How to use the standard Helm Charts

Overview

To standardize how microservices are deployed into Kubernetes, the RDE Deploy team manages a repository of Helm Charts for WBD applications. This document includes a FAQ for how to best use the microservice Helm chart.

FAQ

Helm Chart Logistics Questions

Where do I find documentation for the Helm Chart?

Answer...

The README in the helm-chart repository contains a table showing all the configurable variables the chart uses, along with the default value, and description/example. Clicking on the hyperlink in the `Key` will take you to where that value is in the values file. Be sure to scroll the window to the right to view the description/example

How can I change the Helm Chart version for my microservice?

Answer...

As part of the RDE release and deploy workflow, your microservice is deployed with a helm chart that has some version associated with it. This version can be specified for your entire service and can be specified for individual components. To do this, you must have the `helm-chart` and `helm-chart-version` `make` targets defined in your Makefile which are responsible for outputting a string representing the name and semver of the helm chart to use respectively. See the example set up in wbd-golang-template repository. All components will use the values defined in the root Makefile if they are not defined in its Makefile. **We highly recommend you using the latest available Helm chart version by setting the version value to** `^0` (example). This is the default setup from `wbd-golang/java-template`.

How can I render a Helm Chart locally?

Answer...

Follow the guide here on pulling the `microservice` helm chart locally from Artifactory and testing it against you values files.

Common Configuration Questions

How can I utilize an AWS role to access AWS infrastructure?

- Helm Chart Deployment Failure due to Inability to Access AWS Resources

How can I specify my own readiness, liveness, or startup probe block?

Answer...

You can use the `.probe.[liveness|readiness|startup].override` to specify a full Kubernetes probe block, the documentation for which you can find here Configure Liveness, Readiness and Startup Probes. For example to use a custom liveness probe but the standard readiness probe you can specify something like the following in your values file:

```yaml
probe:
  liveness:
    override:
      httpGet:
        path: /livez
        port: 9090
```

```
        httpHeaders:
        - name: Custom-Header
          value: Awesome
      initialDelaySeconds: 3
      periodSeconds: 3
  readiness:
    path: /healthz
    initialDelaySeconds: 2
```

This will render out:

```
---
# Source: microservice/templates/deployment.yaml
....
      containers:
        ...
        livenessProbe:
          httpGet:
            httpHeaders:
            - name: Custom-Header
              value: Awesome
            path: /healthz
            port: 9090
          initialDelaySeconds: 3
          periodSeconds: 3
        readinessProbe:
          httpGet:
            path: /healthz
            port: 8080
          initialDelaySeconds: 2
...
```

How do I set up Ingress?

Answer...

Refer here: https://wbdstreaming.atlassian.net/wiki/spaces/PEO/pages/89522269/BOLT+Onboarding+Handbook#10.-Boundary-Services-details

How do I enable gRPC?

Answer...

Set the following value.

```
enableGrpc: true
```

Also see: How do I change the container port and service ports?

How do I change the container port and service ports?

Answer...

To change container and service port configurations, your values file must contain the following:

```
containerPort:   # Container Port configuration
  http: 8080
  grpc: 18080

service:        # Service port configuration
```

```
  type: ClusterIP
  ports:
    http: 80
    grpc: 180
```

How do I enable HPA and set CPU/memory threshold?

Answer...

HPA (see documentation) can be configured with the following values:

```
autoscaling:
  enabled: true
  minReplicas: 1
  maxReplicas: 100
  targetCPUUtilizationPercentage: 80
  targetMemoryUtilizationPercentage: 80
```

Both `targetCPUUtilizationPercentage` and `targetMemoryUtilizationPercentage` are optional and will create an element of `type: Resource` under `HorizontalPodAutoscaler.spec.metrics`.

IMPORTANT: A *resource request* **MUST** be defined for the HPA to work. The utilization percentage is calculated based on the amount of cpu/memory requested. You can set a resource block with the following values:

```
# example
resources:
  requests:
    memory: "64Mi"
    cpu: "250m"
  limits:
    memory: "128Mi"
    cpu: "500m"
```

NOTE: `targetCPUUtilizationPercentage` is always set in the `autoscaling` value. To unset it, set `targetCPUUtilizationPercentage: null`

How do I configure HPA scaleUp/scaleDown behavior?

Answer...

VERSION 0.16.1 AND LATER

A full HPA behavior block can be defined under `autoscaling`. Example:

```
autoscaling:
  enabled: true
  minReplicas: 1
  maxReplicas: 100
  targetCPUUtilizationPercentage: 80
  behavior:
    scaleDown:
      policies:
      - type: Percent
        value: 10
        periodSeconds: 60
    scaleUp:
      stabilizationWindowSeconds: 2
      policies:
      - type: Percent
        value: 100
        periodSeconds: 15
```

```
      selectPolicy: Max
```

How do I configure HPA with custom metrics?

Answer...

SRE team has started rolling out KEDA to support this capability. To learn more, click here: KEDA - HELM Configuration

NO LONGER SUPPORTED

This was previously implemented by a tool that's no longer supported by newer versions of Kubernetes. The SRE Team is currently review a tool called KEDA to bring back custom autoscaling capabilities

---

OLD DOCUMENTATION

The chart supports HPA scaling on other metrics besides CPU and memory. See this walkthrough on how to autoscale on multiple metrics and custom metrics. Users can create metrics for type `Pod` , `External` , and `Object` using the values `podMetrics` , `externalMetrics` , and `objectMetrics` . These values expect a list of a block for `metrics.(pods|external|object).*` . See API docs for what different metric types require.

Example values.yaml

```yaml
autoscaling:
  enabled: true
  minReplicas: 2
  maxReplicas: 30
  targetCPUUtilizationPercentage: null
  podMetrics:                              # Two dif
  - metric:
      name: packets_per_second
    target:
      type: AverageValue
      averageValue: 1k
  - metric:
      name: some_other_pod_metric
    target:
      type: AverageUtilization
      averageUtilization: 50

  externalMetrics:                         # One 'Ex
  - metric:
      name: queue_messages_ready
      selector:
        matchLabels:
          queue: "worker_tasks"
    target:
      type: AverageValue
      averageValue: 30

  objectMetrics:                           # One 'Ob
  - metric:
      name: requests-per-second
    describedObject:
      apiVersion: networking.k8s.io/v1
      kind: Ingress
```

Rendered hpa.yaml by Helm

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  ...
spec:
  ...
  minReplicas: 2
  maxReplicas: 30
  metrics:
    - type: Pods
      pods:
        metric:
          name: packets_per_second
        target:
          averageValue: 1k
          type: AverageValue
    - type: Pods
      pods:
        metric:
          name: some_other_pod_metric
        target:
          averageUtilization: 50
          type: AverageUtilization
    - type: External
      external:
        metric:
          name: queue_messages_ready
          selector:
            matchLabels:
              queue: worker_tasks
        target:
          averageValue: 30
          type: AverageValue
```

```
      name: main-route
    target:
      type: Value
      value: 10k
```

```
  - type: Object
    object:
      describedObject:
        apiVersion: networking.k8s.io/v1
        kind: Ingress
        name: main-route
      metric:
        name: requests-per-second
      target:
        type: Value
        value: 10k
```

How do I create a Job and CronJob?

Answer...

Both CronJobs and Jobs are set with the same way. A list of batch jobs are defined in the `cronJobs` value. If the job contains a schedule, it runs as a cronJob, else, it runs as a job. The values that can be set are listed below:

Value	Required	Description	Default
`name`	Yes	Set to `.metadata.name`	None
`schedule`	No	CronJob Schedule. If omitted, creates a Job	None
`env`	No	Set environment variables in the pod	None
`envFieldRefs`	No	Set pod information as environment variables	None
`args`	No	Arguments to run within a container	None
`successfulJobsHistoryLimit`	No	Sets `.spec.successfulJobsHistoryLimit`	`5`
`failedJobsHistoryLimit`	No	Sets `.spec.failedJobsHistoryLimit`	`5`
`suspend`	No	Sets `.spec.suspend`	`false`
`restartPolicy`	No	Sets `.spec.restartPolicy` within the Job template	`OnFailure`
`jobAnnotations`	No	Sets `.metadata.annotations` within the Job template	None

Below is an example on how to create a cronJob “my-cronjob” and Job “my-job”

```
# Toggle if you wish to turn off Deployments and Services
deploymentEnable: false
serviceEnable: false

cronJobs:
```

```yaml
- name: "my-cronjob"                        # CronJob Creation
  schedule: "* * * * *"
  env:
    NAME: val
  envFieldRefs:
    MY_POD_NAMESPACE:
      apiVersion: v1
      fieldPath: metadata.namespace
  successfulJobsHistoryLimit: 10
  failedJobsHistoryLimit: 10
  suspend: true
  restartPolicy: OnFailure
  jobAnnotations:
    annotation1: value
- name: "my-job"                           # Job Creation
  env:
    NAME_1: jobVal
  args:
  - argVal1
  - argVal2
  restartPolicy: Never
  jobAnnotations:
    annotation1: val1
```

How do I expose pod information to my microservice?

Answer...

You can set pod information as environment variables. See documentation: Downward API . To use this in the microservice Helm chart, use the `envFieldRefs` value.

Example:

```yaml
envFieldRefs:
  HOSTNAME:
    apiVersion: v1
    fieldPath: status.podIP
  MY_POD_NAMESPACE:
    apiVersion: v1
    fieldPath: metadata.namespace
```

This sets the the `HOSTNAME` variable as the pod's IP address in and the `MY_POD_NAMESPACE` variable as the namespace. Including the `apiVersion` is optional. It defaults to `v1`

How can I set a JSON or YAML as an environment variable?

Answer...

The chart supports a `toJson` and `toYaml` feature for environment variables.

Example 1:

```yaml
env:
  REGION: "us-east-1"
  toYaml:
    APP_YAML_CONFIG:
      one: 1
      two: 2
    ANOTHER_APP_YAML_CONFIG:
      five: 5
      six: 6
```

```
  toJson:
    SPRING_APPLICATION_JSON:
      {"seven": "7"}
    ANOTHER_SPRING_APPLICATION_JSON:
      {"eight": "8"}
```

How do I change the number of pods running for my microservice?

Answer...

Set `replicaCount` to the desired number

Example:

```
replicaCount: 10
```

Note: this doesn't take effect when `autoscaling` is enabled.

How can I inject a file into a configMap?

Answer...

Configuration files must be embedded within helm values files. As part of the build-microservice-image-and-push workflow, you can specify a path in your repository, and that file will be injected in into the values files prior to being uploaded to the values files being uploaded to Artifactory.

Values files can specify a `fileData` field in the configMap like so:

```
configMap:
  enable: true
  mountPath: /mount/path
  data:
    example_data: example_val
  fileData:
    - key: conf.json
      path: "../config.json"


```

The workflow will update the values file to be:

```
configMap:
  enable: true
  mountPath: /mount/path
  data:
    conf.json: |-
      {
          "data": "value"
      }
     example_data: example_val
  fileData:
    - key: conf.json
      path: "../config.json"

```

The `fileData` field isn't used at all by the helm chart itself.