StatefulSet

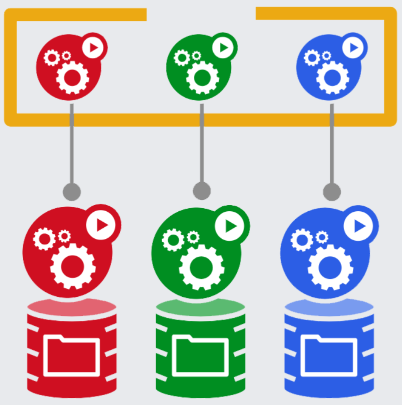
AUTOPILOT (/KUBERNETES-ENGINE/DOCS/CONCEPTS/AUTOPILOT-OVERVIEW)

STANDARD (/KUBERNETES-ENGINE/DOCS/CONCEPTS/TYPES-OF-CLUSTERS)

This page describes the use of StatefulSet objects in Google Kubernetes Engine (GKE). You can also learn how to Deploy a stateful application (/kubernetes-engine/docs/how-to/stateful-apps).

Why use StatefulSets

You can use StatefulSets to deploy stateful applications



(/kubernetes-engine/docs/how-to/deploying-workloads-overview#stateful_applications) and clustered applications that save data to persistent storage, such as Compute Engine persistent disks (/compute/docs/disks#pdspecs). StatefulSets are suitable for deploying Kafka, MySQL, Redis, ZooKeeper, and other applications needing unique, persistent identities and stable hostnames.

StatefulSets represent a set of Pods (https://kubernetes.io/docs/concepts/workloads/pods/) with unique, persistent identities, and stable hostnames that GKE maintains regardless of where they are scheduled. StatefulSets use a Pod template (https://kubernetes.io/docs/concepts/workloads/pods/#pod-templates), which contains a specification for its Pods. The state information and other resilient data for any given StatefulSet Pod is maintained in persistent volumes (/kubernetes-engine/docs/concepts/persistent-volumes) associated with each Pod in the StatefulSet. StatefulSet Pods can be restarted at any time.

For stateless applications (/kubernetes-engine/docs/how-to/deploying-workloads-overview#stateless_applications), use Deployments (https://kubernetes.io/docs/concepts/workloads/controllers/deployment/).

StatefulSet indexing

StatefulSets use an ordinal index for the identity and ordering of their Pods. By default, StatefulSet Pods are deployed in sequential order and are terminated in reverse ordinal order. For example, a StatefulSet named `web` has its Pods named `web-0`, `web-1`, and `web-2`. When the `web` Pod specification is changed, its Pods are gracefully stopped and recreated in an ordered way; in this example, `web-2` is terminated first, then `web-1`, and so on. Alternatively, you can specify the `podManagementPolicy: Parallel` (https://v1-25.docs.kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#statefulsetspec-v1-apps) field to have a StatefulSet launch or terminate all of its Pods in parallel, rather than waiting for Pods to become Running and Ready (https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/) or to be terminated prior to launching or terminating another Pod.

Plan networking for StatefulSets

StatefulSets provide persistent storage in the form of a Persistent Volume and a unique network identity (hostname). The following table includes the caveats that application operators should be aware of when configuring a StatefulSet:

Networking caveat	Description	Best practice
GKE Services instead of fixed IP addresses	Although Pod replicas have a unique ordinal index, support per-replica volumes, and network identity (hostname), the IP addresses that are assigned to a replica can change if a pod is rescheduled	To mitigate networking issues, the architecture should use Kubernetes Service resources. For more information, see Types of Kubernetes Services (https://cloud.google.com/kubernetes-engine/docs/concepts/service#types-of-services). **Note:** Applications on Kubernetes may cache IP addresses if other components are installed. For example, JVM caches name lookups (https://docs.oracle.com/en/java/javase/18/docs/api/java.base/java/net/doc-files/net-properties.html#networkaddress.cache.ttl) indefinitely by default if a security manager is installed. As an application developer, you should audit your application to ensure that it does not cache IP addresses that are resolved from these services.

	or evicted by GKE.	
Headless Services	When initialized, a StatefulSet is paired with a matching headless service.	Ensure that the `metadata.name` in your Service matches the `serviceName` field in your StatefulSet. This enables each Pod in your application to be addressed at a unique, well-defined network address. Additionally, the headless service provides a multi-IP record for each replica in your StatefulSet, allowing full peer discovery.
Peer discovery	Stateful applications require a minimum number (quorum) of replicas to function with full availability.	As Pods can crash, be rescheduled, or evicted, each replica in a StatefulSet should be able to leave and rejoin quorum. Applications that require peering should have the capability to discover other peers through headless Services in Kubernetes.
Health check based on readiness probes and liveness probes	Your application should have properly configured readiness, liveness, and startup probes where applicable. Selecting timeouts for each probe is dependent on the requirements of your application.	For readiness probes, follow these best practices to configure your application to mark readiness when it is ready to serve traffic: • **Liveness probes:** You can use liveness probes to signal if a container is healthy. For example, a database replica can use a liveness probe to indicate that GKE should restart the replica, such as deadlock condition • **Readiness probes:** You can use readiness probes to remove a replica from serving traffic temporarily. For example, if you have a database replica that needs to perform a backup, you might use a readiness probe to temporarily stop receiving requests. • **Startup probe:** You can use startup probes to delay health checks until long running initializations are complete. For example, if you have a database replica, you might use a startup probe to wait for initialization of stored data from disk.

To read more about probes, see Configure Liveness, Readiness, and Startup Probes (https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/#define-readiness-probes)
.

Creating StatefulSets example

You can create a StatefulSet (/kubernetes-engine/docs/how-to/stateful-apps#creating_a_statefulset) using `kubectl apply` (https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands#apply).

Once created, the StatefulSet ensures that the desired number of Pods are running and available at all times. The StatefulSet automatically replaces Pods that fail or are evicted from their nodes, and automatically associates new Pods with the storage resources, resource requests and limits, and other configurations defined in the StatefulSet's Pod specification.

Note: To prevent data loss, PersistentVolumes and PersistentVolumeClaims are not deleted when a StatefulSet is deleted. You must manually delete these objects using **`kubectl delete pv`** and **`kubectl delete pvc`**.

The following is an example of a Service and StatefulSet manifest file:

```
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 # Label selector that determines which Pods belong to the Sta
                 # Must match spec: template: metadata: labels
  serviceName: "nginx"
  replicas: 3
```

```
template:
  metadata:
    labels:
      app: nginx # Pod template's label selector
  spec:
    terminationGracePeriodSeconds: 10
    containers:
    - name: nginx
      image: registry.k8s.io/nginx-slim:0.8
      ports:
      - containerPort: 80
        name: web
      volumeMounts:
      - name: www
        mountPath: /usr/share/nginx/html
volumeClaimTemplates:
- metadata:
    name: www
  spec:
    accessModes: [ "ReadWriteOnce" ]
    resources:
      requests:
        storage: 1Gi
```

In this example:

- A Service (/kubernetes-engine/docs/concepts/service) object named `nginx` is created, indicated by the `metadata: name` field. The Service targets an app called `nginx`, indicated by `labels: app: nginx` and `selector: app: nginx`. The Service exposes port 80 and names it `web`. This Service controls the network domain and to route Internet traffic to the containerized application deployed by the StatefulSet.
- A StatefulSet named `web` is created with three replicated Pods (`replicas: 3`).
- The Pod template (`spec: template`) indicates that its Pods are labelled `app: nginx`.
- The Pod specification (`template: spec`) indicates that the StatefulSet's Pods run one container, `nginx`, which runs the `nginx-slim` image at version `0.8`.
- The Pod specification uses the `web` port opened by the Service.
- `template: spec: volumeMounts` specifies a `mountPath`, which is named `www`. The `mountPath` is the path within the container at which a storage volume should be mounted.

- The StatefulSet provisions a PersistentVolumeClaim (/kubernetes-engine/docs/concepts/persistent-volumes), `www`, with 1GB of provisioned storage.

In summary, the Pod specification contains the following instructions:

- Label each Pod as `app: nginx`.
- In each Pod, run one container named `nginx`.
- Run the `nginx-slim` image at version `0.8`.
- Have Pods use port `80`.
- Save data to the mount path.

For more information about StatefulSet configurations, refer to the StatefulSet API reference (https://v1-25.docs.kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/#statefulset-v1-apps)
.

Updating StatefulSets

You can update a StatefulSet (/kubernetes-engine/docs/how-to/stateful-apps#updating_a_statefulset) by making changes to its Pod specification, which includes its container images and volumes. You can also update the object's resource requests and limits, labels, and annotations. To update a StatefulSet, you can use `kubectl`, the Kubernetes API, or the GKE Workloads menu (/kubernetes-engine/docs/concepts/dashboards#workloads) in the Google Cloud console.

To decide how to handle updates, StatefulSets use an *update strategy* defined in `spec: updateStrategy`. There are two strategies, `OnDelete` and `RollingUpdate`:

- `OnDelete` does not automatically delete and recreate Pods when the object's configuration is changed. Instead, you must manually delete the old Pods to cause the controller to create updated Pods.
- `RollingUpdate` automatically deletes and recreates Pods when the object's configuration is changed. New Pods must be in Running and Ready (https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/) states before their predecessors are deleted. With this strategy, changing the Pod specification automatically triggers a rollout. This is the default update strategy for StatefulSets.

StatefulSets update Pods in reverse ordinal order. You can monitor update rollouts by running the following command:

```
kubectl rollout status statefulset STATEFULSET_NAME
```

Partitioning rolling updates

You can partition (https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/#partitions) rolling updates. Partitioning is useful if you want to stage an update, roll out a canary, or perform a phased roll out.

When you partition an update, all Pods with an ordinal greater than or equal to the partition value are updated when you update the StatefulSet's Pod specification. Pods with an ordinal less than the partition value are not updated and, even if they are deleted, are recreated using the previous version of the specification. If the partition value is greater than the number of replicas, the updates are not propagated to the Pods.

What's next

- Learn how to deploy a stateful application (/kubernetes-engine/docs/how-to/stateful-apps).
- Learn more about deploying workloads in GKE (/kubernetes-engine/docs/how-to/deploying-workloads-overview).
- Read more about StatefulSets in the Kubernetes documentation (https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/).
- Read about StatefulSet basics (https://kubernetes.io/docs/tutorials/stateful-application/basic-stateful-set/).
- Take a tutorial about upgrading a cluster running a stateful workload (/kubernetes-engine/docs/tutorials/upgrading-stateful-workload).

Last updated 2023-12-04 UTC.