bitwarden | Help Center Article

SECRETS MANAGER > GET STARTED

Developer Quick Start

View in the help center:
https://bitwarden.com/help/developer-quick-start/

Developer Quick Start

Bitwarden Secrets Manager enables developers, DevOps, and cybersecurity teams to centrally store, manage, and deploy secrets at scale. The Secrets Manager CLI is your primary vehicle for injecting secrets into your applications and infrastructure through an authenticated machine account.

In this article, we'll demonstrate use of the Secrets Manager CLI by looking at a few ways to retrieve database credentials stored in your vault to be injected at container runtime for a Bitwarden Unified Docker image.

> **Tip**
>
> If you're looking for SDK information and language wrappers for Secrets Manager functionality, refer to this article.

If you haven't already gone through the Secrets Manager Quick Start article, we recommend doing so before reading on.

Basic tutorial

In this most simple example, you'll retrieve database credentials stored in your vault and store them as temporary environment variables on a Linux system. Once stored, you'll inject them at runtime inside a `docker run` command. To do this, you'll need to have installed:

- Bitwarden Secrets Manager CLI
- Docker
- A command-line JSON processor like jq

Authenticate

The Secrets Manager CLI can be logged in to using an access token generated for a particular machine account. This means that **only secrets and projects which the machine account has access to** may be interacted with using the CLI (learn more about machine account permissions). There are a number of ways to authenticate a CLI session, but for the simplest option do so by saving an environment variable `BWS_ACCESS_TOKEN` with the value of your access token, for example:

```bash
export BWS_ACCESS_TOKEN=0.48c78342-1635-48a6-accd-afbe01336365.C0tMmQqHnAp1h0gL8bngprlPOYutt0:B3h5D+YgLvFiQhWkIq6Bow==
```

Retrieve the secret

Next, use the following command to retrieve your database username and store it as a temporary environment variable. In this example, `fc3a93f4-2a16-445b-b0c4-aeaf0102f0ff` represents the specific identifier for the database username secret:

```bash
export SECRET_1=$(bws secret get fc3a93f4-2a16-445b-b0c4-aeaf0102f0ff | jq '.value')
```

This command will save the `value` of your secret to a temporary environment variable, which will be cleared on system reboot, user logout, or in any new shell. Now, run the same command for the database password:

```bash
export SECRET_2=$(bws secret get 80b55c29-5cc8-42eb-a898-acfd01232bbb | jq '.value')
```

Inject the secret

Now that your database credentials are saved as temporary environment variables, they can be injected inside a `docker run` command. In this example, we've omitted many of variables required by Bitwarden Unified to emphasize the injected secrets:

```bash
docker run -d --name bitwarden .... -env BW_DB_USERNAME=$SECRET_1 BW_BD_PASSWORD=$SECRET_2 .... bit
warden/self-host:beta
```

When this command is run, your Docker container will start up and inject your database credentials from the temporarily stored environment variables, allowing you to securely spin up Bitwarden Unified without passing sensitive values as plaintext.

Advanced tutorial

In this example, you'll use the Secrets Manager CLI in your Docker image to inject database credentials stored in your vault at runtime. You'll accomplish this by manipulating your Dockerfile to install the CLI on the image, instead of on the host, and to retrieve the database credentials at container runtime. You'll then prepare your environment variables file for injection and string it all together by running a container.

Setup your Dockerfile

To install the Secrets Manager CLI in your Docker image, you'll need to add the following to your Dockerfile:

```
Plain Text
```

```
# Install dependencies
ENV DEBIAN_FRONTEND=noninteractive
RUN apt-get update && \
  apt-get install -y \
  ca-certificates \
  curl \
  jq \
  unzip && \
  rm -rf /var/lib/apt/lists/*

# Download bws
RUN curl -LO https://github.com/bitwarden/sdk/releases/download/bws-v1.0.0/bws-x86_64-unknown-linux
-gnu-1.0.0.zip && \
  unzip bws-x86_64-unknown-linux-gnu-1.0.0.zip -d /usr/local/bin/ && \
  rm -f bws-x86_64-unknown-linux-gnu-1.0.0.zip

# Add anything else you will need to your image

# Entrypoint script will retrieve secrets at runtime
COPY ./entrypoint.sh /
ENTRYPOINT ["/entrypoint.sh"]
```

Next, use an `entrypoint.sh` file in order to inject secrets at run time. One method is to construct `RUN` statements in your `entrypoint.sh` file that will retrieve each credential . This however, is not the only method you'd be able to implement:

```
Plain Text

#!/usr/bin/env bash
# One way to retrieve individual secrets is to use the `get` command and extract the value:
SECRET_1=$(bws secret get fc3a93f4-2a16-445b-b0c4-aeaf0102f0ff | jq '.value')

# Another option., this method is sensitive to spaces in the secret name. See the `run` command doc
umentation for more options
bws run -- 'echo $SECRET_NAME'

# Run your project
```

These `RUN` statements will prompt your Dockerfile to retrieve the indicated secrets, where `fc3a93f4-2a16-445b-b0c4-aeaf0102f0ff` represents the secret's specific identifier. The other option included in the code example represents the secret's name, `'echo $SECRET_NAME'`.

Build the image

To build the docker image, first make `entrypoint.sh` executable:

```
Plain Text

chmod +x ./entrypoint.sh
```

Build the image:

```
Plain Text

docker build -t image-name
```

Run the container

Now that your database credentials are primed and ready for injection, start up your container:

```
Bash

docker run --rm -it -e BWS_ACCESS_TOKEN=<your-access-token> image-name
```

When this command is run, your Docker container will start up and inject your database credentials from the values retrieved by the container, allowing you to securely spin up Bitwarden Unified without passing sensitive values as plaintext.