Microsoft

SQL Server Big Data Clusters Security Best Practices for Deployments on Red Hat OpenShift

Technical whitepaper

Published: June 2020
Applies to: Microsoft SQL Server 2019 CU5

For the latest deployment information, see:
https://aka.ms/sql-bdc-openshift-docs

Contents

Introduction

Microsoft SQL Server Big Data Clusters (BDC) is a new feature introduced with the SQL Server 2019 release that enables a new deployment pattern for SQL Server by adding Apache Hadoop Distributed File System (HDFS) and Apache Spark for big data storage and analytics. BDC runs as a set of purpose-built Linux container images on Kubernetes. Container software allows packaging and isolation of application/database services with their dependencies, making it easy to build & deploy them anywhere. Kubernetes is the open source container orchestration framework that automates deployment and lifecycle management of containers at scale With BDC you can run analytical workloads at any scale, on an integrated container platform, designed to derive intelligent insights from structured and unstructured data sources.

The Red Hat OpenShift platform is the industry leading Kubernetes based container platform that provides self-service seamless experience for software developers, data scientists, data analysts, database administrators, etc. to help accelerate development and delivery of intelligent applications. OpenShift is the supported reference platform for running SQL Server in containers.

As part of a strategic partnership between the two companies, Red Hat and SQL Server engineers worked together closely to enable support for BDC on OpenShift starting with SQL Server 2019 CU5 release. With BDC on OpenShift, enterprise customers can deploy fully supported data analytics on a container and Kubernetes stack that follows best practices and guidance unique to the OpenShift environment. This allows organizations to achieve the desired scalability, flexibility, security, and portability associated with containers and Kubernetes in the cloud-native era.

As a general best practice, an optimal security design is one in which you have multiple layers of protection (called "defense in depth"). One layer of protection can be to run with as few Linux capabilities as possible. A second is to not run the PID1 process in the container as root. A third layer of protection is to use SELinux on the host to prevent the root user from performing any changes to the host file system or to other processes/users.

The deployment model for BDC has been enhanced to allow you to follow this guidance, so that privileged containers, deployed as part of BDC, are no longer *required*. In addition, BDC containers now run as a non-root user by default. For BDC there are cases where elevation of privileges is necessary to improve process isolation within specific containers. For example, there are processes running within BDC containers like SQL Server or Yarn that run user provided code. To better isolate that user provided code, and make sure it cannot access other user's provided code or data - a separate process is spun off for each execution of that user provided code. This creates a level of isolation where each process runs as its own different user ID and therefore can only access data within the respective process. More, it cannot access data outside of the process boundaries, including other processes or domain system process. To provide this level of security isolation, specific system services within BDC will require to temporarily elevated privileges so they can launch these processes under their own UID.

This document explains *what* additional permissions are necessary and *why* BDC requires certain security policies. In addition to the use cases, this document will specify which containers and applications require these elevated capabilities.

CAP_SETUID and CAP_SETGID Capabilities

Some system services running inside BDC containers require the use of the CAP_SETUID and CAP_SETGID capabilities to launch processes under their own low-privileged UID. The use of multiple UIDs within BDC containers increases security by reducing the possibility that code executed under one user account can affect another user or the system. Using multiple UIDs also reduces the possibility that a vulnerability in a container service will allow sensitive container information to be accessed (e.g. Active Directory credentials). The system services that are granted the CAP_SETUID and CAP_SETGID capabilities run under a low-privileged UID under normal operation. The CAP_SETUID and CAP_SETGID capabilities are only used while executing the operations specified below.

Note that while a process could theoretically be changed to run as root on a host, SELinux prevents taking read/write actions on the host. SELinux is enabled by default in OpenShift.

CAP_SETUID and CAP_SETGID Use Cases

The CAP_SETUID and CAP_SETGID capabilities are used by BDC in the following use cases:

Launching services that host user code

Multiple BDC services provide facilities to launch and execute user-provided code. This user-provided code runs in the same container as its launching service. In order to protect the launching service process, as well as other system and user-provided processes, from malicious user-provided code, the CAP_SETUID and CAP_SETGID capabilities are used to instruct OpenShift to launch processes executing user code under their own unique low-privileged UID. The services that can launch programs to execute user code are: YARN (running Spark or MapReduce jobs) and SQL Server Extensibility (R, Python, and Java jobs typically used for Machine Learning model training or inferencing). If SQL Server Extensibility is not needed, it can be disabled.

Container Agent

Each BDC container has an *agent* that is responsible for monitoring and managing that container. The container agent is responsible for joining the container to Active Directory (optional), performing health checks, and delivering configuration/data files securely. The container agent uses the CAP_SETUID and CAP_SETGIDID privileges in two situations:

- to start the service management program (supervisor) under a low-privileged UID
- to install the cluster root certificate authority certificate within the container

Service Management

Each BDC container has a supervisor program which is responsible for launching application services (e.g. SQL Server, YARN NodeManager). To protect application services from one another, and to prevent those services from accessing sensitive files, each service is launched under its own unique low-privileged UID. Because the supervisor program itself is complex, we use a service launcher program to launch services. The service launcher has been granted CAP_SETUID and CAP_SETGID capabilities and can only be used to start a fixed set of services specified in a configuration file. This configuration file is built into the product and cannot be changed by any user other than root.

Among the processes that the service launcher is starting is the setup step that executed when the user creates a new app through application deployment feature. User can opt in to install additional packages to be used by the application and this will require root permissions. Execution of this step as root does not have any system wide impact and it is confined to the application only. User code deployed as part of the application will run as low privilege user.

Programs Granted CAP_SETUID/CAP_SETGID

Only a limited set of programs are granted CAP_SETUID and CAP_SETGID capabilities. These programs only utilize these capabilities in the circumstances described above.

Program	Default UID	Use Case
Yarn NodeManager	yarn	Launching services that host user code
SQL Server Launchpad	mssql	Launching services that host user code
Container agent	agent	Container Agent
Service Launcher	supervisor	Service Management

CAP_CHOWN Capability

The container agent in BDC is responsible for configuring service and security settings within a container during startup. The agent needs to write to files owned by different UIDs and to set ownership on newly created files and directories. Even though the process can be elevated to root on the host, changes to the host file system are blocked by SELinux.

CAP_CHOWN Use Cases

The CAP_CHOWN capability is used by the container agent in the following use cases:

Writing to /etc/hosts and /etc/resolv.conf

The container agent writes information into the /etc/hosts and /etc/resolv.conf files to configure name resolution to work properly in an Active Directory environment when Active Directory integration is used. We only modify the hosts and resolv.conf within the container file system and this change only affects DNS resolution from inside of the container. In addition, SELinux blocks changes to files at the host level. We are looking at another option for longer term to eliminate the need for CAP_CHOWN/editing these name resolution files.

Configuring file system permissions for files and services

The container agent needs to set permissions under the following use cases:

- Set permissions on directories owned and managed by services so that the service process can access them at startup. For example: /var/opt/mssql is set to be owned by the ‘mssql’ user which is the user of the SQL Server service process.
- Set permissions on log directories such as /var/log/mssql so that the log directory can be shared between the SQL Server service process and the FluentBit service process in the logging sidecar container.

- Set permissions on configuration and security files belonging to a particular service. For example: the SSL certificates and service keytabs for HDFS datanode are configured to be readable only by the ‘hdfs’ user.

Programs Granted CAP_CHOWN

Only a limited set of programs are granted the ability to use the CAP_CHOWN capabilities. These programs only use these capabilities in the circumstances described above.

Program	Default UID	Use Case
Container agent	agent	Writing to /etc/hosts and /etc/resolv.conf Configuring file system permissions for files and services

CAP_SYS_PTRACE Capability

The CAP_SYS_PTRACE capability is an optional capability that can be enabled by a customer to allow the SQL Server BDC cluster controller and SQL Server to save crash dumps to disk when a program crash occurs. It is recommended to run with the CAP_SYS_PTRACE capability enabled so that dumps can be captured if there is a crash and the provided to Microsoft Support for troubleshooting.

The only way to exploit this would be to connect back to the process that has CAP_SYS_PTRACE, but we are not pass CAP_SYS_PTRACE to the user code process.

CAP_SYS_PTRACE Use Cases

The SQL Server BDC cluster controller and SQL Server use the CAP_SYS_PTRACE capability under the following use cases:

Taking a crash dump of a failed process

When a critical error or memory corruption occurs in either the BDC controller a SQL Server process, an external process called sqldumper is used to take a crash dump of the affected process. This process collects the memory and other debugging information about the situation so that Microsoft support engineers can help customers resolve product issues.

Programs Granted CAP_SYS_PTRACE

Only a limited set of programs are granted the ability to use the CAP_SYS_TRACE capabilities. These programs only use these capabilities in the circumstances described above.

Program	Default UID	Use Case
sqldumper	mssql, controller	Taking a crash dump of a failed process

Security impact of enabling capabilities

The CAP_SETUID, CAP_SETGID, and CAP_CHOWN capabilities are enabled to provide a secure environment within each container. They are used to prevent services from interfering with each other or interfering with system programs such as the container agent. They are also used to protect services from user code executing in the container. These capabilities are only granted to control programs which are responsible for managing cluster provided services (e.g. SQL Server Extensibility or YARN NodeManager) or managing the container itself. Programs that are not granted these capabilities cannot use them. There are three places in the product where user-provided code can be executed:

- Machine Learning Services in SQL Server – This enables executing R, Python, and Java scripts in the context of a T-SQL query typically for the purposes of training a machine learning model or to score data against a model. ML Services can be easily ￼enabled or disabled (default) in BDC. Additionally, even when the feature is enabled, concurrent sessions and external scripts execution are restricted to one unique login at a time to guarantee complete user isolation.

- Spark jobs – You can control what users have access to launch Spark jobs in BDC by creating an AD group which has no users in it and assign that group to the ***clusterUsers*** deployment configuration at deployment time. Only users in this group and the AD group associated with the ***clusterAdmins*** deployment configuration setting can execute Spark jobs. See the Active Directory integration article for more information.

- Applications deployed using the app deploy interfaces – This optional feature enables deployment of R, Python, SSIS and MLeap applications on BDC by providing interfaces to create, manage, and run applications. Only the setup step of the application requires running as a high privileged user since it installed additional packages that the application will use. Other user code deployed as part of the application will run as low privilege user.

The CAP_SYS_PTRACE capability is necessary to allow crash dumps to be taken of the SQL Server BDC controller and SQL Server when they crash. When a crash occurs, a process called sqldumper attaches to the crashing process so that it can save the memory of the process to disk. Only the sqldumper process is granted the CAP_SYS_PTRACE capability and can do this. CAP_SYS_PTRACE is optional but disabling it will prevent dumps from being captured for troubleshooting purposes.

The capabilities assigned do not grant any permissions to resources or facilities outside of the container. Barring unknown defects in Kubernetes or the Linux kernel, it should not be possible to escape a container or cause issues on a host by assigning these capabilities. SQL Server BDC undergoes extensive security testing during each release to make sure that everything running in a container is secure. Checking for weak permissions and ways to elevate privileges are two of the many criteria validated.

In OpenShift, the actions that containers within a pod can perform and what are they able to access are controlled using security context constraints (SCCs). OCP provides a set of predefined SCCs that can be used as is, modified or extended by any administrator. As a best practice, you should not modify the built-in SCCs to add the necessary capabilities for BDC. Creating a custom SCC to be used for the BDC project will ensure that other resources running within other project/namespaces in the same OpenShift cluster are not impacted. *Appendix 1* includes a sample custom SCC that specifies the additional capabilities required for BDC to deploy successfully.

Conclusion

To summarize, BDC is a new feature introduced with the SQL Server 2019 release that enables a new deployment pattern for SQL Server by adding Apache Hadoop Distributed File System (HDFS) and Apache Spark for big data storage and analytics. BDC runs as a set of purpose-built Linux container images on Kubernetes. Red Hat OpenShift is the industry leading Kubernetes based container platform that provides self-service seamless experience for software developers, data scientists, data analysts, database administrators, etc. to help accelerate development and delivery of intelligent applications. With BDC on OpenShift, enterprise customers can deploy fully supported data analytics on containers and Kubernetes stack.

The deployment model for BDC has been enhanced to allow you to follow this guidance, so that privileged containers, deployed as part of BDC, are no longer *required*. In addition, BDC containers now run as a non-root user by default. For BDC there are cases where elevation of privileges is necessary to improve process isolation within specific containers. This document explained *the* additional permissions that are necessary and *why* BDC requires certain security policies. In addition to the use cases, this document specified which containers and applications require these capabilities.

As next steps, please see below for additional resources to learn more about how to deploy SQL Server Big Data Clusters on OpenShift and start running your analytics workloads on this comprehensive and secure platform.

Additional resources

- Learn more about SQL Server Big Data Clusters
- Learn more about containers and OpenShift security best practices: OpenShift Protects Against Nasty Container Exploit / Ten Layers of Container Security / Container Hosts and Multi-tenancy / Managing Security Context Constraints
- Deploy SQL Server Big Data Clusters on Red Hat OpenShift

Acknowledgements

Bringing SQL Server and Big Data Clusters to the OpenShift Container Platform has been a real team effort. Red Hat provided our team with valuable help, bootstrapping our initial efforts, as well as providing best practice guidance during implementation. Security and trust are critical for both companies and so we appreciate the valuable input and contributions of Dan Walsh, Senior Distinguished Engineer at Red Hat, and Michael Nelson, Principal Software Engineering Manager at Microsoft, who collaborated on this technical paper.

Appendix 1 – Custom BDC SCC

```
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: true
allowPrivilegedContainer: false
allowedCapabilities:
- SETUID
- SETGID
- CHOWN
- SYS_PTRACE
apiVersion: security.openshift.io/v1
defaultAddCapabilities: null
fsGroup:
  type: RunAsAny
kind: SecurityContextConstraints
metadata:
  annotations:
    kubernetes.io/description: SQL Server BDC custom SCC is based on 'nonroot' built-in
SCC plus additional capabilities.
  generation: 2
  name: bdc-scc
readOnlyRootFilesystem: false
requiredDropCapabilities:
- KILL
- MKNOD
runAsUser:
  type: MustRunAsNonRoot
seLinuxContext:
  type: MustRunAs
supplementalGroups:
  type: RunAsAny
volumes:
- configMap
- downwardAPI
- emptyDir
- persistentVolumeClaim
- projected
- secret
```

Microsoft

Copyright