatefulSets are valuable for applications that require one or more of the following.
Stable, unique network identifier.
Stable, persistent storage.
Ordered, graceful deployment and scaling.
Ordered, automated rolling updates.

In the above, stable is synonymous with persistence across Pod (re)scheduling. If an application doesn't require any stable identifiers or ordered deployment, deletion, or scaling, you should deploy your application with a controller that provides a set of stateless replicas. Controllers such as Deployment or ReplicaSet may be better suited to your stateless needs

Components

The example below demonstrates the components of a StatefulSet.
1) A Headless Service, named nginx, is used to control the network domain.
2) The StatefulSet, named web, has a Spec that indicates that 3 replicas of the nginx container will be launched in unique Pods.
3) The volumeClaimTemplates will provide stable storage using PersistentVolumes provisioned by a PersistentVolume Provisioner.

Deployment and Scaling Guarantees

1) For a StatefulSet with N replicas, when Pods are being deployed, they are created sequentially, in order from {0..N-1}.
2) When Pods are being deleted, they are terminated in reverse order, from {N-1..0}.
3) Before a scaling operation is applied to a Pod, all of its predecessors must be Running and Ready.
4) Before a Pod is terminated, all its successors must be completely shutdown

Pod Management Policies

In Kubernetes 1.7 and later, StatefulSet allows you to relax its ordering guarantees while preserving its uniqueness and identity guarantees via its .spec.podManagementPolicy field.

OrderedReady Pod Management

OrderedReady pod management is the default for StatefulSets. It implements the behavior described above.

Parallel Pod Management

Parallel pod management tells the StatefulSet controller to launch or terminate all Pods in parallel, and to not wait for Pods to become Running and Ready or completely terminated prior to launching or terminating another Pod. This option only affects the behavior for scaling operations. Updates are not affected.

Update Strategies

In Kubernetes 1.7 and later, StatefulSet's .spec.updateStrategy field allows you to configure and disable automated rolling updates for containers, labels, resource request/limits, and annotations for the Pods in a StatefulSet.

On Delete

The OnDelete update strategy implements the legacy (1.6 and prior) behavior. When a StatefulSet's .spec.updateStrategy.type is set to OnDelete, the StatefulSet controller will not automatically update the Pods in a StatefulSet. Users must manually delete Pods to cause the controller to create new Pods that reflect modifications made to a StatefulSet's .spec.template.

Rolling Updates

The RollingUpdate update strategy implements automated, rolling update for the Pods in a StatefulSet. It is the default strategy when .spec.updateStrategy is left unspecified. When a StatefulSet's .spec.updateStrategy.type is set to RollingUpdate, the StatefulSet controller will delete and recreate each Pod in the StatefulSet. It will proceed in the same order as Pod termination (from the largest ordinal to the smallest), updating each Pod one at a time. It will wait until an updated Pod is Running and Ready prior to updating its predecessor.

Partitions:

The RollingUpdate update strategy can be partitioned, by specifying a .spec.updateStrategy.rollingUpdate.partition. If a partition is specified, all Pods with an ordinal that is greater than or equal to the partition will be updated when the StatefulSet's .spec.template is updated. All Pods with an ordinal that is less than the partition will not be updated, and, even if they are deleted, they will be recreated at the previous version. If a StatefulSet's .spec.updateStrategy.rollingUpdate.partition is greater than its .spec.replicas, updates to its .spec.template will not be propagated to its Pods. In most cases you will not need to use a partition, but they are useful if you want to stage an update, roll out a canary, or perform a phased roll out.

Forced Rollback:

When using Rolling Updates with the default Pod Management Policy (OrderedReady), it's possible to get into a broken state that requires manual intervention to repair.

If you update the Pod template to a configuration that never becomes Running and Ready (for example, due to a bad binary or application-level configuration error), StatefulSet will stop the rollout and wait.

In this state, it's not enough to revert the Pod template to a good configuration. Due to a known issue, StatefulSet will continue to wait for the broken Pod to become Ready (which never happens) before it will attempt to revert it back to the working configuration.

After reverting the template, you must also delete any Pods that StatefulSet had already attempted to run with the bad configuration. StatefulSet will then begin to recreate the Pods using the reverted template

##Create a statefull set with default headless service##

cat nginx-statefullset.yml

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  ports:
  - port: 80
    name: web
  clusterIP: None
  selector:
    app: nginx
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  selector:
    matchLabels:
      app: nginx # has to match .spec.template.metadata.labels
  serviceName: "nginx"
  replicas: 2 # by default is 1
  podManagementPolicy: OrderedReady
  template:
    metadata:
      labels:
        app: nginx # has to match .spec.selector.matchLabels
    spec:
      terminationGracePeriodSeconds: 10
      containers:
      - name: nginx
        image: nginx:1.16
        ports:
        - containerPort: 80
          name: web
```

kubectl apply -f nginx-statefullset.yml

kubectl get svc

```
[root@ansikube manifest]# kubectl get svc
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.36.0.1    <none>        443/TCP   56m
nginx        ClusterIP   None         <none>        80/TCP    4m12s
```

kubectl get sts -o wide

```
[root@ansikube manifest]# kubectl get sts -o wide
NAME   READY   AGE   CONTAINERS   IMAGES
web    2/2     28s   nginx        nginx:1.16
```

kubectl get pods -o wide

```
[root@ansikube manifest]# kubectl get pods -o wide
NAME    READY   STATUS    RESTARTS   AGE     IP          NODE                                        NOMINATED NODE   READINESS GATES
web-0   1/1     Running   0          5m22s   10.32.0.5   gke-robo-default-pool-682616c4-ml55   <none>           <none>
web-1   1/1     Running   0          5m14s   10.32.1.5   gke-robo-default-pool-682616c4-ldd4   <none>           <none>
```

Note:- Here pod names are created in order and uniqueness because we have mentioned podManagementPolicy: OrderedReady in statefulset parameter, which is default.

kubectl describe sts web

```
[root@ansikube manifest]# kubectl describe sts web
Name:               web
Namespace:          default
CreationTimestamp:  Wed, 06 Nov 2019 00:52:23 +0000
Selector:           app=nginx
Labels:             <none>
Annotations:        kubectl.kubernetes.io/last-applied-configuration:
                      {"apiVersion":"apps/v1","kind":"StatefulSet","metadata":{"annotations":{},"name":"web","namespace":"default"},"spec":{"podManag
ementPolicy...
Replicas:           2 desired | 2 total
Update Strategy:    RollingUpdate
  Partition:        824638022300
Pods Status:        2 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  app=nginx
  Containers:
   nginx:
    Image:        nginx:1.16
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Volume Claims:    <none>
Events:
  Type    Reason            Age    From                    Message
  ----    ------            ----   ----                    -------
  Normal  SuccessfulCreate  6m15s  statefulset-controller  create Pod web-0 in StatefulSet web successful
  Normal  SuccessfulCreate  6m7s   statefulset-controller  create Pod web-1 in StatefulSet web successful
```

Note:- If you notice above output, update strategy is become RollingUpdate and which is default to stateful set.

The RollingUpdate update strategy implements automated, rolling update for the Pods in a StatefulSet. It is the default strategy when .spec.updateStrategy is left unspecified. When a StatefulSet's .spec.updateStrategy.type is set to RollingUpdate, the StatefulSet controller will delete and recreate each Pod in the StatefulSet. It will proceed in the same order as Pod termination (from the largest ordinal to the smallest), updating each Pod one at a time. It will wait until an updated Pod is Running and Ready prior to updating its predecessor.