Buf Schema Registry on-premises documentation

Dependencies

The BSR uses PostgreSQL (version 14) for storing application data, Redis (version 5) for caching, S3-compatible storage for persisting modules and plugins, as well as an Identity Provider (IdP) supporting SAML for authentication.

Interacting with the BSR via the Buf CLI requires a minimum Buf CLI version of v1.23.0.

Installation

The BSR is designed to run on Kubernetes, and is distributed as a Helm Chart and accompanying Docker images through an OCI registry. The Helm Chart and Docker images are versioned, and are expected to be used together—that is, the default values in the Chart use Docker images with the same version as the Chart itself.

1. Authenticate `helm`

To get started, authenticate `helm` with the Buf OCI registry using the keyfile that was sent alongside this documentation.

```
cat keyfile | helm registry login -u _json_key_base64 --password-stdin \
https://us-docker.pkg.dev/buf-images-1/bsr
```

2. Create a namespace

Create a Kubernetes namespace in the k8s cluster for the `bsr` Helm Chart to use:

```
kubectl create namespace bsr
```

3. Create a pull secret

Create a pull secret using the provided keyfile, this will be used by the cluster to pull images from the Buf OCI registry:

```
kubectl create secret --namespace bsr \
docker-registry bufpullsecret \
--docker-server=us-docker.pkg.dev/buf-images-1/bsr \
--docker-username=_json_key_base64 \
--docker-password=$(cat keyfile)
```

4. Configure the BSR's Helm values

The BSR is configured using Helm values through the `bsr` Helm Chart.

1. Create a file named `bsr.yaml` to store the Helm values.
   - This file can be in any location, but we recommend creating it in the same directory where the helm commands are run. It is required by the `helm install` step below.
2. Set the desired `host` .
3. Configure the chart to use the image pull secret (created above):

   ```
   host: example.com # Hostname that the BSR will be served from
   imagePullSecrets:
     - name: bufpullsecret # The image pull secret that was created above
   ```

4. Put the values from the steps below in the `bsr.yaml` file. You may skip to "Install the Helm Chart" for a full example Helm chart.

Configure object storage

The BSR requires S3-compatible object storage.

1. To configure the storage set the following Helm values, filling in your S3 variables:

   ```
   storage:
     use: s3
     s3:
       bucketName: "my-bucket-name"
       endpoint: "s3.us-east-1.amazonaws.com"
       accessKeyId: "AKIAIOSFODNN7EXAMPLE"
       region: "us-east-1"
       # forcePathStyle: false # Optional, use path-style bucket URLs (http://s3.amazonaws.com/BUCKET/KEY)
       # insecure: false # Optional, disable TLS
   ```

2. Then create a k8s secret containing the s3 access secret key:

   ```
   kubectl create secret --namespace bsr generic bufd-storage --from-literal=secret_access_key=<s3 secret access key>
   ```

Create a Postgres database

The BSR requires a PostgreSQL database. We support PostgreSQL **version 14.** The BSR `postgres` user requires full access to the database, and additionally must be able to create the `pgcrypto` and `pg_trgm` extensions.

1. To configure Postgres, set the following helm values:

```
postgres:
  host: "postgres.example.com"
  port: 5432
  database: postgres
  user: postgres
```

2. Then create a k8s secret containing the postgres user password:

```
kubectl create secret --namespace bsr generic bufd-postgres --from-literal=password=<postgres password>
```

Configure Redis

The BSR requires a Redis instance. We support Redis **version 5.**

To configure Redis, create a k8s secret containing the address:

```
kubectl create secret --namespace bsr generic bufd-redis \
    --from-literal=address=redis.example.com:6379 # Host
```

Optionally, authentication and TLS for Redis are also supported. These can be set with the following Helm values:

```
redis:
  # Set to true to enable auth for redis.
  # The auth token will be read from the "auth" field in the "bufd-redis" secret
  auth: true
  tls:
    # Whether to use TLS for connecting to Redis
    # Set to "false" to disable TLS
    # Set to "local" to use certs from the "ca" field in the "bufd-redis" secret
    # Set to "system" to use the system trust store
      use: "false"
```

- If authentication is enabled, the redis auth string should be added to the `bufd-redis` secret in the `auth` field.
- If TLS is enabled and `use` is set to `local` , the CA certificate(s) to trust should be added to the `bufd-redis` secret in the `ca` field.

Example of a secret containing both an authentication token and a CA certificate:

```
kubectl create secret --namespace bsr generic bufd-redis \
    --from-literal=address=redis.example.com:6379 \ # Host
    --from-literal=auth=<redis auth string> \ # Auth string
    --from-file=ca=<redis ca.crt> \ # Redis CA certificate
```

Configure SAML authentication

The BSR supports authentication using an external identity provider (IdP), through Security Assertion Markup Language (SAML).

In the SAML IdP, create a new application to represent the BSR. It should return a single sign-on URL and IdP metadata. Either a public URL or raw XML can be specified for the SAML config. If SAML is being configured in Okta, please follow our guide.

1. To configure SAML authentication in the BSR, set the following Helm values:

```
auth:
  method: saml
  saml:
    # Endpoint where the XML metadata is available
    idpMetadataURL: "https://example-provider.com/app/12345/sso/saml/metadata"
    # If the authentication provider does not have a metadata url,
    # the raw XML metadata can be configured using the idpRawMetadata,
    # value instead.
    idpRawMetadata: ""
  # Optional
  # A list of emails which will be granted server admin permissions on login
  # Note that this list is case-sensitive
  autoProvisionedAdminEmails:
    - "user@example.com"
```

2. Additionally, a Kubernetes TLS secret named `bsr-saml-cert` containing a certificate pair is required in order for SAML to function. The certificate pair may be self-signed. Given the certificate pair, create the Kubernetes secret:

```
kubectl create secret --namespace bsr tls bsr-saml-cert \
  --cert=path/to/cert/file \
  --key=path/to/key/file
```

Configure Ingress

The BSR uses a Kubernetes Ingress resource to handle incoming traffic and for terminating TLS. The domain used here must match the `host` set in the Helm values above.

> ℹ️ TLS is required for the BSR to function properly.
> HTTP2 is preferred to allow for gRPC support.

```
bufd:
  ingress:
    enabled: true
    className: ""   # Optional ingress class to use
    annotations: {} # Optional ingress annotations
    hosts:
      - host: example.com
        paths:
          - path: /
            portName: http
    # Optional TLS configuration for the ingress.
    # May be omitted to configure TLS termination, depending on the ingress.
    # Requires a kubernetes TLS secret.
    tls:
      - secretName: bsr-tls-cert
        hosts:
          - example.com
```

If the load balancer does not support H2C, TLS can optionally be used for communication between the load balancer and the BSR by enabling TLS on the listening ports of the `bufd` application. This requires a Kubernetes TLS secret named `bsr-tls-cert`.

```
bufd:
  tls:
    enabled: true
    # Optional. Secret name for the TLS cert
    # secretName: bsr-tls-cert
  # Optional. Used to add annotations to the ingress service.
  # May be needed for some ingress controllers to function correctly.
  service:
    annotations: {}
```

5. Install the Helm Chart

After following the steps above, the set of Helm values should be similar to the example below:

```
host: example.com
imagePullSecrets:
  - name: bufpullsecret
storage:
  use: s3
  s3:
    bucketName: "my-bucket-name"
    endpoint: "s3.us-east-1.amazonaws.com"
    accessKeyId: "AKIAIOSFODNN7EXAMPLE"
    region: "us-east-1"
postgres:
  host: "postgres.example.com"
  port: 5432
  database: postgres
  user: postgres
auth:
  method: saml
  saml:
    idpMetadataURL: "https://example-provider.com/app/12345/sso/saml/metadata"
  autoProvisionedAdminEmails:
    - "user@example.com"
bufd:
  ingress:
    enabled: true
    hosts:
      - host: example.com
        paths:
          - path: /
            portName: http
    tls:
      - secretName: bsr-tls-cert
        hosts:
          - example.com
```

Using the `bsr.yaml` Helm values file, install the Helm Chart for the cluster:

```
helm install bsr \
oci://us-docker.pkg.dev/buf-images-1/bsr/charts/bsr \
--version 1.0.1 \
--namespace=bsr \
--values bsr.yaml
```

The BSR instance should now be up and running on `https://<host>`.

Optional configuration

Maintenance mode

The BSR has a maintenance mode in which the BSR will start up, but API calls are prevented and users of the web interface are informed that maintenance is in progress, and no database/object storage writes will occur. To enable this mode, set the `maintenance` Helm value:

```
maintenance: true
```

Feature flags

Certain BSR functionality is gated behind feature flags, which can be enabled through the `featureFlags` Helm value. The currently supported feature flags are:

```
featureFlags:
  # Prevent users from creating organizations in the BSR
  # Server admins can still create organizations when this flag is enabled
  disable_user_org_creation: true
```

Automatically adding members to organizations

To automatically add all members to an organization upon login, set the `auth.autoProvisionedMembershipOrganizations` Helm value:

```
auth:
  # Map of organizations which all members will be added to on login
  autoProvisionedMembershipOrganizations:
    exampleorg: ORGANIZATION_ROLE_MEMBER
```

Helm value reference

The `bsr` Helm Chart supports additional Helm values to adjust the deployment of the BSR.
Below is a reference of all supported values and their use.

⚠️ There may be additional low-level values defined in the `values.yaml` chart or Helm templates. Unless explicitly instructed to do so, please avoid editing any values that are not detailed below, as it could have an adverse affect on your installation.

```
host: "" # Hostname that the BSR will be served from
maintenance: false # BSR maintenance mode
imagePullSecrets: [] # Image pull secrets to use for deployments
auth:
  method: saml
  saml:
    # Endpoint where the XML metadata is available
    idpMetadataURL: "https://example-provider.com/app/12345/sso/saml/metadata"
    # If the authentication provider does not have a metadata url,
    # the raw XML metadata can be configured using the idpRawMetadata
    # value instead.
    idpRawMetadata: ""
```

```
  # A list of emails which will be granted server admin permissions on login
  # Note that this list is case-sensitive
  autoProvisionedAdminEmails: []
  # - "user@example.com"
  # Map of organizations which all members will be added to on login
  autoProvisionedMembershipOrganizations: {}
  # exampleorg: ORGANIZATION_ROLE_MEMBER
featureFlags:
  # Prevent users from creating organizations in the BSR
  # Server admins can still create organizations when this flag is enabled
  # disable_user_org_creation: true
postgres:
  # host: ""
  # port: 5432
  database: postgres
  user: postgres
redis:
  # Set to true to enable auth for redis.
  # The auth token will be read from the "auth" field in the "bufd-redis" secret
  auth: false
  tls:
    # Whether to use TLS for connecting to Redis
    # Set to "false" to disable TLS
    # Set to "local" to use certs from the "ca" field in the "bufd-redis" secret
    # Set to "system" to use the system trust store
    use: "false"
storage:
  use: s3
  s3:
    bucketName: ""
    endpoint: ""
    accessKeyId: ""
    region: ""
    forcePathStyle: false # Optional, use path-style bucket URLs (http://s3.amazonaws.com/BUCKET/KEY)
    insecure: false # Optional, disable TLS
bufd:
  service:
    type: ClusterIP
    annotations: {}
  serviceAccount:
    annotations: {}
  deployment:
    args: []
    replicaCount: 2
    podAnnotations: {}
    # extraEnv: []
    #   - name: GOMEMLIMIT
    #     value: 1792MiB # ~90% of 2Gi (2048*.9-((2048*.9)%64))
    resources: {}
      # requests:
      #   cpu: 800m
      #   memory: 2Gi
      # limits:
      #   memory: 2Gi
    nodeSelector: {}
    affinity: {}
    tolerations: {}
  podDisruptionBudget:
    enabled: false
    # -- Number of pods that are available after eviction as number or percentage (eg.: 50%)
    # Defaults to 0 if not specified
    minAvailable: ""
    # -- Number of pods that are unavailable after eviction as number or percentage (eg.: 50%).
    ## Has higher precedence over `minAvailable`
    maxUnavailable: ""
  tls:
    enabled: false
    secretName: bsr-tls-cert
  # The number of trusted hops in the XFF header
  numTrustedHops: 1
  # useRemoteAddress ignores the numTrustedHops setting and the X-Forwarded-For header entirely,
  # and uses the value of RemoteAddr in the http.Request to determine the trusted client IP instead
```

```
  # this should only be used when the application is open to the internet
  # useRemoteAddress: true
  ingress:
    enabled: false
    className: ""
    annotations: {}
    # tls:
    #   - secretName: bsr-tls-cert
    #     hosts:
    #       - buf.domain
    # hosts:
    #   - host: buf.domain
    #     paths:
    #       - path: /
    #         portName: http
  image:
    repository: us-docker.pkg.dev/buf-images-1/bsr/images/bufd
    pullPolicy: IfNotPresent
    tag: "1.0.1"
bufjavacompilerd:
  initialHeapMB: 512
  maxHeapMB: 1536 # This value should be 3/4 of the memory limit given to the deployment, if there is one
  service:
    type: ClusterIP
    annotations: {}
  serviceAccount:
    annotations: {}
  deployment:
    args: []
    replicaCount: 2
    podAnnotations: {}
    extraEnv: []
      # Set GOMEMLIMIT to less than the memory resource request/limits,
      # so that the binary does not think it has more memory available to
      # it than it does.
      # In our case, we want the JVM to be able to use 3/4 of the
      # available memory for its heap, so we allocate 1/4 of the
      # available memory for the Go binary.
      # - name: GOMEMLIMIT
      #   value: 512MiB
    # On changing requests.memory or limits.memory, you may also want
    # to consider updating the GOMEMLIMIT above and the
    # initial_heap_mb and max_heap_mb in the configuration.
    resources: {}
      # requests:
      #   cpu: 200m
      #   memory: 2Gi
      # limits:
      #   memory: 2Gi
    nodeSelector: {}
    affinity: {}
      # podAntiAffinity:
      #   preferredDuringSchedulingIgnoredDuringExecution:
      #     - podAffinityTerm:
      #         labelSelector:
      #           matchLabels:
      #             app: bufd
      #         topologyKey: kubernetes.io/hostname
      #       weight: 5
    tolerations: {}
  podDisruptionBudget:
    enabled: false
    # -- Number of pods that are available after eviction as number or percentage (eg.: 50%)
    # Defaults to 0 if not specified
    minAvailable: ""
    # -- Number of pods that are unavailable after eviction as number or percentage (eg.: 50%).
    ## Has higher precedence over minAvailable
    maxUnavailable: ""
  image:
    repository: us-docker.pkg.dev/buf-images-1/bsr/images/bufjavacompilerd
    pullPolicy: IfNotPresent
    tag: "1.0.1"
```

```
ociregistry:
  service:
    type: ClusterIP
    annotations: {}
  serviceAccount:
    annotations: {}
  deployment:
    args: []
    replicaCount: 1
    podAnnotations: {}
      # prometheus.io/port: "5001"
      # prometheus.io/scrape: "true"
    extraEnv: []
    # to avoid load balancing issues, the http secret must be
    # set when there is more than 1 replica of the registry:
    # https://github.com/distribution/distribution/blob/bac7f02e02a1ba4b8e4a5a79a5a0b5b4055d2b9a/docs/deploying.md#load-b
alancing-considerations
    # - name: REGISTRY_HTTP_SECRET
    #   valueFrom:
    #     secretKeyRef:
    #       name: oci-registry-http-secret
    #       key: oci-registry-http-secret
    resources: {}
      # requests:
      #   cpu: 300m
      #   memory: 200Mi
      # limits:
      #   memory: 200Mi
    # - name: "MY_VAR"
    #   value: "value"
    # - name: VAR_FROM_SECRET
    #   valueFrom:
    #     secretKeyRef:
    #       name: secret-name
    #       key: secret_key
    nodeSelector: {}
    affinity: {}
    tolerations: {}
  podDisruptionBudget:
    enabled: false
    # -- Number of pods that are available after eviction as number or percentage (eg.: 50%)
    # Defaults to 0 if not specified
    minAvailable: ""
    # -- Number of pods that are unavailable after eviction as number or percentage (eg.: 50%).
    ## Has higher precedence over `minAvailable`
    maxUnavailable: ""
  image:
    repository: registry
    pullPolicy: IfNotPresent
    tag: "2.8.1"
sandbox:
  service:
    type: ClusterIP
    annotations: {}
  serviceAccount:
    annotations: {}
  deployment:
    args: []
    replicaCount: 2
    podAnnotations: {}
    extraEnv: []
      # - name: GOMEMLIMIT
      #   value: 1792MiB # ~90% of 2Gi (1024*.9-((1024*.9)%64))
    resources: {}
      # requests:
      #   cpu: 1000m
      #   memory: 2Gi
      # limits:
      #   memory: 2Gi
    nodeSelector: {}
    affinity: {}
      # podAntiAffinity:
```

```
    #   preferredDuringSchedulingIgnoredDuringExecution:
    #     - podAffinityTerm:
    #         labelSelector:
    #           matchLabels:
    #             app: sandbox
    #         topologyKey: kubernetes.io/hostname
    #       weight: 5
  tolerations: {}
podDisruptionBudget:
  enabled: false
  # -- Number of pods that are available after eviction as number or percentage (eg.: 50%)
  # Defaults to 0 if not specified
  minAvailable: ""
  # -- Number of pods that are unavailable after eviction as number or percentage (eg.: 50%).
  ## Has higher precedence over minAvailable
  maxUnavailable: ""
image:
  repository: us-docker.pkg.dev/buf-images-1/bsr/images/bufsandboxd
  pullPolicy: IfNotPresent
  tag: "1.0.1"
runtime:
  image:
    repository: us-docker.pkg.dev/buf-images-1/bsr/images/sandboxruntime
    pullPolicy: IfNotPresent
    tag: "1.0.1"
  deployment:
    resources: {}
      # requests:
      #   cpu: 800m
      #   memory: 2Gi
```