

Guest App Onboarding document

Onboarding New Guest Applications on the BDHP (Baxter Digital Health Platform) Platform

Welcome Aboard

Summary

Cloud Environments

DevOps Application Lifecycle Process for Kubernetes Applications

Job Role Management

DevOps/GitHub Setup

ACR Setup

- ACR Access For Apps Using Different Toolset (Type 3)

- ACR Versioning

 - Master Chart Versioning

 - Guest Application Chart Versioning

Application Deployment Type

- Generic Helper Service

- Unique App Specific Helper Service

Databases

Specialty Azure Resources

Deployment Strategy/Timing

Kubernetes Information

- Namespaces

- Secrets

- Endpoints (http, and data)

 - HTTP specific traffic only

 - HTTP and TCP traffic with Server Name Indication (SNI)

 - HTTP and TCP traffic without SNI

Networking

- Firewall/NSG

- Cloudflare

- Public IP (Internet Protocol)

- Routing

- VPN

Additional Details

- Naming standards

- Backup/Recovery

- Build Options

- Code Coverage

- SonarCloud

- Docker

- Helm

Specific Application Type Details

- NuGet

- Dotnet

- Maven

- Gradle

[Vault Integration Guide](#)

[Downloads](#)

[Authentication \[cli\]](#)

[List Keys \[cli\]](#)

[Get Values \[cli\]](#)

[Support Information](#)

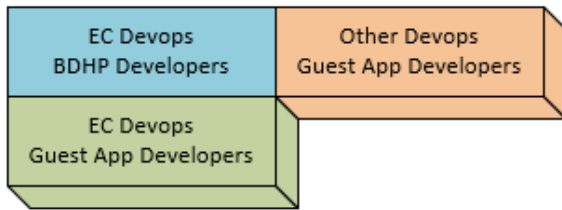
Welcome Aboard [🔗](#)

We have worked hard to bring you a platform to integrate all the great products of Baxter into one central communication hub. For all the applications to be able to work together, we have developed a methodology to ensure that the integration can be as smooth as possible. While some things do appear to be minor, every bit of this build, and in some cases deployment, is done by code and needs to be accurate. This document helps ensure that all application interactions and expectations are fully understood. Guest application teams will need to fully complete the onboarding questionnaire. This is necessary to ensure that each application has the correct setup to smoothly integrate with the platform.

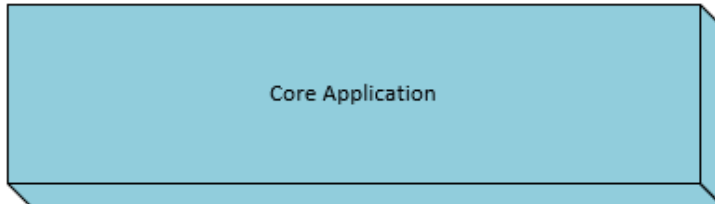
The Visual Organization of Application Integration diagram below depicts the different types of integrations. It is expected that guest applications will fit into one of these integration types.

Visual Organization of Application Integration

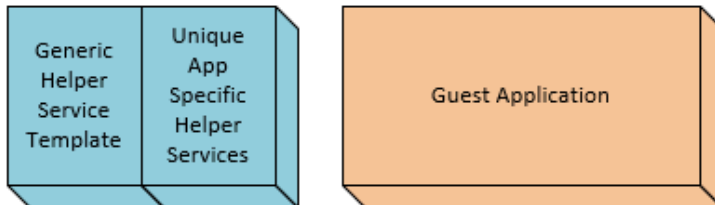
Color Key



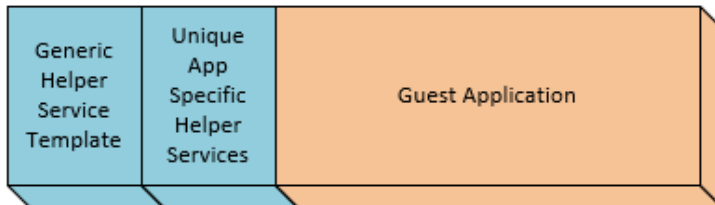
Type 1 - Core BDHP Application



Type 2 – Interfaced Application



Type 3 – Integrated Guest Application – Other Devops



Type 4 – Integrated Guest Application – EC Devops



Summary [🔗](#)

We have provided this document to assist in onboarding and understanding how your application can integrate with the BDHP cloud platform. We utilize Infrastructure as Code (IAC) managed resources to provide a consistent and manageable experience for development, changes, and release management. Outlined below is the process for needed information and details how the environment will be made available for guest application teams to use. Some processes are universal among all of the application types, while others only apply depending on the support model used. (See Visual Organization of Application Integration)

Cloud Environments [🔗](#)

As a part of an agile process, we provide multiple environments for the applications.

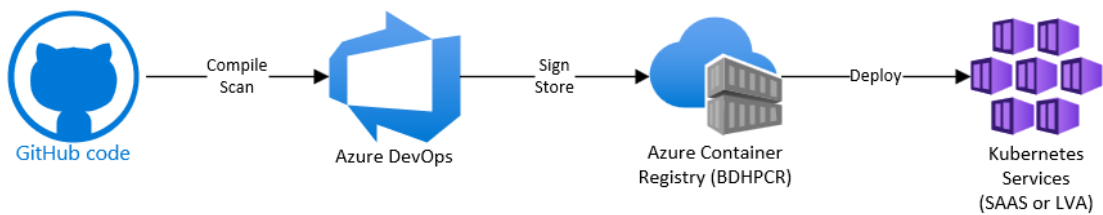
- DEV
 - The DEV environment is the environment to which all commits are posted immediately.
 - Developers have a high level of control of resources.
 - Quality Engineers (QEs) have a high level of control of resources.
- DEVF
 - The DEV Future environment is the environment, which is used for future development.
 - Developers have a high level of control of resources.
 - Quality Engineers (QEs) have a high level of control of resources.
- TEST
 - This is for functional testing and a chance to get the QE's involved.
 - Developers have a high level of control of resources.
 - QEs have a high level of control of resources.
- STAGE
 - This is for formal testing.
 - Developers have read only access.
 - QEs have a high level of control of resources.
- PERF
 - This is for performance testing as well as test PROD issues on older versions.
 - Developers have a high level of control of resources.
 - QEs have a high level of control of resources.
- SBX
 - This is a Prod level install.
 - Intended for customers to use to test configurations.
 - **No one** has a high level of control of resources.
 - QEs have access during validation.
- SBXGA
 - This is a Prod level install.
 - This is a sandbox environment dedicated for application teams can use for development and testing configurations.
 - **No one** has a high level of control of resources.
 - QEs have access during validation.
- PROD
 - This is a Prod level install.
 - This is the primary production level environment.
 - **No one** has a high level of control of resources.
 - QEs have access during validation.

Type 1,3,4 applications can make use of all environments. Type 2 applications are recommended to only use STAGE and above environments as the stability is much higher.

Being a Prod level install means the software in this environment is released through the software development process.

DevOps Application Lifecycle Process for Kubernetes Applications

The lifecycle of an application begins with the source code. We utilize Github to store source code. That code is scanned and then compiled by Azure DevOps Pipelines into usable containers or images. From there the images are signed with a Baxter signing certificate and then stored into a BDHP Container Registry. This makes software downloadable by the installation processes. The container images can be installed to an on-premises appliance (LVA), an Azure SAAS environment, or a developer machine as needed.



Job Role Management

The technical process of onboarding needs information to integrate into BDHP’s IAC platform. Guest application team members will need to have job role groups set up in the HRC active directory even after we have fully integrated logins with Baxter credentials. (See examples below).

There are some pieces of the environment that exist in the Hill-Rom Azure that need HRC credentials.

Guest application teams should work with Joe Wolf’s team to facilitate getting invites for team members as well as getting GG-ROLE groups created for the various job roles. This setup piece is critical to both the Role Based Access Control (RBAC) and the applications rights within the platform, so this must be completed before applications can be officially onboarded. See this [document](#) and the ‘List of Enterprise Connectivity (EC) Product Families and their Products’ as a guide. Below is an example of role groups for EC.

Groups | All groups

Hillrom - Azure Active Directory

EXAMPLE ONLY

New group

Download groups

Refresh

Manage view

Delete

Got feedback?

GG-ROLE-ec-rd-bdhp

Search mode

Contains

10 groups found

	Name	Object Id	Group type	Membership type
	GG-ROLE-ec-rd-bdhp-devops-ReadOnly	68552d50-1cb8-4249-8bd7-8d502b8016dd	Security	Assigned
	GG-ROLE-ec-rd-bdhp-automationlead	123cabf7-82d0-4e46-a8a3-c762e2608185	Security	Assigned
	gg-role-ec-rd-bdhp-auditor	2cc8b2fb-6d62-4b23-87f6-e9ea75bb0c9c	Security	Assigned
	GG-ROLE-ec-rd-bdhp-projectmanager	74469058-836d-4a1b-8e33-37c8d88e2c8f	Security	Assigned
	GG-ROLE-ec-rd-bdhp-quality	80e37690-b2db-41a4-9d09-ea6940666b0b	Security	Assigned
	GG-ROLE-ec-rd-bdhp-software	980f6e6e-bfdf-47f4-90b9-8e46dfe716db	Security	Assigned
	GG-ROLE-ec-rd-bdhp-qualitylead	a65cb6d1-ea31-4b4b-abae-b2d88a6d58bc	Security	Assigned
	GG-ROLE-ec-rd-bdhp-devops	b3fd61dc-7a79-470f-bfdf-213342c1320d	Security	Assigned
	GG-ROLE-ec-rd-bdhp-automation	bd9046a-cd18-40d0-ae88-244fa99fc96c	Security	Assigned
	GG-ROLE-ec-rd-bdhp-softwarelead	f1e45db4-6caa-47d2-8f71-4c70c9083033	Security	Assigned

DevOps/GitHub Setup

A standardized location for your builds and releases will be set up to onboard the IAC automation. Every pipeline needs a source code repo to run against. After we have processed the questionnaire, a GitHub repository will be created for any application for which the EC team will be providing direct DevOps support, and each will be automatically integrated with the Azure DevOps (AZDO) pipelines. Any

applications that are supported by an external DevOps team or are interfaced applications will still have a DevOps project and pipeline created to facilitate any needed certificates, service principals, or DNS (Domain Name Service) updates necessary. Those teams will provide any details for pipeline and repository locations. For interfaced applications, shared resources will be accessible via connection to the HashiCorp Vault (HCV) within the Kubernetes cluster for things like URL's, username/passwords, and certificates. A client certificate and URL will be provided to the guest application team (one for non-Prod and one for Prod), which will allow authentication to the specific HCV to retrieve these resources in a secure and programmatic way. [TLS Certificates - Auth Methods | Vault | HashiCorp Developer](#)

For all applications that are integrated with the system, an AZDO project will be created for your team with GitHub integration built in which can be found here [Projects - Home \(azure.com\)](#). The project name will be based on the guest application team's name. For example, if Enterprise Connectivity had a guest application, the project would be (IAC – BDHP GA – EC). Each guest application team that is onboarded will also have a GitHub repo created and will be linked to a build definition in the project in AZDO. A sample application will be created for your team as a visual guide. We can remove the sample application once comfortable with the final setup. Here is an example of the EC team which has 2 applications integrated: testpoc1 and testpoc2.

Azure DevOps hrc-zenith / IAC - BDHP GA - EC / Pipelines

IAC - BDHP GA - EC

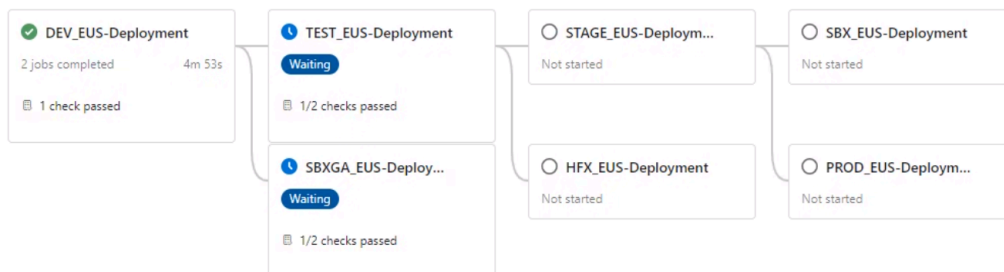
Overview
Pipelines
Environments
Releases
Library
Task groups
Deployment groups

Pipelines

Recent All Runs

All pipelines

Name
Build
<input type="radio"/> bdhp-ga-ec-testpoc1-build No runs yet
<input type="radio"/> bdhp-ga-ec-testpoc2-build No runs yet
Deploy
<input type="radio"/> bdhp-ga-ec-testpoc1-release No runs yet
<input type="radio"/> bdhp-ga-ec-testpoc2-release No runs yet



ACR Setup [🔗](#)

The EC DevOps team will provide a standardized location for your guest application deployment. No one should be applying images directly to the ACR (Azure Container Registry) unless provided specific access, but instead utilize the pipelines to create them. This ensures appropriate image signing and version control, while also ensuring that unintentional deployment issues are prevented. Manual connection keys for pulling images can be provided, but we won't be able to provide one that has the right to directly write to the ACR except for Type 3 apps. The pipelines themselves create a connection to the ACR when an application has been scanned by Sonar Cloud and signed by a trusted certificate. That connection is deleted after every build.

ACR Access For Apps Using Different Toolset (Type 3) [🔗](#)

For the teams that are not using GitHub or Azure DevOps, EC DevOps team will provide Reader access to the ACR repos via the respective team GG role groups and a token that has access to push images/charts to the ACR. The scope map is restricted to only the list of repositories created specifically for those teams, which is to be provided beforehand. This token can be utilized to authenticate and push images/charts using the password that will be provided. This token can only be used to push the images/charts and the signing is to be done via a pipeline or other necessary steps. These teams are expected to follow the naming conventions and versioning as stated below.

ACR Versioning [🔗](#)

Master Chart Versioning [🔗](#)

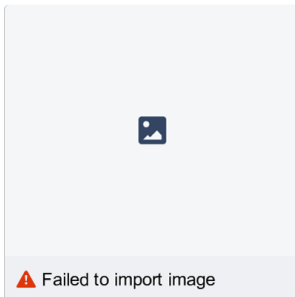
Applications are tagged in the ACR in a way that identifies three things: where it is installed, when it was created, and what its purpose. A typical Kubernetes build will include 2 images. One with the Helm chart and one with the actual deployable image. whenever GA pipeline trigger master helm chart is getting executed for the 1st time and it generate below build images which is managed by BDHP.

- charts/(appname):(Version)
- images/(appname):(Version)

In the above locations, “appname” represents the application, and “version” is the platform version.

. This means that there will be *multiple* image TAGS on the same image, allowing it to be deployed into multiple environments. An example would be **charts/bdhp-testapp:4.1.0**

Once an image or chart is in the ACR there are rules about how long each is kept. Any entry that doesn't have a “-(environment)” is the base image tag, and for base images, the last 5 images are kept within a month, until they are not deployed to any environment. The base image will be there until all the “-(environment)” tags are gone. Each environment will have the last 5 images available and any that same month. After that, the images are purged. Productions images are kept forever.



If build will be not empty, then master helm chart version will generate with “-beta”.

Guest Application Chart Versioning [🔗](#)

tags:1.1.0 -major version.minorversion.bug/fix

Major version: This value comes from helm charts.

Minor version: Whenever PR has been created for dev branch and it will increase by 1.

Bug/Fix - Whenever bug or fix will happen, bug/fix value will increase by 1.

Note- During the commit if it is bug/fix mention the pattern bug/fix in the commit message.

This is standard version set by platform team across the organization.

Application Deployment Type [🔗](#)

There are 2 guest application deployment types that can be onboarded: Multi-Tenant (MT) and Single-Tenant (ST).

A Multi-Tenant (MT) guest application installs 1 application per environment. This means *all* customers interface the same application.

A Single-Tenant guest application (ST) has a specific instance per customer. This means each customer has its own specific instance of the application installed.

Multi-Tenant guest applications are treated differently, as they have distinct deployments and expectations. For each MT guest application, the EC DevOps team will create a single Build pipeline, along with one Deployment pipeline setup per environment. The AZDO will be configured to automatically deploy every commit from the DEV branch to the DEV Kubernetes environment. AZDO will also commit RELEASE* branches to the stage environment. From that location they can be promoted along the DEV or RELEASE track. More details later in the onboarding application.

Initially, for each Single-Tenant (ST) guest application, only the build pipeline will be created and not the deployment pipeline. Because the deployment process is different for each build, as each deployment is for a different customer, there will be no automation on the deployment. Instead, there will be a project created to deploy any customer installation on a per customer basis. A common GitHub repo will contain each of these profiles, and a release pipeline file will be created for each customer utilizing a deployment template. That template will allow us to configure which ST guest applications need to be installed for that customer.

Generic Helper Service [🔗](#)

(Comments here)

Intra-cluster certificate

Service principals

KeyCloak roles

Unique App Specific Helper Service [🔗](#)

(Comments here)

Anything not covered by the generic helper service but required for installation.

Databases [🔗](#)

The questionnaire will help identify which types of databases will need to be created. The platform has multiple options available, but **the preference is to use PostgreSQL whenever possible**. We will provide a Kubernetes secret (type 1,3,4) or a vault certificate (type 2) that contains credentials and connection string to the database. One certificate will be provided for each environment for each application. If a guest application needs to connect to databases other than the one database created, then connection strings in the appropriate vault will be provided.

PostgreSQL - [Azure Database for PostgreSQL - Flexible Server | Microsoft Learn](#)

MS SQL - [Serverless compute tier - Azure SQL Database | Microsoft Learn](#)

Redis - [Azure Cache for Redis | Microsoft Azure](#)

MySQL - [Overview - Azure Database for MySQL - Flexible Server | Microsoft Learn](#)

Cosmos - [Azure Cosmos DB - NoSQL and Relational Database | Microsoft Azure](#)

Connecting to Postgres SQL server and Databases

Steps


1. Obtaining Connection Details

• Azure Portal:

- Navigate to your Azure PostgreSQL Flexible Server instance in the Azure portal.
- Under the "Settings" blade, locate the "Connection strings" section.

- Copy the connection string for your preferred client (e.g., psql, JDBC, etc.). The connection string will typically look like this:
- `psql "host=<server_name>.postgres.database.azure.com port=5432 dbname=<database_name> user=<username> password=<your_password> sslmode=require"`

Installing and Configuring the Client Tool

Download and Install: Download and install pgAdmin from  [Download](#)

A. Using psql

Obtain Connection String:

- Navigate to your Azure Postgres server in the Azure portal.
- Go to the "Connection strings" section under the "Settings" blade.
- Copy the connection string under the "psql" heading. It will resemble:
- `psql "host=<server_name>.postgres.database.azure.com port=5432 dbname=<database_name> user=<username> password=<your_password> sslmode=require"`
- **Creating a New Server Connection:** Follow these steps:
 - a. Open pgAdmin.
 - b. Right-click "Servers" and select "Create" -> "Server.."
 - c. **General Tab:**
 - **Name:** Provide a descriptive name for your server.
 - d. **Connection Tab:**
 - **Host Name/Address:** Enter the hostname from your connection string.
 - **Port:** Use 5432 (default).
 - **Maintenance Database:** Enter the database name.
 - **Username:** Enter the username.
 - **Password:** Enter your secure password.
 - **SSL Mode:** Select "Require".
 - e. Click "Save".

Connecting to Cosmos Instances and Databases

Common Azure Cosmos DB APIs

Core (SQL) API: The original SQL-like query language for Cosmos DB.

API for MongoDB: Provides compatibility with MongoDB client tools and protocols.

Cassandra API: Offers compatibility with Cassandra tools using CQL.

Gremlin API: Enables graph data representation and traversal.

Table API: Provides a key-value store model.

General Steps

Here's a breakdown of the general steps involved in connecting to Azure Cosmos DB, followed by more specific instructions depending on your chosen API.

Obtain Connection Information:

Navigate to your Azure Cosmos DB account in the Azure portal.

Keys: Locate the "Keys" section under "Settings".

Primary or Secondary Connection String: You'll find connection strings (URI) and keys. Copy the appropriate connection string based on your chosen API and tools.

Choose a Client Tool/SDK:

API-Specific SDKs: Azure Cosmos DB offers SDKs for various programming languages (e.g., Python, Java, .NET). These SDKs provide convenient ways to interact with your Cosmos DB data.

Data Exploration Tools: The Azure portal has a built-in "Data Explorer" for interacting with certain APIs.

API-Compatible Tools: For APIs like MongoDB or Cassandra, you can often use standard client tools designed for those databases.

Specialty Azure Resources [🔗](#)

Any specialty Azure resources will need to be called out in advance through the questionnaire. Any resources needed can be created, but those will need to be added into the base deployment code. Possible specialty Azure resources include, but are not limited to, the following:

- Storage Accounts
- Azure container
- Unique App Configuration
- Azure Principals
- Unique Networking Features

Deployment Strategy/Timing [🔗](#)

Each environment currently has a specific cadence to deployment. For MT guest applications, deployment to the DEV environment is done with every commit. Deployment to the TEST and/or HFX environments may be pushed by QEs or developers. Deployment to the STAGE environment may only be executed by QEs. Deployment to the SBX and/or PROD environments must be coordinated with and executed by DevOps. This cadence could be changed for certain MT guest applications, but the guest application team would need to ensure that there are no dependencies with other features of the platform.

Single tenant app deployment procedures still need to be determined.

Kubernetes Information [🔗](#)

Namespaces [🔗](#)

Every multi-tenant guest application is installed into its own namespace which denotes both the environment and the application in the form BDHP-(environment)-GA-(team).

Example: BDHP-DEV-GA-CSA

All single-tenant guest applications will be installed into a shared namespace for the customer in the form BDHP-(Environment)-ST-(customer)

Example: BDHP-DEV-ST-Hospital1

Secrets [🔗](#)

Any secrets that are created for any given application will be limited to that namespace.

Any secrets you need from other namespaces will need to be called out in the questionnaire so the EC DevOps team can reflect them into that namespace.

Any provided secrets from databases, certificates, or service principal information will be available as secrets in the namespace of the app.

Endpoints (http, and data) [🔗](#)

Because of the nature of the shared environment, every application will **not** be able to have its own public endpoint. We have one endpoint which is provided by NGINX which can process and route HTTP traffic to the services. We also have an HA-PROXY endpoint setup that can route TCP data.

HTTP specific traffic only [🔗](#)

The URL can be processed in such a way that we can use the same host to route to different applications.

`https://Kube-dev-eus.connex.baxter.com/application1` can route to application1

`https://Kube-dev-eus.connex.baxter.com/application2` can route to application2

`https://Kube-dev-eus.connex.baxter.com/application2/happy` can route to application happy

HTTP and TCP traffic with Server Name Indication (SNI) [🔗](#)

If the hostname needs to determine which application to call, the devices must support Server Name Indication (SNI) to inform the proxy what the hostname is. This is the only way to share the port to different applications/customers.

`https://application1.connex.baxter.com` can route to application1

`https://application2.connex.baxter.com` can route to application2

`TCP://datatrafic1.connex.baxter.com` can route data traffic to datatrafic1 app/endpoint

HTTP and TCP traffic without SNI [🔗](#)

If the devices DO NOT support SNI, then there is no way to decide which host is being called to that address from the endpoint. That means the port becomes the only indicator as to what is being routed. The consequence is that each endpoint WILL REQUIRE its own unique port for the application, and that port becomes globally locked to that specific application, meaning that ANY HOSTNAME can no longer use that port also. It is advisable to not use this method and ensure that devices support SNI. SNI is an extension to the TLS protocol which has been around for the last several decades.

[Server Name Indication - Wikipedia](#)

Networking [🔗](#)

Firewall/NSG [🔗](#)

Currently the environment has only a few public endpoints. Each of these endpoints are restricted to only receiving data from Cloudflare. Any direct connection to Azure will fail. This means that to get to the applications, we must either create an HTTPS endpoint off of the default http endpoint or we must route it via the data port object into HA-PROXY.

There will also be endpoints available for services/devices that will be used over the Virtual Private Network (VPN). These will have the same endpoint mechanism, only locked to access via the VPN connect.

Cloudflare [🔗](#)

Cloudflare is a provider we use to provide security and help with things like DDOS and hacking attempts. For more information you can reference: [Spectrum](#) | [DDoS Protection for Apps](#) | [Cloudflare](#)

Public IP (Internet Protocol) [🔗](#)

Public IP's are limited in Azure. To address this, we use shared proxy endpoints which can service all our HTTP and TCP traffic over shared public IP's. We do this by creating a CNAME record, which is an alias that points to our endpoints for each environment. From the data provided by SNI we can decide which DNS address was called and then use that information for routing purposes.

Routing [🔗](#)

Currently the routing is all done inside of 2 sets of networks. Internally to Kubernetes all the pods exist in the IP range 10.0.0.0/16. This means we can have 65534 pods theoretically in one cluster. The nodes themselves sit inside of a network of 192.168.120.0/17 which means there could be as many as 32766 nodes running Kubernetes. Public ingress is ONLY ACCESSABLE from the internet through CloudFlare. The private ingresses will only be accessible over the VPN though NAT addresses. (TBD addresses)

VPN [🔗](#)

There will be a double NAT VPN setup to allow communication between the customer network and the BDHP solution. This means all customers will use the same address to get to our endpoints (TBD). However, to get to their EMR machines, a unique mapping will be done so that every device is unique even with overlapping customer networks.

More Details coming

Additional Details [🔗](#)

Naming standards [🔗](#)

To ensure that things are either locally unique or globally unique, please be descriptive in naming resources.

Backup/Recovery [🔗](#)

Kubernetes pods are ephemeral by nature. Please ensure that any data is stored in a permanent location if the data needs to persist. There is a snapshot taken once a day. Additionally, transaction snapshots are done depending on the workload and when the archive file is ready to be processed. The RPO (Recovery Point Objective) is set to 15 minutes. For more details please reference: [Backup and restore in Azure Database for PostgreSQL - Flexible Server | Microsoft Learn](#)

The recovery process does not restore a single database. Instead, when a restore is initiated, the whole server is restored to a new name. This process takes a few minutes. The requested database is then exported and imported back into the old server. The complete process can usually be done in less than an hour.

Build Options [🔗](#)

Code Coverage [🔗](#)

Every build will have code coverage details provided as a part of the build pipeline. This report has live clickable results that let you see where issues in code coverage may be.

Azure DevOps hrc-zenith / IAC - BDHP / Pipelines / bdhp-catalog-build / 4.1.0-build.20230605193614

IAC - BDHP +

- Overview
- Pipelines
- Environments
- Releases
- Library
- Task groups
- Deployment groups

#4.1.0-build.20230605193614 • Merge pull request #292 from Hillrom-Enterprise/MDAP-8256

bdhp-catalog-build 64582

This run is being retained as one of 3 recent runs by dev (Branch).

View retention leases

Summary Tests Code Coverage Extensions Associated pipelines

Summary

Sponsor Star

Information

Parser: Cobertura

Assemblies: 4

Classes: 353

Files: 321

Coverage date: 06/05/2023 - 19:43:12

Line coverage

45%

Covered lines: 6382
Uncovered lines: 7635
Coverable lines: 14017
Total lines: 36608
Line coverage: 45.5%

Branch coverage

30%

Covered branches: 1150
Total branches: 3792
Branch coverage: 30.3%

Method coverage

Feature is only available for sponsors

[Upgrade to PRO version](#)

Risk Hotspots

Assembly	Class	Method	Cyclomatic complexity
CatalogService.Core	CatalogService.Core.Adapters.GatewayAdapter	ToEnv(...)	130
CatalogService.Core	CatalogService.Core.Models.RequestResponse.Integration.Mirth.BaseLocation	GetLocationTypeCode(...)	72
CatalogService.Core	CatalogService.Core.Adapters.GatewayAdapter	ToDeployment(...)	66
CatalogService.Core	CatalogService.Core.CatalogManager	CreateKeycloakRealm()	64
CatalogService.Core	CatalogService.Core.Models.RequestResponse.Integration.Mirth.GetLocationListResponse	FromCSVFile(...)	64
CatalogService	CatalogService.Controllers.Integration.MirthController	CreateOrUpdateDeployment()	56
CatalogService.Core	CatalogService.Core.Models.RequestResponse.Integration.Mirth.Physical	FromFile(...)	48

SonarCloud [↗](#)

SonarCloud automatically runs after every build. A link to the report is added to the build details for every project. SonarCloud reports are tied to your GitHub credentials.

Azure DevOps hrc-zenith / IAC - BDHP / Pipelines / bdhp-catalog-build / 4.1.0-build.20230605193614

IAC - BDHP +

- Overview
- Pipelines
- Environments
- Releases
- Library
- Task groups
- Deployment groups

#4.1.0-build.20230605193614 • Merge pull request #292 from Hillrom-Enterprise/MDAP-8256

bdhp-catalog-build 64582

This run is being retained as one of 3 recent runs by dev (Branch).

Summary Tests Code Coverage Extensions Associated pipelines

PipelineSummary

Run ID = 64582

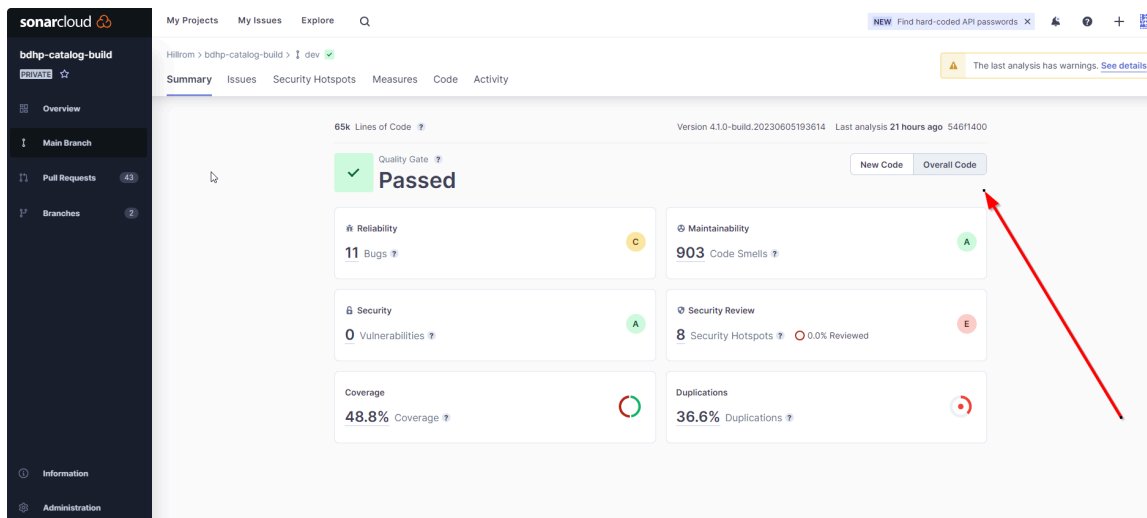
SonarCloud Analysis Report

bdhp-catalog-build Quality Gate **Passed**

Detailed SonarCloud report > [↗](#)

By default, it only shows you the updates from the last code update.

Make sure you click Overall Code to get a full report.



Please work to eliminate ALL Bugs, Vulnerabilities, and Security Hotspots.

In addition, please work to eliminate Code Smells of Blocker or Critical Severity.

The Coverage section refers to how much of the committed code is covered by at least one unit test.

Code should be written to be unit testable.

Unit testing failure routes is far more important than unit testing success routes, as one usually doesn't commit code without having tested success cases while executing the code.

[SonarCloud details](#)

Docker [🔗](#)

This is a Build parameter that needs to be set to 'true' if the pipeline needs to run all the docker steps. By default, it is set to false which will skip all the docker steps in the pipeline. Following are the steps that are currently implemented:

1. Build and Push Docker Image with required tags
2. Verify the docker image tags in the ACR
3. Sign docker images using notation

```
bdhp-vitals-device-service-build

dev
Hillrom-Enterprise/bdhp-vitals-device-service / azure-pipelines-build.yml

1 # https://aka.ms/yaml
2 # File: azure-pipelines-build.yml
3
4 name: $(Date:yyyyMMdd)$(Rev:.r)
5
6 trigger:
7   branches:
8     include:
9       - dev
10    exclude:
11      - release/*
12   paths:
13     exclude:
14       - azure-pipelines-release.yml
15
16 resources:
17   repositories:
18     - repository: templates
19       type: github
20       name: 'Hillrom-Enterprise/bdhp-platform'
21       ref: 'refs/heads/dev'
22       endpoint: 'Hillrom-Enterprise-IAC - BDHP'
23
24 pool:
25   vmImage: ubuntu-latest
26
27 extends:
28   validate
29   template: 'workflows/ci-cd/build/stages/build_stage.yaml@templates' # Template reference
30   parameters:
31     dotnet: 'true'
32     docker: 'true'
33     helm: 'true'
```

Jobs in run #4.0.0-build.20230609150751		
✓	Update Pipeline Run Number - ...	1s
✓	Add Tag and Summary to Pipeli...	1s
✓	Generate Build Props File - Pow...	1s
✓	Use .NET 6.0	7s
✓	dotnet restore	27s
✓	Build in docker and export arti...	48s
✓	Build an image with argument	24s
✓	Push an image to container re...	10s
✓	Verify Docker Image push to a...	14s
✓	Bash Script - Download, Extract...	5s
✓	Docker sign Image using Notar...	29s
✓	Install Helm latest	2s
✓	Helm ACR Registry Login - Az...	19s
✓	Push helm charts to azure con...	9s
✓	Copy Helm Override Values Fi...	< 1s
✓	Get Latest Provisioner Image De...	8s
✓	Helm ACR Registry Token Delet...	6s
✓	Azure Key Vault: dhp-devops...	< 1s
✓	SonarCloud project prep - Pow...	2s
✓	Prepare analysis on SonarCloud	5s
✓	dotnet build	1m 12s

Please base Docker images on Alpine images, whenever possible, as Alpine images are designed to be lightweight, fast, and secure.

Soon, security scanning of Docker images will be added to pipelines, and past scans of Alpine-based Docker images have fared far better in these scans than non-Alpine-based images.

Helm [🔗](#)

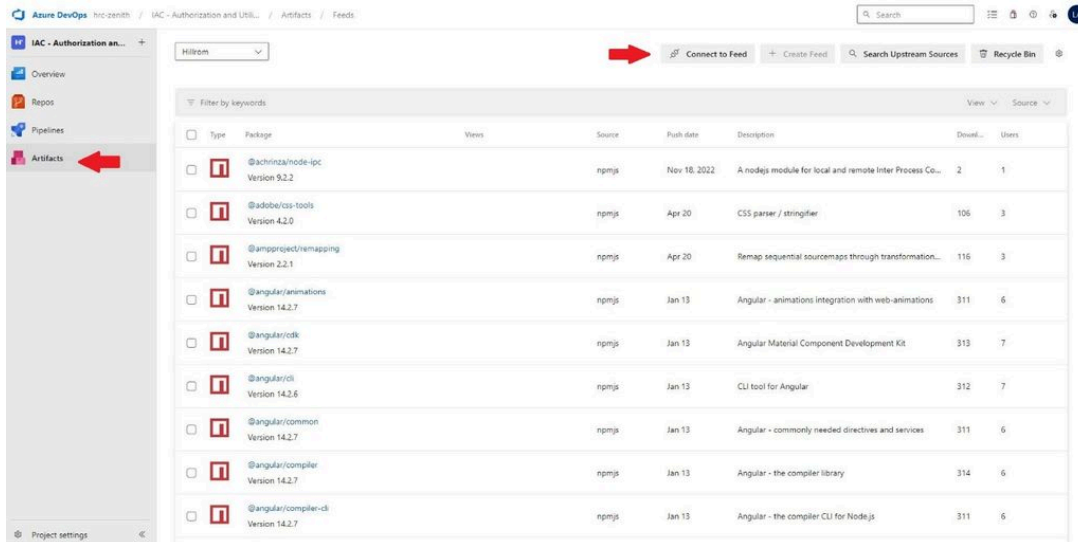
This is a Build parameter that needs to be set to 'true' if the pipeline needs to run all the Helm steps. By default, it is set to false, which will skip all the Helm steps in the pipeline. The following are the steps that are currently implemented:

1. Package helm charts and push to ACR
2. Verify the Helm Package in the ACR

3. Copy environment specific helm override values files for deployment

Specific Application Type Details [↗](#)

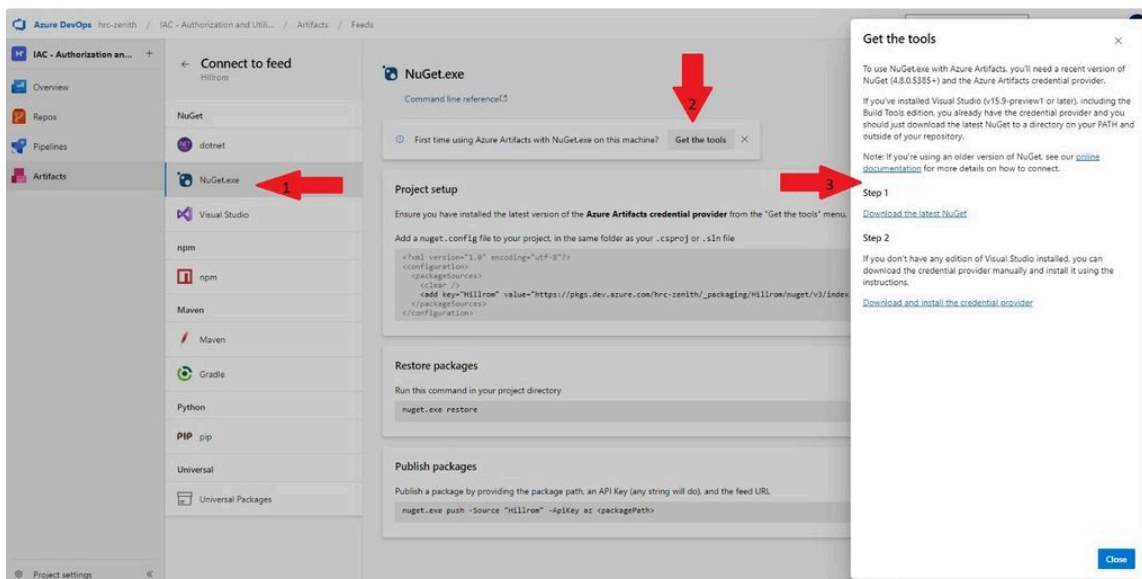
To download any artifacts/packages, you would first need to connect to a feed on the Azure DevOps dashboard. To do so, start by opening a project on the DevOps dashboard, and then click Artifacts on the left-hand menu. Next, click on the “Connect to Feed” button on the top bar as seen in the screenshot below:



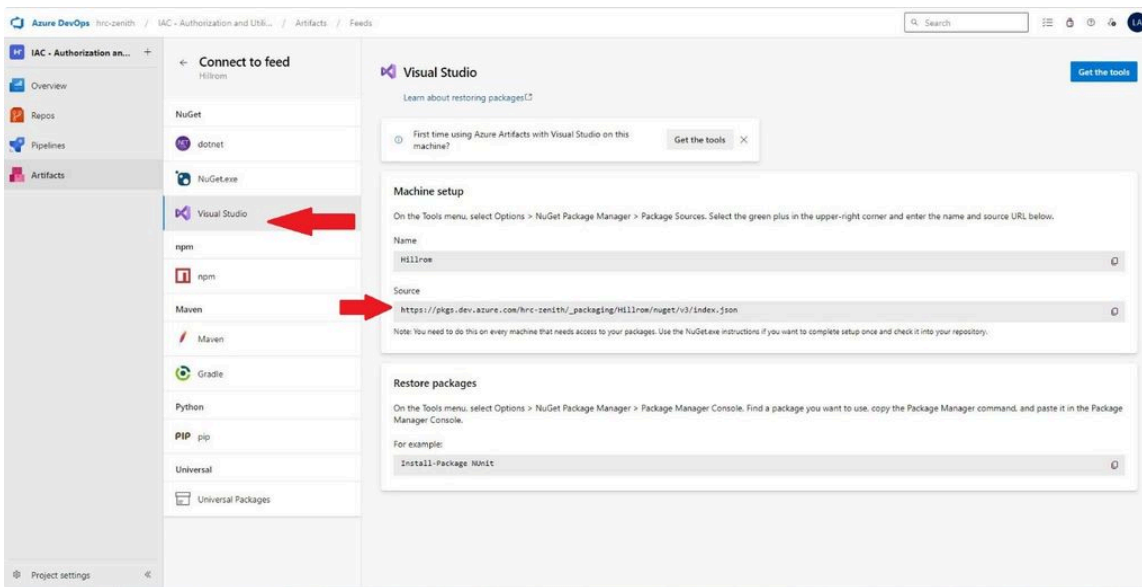
From here you can follow the steps for each type below:

NuGet [↗](#)

1. Select NuGet from the left-hand menu (within the “Connect to feed” page), and, if this is the first-time using Azure Artifacts with Nuget.exe, click on the “Get the tools” button and follow the instructions to do the following:
 - a. Install the latest NuGet version.
 - b. Install Azure Artifacts Credential Provider.



1. Follow the instructions in the Project setup on the dashboard to add a nuget.config file to your project, in the same folder as your .csproj or .sln file.
2. Once the configuration setup is completed, you can Select Visual Studio from the left navigation panel and copy the feed's Source URL into the NuGet package manager to download the artifacts.

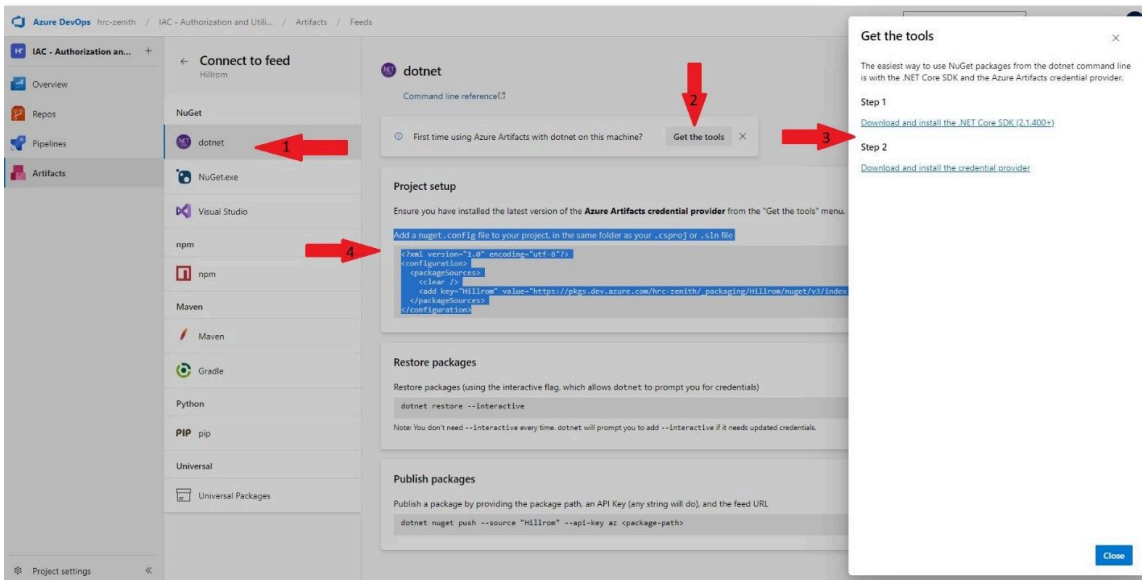


Reference: [Get started with NuGet packages and Azure Artifacts - Azure Artifacts | Microsoft Learn](#)

Dotnet [🔗](#)

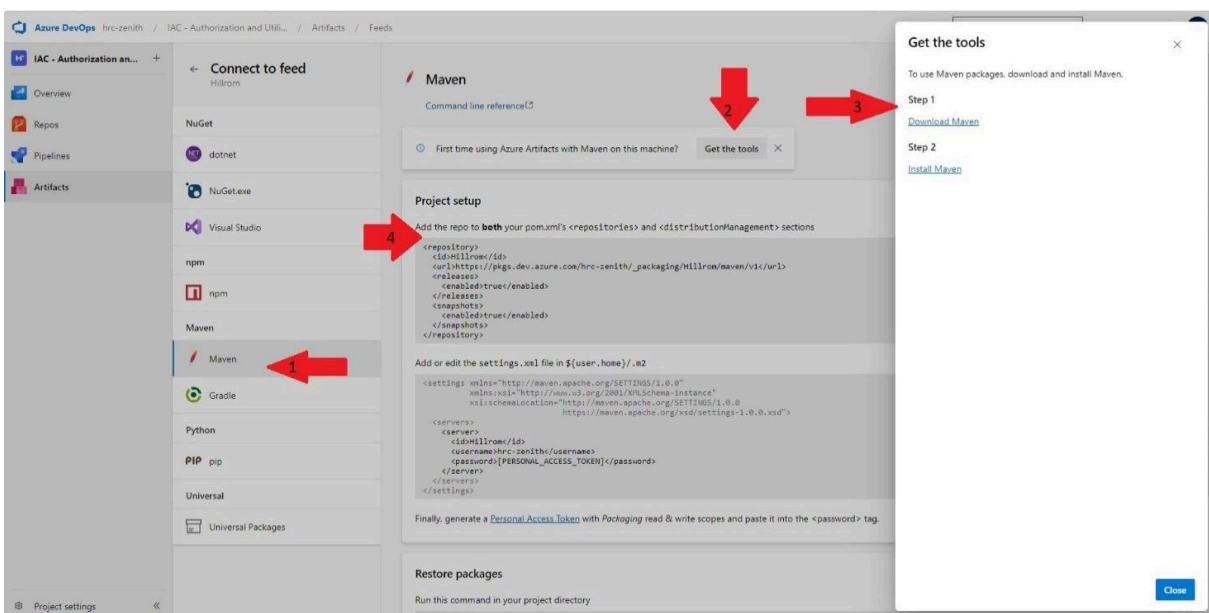
Similar to the above if you would like to create a feed for the dotnet packages:

1. Select Dotnet from the left-hand menu (within the "Connect to feed" page), and, if this is the first-time using Azure Artifacts with Nuget.exe, click on the "Get the tools" button and follow the instructions to do the following:
 - a. Download and install the .NET Core SDK (2.1.400+)
 - b. Download and install the credential provider
2. Follow the instructions in the Project setup to add a nuget.config file to your project, in the same folder as your .csproj or .sln file



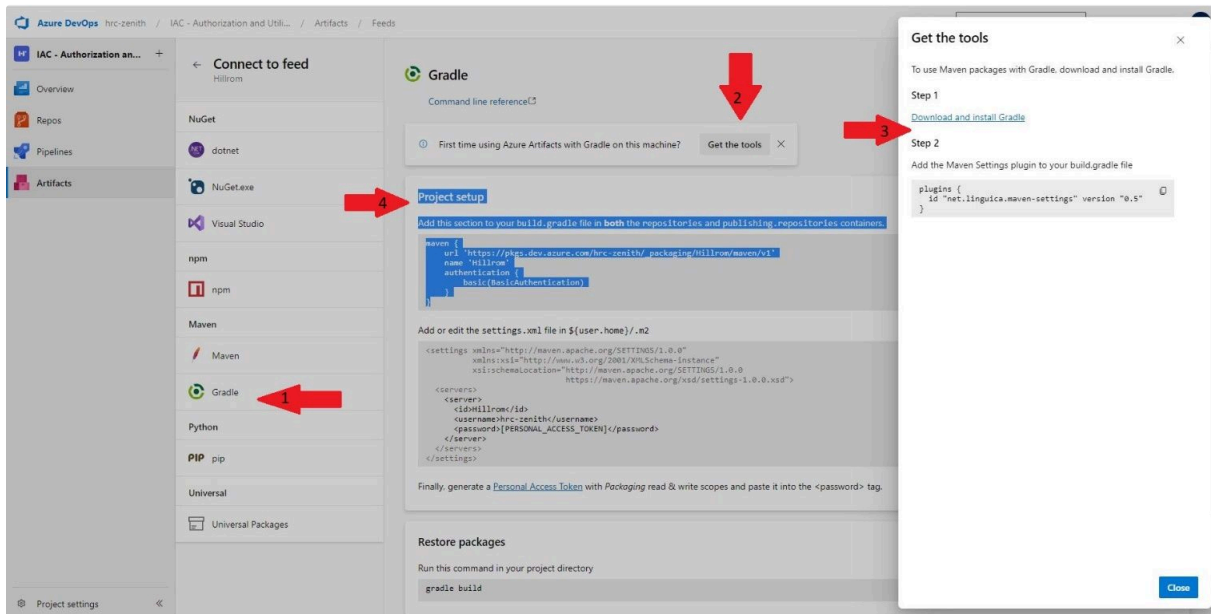
Maven

1. Select Maven from the left-hand menu (within the "Connect to feed" page), and, if this is the first-time using Azure Artifacts with Nuget.exe, click on the "Get the tools" button and follow the instructions to do the following:
 - a. Download Maven
 - b. Install Maven
2. Follow the instructions to add the repo to both your pom.xml's <repositories> and <distributionManagement> sections
3. Next, Add or edit the settings.xml file in \${user.home}/.m2
4. Finally, generate a Personal Access Token with Packaging read & write scopes and paste it into the <password> tag.



Gradle [🔗](#)

1. Select Gradle from the left-hand menu (within the "Connect to feed" page), and, if this is the first-time using Azure Artifacts with NuGet.exe, click on the "Get the tools" button and follow the instructions to do the following:
 - a. Download and install Gradle
 - b. Add the Maven Settings plugin to your build.gradle file
2. Follow the instructions in the Project setup to configure the build.gradle file and edit the settings.xml file in \${user.home}/.m2
3. Finally, generate a Personal Access Token with Packaging read & write scopes and paste it into the <password> tag.



Vault Integration Guide [🔗](#)

Downloads [🔗](#)

[CLI] Download the client [any os] at [Install | Vault | HashiCorp Developer](#) and install it in the path.

[API Integration] [HTTP API | Vault | HashiCorp Developer](#)

WARNING: THE ORDER OF THE COMMANDS IS FINICKY. BE AWARE.

Authentication [cli] [🔗](#)

A cert will be provided to authenticate your application and any necessary secrets. There will be a public (tls.pem) and private portion (tls.key) of the key. This example assumes they are in the current directory. The connection address changes depending on prod (p) or non-prod (np).

EXAMPLE:

```
vault login -method=cert -client-cert="tls.pem" -client-key="tls.key" -address=https://vault-np-eus.connex.baxter.com:8200
```

```
PS C:\CodeBase\Infrastructure\bdhp-terraform-infrastructure-base> vault login -method=cert -client-cert="tls.pem" -client-key="tls.key" -address=https://vault-np-eus.connex.baxter.com:8200
Success! You are now authenticated. The token information displayed below
is already stored in the token helper. You do NOT need to run "vault login"
again. Future Vault requests will automatically use this token.

Key                               Value
-----
token                             hvs.eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1b29udmV4IjoiYmhpdev-ga-cs-dashboar...
token_accessor                     MFLdXit0i0micTjx31K6Rz0I
token_duration                     5m
token_renewable                   true
token_policies                    ["default"]
identity_policies                  []
policies                          ["default"]
token_meta_serial_number           24432108217847550869091002521755625442
token_meta_subject_key_id          n/a
token_meta_authority_key_id        dd:96:cf:08:dd:ae:53:26:e6:78:84:b7:d5:e4:1e:82:ac:65:00:1f
token_meta_cert_name               cacert
token_meta_common_name             bdhp-dev-ga-cs-dashboard
```

List Keys [cli] [↗](#)

To list keys you need to interface with the kv Engine and tell it where to look. The example below is for a fake bdhp-dev-ga-cs-dashboard certificate. **NOTE: ALL VALUES RETRIEVED ARE BASE64 ENCODED**

EXAMPLE:

```
vault kv list -address=https://vault-np-eus.connex.baxter.com:8200 kv/bdhp-dev-ga-cs-dashboard
```

```
PS C:\CodeBase\Infrastructure\bdhp-terraform-infrastructure-base> vault kv list -address=https://vault-np-eus.connex.baxter.com:8200 kv/bdhp-dev-ga-cs-dashboard
Keys
----
platform-shared-config
```

Get Values [cli] [↗](#)

To target all values in the key just use the key name. **NOTE: ALL VALUES RETRIEVED ARE BASE64 ENCODED.**

EXAMPLE:

```
vault kv get -address=https://vault-np-eus.connex.baxter.com:8200 kv/bdhp-dev-ga-cs-dashboard/platform-shared-config
```

```
PS C:\CodeBase\Infrastructure\bdhp-terraform-infrastructure-base> vault kv get -address=https://vault-np-eus.connex.baxter.com:8200 kv/bdhp-dev-ga-cs-dashboard/platform-shared-config
===== Secret Path =====
kv/data/bdhp-dev-ga-cs-dashboard/platform-shared-config

===== Metadata =====
Key                               Value
----
created_time                       2023-06-29T20:18:50.314370506Z
custom_metadata                    <nil>
deletion_time                      n/a
destroyed                         false
version                           1

===== Data =====
Key                               Value
----
admin                             a3ViZS1kZXVtZXVzLmVmbm5leC5lYXh0ZXUvY29t
amqp                               YW1xcC1kZXVtZXVzLmVmbm5leC5lYXh0ZXUvY29t
amqp-port                         NTY3MQ==
device                            ZGF0eXBvcnQtZGV2LW1cy5jb25uZG9uYmF4dG9yLmVmbm5leC5lYXh0ZXUvY29t
keycloak                           aHR0cH9yY29udmV4LmVmbm5leC5lYXh0ZXUvY29t
main                              a3ViZS1kZXVtZXVzLmVmbm5leC5lYXh0ZXUvY29t
mirth                             Z3ctZGV2LW1cy5jb25uZG9uYmF4dG9yLmVmbm5leC5lYXh0ZXUvY29t
mqtt                              bXFsOGE0LmVmbm5leC5lYXh0ZXUvY29t
mqtt-port                         ODg0MA==
vault                             aHR0cH9yY29udmV4LmVmbm5leC5lYXh0ZXUvY29t
```

To target the value of a specific "K=V" pair simply add the field parameter. **NOTE: ALL VALUES ARE BASE64 ENCODED.**

EXAMPLE:

```
vault kv get -address=https://vault-np-eus.connex.baxter.com:8200 -field amqp kv/bdhp-dev-ga-cs-dashboard/platform-shared-config
```

```
PS C:\CodeBase\Infrastructure\bdhp-terraform-infrastructure-base> vault kv get -address=https://vault-np-eus.connex.baxter.com:8200 -field amqp kv/bdhp-dev-ga-cs-dashboard/platform-shared-config
YW1xcC1kZXVtZXVzLmVmbm5leC5lYXh0ZXUvY29t
```

Support Information [↗](#)

The Enterprise Connectivity DevOps team is available for assistance with the platform and to provide infrastructure level assistance and testing.

SLA (Service Level Agreement)

(info)

JIRA

(Info)

TICKETS

(Info)