



OpenFN Retail Application Development

This document will walk you through a demonstration of the development and DevOps process that is used to deliver the dev instance of the OpenFN retail web application to the FS Cloud Reference architecture. The intent of this demo is to introduce OpenShift (Tekton) pipelines, and the pipeline templates that are available at <https://cloudnativetoolkit.dev/>.

Goals for the Demo:

- Familiarize the audience with the OpenShift developer experience and OpenShift pipelines
- Show a functional real-world Pipeline
- Demonstrate the process used to deliver the OpenFN retail banking application on to the FS Cloud Demo instance

Prerequisites:

- If you have not already done so, request access to the FS Cloud demo environment at: <https://techzone.ibm.com/collection/ibm-cloud-for-financial-services>
- Download and install the OpenVPN client
 - Windows <https://openvpn.net/community-downloads/>
 - MacOS <https://openvpn.net/client-connect-vpn-for-mac-os/>
 - Linux <https://openvpn.net/download-open-vpn/>
- Download the techzone.ovpn VPN certificate and add it to the OpenVPN client
 - Link <https://techzone-iam-agent.eqtyaj6hk2k.eu-de.codeengine.appdomain.cloud/vpn/download>



IBM Cloud for Financial Services – Tech Zone Demo Environment Hybrid Cloud Ecosystem – Ecosystem Labs

Demo Steps:

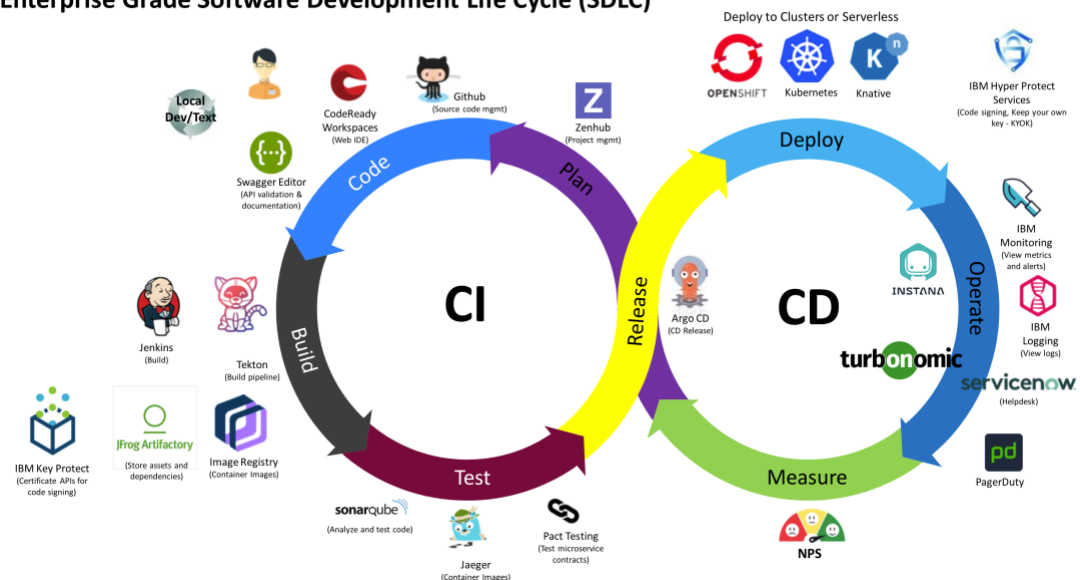
1. Set the stage... the tools and workflow that we're about to show are part of an enterprise-grade software development life cycle. This is the iterative process that enterprises use to build and deliver software reliably and consistently.

It covers everything in the software development lifecycle, including continuous integration (CI) and continuous delivery (CD) phases.

The continuous integration phase covers source control management, automated building, automated testing, container image and artifact management, and inspection/analysis for code quality and vulnerabilities.

The continuous delivery phase covers the cycle of delivering those container images and artifacts into production, monitoring and measuring performance, and using this cycle to feed improvements and new requirements back into the continuous integration cycle to improve the overall solution.

Enterprise Grade Software Development Life Cycle (SDLC)



The tools that we are using for an enterprise software development lifecycle (and are covered in this document) are components of the Cloud Native Toolkit (<https://cloudnativetoolkit.dev/>)

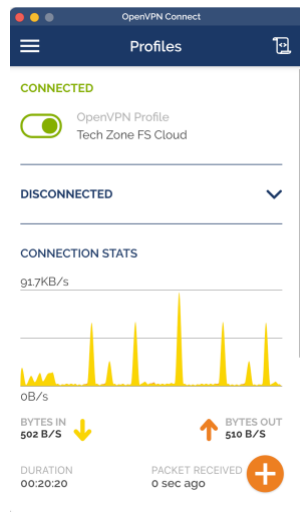
In this document we're going to focus on the continuous integration phase of the



IBM Cloud for Financial Services – Tech Zone Demo Environment
Hybrid Cloud Ecosystem – Ecosystem Labs

enterprise software development lifecycle.

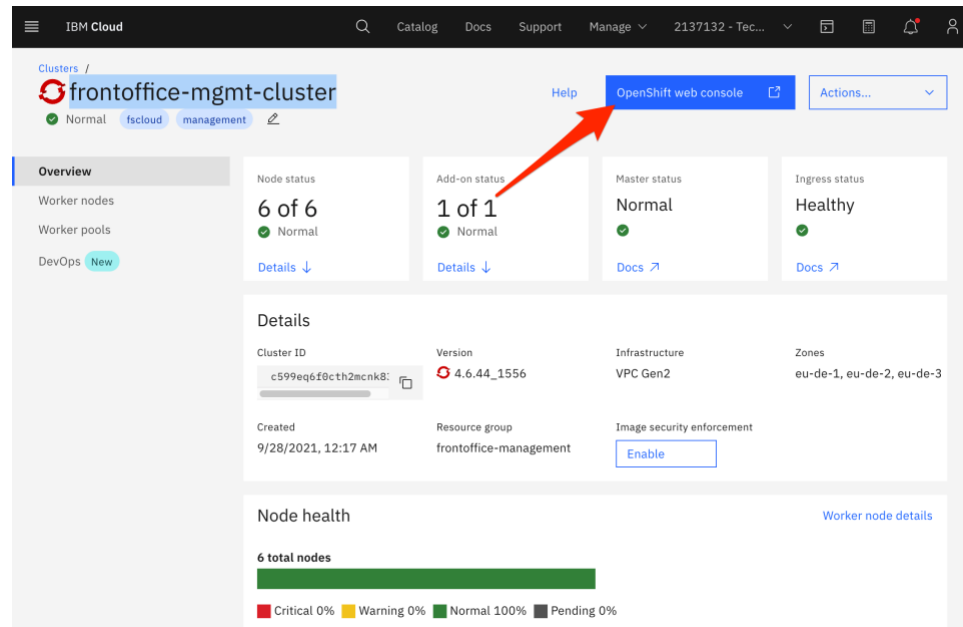
2. First, we need to connect to the management cluster
 - a. Connect the OpenVPN Client with the Tech Zone demo profile



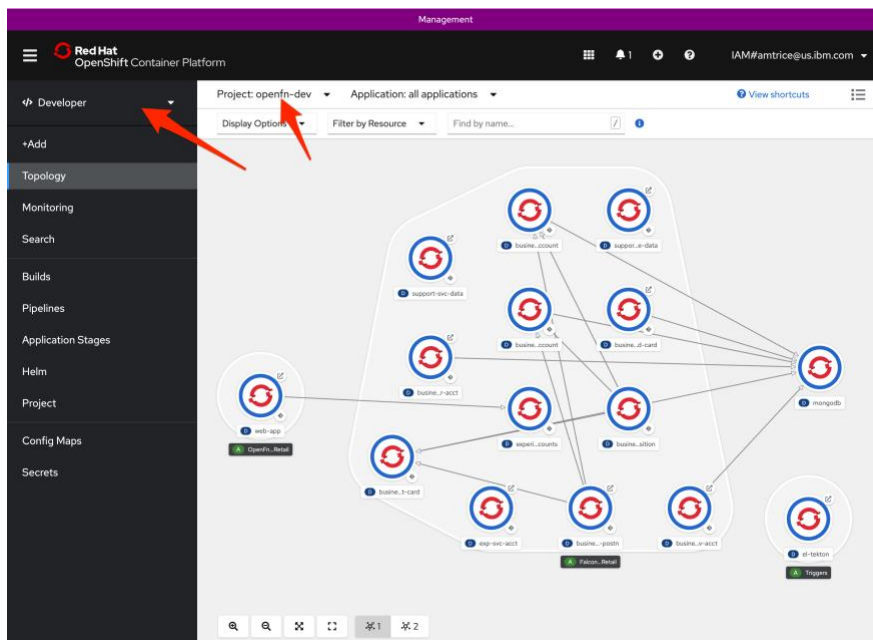
- b. Navigate to <https://cloud.ibm.com/kubernetes/clusters>
 - c. Select the “**frontoffice-mgmt-cluster**” instance to view the cluster details
 - d. Click the “OpenShift web console” button to bring up the OpenShift Dashboard



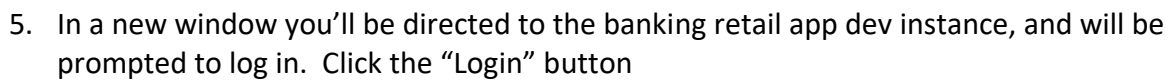
IBM Cloud for Financial Services – Tech Zone Demo Environment Hybrid Cloud Ecosystem – Ecosystem Labs



3. In the OpenShift dashboard, select the “Developer” view and select the project “openfn-dev”.

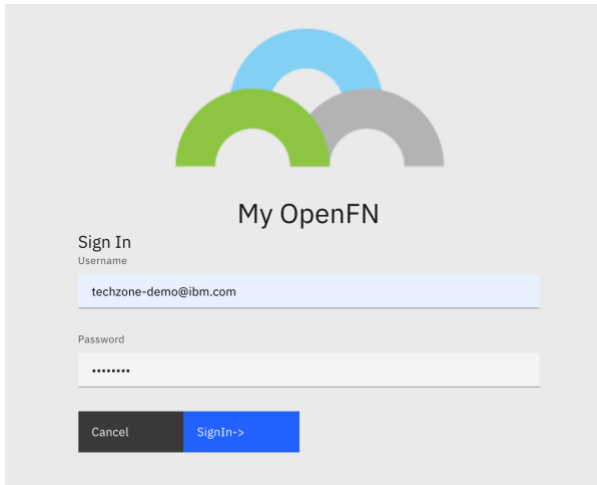


4. Here you can see a topographic view of the components deployed in the OpenFN banking application. This is made up of multiple microservices, and a MongoDB database instance. Click on the “Open URL” button for the “web-app” pod to view the **dev instance** of the OpenFN application.





IBM Cloud for Financial Services – Tech Zone Demo Environment Hybrid Cloud Ecosystem – Ecosystem Labs



My OpenFN

Sign In

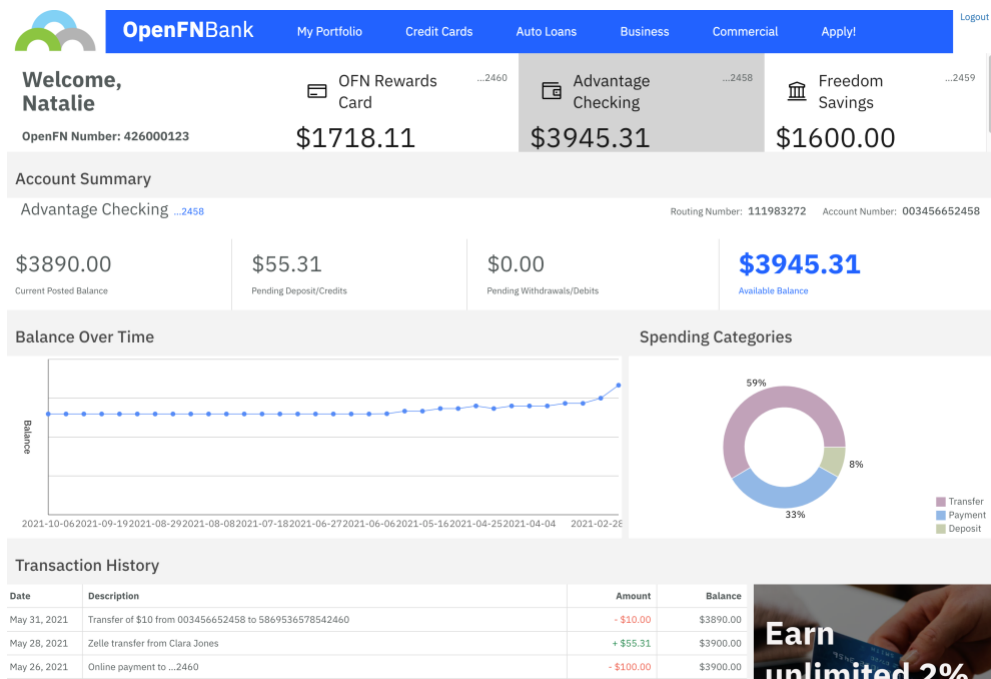
Username

techzone-demo@ibm.com

Password

Cancel Sign In->

- Once logged in, show the retail banking application user interface. This is to simulate a real banking application.

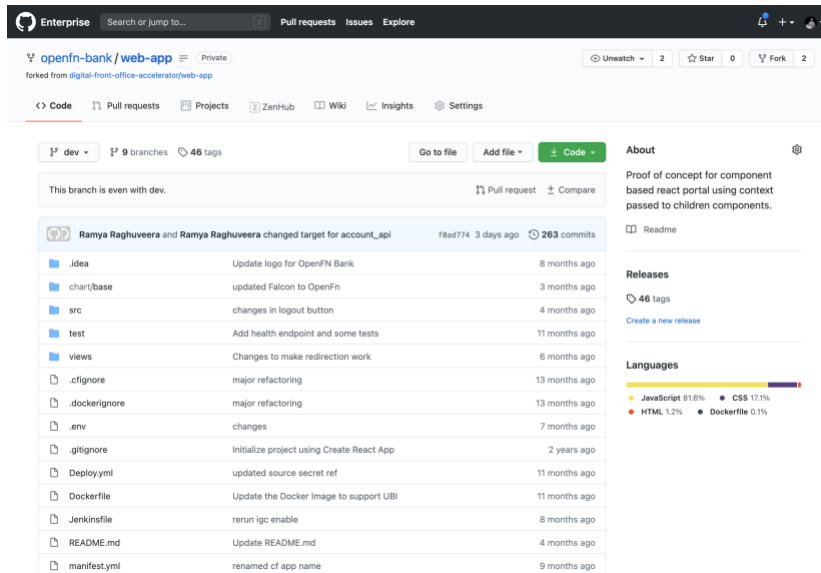


This is exactly the same process that an enterprise software development team would be used to build and deliver a real banking application into production.



IBM Cloud for Financial Services – Tech Zone Demo Environment Hybrid Cloud Ecosystem – Ecosystem Labs

8. In a new tab open up <https://github.ibm.com/openfn-bank/web-app>



This is the repository containing the front-end web application source code. This is where the continuous integration cycle starts. The repository is used to manage application source code. It is used to store source code, manage contributions, and merge conflicts, and track issues.

Commits to this repository will trigger the devops pipeline for the web application. So, a developer will make changes to their copy of the application source code, and once those changes are committed back to the source repository, a new pipeline run will be invoked.

9. Once you have shown the banking application interface, go back to your OpenShift dashboard, and click on the “Pipelines” link in the left menu. Here you can see the DevOps pipelines that are configured for the OpenFN banking application microservices.



IBM Cloud for Financial Services – Tech Zone Demo Environment Hybrid Cloud Ecosystem – Ecosystem Labs

Name	Last Run	Task Status	Last Run Status	Last Run Time
web-app	web-app-lsow4		Succeeded	Oct 6, 10:19 am
business-service-savings-account	business-service-savings-account-ntrjs		Succeeded	Oct 6, 10:16 am
business-service-current-account	business-service-current-account-46uux		Succeeded	Oct 6, 10:16 am
support-service-data	support-service-data-ghla9s		Succeeded	Oct 6, 10:08 am
business-service-credit-card	business-service-credit-card-od327		Succeeded	Oct 6, 10:07 am
experience-service-accounts	experience-service-accounts-5uv3vb		Succeeded	Oct 4, 3:58 am
business-service-customer-position	business-service-customer-position-ptp9a		Succeeded	Oct 4, 3:57 am

10. Click on the “web-app” pipeline to view its details.

Here you can see the tasks and trigger templates that are associated with the pipeline instance.

Pipeline Details

Workflow: setup → test → dockerfile-lint → build → deploy → health → tag-rel

Name: web-app

Namespace: openfn-dev

Labels: No labels

Annotations: 2 Annotations

Created At: Oct 1, 11:2 pm

Owner: No owner

Trigger Templates:

- web-app

Tasks:

- ibm-setup-v2-7-6 (setup)
- ibm-nodejs-test-v2-7-6 (test)
- ibm-dockerfile-lint-v2-7-6 (dockerfile-lint)
- ibm-build-tag-push-v2-7-6 (build)
- ibm-deploy-v2-7-6 (deploy)
- ibm-health-check-v2-7-6 (health)
- ibm-tag-release-v2-7-6 (tag-release)
- ibm-img-release-v2-7-6 (img-release)
- ibm-helm-release-v2-7-6 (helm-release)
- ibm-gitops-v2-7-6 (gitops)

Here you should explain the DevOps pipeline process in more detail. The general lifecycle is:



IBM Cloud for Financial Services – Tech Zone Demo Environment

Hybrid Cloud Ecosystem – Ecosystem Labs

- Developer creates a branch off of the “dev” branch, makes changes, commits them back to the repository, and creates a pull request back into dev.
- When changes are merged/committed into the dev branch, it will kick off a pipeline run.
- The pipeline run consists of 11 tasks executed in sequential order. If there’s an error in any of the tasks, execution will stop at that error.

11. Click on the “Pipeline Runs” tab to see the list of pipeline executions.

Project: openfn-dev ▾

Pipelines > Pipeline Details Tech Preview

web-app Actions ▾

Details Pipeline Runs Parameters Resources

Filter ▾ Name ▾ Search by name... [Z]

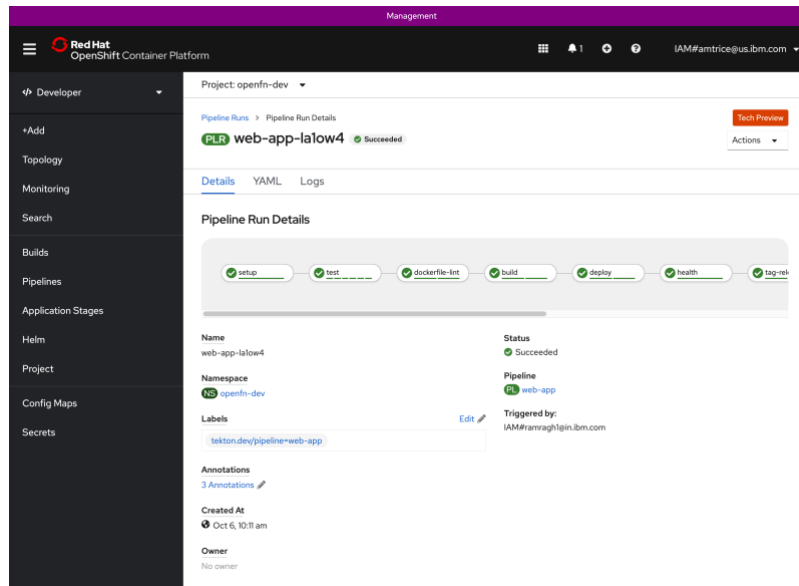
Name	Status	Task Status	Started	Duration
web-app-la1ow4	Succeeded	<div></div>	Oct 6, 10:11 am	about 8 minutes
web-app-rhax0z	Succeeded	<div></div>	Oct 4, 3:52 am	about 9 minutes
web-app-46t35d	Succeeded	<div></div>	Oct 4, 3:30 am	about 9 minutes
web-app-dy7yn	Succeeded	<div></div>	Oct 3, 2:01 pm	about 9 minutes
web-app-gwbjdc	Failed	<div></div>	Oct 3, 1:55 pm	about an hour
web-app-ll3653	Failed	<div></div>	Oct 3, 1:41 pm	about 2 minutes
web-app-rjlnb5	Failed	<div></div>	Oct 3, 12:34 pm	about 2 minutes
web-app-zfa8g2	Failed	<div></div>	Oct 3, 12:28 pm	about 2 minutes
web-app-17c3cd7e8e5	Failed	<div></div>	Oct 1, 1:12 pm	about 2 minutes

Here you can see a list of the most recent invocations of the web-app pipeline, and their execution status. At a glance, you can quickly see how recently the application was built, and whether or not the pipeline was successful.

12. Click on the most recent pipeline run to view the instance details (top row in the list).



IBM Cloud for Financial Services – Tech Zone Demo Environment Hybrid Cloud Ecosystem – Ecosystem Labs

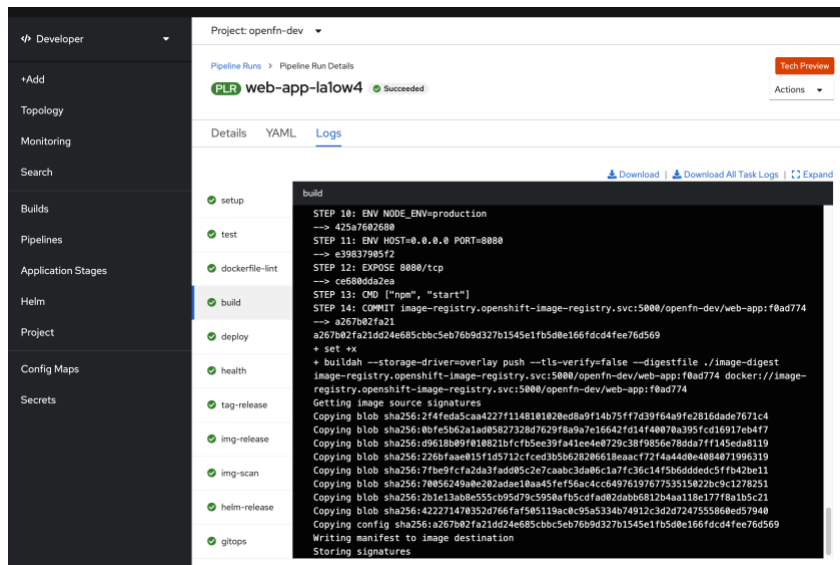


Next examine the steps in more detail. The steps are:

- a. **setup** - clones the repo contents for use in the pipeline
 - b. **test** - executes test scripts for the application, including unit tests, PACT contract testing, and a SonarQube code quality scan
 - c. **dockerfile-lint** - if configured, will lint the Dockerfile for programmatic or stylistic errors
 - d. **build** - compiles & builds the container image and publishes to an internal repository
 - e. **deploy** - deploys the containerized application into the openfn-dev namespace/project within the cluster
 - f. **health** – performs a health check to the deployed application
 - g. **tag-release** – tags the source code repository with the build/release version
 - h. **img release** – signs and versions the application’s container image and copies it to the IBM Container Registry
 - i. **helm-release** – publishes a versioned helm chart (with references to the versioned image) to Artifactory. These are versioned and published to Artifactory to make it easy to reproduce specific application versions, thus making it easy to roll back to a specific version when needed.
 - j. **gitops** – this task updates the web app’s gitops repository with references to the versioned helm chart. This information will be used by the ArgoCD continuous delivery tool to deliver the application to test or production instances.
13. Click on the “build” task to view task details. Scroll through the log to show execution of the docker build command



IBM Cloud for Financial Services – Tech Zone Demo Environment Hybrid Cloud Ecosystem – Ecosystem Labs



14. Click on the **“img-release”** task, and scroll to show details where the container image is being signed when copied to the IBM Container Registry per the [Cloud Native Toolkit Image Security Enforcement guide](#).

The general flow for signing an image is:

- Signing keys are retrieved from the Hyper Protect Crypto Services vault. Hyper Protect Crypto services is based on a FIPS 140-2 hardware security module. This offers the highest level of encryption of any cloud provider, and allows for a keep-your-own-key usage model where IBM never has access to your keys.

You can view the [crypto-hsm-kyok](#) hyper protect instance used by the OpenFN application at <https://cloud.ibm.com/services/hs-crypto/crn%3Av1%3Abluemix%3Apublic%3Ahs-crypto%3Aeu-de%3Aa%2Fea243e63212643dc927f7fe26b6e726c%3A50423a85-9f41-470d-8527-3c3c5f7295ce%3A%3A>, or by accessing it from the IBM Cloud resource list.



IBM Cloud for Financial Services – Tech Zone Demo Environment Hybrid Cloud Ecosystem – Ecosystem Labs

The screenshot shows the IBM Cloud console interface. The top navigation bar includes 'IBM Cloud', a search bar, and links for 'Catalog', 'Docs', 'Support', and 'Manage'. The main content area is titled 'crypto-hsm-kyok' and shows 'Key management service keys' with a count of 78. A table lists the keys with columns: Name, ID, Alias, Key ring ID, Type, State, Origin, and Last updated. Two keys are visible: 'portieris-image-signing-public-key' and 'portieris-image-signing-private-key', both in 'Active' state and 'Imported' origin. The table has pagination controls at the bottom showing '1-2 of 2 items'.

Name	ID	Alias	Key ring ID	Type	State	Origin	Last updated
portieris-image-signing-public-key	4e58...9064		default	Standard key	Active	Imported	2021-10-08 14:52:51
portieris-image-signing-private-key	e363...a0b3		default	Standard key	Active	Imported	2021-10-08 14:52:47

- The keys are used to sign the container image as it is copied into the IBM Container Registry.

You can see this in the “img-release” task for the web-app pipeline:

The screenshot shows a Jenkins pipeline log for the 'img-release' task. The log output includes the following lines: 'Getting private key from keystore for image signing', 'Importing key', 'gpg: directory '/tekton/home/.gnupg' created', 'gpg: keybox '/tekton/home/.gnupg/pubring.kbx' created', 'gpg: /tekton/home/.gnupg/trustdb.gpg: trustdb created', 'gpg: key DF102246542FE08B: public key "Techzone Demo <amtrice@us.ibm.com>" imported', 'gpg: key DF102246542FE08B: secret key imported', 'gpg: Total number processed: 1', 'gpg: imported: 1', 'gpg: secret keys read: 1', 'gpg: secret keys imported: 1', 'skopeo --insecure-policy --sign-by 134E58C247B8F0CB437009450F102246542FE08B copy --src-tls-verify=false --dest-tls-verify=true do', 'Getting image source signatures', 'Copying blob sha256:43dc9ef977174119841185896fc021aa3af745ecd5c892fe14668f21d4b069', 'Copying blob sha256:4632d477598a4176b858a910f5102c05db036d2a4d206a0aad1f62952f1508d6', 'Copying blob sha256:7a6132c48dd4568bdc334179b3a74dc9145d9425a1075a21bcb08fa79dacc6a', 'Copying blob sha256:6eeb9b4a640ff7b7b8bbac12b72740c753337a5d820fd5ebc1d9244a787ca7db', 'Copying blob sha256:741a0a0f93f892b09099937b42242ae98b199278798bb5277ce7906af126957', 'Copying blob sha256:e3179acfe7d83969c1e385916f53ae18b0dadd6877d3bd7631d59521817c595f', 'Copying blob sha256:ed2fddbb30c3ee1ac1429c41508f2a2e2354f1b3739a84387929ef034c0f61', 'Copying blob sha256:b80ee16c866200b7aca5ae763b95a878e756c7bbbd7cc3b19a033bf1372efc61', 'Copying config sha256:7595a3327d0516e9783642c08cb3f2dda2432888cf08cb9c5e10aa8ef846d65', 'Writing manifest to image destination', 'Signing manifest', and 'Storing signatures'. A red arrow points to the line 'Getting private key from keystore for image signing'.

```
img-release
Getting private key from keystore for image signing
Importing key
gpg: directory '/tekton/home/.gnupg' created
gpg: keybox '/tekton/home/.gnupg/pubring.kbx' created
gpg: /tekton/home/.gnupg/trustdb.gpg: trustdb created
gpg: key DF102246542FE08B: public key "Techzone Demo <amtrice@us.ibm.com>" imported
gpg: key DF102246542FE08B: secret key imported
gpg: Total number processed: 1
gpg: imported: 1
gpg: secret keys read: 1
gpg: secret keys imported: 1
skopeo --insecure-policy --sign-by 134E58C247B8F0CB437009450F102246542FE08B copy --src-tls-verify=false --dest-tls-verify=true do
Getting image source signatures
Copying blob sha256:43dc9ef977174119841185896fc021aa3af745ecd5c892fe14668f21d4b069
Copying blob sha256:4632d477598a4176b858a910f5102c05db036d2a4d206a0aad1f62952f1508d6
Copying blob sha256:7a6132c48dd4568bdc334179b3a74dc9145d9425a1075a21bcb08fa79dacc6a
Copying blob sha256:6eeb9b4a640ff7b7b8bbac12b72740c753337a5d820fd5ebc1d9244a787ca7db
Copying blob sha256:741a0a0f93f892b09099937b42242ae98b199278798bb5277ce7906af126957
Copying blob sha256:e3179acfe7d83969c1e385916f53ae18b0dadd6877d3bd7631d59521817c595f
Copying blob sha256:ed2fddbb30c3ee1ac1429c41508f2a2e2354f1b3739a84387929ef034c0f61
Copying blob sha256:b80ee16c866200b7aca5ae763b95a878e756c7bbbd7cc3b19a033bf1372efc61
Copying config sha256:7595a3327d0516e9783642c08cb3f2dda2432888cf08cb9c5e10aa8ef846d65
Writing manifest to image destination
Signing manifest
Storing signatures
```

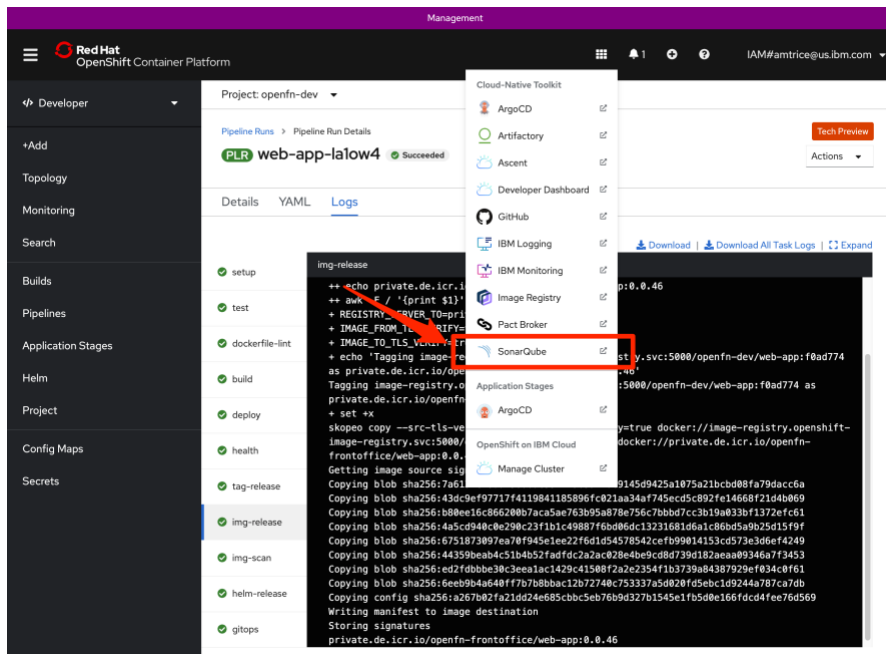
The reason that we sign container images is so that we can enforce trusted workloads. We can ensure that our applications have not been modified or (more importantly) compromised after they have left our build process. This is done by signing an image at build time, and then verifying the image at the time of deployment into the cluster. By using signed images with IBM Container Security Enforcement in the OpenShift cluster we can block container images that do not match the keys that were used to sign them. This means that any container image that was modified outside of our workflow will be



IBM Cloud for Financial Services – Tech Zone Demo Environment Hybrid Cloud Ecosystem – Ecosystem Labs

rejected from the cluster.

- Next, click on the application launcher menu, and select SonarQube. This will launch the SonarQube interface in a new tab.



SonarQube performs static code analysis to evaluate code quality, using analysis rules that focus on three areas:

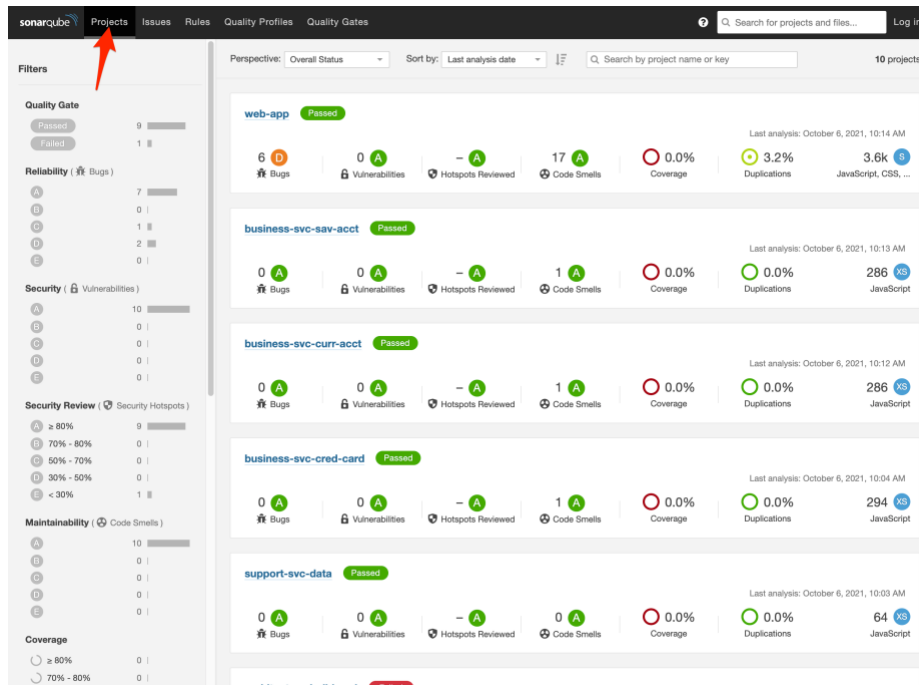
- Code Reliability:** Detect bugs that will impact end-user functionality
- Application Security:** Detect vulnerabilities and hot spots that can be exploited to compromise the program
- Technical Debt:** Keep your codebase maintainable to increase developer velocity

This type of analysis is important in an enterprise software development lifecycle because over time it helps improve overall code quality and reduce vulnerabilities and risk associated with the codebase.

- Click on “Projects” in the header to view the results for SonarQube code analysis scans.



IBM Cloud for Financial Services – Tech Zone Demo Environment Hybrid Cloud Ecosystem – Ecosystem Labs



This provides you with an at-a-glance view of all of your projects that are being scanned with SonarQube. This can allow you to quickly see if your codebase has vulnerabilities or poor coding practices.

17. Now let's dig in deeper to look at the web-app project's scan results.

- Click on the "web-app" project to view the details from the OpenFN retail banking web app's SonarQube scan.
- Click the "Issues" tab to view code issues that are detected from the SonarQube analysis.



IBM Cloud for Financial Services – Tech Zone Demo Environment Hybrid Cloud Ecosystem – Ecosystem Labs

The screenshot displays the SonarQube web interface for a project named 'web-app' on the 'master' branch. The top navigation bar includes links for Projects, Issues, Rules, Quality Profiles, and Quality Gates. A search bar and a 'Log in' button are also present. The main header shows the date 'October 6, 2021, 10:14 AM' and 'Version not provided'. Below the header, there are tabs for Overview, Issues (selected), Security Hotspots, Measures, Code, and Activity. A 'Project information' icon is visible on the right. The left sidebar contains a 'Filters' section with expandable categories: Type (Bug, Vulnerability, Code Smell), Severity (Blocker, Critical, Major, Minor, Info), Resolution, Status, Security Category, Creation Date, Language, Rule, Tag, Directory, File, Assignee, and Author. The main content area shows a list of 23 issues. The first issue is 'Remove this useless assignment to variable "languageQuery"', categorized as a Code Smell with Major severity, Open status, and 15min effort. Other issues include 'Remove the declaration of the unused "languageQuery" variable', 'Unnecessary semicolon', 'Unexpected missing generic font family', 'Remove this unused import of "jsx"', and 'Remove the declaration of the unused "Heading" variable'. Each issue entry includes a brief description, a reason for the issue, a timestamp, a complexity score (e.g., L17, L23), and a status (e.g., unused, pitfall, suspicious).

Here you can see issues or risky code patterns that are identified within the application codebase.

At this point you should emphasize that the continuous integration automation is improving the overall quality of the application. It covers an automated build process, automated publishing and versioning of images and assets, and automated deployment using GitOps, which we will cover in the next demo steps.

THIS CONCLUDES THE DEMO STEPS