Solace OpenShift Template Samples

This document is intended to describe several OpenShift Templates that are available as Samples to deploy Solace PubSub+ brokers into the OpenShift Cloud.

solace

Table of Contents

1 Introduction

Moving to the Cloud and applying Container Management has become a very important shift for many of today's application deployments. There are many container frameworks available. One of the most popular is Kubernetes, which is an Open Source container-orchestration system that manages application containers based on the Docker Engine.

A popular commercial container orchestration implementation is the OpenShift product that provides Docker image management via Kubernetes on RedHat Enterprise Linux. OpenShift Origin (now known as OKD) is the upstream community OpenShift distribution. OpenShift has enhanced the Kubernetes software to make it more "industrial-strength" by improving security and adding stronger multi-tenancy capabilities. OpenShift also provides a powerful application development environment with its Source-to-Image (S2I) feature that allows developers to deploy complete running Docker containers in OpenShift directly from the application's source code.

Although OpenShift is based on Kubernetes, the application templates that are used to deploy the containers are not exactly the same between these similar technologies. However, the different template implementations are more similar than different.

This document describes several sample templates that can be used to deploy Solace PubSub+ (PS+) brokers into the OpenShift managed container environment. OpenShift (and Kubernetes) have some advanced capabilities that are expressed through image deployment via template files. When deploying Solace PubSub+ brokers, there is no "recommended" or "silver bullet" template configuration. The requirements can range from a simple single PS+ broker for local development to a full Stateful Set deployment with load-balanced OpenShift Services for local and remote access.

This GitHub repository includes an expanding set of template examples that demonstrate increasingly complex (feature-rich), template samples. Each sample builds on the previous sample; therefore, It should be possible to create production-level Solace PS+ templates for production deployment based on a hybrid of the included samples.

If you are quite familiar with OpenShift you can jump right to the templates of interest. If you are interested in learning more details about Solace in an OpenShift cluster, it is recommended to follow this document from the beginning as the template samples sequentially add more features and functionality as they relate to Solace within OpenShift.

There is also a sample producer/consumer application included to show how the S2I can be used to deploy a Solace client application against the deployed PS+ brokers that are running as containers in the OpenShift environment. It is important to note that applications deployed in OpenShift can access any PS+ broker (software, appliance or Solace Cloud). There is no requirement for the applications and brokers to both exist in an OpenShift cluster. Also, applications that are not in OpenShift can access PS+ brokers that are deployed as docker containers in OpenShift.

1.1 Solace OpenShift QuickStart

It is important to note that there are additional details available on GitHub that describe the deployment of Solace in OpenShift. These details can be found here:

https://github.com/SolaceProducts/solace-openshift-quickstart

The purpose of these alternate examples is to provide detail on deploying an OpenShift/Solace Quick Start and the creation of a "production-like" deployment of Solace in OpenShift. The Cloud Environment described for OpenShift Quick Start is based on a deployment in AWS.

There are several deployment options available for the Solace Quick Start, including Helm Charts.

The Quick Start repository assumes the user already has an advanced understanding of OpenShift and Solace and is looking for a quick and simple mechanism for deploying a production-like Solace environment in AWS/OpenShift. This document will focus on users that may only have a rudimentary understanding of Solace and OpenShift.

The Quick Start also describes testing the Solace PS+ deployment after it is installed in OpenShift. This is also applicable to testing the deployment of Solace using the samples that are part of this GitHub repository. Therefore, it is recommended to scan the Quick Start documentation and repository for suggestions on testing that will not be repeated in this document.

1.2 Dependencies and Requirements

To make this document a bit simpler to follow and to make it easier to understand the templates, all the template samples assume the following defaults:

- OpenShift Project Name: solace
- OpenShift Username: solace
- OpenShift Password: solace
- OpenShift PS+ Container name: vmr
- PS+ admin username: admin
- PS+ admin password: admin

When using the OpenShift GUI to deploy the templates it is possible to change the appropriate information to match your specific test environment at deployment time. However, for simplicity, configuring the OpenShift and Solace PS+ environments with the same Project and Client details will allow direct deployment of the templates without changing the default optional parameters.

All the templates have been tested using Solace PS+ version 9.3.X. This version has been released to support the most restrictive (and default) OpenShift Security Context Constraints (SCC). This allows the templates to deploy with the "privilege: false" container specification setting. Earlier versions of PS+ require either privileged access or a custom SCC to be deployed and reflected in the template. (The topic of custom SCC is described in the aforementioned Solace/OpenShift Quick Start repository and will not be discussed further in this document.)

The templates have been tested with OpenShift 3.11 and MiniShift v1.28 (based on OpenShift 3.11). Earlier versions of OpenShift also support Solace (version 3.7 and higher) but these specific templates were not tested in the earlier versions but should still operate correctly. Please post a GitHub issue if you are having problems with an earlier version of OpenShift.

MiniShift is an excellent single-node "cluster" software that can be used to test and build OpenShift templates and applications.

While the setup and deployment of OpenShift is beyond the scope of this document, there will be some discussion of issues when deploying OpenShift in the Cloud. One of the most common mechanisms for easily installing OpenShift in the Cloud for testing or production is to use the AWS OpenShift Quick Start Cloud Formation Template. We will point out some issues that are cloud-vendor specific, but for the most cases, it is assumed the reader is already familiar with their own OpenShift environment.

All the templates can be run from the OpenShift Command Line Interface; however, for the purpose of discussion we will mostly be using the simple GUI interface and the OpenShift Template Repository.

2 Solace Singleton Persistent Pod Template

The Solace Singleton template is a very basic template that is used to deploy a single Solace PS+ broker into OpenShift as a Pod. It can be useful for application development or simple testing of Solace applications or PS+ non-clustered configurations. It includes both ClusterIP and NodePort Services that will allow access to the PS+ broker both within the OpenShift environment and externally as well. The template makes use of persistent storage so that if the Pod is accidentally deleted or stopped/restarted, the persisted messages and the PS+ configuration are not lost.

In this GitHub repository, this template is called:

```
sol-single-ps+-persist-pod.yml
```

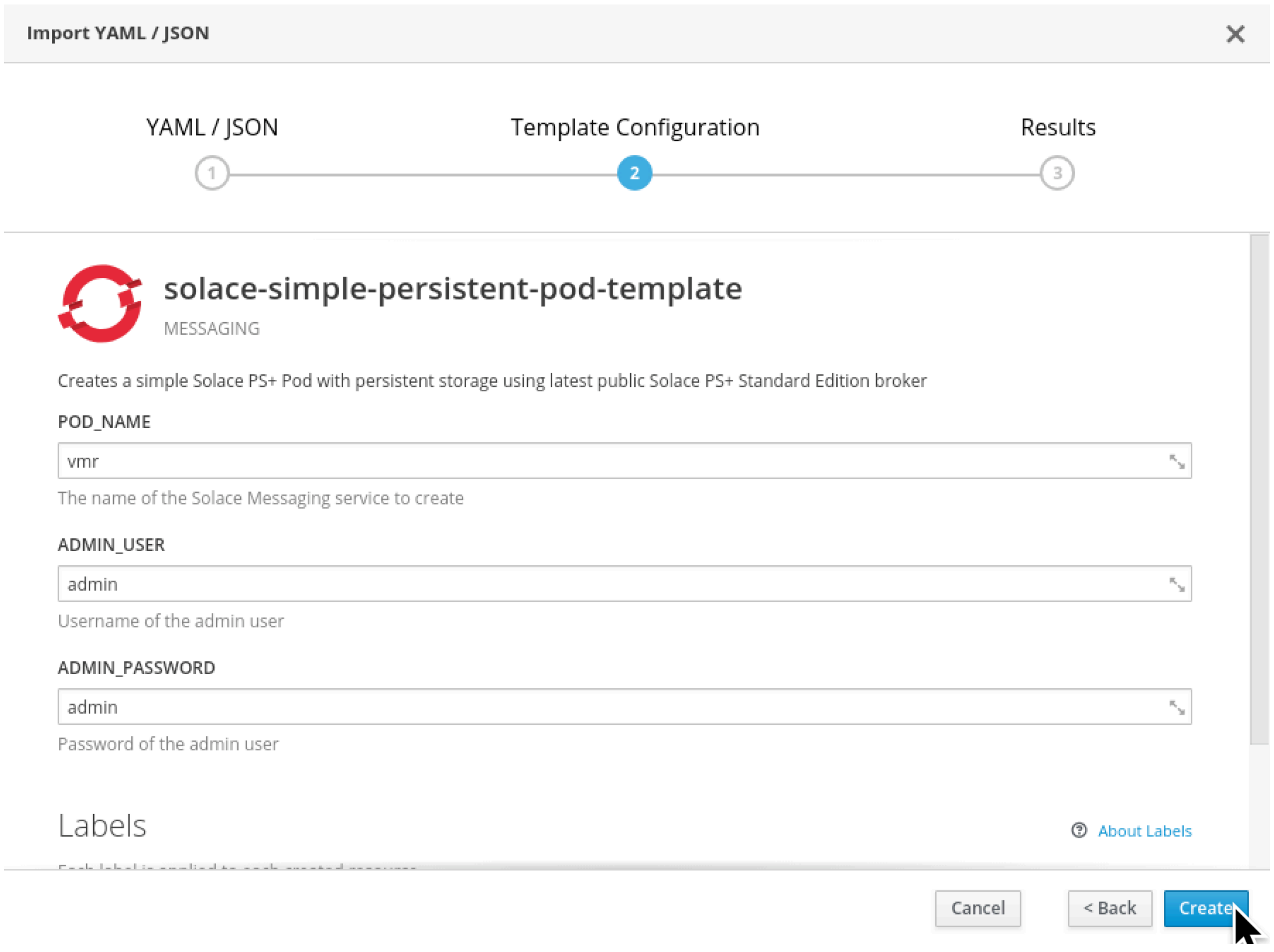
2.1 Singleton Template Parameters

To simplify the PS+ deployment, the template has incorporated "parameters" into the template, which can be found at the end of the YAML file:

```
parameters:
- description: The name of the Solace Messaging service to create
  from: '[a-z0-9]{40}'
  generate: expression
  name: POD_NAME
  value: vmr
- description: Username of the admin user
  from: '[a-zA-Z0-9]{16}'
  generate: expression
  name: ADMIN_USER
  value: admin
- description: Password of the admin user
  from: '[a-zA-Z0-9]{16}'
  generate: expression
  name: ADMIN_PASSWORD
  value: admin
```

In this case, the Web GUI will display the optional parameters. The displayed "value" is the default that can be over-ridden at deployment time. The use of template parameters allows these values (default or overridden value) to be used with macro-substitution when the template is deployed. Within the template the macro-substitution placeholders take the form: ${name_of_parameter}.

Below is a sample screenshot for the parameters fields as outlined for this first sample template.



solace-simple-persistent-pod-template
MESSAGING

Creates a simple Solace PS+ Pod with persistent storage using latest public Solace PS+ Standard Edition broker

POD_NAME: vmr
The name of the Solace Messaging service to create

ADMIN_USER: admin
Username of the admin user

ADMIN_PASSWORD: admin
Password of the admin user

It is also possible to deploy the singleton template from the command line. First it is necessary to get all the expected parameters that are defined in the template and their value restrictions. For example:

```
oc process --parameters -f sol-single-ps+-persist-pod.yml
NAME                DESCRIPTION                                  GENERATOR           VALUE
POD_NAME            The name of the Solace Messaging service to create   expression          [a-z0-9]{40}
ADMIN_USER          Username of the admin user                   expression          [a-zA-Z0-9]{16}
ADMIN_PASSWORD      Password of the admin user                   expression          [a-zA-Z0-9]{16}
```

According to the template definition shown above the default “POD_NAME” is “vmr”. If you want to deploy the template with the POD_NAME=vmr1 then the following CLI command can be used:

```
oc process -f sol-single-ps+-persist-pod.yml -p POD_NAME=vmr1 | oc create -f -
```

The “oc process” command will process the template and output the final template with all the “POD_NAME” variable substations. This output is then piped to “oc create” with the input filename of standard input (or the shell shortcut of “-“).

2.2 Singleton Template Container Definitions

The singleton Solace template also defines details for the Docker container that is installed into the OpenShift managed container service:

```
spec:
    containers:
      - env:
        - name: username_${ADMIN_USER}_password
          value: ${ADMIN_PASSWORD}
        - name: username_${ADMIN_USER}_globalaccesslevel
          value: admin
        - name: service_ssh_port
          value: "2222"
        - name: service_webtransport_port
          value: "60080"
        - name: service_webtransport_tlsport
          value: "60443"
        - name: service_semp_tlsport
          value: "60943"
        - name: logging_debug_output
          value: stdout
        - name: logging_kernel_output
          value: stdout
        name: ${POD_NAME}
        image: 'solace/solace-pubsub-standard'
```

Note the extensive use of variable substitutions based on parameters defined in the template as described earlier in Section 2.1. The templates also make use of the Solace Configuration Key capabilities. Solace Configuration Keys are system environment variables that are used by the Solace Docker image when it is first created as a Docker container. The Configuration Keys make it easy to initialize many of the Solace PS+ configuration parameters by passing container environment variables via the OpenShift template. The complete list of available Configuration Keys and options is available here:
https://docs.solace.com/Configuring-and-Managing/SW-Broker-Specific-Config/Docker-Tasks/Config-SW-Broker-Container-Cfg-Keys.htm#Env-Var

With the simple singleton example, the template environment variables are used to set the default administrator username and password for the Solace PS+ container. Several of the Solace service TCP ports are also initialized. The output for the debug log and the kernel log are both set to use standard output (stdout), which means they are available and interleaved with the all the OpenShift logs.

Finally, the Solace image to install into a Docker Container is referenced against the “image:” tag. In this case, “solace/solace-pub-sub-standard”, means the latest free Solace public Docker image that is stored in the Docker public repository will be automatically be downloaded and used in the template.

2.3 Solace Service TCP Ports

The Solace PS+ services are exposed as TCP ports that must be defined in the Docker Container for them to be accessible to the Solace image. As mentioned in the previous section, the PS+ broker image can be configured automatically as part of the container creation via the Solace Configuration Keys. Using the Configuration Keys as part of the OpenShift template allows for automatic Solace configuration of the required Solace ports.

However, it is also necessary to have the Docker Container configured to expose the Docker Container TCP ports as part of the Container security. While the Configuration Keys shown in the previous section define the Solace image configuration, the following is required in configuring those same ports in Docker:

```
ports:
  - containerPort: 60080
    protocol: TCP
  - containerPort: 8080
    protocol: TCP
  - containerPort: 55555
    protocol: TCP
  - containerPort: 2222
    protocol: TCP
```

As described in Section 1.2, the Solace v9.2+ image automatically supports the use of the OpenShift Restricted SCC. This means that the Solace Image container is run without "root" administrative privileges. Root administrative privilege is required to start TCP listening services that make use of TCP ports below 1024. Therefore, all Solace services ports must be assigned above TCP port 1024. To deal with the default OpenShift Restricted SCC security (which is the most restrictive security configuration) the template segment described in the previous section shows the typical TCP service ports that are normally assigned below 1024 as prefaced with "60" (any preface to make the number greater than 1024 will work) or the Solace container will fail to deploy. For example, the default TLS port is 443, but for this Solace Docker Container and Solace PS+ service, it will be exposed as 60443.

It is possible to use templates with default ports that require root privileges for the Solace PS+ container, but not without either making use of a customized OpenShift SCC or having the OpenShift administrator provide a Project Service Account (or a Project User) with OpenShift administrator privileges (and of course, setting the template to allow OpenShift Privilege mode -- i.e. deploy the container with "root" privileges). However, for most Solace customer deployments, setting PS+ TCP ports with a socket numbered below 1024 is not going to be required or desired (in fact, with some security administrators, it will be forbidden).

To ensure the strongest OpenShift security via the Restricted SCC, the template contains the entry:

```
securityContext:
  privileged: false
```

With the default "restricted SCC" OpenShift security definition, the Solace container service ports must use TCP ports above 1024 and, of course, ports that do not conflict with other Docker ports.

2.4 Solace Message Persistence and Configuration Persistence

Solace PS+ requires OpenShift persistent storage for unprocessed Queue messages and the storage of Solace configuration data as defined for use against the Solace Control Plane processes. The Solace templates support both the use of dynamic Persistent Volume Claims (PVC) or the older mechanism where Persistent Volumes (PV) are defined in a pool and the PVC is pulled into a specific project by claiming a PV from the cluster PV Pool. All the sample Solace templates assume that dynamic PVC creation is available, and the default Storage Class is used to define the OpenShift volumes that will be mounted by the Solace PS+ images in the Docker Container.

If you make use of Minishift to run the Solace PS+ templates, a PV pool is automatically available and is used to create the PVCs.
OpenShift has three levels of configuration required for persistent storage:

- Claim a Persistent Volume (PV)
- Associate the Persistent Volume Claim (PVC) with the Docker Volumes
- Create the disk mounts in the Solace Container based on the Docker Volumes.

First, there is a requirement to claim a PV. The Persistent Volume is usually an OpenShift reference to a local disk volume. For example, it you deployed OpenShift in AWS, the PV would be a reference to, for example, an AWS GP2 disk. Using dynamic PVC, the OpenShift Storage Class would automatically create the underlying disk volume (e.g. a GP2 Volume) and then associate a PV to the physical volume and then claim that new PV to create the PVC.
In the template, the PVC is defined with the following:

```
- kind: PersistentVolumeClaim
  apiVersion: v1
  metadata:
    name: poddata-${POD_NAME}
  spec:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: 64Gi
```

The template defines a request for OpenShift to create a PVC of size 64 Gig. The OpenShift Storage Class (which is available as part of the OpenShift installation) will define the automatic creation of the PV and the subsequent claim to create the PVC.
The second level of OpenShift persistence configuration is associating the PVC with the Docker Container for use by the image in the container. From the template this is accomplished with the following from the "container/spec" section of the template:

```
  volumes:
    - emptyDir:
        medium: Memory
      name: dshm
    - name: poddata-${POD_NAME}
      persistentVolumeClaim:
        claimName: poddata-${POD_NAME}
```

Notice that the PVC name and the Volume name for the container make use of macro-substitutions based on the template's "parameters" definitions as described in Section 2.1. This will help define the resources with the required unique names.
The PVC is used as the Container Volume. This associates the physical Cloud (or bare metal) volume with the Docker container.

Solace PS+ also makes use of ephemeral storage as part of the control plane and IPC processing. This data is not required to survive a Pod failure or restart. This storage is created as an “emptyDir” that uses memory for IPC-like processing.

The third and final persistence configuration is the definition of the mounts within the Solace image against the Docker Container Volume. In this case the Solace PS+ image has the following configuration:

```
       volumeMounts:
         - mountPath: /dev/shm
           name: dshm
         - mountPath: /usr/sw/internalSpool/softAdb
           subPath: softAdb
           name: poddata-${POD_NAME}
         - mountPath: /usr/sw/adb
           subPath: adb
           name: poddata-${POD_NAME}
         - mountPath:  /usr/sw/var
           subPath: var
           name: poddata-${POD_NAME}
         - mountPath:  /usr/sw/internalSpool
           subPath: internalSpool
           name: poddata-${POD_NAME}
         - mountPath:  /var/lib/solace/diags
           subPath: diags
           name: poddata-${POD_NAME}
         - mountPath:  /usr/sw/jail
           subPath: jail
           name: poddata-${POD_NAME}
```

The shared memory is mounted, and then specific mounts are created against specific Solace image directories. In this case, there is a single Docker Volume and all Solace mounts are against this single volume. While this makes the template configuration simple, it is not necessarily recommended for production if high-performance persistence is a requirement against the PS+ broker. For performance multiple PVC should be defined, and each of those PVCs should become Docker volumes. The Solace image mounts should then each be against one of the multiple Docker Volumes. Some of the Solace mounts can still be against a single volume without affecting performance; however, a persistence performance discussion is beyond the scope of this document.

2.5 Solace Image Scaling

Unlike almost all other message/event brokers that implement restricted threading models, Solace PS+ software brokers scale is based on the number of CPUs and the allocated memory. These resources are also defined in the Template to ensure the required resources are created for the Docker Container.

In the template the container CPU and Memory are defined with the following:

```
        resources:
          requests:
            memory: "4096Mi"
            cpu: "2"
          limits:
            memory: "8192Mi"
            cpu: "4"
```

This configuration indicates that 2 CPUs and 4 Gig of memory should be initially deployed. But the container can expand to include up to 4 CPUs and 8Gig of memory if the PS+ broker experiences resource utilization requirement beyond the initial default values.

Some recommended Solace PS+ container CPU and Memory configurations can be found here:

https://docs.solace.com/Solace-SW-Broker-Set-Up/Docker-Containers/Set-Up-Single-Linux-Container.htm

2.6 Internal and External Access to the Solace PS+ Broker

OpenShift provides several network services that simplify access to containers that are managed inside the cluster.

For internal access, OpenShift provides a ClusterIP service. This service provides two main functions. First, a ClusterIP service allows dynamic DNS access from within the cluster to containers. Since a restart of a container may result in a new local IP address, the use of the ClusterIP service DNS name will ensure consistent access to the Docker Container even if the container IP address changes. The second function of the Cluster IP services is to restrict the cluster access to a limited number of IP socket numbers regardless of the ports exposed in the Docker Container or the Solace Image.

For this template, the following ClusterIP Services are defined:

```
- kind: Service
  apiVersion: v1
  metadata:
    name: ${POD_NAME}-ci-svc
    annotations:
      description: 'Exposes PS+ ClusterIP services'
  spec:
    type: ClusterIP
    clusterIP: None
    selector:
      pod: ${POD_NAME}
    ports:
    - name: 'smf'
      port: 55555
      targetPort: 55555
    - name: 'ssh'
      port: 2222
      targetPort: 2222
```

In this case a dynamic IP address is available (no IP address was defined) and only access to the Solace CLI console and the Solace messaging data plane are allowed. A selector is used to associate this ClusterIP service to the Docker Container Label.

The Label for the container is defined as part of the object configuration in the template:

```
objects:
- apiVersion: v1
  kind: Pod
  metadata:
    name: ${POD_NAME}
    labels:
      pod: ${POD_NAME}
```

In this case, the ClusterIP selector is associated with a POD that has the label "pod: ${POD_NAME}" where a macro substation is used based on the template parameters section.

Access from outside the OpenShift cluster may also be required. Access to specific Containers managed in OpenShift is accomplished via the use of NodePort Services. A NodePort service essentially provides Network Address Translation (NAT) services directly within the OpenShift Cluster. If access to the PS+ broker is not required outside of the Cluster, then a NodePort service is not required.

As with the ClusterIP service, the NodePort service can restrict which ports will be exposed for remote access regardless of the ports exposed in the Docker container.

For this simple template, the following NodePort Service is defined:

```
- kind: Service
  apiVersion: v1
  metadata:
    name: ${POD_NAME}-np-svc
    annotations:
      description: 'Exposes PS+ NodePort services'
  spec:
    type: NodePort
    selector:
      pod: ${POD_NAME}
    portalIP:
    ports:
    - name: 'smf-web'
      nodePort:
      port: 60080
      protocol: TCP
      targetPort: 60080
    - name: 'semp'
      nodePort:
      port: 8080
      protocol: TCP
      targetPort: 8080
    - name: 'smf'
      nodePort:
      port: 55555
      protocol: TCP
      targetPort: 55555
```

Note, that as with the ClusterIP service, the NodePort service is associated with a POD based on the Selector defined in the Service and the Label associated with the POD.

For simplicity, all the container-exposed services were exposed as NodePort services. In production, only a restricted set would be exposed. At first glance the NodePort service and ClusterIP service appear very similar. However, have a look at the NodePort service after it is deployed:

```
oc get services
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                                                      AGE
vmr-ci-svc   ClusterIP   None             <none>        55555/TCP,2222/TCP                                           6s
vmr-np-svc   NodePort    172.30.167.230   <none>        60080:31239/TCP,8080:30533/TCP,55555:30362/TCP,2222:31852/TCP   6s
```

For the NodePort service, it is showing a mapping of the defined ports to new values. For example, the default Solace port for client data connectivity (55555) is showing a reference to a value of 30362. When your template defines a NodePort service an interesting thing happens when the template is deployed. When deployed, each node in the cluster opens the same TCP listening port on all nodes in the cluster. The port is restricted to always be somewhere in the range of port 30,000 to 32,999. In this case, every node in the OpenShift cluster started listening on port 30362. Now when any application connects a TCP socket to any node in the cluster on port 30362, it will automatically be routed to the container that is referenced via the service's selector and connected to the associated container port, which in this case is port 55555.

Therefore, any Solace Client can connect from anywhere on the Internet (or any routed network with access to the OpenShift cluster) to this specific PS+ broker in OpenShift by simply connecting to any OpenShift node in the cluster and connect using socket number 30362.

However, it is important to note that access to the OpenShift cluster nodes may also be controlled by the Cloud Vendor. For example, when deploying the OpenShift Quick Start into AWS (as described in Section 1.1), the AWS Cloud Formation scripts automatically creates a load-balancer that distributes the external connections among the OpenShift Master Nodes. By default, this AWS Load-Balance service is restricted by Security Groups to be the only way you can get access to the OpenShift Cluster from outside AWS. This load-balancer is controlled by a security group that by default only allows connections via ports 22, 80 and 443. Therefore, by default, AWS will not allow the required external access via the ports exposed in the NodePort Service.

To solve this NodePort Service issue in AWS is actually quite simple. You must update the Load-Balancer Security Group to allow incoming access from ports 30,000 to 32,999. Secondly, you must add a Load-Balancer Listener to map access to the exposed PS+ container NodePort service ports.

3 Solace Singleton Template using Image Repository

The previous Solace template made use of the free Solace Public PubSub+ broker image and was downloaded from the Public Docker repository. However, the Enterprise PubSub+ broker is not publicly available for download directly into OpenShift.

To deploy the Enterprise PubSub+ broker into OpenShift requires the manual download of the image and its placement in to the local OpenShift Docker Repository. (Other repositories supported by OpenShift can also be used.).

The new template in this GitHub repository that makes use of the OpenShift Repository is called:

```
sol-single-ps+-persist-pod-reg-img.yml
```

This template is very similar to the previous template, with the exception of the changes that are required to use the OpenShift repository to load the Solace image to create the Solace OpenShift Container.

3.1 Loading the Solace Image into OpenShift

To be able to make use of the Solace Docker image from the OpenShift repository requires that the Solace Docker Tar image must first be loaded onto one of the OpenShift Master Nodes. From here it can be placed into the local Docker engine and then pushed into the OpenShift repository.

First you must log into OpenShift and be in the Project you want to be able to access the Solace Image. (Recall for our sample the user was “solace” in a project called “solace”). You will log into OpenShift while on one of the OpenShift master nodes using the “oc login” command. To ensure you have the required authentication token you should execute the following:

```
[root@linux91 SecureMode]# oc login
Authentication required for https://192.168.42.50:8443 (openshift)
Username: solace
Password:
Login successful.

You have one project on this server: "solace"

Using project "solace".
[root@linux91 SecureMode]# oc whoami -t
UZsFyuWuHZrF-WlEaGu-g9Lds6EdtRCrinH9wtDqDmU
[root@linux91 SecureMode]#
```

The “oc whoami -t” command must return a hash security token value similar to above for you to able to continue.

Now that you are logged into OpenShift, it is time to log in to the local Docker engine with the following command:

```
docker login -u solace -p $(oc whoami -t) docker-registry.default.svc:5000
```

(Note: for Mindshift you will need to substitute “docker-registry.default.svc” with the IP address of the service, which can be determined with the command “oc describe service -n default docker-registry”).

Once you have successfully logged into Docker, you can load the Solace image with:

```
docker load -i soltr-9.2.1008-enterprise-docker.tar.gz
```

If you were successful, the Solace PS+ image should be in docker:

```
# docker images
REPOSITORY                    TAG                                        IMAGE
ID            CREATED              SIZE
solace-app              9.2.0.1008-enterprise     45e971e3dbe8           5  weeks
ago          984.7 MB
```

Next you need to tag the image:

```
docker tag solace-app:9.2.0.1008-enterprise docker-registry.default.svc:5000/solace/solace-app:latest
```

The tag may need to be modified if your OpenShift environment does not match the configuration described in Section 1.2. The value after the service port refers to the OpenShift project where the image is going to be available. In this case it is "solace". For purposes of discussion, the image name will be pushed as "solace-app" with a tag of "latest".

The registry is referenced as a ClusterIP DNS name: "docker-registry.default.svc". This string indicates the IP address is against the "docker-registry" Pod in the "default" project namespace.

Finally, you can load the image with:

```
docker push docker-registry.default.svc:5000/solace/solace-app:latest
```

You will now have a Solace PS+ docker image located in the cluster's OpenShift Docker Registry. The following command should confirm the image is available:

```
oc get imagestream
```

3.2 Using the Solace Image from the OpenShift Docker Registry

You will now notice that in the new template, the "parameters" section includes a reference to an image:

```
- description: 'The name PS+ image in the local Docker Registry in the form:
registryIP:port/project/image:tag'
  from: '[a-z0-9]{40}'
  name: SOLACE_IMAGE
  value: 'docker-registry.default.svc:5000/solace/solace-app:latest'
```

At deployment time you can override the default value. You will also notice the container/spec/env section of the template references the new macro-substitution from the template parameters section described above:

```
image: ${SOLACE_IMAGE}
```

At this point, if you have followed all the instructions, an Enterprise PubSub+ broker was downloaded and pushed into Docker. The Docker image was then tagged and pushed into the OpenShift Docker Registry where it is now referenced and available for any template that is deployed in the "solace" project.

4 Deploying a Solace HA Cluster into OpenShift using simple Pods

The previous two OpenShift template samples were based on PS+ singletons. Many production environments require the Solace PS+ brokers to be deployed as a High-Availability (HA) cluster.

This template is designed to deploy Solace PS+ brokers as a full HA cluster. Once the cluster is deployed, only minimal configuration is required to complete the cluster installation. The next series of template samples will reduce/eliminate the requirement to perform post-deployment configuration tasks for the HA cluster.

The template that deploys PS+ as a cluster of simple Pods is found in GitHub and is called:

```
sol-ha-ps+-persist-pod.yml
```

It should become rather obvious that the HA-based template is essentially three separate singleton templates incorporated into a single template. There is a separate Pod object defined for each of the three members of Solace HA Cluster.

The template makes use of Solace Configuration Keys (as described in Section 2.2). The Solace Configuration Keys make it easy to configure the Solace PubSub+ broker by passing System Variables to the Docker Container. For this template, the following was added to the template container environment definitions:

```
        - name: configsync_enable
          value: "yes"
        - name: redundancy_activestandbyrole
          value: primary
        - name: redundancy_enable
          value: yes
        - name: redundancy_group_node_${POD_NAME}prim_connectvia
          value: "prim-${POD_NAME}-cluster-svc.${PROJECT}.svc:8300"
        - name: redundancy_group_node_${POD_NAME}prim_nodetype
          value: message_routing
        - name: redundancy_group_node_${POD_NAME}sec_connectvia
          value: "sec-${POD_NAME}-cluster-svc.${PROJECT}.svc:8300"
        - name: redundancy_group_node_${POD_NAME}sec_nodetype
          value: message_routing
        - name: redundancy_group_node_${POD_NAME}mon_connectvia
          value: "mon-${POD_NAME}-cluster-svc.${PROJECT}.svc:8300"
        - name: redundancy_group_node_${POD_NAME}mon_nodetype
          value: monitoring
        - name: redundancy_group_password
          value: admin
        - name: redundancy_matelink_connectvia
          value: "sec-${POD_NAME}-cluster-svc.${PROJECT}.svc:8741"
        - name: service_redundancy_firstlistenport
          value: "8300"
```

The Solace Configuration Keys automatically will enable the Solace Config Sync capabilities and establish the network connectivity between the Primary, Secondary and Monitor nodes in the Solace HA Cluster. The values above are from the section of the template related to the Primary Pod.

It is necessary to create some unique key details associated with each of the three pods. To accomplish this, Macro Substitution Variables are again used as part of the configuration keys. The variables are based on the "parameters" section of the template and can be used as substrings both in the name (key) and the value.

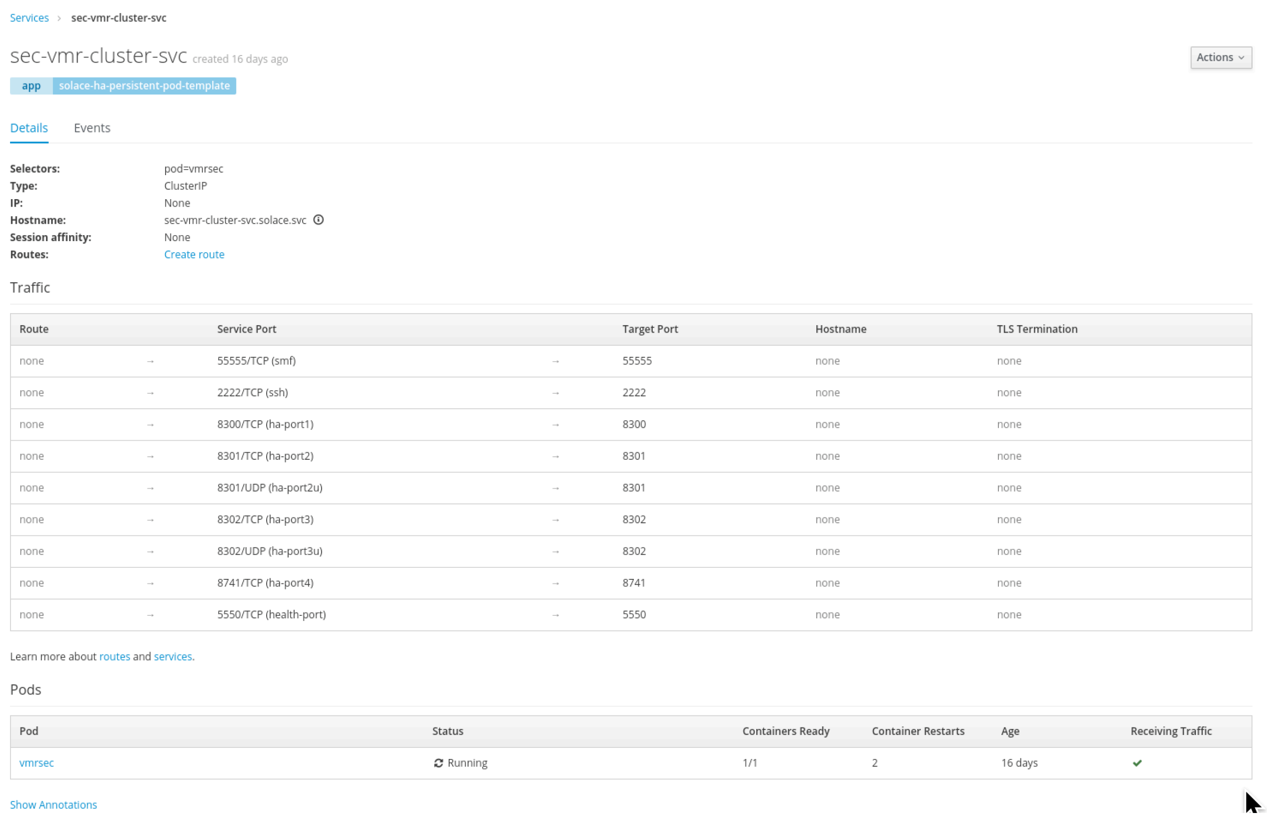
ClusterIP and NodePort services are also created for each Pod in the cluster. It is impossible to know the IP address that will be assigned (well, it could have been done statically, but this is never recommended if possible), so the ClusterIP service name is used. For example:

```
sec-${POD_NAME}-cluster-svc.${PROJECT}.svc:8741
```

that after substitution maps to:

```
sec-vmr-cluster-svc.solace.svc:8741
```

The sec-vmr pod show the following associated service (note the hostname value):



Services › sec-vmr-cluster-svc

sec-vmr-cluster-svc created 16 days ago

app solace-ha-persistent-pod-template

Details Events

Selectors:	pod=vmrsec
Type:	ClusterIP
IP:	None
Hostname:	sec-vmr-cluster-svc.solace.svc
Session affinity:	None
Routes:	Create route

Traffic

Route		Service Port		Target Port	Hostname	TLS Termination
none	→	55555/TCP (smf)	→	55555	none	none
none	→	2222/TCP (ssh)	→	2222	none	none
none	→	8300/TCP (ha-port1)	→	8300	none	none
none	→	8301/TCP (ha-port2)	→	8301	none	none
none	→	8301/UDP (ha-port2u)	→	8301	none	none
none	→	8302/TCP (ha-port3)	→	8302	none	none
none	→	8302/UDP (ha-port3u)	→	8302	none	none
none	→	8741/TCP (ha-port4)	→	8741	none	none
none	→	5550/TCP (health-port)	→	5550	none	none

Learn more about routes and services.

Pods

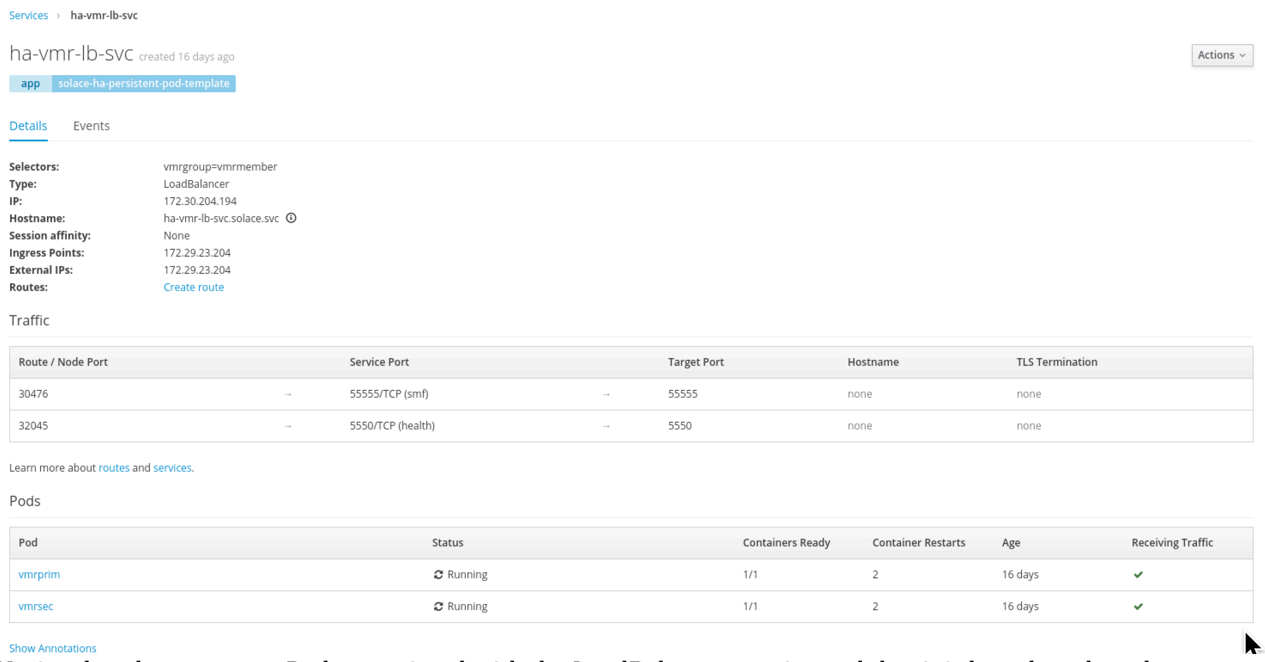
Pod	Status	Containers Ready	Container Restarts	Age	Receiving Traffic
vmrsec	Running	1/1	2	16 days	✓

Show Annotations

4.1 Load Balanced Service

This HA cluster template adds a new service to the OpenShift environment. This service is a LoadBalancer Service. This service is very similar to the NodePort service, except that the service can be associated to more than one Pod. Therefore, if you connect to the node port of a LoadBalancer service, it will connect the client connection to associated Pods in a round-robin manner. As with most associations in OpenShift, the LoadBalancer Service is associated with Pods via selectors against Labels in the Pod.

Consider the LoadBalancer service that is created in this template:



Services › ha-vmr-lb-svc

ha-vmr-lb-svc created 16 days ago

app solace-ha-persistent-pod-template

Details Events

Selectors:	vmrgroup=vmrmember
Type:	LoadBalancer
IP:	172.30.204.194
Hostname:	ha-vmr-lb-svc.solace.svc
Session affinity:	None
Ingress Points:	172.29.23.204
External IPs:	172.29.23.204
Routes:	Create route

Traffic

Route / Node Port		Service Port		Target Port	Hostname	TLS Termination
30476	→	55555/TCP (smf)	→	55555	none	none
32045	→	5550/TCP (health)	→	5550	none	none

Learn more about routes and services.

Pods

Pod	Status	Containers Ready	Container Restarts	Age	Receiving Traffic
vmrprim	Running	1/1	2	16 days	✓
vmrsec	Running	1/1	2	16 days	✓

Show Annotations

Notice that there are two Pods associated with the LoadBalancer service and that it is based on the selector:

```
vmrgroup=vmrmember
```

The template also shows the following for the Primary Pod:

```
vmrgroup: ${POD_NAME}member
```

It is that simple to associate Pods with LoadBalancer Services.

When connecting to the Solace HA cluster, by indicating the OpenShift Node IP and the LoadBalancer NodePort (port 30476 in the sample above), you will be connected to one of the processing PubSub+ brokers in the HA pair. However, you might connect to the standby broker and be disconnected. Therefore, you must ensure the client API is set to perform automatic retries to ensure you will quickly be connected to the active broker.

4.2 Solace Config-Sync

Solace has a unique feature available when deploying PubSub+ brokers as HA and /or Disaster Recovery (DR) clusters. To guarantee all configurations among the cluster members are the same, Solace added a config-sync capability to ensure that a change (addition, modification or deletion) on any one broker in the HA/DR cluster is automatically replicated among all members of cluster.

When the HA cluster is deployed in OpenShift using this template, there is no guarantee that the primary is created and available before the secondary. As a result, config-sync may not actually have the HA members in sync. This needs to be confirmed before using the HA cluster in OpenShift after it is deployed.

Consider the following cluster details right after deployment:

```
[root@linux91 ~]# minishift ip
192.168.42.50
[root@linux91 ~]# ssh -p 30682 admin@192.168.42.50
The authenticity of host '[192.168.42.50]:30682 ([192.168.42.50]:30682)' can't be established.
ECDSA key fingerprint is SHA256:tcVF9+l4CwNu3c+fvWgYmavGEGhKI5DvQfrX/fBzVzw.
ECDSA key fingerprint is MD5:31:a8:cc:59:11:e3:24:90:68:ef:ee:f2:d7:49:66:13.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[192.168.42.50]:30682' (ECDSA) to the list of known hosts.
Solace PubSub+ Enterprise
Password:

Solace PubSub+ Enterprise Version 100.0openshift_scc.0.18

The Solace PubSub+ Enterprise is proprietary software of
Solace Corporation. By accessing the Solace PubSub+ Enterprise
you are agreeing to the license terms and conditions located at
http://www.solace.com/license-software

Copyright 2004-2019 Solace Corporation. All rights reserved.

Operating Mode: Message Routing Node

vmrprim> en
vmrprim# show config-sync database

Legend:
O - Ownership (M=Master, S=Slave)
S - Status (I=In-Sync, R=Reconciling, B=Blocked, O=Out-Of-Sync, D=Down,
            U=Unknown)

Type   Name                             O S Time In State
------ -------------------------------- - - ------------------
Router site                             M O 0d 3h 16m 24s
Vpn    default                          M I 0d 3h 16m 43s

vmrprim# admin
vmrprim(admin)# config-sync assert-master

 Invalid command input
          config-sync assert-master
                                   ^
           -> missing parameter
vmrprim(admin)# config-sync assert-master
message-vpn  router
vmrprim(admin)# config-sync assert-master router
Processed 1 config-sync tables.
vmrprim(admin)# show config-sync database

Legend:
O - Ownership (M=Master, S=Slave)
S - Status (I=In-Sync, R=Reconciling, B=Blocked, O=Out-Of-Sync, D=Down,
            U=Unknown)

Type   Name                             O S Time In State
------ -------------------------------- - - ------------------
Router site                             M I 0d 0h 0m 8s
Vpn    default                          M I 0d 3h 18m 32s

vmrprim(admin)#
```

In this case we are using Mindshift, so we first need to know the IP address for the Minishift cluster. Next, we connect to the PubSub+ console port (2222) using the NodePort exposed service (in this case the port is 30682). By displaying the config-sync database we can see that the cluster is not in sync. Therefore, the primary uses the “assert-master” command to resynchronize the cluster.

At this point the installation of the HA cluster into OpenShift is complete.

It is important to note that subsequent template samples will illustrate how to automate the config-sync operations so that the cluster is immediately available after deployment without post -dministration tasks.

5 Deploying a Solace HA Cluster using OpenShift Deployment Configuration and Configuration Map

With this Template sample we are introducing three new OpenShift capabilities that can be used when creating Solace HA cluster in OpenShift. This new Sample template will make use of:

- Deployment Configurations
- Configuration Maps
- Secrets

The sample template with these additional features is found in this GitHub repository under the name:

```
sol-ha-ps+-persist-deploymentconfig-configmap.yml
```

The Configuration Map template is called:

```
solace-configmap.yml
```

The secrets file is called:

```
solace-admin-secrets.yml
```

So far, the sample templates used to deploy Solace PS+ software were based on the generation of simple Pods. If the Pod was accidentally deleted or the node that the Pod was running on disappeared, then the Solace Service would be gone. OpenShift solves this issue through the use of Deployment Configurations.
An OpenShift Deployment adds more fine-grain control over the created Pods. A Deployment describes:

- a desired state for the Pod;
- a replication controller that maintains a point-in-time reference of the state of a deployment configuration as a pod template;
- one or more pods, which represent an instance of a particular version of an application

It is possible to make use of Deployment Configurations when deploying Solace PubSub+ containers. Although you would not create multiple replicas for scaling the number of PubSub+ broker containers (see Section 2.5 concerning scaling), however, Deployment Configurations would ensure that the Solace containers would be available even if the Node running the container fails or if the container was accidentally deleted form OpenShift. The Deployment's replication controller will ensure the Solace PubSub+ container would automatically be restarted.
When using Deployments, it is also possible to make use of OpenShift Configuration Maps. A Configuration Map is an OpenShift Object that stores data independent from the Pods. A Configuration Map object can be mounted to any Container and the details for the name/value pairs is available to any Container that references the Map Object.
One of the interesting features of Configuration Map objects is that the value for a name/value pair can be scripts that are used as commands that are directly executable from the container. In this new template, we will use Map

Object scripts to determine when the HA cluster is fully up and running and then automatically have the Primary PubSub+ container automatically execute the “config-sync assert master” command that was required to be done manually in the previous sample template.

We will also introduce OpenShift Secrets Objects in this template. In the previous template definitions, we used Solace Configuration Keys to create a default administration user and password. Unfortunately, the password was in the clear, which is a security issue. The Secrets Objects allow the use of obfuscated values to be stored in OpenShift and used in Templates to create “hidden” passwords.

5.1 Preparation before Deployment Configuration Template is Installed

Before the new template can be deployed, the secret and configuration map objects must be available in the OpenShift project. They are easy enough to deploy from the command line.

5.1.1 Deploy Secrets Object

Once you have logged into you project (in this case we are using “solace” as the project) you can load the secret object with:

```
oc create -f solace-admin-secrets.yml
```

This command assumes you were in the directory where the YAML file was located.

The template shows that the name of the secret object should be set to “admin-secret”. Using this information, you can now see the secret object is available:

```
oc describe secret admin-secret
Name:         admin-secret
Namespace:    solace
Labels:       <none>
Annotations:  <none>

Type:  Opaque

Data
====
password:  5 bytes
username:  5 bytes
```

We are again using the admin/admin username/password. Obviously, in production stronger passwords would be used. The key point is that the password is obfuscated. We also obfuscated the username, which usually may not be required.

To make use of the secrets object, the template was modified to allow the container to use this object:

```
      spec:
        containers:
          - env:
            - name: USERNAME_${ADMIN_USER}_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: admin-secret
                  key: username
            - name: USERNAME_${ADMIN_USER}_GLOBALACCESSLEVEL
              valueFrom:
                secretKeyRef:
                  name: admin-secret
                  key: password
```

This template now gets the username and password from the obfuscated values from the Secrets Object and uses those to create the default username and password for the Solace PubSub+ broker. Once the broker is created, if you do not know the username and password, you will not get access. On the person who created the secret object will know the password (and of course whoever they share the information).

5.1.2 Deploy Configuration Map (ConfigMap) Object

As mentioned above, the ConfigMap object is very useful for storing configuration details that will be used and reused by Pods. In this case, we have created a ConfigMap template that loads shell scripts that will be used by the Deployment Configurations to execute these scripts.

The ConfigMap file used from this GitHub repo is called:

```
solace-configmap.yml
```

It is simple to load this ConfigMap from the command-line:

```
oc create -f ./solace-configmap.yml
oc process solace-configmap | oc create -f –
```

The first command loads the template into the local project template repository. The second command executes the instructions to process the template from the repository and create the ConfigMap.

The config map is now available to provide three commands that are available to any container that mounts the ConfigMap. These three commands are Linux Shell Scripts. The purpose of theses commands is to automate the config-sync setup for the HA.

Recall, as discussed for the previously described template it was necessary to manually assert config-sync database consolidation. These shell commands are used to execute Solace SEMP commands to determine if the primary node has become active for redundancy, and when it does become active it will force the config-sync assert activity.

5.2 Deploying Solace HA Deployment Configuration with Configuration Maps

The template that creates the Solace PubSub+ HA cluster is very similar to the template that deploys the HA cluster as simple Pods. Now the deployment of Pods is controlled by the OpenShift Deployment Replication Controller. Although only one instance of each PubSub+ brokers is started, the controller ensures there will always be at least one Pod available in OpenShift.

Consider the following for the Spec for the new primary Pod container:

```
- apiVersion: v1
  kind: DeploymentConfig
  metadata:
    name: ${POD_NAME}prim
    labels:
      pod: ${POD_NAME}prim
      deploymentconfig: ${POD_NAME}prim
      vmrgroup: ${POD_NAME}-member
  spec:
    strategy:
      type: Rolling
      rollingParams:
        updatePeriodSeconds: 1
        intervalSeconds: 1
        timeoutSeconds: 600
      resources: {}
      triggers:
        - type: ConfigChange
        - type: ImageChange
          imageChangeParams:
            automatic: true
    replicas: 1
    selector:
        deploymentconfig: ${POD_NAME}prim
    template:
      metadata:
        labels:
          deploymentconfig: ${POD_NAME}prim
          vmrgroup: ${POD_NAME}-member
```

This snippet associates the deployment configuration with the template via a selector and is configured to ensure there is at least one instance available.

The ConfigMap is mounted as a volume to the Solace Pod. The following associates the ConfigMap as a Volume:

```
            volumeMounts:
            - name: config-map
              mountPath: /mnt/disks/solace
```

And this is based on the following "volumes" definition in the template:

```
        - name: config-map
          configMap:
            name: "solace-configmap"
            defaultMode: 0755
```

When you connect as a shell user to one of the Solace HA broker's docker container, it will show that the following commands are available based on the fact they were in the ConfigMap object that was mounted into the Solace broker.

```
sh-4.2$ cd /mnt/disks/solace
sh-4.2$ ls
config-sync-check.sh  readiness_check.sh  semp_query.sh
sh-4.2$
```

Tthe "config-sync-check.sh" command checks if the containers config-sync status is "up". If so, the "config-synch-check.sh" command forces a config-sync assert and the Solace HA Cluster is ready for use. The script is started automatically by defining in the container template the following:

```
            command:
              - bash
              - "-ec"
              - |
               #source /mnt/disks/solace/init.sh
               # not using postinstall hooks because of order dependencies
               # launch config check then Solace so VCR can provide return code
               nohup /mnt/disks/solace/config-sync-check.sh &
               /usr/sbin/boot.sh
```

Normally, the Solace container will automatically "execute boot.sh". In this case, the "config-sync-check.sh" command is started first and is run in the background since it needs results from the boot.sh script.
Once the configuration map and secret objects are loaded it is possible to deploy the new template that makes use of deployment configurations. Once the template is deployed, the Solace HA cluster is fully operation aalnd no longer requires the manual config-sync assert. The cluster pods are also now protected from failure since they are automatically restarted with their full configuration and disk volumes based on the work by the replication controller. Higher security is available since the administrator credentials are no longer using clear-text for their definition.

6 Deploying Solace HA Cluster using OpenShift Stateful Sets

OpenShift has introduced a more advanced controller capability that exceeds the capabilities of the Deployment Configuration. The new controller is called a StatefulSet.

StatefulSet controllers are similar to the Deployment controllers, except that the StatefulSet objects are not interchangeable. As described in the OpenShift documentation:

"StatefulSets are valuable for applications that require one or more of the following.

- Stable, unique network identifiers.
- 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.

While Deployments are more than acceptable for the creation of Solace PubSub+ HA clusters in OpenShift, StatefulSets add more control for updates; however, both are acceptable for Solace deployments.

In this repository, the StatefulSet template is called:

```
sol-ha-ps+-persist-statefulset-configmap.yml
```

This template is based on the Solace OpenShift QuickStart, that can be found here:

```
https://docs.solace.com/Developer-Tools/QuickStarts-Connectors/Quickstart-OpenShift.htm
```

This sample template has some minor changes from the Quick Start (for more details on the Quick Start see Section 1.1) and will operate in Minishift and most OpenShift environments where either a default storage class or Persistent Volume (PV) pool is available.

The quick start supports AWS availability zones to ensure each of the Solace PubSub+ cluster members are deployed in separate availability zones. This simple sample assumes a single zone. Please refer to the Quick Start for more details that apply to this template.

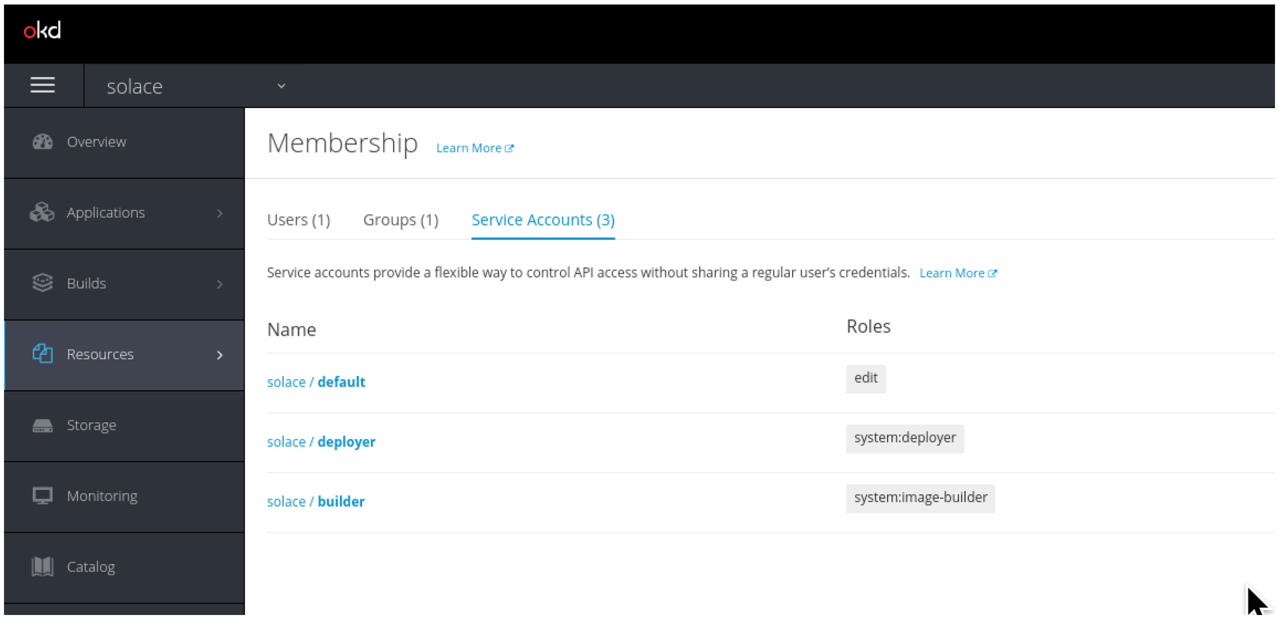
The Quick Start also provides details for using Helm Charts rather than Templates for deploying the Solace Cluster. This sample only makes use of an OpenShift Template.

6.1 Preparation before the StatefulSet Template is Deployed

So far, all the template examples assumed that the OpenShift User and project have the default restricted security enabled. One of the features of this StatefulSet template is that the ConfigMap includes a script that determines which of the HA pair is the active member. The script updates the active Pod's labels to set "active=true".

The StatefulSet template also adds a LoadBalancer service. This automatically binds Pods based on a selector of "active=true". Therefore, the LoadBalancer service will only connect remote clients to the Solace HA Pod that is the active member. As a result, the client will not need to set "reconnect" for the initial Solace client session since the service will only ever connect to the active HA member. This is different than the activity of using a LoadBalancer service as described in Section 4.1.

Unfortunately, the default restricted access for users will not allow the update of the Pod labels once the Pod has been created. However, this can be remedied by updating the service account associated with the user in the project. This is easily done using the GUI by one of the OpenShift administrators. The administrator needs to provide "edit" permissions to the default user in the project, which in our case is "solace/default" (we are using a project called "solace").



Service accounts provide a flexible way to control API access without sharing a regular user's credentials. Learn More

Name	Roles
solace / default	edit
solace / deployer	system:deployer
solace / builder	system:image-builder

If your configuration is similar to above, then you are ready to proceed to run the StatefulSet template.

6.2 Deploying the StatefulSet Template

This sample template automatically deploys everything it needs, including the configuration map object, secrets object and all the required services. Therefore, if you want to re-install the exact same environment, it is necessary to either change the name of the deployment or delete all the automatically installed objects and services.

The StatefulSet template is different from other Solace HA templates that have been described so far. In this case, there are not three separate Pods/containers defined, there is only one Pod definition for the StatefulSet object with a replica set to 3. Therefore, the same Pod is created three times using the same Solace PubSub+ image that is stored in the OpenShift repository.

With the previously described templates, the Solace PubSub+ container was configured via Configuration Map system memory variables defined in the template. With this StatefulSet template, it creates the required environment by having the container execute an “init.sh” script as defined in the container object command section:

```
          command:
           - bash
           - "-ec"
           - |
             source /mnt/disks/solace/init.sh
             # not using postinstall hooks because of order dependencies
             # launch config check then Solace so VCMR can provide return code
             nohup /mnt/disks/solace/config-sync-check.sh &
             /usr/sbin/boot.sh
```

This is slightly different from the command definition in the previous Deployment template sample. It is important to note that each StatefulSet Pod is set with a unique ordinal value. Based on the ordinal value the “init.sh” script will process the Pod replicas differently; where 0 is the primary container, 1 is the secondary container, and 2 is the monitor container.

The StatefulSet object also defines a liveness and readiness probe. The readiness probe executes a script from the mounted ConfigMap object. The liveness probe checks if port 8080 is still active for the containers.

```
         livenessProbe:
            tcpSocket:
              port: 8080
            initialDelaySeconds: 300
            timeoutSeconds: 5
          readinessProbe:
            initialDelaySeconds: 30
            periodSeconds: 5
            exec:
              command:
              - /mnt/disks/solace/readiness_check.sh
              - "7"
```

The same readiness probe is used for the previous deployment configuration template. It tests if there is an HA switch (manually or automatically) between the processing cluster members.

The processing for the config-sync is similar to the previous deployment configuration template sample.

The secret for this template uses a different password. In this case the default password (which you can change in the parameters at deployment time) is:

```
p0assw0rd
```

It is important to note that with this template example the only access to the Solace PubSub+ brokers is via the LoadBalancer Service. Therefore, only direct docker image shell access is available to configure the brokers via the Solace CLI. Access to the Docker shell can also be accomplished by getting access via the docker container from the docker commands or via the OpenShift Console GUI.

NodePort services similar to those used in previous template samples could also be added to this sample template if there is desire to allow "ssh" access or Solace GUI configuration access to configure the Solace HA Cluster members.

7 Scaling a Solace Application in OpenShift

Up until this point, all descriptions and discussions were directed toward deploying Solace PubSub+ brokers into OpenShift. However, it also expected that many applications will be deployed into OpenShift that will require access to the Solace brokers.

It is important to note that applications deployed in OpenShift are NOT restricted to connectivity to brokers also running in OpenShift. Solace applications running in OpenShift can connect to any Solace PubSub+ broker running anywhere as long as there is IP connectivity available to the broker from OpenShift.

The OpenShift Sample Application template found in this repository is called:

```
sample-aggregator-app.yml
```

This sample template is based on the Java code found in this repository:

```
https://github.com/SolaceLabs/solace-messaging-demo.git
```

This sample application can be run as a standalone SpringBoot application. The purpose of this set of applications is to demonstrate that if you create more worker applications the rate of processing improves linearly. It was originally created to show how scaling worked in Cloud Foundry. However, the Spring Boot source code also works as expected in OpenShift.

Please refer to the GitHub URL Readme.md for more details concerning this application.

7.1 Deploying in OpenShift

The sample template makes use of concepts described throughout this document and accomplishes three things:

- Deploys the aggregator application directly from the GitHub Source Code in GitHub using OpenShift's Source-to-Image capabilities.
- Deploys the worker application directly from the GitHub Source Code in GitHub using OpenShift's Source-to-Image (S2I) capabilities.
- Deploys the Solace PubSub+ broker and the services required for access to the brokers and external access to the aggregator application.

There are several S2I images available for multiple languages and development frameworks. These sample applications were based on Java and Gradle. While there are several S2I images available for Java via the OpenShift XPAAS addons, they do not support Gradle. Therefore, you must first add a S2I Java/Gradle -ompliant image to your project. I recommend using the command:

```
oc create -f https://raw.githubusercontent.com/jorgemoralespou/s2i-java/master/ose3/s2i-java-imagestream.json
```

Details of this Open Source image can be found here:

```
https://github.com/jorgemoralespou/s2i-java
```

If the command was successful you show now have an image stream similar to the following:

```
[root@linux91 ~]# oc get is
NAME                          DOCKER REPO                                        TAGS      UPDATED
s2i-java                      172.30.1.1:5000/solace/s2i-java                    latest    14 hours ago
solace-app                    172.30.1.1:5000/solace/solace-app                  latest    41 hours ago
```

The “s2i-java” image is the one that is used in this template. Basically, it takes details from the GitHub location to automatically download the source code, compile it using Gradle, create a Docker image and then place Docker image into the OpenShift Project’s Docker Repository. This image is then automatically used to as part of the Deployment Config to load the application into OpenShift as a Pod.

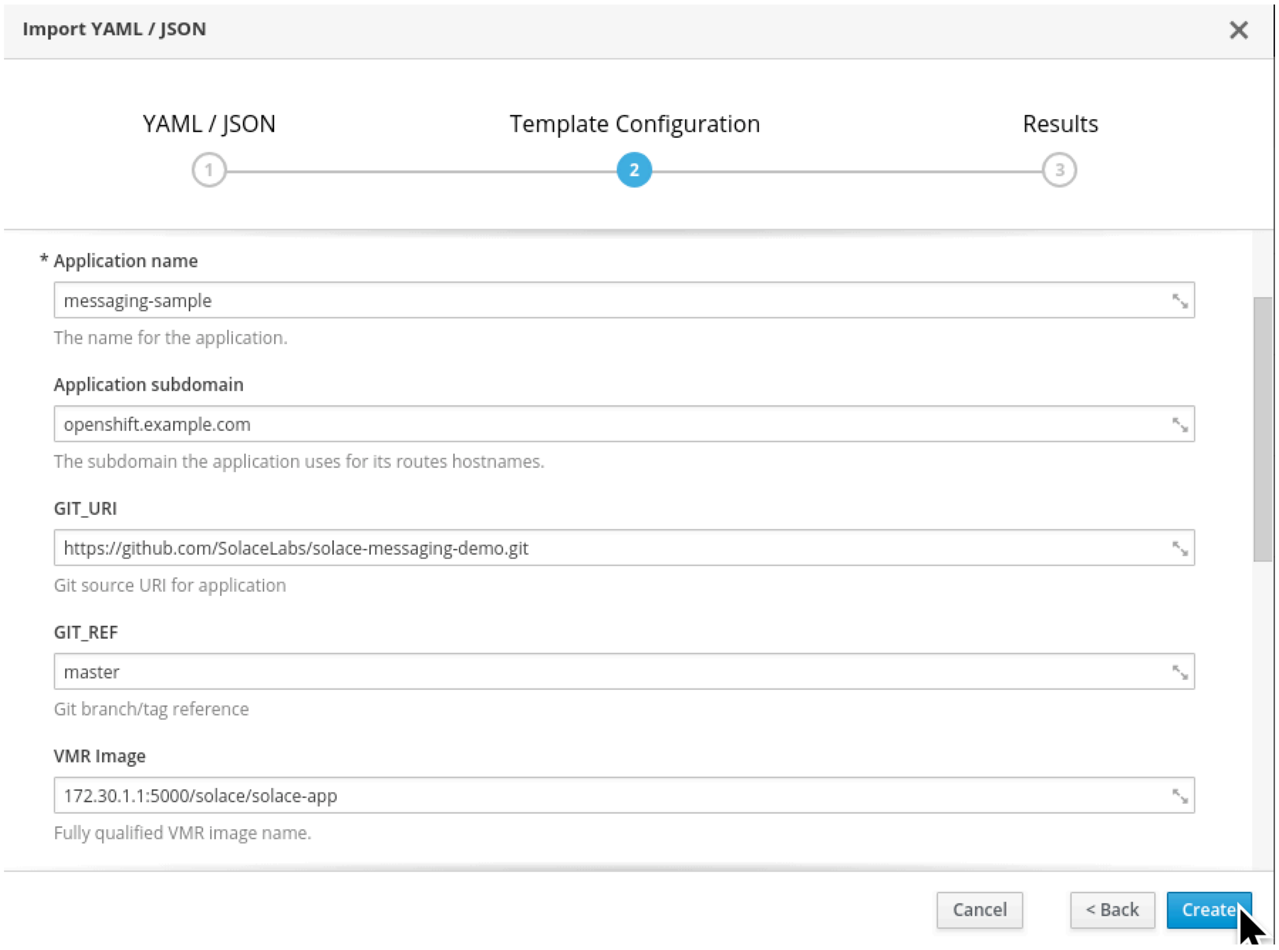
Once the template has successfully been processed, the Java images from the source code will be available as images in the repository and you will see something similar to:

```
[root@linux91 ~]# oc get is
NAME                          DOCKER REPO                                        TAGS      UPDATED
messaging-sample-aggregator   172.30.1.1:5000/solace/messaging-sample-aggregator latest    About an hour ago
messaging-sample-worker       172.30.1.1:5000/solace/messaging-sample-worker     latest    About an hour ago
s2i-java                      172.30.1.1:5000/solace/s2i-java                    latest    14 hours ago
solace-app                    172.30.1.1:5000/solace/solace-app                  latest    41 hours ago
```

This entire automated process is controlled by an OpenShift Build Configuration. For the “aggregator” application, the definition in the template is as follows:

```
  - kind: BuildConfig
    apiVersion: v1
    metadata:
      name: '${APPLICATION_NAME}-aggregator'
    spec:
      triggers:
        - type: ImageChange
          imageChange: {}
      source:
        type: Git
        git:
          uri: '${GIT_URI}'
          ref: '${GIT_REF}'
        contextDir: 'aggregator'
      strategy:
        type: Source
        sourceStrategy:
          from:
            kind: ImageStreamTag
            name: 's2i-java:latest'
      output:
        to:
          kind: ImageStreamTag
          name: '${APPLICATION_NAME}-aggregator:latest'
      resources: {}
```

The GitHub-related macro substitution variables are based to on OpenShift Template “parameters” default definition:



* Application name

messaging-sample

The name for the application.

Application subdomain

openshift.example.com

The subdomain the application uses for its routes hostnames.

GIT_URI

https://github.com/SolaceLabs/solace-messaging-demo.git

Git source URI for application

GIT_REF

master

Git branch/tag reference

VMR Image

172.30.1.1:5000/solace/solace-app

Fully qualified VMR image name.

If you clone the project, you will need to change the repository location to your own location.

The application is a Spring Boot application. While Spring Boot makes use of a local configuration file to pass configuration details to the application, it also supports making use of System Memory Variables that will override the details in the configuration file.

For example, in the template definition for the aggregator application's deployment configuration definition you will find the following:

```
        spec:
          containers:
            - name: '${APPLICATION_NAME}-aggregator'
              image: '${APPLICATION_NAME}-aggregator'
              ports:
                - containerPort: 8090
                  protocol: TCP
              env:
              - name: 'solace_java_host'
                value: '${APPLICATION_NAME}-vmr'
              - name: 'solace_java_msgVpn'
                value: 'default'
              - name: 'solace_java_clientUsername'
                value: 'default'
              - name: 'solace_java_clientPassword'
                value: 'default'
```

This section in the template defines the required variables that are created in the container that are then used by the aggregator Spring Boot application. The most interesting one is the "solace_java_host". Here the value is again an OpenShift macro substitution. The application will use this value as "hostname" for the Solace PubSub+ broker. Since the default from the template parameters section is "APPLICATION_NAME=messaging-sample" the hostname would be "messaging-sample-vmr". But how does the application find the VMR at that hostname?
In OpenShift the hostnames are actually the ClusterIP services. The following services were also created by this template:

```
[root@linux91 ~]# oc get services
NAME                           TYPE          CLUSTER-IP       EXTERNAL-IP    PORT(S)
AGE
aggregator                 NodePort     172.30.146.59    <none>        8090:32315/TCP
1h
messaging-sample-vmr           ClusterIP          172.30.25.219           <none>
8080/TCP,60943/TCP,55555/TCP,55003/TCP,55556/TCP,55443/TCP,6080/TCP,60443/TCP,18
83/TCP,8883/TCP,8000/TCP,8443/TCP,9000/TCP,9443/TCP,2222/TCP   1h
```

Notice that there is a "messaging-sample-vmr" ClusterIP service that was created. This is used by the applications as the hostname for connection to the Solace Broker. Since we are in the "solace" project, the hostname could also be the internal OpenShift dynamic DNS name: "messaging-sample-vmr.solace.svc".

7.2 Running the Solace OpenShift Applications

Details on running the Solace Applications are found in the original GitHub Repository:

```
https://github.com/SolaceLabs/solace-messaging-demo.git
```

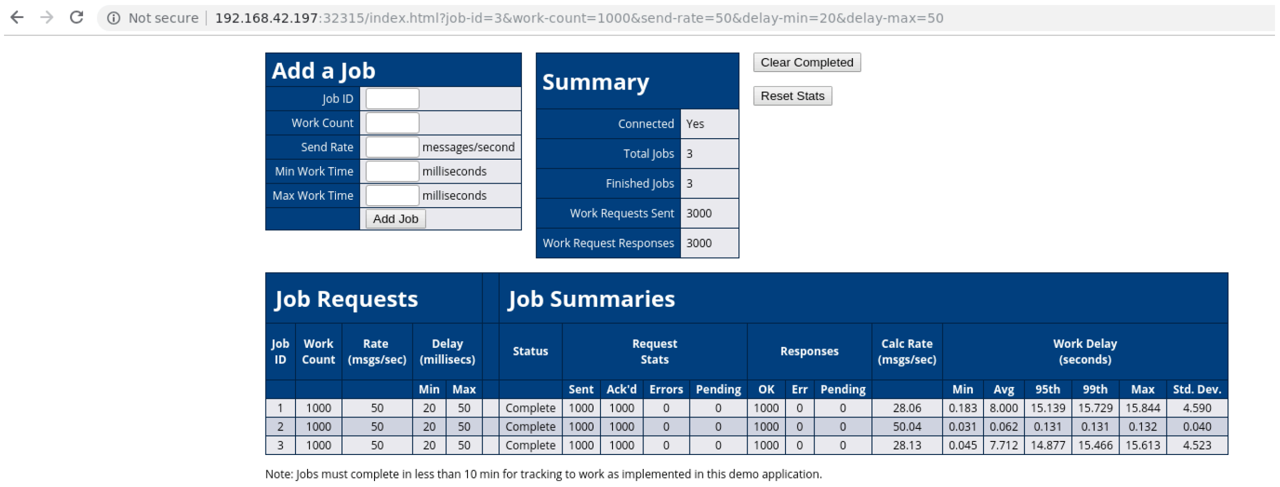
However, in this case the HTTP access to the aggregator application is the IP address of any OpenShift Cluster node and the NodePort Service port mapping. In this case the aggregator NodePort service is as follows:

```
NAME                         TYPE          CLUSTER-IP         EXTERNAL-IP    PORT(S)
AGE
aggregator                 NodePort     172.30.146.59    <none>          8090:32315/TCP
```

As a result, for access to the application running in, for example Minishift, the URL for connection to port 8090 of the aggregator application would be:

```
http://192.168.42.197:32315
```

The following shows the output after running the application over multiple runs (note the Connection URL):



Not secure | 192.168.42.197:32315/index.html?job-id=3&work-count=1000&send-rate=50&delay-min=20&delay-max=50

Add a Job

Job ID		
Work Count		
Send Rate		messages/second
Min Work Time		milliseconds
Max Work Time		milliseconds
	Add Job	

Summary

Connected	Yes
Total Jobs	3
Finished Jobs	3
Work Requests Sent	3000
Work Request Responses	3000

Clear Completed

Reset Stats

Job Requests / **Job Summaries**

Job ID	Work Count	Rate (msgs/sec)	Delay (millisecs)		Status	Request Stats				Responses			Calc Rate (msgs/sec)	Work Delay (seconds)					
			Min	Max		Sent	Ack'd	Errors	Pending	OK	Err	Pending		Min	Avg	95th	99th	Max	Std. Dev.
1	1000	50	20	50	Complete	1000	1000	0	0	1000	0	0	28.06	0.183	8.000	15.139	15.729	15.844	4.590
2	1000	50	20	50	Complete	1000	1000	0	0	1000	0	0	50.04	0.031	0.062	0.131	0.131	0.132	0.040
3	1000	50	20	50	Complete	1000	1000	0	0	1000	0	0	28.13	0.045	7.712	14.877	15.466	15.613	4.523

Note: Jobs must complete in less than 10 min for tracking to work as implemented in this demo application.

The first job was the default run with one "worker" Pod. For the second job, we used OpenShift to manually increase the deployment controller to increase the number of "worker" Pods from 1 to 2. Notice, as expected,the calculated rate doubled. Because the worker consumer is using a Solace Non-Exclusive Queue, as more consumers bind to the queue the queue will load balance processing among the active bound consumers.

The third job was run after the replication controller for the work deployment changed the Pod count back to 1. As expected, the throughput was reduced back to the same as job 1.

This application template sample demonstrates most of the "moving parts" that are used to easily deploy and scale Spring Boot Solace applications directly from the Solace Java source code found in a GitHub repository.